Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High-Speed Networks

Ion Stoica, Scott Shenker, *Fellow, IEEE*, and Hui Zhang, *Member, IEEE*

***Abstract*—Router mechanisms designed to achieve fair bandwidth allocations, such as Fair Queueing, have many desirable properties for congestion control in the Internet. However, such mechanisms usually need to maintain state, manage buffers, and/or perform packet scheduling on a per-flow basis, and this complexity may prevent them from being cost-effectively implemented and widely deployed. In this paper, we propose an architecture that significantly reduces this implementation complexity yet still achieves approximately fair bandwidth allocations. We apply this approach to an island of routers–that is, a contiguous region of the network–and we distinguish between edge routers and core routers. Edge routers maintain per-flow state; they estimate the incoming rate of each flow and insert a label into each packet based on this estimate. Core routers maintain *no* per-flow state; they use first-in-first-out packet scheduling augmented by a probabilistic dropping algorithm that uses the packet labels and an estimate of the aggregate traffic at the router. We call the scheme *Core*-Stateless Fair Queueing. We present simulations and analysis on the performance of this approach.**

***Index Terms*—Binary linear codes, covering radius, least covering radius.**

I. Introduction

A CENTRAL tenet of the Internet architecture is that congestion control is achieved mainly through end-host algorithms. However, starting with Nagle [23], many researchers observed that such end-to-end congestion-control solutions are greatly improved when routers have mechanisms that allocate bandwidth in a fair manner. Fair bandwidth allocation protects well-behaved flows from ill-behaved ones, and allows a diverse set of end-to-end congestion-control policies to co-exist in the network [9]. As we discuss in Section IV, some maintain that fair bandwidth allocation[1] plays a necessary, not just beneficial, role in congestion control [9], [28].

Until now, fair allocations were typically achieved by using per-flow queueing mechanisms—such as Fair Queueing [9], [26] and its many variants [2], [14], [29]—or per-flow dropping mechanisms such as Flow Random Early Drop (FRED) [20]. These mechanisms are more complex to implement than traditional first-in-first-out (FIFO) queueing with drop-tail, which is the most widely implemented and deployed mechanism in routers today. In particular, fair allocation mechanisms inherently require the routers to maintain state and perform operations on a per-flow basis. For each packet that arrives at the router, the routers needs to *classify* the packet into a flow, update per-flow state variables, and perform certain operations based on the per-flow state. The operations can be as simple as deciding whether to drop or queue the packet (e.g., FRED), or as complex as manipulation of priority queues (e.g., Fair Queueing). While a number of techniques have been proposed to reduce the complexity of the per packet operations [1], [29], [33], and commercial implementations are available in some intermediate class routers, it is still unclear whether these algorithms can be cost-effectively implemented in high-speed backbone routers because all these algorithms still require packet classification and per-flow state management.

In this paper, we start with the assumptions that: 1) fair allocation mechanisms play an important, perhaps even necessary, role in congestion control; and 2) the complexity of existing fair allocation mechanisms is a substantial hindrance to their adoption. Both of these points are debatable; developments in router technology may make such algorithms rather inexpensive to implement, and there may be solutions to congestion control that do not require fair allocation (see Section V for a discussion on the related approaches). By using these two assumptions as our starting points, we are not claiming that they *are* true, but rather are only looking at the implications if indeed they *were* true. If one starts with these assumptions, then overcoming the complexity problem in achieving fair allocation becomes a vitally important problem.

To this end, we propose and examine an architecture and a set of algorithms that allocate bandwidth in an approximately fair manner while allowing the routers on high-speed links to use FIFO queueing and maintain no per-flow state. In this approach, we identify a contiguous region of network, called an *island*, and distinguish between the edge and the core of the island. Edge

Manuscript received July 27, 1999; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor E. Zegura. This work was supported by the Defense Advanced Research Projects Agency (DARPA) under Contract F30602-99-1-0518, and by the National Science Foundation (NSF) under Grants NCR-9624979, ANI-9730105, ITR Award ANI-0085920, and ANI-9814929, and in part by Intel. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, NSF, Intel, or the U.S. government.

I. Stoica is with the Computer Science Division, University of California, Berkeley, CA 94720-1776 USA (e-mail: istoica@cs.berkeley.edu).

S. Shenker is with the ICSI Center for Internet Research, International Computer Science Institute (ICSI), Berkeley, CA 94704-1198 USA (e-mail: shenker@icsi.berkeley.edu).

H. Zhang is with Carnegie Mellon University, Pittsburgh, PA 15213 USA, and also with Turin Networks, Petaluma, CA 94954 USA (e-mail: hzhang@cs.cmu.edu).

Digital Object Identifier 10.1109/TNET.2002.808414

[1] In this paper, we use the max–min definition of fairness [17].

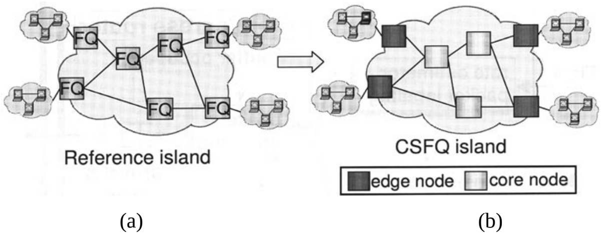


Fig. 1. (a) Reference network (island) in which all nodes implement Fair Queueing, whose functionality is approximated by (b) a CSFQ island. In CSFQ, only edge routers maintain per-flow state; core nodes do not maintain per-flow state.

routers compute per-flow rate estimates and *label* the packets passing through them by inserting these estimates into each packet header. Core routers use FIFO queueing and keep no per-flow state. They employ a probabilistic dropping algorithm that uses the information in the packet labels along with the router's own measurement of the aggregate traffic. The bandwidth allocations within this island of routers are approximately fair. Thus, if this approach were adopted within the high-speed interiors of ISPs, and fair allocation mechanisms were adopted for the slower links outside of these high-speed interiors, then approximately fair allocations could be achieved everywhere. However, this approach, like Fair Queueing [9] or Random Early Detection (RED) [12], still provides benefit if adopted in an incremental fashion, although the incremental adoption must be done on an island-by-island basis, not on a router-by-router basis.

We call this approach *Core*-Stateless Fair Queueing (CSFQ) since the core routers keep no per-flow state but instead use the state that is carried in the packet labels.[2] In this paper, we show that this approach can *approximate* the functionality of a reference network (island) in which *all* nodes implement Fair Queueing, i.e., the bandwidth allocations in both the reference and CSFQ islands are approximately the same (see Fig. 1). We describe the details of this approach—such as the rate estimation algorithm and the packet dropping algorithm—in Section II.

While our scheme does not achieve the near-perfect levels of fairness obtained by Fair Queueing and other sophisticated and stateful queueing algorithms, we derive a worst-case bound for the performances of CSFQ in an idealized setting. This bound is presented in Section II.

This worst-case analysis does not give an adequate guide to the typical functioning of CSFQ. In Section III, we present results from simulation experiments to illustrate the performance of our approach and to compare it to several other schemes, namely, Deficit Round Robin (DRR) (a variant of Fair Queueing), FRED, RED, and FIFO. We also discuss therein the relative mechanistic complexities of these approaches.

Sections II and III of the paper are narrowly focused on the details of the mechanism and its performance (both absolute and relative), with the need for such a mechanism taken for granted. In Section IV, we return to the basic question of why fair allocations are relevant to congestion control. We present the related work in Section V and conclude with a summary in Section VI. The CSFQ implementation for ns-2 [24] is available at http://www.cs.berkeley.edu/istoica/csfq.

II. CSFQ

In this section, we propose an architecture that approximates the service provided by an island of Fair Queueing routers, but has a much lower complexity in the core routers. The architecture has two key aspects. First, to avoid maintaining per-flow state at each router, we use a distributed algorithm in which only edge routers maintain per-flow state, while core (nonedge) routers do not maintain per-flow state but instead utilize the per-flow information carried via a label in each packet's header. This label contains an estimate of the flow's rate; it is initialized by the edge router based on per-flow information, and then updated at each router along the path based only on aggregate information at that router.

Second, to avoid per-flow buffering and scheduling, as required by Fair Queueing, we use FIFO queueing with probabilistic dropping on input. The probability of dropping a packet as it arrives to the queue is a function of the rate estimate carried in the label and of the fair share rate at that router, which is estimated based on measurements of the aggregate traffic.

Thus, our approach avoids both the need to maintain per-flow state and the need to use complicated packet scheduling and buffering algorithms at core routers. To give a better intuition about how this works, we first present the idealized bit-by-bit or *fluid* version of the probabilistic dropping algorithm, and then extend the algorithm to a practical packet-by-packet version.

A. Fluid Model Algorithm

We first consider a bufferless fluid model of a router with output link speed C, where the flows are modeled as a continuous stream of bits. We assume each flow's arrival rate $r_i(t)$ is known precisely. Max–min fair bandwidth allocations are characterized by the fact that all flows that are bottlenecked by this router have the same output rate. We call this rate the *fair share rate* of the link; let $\alpha(t)$ be the fair share rate at time t. In general, if max–min bandwidth allocations are achieved, each flow i receives service at a rate given by $\min(r_i(t), \alpha(t))$. Let $A(t)$ denote the total arrival rate: $A(t) = \sum_{i=1}^{n} r_i(t)$. If $A(t) > C$, then the fair share $\alpha(t)$ is the unique solution to

$$C = \sum_{i=1}^{n} \min(r_i(t), \alpha(t)). \tag{1}$$

If $A(t) \leq C$, then no bits are dropped and we will, by convention, set $\alpha(t) = \max_i r_i(t)$.

If $r_i(t) \leq \alpha(t)$, i.e., flow i sends no more than the link's fair share rate, all of its traffic will be forwarded. If $r_i(t) > \alpha(t)$, then a fraction $(r_i(t) - \alpha(t))/r_i(t)$ of its bits will be dropped, so it will have an output rate of exactly $\alpha(t)$. This suggests a very simple probabilistic forwarding algorithm that achieves fair allocation of bandwidth: each incoming bit of flow i is dropped with the probability

$$\max\left(0, 1 - \frac{\alpha(t)}{r_i(t)}\right). \tag{2}$$

[2] Obviously, these core routers keep some state, but none of it is per-flow state, so when we say "stateless" we are referring to the absence of per-flow state.

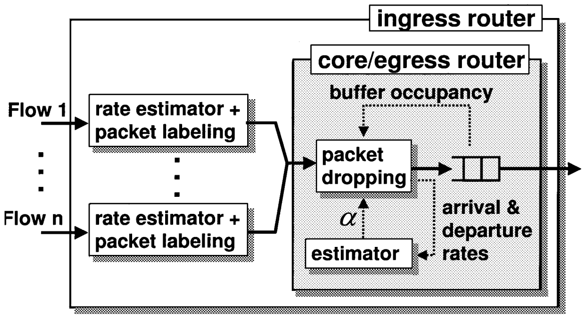


Fig. 2. Architecture of the output port of an edge router and a core router, respectively.

When these dropping probabilities are used, the arrival rate of flow i at the next hop is given by $\min[r_i(t), \alpha(t)]$.

B. Packet Algorithm

The above algorithm is defined for a bufferless fluid system in which the arrival rates are known exactly. Our task now is to extend this approach to the situation in real routers where transmission is packetized, there is substantial buffering, and the arrival rates are not known. Fig. 2 presents the architecture of a CSFQ node.

We still employ a drop-on-input scheme, except that now we drop packets rather than bits. Because the rate estimation (described below) incorporates the packet size, the dropping probability is independent of the packet size and depends only, as above, on the rate $r_i(t)$ and fair share rate $\alpha(t)$.

We are left with two remaining challenges: estimating the rates $r_i(t)$ and the fair share $\alpha(t)$. We address these two issues in turn in Sections II-B1 and B2, and then discuss the rewriting of the labels. Pseudocode reflecting this algorithm is described in Figs. 3 and 4. We should note, however, that the main point of our paper is the overall architecture and that the detailed algorithm presented below represents only an initial prototype. While it serves adequately as a proof-of-concept of our architecture, we fully expect that the details of this design will continue to evolve.

1) Computation of Flow Arrival Rate: Recall that in our architecture, the rates $r_i(t)$ are estimated at the edge routers and then these rates are inserted into the packet labels. At each edge router, we use exponential averaging to estimate the rate of a flow. Let t_i^k and l_i^k be the arrival time and length of the kth packet of flow i. The estimated rate of flow i, r_i, is updated every time a new packet is received

$$r_i^{\text{new}} = \left(1 - e^{-T_i^k/K}\right) \frac{l_i^k}{T_i^k} + e^{-T_i^k/K} r_{i,\text{old}} \tag{3}$$

where $T_i^k = t_i^k - t_i^{k-1}$ and K is a constant. We discuss the rationale for using the form $e^{-T_i^k/K}$ for the exponential weight in Section II-G.

```
on receiving packet $p$
  if (edge router)
    $i$ =classify($p$);
    $p.label$ = estimate_rate($r_i, p$); // use Eq. (??)
  $prob$ =max($0, 1 - \alpha/p.label$);
  if ($prob$ >unif_rand($0, 1$))
    $\alpha$ =estimate_$\alpha$ ($p, 1$);
    drop(p);
  else
    $\alpha$ =estimate_$\alpha$ ($p, 0$);
    enqueue(p);
    if ($prob > 0$)
      $p.label = \alpha$; // relabel p
```

Fig. 3. Pseudocode of CSFQ.

2) Link Fair Rate Estimation: Next, we present an estimation algorithm for $\alpha(t)$. To give intuition, consider again the fluid model in Section II-A where the arrival rates are known exactly, and assume the system performs the probabilistic dropping algorithm according to (2). Then, the rate with which the algorithm accepts packets is a function of the current estimate of the fair share rate, which we denote by $\widehat{\alpha}(t)$. Letting $F(\widehat{\alpha}(t))$ denote this acceptance rate, we have

$$F(\widehat{\alpha}(t)) = \sum_{i=1}^{n} \min\left(r_i(t), \widehat{\alpha}(t)\right). \tag{4}$$

Note that $F(\cdot)$ is a continuous nondecreasing concave and piecewise-linear function of $\widehat{\alpha}$. If the link is congested $(A(t) > C)$, we choose $\widehat{\alpha}(t)$ to be the unique solution to $F(x) = C$. If the link is not congested $(A(t) < C)$, we take $\widehat{\alpha}(t)$ to be the largest rate among the flows that traverse the link, i.e., $\widehat{\alpha}(t) = \max_{1 \le i \le n}(r_i(t))$. From (4), note that if we knew the arrival rates $r_i(t)$, we could then compute $\alpha(t)$ directly. To avoid having to keep such per-flow state, we seek instead to implicitly compute $\widehat{\alpha}(t)$ by using only aggregate measurements of F and A.

We use the following heuristic algorithm with three aggregate state variables: $\widehat{\alpha}$, the estimate for the fair share rate; $\widehat{A}$, the estimated aggregate arrival rate; $\widehat{F}$, the estimated rate of the accepted traffic. The last two variables are updated upon the arrival of each packet. For $\widehat{A}$, we use exponential averaging with a parameter e^{-T/K_α} where T is the interarrival time between the current and the previous packet

$$\widehat{A}_{\text{new}} = \left(1 - e^{-T/K_\alpha}\right) \frac{l}{T} + e^{-T/K_\alpha} \widehat{A}_{\text{old}} \tag{5}$$

where $\widehat{A}_{\text{old}}$ is the value of $\widehat{A}$ before the updating. We use an analogous formula to update $\widehat{F}$.

The updating rule for $\widehat{\alpha}$ depends on whether the link is congested or not. To filter out the estimation inaccuracies due to exponential smoothing, we use a window of size K_c. A link is assumed to be *congested* if $\widehat{A} \ge C$ at all times during an interval of length K_c. Conversely, a link is assumed to be *uncongested* if $\widehat{A} \le C$ at all times during an interval of length K_c. The value $\widehat{\alpha}$ is updated only at the end of an interval in which the link is either congested or uncongested according to these definitions. If the link is congested, then $\widehat{\alpha}$ is updated based on $F(\widehat{\alpha}) = C$. We approximate $F(\cdot)$ by a linear function that intersects the origin and has slope $\widehat{F}/\widehat{\alpha}_{\text{old}}$. This yields

$$\widehat{\alpha}_{\text{new}} = \widehat{\alpha}_{\text{old}} \frac{C}{\widehat{F}}. \tag{6}$$

```
estimate_$\alpha$ ($p$, dropped)
  // $\widehat{\alpha}$ is initialized to $C$, and $K_c\_\alpha$ to 0;
  // $K_c\_\alpha$ is used to compute the largest label seen
  // during a window of size $K_c$
  $\widehat{A}$ = estimate_rate($\widehat{A}, p$); // est. arrival rate (use Eq. (??))
  if (dropped == $FALSE$)
    $\widehat{F}$ = estimate_rate($\widehat{F}, p$); // est. accepted traffic rate
  if ($\widehat{A} \geq C$)
    if (congested == $FALSE$)
      congested = $TRUE$;
      start_time = crt_time;
      if ($\widehat{\alpha} == 0$)
        // $\widehat{\alpha}$ is 0 if no packet is received during a widow $K_c$
        $\widehat{\alpha} = \min(p.label, C)$; //initialize $\widehat{\alpha}$
    else
      if (crt_time > start_time + $K_c$)
        $\widehat{\alpha} = \widehat{\alpha} \times C/\widehat{F}$;
        start_time = crt_time;
  else // $\widehat{A} < C$
    if (congested == $TRUE$)
      congested = $FALSE$;
      start_time = crt_time;
      $K_c\_\alpha = 0$;
    else
      if (crt_time < start_time + $K_c$)
        $K_c\_\alpha = \max(K_c\_\alpha, p.label)$;
      else
        $\widehat{\alpha} = K_c\_\alpha$;
        start_time = crt_time;
        $K_c\_\alpha = 0$;
  return $\widehat{\alpha}$;
```

Fig. 4. Pseudocode of CSFQ (fair rate estimation).

If the link is not congested, $\widehat{\alpha}_{\text{new}}$ is set to the largest rate of any active flow (i.e., the largest label seen) during the last K_c time units. The value of $\widehat{\alpha}_{\text{new}}$ is then used to compute dropping probabilities, according to (2). For completeness, we give the pseudocode of the CSFQ algorithm in Fig. 4.

We now describe two minor amendments to this algorithm related to how the buffers are managed. The goal of estimating the fair share $\widehat{\alpha}$ is to match the accepted rate to the link bandwidth. Due to estimation inaccuracies, load fluctuations between $\widehat{\alpha}$'s updates, and the probabilistic nature of our algorithm, the accepted rate may occasionally exceed the link capacity. While ideally the router's buffers can accommodate the extra packets, occasionally the router may be forced to drop the incoming packet due to lack of buffer space. Since drop-tail behavior will defeat the purpose of our algorithm, and may exhibit undesirable properties in the case of adaptive flows such as TCP [12], it is important to limit its effect. To do so, we use a simple heuristic: every time the buffer overflows, $\widehat{\alpha}$ is decreased by a small fixed percentage (taken to be 1% in our simulations). Moreover, to avoid overcorrection, we make sure that during consecutive updates $\widehat{\alpha}$ does not decrease by more than 25%. While the choice of these percentage values is somewhat arbitrary, as shown in Section III, they offer consistent performance over a wide range of simulation scenarios.

In addition, since there is little reason to consider a link congested if the buffer is almost empty, we apply the following rule. If the link becomes uncongested by the test in Fig. 4, then we assume that it remains uncongested as long as the buffer occupancy is less than some predefined threshold. In this paper, we use a threshold that is half of the total buffer capacity.

3) Label Rewriting: Our rate estimation algorithm in Section II-B1 allows us to label packets with their flow's rate as they enter the island. Our packet dropping algorithm described in Section II-B2 allows us to limit flows to their fair share of the bandwidth. After a flow experiences significant losses at a congested link inside the island, however, the packet labels are no longer an accurate estimate of its rate. We cannot rerun our estimation algorithm, because it involves per-flow state. Fortunately, as noted in Section II-A, the outgoing rate is merely the *minimum* between the incoming rate and the fair rate α. Therefore, we rewrite the the packet label L as

$$L_{\text{new}} = \min(L_{\text{old}}, \alpha). \tag{7}$$

By doing so, the outgoing flow rates will be properly represented by the packet labels.

C. Weighted CSFQ

The CSFQ algorithm can be extended to support flows with different weights. Let w_i denote the weight of flow i. Returning to our fluid model, the meaning of these weights is that we say a *fair* allocation is one in which all bottlenecked flows have the same value for r_i/w_i. Then, if $A(t) > C$, the *normalized* fair rate $\alpha(t)$ is the unique value such that $\sum_{i=1}^{n} w_i \min(\alpha, r_i/w_i) = C$. The expression for the dropping probabilities in the weighted case is $\max(0, 1 - \alpha(w_i/r_i))$. The only other major change is that the label is now r_i/w_i, instead simply r_i. Finally, without going into detail, we note that the weighted packet-by-packet version is virtually identical to the corresponding version of the plain CSFQ algorithm.

It is important to note that with weighted CSFQ, we can only approximate islands in which each flow has the same weight at all routers in an island. That is, our algorithm cannot accommodate situations where the relative weights of flows differ from router to router within an island. However, even with this limitation, weighted CSFQ may prove a valuable mechanism in implementing differential services, such as the one proposed in [36].

D. Performance Bounds

We now present the main theoretical result of the paper. For generality, this result is given for weighted CSFQ. The proof is given in the technical report [34].

Our algorithm is built around several estimation procedures and, thus, is inherently inexact. One natural concern is whether a flow can purposely "exploit" these inaccuracies to get more than its fair share of bandwidth. We cannot answer this question in full generality, but we can analyze a simplified situation where the normalized fair share rate α is held fixed and there is no buffering, so the drop probabilities are precisely given by (2). In addition, we assume that when a packet arrives, a fraction of that packet equal to the flow's forwarding probability is transmitted. Note that during any time interval $[t_1, t_2)$ a flow with weight w is entitled to receive at most $w\alpha(t_2 - t_1)$ service time; we call any amount above this the *excess service*. We can bound this excess service, and the bounds are independent of both the arrival process and the length of the time interval during which the flow is active. The bound does depend crucially on the maximal rate R at which a flows packets can arrive at a router (limited, for example, by the speed of the flow's access link); the smaller this rate R, the tighter the bound.

Theorem 1: Consider a link with a constant normalized fair rate α and a flow with weight w. Then, the excess service received by a flow with weight w that sends at a rate no larger than R is bounded above by

$$r_\alpha K \left(1 + \ln \frac{R}{r_\alpha}\right) + l_{\max} \tag{8}$$

where $r_\alpha = \alpha w$ and $l_{\max}$ represents the maximum length of a packet.

By bounding the excess service, we have shown that in this idealized setting the asymptotic throughput cannot exceed the fair share rate. Thus, flows can only exploit the system over short time scales; they are limited to their fair share over long time scales.

E. Implementation Complexity

At core routers, both the time and space complexity of our algorithm are constant with respect to the number of competing flows and, thus, we think CSFQ could be implemented in very high-speed core routers. At each edge router, CSFQ needs to maintain per-flow state. Upon each arrival of each packet, the edge router needs to: 1) classify the packet to a flow; 2) update the fair share rate estimation for the corresponding outgoing link; 3) update the flow rate estimation; and 4) label the packet. All these operations with the exception of packet classification can be efficiently implemented today.

Efficient and general-purpose packet classification algorithms are still under active research [15], [19], [31], [32]. We expect to leverage these results. We also note that packet classification at ingress nodes is needed for a number of other purposes, such as in the context of Multiprotocol Label Switching (MPLS) [4] or for accounting purposes; therefore, the classification required for CSFQ may not be an extra cost. In addition, if the edge routers are typically not on the high-speed backbone links, then there is no problem as classification at moderate speeds is quite practical.

F. Architectural Considerations

We have used the term "flow" without defining what we mean. This was intentional, as the CSFQ approach can be applied to varying degrees of flow granularity; that is, what constitutes a flow is arbitrary as long as all packets in the flow follow the same path within the core. In this paper, for convenience, a flow is implicitly defined as an IP source–destination address pair, but one could easily assign fair rates to many other granularities such as source–destination–ports. Moreover, the unit of "flow" can vary from island to island as long as the rates are re-estimated when entering a new island.

Similarly, we have not been precise about the size of these CSFQ islands. In one extreme, we could take each router as an island and estimate rates at every router; this would allow us to avoid the use of complicated per-flow scheduling and dropping algorithms, but would require per-flow classification. Another possibility is that ISPs could extend their island of CSFQ routers to the very edge of their network, having their edge routers at the points where customer's packets enter the ISP's network. Building on the previous scenario, multiple ISPs could combine their islands so that classification and estimation did not have to be performed at ISP–ISP boundaries. The key obstacle here is one of trust between ISPs.

G. Miscellaneous Details

Having presented the basic CSFQ algorithm, we now return to discuss a few aspects in more detail.

We have used exponential averaging to estimate the arrival rate in (3). However, instead of using a constant exponential weight, we used $e^{-T/K}$ where T is the interpacket arrival time and K is a constant. Our motivation was that $e^{-T/K}$ more closely reflects a fluid averaging process which is independent of the packetizing structure. More specifically, it can be shown that if a constant weight is used, the estimated rate will be sensitive to the packet length distribution and there are pathological cases where the estimated rate differs from the real arrival rate by a factor; this would allow flows to exploit the estimation process and obtain more than their fair share. In contrast, by using a parameter of $e^{-T/K}$, the estimated rate will asymptotically converge to the real rate, and this allows us to bound the excess service that can be achieved (as in Theorem 1). We used a similar averaging process in (5) to estimate the total arrival rate A.

The choice of K in the above expression $e^{-T/K}$ presents us with several tradeoffs. First, while a smaller K increases the system responsiveness to rapid rate fluctuations, a larger K better filters the noise and avoids potential system instability. Second, K should be large enough such that the estimated rate, calculated at the edge of the network, remains reasonably accurate after a packet traverses multiple links. This is because the delay jitter changes the packets' interarrival pattern, which may result in an increased discrepancy between the estimated rate (received in the packets' labels) and the real rate. To counteract this effect, as a rule of thumb, K should be one order of magnitude larger that the delay jitter experienced by a flow over a time interval of the same size K. Third, K should be no larger than the average duration of a flow. Based on these constraints, an appropriate value for K would be between 100 and 500 ms.

A second issue relates to the requirement of CSFQ for a label to be carried in each packet. One possibility is to use the Type Of Service byte in the IP header. For example, by using a floating-point representation with 4 bits for mantissa and 4 bits for exponent, we can represent any rate between 1 kb/s and 65 Mb/s with an accuracy of 6.25%. Another possibility is to define an IP option in the case of IPv4, or a hop-by-hop extension header in the case of IPv6.

III. Simulations

In this section, we evaluate our algorithm by simulation. To provide some context, we compare CSFQ's performance to three additional algorithms. Two of these, FIFO and RED, represent baseline cases where routers do not attempt to achieve fair bandwidth allocations. The other two algorithms, FRED and DRR, represent different approaches to achieving fairness.

- FIFO: Packets are served in a first-in-first-out order, and the buffers are managed using a simple drop-tail strategy; i.e., incoming packets are dropped when the buffer is full.
- RED: Packets are served in a first-in-first-out order, but the buffer management is significantly more sophisticated than drop-tail. RED [12] starts to probabilistically drop packets long before the buffer is full, providing early congestion indication to flows which can then gracefully back off before the buffer overflows. RED maintains two buffer thresholds. When the exponentially averaged buffer occupancy is smaller than the first threshold, no packet is dropped, and when the exponentially averaged buffer occupancy is larger than the second threshold, all packets are dropped. When the exponentially averaged buffer occupancy is between the two thresholds, the packet dropping probability increases linearly with buffer occupancy.
- FRED: This algorithm extends RED to provide some degree of fair bandwidth allocation [20]. To achieve fairness, FRED maintains state for all flows that have at least one packet in the buffer. Unlike RED, where the dropping decision is based only on the buffer state, in FRED dropping decisions are based on this flow state. Specifically, FRED preferentially drops a packet of a flow that has either 1) had many packets dropped in the past, or 2) a queue larger than the average queue size. FRED has two variants (which we will call FRED-1 and FRED-2). The main difference between the two is that FRED-2 guarantees to each flow a minimum number of buffers. As a general rule, FRED-2 performs better than FRED-1 only when the number of flows is large. In the following data, when we do not distinguish between the two, we are quoting the results from the version of FRED which performed the best.
- DRR: This algorithm represents an efficient implementation of the well-known weighted fair queueing (WFQ) discipline. The buffer management scheme assumes that when the buffer is full, the packet from the longest queue is dropped. DRR is the only one of the four to use a sophisticated per-flow queueing algorithm and, thus, achieves the highest degree of fairness.

These four algorithms represent four different levels of complexity. DRR and FRED have to classify incoming flows, whereas FIFO and RED do not. DRR in addition has to implement its packet scheduling algorithm (whereas the rest all use first-in-first-out scheduling). CSFQ edge routers have complexity comparable to FRED, and CSFQ core routers have complexity comparable to RED.

We have examined the behavior of CSFQ under a variety of conditions. We use an assortment of traffic sources [mainly, constant bit rate (CBR) and TCP, but also some ON–OFF sources] and topologies. All simulations were performed in ns-2 [24], which provide accurate packet-level implementation for various network protocols, such as TCP and Receiver-driven Layered Multicast (RLM) [22], and various buffer management and scheduling algorithms, such as RED and DRR.

Unless otherwise specified, we use the following parameters for the simulations in this section. Each output link has a latency of 1 ms, a buffer of 64 kB, and a buffer threshold for CSFQ is 16 kB. In the RED and FRED cases, the first threshold is set to 16 kB, while the second one is set to 32 kB. The averaging constant used in estimating the flow rate is $K = 100$ ms, while the averaging constant used in estimation the fair rate α is $K_\alpha = 200$ ms. Finally, in all topologies we use, the first router (gateway) on the path of a flow is always assumed to be the edge router; all other routers are assumed without exception to be core routers.

We simulated the other four algorithms to give us benchmarks against which to assess these results. We use DRR as our model of fairness and use the baseline cases, FIFO and RED, as representing the (unfair) status quo. The goal of these experiments is determine where CSFQ sits between these two extremes. FRED is a more ambiguous benchmark, being somewhat more complex than CSFQ but not as complex as DRR.

In general, we find that CSFQ achieves a reasonable degree of fairness, significantly closer to DRR than to FIFO or RED. CSFQ's performance is typically comparable to FRED's, although there are a few situations where CSFQ significantly outperforms FRED. There are a large number of experiments and each experiment involves rather complex dynamics. Due to space limitations, in the following sections, we will merely highlight a few important points and omit detailed explanations of the dynamics.

A. Single Congested Link

We first consider a single congested link shared by N flows [see Fig. 5(a)]. We performed three related experiments.

In the first experiment, we have 32 CBR flows, indexed from 0, where flow i sends $i + 1$ times more than its fair share of 0.3125 Mb/s. Thus flow 0 sends 0.3125 Mb/s, flow 1 sends 0.625 Mb/s, and so on. Fig. 5(a) shows the average throughput of each flow over a 10-s interval; FIFO, RED, and FRED-1 fail to ensure fairness, with each flow getting a share proportional to its incoming rate, while DRR is extremely effective in achieving a fair bandwidth distribution. CSFQ and FRED-2 achieve a less precise degree of fairness; for CSFQ, the throughputs of all flows are between -11% and $+12\%$ of the ideal value.

In the second experiment, we consider the impact of an ill-behaved CBR flow on a set of TCP flows. More precisely, the traffic of flow 0 comes from a CBR source that sends at 10 Mb/s, while all the other flows (from 1 to 31) are TCP flows. Fig. 6(a) shows the throughput of each flow averaged over a 10-s interval. The only two algorithms that can most effectively contain the CBR flow are DRR and CSFQ. Under FRED, the CBR flow gets almost 1.8 Mb/s—close to six times more than its fair share—while the CBR only gets 0.396 and 0.355 Mb/s under DRR and CSFQ, respectively. As expected, FIFO and RED perform poorly, with the CBR flow getting over 8 Mb/s in both cases.

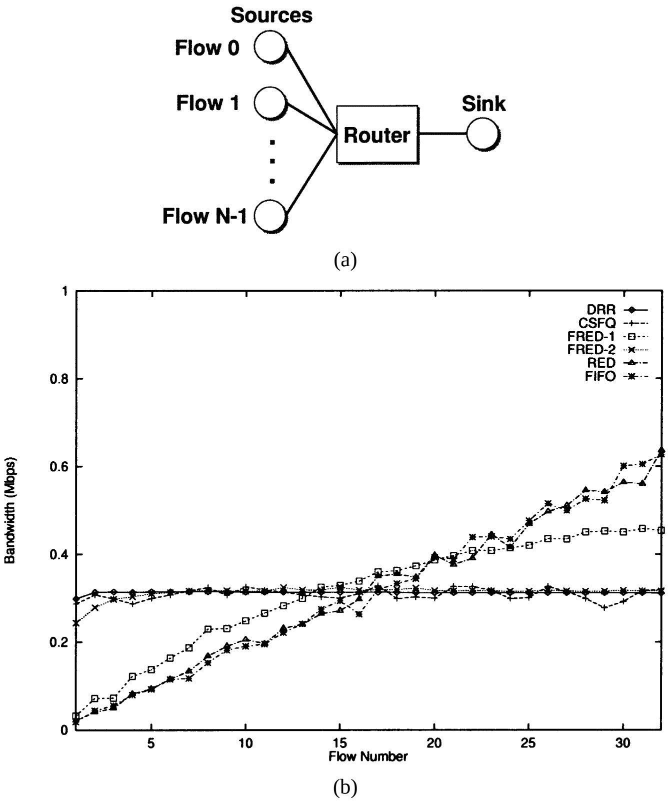


Fig. 5. (a) 10-Mb/s link shared by N flows. (b) Average throughput over 10 s when $N = 32$ and all flows are CBR flows. The arrival rate for flow i is $(i+1)$ times larger than its fair share. The flows are indexed from 0.

In the final experiment, we measure how well the algorithms can protect a single TCP flow against multiple ill-behaved flows. We perform 31 simulations, each for a different value of N, $N = 1, \ldots, 31$. In each simulation, we take one TCP flow and N CBR flows; each CBR sends at twice its fair share rate of $10/(N+1)$ Mb/s. Fig. 6(b) plots the ratio between the average throughput of the TCP flow over 10 s and the total bandwidth it should receive as a function of the total number of flows in the system $N + 1$. There are three points of interest. First, DRR performs very well when there are less than 22 flows, but its performance decreases afterwards, because then the TCP flow's buffer share is less than three packets, which is known to significantly affect its throughput. Second, CSFQ performs better than DRR when the number of flows is large. This is because CSFQ is able to cope better with the TCP burstiness by allowing the TCP flow to have more than two packets buffered for short time intervals. Upon receiving a burst of packets, it takes a while for the estimated rate of the flow to catch up with the actual rate. During this time, the packets of the flow might be admitted in the queue even if the actual rate of the flow is higher than the fair rate. Furthermore, note that unlike DRR, once a packet is enqueued, it cannot be dropped later. Finally, across the entire range, CSFQ provides similar or better performance as compared with FRED.

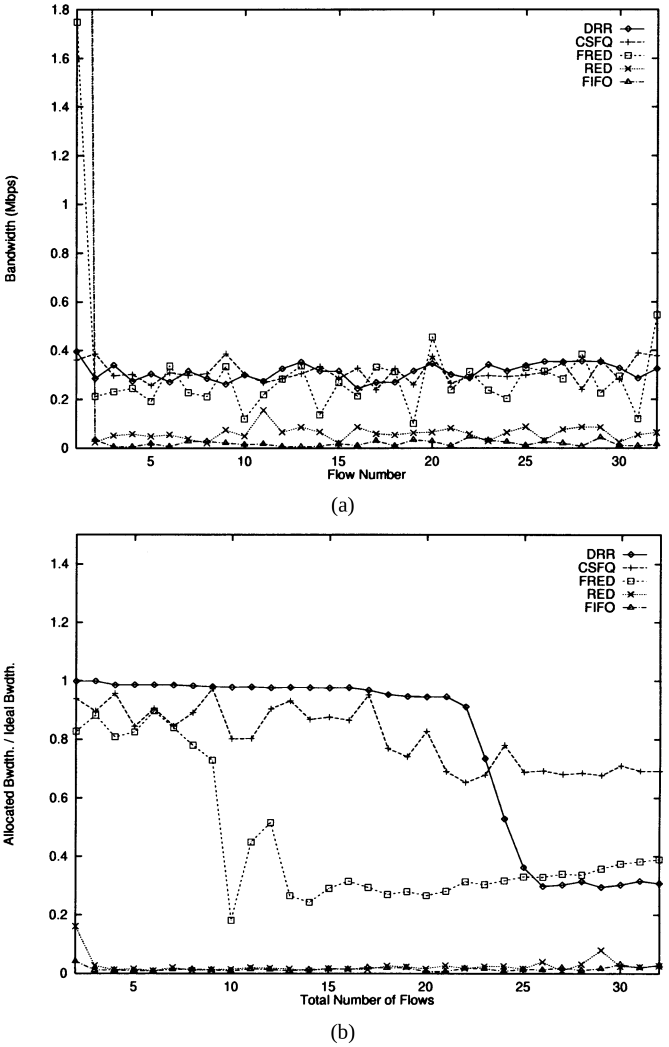


Fig. 6. (a) Throughputs of one CBR flow (0 indexed) sending at 10 Mb/s, and of 31 TCP flows sharing a 10-Mb/s link. (b) Normalized bandwidth of a TCP flow that competes with N CBR flows sending at twice their allocated rates, as a function of N.

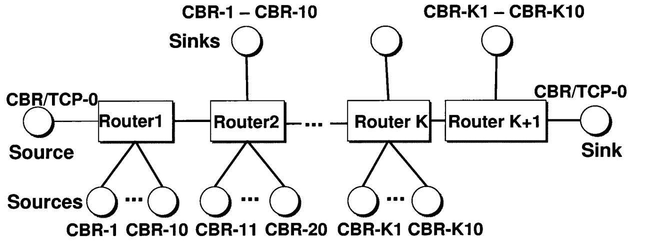
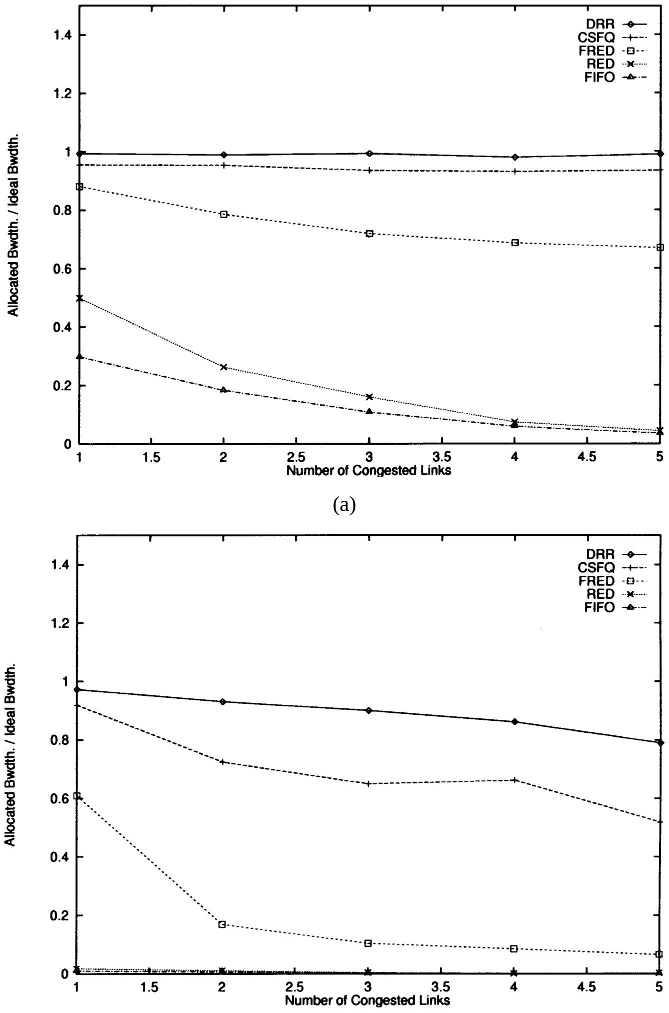


Fig. 7. Topology for analyzing the effects of multiple congested links on the throughput of a flow. Each link has ten cross flows (all CBR flows). All links have 10-Mb/s capacities. The sending rates of all CBR flows, excepting CBR-0, are 2 Mb/s, which leads to all links between routers being congested.

B. Multiple Congested Links

We now analyze how the throughput of a well-behaved flow is affected when the flow traverses more than one congested link. We performed two experiments based on the topology shown in Fig. 7. All CBR flows, except CBR-0, send at 2 Mb/s. Since each link in the system has 10-Mb/s capacity, this will result in all links between routers being congested.

In the first experiment, we have a CBR flow (denoted CBR-0) sending at its fair share rate of 0.909 Mb/s. Fig. 8(a) shows the fraction of CBR-0's traffic that is forwarded, versus the number of congested links. CSFQ and FRED perform reasonably well, although not quite as well as DRR.



(a)

(b)

Fig. 8. (a) Normalized throughput of CBR-0 as a function of the number of congested links. (b) Same plot when CBR-0 is replaced by a TCP flow.

In the second experiment, we replace CBR-0 with a TCP flow. Similarly, Fig. 8(b) plots the normalized TCP throughput against the number of congested links. Again, DRR and CSFQ prove to be effective. In comparison, FRED performs significantly worse though still much better than RED and FIFO. The reason is that while DRR and CSFQ try to allocate bandwidth fairly among competing flows during congestion, FRED tries to allocate the buffer fairly. Flows with different end-to-end congestion-control algorithms will achieve different throughputs even if routers allocate the buffer fairly. In the case of Fig. 8(a), all sources are CBR, i.e., none are adopting any end-to-end congestion-control algorithms, and FRED provides performance similar to CSFQ and DRR. In the case of Fig. 8(b), a TCP flow is competing with multiple CBR flows. Since the TCP flow slows down during congestion while CBQ does not, it achieves significantly less throughput than a competing CBR flow.

C. Coexistence of Different Adaptation Schemes

In this experiment, we investigate the extent to which CSFQ can deal with flows that employ different adaptation schemes. RLM [22] is an adaptive scheme in which the source sends the information encoded into a number of layers (each to its own multicast group) and the receiver joins or leaves the groups associated with the layers based on how many packet drops it is experiencing. We consider a 4-Mb/s link traversed by one TCP and three RLM flows. Each source uses a seven-layer encoding, where layer i sends 2^{i+4} kb/s; each layer is modeled by a CBR traffic source. The fair share of each flow is 1 Mb/s. In the RLM case, this will correspond to each receiver subscribing to the first five layers.[3]

The average receiving rates averaged over a 1-s interval for each algorithm are plotted in Fig. 9. We have conducted two separate simulations of CSFQ.[4] In the first one, we have used the same averaging constant as in the rest of this paper: $K = 100$ ms and $K_\alpha = 200$ ms. Here, one RLM flow does not get its fair share (it is one layer below where it should be). We think this is due to the bursty behavior of the TCP that is not detected by CSFQ soon enough, allowing the TCP to opportunistically grab more bandwidth than its share at the expense of less aggressive RLM flows. To test this hypothesis, we have changed the averaging time intervals to $K = 20$ ms and $K_\alpha = 40$ ms, respectively, which result in the TCP flows bandwidth being restricted much earlier. As shown in Fig. 9(d), with these parameters all flows receive roughly 1 Mb/s.

An interesting point to notice is that FRED does not provide fair bandwidth allocation in this scenario. Again, as discussed in Section III-B, this is due to the fact that RLM and TCP use different end-to-end congestion-control algorithms.

Finally, we note that we have performed two other similar experiments (not included here due to space limitations): one in which the TCP flow is replaced by a CBR that sends at 4 Mb/s, and another in which we have both the TCP and the CBR flows together along with the three RLM flows. The overall results were similar, except that in both experiments *all* flows received their shares under CSFQ when using the original settings for the averaging intervals, i.e., $K = 100$ ms and $K_\alpha = 200$ ms. In addition, in some of these other experiments where the RLM flows are started before the TCP, the RLM flows get more than their share of bandwidth when RED and FIFO are used.

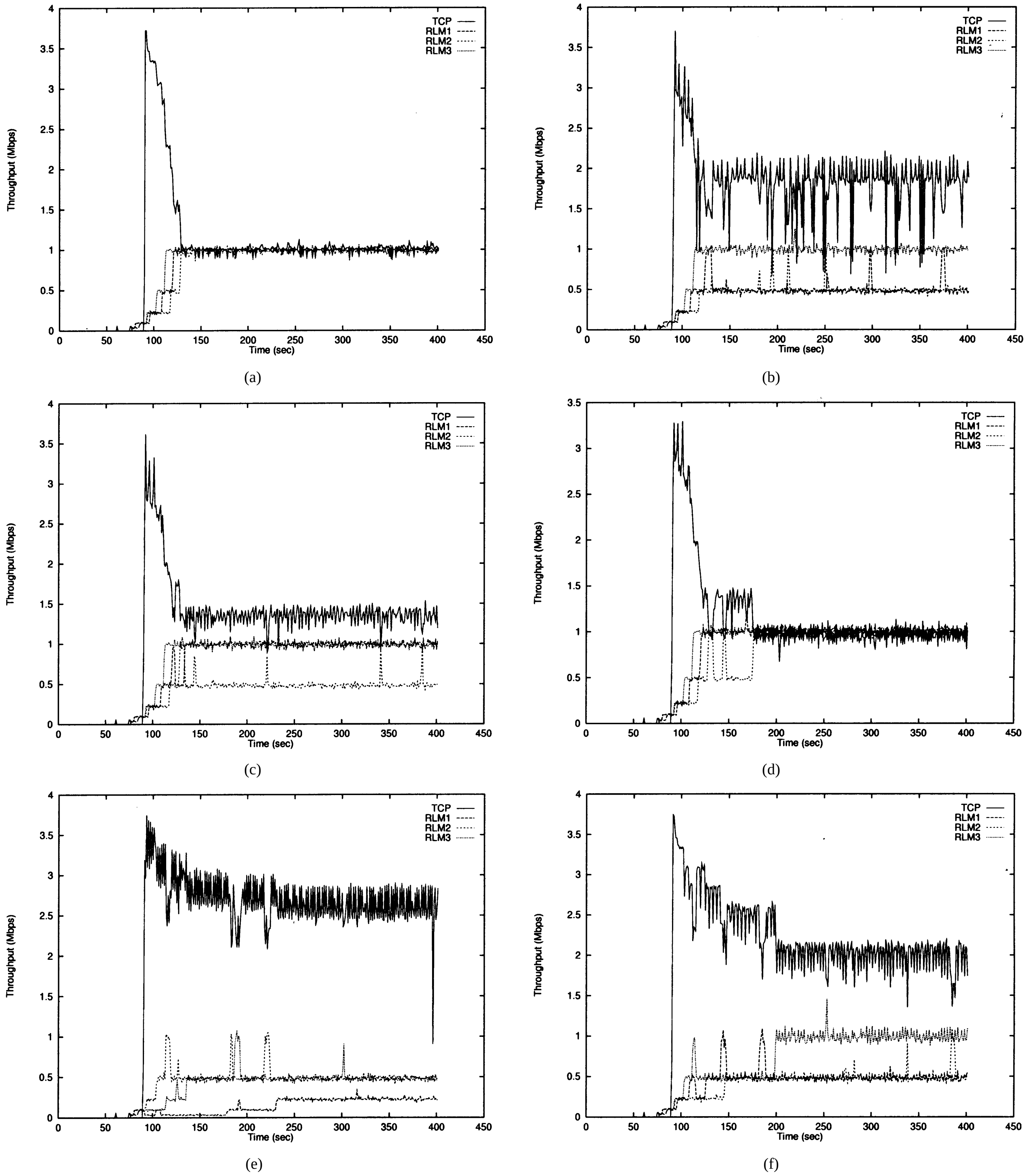
D. Different Traffic Models

So far, we have only considered CBR and TCP traffic sources. We now look at two additional source models with greater degrees of burstiness. We again consider a single 10-Mb/s congested link. In the first experiment, this link is shared by one ON–OFF source and 19 CBR flows that send at exactly their share, 0.5 Mb/s. The ON and OFF periods of the ON–OFF source are both drawn from exponential distributions with means of 200 ms and $19 * 200$ ms, respectively. During the ON period, the ON–OFF source sends at 10 Mb/s. Note that the ON-time is on the same order as the averaging interval $K = 200$ ms for CSFQ's rate estimation algorithm, so this experiment is designed to test to what extent CSFQ can react over short time scales.

The ON–OFF source sent 4899 packets over the course of the experiment. Table I shows the number of packets from the

[3] More precisely, we have $\sum_{i=1}^{5} 2^{i+4}$ kb/s $= 0.992$ Mb/s.

[4] See also [22] for additional simulations of RLM and TCP.



(a)

(b)

(c)

(d)

(e)

(f)

Fig. 9. Throughput of three RLM flows and one TCP flow along a 4-Mb/s link. (a) DRR. (b) FRED. (c) CSFQ ($K = 100$ ms, $K_\alpha = 200$ms). (d) CSFQ $K = 20$ ms, $K_\alpha = 40$ms. (e) RED. (f) FIFO.

ON–OFF source that were dropped at the congested link. The DRR results show what happens when the ON–OFF source is restricted to its fair share at all times. FRED and CSFQ also are able to achieve a high degree of fairness.

Our next experiment simulates Web traffic. There are 60 TCP transfers whose interarrival times are exponentially distributed with the mean of 0.1 ms, and the length of each transfer is drawn from a Pareto distribution with a mean of 40 packets (1 packet

TABLE I
Statistics for an On–Off Flow With 19 Competing CBR Flows (All Numbers are in Packets)

Algorithm	delivered	dropped
DRR	1080	3819
CSFQ	1000	3889
FRED	1064	3825
RED	2819	2080
FIFO	3771	1128

TABLE II
Mean Transfer Times (in Milliseconds) and the Corresponding Standard Deviations for 60 Short TCP Flows in the Presence of a CBR Flow That Sends at the Link Capacity, i.e., 10 Mb/s

Algorithm	mean time	std. dev
DRR	46.38	197.35
CSFQ	88.21	230.29
FRED	73.48	272.25
RED	790.28	1651.38
FIFO	1736.93	1826.74

TABLE III
Mean Throughputs (in Packets) and Standard Deviations for 19 TCP Flows in the Presence of a CBR Flow Along a Link With Propagation Delay of 100 ms. The CBR Sends at the Link Capacity of 10 Mb/s

Algorithm	mean	std. dev
DRR	5857.89	192.86
CSFQ	5135.05	175.76
FRED	4967.05	261.23
RED	628.10	80.46
FIFO	379.42	68.72

$= 1$ kB) and a shaping parameter of 1.06. These values are consistent with those presented in [8]. In addition, there is a single 10-Mb/s CBR flow.

Table II presents the mean transfer time and the corresponding standard deviations. Here, CSFQ and FRED do less well than DRR, but one order of magnitude better than FIFO and RED.

E. Large Latency

All of our experiments, so far, have had minimal latencies. In this experiment, we again consider a single 10-Mb/s congested link, but now the flows have propagation delays of 100 ms in getting to the congested link. The load is comprised of one CBR that sends at the link capacity and 19 TCP flows. Table III shows the mean number of packets forwarded for each TCP flow during a 100-s time interval. CSFQ and FRED both perform reasonably well.

F. Packet Relabeling

Recall that when the dropping probability P of a packet is nonzero, we relabel it with a new label where $L_{\text{new}} = (1–P)L_{\text{old}}$ so that the label of the packet will reflect the new rate of the flow. To test how well this works in practice, we consider the topology in Fig. 10, where each link is 10 Mb/s. Note that as long as all three flows attempt to use their full fair share, the fair shares of flows 1 and 2 are less on link 2 (3.33 Mb/s) than on link 1 (5 Mb/s), so there will be dropping on both links. This will test the relabeling function to make sure that the incoming rates are accurately reflected on the second link. We perform two experiments (only looking at CSFQ's performance). In the first, there are three CBR flows sending data at 10 Mb/s each. Table IV shows the average throughputs over 10 s of the three CBR flows. As expected, these rates are closed to 3.33 Mb/s. In the second experiment, we replace the three CBR flows by three TCP flows. Again, despite the TCP burstiness which may negatively affect the rate estimation and relabeling accuracy, each TCP gets its fair share.

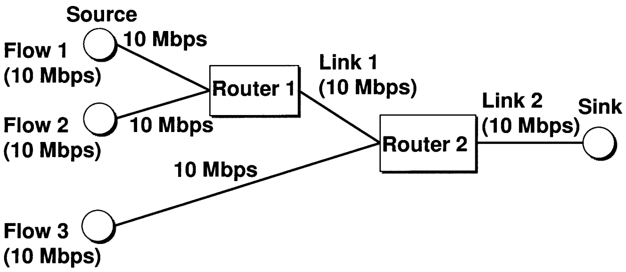


Fig. 10. Simulation scenario for the packet relabeling experiment. Each link has 10-Mb/s capacity and a propagation delay of 1 ms.

TABLE IV
Throughputs Resulting From CSFQ Averaged Over 10 s for the Three Flows in Fig. 10 Along Link 2

Traffic	Flow 1	Flow 2	Flow 3
CBR	3.267	3.262	3.458
TCP	3.232	3.336	3.358

G. Discussion of Simulation Results

We have tested CSFQ under a wide range of conditions, conditions purposely designed to stress its ability to achieve fair allocations. These tests, and the others we have run but cannot show here because of space limitations, suggest that CSFQ achieves a reasonable approximation of fair bandwidth allocations in most conditions. Certainly, CSFQ is far superior in this regard to the status quo (FIFO or RED). Moreover, in all situations, CSFQ is roughly comparable with FRED, and in some cases it achieves significantly fairer allocations. Recall that FRED requires per-packet flow classification while CSFQ does not, so we are achieving these levels of fairness in a more scalable manner. However, there is clearly room for improvement in CSFQ. We think our buffer management algorithm may not be well tuned to the vagaries of TCP buffer usage, and so are currently looking at adopting an approach closer in spirit to RED for buffer management, while retaining the use of the labels to achieve fairness.

IV. Why Are Fair Allocations Important?

In Section I, we stated that one of the underlying assumptions of this work is that fairly allocating bandwidth was beneficial, and perhaps even crucial, for congestion control. In this section, we motivate the role of fair allocations in congestion control by discussing the problem of unfriendly flows, and then end this section with a discussion of the role of punishment. In what follows, we borrow heavily from [3], [9], and [11], and have benefited greatly from conversations with S. Deering and S. Floyd. We should note that the matters addressed in this section are rather controversial and this overview unavoidably reflects our prejudices. This section, however, is merely intended to provide some perspective on our motivation for this work, and any biases in this overview should not undercut the technical aspects of the CSFQ proposal that are the main focus of Sections II and III.

A. *Unfriendly Flow Problem*

Data networks such as the Internet, because of their reliance on statistical multiplexing, must provide some mechanism to control congestion. The current Internet, which has mostly FIFO queueing and drop-tail mechanisms in its routers, relies on end-to-end congestion control in which hosts curtail their transmission rates when they detect that the network is congested. The most widely utilized form of end-to-end congestion control is that embodied in TCP [16], which has been tremendously successful in preventing congestion collapse.

The efficacy of this approach depends on two foundational assumptions: 1) all (or almost all) flows are *cooperative* in that they implement congestion-control algorithms; and 2) these algorithms are *homogeneous*—or roughly equivalent—in that they produce similar bandwidth allocations if used in similar circumstances. In particular, assumption 2 requires, in the language of [11], that all flows are TCP-friendly.[5]

The assumption of universal cooperation can be violated in three general ways. First, some applications are *unresponsive* in that they do not implement any congestion-control algorithms at all. Most of the early multimedia and multicast applications, like *vat*, *nv*, *vic*, *wb*, and RealAudio fall into this category. Second, some applications use congestion-control algorithms that, while responsive, are not TCP-friendly. As we saw in Section III-C, RLM is such an algorithm. Third, some users will cheat and use a non-TCP congestion-control algorithm to get more bandwidth. An example of this would be using a modified form of TCP with, for instance, larger initial window and window opening constants.

Each of these forms of noncooperation can have a significant negative impact on the performance obtained by cooperating flows. At present, we do not yet know how widespread noncooperation will be and, thus, cannot assess the level of harm it will cause. However, in lieu of more solid evidence that noncooperation will not be a problem, it seems unsound to base the Internet's congestion-control paradigm on the assumption of universal cooperation. Therefore, we started this paper with the fundamental assumption that one needs to deal with the problem of unfriendly flows.

B. *Punishment*

In Section IV-A, we argued that the allocation approach gave drop-intolerant flows an incentive to adopt end-to-end congestion control; now we ask: what about drop-tolerant flows?

We consider, for illustration, *fire-hose* applications that have complete drop tolerance: they send at some high rate ρ and get as much value out of the fraction of arriving packets, call it x, as if they originally just sent a stream of rate $x\rho$. That is, these fire-hose applications care only about the ultimate throughput rate, not the dropping rate.[6] In a completely static world where bandwidth shares were constant, such fire-hose protocols would not provide any advantage over just sending at the fair share rate. However, if the fair shares along the path were fluctuating significantly, then fire-hose protocols might better utilize instantaneous fluctuations in the available bandwidth. Moreover, fire-hose protocols relieve applications of the burden of trying to adapt to their fair share. Thus, even when restrained to their fair share, there is some incentive for flows to send at significantly more than the current fair share.[7] In addition, such fire-hoses decrease the bandwidth available to other flows because packets destined to be dropped at a congested link represent an unnecessary load on upstream links. With universal deployment of the allocation approach, every other flow would still obtain their fair share at each link, but that share may be smaller (by no more than a factor of $(n-f)/n$, where n is the total number of flows and f is the number of fire-hoses) than it would have been if the fire-hose had been using responsive end-to-end congestion control. It is impossible to know now whether this will become a serious problem. Certainly, though, the problem of fire-hoses in a world with fair bandwidth allocation is far less dire than the problem of unfriendly flows in our current FIFO Internet, since the incentive to be unfriendly and the harmful impact on others are considerably greater in the latter case. As a consequence, our paper emphasizes the problem of unfriendly flows in our current FIFO Internet, and is less concerned with fire-hose flows in an Internet with fair bandwidth allocation.

Nonetheless, the fire-hose problem should not be ignored; flows should be given an incentive to adopt responsive end-to-end congestion. A possible method is to explicitly punish unresponsive flows by denying them their fair share.[8]

One way to implement this method in CSFQ is to change the computation of drop probability p [see (2)] to

$$\max\left(0, 1 - \left(\frac{\alpha(t)}{r_i(t)}\right)^2\right). \tag{9}$$

[5] Actually, the term TCP-friendly in [11] means that "their arrival rate does not exceed that of any TCP connection in the same circumstances." Here, we use it to mean that the arrival rates are roughly comparable. We blur this distinction to avoid an overly unwieldy set of terms. However, we think the distinction may be rendered moot since it is unlikely that congestion-control algorithms that are not TCP-equivalent but are TCP-friendly—i.e., they get much less than their fair share—will be widely deployed.

[6] Approximations to complete drop-tolerance can be reached in video transport using certain coding schemes or file transport using selective acknowledgment.

[7] These fire-hose coding and file transfer methods also have some overhead associated with them, and it is not clear whether, in practice, the overheads are greater or less than the advantages gained. However, one can certainly not claim, as we did above for drop-intolerant applications, that the allocation approach gives drop-tolerant applications a strong incentive to use responsive end-to-end congestion-control algorithms.

[8] Another possible method, used in ATM Available Bit Rate (ABR), is to have network provide explicit per-flow feedback to ingress nodes and have edge nodes police the traffic on a per-flow basis. We assume this is too heavyweight a mechanism for the Internet.

As a result, a flow i with an arrival rate k (> 1) times larger than the fair rate α will receive an allocation of only α/k, instead of α as in the original CSFQ. Thus, the higher the bandwidth of a flow is, the less bandwidth it receives. Another important point to note here is that, unlike other proposals that aim to punish high-bandwidth flows [11], our solution does not require core routers to maintain any state.

V. Related Work

There are, in the literature, two general approaches that address the problem of unfriendly flows: 1) the allocation approach, and 2) the identification approach. In the rest of this section, we discuss these two approaches in more detail.

A. Allocation Approach

In the allocation approach, a router isolates flows from each other by allocating the bandwidth fairly between them. As a result, unfriendly flows can only have a very limited impact on other flows. It is important to note that the allocation approach does not demand that all flows adopt some universally standard end-to-end congestion-control algorithm; flows can choose to respond to the congestion in whatever manner that best suits them without harming other flows.

1) Stateful Solutions: Traditionally, fair allocations were achieved by using *per-flow* mechanisms such as Fair Queueing [9], [26] and its many variants [2], [14], [29], or per-flow dropping mechanisms such as FRED [20]. However, these mechanisms require each router to maintain state and perform operations on a per-flow basis. For each packet that arrives at the router, the router needs to classify the packet into a flow, update per-flow state variables, and perform certain operations based on the per-flow state. The operations can be as simple as deciding whether to drop or queue the packet (e.g., FRED), or as complex as manipulation of priority queues (e.g., Fair Queueing). While a number of techniques have been proposed to reduce the complexity of the per-packet operations [1], [29], [33], and commercial implementations are available in some intermediate class routers, it is still unclear whether these algorithms can be cost-effectively implemented in high-speed backbone routers because all these algorithms still require packet classification and per-flow state management.

While in this paper we have assumed that a packet is classified based on its source and destination IP addresses, in general a packet can be classified on any combination of *partially* specified fields in the packet header. Besides source and destination IP addresses, these fields usually include the port numbers and the protocol type.[9] Unfortunately, the general problem of classification is inherently difficult. Current solutions [15], [19], [31], [32] work well only for a relatively small number of classes, i.e., no more than several thousand. In particular, Gupta and McKeown [15] have shown that the packet classification problem is similar to the point location problem in the domain of computation geometry. Given a point in an F-dimensional space, this problem asks to find the enclosing region among a set of regions. In the case of nonoverlapping regions, the best bounds for n regions in an F-dimensional space are $O(\log n)$ in time and $O(n^F)$ in space, or, alternatively, $O(\log^{F-1} n)$ in time and $O(n)$ in space. This suggests a clear tradeoff between space and time complexities. It also suggests that it is very hard to simultaneously achieve both speed and efficient memory usage.

2) Stateless Solutions: Recently, several solutions using a CSFQ-like architecture were proposed to address the scalability limitations of traditional fair queueing solutions [5], [7], [35]. Like CSFQ, these solutions differentiate between edge and core routers. Edge routers maintain per-flow state and insert state in the packet headers, while core routers maintain no per-flow state. However, unlike CSFQ where the packet labels represent the estimated rate of their flow, in these schemes the packets of the same flow are marked with *different* labels. In turn, each core router computes a threshold and then drops all packets with labels higher than this threshold.[10] By properly assigning labels to the packets and computing the threshold at a congested link such that the sum of the allocated rates equals the link capacity, all these schemes are able to accurately approximate fair queueing. One potential advantage of these solutions over CSFQ is that they do not require core routers to relabel packets. In addition, Tag Unified Fair Queueing differentiates between the TCP and UDP flows in order to avoid the TCP flows being overpenalized due to their response to losses [7].

CHOKe is another recent proposal for approximating fair queueing [25]. With CHOKe, an incoming packet is matched against a random packet in the buffer. If the packets belong to the same flow both packets are dropped, otherwise the packet is enqueued. As a result, this simple mechanism preferentially drops the packets of high-bandwidth flows. While CHOKe is very simple to implement and does not require edge routers to maintain any per-flow state, it may fail to accurately approximate fair queueing when the number of flows is large or in the presence of very high-speed flows. For instance, the simulations reported in [25] show that a high-speed UDP can get several times more bandwidth than it deserves.

3) ATM ABR: The problem of estimating the fair share rate has also been studied in the context of designing ABR algorithms for ATM networks. While the problems are similar, there are also several important differences. First, in ATM ABR, the goal is to provide explicit feedback to end systems for policing purposes so that cell losses within the network can be prevented. In CSFQ, there is no explicit feedback and edge policing. Packets from a flow may arrive at a much higher rate than the flow's fair share rate and the goal of CSFQ is to ensure, by probabilistic dropping, that such flows do not get more service than their fair shares. Second, since ATM already keeps per-virtual-circuit (VC) state, additional per-VC state is usually added to improve the accuracy and reduce the time complexity of estimating the fair share rate [6], [10], [18]. However, there are several algorithms that try to estimate the fair share without keeping per-flow state [27], [30]. These algorithms rely on the flow rates communicated by the end system. These estimates

[9] For instance, one could use a general packet classifier in conjunction with the weighted version of CSFQ to allocate a higher bandwidth to the Web traffic, or to control the allocation of the entire traffic between two given subnetworks.

[10] Tag Unified Fair Queueing is slightly different in that it drops the packet with the highest label from the queue when the queue overflows [7].

are assumed to remain accurate over multiple hops, due to the accurate explicit congestion control provided by ABR. In contrast, in CSFQ, since the number of dropped packets cannot be neglected, the flow rates are recomputed at each router, if needed (see Section II-B3).

B. Identification Approach

In the *identification* approach, a router uses a lightweight detection algorithm to identify ill-behaved (unfriendly) flows [11], [21]. Once a router detects the ill-behaved flows, it uses either a scheduling scheme such as Class-Based Queueing [13] or preferential dropping to manage the bandwidth of these unfriendly flows. This bandwidth management can range from merely restricting unfriendly flows to no more than the current highest friendly flow's share to the extreme of severely punishing unfriendly flows by dropping all their packets.

While these solutions can be incrementally deployed on a router-by-router basis, they have several drawbacks when compared to CSFQ. First, these solutions require routers to maintain state for each flow that has been classified as unfriendly. In contrast, CSFQ does not require core routers to maintain any per-flow state. Second, designing accurate identification tests for unfriendly flows is inherently difficult. This is mainly because the throughput of a TCP depends on the round-trip time (RTT), and it is very hard for a router to accurately estimate RTT with only local information. For example, in [34] we show using simulations that the solution presented in [11] may fail to identify an unfriendly flow that uses up to three times more bandwidth than a regular TCP flow. Finally, the identification approach requires that all flows implement a similar congestion-control mechanism, i.e., TCP-friendly. We believe this is overly restrictive as it severely limits the freedom of designing new congestion protocols that best suit application needs.

VI. Summary

This paper presents an architecture for achieving reasonably fair bandwidth allocations while not requiring per-flow state in core routers. Edge routers estimate flow rates and insert them into the packet labels. Core routers merely perform probabilistic dropping on input based on these labels and an estimate of the fair share rate, the computation of which requires only aggregate measurements. Packet labels are rewritten by the core routers to reflect output rates, so this approach can handle multihop situations.

We tested CSFQ and several other algorithms on a wide variety of conditions. We find that CSFQ achieves a significant degree of fairness in all of these circumstances. While not matching the fairness benchmark of DRR, it is comparable or superior to FRED and vastly better than the baseline cases of RED and FIFO.

The main thrust of CSFQ is to use rate estimation at the edge routers and packet labels to carry rate estimates to core routers. The details of our proposal, such as the exact form of buffer management or the constants used in the averaging procedures, are still very much the subject of active research. However, the results of our initial experiments with a rather untuned algorithm are quite encouraging.

One open question is the effect of large latencies. The logical extreme of the CSFQ approach would be to do rate estimation at the entrance to the network (at the customer/ISP boundary), and then consider everything else the core. This introduces significant latencies between the point of estimation and the points of congestion; while our initial simulations with large latencies did not reveal any significant problems, we do not yet understand CSFQ well enough to be confident in the viability of this "all-core" design. However, if viable, this "all-core" design would allow all interior routers to have only very simple forwarding and dropping mechanisms, without any need to classify packets into flows.

One of the initial assumptions of this paper was that the more traditional mechanisms used to achieve fair allocations, such as Fair Queueing or FRED, were too complex to implement cost effectively at sufficiently high speeds. If this is the case, then a more scalable approach like CSFQ is necessary to achieve fair allocations. The CSFQ islands would be comprised of high-speed backbones, and the edge routers would be at lower speeds where classification and other per-flow operations were not a problem. However, CSFQ may still play a role even if router technology advances to the stage where the more traditional mechanisms can reach sufficiently high speeds. Because the core version of CSFQ could presumably be retrofit on a sizable fraction of the installed router base (since its complexity is roughly comparable to RED and can be implemented in software), it may be that CSFQ islands are not high-speed backbones but rather are comprised of legacy routers.

Finally, we should note that the CSFQ approach requires some configuration, with edge routers distinguished from core routers. Moreover, CSFQ must be adopted one island at a time rather than router by router. We do not know if this presents a serious impediment to CSFQ's adoption.

References

[1] J. C. R. Bennett, D. C. Stephens, and H. Zhang, "High speed, scalable, and accurate implementation of packet fair queueing algorithms in ATM networks," in *Proc. IEEE Int. Conf. Network Protocols*, Atlanta, GA, Oct. 1997, pp. 7–14.

[2] J. C. R. Bennett and H. Zhang, "WF^{2}Q: Worst-case fair weighted fair queueing," in *Proc. IEEE INFOCOM*, San Francisco, CA, Mar. 1996, pp. 120–128.

[3] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. K. Ramakrishnan, S. Shenker, and J. Wroclawski. (1998) Recommendations on queue management and congestion avoidance in the Internet. LBL Network Research Group. [Online]. Available: http://ftp.ee.lbl.gov/nrg-other.html

[4] R. Callon, P. Doolan, N. Feldman, A. Fredette, G. Swallow, and A. Viswanathan. (1997) A framework for multiprotocol label switching. IETF. [Online]draft-ietf-mpls-framework-02.txt

[5] Z. Cao, Z. Wang, and E. Zegura, "Rainbow fair queueing: Fair bandwidth sharing without per-flow state," in *Proc. INFOCOM*, Tel-Aviv, Israel, Mar. 2000, pp. 922–931.

[6] A. Charny, "An algorithm for rate allocation in a packet-switching network with feedback," Master's thesis, Massachusetts Inst. Technol., Comput. Sci. Div., Cambridge, MA, 1994.

[7] A. Clerget and W. Dabbous, "Tag-based unified fairness," in *Proc. IEEE INFOCOM*, Anchorage, AK, Apr. 2001, pp. 498–507.

[8] M. E. Crovella and A. Bestavros, "Self-similarity in World-Wide Web traffic evidence and possible causes," in *Proc. ACM SIGMETRICS*, Philadelphia, PA, May 1996, pp. 160–169.

[9] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *J. Internetw. Res. Experience*, pp. 3–26, Oct. 1990.

[10] S. Fahmy, R. Jain, S. Kalyanaraman, R. Goyal, and B. Vandalore, "On determining the fair bandwidth share for ABR connections in ATM networks," in *Proc. IEEE Int. Conf. Communications (ICC)*, vol. 3, Atlanta, GA, June 1998, pp. 1485–1491.

[11] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the internet," *IEEE/ACM Trans. Networking*, vol. 7, pp. 458–472, Aug. 1999.

[12] S. Floyd and V. Jacobson, "Random early detection for congestion avoidance," *IEEE/ACM Trans. Networking*, vol. 1, pp. 397–413, July 1993.

[13] ——, "Link-sharing and resource management models for packet networks," *IEEE/ACM Trans. Networking*, vol. 3, pp. 365–386, Aug. 1995.

[14] S. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *Proc. IEEE INFOCOM*, Toronto, ON, Canada, June 1994, pp. 636–646.

[15] P. Gupta and N. McKeown, "Packet classification on multiple fields," in *Proc. ACM SIGCOMM*, Sept. 1999, pp. 147–160.

[16] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SIGCOMM*, Aug. 1988, pp. 314–329.

[17] J. Jaffe, "Bottleneck flow control," *IEEE Trans. Commun.*, vol. 29, pp. 954–962, July 1981.

[18] R. Jain, S. Kalyanaraman, R. Goyal, S. Fahmy, and R. Viswanathan, "ERICA switch algorithm: A Complete description," ATM Forum, 96-1172, 1996.

[19] T. V. Lakshman and D. Stiliadis, "High speed policy-based packet forwarding using efficient multi-dimensional range matching," in *Proc. ACM SIGCOMM*, Vancouver, BC, Canada, Sept. 1998, pp. 203–214.

[20] D. Lin and R. Morris, "Dynamics of random early detection," in *Proc. ACM SIGCOMM*, Cannes, France, Oct. 1997, pp. 127–137.

[21] R. Mahajan, S. Floyd, and D. Wheterall, "Controlling high-bandwidth flows at the congested route," in *Proc. IEEE Int. Conf. Network Protocols*, Riverside, CA, Nov. 2001, pp. 192–201.

[22] S. McCanne, "Scalable compression and transmission of internet multicast video," Ph.D. dissertation, Comput. Sci. Div., Univ. California, Berkeley, 1996.

[23] J. Nagle, "On packet switches with infinite storage," *IEEE Trans. Commun.*, vol. 35, pp. 435–438, Apr. 1987.

[24] Ucb/lbnl/vint Network Simulator—ns (Version 2) [Online]. Available: http://www-mash.cs.berkeley.edu/ns/

[25] R. Pan, B. Prabhakar, and K. Psounis, "CHOKe—A stateless active queue management scheme for approximating fair bandwidth allocation," in *Proc. IEEE INFOCOM*, Tel Aviv, Israel, Mar. 2000, pp. 942–951.

[26] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control—The single node case," *IEEE/ACM Trans. Networking*, vol. 1, pp. 344–357, June 1993.

[27] L. Roberts, "Enhanced PRCA (proportional rate control algorithm)," ATM Forum, 94-0735R1, 1994.

[28] S. Shenker, "Making greed work in networks: A game theoretical analysis of switch service disciplines," in *Proc. ACM SIGCOMM*, London, U.K., Aug. 1994, pp. 47–57.

[29] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in *Proc. ACM SIGCOMM*, Boston, MA, Sept. 1995, pp. 231–243.

[30] K. Y. Siu and H. Y. Tzeng, "Intelligent congestion control for ABR service in ATM networks," Dept. Electr. Comput. Eng., Univ. California, Irvine, CA, Tech. Rep. 1102, 1994.

[31] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," in *Proc. ACM SIGCOMM*, Cambridge, MA, Sept. 1999, pp. 135–146.

[32] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast scalable algorithms for level four switching," in *Proc. ACM SIGCOMM*, BC, Canada, Sept. 1998, pp. 191–202.

[33] D. Stilliadis and A. Varma, "Efficient fair queueing algorithms for packet-switched networks," *IEEE/ACM Trans. Networking*, vol. 6, pp. 175–185, Apr. 1998.

[34] I. Stoica, S. Shenker, and H. Zhang, "*Core*-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks," Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-98-136, 1998.

[35] N. Venkitaraman, J. Mysore, R. Srikant, and R. Barnes, "Stateless prioritized fair queueing," SPFQ, draft-venkitaraman-diffserv-spfq-00.txt, 2000.

[36] Z. Wang, "User-share differentiation (USD) scalable bandwidth allocation for differentiated services," IETF, draft-wang-diff-serv-usd-00.txt, 1998.

Ion Stoica received the Ph.D. degree from Carnegie Mellon University, Pittsburgh, PA, in 2000.

He is currently an Assistant Professor with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, where he does research on resource management, scalable solutions for end-to-end quality of service, and peer-to-peer network technologies in the Internet.

Dr. Stoica is the recipient of a National Science Foundation CAREER Award in 2002, and the Association for Computing Machinery (ACM) Doctoral Dissertation Award in 2001. He is a member of the ACM.

Scott Shenker (M'87–SM'96–F'00) received the B.Sc. degree from Brown University, Providence, RI, and the Ph.D. degree in theoretical physics from the University of Chicago, both in theoretical physics.

After a postdoctoral year with the Department of Physics, Cornell University, Ithaca, NY, in 1983, he joined the Xerox Palo Alto Research Center (PARC). He left PARC in 1999 to head up a newly established Internet research group at the International Computer Science Institute (ICSI), Berkeley, CA. His research over the past 15 years has ranged from computer performance modeling and computer networks to game theory and economics. Most of his recent work has focused on Internet architecture and related issues.

Dr. Shenker received the ACM SIGCOMM Award in 2002. He has been a Member of the Association for Computing Machinery (ACM) since 1986.

Hui Zhang (M'95) received the B.S. degree in computer science from Beijing University, Beijing, China, in 1988, the M.S. degree in computer engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1989, and the Ph.D. degree in computer science from the University of California at Berkeley in 1993.

He is currently Chief Technical Officer with Turin Networks and an Associate Professor with the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. His research interests are in Internet, quality-of-service, multicast, peer-to-peer systems, and metro networks.

Dr. Zhang received the National Science Foundation CAREER Award in 1996 and the Alfred Sloan Fellowship in 2000. He has been a Member of the Association for Computing Machinery (ACM) since 1995.