Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks*

Ion Stoica
CMU
istoica@cs.cmu.edu

Scott Shenker
Xerox PARC
shenker@parc.xerox.com

Hui Zhang
CMU
hzhang@cs.cmu.edu

Abstract

Router mechanisms designed to achieve fair bandwidth allocations, like 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 header based on this estimate. Core routers maintain *no* per flow state; they use FIFO 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, and discuss an alternate approach.

1 Introduction

A central tenet of the Internet architecture is that congestion control is achieved mainly through end-host algorithms. However, starting with Nagle [16], 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 [7]. As we discuss in Section 4, some maintain that fair bandwidth allocation[1] plays a necessary, not just beneficial, role in congestion control [7, 19].

Until now, fair allocations were typically achieved by using per-flow queueing mechanisms – such as Fair Queueing [7, 18] and its many variants [2, 10, 20] – or per-flow dropping mechanisms such as Flow Random Early Drop (FRED) [14]. These mechanisms are more complex to implement than traditional 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, 20, 21], 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 assumption 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 (we discuss this point more fully in Section 4). 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.

*This research was sponsored by DARPA under contract numbers N66001-96-C-8528, E30602-97-2-0287, and DABT63-94-C-0073, and by a NSF Career Award under grant number NCR-9624979. Additional support was provided by Intel Corp., MCI, and Sun Microsystems. Views and conclusions contained in this document are those of the authors and should no be interpreted as representing the official policies, either expressed or implied, of DARPA, NSF, Intel, MCI, Sun, or the U.S. government.

[1] We use the max-min definition of fairness [12] which, while not the only possible candidate for fairness, is certainly a reasonable one and, moreover, can be implemented with only local information.

In this approach, we identify an *island* of routers[2] and distinguish between the edge and the core of the island. Edge 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 ISP's, 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 [7] or RED [9], 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.[3] We describe the details of this approach – such as the rate estimation algorithm and the packet dropping algorithm – in Section 2.

Such a scheme cannot hope to achieve the nearly-perfect levels of fairness obtained by Fair Queueing and other sophisticated and stateful queueing algorithms. However, our interest is not in perfection, but only in obtaining reasonable approximations to the fair bandwidth allocations. We derive a worst-case bound for the performance of this algorithm in an idealized setting. This bound is presented in Section 2.

This worst-case analysis does not give an adequate guide to the typical functioning of CSFQ. In Section 3 we present results from simulation experiments to illustrate the performance of our approach and to compare it to several other schemes: DRR (a variant of Fair Queueing), FRED, RED, and FIFO. We also discuss, therein, the relative mechanistic complexities of these approaches.

The first 3 sections of the paper are narrowly focussed 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 4 we return to the basic question of why fair allocations are relevant to congestion control. Allocating bandwidth fairly is one way to address what we call the *unfriendly flow problem*; we also discuss an alternate approach to addressing this problem, the *identification* approach as described in [8]. We conclude with a summary in Section 5. A longer version of this paper, containing proofs of the theoretical results as well as more complete pseudocode, can be found at `http://www.cs.cmu.edu/~istoica/csfq`.

2 Core-Stateless Fair Queueing (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 (non-edge) 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.

2.1 Fluid Model Algorithm

We first consider a bufferless fluid model of a router with output link speed C, where the flows are modelled 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 (*i.e.*, have bits dropped) by this router have the same output rate. We call this rate the *fair share rate* of the server; 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 server's fair share rate, all of its traffic will be forwarded. If $r_i(t) > \alpha(t)$, then a fraction $\frac{r_i(t)-\alpha(t)}{r_i(t)}$ 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

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

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

2.2 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.

[2] By island we mean a contiguous portion of the network, with well-defined interior and edges.

[3] 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.

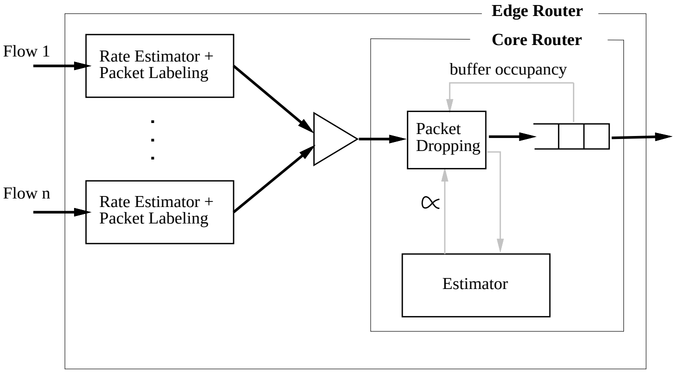


Figure 1: The architecture of the output port of an edge router, and a core router, respectively.

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 the next two subsections, and then discuss the rewriting of the labels. Pseudocode reflecting this algorithm is described in Figure 2. 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.

2.2.1 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 k^{th} packet of flow i. The estimated rate of flow i, r_i, is updated every time a new packet is received:

$$r_i^{new} = (1 - e^{-T_i^k/K})\frac{l_i^k}{T_i^k} + e^{-T_i^k/K} r_i^{old}, \qquad (3)$$

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 2.7. In the longer version of this paper [22] we show that, under a wide range of conditions, this estimation algorithm converges.

2.2.2 Link Fair Rate Estimation

In this section, we present an estimation algorithm for $\alpha(t)$. To give intuition, consider again the fluid model in Section 2.1 where the arrival rates are known exactly, and assume the system performs the probabilistic dropping algorithm according to Eq. (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

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

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 Eq (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 inter-arrival time between the current and the previous packet:

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

where $\widehat{A}_{old}$ 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 the equation $F(\widehat{\alpha}) = C$. We approximate $F(\cdot)$ by a linear function that intersects the origin and has slope $\widehat{F}/\widehat{\alpha}_{old}$. This yields

$$\widehat{\alpha}_{new} = \widehat{\alpha}_{old}\frac{C}{\widehat{F}} \qquad (6)$$

If the link is not congested, $\widehat{\alpha}_{new}$ 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 $\widehat{\alpha}_{new}$ is then used to compute dropping probabilities, according to Eq. (2). For completeness, we give the pseudocode of the CSFQ algorithm in Figure 2.

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 [9], it is important to limit its effect. To do so, we use a simple heuristic: every

```
on receiving packet p
  if (edge router)
    i = classify(p);
    p.label = estimate_rate(r_i, p); /* use Eq. (3) */
  prob = max(0, 1 − α/p.label);
  if (prob > unif_rand(0, 1))
    α = estimate_α (p, 1);
    drop(p);
  else
    α = estimate_α (p, 0);
    enqueue(p);
  if (prob > 0)
    p.label = α; /* relabel p */

estimate_α (p, dropped)
  estimate_rate(Â, p); /* est. arrival rate (use Eq. (5)) */
  if (dropped == FALSE)
    estimate_rate(F̂, p); /* est. accepted traffic rate */
  if (Â ≥ C)
    if (congested == FALSE)
      congested = TRUE;
      start_time = crt_time;
    else
      if (crt_time > start_time + K_c)
        α̂ = α̂ × C/F̂;
        start_time = crt_time;
  else /* Â < C */
    if (congested == TRUE)
      congested = FALSE;
      start_time = crt_time;
      tmp_α = 0; /* use to compute new α */
    else
      if (crt_time < start_time + K_c)
        tmp_α = max(tmp_α, p.label);
      else
        α̂ = tmp_α;
        start_time = crt_time;
        tmp_α = 0;
  return α̂;
```

Figure 2: The pseudocode of CSFQ.

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%.

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 Figure 2, 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.

2.2.3 Label Rewriting

Our rate estimation algorithm in Section 2.2.1 allows us to label packets with their flow's rate as they enter the island. Our packet dropping algorithm described in Section 2.2.2 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 note in Section 2.1 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

$$L_{new} = \min(L_{old}, \alpha), \qquad (7)$$

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

2.3 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 $\frac{r_i}{w_i}$. Then, if $A(t) > C$, the *normalized* fair rate $\alpha(t)$ is the unique value such that $\sum_{i=1}^{n} w_i \min\left(\alpha, \frac{r_i}{w_i}\right) = C$. The expression for the dropping probabilities in the weighted case is $\max\left(0, 1 - \alpha\frac{w_i}{r_i}\right)$. The only other major change is that the label is now r_i/w_i, instead simply r_i. Finally, without going into details 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 [24].

2.4 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 [22].

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 Eq. (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*

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

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.

2.5 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. 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.

2.6 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 a source-destination 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 ISP's 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 ISP's 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.

2.7 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 Eq. (3). However, instead of using a constant exponential weight we used $e^{-T/K}$ where T is the inter-packet 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 Eq. (5) to estimate the total arrival rate A.

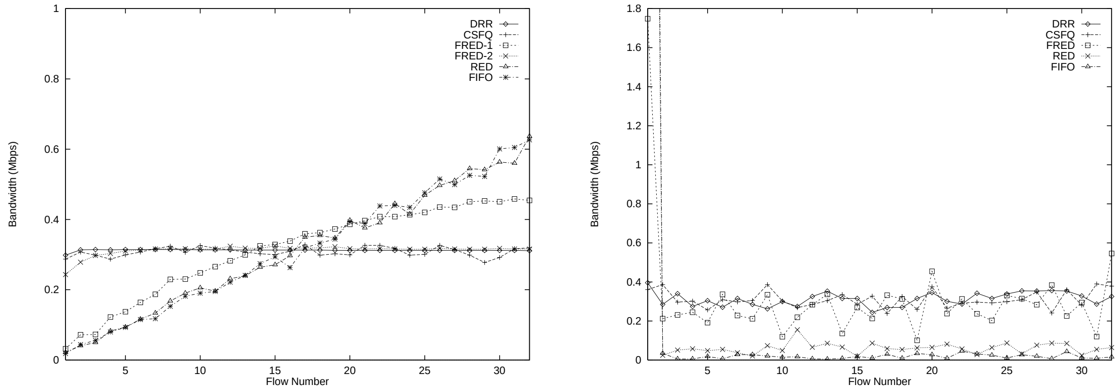
The choice of K in the above expression $e^{-T/K}$ presents us with several tradeoffs. First, while a smaller K increases the system responsiveness to rapid rate fluctuations, a larger K better filters the noise and avoids potential system instability. Second, K should be large enough such that the estimated 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' inter-arrival 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 this 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 four bits for mantissa and four bits for exponent we can represents any rate between 1 Kbps and 65 Mbps 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.

3 Simulations

In this section we evaluate our algorithm by simulation. To provide some context, we compare CSFQ's performance to four 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 (First In First Out) - 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 (Random Early Detection) - Packets are served in a first-in first-out order, but the buffer management is significantly more sophisticated than drop-tail. RED [9] starts to probabilistically drop packets long



(a) (b)

Figure 3: Simulation results for a 10 Mbps link shared by N flows. (a) The average throughput over 10 sec when N = 32, and all flows are UDPs. The arrival rate for flow i is (i + 1) times larger than its fair share. The flows are indexed from 0. (b) The throughputs of one UDP flow (indexed 0) sending at 10 Mbps, and of 31 TCP flows sharing a 10 Mbps link.

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 (Flow Random Early Drop) - This algorithm extends RED to provide some degree of fair bandwidth allocation [14]. 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 better.

- DRR (Deficit Round Robin) - This algorithm [20] 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 TCP sources and constant bit rate UDP sources,[4] but also some on-off sources) and topologies. Due to space limitations, we only report on a small sampling of the simulations we have run.[5] All simulations were performed in ns-2 [17], which provide accurate packet-level implementation for various network protocols, such as TCP and RLM [15] (Receiver-driven Layered Multicast), and various buffer management and scheduling algorithms, such as RED and DRR. All algorithms used in the simulation, except CSFQ and FRED, were part of the standard ns-2 distribution.

Unless otherwise specified, we use the following parameters for the simulations in this section. Each output link has a capacity of 10 Mbps, a latency of 1 ms, and a buffer of 64 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 constants K (used in estimating the flow rate), K_α (used in estimating the fair rate), and K_c (used in making the decision of whether a link is congested or not) are all set to 100 ms unless specified otherwise. The general rule of thumb we follow in this paper is to choose these constants to be roughly two times larger than the maximum queueing delay (*i.e.*, 64KB/10Mbps = 51.2 ms).[6] Finally, in all topologies the first router on the path of each flow is always assumed to be an 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

[4] This source, referred to as UDP in the remainder of the paper, has fixed size packets and the packet interarrival times are uniformly distributed between $[0.5 \times avg, 1.5 \times avg)$, where avg is the average interarrival time.

[5] A fuller set of tests, and the scripts used to run them, is available at `http://www.cs.cmu.edu/~istoica/csfq`

[6] It can be shown that by using this rule an idle link that becomes suddenly congested by a set of identical UDP sources will not experience buffer overflow *before* the algorithm detects the congestion, as long as the aggregate arrival rate is less than 10 times the link capacity (see [22]).

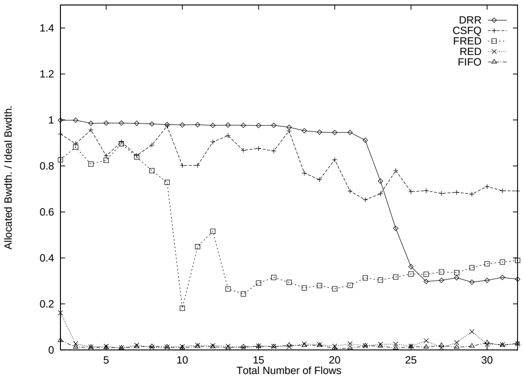


Figure 4: The normalized bandwidth of a TCP flow that competes with $N-1$ UDP flows sending at twice their allocated rates, as a function of N.

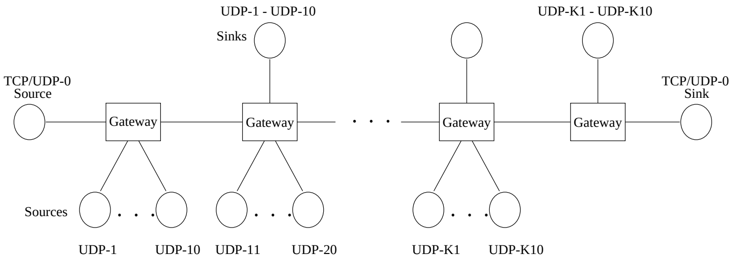


Figure 5: Topology for analyzing the effects of multiple congested links on the throughput of a flow. Each link has ten cross flows (all UDPs). All links have 10 Mbps capacities. The sending rates of all UDPs, excepting UDP-0, are 2 Mbps, which leads to all links between routers being congested.

RED, as representing the (unfair) status quo. The goal of these experiments is to 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 several 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 sections that follow we will merely highlight a few important points and omit detailed explanations of the dynamics.

3.1 A Single Congested Link

We first consider a single 10 Mbps congested link shared by N flows. The propagation delay along the link is 1 ms. We performed three related experiments.

In the first experiment, we have 32 UDP flows, indexed from 0, where flow i sends $i+1$ times more than its fair share of 0.3125 Mbps. Thus flow 0 sends 0.3125 Mbps, flow 1 sends 0.625 Mbps, and so on. Figure 3(a) shows the average throughput of each flow over a 10 sec 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 $+5\%$ of the ideal value.

In the second experiment we consider the impact of an ill-behaved UDP flow on a set of TCP flows. More precisely, the traffic of flow 0 comes from a UDP source that sends at 10 Mbps, while all the other flows (from 1 to 31) are TCPs. Figure 3(b) shows the throughput of each flow averaged over a 10 sec interval. The only two algorithms that can most effectively contain the UDP flow are DRR and CSFQ. Under FRED the UDP flow gets almost 1.8 Mbps – close to six times more than its fair share – while the UDP only gets 0.396 Mbps and 0.361 Mbps under DRR and CSFQ, respectively. As expected FIFO and RED perform poorly, with the UDP flow getting over 8 Mbps in both cases.

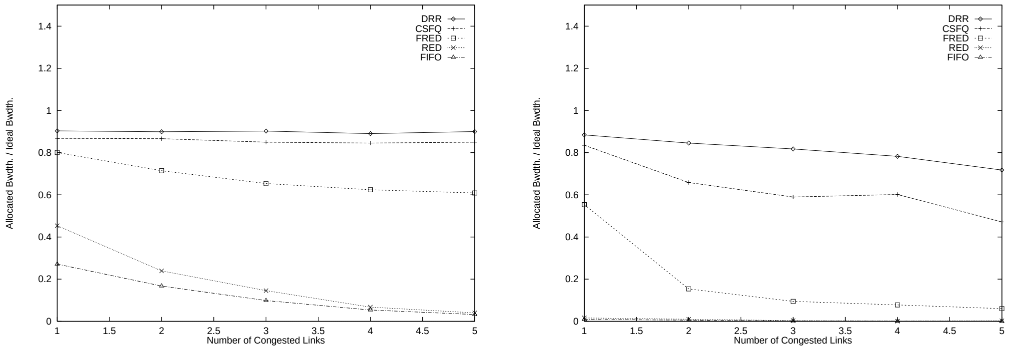
In the final experiment, we measure how well the algorithms can protect a single TCP flow against multiple ill-behaved flows. We perform 31 simulations, each for a different value of N, $N = 2 \cdots 32$. In each simulation we take one TCP flow and $N-1$ UDP flows; each UDP sends at twice its fair share rate of $\frac{10}{N}$Mbps. Figure 4 plots the ratio between the average throughput of the TCP flow over 10 seconds and the fair share bandwidth it should receive as a function of the total number of flows in the system N. There are three points of interest. First, DRR performs very well when there are less than 22 flows, but its performances decreases afterwards. This is because the TCP flow's buffer share is less than three buffers, which significantly affects 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 several packets buffered for short time intervals. Finally, across the entire range, CSFQ provides similar or better performance as compared to FRED.

3.2 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 Figure 5. All UDPs, except UDP-0, send at 2 Mbps. Since each link in the system has 10 Mbps capacity, this will result in all links between routers being congested.

In the first experiment, we have a UDP flow (denoted UDP-0) sending at its fair share rate of 0.909 Mbps. Figure 6(a) shows the fraction of UDP-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.

In the second experiment we replace UDP-0 with a TCP flow. Similarly, Figure 6(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 buffers fairly. Flows with different end-to-end congestion control algorithms will achieve different throughputs even if routers try to fairly allocate buffer.



(a) (b)

Figure 6: (a) The normalized throughput of UDP-0 as a function of the number of congested links. (b) The same plot when UDP-0 is replaced by a TCP flow.

Algorithm	delivered	dropped
DRR	601	6157
CSFQ	1680	5078
FRED	1714	5044
RED	5322	1436
FIFO	5452	1306

Table 1: Statistics for an ON-OFF flow with 19 competing TCPs flows (all numbers are in packets).

Algorithm	mean time	std. dev
DRR	25	99
CSFQ	62	142
FRED	40	174
RED	592	1274
FIFO	840	1695

Table 2: The mean transfer times (in ms) and the corresponding standard deviations for 60 short TCPs in the presence of a UDP flow that sends at the link capacity, *i.e.*, 10 Mbps.

3.3 Coexistence of Different Adaptation Schemes

In this experiment we investigate the extent to which CSFQ can deal with flows that employ different adaptation schemes. Receiver-driven Layered Multicast (RLM) [15] 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 Mbps link traversed by one TCP and three RLM flows. Each source uses a seven layer encoding, where layer i sends 2^{i+4} Kbps; each layer is modeled by a UDP traffic source. The fair share of each flow is 1Mbps. In the RLM case this will correspond to each receiver subscribing to the first five layers[7].

The receiving rates averaged over 1 second interval for each algorithm are plotted in Figure 7. Since in this experiment the link bandwidth is 4 Mbps and the router buffer size

Algorithm	mean	std. dev
DRR	6080	64
CSFQ	5761	220
FRED	4974	190
RED	628	80
FIFO	378	69

Table 3: The mean throughputs (in packets) and standard deviations for 19 TCPs in the presence of a UDP flow along a link with propagation delay of 100 ms. The UDP sends at the link capacity of 10 Mbps.

is 64 KB, we set constants K, K_α, and K_c to be 250 ms, i.e., about two times larger than the maximum queue delay. An interesting point to notice is that, unlike DRR and CSFQ, FRED does not provide fair bandwidth allocation in this scenario. Again, as discussed in Section 3.2, this is due to the fact that RLM and TCP use different end-to-end congestion control algorithms.

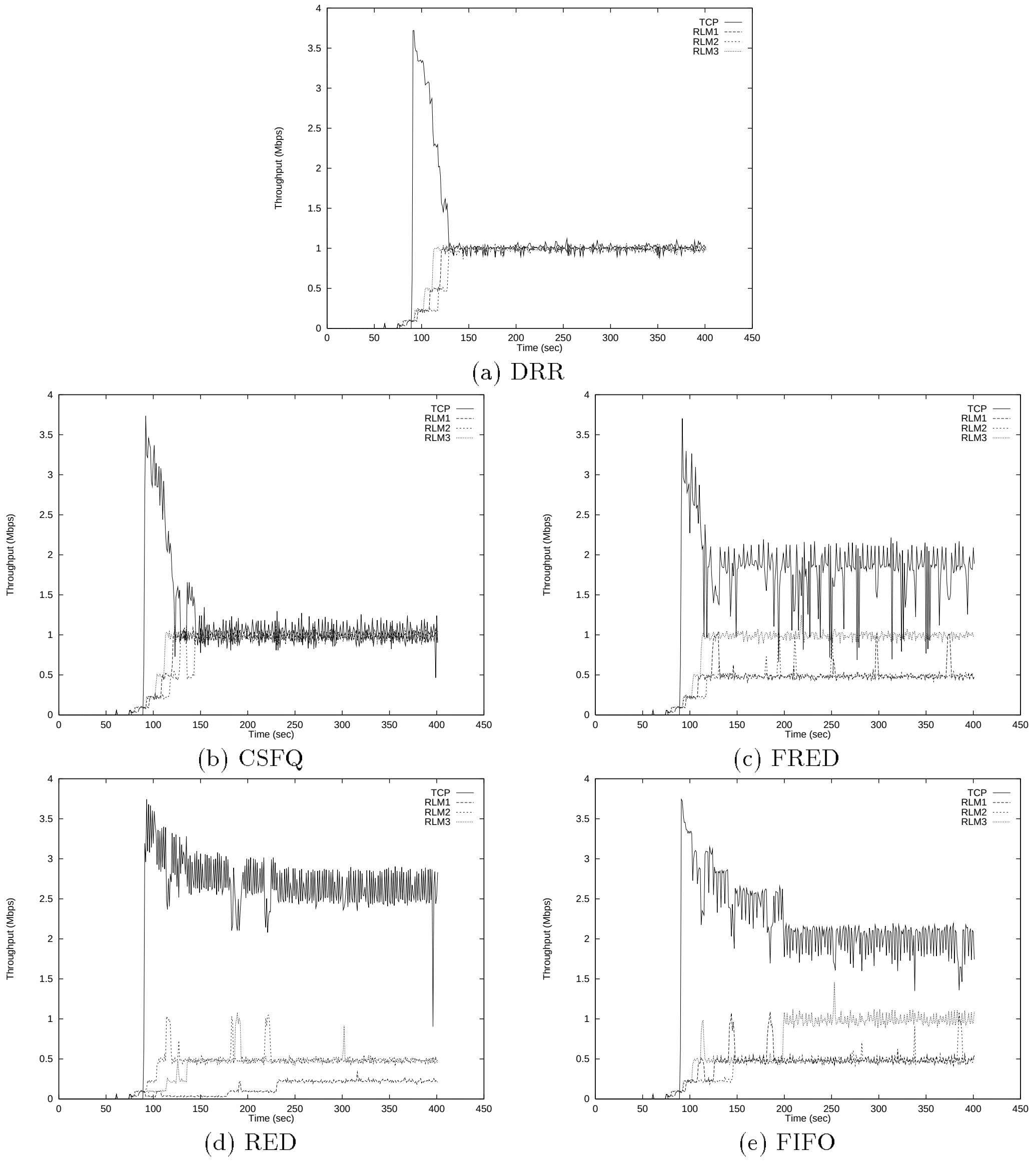
3.4 Different Traffic Models

So far we have only considered UDP, TCP and layered multicast traffic sources. We now look at two additional source models with greater degrees of burstiness. We again consider a single 10 Mbps congested link. In the first experiment, this link is shared by one ON-OFF source and 19 TCPs. The ON and OFF periods of the ON-OFF source are both drawn from exponential distributions with means of 100 ms and 1900 ms respectively. During the ON period the ON-OFF source sends at 10 Mbps. Note that the ON-time is on the same order as the averaging intervals K, K_α, and K_c which are all 100 ms, so this experiment is designed to test to what extent CSFQ can react over short timescales.

The ON-OFF source sent 6758 packets over the course of the experiment. Table 1 shows the number of packets from the ON-OFF source 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 inter-arrival times are exponentially distributed with the mean of 0.05 ms, and the length of each

[7] More precisely, we have $\sum_{i=1}^{5} 2^{i+4}$ Kbps = 0.992 Mbps.



(a) DRR

(b) CSFQ

(c) FRED

(d) RED

(e) FIFO

Figure 7: The throughput of three RLM flows and one TCP flow along a 4 Mbps link .

transfer is drawn from a Pareto distribution with a mean of 20 packets (1 packet = 1 KB) and a shaping parameter of 1.06. These values are consistent with those presented in the [5]. In addition, there is a single 10 Mbps UDP flow.

Table 2 presents the mean transfer time and the corresponding standard deviations. Here, CSFQ performs worse than FRED, mainly because it has a larger average queue size, but still almost one order of magnitude better than FIFO and RED.

3.5 Large Latency

All of our experiments so far have had small link delays (1 ms). In this experiment we again consider a single 10 Mbps congested link, but now with a propagation delay of 100 ms. The load is comprised of one UDP flow that sends at the link speed and 19 TCP flows. Due to the large propagation delay, in this experiment we set the buffer size to be 256 KB, and K, K_α, and K_c to be 400 ms. Table 3 shows the average number of packets of a TCP flow during a 100 seconds interval. Both CSFQ and FRED perform reasonably well.

3.6 Packet Relabeling

Recall that when the dropping probability of a packet is non-zero we relabel it with the fair rate α 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 Figure 8, where each link is 10 Mbps. 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 Mbps) than on link 1 (5 Mbps), so there will be dropping on both links. This will test the relabelling 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 UDPs sending data

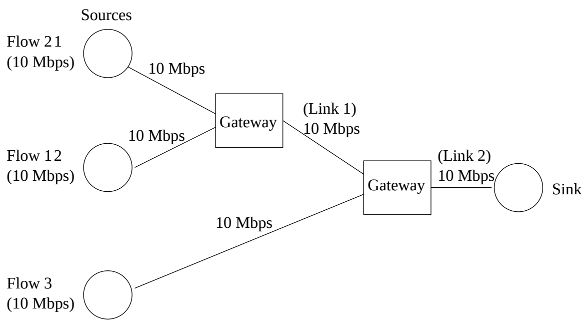


Figure 8: Simulation scenario for the packet relabeling experiment. Each link has 10 Mbps capacity, and a propagation delay of 1 ms.

Traffic	Flow 1	Flow 2	Flow 3
UDP	3.36	3.32	3.28
TCP	3.43	3.13	3.43

Table 4: The throughputs resulting from CSFQ averaged over 10 seconds for the three flows in Figure 8 along link 2.

at 10 Mbps each. Table 4 shows the average throughputs over 10 sec of the three UDP flows. As expected these rates are closed to 3.33 Mbps. In the second experiment, we replace the three UDPs by three TCPs. Again, despite the TCP burstiness which may negatively affect the rate estimation and relabeling accuracy, each TCP gets close to its fair share.

3.7 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, as there are cases where its performance is significantly below that of its benchmark, DRR. We do not yet know if these are due to our particular choices for the estimation algorithms, or are inherent properties of the CSFQ architecture.

4 Why Are Fair Allocations Important?

In the Introduction 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 presenting two approaches to this problem; we end this section with a discussion of the role of punishment. In what follows we borrow heavily from [7], [3], and [8], and have benefited greatly from conversations with Steve Deering and Sally 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 the previous sections.

4.1 The 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 [11], which has been tremendously successful in preventing congestion collapse.

The efficacy of this approach depends on two fundamental 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 [8], that all flows are TCP-friendly.[8]

The assumption of universal cooperation can be violated in three general ways. First, some applications are *unresponsive* in that they don't 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. RLM is such an algorithm.[9] 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, a 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. We therefore started this paper with the fundamental assumption that one needs to deal with the problem of unfriendly flows.

[8] Actually, the term TCP-friendly in [8] 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, a property that should be more precisely called *TCP-equivalent*. We blur the distinction between TCP-friendly and TCP-equivalent to avoid an overly unwieldy set of terms in this short overview. 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 fare share – will be widely deployed.

[9] Although our data in Section 3.3 showed RLM receiving less than its fair share, when we change the simulation scenario so that the TCP flow starts after all the RLM flows then it receives less than half of its fair share. This hysteresis in the RLM versus TCP behavior was first pointed out to us by Steve McCanne [15].

4.2 Two Approaches

There are, in the literature, two general approaches to addressing the problem of unfriendly flows. The first is the allocation approach. Here, the router itself ensures that bandwidth is allocated fairly, isolating flows from each other so that unfriendly flows can only have a very limited impact on other flows. Thus, the allocation approach need 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 best suits them without unduly harming other flows. Assuming that flows prefer to not have significant levels of packet drops, these allocation approaches give an incentive for flows to use end-to-end congestion control, because being unresponsive hurts their own performance. Note that the allocation approach does not provide an incentive for flows to be TCP-friendly (an example of an alternative end-to-end congestion control algorithm is described in [13]), but does provide strong incentives for drop-intolerant applications to use some form of end-to-end congestion control.[10] Of course, the canonical implementations of the allocation approach, such as Fair Queueing, all require significant complexity in routers. Our goal in this paper was to present a more scalable realization of the allocation approach.

The problem of unfriendly flows can be addressed in another manner. In the *identification* approach, as best exemplified by [8], routers use a lightweight detection algorithm to identify unfriendly flows, and then explicitly manage the bandwidth of these unfriendly flows. This bandwidth management can range from merely restricting unfriendly flows to no more than the currently highest friendly flow's share[11] to the extreme of severely punishing unfriendly flows by dropping all of their packets.

This approach relies on the ability to accurately identify unfriendly flows with relatively lightweight router mechanisms. This is a daunting task. Below we discuss the process of identifying unfriendly flows, and then present simulation results of the identification algorithm in [8]; we are not aware of other realizations of the identification approach.

One can think of the process of identifying unfriendly flows as occurring in two logically distinct stages. The first, and relatively easy, step is to estimate the arrival rate of a flow. The second, and harder, step is to use this arrival rate information (along with the dropping rate and other aggregate measurements) to decide if the flow is unfriendly. Assuming that friendly flows use a TCP-like adjustment method of increase-by-one and decrease-by-half, one can derive an expression (see [8] for details) for the bandwidth share S as a function of the dropping rate p, round-trip time R, and packet size B: $S \approx \frac{\gamma B}{R\sqrt{p}}$ for some constant γ. Routers do not know the round trip time R of flows, so must use the lower bound of double the propagation delay of the attached link; this allows flows further away from the link to behave more aggressively without being identified as being unfriendly.[12]

Algorithm	Simulation 1			Simulation 2	
	UDP	TCP-1	TCP-2	TCP-1	TCP-2
REDI	0.906	0.280	0.278	0.565	0.891
CSFQ	0.554	0.468	0.478	0.729	0.747

Table 5: (Simulation 1) The throughputs in Mbps of one UDP and two TCP flows along a 1.5 Mbps link under REDI [8], and CSFQ, respectively. (Simulation 2) The throughputs of two TCPs (where TCP-2 opens its congestion window three times faster than TCP-1), under REDI, and CSFQ, respectively.

To see how this occurs in practice, consider the following two experiments using the identification algorithm described in [8], which we call RED with Identification (REDI).[13] In each case there are multiple flows traversing a 1.5 Mbps link with a latency of 3 ms; the output buffer size is 32 KB and all constants K, K_α, and K_c, respectively, are set to 400 ms. Table 5 shows the bandwidth allocations under REDI and CSFQ averaged over 100 sec. In the first experiment (Simulation 1), we consider a 1 Mbps UDP flow and two TCP flows; in the second experiment (Simulation 2) we have a standard TCP (TCP-1) and a modified TCP (TCP-2) that opens the congestion window three times faster. In both cases REDI fails to identify the unfriendly flow, allowing it to obtain almost two-thirds of the bandwidth. As we increase the latency of the congested link, REDI starts to identify unfriendly flows. However, for some values as high as 18 ms, it still fails to identify such flows. Thus, the identification approach still awaits a viable realization and, as of now, the allocation approach is the only demonstrated method to deal with the problem of unfriendly flows.

4.3 Punishment

Earlier in this section we argued that the allocation approach gave drop-intolerant flows an incentive to adopt end-to-end congestion control. 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.[14] 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.[15] In addition, such

[10] As we discuss later, if flows can tolerate significant levels of loss, the situation changes somewhat.

[11] If identification were perfect, and this management goal achieved, all flows would get their max-min fair allocations. However, we are not aware of any algorithm that can achieve this management goal.

[12] We are not delving into some of the details of the approach layed out in [8] where flows can also be classified as very-high-bandwidth but not necessarily unfriendly, and as unresponsive (and therefore unfriendly).

[13] We are grateful to Sally Floyd who provided us her script implementing the REDI algorithm. We used a similar script in our simulation, with the understanding that this is a preliminary design of the identification algorithm. Our contention is that the design of such an identification algorithm is fundamentally difficult due to the uncertainty of RTT.

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

[15] These fire-hose coding and file transfer methods also have some

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 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. One possible method is to explicitly punish unresponsive flows by denying them their fair share.[16] Punishment is discussed as one possible bandwidth management approach in [8] (the approach described there is informally referred to as RED-with-a-penalty-box). Accurately identifying flows as unresponsive may be far easier than identifying them as unfriendly. However, as we saw in our simulations, doing so in the context of the identification approach is far from a solved problem; the challenge is to determine if a flow has decreased usage in response to increases in overall packet drop rates [8].

Identifying unresponsive flows is more straightforward in the allocation approach, since here one need only determine if a flow has had significantly high drop rates over a long period of time. As a proof of concept we have implemented a simple identification and punishment mechanism. First, we examine off-line the last n dropped packets and then monitor the flows with the most dropped packets. Second, we estimate the rate of each of these monitored flows; when a flow's rate is larger than $a \times \alpha$ ($a > 1$), we start dropping all of its packets. Third, we continue to monitor penalized flows, continuing punishment until their arrival rate decreases below $b \times \alpha$ ($b < 1$). Using the parameters $a = 1.2$, $b = 0.6$, and $n = 100$, we applied this algorithm to Simulation 1 in Table 5; the UDP flow was identified and penalized in less than 3 seconds. Our task was easy because the identification of unresponsive flows can be based on the result (packet drops over long periods of time) rather than on trying to examine the algorithm (detecting whether it actually decreased its rate in response to an increase in the drop rate). Note also that the allocation approach need only distinguish between responsive and unresponsive in the punishment phase, an inherently easier task than distinguishing friendly from unfriendly.

In summary, to provide incentives for drop-tolerant flows to use responsive end-to-end congestion control, it may be necessary to identify, and then punish, unresponsive flows. CSFQ with this punishment extension may be seen as a marriage of the allocation and identification approaches; the difference between [8] and our approach is largely one of the relative importance of identification and allocation. We start with allocation as fundamental, and then do identification only when necessary; [8] starts with identification, and then considers allocation only in the context of managing the bandwidth of identified flows.

5 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 achieve 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. We know of no other approach that can achieve comparable levels of fairness without any per-flow operations in the core routers.

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 estimation algorithms, 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.

In addition, we should note that it is possible to use a CSFQ-like architecture to provide service guarantees. A possible approach would be to use the route pinning mechanisms described in [23], and to shape the aggregate guaranteed traffic at each output link of core routers [6].

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

overhead associated with them, and it isn't 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.

[16] Another possible method, used in ATM 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 a too heavyweight a mechanism for the Internet.

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.

Lastly, we should note that the CSFQ approach requires some configuration, with edge routers distinguished from core routers. Moreover, CSFQ must be adopted an 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 *Proceedings of IEEE ICNP '97*, pages 7–14, Atlanta, GA, October 1997.

[2] J.C.R. Bennett and H. Zhang. WF^2Q: Worst-case fair weighted fair queueing. In *Proceedings of IEEE INFOCOM'96*, pages 120–128, San Francisco, CA, March 1996.

[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. Recommendations on queue management and congestion avoidance in the internet, January 1998. Internet Draft.

[4] R. Callon, P. Doolan, N. Feldman, A. Fredette, G. Swallow, and A. Viswanathan. A Framework for Multiprotocol Label Switching, November 1997. Internet Draft.

[5] M. E. Crovella and A. Bestavros. Self-similarity in world wide web traffic evidence and possible causes. In *Proceedings of the ACM SIGMETRICS 96*, pages 160–169, Philadelphia, PA, May 1996.

[6] R. L. Cruz. SCED+: Efficient Management of Quality of Service Guarantees. In *Proceedings of INFOCOM'98*, pages 625–642, San Francisco, CA, 1998.

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

[8] S. Floyd and K. Fall. Router mechanisms to support end-to-end congestion control, February 1997. LBL Technical Report.

[9] S. Floyd and V. Jacobson. Random early detection for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, July 1993.

[10] S. Golestani. A self-clocked fair queueing scheme for broadband applications. In *Proceedings of IEEE INFOCOM'94*, pages 636–646, Toronto, CA, June 1994.

[11] V. Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM'88*, pages 314–329, August 1988.

[12] J. Jaffe. Bottleneck flow control. *IEEE Transactions on Communications*, 7(29):954–962, July 1980.

[13] S. Keshav. A control-theoretic approach to flow control. In *Proceedings of ACM SIGCOMM'91*, pages 3–15, Zurich, Switzerland, September 1991.

[14] D. Lin and R. Morris. Dynamics of random early detection. In *Proceedings of ACM SIGCOMM '97*, pages 127–137, Cannes, France, October 1997.

[15] S. McCanne. *Scalable Compression and Transmission of Internet Multicast Video.* PhD dissertation, University of California Berkeley, December 1996.

[16] J Nagle. On packet switches with infinite storage. *IEEE Trans. On Communications*, 35(4):435–438, April 1987.

[17] Ucb/lbnl/vint network simulator - ns (version 2).

[18] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control - the single node case. In *Proceedings of the INFOCOM'92*, 1992.

[19] S. Shenker. Making greed work in networks: A game theoretical analysis of switch service disciplines. In *Proceedings of ACM SIGCOMM'94*, pages 47–57, London, UK, August 1994.

[20] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *Proceedings of SIGCOMM'95*, pages 231–243, Boston, MA, September 1995.

[21] D. Stilliadis and A. Varma. Efficient fair queueing algorithms for packet-switched networks. *IEEE/ACM Transactions on Networking*, 6(2):175–185, April 1998.

[22] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: Achieving approximately fair banwidth allocations in high speed nteworks, June 1998. Technical Report CMU-CS-98-136, Carnegie Mellon University.

[23] I. Stoica and H. Zhang. LIRA: A model for service differentiation in the internet. In *Proceedings of NOSSDAV'98*, London, UK, July 1998.

[24] Z. Wang. User-share differentiation (USD) scalable bandwidth allocation for differentiated services, May 1998. Internet Draft.