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

Ion Stoica, Scott Shenker, and Hui Zhang
SIGCOMM’98, Vancouver, August 1998
subsequently
IEEE/ACM Transactions on Networking
11(1), 2003, pp. 33-46.

Presented by Bob Kinicki
Outline

- Introduction
- Core-Stateless Fair Queueing (CSFQ)
  - Fluid Model Algorithm
  - Packet Algorithm
  - Flow Arrival Rate
  - Link Fair Share Rate Estimation
- NS Simulations
- Conclusions
Introduction

- This paper brings forward the concept of “fair” allocation.
- The claim is that fair allocation inherently requires routers to maintain state and perform operations on a per flow basis.
- The authors present an architecture and a set of algorithms that is “approximately” fair while using FIFO queueing at internal routers.
An “Island” of Routers
Outline

- Introduction
- **Core-Stateless Fair Queueing (CSFQ)**
  - Fluid Model Algorithm
  - Packet Algorithm
  - Flow Arrival Rate
  - Link Fair Share Rate Estimation
- NS Simulations
- Conclusions
Core-Stateless Fair Queueing

- Ingress edge routers compute per-flow rate estimates and insert these estimates as *labels* into each packet header.
- Only edge routers maintain per flow state.
- Labels are updated at each router based only on aggregate information.
- FIFO queuing with probabilistic dropping of packets on input is employed at the core routers.
Fig. 2. The architecture of the output port of an edge router, and a core router, respectively.
Fluid Model Algorithm

- Assume the bottleneck router has an output link with capacity $C$.
- Assume each flow’s arrival rate, $r_i(t)$, is known precisely.

The main idea is that max-min fair bandwidth allocations are characterized such that all flows that are bottlenecked by a router have the same output rate.

- This rate is called the fair share rate of the link.
- Let $\alpha(t)$ be the fair share rate at time $t$. 
Fluid Model Algorithm

If max-min bandwidth allocations are achieved, each flow 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 is the unique solution to

\[ C = \sum_{i=1}^{n} \min(r_i(t), \alpha(t)), \]
Thus, the probabilistic fluid forwarding algorithm that achieves fair bandwidth allocation is:

Each incoming bit of flow $i$ is dropped with probability

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

These dropping probabilities yield fair share arrival rates at the next hop.
Packet Algorithm

- Moving from a bit-level, buffer-less fluid model to a packet-based, buffer model leaves two challenges:
  - Estimate the flow arrival rates \( r_i(t) \)
  - Estimate the fair share \( \alpha(t) \)
- This is possible because the rate estimator incorporates the packet size.
Flow Arrival Rate

At each edge router, use exponential averaging to estimate the rate of a flow. For flow $i$, let

- $l_i^k$ be the length of the $k^{th}$ packet.
- $t_i^k$ be the arrival time of the $k^{th}$ packet.

Then the estimated rate of flow $i$, $r_i$, is updated every time a new packet is received:

$$r_i^{new} = (1-e^{-T/K}) \cdot L / T + e^{-T/K} \cdot r_i^{old} \quad (3)$$

where

- $T = T_i^k = t_i^k - t_i^{k-1}$
- $L = l_i^k$

and $K$ is a constant.
Link Fair Rate Estimation

If we denote the estimate of the fair share by \( \hat{\alpha}(t) \) and the acceptance rate by \( F(\hat{\alpha}(t)) \), we have

\[
F(\hat{\alpha}(t)) = \sum_{i=1}^{n} \min (r_i(t), \hat{\alpha}(t))
\]

Note – if we know \( r_i(t) \) then \( \hat{\alpha}(t) \) can be determined by finding the unique solution to \( F(x) = C \).

However, this requires per-flow state!
Heuristic Algorithm

- The heuristic algorithm needs three aggregate state variables: \( \hat{\alpha}(t) \), \( \hat{A} \), \( \hat{F} \), where \( \hat{A} \) is the estimated aggregate arrival rate.

- When a packet arrives, the router computes:

\[
\hat{A}_{new} = (1 - e^{-T/K\alpha}) \frac{l}{T} + e^{-T/K\alpha} \hat{A}_{old}
\]  

(5)

- and similarly computes \( \hat{F} \).
CSFQ Algorithm

When a packet arrives, $\hat{A}$ is updated using exponential averaging (equation 5).

If the packet is dropped, $\hat{F}$ remains the same.

If the packet is not dropped, $\hat{F}$ is updated using exponential averaging.

At the end of an epoch (defined by $K_c$), if the link is congested during the whole epoch, update $\hat{\alpha}(t)$:

$$\hat{\alpha}_{\text{new}} = \hat{\alpha}_{\text{old}} \frac{C}{\hat{F}}$$
If the link is not congested, \( \hat{\alpha}_{\text{new}} \) is set to the largest rate of any active flow.

\( \hat{\alpha}_{\text{new}} \) feeds into the calculation of drop probability, \( p \), for the next arriving packet as \( \alpha \) in

\[
p = \max(0, 1 - \alpha / \text{label})
\]
CSFQ Algorithm (cont.)

- Estimation inaccuracies may cause to exceed link capacity.
- Thus, to limit the effect of Drop Tail buffer overflows, every time buffer overflows is decreased by 1% in simulations.
CSFQ Pseudo Code

on receiving packet p
  if (edge router)
    i = classify(p);
    p.label = estimate_rate(r_i, p); // use Eq. (3)
    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
\texttt{estimate} \( \alpha (p, \text{dropped}) \)

// \( \hat{\alpha} \) and \( \alpha.K.c \) are initialized to 0;
// \( \alpha.K.c \) is used to compute the largest packet label seen
// during a window of size \( K.c \)
\( \hat{A} = \text{estimate.rate}(\hat{A}, p); // \text{est. arrival rate (use Eq. (5)}) \)
\( \hat{F} = \text{estimate.rate}(\hat{F}, p); // \text{est. accepted traffic rate} \)
\textbf{if} (\( \text{dropped} == \text{FALSE} \))
\( \hat{\alpha} = \text{estimate.rate}(\hat{F}, p); // \text{est. accepted traffic rate} \)
\textbf{if} (\( \hat{A} \geq C \))
\textbf{if} (\( \text{congested} == \text{FALSE} \))
\( \text{congested} = \text{TRUE}; \)
\( \text{start.time} = \text{crt.time}; \)
\textbf{if} (\( \hat{\alpha} == 0 \))
\( // \hat{\alpha} \text{ can be set to 0 if no packet is received} \)
\( // \text{during a widow of size} \ K.c \)
\( \hat{\alpha} = \max(p.label, \alpha.K.c); \)
\textbf{else}
\textbf{if} (\( \text{crt.time} > \text{start.time} + K.c \))
\( \hat{\alpha} = \hat{\alpha} \times C / \hat{F}; \)
\( \text{start.time} = \text{crt.time}; \)
\textbf{else} // \( \hat{A} \text{ < C} \)
\textbf{if} (\( \text{congested} == \text{TRUE} \))
\( \text{congested} = \text{FALSE}; \)
\( \text{start.time} = \text{crt.time}; \)
\( \alpha.K.c = 0; \)
\textbf{else}
\textbf{if} (\( \text{crt.time} < \text{start.time} + K.c \))
\( \alpha.K.c = \max(\alpha.K.c, p.label); \)
\textbf{else}
\( \hat{\alpha} = \alpha.K.c; \)
\( \text{start.time} = \text{crt.time}; \)
\( \alpha.K.c = 0; \)
\textbf{return} \( \hat{\alpha} \);
Label Rewriting

- At core routers, outgoing rate is merely the minimum between the incoming rate and the fair rate, $\alpha$.
- Hence, the packet label $L$ can be rewritten by

$$L_{\text{new}} = \min (L_{\text{old}}, \alpha)$$
Outline

- Introduction
- Core-Stateless Fair Queueing (CSFQ)
  - Fluid Model Algorithm
  - Packet Algorithm
  - Flow Arrival Rate
  - Link Fair Share Rate Estimation
- **NS Simulations**
- Conclusions
A major effort of the paper is to compare CSFQ to four algorithms via ns-2 simulations.

- FIFO
- RED
- FRED (Flow Random Early Drop)
- DRR (Deficit Round Robin)
FRED (Flow Random Early Drop)

- Maintains per flow state in router.
- FRED preferentially drops a packet of a flow that has either:
  - Had many packets dropped in the past
  - A queue larger than the average queue size
- Main goal: Fairness
- FRED-2 guarantees to each flow a minimum number of buffers.
**DRR (Deficit Round Robin)**

- Represents an efficient implementation of WFQ.
- A sophisticated per-flow queueing algorithm.
- Scheme assumes that when router buffer is full the packet from the longest queue is dropped.
- Can be viewed as “best case” algorithm with respect to fairness.
ns-2 Simulation Details

- Use TCP, UDP, RLM and On-Off traffic sources in separate simulations.
- Bottleneck link: 10 Mbps, 1ms latency, 64KB buffer
- RED, FRED (min, max) thresholds: (16KB, 32KB)
- $K$ and $K_c = 100 \text{ ms}$, $K_\alpha = 200\text{ms}$.
A Single Congested Link

- **First Experiment**: 32 UDP CBR flows
  - Each UDP flow is indexed from 0 to 31 with flow 0 sending at 0.3125 Mbps and each of the $i$ subsequent flows sending $(i+1)$ times its fair share of 0.3125 Mbps.

- **Second Experiment**: 1 UDP CBR flow, 31 TCP flows
  - UDP flow sends at 10 Mbps
  - 31 TCP flows share a single 10 Mbps link.
Figure 5b: 32 UDP Flows

Only CSFQ, DRR and FRED-2 can contain UDP flows!!
Figure 6b: One UDP Flow, 31 TCP Flows

Only CSFQ and DRR can contain Flow 0 – the only UDP flow!
A Single Congested Link

Third Experiment Set: 31 simulations
- Each simulation has a different N,
  \[ N = 2 \ldots 32. \]
- One TCP and N-1 UDP flows with each UDP flow sending at twice fair share rate of 10/N Mbps.
Figure 6b: One TCP Flow, N-1 UDP Flows

- DRR good for less than 22 flows.
- CSFQ better than DRR when a large number of flows.
- CSFQ beats FRED.

Normalized fair share throughput for one TCP source.
Multiple Congested Links

TCP/UDP-0 Source

UDP Sources
1 10 11 20

TCP/UDP-0 Sink

K1-K10

UDP Sinks
1-10
Multiple Congested Links

- First experiment: UDP flow 0 sends at its fair share rate, 0.909 Mbps while the other ten “crossing” UDP flows send at 2 Mbps.
- Second experiment: Replace the UDP flow with one TCP flow and leave the ten crossing UDP flows.
Figure 8a: UDP source

Fraction of UDP-0 traffic forwarded versus the number of congested links.
Figure 8b: TCP Source

Fraction of TCP-0 traffic forwarded versus the number of congested links.
Receiver-driven Layered Multicast

- RLM is an adaptive scheme in which the source sends the information encoded in a number of layers.
- Each layer represents a different multicast group.
- Receivers join and leave multicast groups based on packet drop rates experienced.
Receiver-driven Layered Multicast

- Simulation of three RLM flows and one TCP flow with a 4 Mbps link.
- Fair share for each is 1 Mbps.
- Since router buffer set to 64 KB, $K$, $K_c$, and $K_\alpha$ are set to 250 ms.
- Each RLM layer $I$ sends $2^{i+4}$ Kbps with each receiver subscribing to the first five layers.
Figure 9b: FRED
Figure 9e: RED
Figure 9f: FIFO

![Graph showing throughput over time with different protocols: TCP, ALM1, ALM2, ALM3. The graph displays variations in throughput with time.](image-url)
Figure 9a: DRR
Conference Figure : CSFQ

K = Kc = Kα = 250 ms.
Figure 9c: CSFQ

(c) CSFQ ($K = 100$ ms, $K_\alpha = 200$ ms.)
Figure 9d: CSFQ

(d) CSFQ ($K = 20$ ms, $K_\alpha = 40$ ms.)
On-Off Flow Model

- One approach to modeling interactive, Web traffic :: OFF represents “think time”
- ON and OFF drawn from exponential distribution with means of 100 ms and 1900 ms respectively.
- During ON period source sends at 10 Mbps.
### Table 1: One On-Off Flow, 19 TCP Flows

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Delivered</th>
<th>Dropped</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRR</td>
<td>1080</td>
<td>3819</td>
</tr>
<tr>
<td>CSFQ</td>
<td>1000</td>
<td>3889</td>
</tr>
<tr>
<td>FRED</td>
<td>1064</td>
<td>3825</td>
</tr>
<tr>
<td>RED</td>
<td>2819</td>
<td>2080</td>
</tr>
<tr>
<td>FIFO</td>
<td>3771</td>
<td>1128</td>
</tr>
</tbody>
</table>

4899 packets sent!
Web Traffic

- A second approach to modeling Web traffic that uses Pareto Distribution to model the length of a TCP connection.
- In this simulation 60 TCP flows whose interarrivals are exponentially distributed with mean 0.05 ms and Pareto distribution that yields a mean connection length of 20,1 KB packets.
Table 2: 60 Short TCP Flows, One UDP Flow

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mean Transfer Time for TCP</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRR</td>
<td>25</td>
<td>99</td>
</tr>
<tr>
<td>CSFQ</td>
<td>62</td>
<td>142</td>
</tr>
<tr>
<td>FRED</td>
<td>40</td>
<td>174</td>
</tr>
<tr>
<td>RED</td>
<td>592</td>
<td>1274</td>
</tr>
<tr>
<td>FIFO</td>
<td>840</td>
<td>1695</td>
</tr>
</tbody>
</table>
Table 3: 19 TCP Flows, One UDP Flow with propagation delay of 100 ms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mean Throughput</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRR</td>
<td>6080</td>
<td>64</td>
</tr>
<tr>
<td>CSFQ</td>
<td>5761</td>
<td>220</td>
</tr>
<tr>
<td>FRED</td>
<td>4974</td>
<td>190</td>
</tr>
<tr>
<td>RED</td>
<td>628</td>
<td>80</td>
</tr>
<tr>
<td>FIFO</td>
<td>378</td>
<td>69</td>
</tr>
</tbody>
</table>
Packet Relabeling

Sources

Flow 1

10 Mbps

Flow 2

10 Mbps

Flow 3

10 Mbps

Router

10 Mbps

Link 1

10 Mbps

Router

Link 2

10 Mbps

Sink
Table 4: UDP and TCP with Packet Relabeling

<table>
<thead>
<tr>
<th>Traffic</th>
<th>Flow 1</th>
<th>Flow 2</th>
<th>Flow 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP</td>
<td>3.267</td>
<td>3.262</td>
<td>3.458</td>
</tr>
<tr>
<td>TCP</td>
<td>3.232</td>
<td>3.336</td>
<td>3.358</td>
</tr>
</tbody>
</table>
Unfriendly Flows

- Using TCP congestion control requires cooperation from other flows.
- Three types cooperation violators:
  - Unresponsive flows (e.g., Real Audio)
  - Not TCP-friendly flows (e.g., RLM)
  - Flows that lie to cheat.

This paper deals with unfriendly flows!!
Outline

- Introduction
- Core-Stateless Fair Queueing (CSFQ)
  - Fluid Model Algorithm
  - Packet Algorithm
  - Flow Arrival Rate
  - Link Fair Share Rate Estimation
- NS Simulations
- Conclusions
Conclusions

- This paper presents Core Stateless Fair Queueing and offers many simulations to show how CSFQ provides better fairness than RED or FIFO.
- They mention issue of “large latencies”. This is the robust versus fragile flow issue from FRED paper.
- CSFQ ‘clobbers’ UDP flows!
Significance

- First paper to use hints from the edge of the subnet.
- Deals with UDP. Many AQM algorithms ignore UDP.
- Makes a reasonable attempt to look at a variety of traffic types.
Problems/ Weaknesses

- “Epoch” is related to three constants in a way that can produce different results.
- How does one set K constants for a variety of situations.
- No discussion of algorithm “stability”
Acknowledgments

- Figures extracted from presentation by Nagaraj Shirali and Choong-Soo Lee in Spring 2002 and modified for annotations.