Chapter - 4

RESULTS AND DISCUSSION OF FAIR QUEUING ALGORITHMS
Chapter-4

FAIR QUEUING MECHANISMS

4.1 Introduction

As the Internet has become a public communications infrastructure, providing fairness among individual flows is essential for service providers if they are to offer competitive services at a reasonable cost. A router mechanism is thus needed that can allocate bandwidth fairly among individual flows [120]. As discussed in the previous chapter, RED strategies are implemented for congestion avoidance, which either drops incoming packets or tags them with a dynamically calculated probability to prevent link congestion. This dropping or tagging is used to notify all source and/or destination end terminals where link congestion is a possibility. But it has disadvantages of sensitivity in parameter setting, underutilization of link capacity in case of serious burstiness and need considerable buffer space. Also, fairness cannot be achieved for heterogeneous network. To overcome these disadvantages, a Fair Queuing (FQ) [121] and Core Stateless Fair Queuing (CSFQ) [16] algorithms are proposed, which are discussed in detail.

4.2 Fair Queuing

FQ can allocate bandwidth fairly in a heterogeneous network environment. With FQ, separate queues are maintained for each flow, and packets are output fairly from those queues. During congestion, the output rate via FQ for each flow is restricted to the same rate. As a result, FQ allocates the same bandwidth to each flow. This discipline is called output restriction. Unfortunately, the complexity of maintaining the separate queues make
this mechanism difficult to implement in high-speed backbone routers at a reasonable cost because such routers must handle many flows.

Until now, fair allocations were typically achieved by using per-flow queuing mechanisms - such as Fair Queuing [13,121] and its many variants [122-124] or per-flow dropping mechanisms such as Random Early Drop (RED) [38]. 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., RED), or as complex as manipulation of priority queues (e.g., Fair Queuing). While a number of techniques have been proposed to reduce the complexity of the per packet operations [71,125-126], 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.

To overcome these limitations, an architecture and a set of algorithms of CSFQ was proposed [16], that allocate bandwidth in an approximately fair manner while allowing the routers on high-speed links to use First In First Out (FIFO) queuing and maintain no per-flow state. This chapter presents simulations and analysis to the typical functioning of CSFQ, and results from simulation experiments to illustrate the performance of CSFQ compared to RED and FQ, also discuss, therein, the relative mechanistic complexities of this approach.
4.3 Core-stateless fair queuing (CSFQ)

Router mechanisms designed to achieve fair bandwidth allocations, like Fair Queuing, have many desirable properties for congestion avoidance 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. To overcome these disadvantages, Ion Stoica and et al [16] proposed CSFQ architecture that significantly reduces implementation complexity of FQ, yet still achieves approximately fair bandwidth allocations.

Core-stateless fair queuing (CSFQ) mechanism can allocate bandwidth in approximately the same way the output restriction does, but do not need a separate queue for each flow. The most distinct benefit of CSFQ is that it allows two distinct functions to be implemented in the router, an edge function that identifies flows and maintains the per-flow state (the state of each flow), and a core function that drops packets to regulate the flow bandwidth without their per-flow state. Having these two functions in the router, the CSFQ enables approximately fair bandwidth allocation over the whole network.

![Fig. 4.1a. Fair Queuing and Fig. 4.1b. Core-Stateless Fair Queuing (CSFQ) mechanism](image)
Fig. 4.1a shows a reference network in which all nodes implement FQ, whose functionality is approximated by CSFQ as shown in Fig. 4.1b. In CSFQ an 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.

A central tenet of the Internet architecture is that congestion control is achieved mainly through end-host algorithms. Many researchers observed that such end-to-end congestion avoidance 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 [13].

### 4.3.1 Architecture of Core-Stateless Fair Queuing

The architecture of CSFQ is shown in Fig. 4.2, that approximates the service provided by an island of Fair Queuing router, but has a much lower complexity in the core routers [16]. The architecture has two key aspects. First, to avoid maintaining per flow state at each router, the distributed algorithm is used 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.

Second, to avoid per flow buffering and scheduling, as required by Fair Queuing, FIFO queuing is used 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, proposed 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, first, the idealized bit-by-bit or fluid version of the probabilistic dropping algorithm is explained, and then extended the algorithm to a practical packet-by-packet version.

A buffer less fluid model of a router with output link speed $C$ is considered where the flows are modeled as a continuous stream of bits. Each flow’s assume arrival rate $r_i(t)$ 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. This rate is called 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 $\min(r_i(t), \alpha(t))$. 

Fig. 4.2. The architecture of the output port of an edge router and a core router
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 $a(t)$ is the unique solution to

$$
C = \sum_{i=1}^{n} \min (r_i(t), a(t)) \quad \text{(4.1)}
$$

If $A(t) \leq C$, then no bits are dropped and $a(t) = \max_i r_i(t)$.

If $r_i(t) \leq a(t)$, i.e., flow $i$ sends no more than the server’s fair share rate, all of its traffic will be forwarded. If $r_i(t) > a(t)$, then a fraction $(r_i(t) - a(t)) / r_i(t)$ of its bits will be dropped, so it will have an output rate of exactly $a(t)$. 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 $\alpha(t)$.

$$
\max \left( 0, 1 - \frac{\alpha(t)}{r_i(t)} \right) \quad \text{(4.2)}
$$

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

### 4.3.2 Packet Algorithm

The above equations applicable for a buffer less fluid system in which the arrival rates are known exactly. These equations are extended to the situation in real routers where transmission is packetized, there is substantial buffering, and the arrival rates are not known. Because the rate estimation incorporates the packet size, the dropping probability is independent of the packet size and depends only on the flow arrival rate $r_i(t)$ and fair share rate $a(t)$. 

96
4.3.2.1 Computation of Flow Arrival Rate

In the 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, exponential averaging is used 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:

$$r_i^{new} = \left(1 - e^{-T_i^k / k}\right) \frac{t_i^k}{T_i^k} + e^{-T_i^k / k} r_i \text{old}.$$  (4.3)

Where $T_i^k = t_i^k - t_i^{k-1}$ and $k$ is a constant.

4.3.2.2 Link Fair Rate Estimation

To estimate link fair rate algorithm $a(t)$, the fluid model is considered, where the arrival rates are known exactly, and the probabilistic dropping algorithm is assumed according to Eq. (4.2). Then, the rate with which the algorithm accepts packets is a function of the current estimate of the fair share rate, which is denoted by $\hat{\alpha}(t)$. Letting $F(\hat{\alpha}(t))$ denote this acceptance rate, then

$$F(\hat{\alpha}(t)) = \sum_{i=1}^{n} \min (r_i(t), \hat{\alpha}(t)).$$  (4.4)

$F$ is a continuous, non decreasing, concave, and piecewise-linear function of $\hat{\alpha}$. If the link is congested ($A(t) > C$) then $\hat{\alpha}(t)$ will be the unique solution to $F(x) = C$. If the link is not congested ($A(t) < C$) then $\hat{\alpha}(t)$ to be the largest rate among the flows that traverse the link, i.e., $\hat{\alpha}(t) = \max_{1 \leq i \leq n} (r_i(t))$. From Eq. (4.4) if the arrival rate is $r_i(t)$ then $\alpha(t)$ can be computed directly.
The following Eq. (4.5) with three aggregate state variables are used to estimate total arrival rate: \( \hat{A} \), the estimate for the fair share rate; \( \hat{A} \), the estimated aggregate arrival rate; \( \hat{F} \), the estimated rate of the accepted traffic. The last two variables are updated upon the arrival of each packet. If \( \hat{A} \) is the exponential averaging with a parameter \( e^{-T/K_c} \) where \( T \) is the inter-arrival time between the current and the previous packet then:

\[
\hat{A}_{\text{new}} = (1 - e^{-T/K_c}) \frac{l}{T} + e^{-T/K_c} \hat{A}_{\text{old}} \quad \text{................. (4.5)}
\]

where \( \hat{A}_{\text{old}} \) is the value of \( \hat{A} \) before the updating. The updating rule for \( \hat{A} \) depends on whether the link is congested or not. To filter out the estimation inaccuracies due to exponential smoothing the window of size \( K_c \) is used. A link is assumed to be congested, if \( \hat{A} \geq C \) at all times during an interval of length \( K_c \). Conversely, a link is assumed to be uncongested, if \( \hat{A} \leq C \) at all times during an interval of length \( K_c \). The value \( \hat{A} \) 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 \( \hat{A} \) is updated based on the equation \( F(\hat{A}) = C \). This is approximated by a linear function that intersects the origin and has slope \( \hat{F} / \hat{A}_{\text{old}} \). This yield

\[
\hat{A}_{\text{new}} = \hat{A}_{\text{old}} \frac{C}{\hat{F}} \quad \text{.................. (4.6)}
\]

If the link is not congested, \( \hat{A}_{\text{new}} \) 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 \( \hat{A}_{\text{new}} \) is then used to compute roping probabilities, according to Eq. (4.2). For completeness, the pseudo code of the CSFQ is given in Alg. 4.1 [[16]].
Every receiving packet p

if (edge router)
    i = classify(p);
    p, label = estimate_rate(r, p); /* use Eq. (4.3) */

prob = max(0, 1 - a / p.label);

if (prob > unif_rand(0, 1))
    a = estimate_a(p, 1);
    drop(p);
else
    a = estimate_a(p, 0);
    enqueue(p);

if (prob > 0)
    p.label = a; /* relabel p */

estimate_a(p, dropped)
    estimate_rate(\hat{A}, p); /* est. arrival rate (use Eq. (4.5)) */

if (dropped = = FALSE)
    estimate_rate(\hat{F}, p); /* est. accepted traffic rate */

if (\hat{A} \geq C)
    if (congested = = FALSE)
        congested = TRUE;
        start_time = crt_time;
    else
        if (crt_time > start_time + K_c)
            \hat{a} = \hat{\alpha} x C / \hat{F} ;
\begin{verbatim}
start_time = crt_time;
else /* \hat{A} < C */
    if (congested = = TRUE)
        congested = FALSE;
        start_time = crt_time;
        tmp_a = 0; /* use to compute new \alpha */
    else
        if (crt_time < start_time + Kc)
            tmp_a =\max(tmp_a, p.label);
        else
            \hat{\alpha} = tmp_a;
            start_time = crt_time;
            tmp_a = 0;
return \hat{\alpha};
\end{verbatim}

Alg 4.1. The pseudocode of CSFQ (fair rate estimation).

Two minor amendments are described in this algorithm related to how the buffers are managed. The goal of estimating the fair share \(\hat{\alpha}\) is to match the accepted rate to the link bandwidth. Due to estimation inaccuracies, load fluctuations between \(\hat{\alpha}\)'s updates, and the probabilistic nature of this 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 this algorithm, and may exhibit undesirable properties in the case of adaptive flows such as TCP [38], it is important to limit its effect.
4.3.3 Simulation of CSFQ

The simple topology model used for simulation is shown in Fig. 4.3. The parameters used for the simulation are, each link has a capacity of 10 Mbps, a latency of 5ms, and a buffer of 32 KB. The averaging constants $k$ (used in estimating the flow rate), $k_a$ (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 10ms. In the simulation 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. NS2 is used for simulation and the parameters considered for the simulation are number of flows, Bandwidth, Time with different buffer capacity.

![Topology for analyzing the effects of congested link](image)

Fig. 4.3. Topology for analyzing the effects of congested link

The table 4.1 shows the simulation results for different time interval. Increasing the estimation interval to 10, 20, 30 and 40 ms reduced the tail-dropping ratio and improved fairness. The longer the interval, the lower oscillation, so the less likely the misjudgment of congestion. However, a longer interval meant it took the system longer to detect changes in network conditions. Fig. 4.4 to Fig. 4.7 shows the performance of CSFQ
algorithm. These graphs show that, as the number of flows and transmission time increases, bandwidth and packets transmitted decreases and packet loss rate increases.

Table 4.1. Results of CSFQ for different time interval

<table>
<thead>
<tr>
<th>Estimation Interval</th>
<th>10ms</th>
<th>20ms</th>
<th>30ms</th>
<th>40ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average output rate [Mbps]</td>
<td>0.991</td>
<td>0.987</td>
<td>0.987</td>
<td>0.988</td>
</tr>
<tr>
<td>Standard dev. [Mbps]</td>
<td>0.200</td>
<td>0.123</td>
<td>0.117</td>
<td>0.054</td>
</tr>
<tr>
<td>Tail-dropping ratio</td>
<td>0.326</td>
<td>0.253</td>
<td>0.185</td>
<td>0.114</td>
</tr>
</tbody>
</table>

Fig. 4.4. Simulation results for No. of flows v/s bandwidth received
Fig. 4.5. Simulation results for No. of flows v/s Packet loss

Fig. 4.6. Simulation results for No. of flows v/s Packets received
4.4 Results and Discussions for RED, FQ and CSFQ Algorithms

The CSFQ algorithm is evaluated comparing its performance with RED and FQ. For evaluating the network parameters (network throughput, link utilization, network average delay and loss rate of packets), a network simulator is used as shown in Fig. 4.8. The simulator implemented over two strategies of congestion avoidance, which are FQ and RED. Table 4.2 and 4.3 depicts the performance of RED, FQ and CSFQ for Homogeneous and Heterogeneous data flows respectively. These algorithms are studied in detail and their performances were evaluated using a sophisticated package NS2 [23-24].

Fig. 4.7. Simulation results for Time v/s Bandwidth received
In order to study the performance, the parameters such as number of packets, flows and transmission time are considered for the values of 200000 packets, 6 flows, with a transmission time of 15ms receptivity and were studied under two cases for homogeneous and heterogeneous data flow models as shown in Fig. 4.9 and Fig. 4.13 using NS2. For homogeneous network topology the RED, FQ and CSFQ transmitted 158060, 165366, and 170852 packets in a transmission time of 12.35ms, 10.5ms and 7ms respectively. For heterogeneous network RED, FQ and CSFQ transmitted 98060, 154604 and 160956 packets, in a transmission time of 14.25 ms, 12.5ms and 8.8 ms respectively.

Table 4.2. Performance of RED, FQ and CSFQ for homogeneous data flow

<table>
<thead>
<tr>
<th></th>
<th>Throughput</th>
<th>Delay (ms)</th>
<th>Total Drop</th>
</tr>
</thead>
<tbody>
<tr>
<td>RED</td>
<td>158060</td>
<td>12.35</td>
<td>41940</td>
</tr>
<tr>
<td>FQ</td>
<td>165366</td>
<td>10.5</td>
<td>34634</td>
</tr>
<tr>
<td>CSFQ</td>
<td>170852</td>
<td>7.00</td>
<td>29148</td>
</tr>
</tbody>
</table>
Table 4.3. Performance of RED, FQ and CSFQ for heterogeneous data flow

<table>
<thead>
<tr>
<th></th>
<th>Throughput</th>
<th>Delay (ms)</th>
<th>Total Drop</th>
</tr>
</thead>
<tbody>
<tr>
<td>RED</td>
<td>98060</td>
<td>14.25</td>
<td>101940</td>
</tr>
<tr>
<td>FQ</td>
<td>154604</td>
<td>12.5</td>
<td>45396</td>
</tr>
<tr>
<td>CSFQ</td>
<td>160956</td>
<td>8.8</td>
<td>39044</td>
</tr>
</tbody>
</table>

The two cases from Fig. 4.9 to Fig. 4.16 illustrate the behavior of each strategy for homogeneous and heterogeneous flows. It is clear from these figures that CSFQ points a primacy over other two strategies in the range of high throughput and less packet drop product. The average delay time in CSFQ is less, because in CSFQ only the edge function identifies the flows and estimates their respective transmission rates. It then inserts the appropriate estimated rate information as a label into each packet header for reference by the core function. This estimation process is passive and imposes no restriction on the packet flows. The core function multiplexes packets into a shared link without per-flow state maintenance. During a period of congestion, packets with high rates, indicated by the rate labels in their headers are preferentially discarded with a probability calculated from the rate labels and the fair share rate. CSFQ achieves a reasonable approximation of fair bandwidth allocations in most conditions.
Case 1: Performance of RED, FQ and CSFQ for homogeneous data flow

Fig. 4.9. Homogeneous data flow model

Fig. 4.10. Simulation results for Time v/s Packets transmitted
Fig. 4.11. Simulation results for Packet size v/s Throughput

Fig. 4.12. Simulation results for Packets transmitted v/s latency
Case 2: Performance of RED, FQ and CSFQ for Heterogeneous data flow

Fig. 4.13. Heterogeneous Data flow model

Fig. 4.14. Simulation results for Time v/s Packet transmitted
Chapter-4  
Fair Queuing Mechanisms

Fig. 4.15. Simulation results for Packet size v/s Throughput

Fig. 4.16. Simulation results for Packets transmitted v/s Latency
4.5 Conclusions

CSFQ mechanism overcomes the drawbacks of RED and FQ algorithms. This mechanism can allocate bandwidth in approximately the same way the output restriction does, but do not need a separate queue for each flow. Having edge and core functions in the router enables approximately fair allocation, and distributing these functions throughout a network enables approximately fair bandwidth allocation over the whole network. Additionally, because per-flow maintenance is unnecessary in transit routers, computing power is no longer expended for complex per-flow management in those routers. This type of distributed architecture is promising as a means of fair bandwidth allocation and enhances expandability by simplifying the transit router.

Although CSFQ achieves a much higher degree of fairness in bandwidth allocation compared with other routing mechanisms with similar implementation complexity, there are still several disadvantages that CSFQ fails when congestion-avoidance algorithms that prevent packet loss are used, because it does not accurately approximate the delay characteristics of fair queuing. In fair queuing, flows transmitting at rates less than or equal to their fair share are guaranteed timely delivery of their packets since they do not share the same buffer as packets from other flows. In the core-stateless approximations of fair queuing, this is not the case, since they aggregate packets from all flows into a single buffer and rely on packet discarding to balance the service of each flow. Hence, the existing core-stateless mechanisms are incompatible with congestion-avoidance mechanisms that maintain small router buffers or rely on round-trip time measurements to indicate incipient congestion.
Though CSFQ algorithm overcomes the disadvantages of RED algorithm under heterogeneous flow condition, but both RED and CSFQ fail, if the flow is bursty in nature. Therefore the existing mechanisms require more study and needs to be evaluated in various network environments. Hence, in order to overcome the disadvantages of existing FQ, CSFQ and RED algorithms, a new mechanism has been proposed called Enhanced Core Stateless Fair Queuing (ECSFQ) which is discussed in the next chapter.