Improvements to Core Stateless Fair Queueing*

Cristel Pelsser[1] and Stefaan De Cnodder[2]

[1] University of Namur, Belgium,
cpe@infonet.fundp.ac.be
[2] Alcatel, Belgium,
stefaan.de_cnodder@alcatel.be

Abstract. Core Stateless Fair Queueing (CSFQ) is a scalable mechanism to provide per-flow fairness in high-speed networks in that it does not need to maintain per-flow state in the core routers. This is possible because the state for each flow is encoded as special labels inside each packet. In this paper, we propose and evaluate by simulations two improvements to CSFQ. First, we show that CSFQ does not provide a fair service when some links are not congested. Our first improvement solves this issue. Second, we propose an algorithm to allow CSFQ to provide a service with a minimum guaranteed bandwidth and evaluate its performance with TCP traffic.

1 Introduction

The Internet was initially designed to provide a best-effort service. During the last ten years, two architectures have been proposed to allow the Internet to support other types of services. The Integrated Services architecture (IntServ) [2] aims at providing end-to-end guarantees to individual flows. The Differentiated Services architecture (DiffServ) [1] aims at providing several grades of service to different aggregated flows. Although these two architectures differ in most aspects, the routers supporting these advanced services will have to rely on classifiers, buffer acceptance algorithms, markers, traffic conditioners and schedulers to provide the required service differentiation or guarantees. The performance and the complexity of these mechanisms is a key issue to be considered when designing high speed routers.

From an application point of view, the IntServ architecture provides the best service since the guarantees are associated with each individual flow. However, this often forces the routers to perform a complex classification and to maintain some state for each individual flow. This severely limits the scalability of the IntServ architecture and its capability of being used in high speed networks. The DiffServ architecture on the other hand started from the assumption that it must be implementable with today's equipment. For this, DiffServ traded the per-flow guarantees in favor of per traffic aggregate guarantees. This improves the scalability of DiffServ. In DiffServ, a network is composed of two types

* This work was partially supported by the European Commission within the IST ATRIUM project.

G. Carle and M. Zitterbart (Eds.): PfHSN 2002, LNCS 2334, pp. 164–179, 2002.

of routers : edge routers and core routers. The edge routers typically serve a small number of customers with a few links. Their role is to classify the packets received from the customers and mark each packet with the class of service that they should receive. By putting the complexity at the edge routers this way, DiffServ is much more scalable and performant than IntServ but it lowers the quality of the service from the application point of view.

Inside high-speed networks, the performance of the routers is a key issue. Congestion should not occur at the routers. In other words, the routers should not be bottlenecks. They should be able to use all the bandwidth available on the links to which they are connected. As a consequence, the mechanisms that aim to provide per-flow guarantees should be able to work at a high pace, to be usable in high-speed networks. Therefore, Dynamic Packet State (DPS) [9, 11] is an interesting solution to provide per-flow service differentiation in a performant and scalable manner. DPS removes the requirement to maintain per-flow state by encoding the state for each flow as labels inside the IP packets. By this mean, it also removes the need for packet classification at each node. With DPS, as with DiffServ, there are edge and core routers. The edge routers classify the packets received from the customers and mark each packet with the per-flow state required by downstream routers to provide the required fairness or guarantees. Edge and core routers then process the packets according to these markings[1]. Since all the per-flow state information is included inside each IP packet, the core routers do not need to maintain per-flow state and to perform classification ; their complexity is reduced. To respect transparency, when a packet reaches the edge of the network, the packet state introduced by the ingress edge router is removed. Several mechanisms relying on the DPS principle or on the Differentiated Services Code Point (DSCP) have been proposed and analyzed in the literature [3, 4, 6, 7, 9, 10, 11, 12] for the provision of a large variety of services in high-speed networks. In this paper, we focus exclusively on Core Stateless Fair Queueing (CSFQ) due to space limitations. A more detailed overview of these mechanisms may be found in [8].

This paper is organized as follows. First, in Sect. 2, we provide a detailed description of CSFQ as it was proposed in [10]. Then, in Sect. 3, we show that CSFQ does not provide fairness when some links are non congested. We propose and evaluate a solution to this problem that occurs on many links inside high-speed networks. In Sect. 4, we show how to extend the CSFQ algorithm to provide a minimum guaranteed bandwidth to each flow and evaluate the performance of this extension with TCP traffic.

2 CSFQ Algorithm

By fairness we mean, in this paper, that all flows sharing a link, and having packets dropped at the router upstream of the link, should get the same amount of bandwidth. Such flows are said to be bottlenecked on the link. All other flows on the link get a smaller amount of bandwidth. The amount of bandwidth that

[1] For information concerning the storage of the label in the packets refer to [10, 11].

is allocated, in a fair situation, to a flow bottlenecked on a link is called the fair share of the link. In addition to the fair provision of bandwidth, this paper deals with the support of minimum bandwidth guarantees, allowing certain flows to use a portion of bandwidth in all case, in addition to their fair allocation.

CSFQ [10] aims at distributing bandwidth fairly between the flows sharing the network. It also allows to associate a weight to the different flows and tries to distribute the bandwidth in proportion to those weights. CSFQ is based on DPS. Since it does not need to perform classification, it is suitable for high-speed networks. The complexity is put into the edge routers, that are connected only to a few links, leaving the core routers with a lower complexity.

On packet p arrival, the edge routers perform classification. They determine the flow to which the packet belongs, $flow_i$. The state of this flow is then used to compute the arrival rate of the flow, AR_i. The arrival rate of $flow_i$ is determined for any i by

$$AR_{i,\text{new}} = (1 - e^{-\frac{T}{K}})\frac{L}{T} + e^{-\frac{T}{K}} AR_{i,\text{old}} \ , \tag{1}$$

where $AR_{i,\text{old}}$ is the previous estimation of the arrival rate of $flow_i$ stored in the flow state, K is a constant, L is the length of the current packet p and T is the time elapsed between the arrival of the previous packet of $flow_i$ and the current packet arrival. This is the formula used by the `estimate_rate` function in Fig. 1. The new arrival rate estimation of $flow_i$ ($AR_{i,\text{new}}$) is used to label the current packet p belonging to $flow_i$.

When a packet p arrives, the edge and core routers compute the drop probability of the packet using

$$P_{\text{drop}} = \max(0, 1 - \frac{FS}{\text{p.label}}) \ , \tag{2}$$

where the fair share, FS, has been estimated at the previous packet arrival and p.label is the label of the current packet. It can be deduced that, if p.label $> FS$

```
On receiving packet p
    if (edge router)
        i = classify(p);
        p.label = estimate_rate(AR_i,p);
    P_drop = max(0,1-FS/p.label);
    if (P_drop > unifrand(0,1))
        FS = estimate_FS(p,1);
        drop(p);
    else
             if (P_drop > 0)
             p.label = FS;  /* relabel p*/
         FS = estimate_FS(p,0);
         enqueue(p);
```

Fig. 1. CSFQ pseudo-code

then $P_{\text{drop}} > 0$. And, when p.label $\leq FS$, we have $\frac{FS}{\text{p.label}} \geq 1$. It follows that $P_{\text{drop}} = 0$.

Then, if the drop probability P_{drop} of the packet is greater than a random number, the packet is in excess of the fair share. A new estimation of the fair share is done and the packet is dropped. Otherwise, P_{drop} is lower than the random number and the packet belongs to the fair allocation. If the drop probability is above zero, the packet is relabeled with the fair share, which should be an approximation of the rate of $flow_i$ on the output link. Then, independently of the value of P_{drop}, the fair share is estimated. And, after the estimation, the packet is stored in the queue provided that it is not full. Figure 1 shows the pseudo-code of the CSFQ algorithm.

The estimation of the fair share is shown in Fig. 2. First of all, the arrival rate of the aggregate traffic, AR, at the router and the rate at which the packets are forwarded to the FIFO queue, FR, are estimated using the exponential averaging (1). The link is considered congested if the arrival rate is larger than the link rate (BW) and congested otherwise.

When the link becomes congested, the variable `congested` is set and, the beginning of the time window, `start_time`, is set to the current time. If the link remains congested for K_c seconds, the fair share is updated according to

```
estimate_FS(p,dropped)
    estimate_rate(AR,p);  /* estimate arrival rate */
    if (dropped == FALSE)
        estimate_rate(FR,p);
    if (AR≥BW)
        if (congested == FALSE)
            congested = TRUE;
            start_time = current_time;
        else
            if (current_time > start_time + Kc);
                FS = FS * BW/FR;
                start_time = current_time;
    else  /* AR < BW */
        if (congested == TRUE)
            congested = FALSE;
            start_time = current_time;
            temp_FS = 0;  /* Used to compute new FS */
        else
            if (current_time < start_time + Kc)
                temp_FS = max(temp_FS,p.label);
            else
                FS = temp_FS;
                start_time = current_time;
                temp_FS = 0;
    return FS;
```

Fig. 2. Fair share estimation pseudo-code

$$FS_{\text{new}} = FS_{\text{old}} \frac{BW}{FR} \quad . \tag{3}$$

An explanation of (3) is given in [10]. Intuitively, it can be seen that when the forwarding rate estimation is above the link rate, the fair share estimation will decrease leading to less packets being transmitted when the number of flows stays constant or decreases. Otherwise, the fair share estimation increases.

The link is considered uncongested when the arrival rate is below the link rate. If the link was congested at the previous estimation but is now uncongested, the variable `congested` is set to false and `temp_FS` is set to zero. The variable `temp_FS` is used to compute the maximum of the labels carried by the packets arriving during an interval of length K_c and starting at `start_time`.

It can be noticed that, when the link isn't a bottleneck, there is no need to drop packets from any flow. In this case, the fair share is set to the arrival rate of the flow with the highest bandwidth. By setting the fair share at the maximum flow rate, no packets will be dropped because, according to (2), the dropping probability will be zero for each flow.

The fair share FS is estimated each time the link is congested for at least K_c seconds. It is also computed when the link is uncongested during the last K_c seconds. We can already see that if the link passes from uncongested to congested and vice-versa too often (i.e. in less than K_c seconds) the fair share will not be reestimated.

3 Improvement to CSFQ

3.1 Uncongested Network Problem

It has to be noticed that in CSFQ (Fig. 2), as exposed in [10], when there is no congestion, the fair share is set to the maximum of the packets' labels passing through the node during a fixed interval, called a window size, of length K_c. But, during such intervals, sometimes not all flows have packets arriving at a node. Because TCP is bursty, the rate of a TCP flow may be high even if no packets are sent during certain periods. Consequently, the fair share might be smaller than the label of certain packets. As a result, the drop probability of certain packets could be higher than zero. Some packets may be dropped even if the link on which they have to be transmitted is not congested.

Another problem with the estimation of the fair share as the maximum of the labels, in uncongested mode, is that the flows have difficulties to increase their sending rate. For example, if the maximum arrival rate among the flows at a link increases, and the new arrival rate of the flow is not yet incorporated in the fair share estimation, some packets of the flow are dropped. Again, we observe dropping of packets during an uncongested period. But, during uncongested periods, TCP always tries to increase its sending rate. If packets of the TCP flows are dropped, their sending rate will decrease because dropped packets are interpreted as a congestion indication by TCP. Therefore, this estimation of the fair share done by CSFQ for uncongested links can be seen as unfriendly toward

TCP flows. Additionally, forcing the flows to decrease their sending rate leads to an under-utilization of the resources.

We can also imagine the case where a network node does not have any packet to send on a particular output link. Then, the estimation of the arrival rate should be around 0 bps and the link is considered as uncongested. Because no packets arrive, the fair share is estimated to 0 bps. And, when packets will arrive for this link, they will all be dropped. If the link stays uncongested during a window size, the fair share estimation will increase to the maximum of the labels that passed during the last window size period but if the link becomes congested before this update, the fair share estimation will remain zero according to (3).

3.2 Proposed Solution

To avoid the dropping of packets when there is no congestion (Sect. 3.1), we propose to estimate the fair share on the entire period that the network is uncongested, instead of during K_c seconds. And, the fair share is still updated every K_c seconds. This means that the last estimation of the fair share is taken into account in the new estimation. Instead of computing the maximum starting from 0 at the beginning of the K_c windows, the maximum is computed starting from the previous fair share estimation. In other words, every K_c seconds an estimation is made over the last K_c seconds, and the final estimation is set to the maximum of the current estimation and the new maximum, calculated over the last period. Figure 3 shows the pseudo code of the fair share estimation in our Modified version of Core Stateless Fair Queueing (MCSFQ). This requires only changes in the non congested case of the fair share estimation.

The solution proposed is interesting because, when the link becomes uncongested, it can be considered that the right fair share has been found. The new fair share should not be lower than the fair share used when there was congestion because the arrival rate decreased, so, less packets have to be dropped. It is logical that the fair share should be the maximum of the fair share at the last congestion update and the packet's labels that passed. When the link stays uncongested, the temporary variable is not reset because all packets can be forwarded. When resetting it, like in the proposal made in [10], if the labels of the packets that pass during a period of a window size length are strictly lower than the maximum of the packets' labels passed during the previous period, the fair share decreases. But, when there is no congestion, either each flow has already reached its fair share or the flows are not using all network resources. There is no need to limit the use of these resources.

Now, during uncongested periods, the fair share is estimated by the maximum of the packets' labels, received since the end of the congestion and of the last fair share update before the uncongested period. It follows that the fair share estimation is higher with MCSFQ than with CSFQ, allowing to forward packets at a higher rate. Additionally, TCP will more easily be able to increase its sending rate. When TCP increases its sending rate, the first packets with a high label may be dropped but their labels are taken into account in the next fair share

```
estimate_FS(p,dropped)
    estimate_rate(AR,p);  /* estimate arrival rate */
    if (dropped == FALSE)
        estimate_rate(FR,p);
    if (AR≥BW)
        if (congested == FALSE)
            congested = TRUE;
            start_time = current_time;
        else
            if (current_time > start_time + K_c);
                FS = FS * BW/FR;
                start_time = current_time;
    else  /* AR < BW */
        if (congested == TRUE)
            congested = FALSE;
            start_time = current_time;
            temp_FS = FS;  /* Used to compute new FS */
        else
            if (current_time < start_time + K_c)
                temp_FS = max(temp_FS,p.label);
            else
                FS = temp_FS;
                start_time = current_time;
    return FS;
```

Fig. 3. MCSFQ fair share estimation : pseudo-code

updates. It follows that TCP is able to send at a higher rate than before once its window size has increased again, when the network stays uncongested.

It can be noticed that, when the link becomes congested for a window size period, the fair share will be updated according to (3). If the fair share is too high when the link becomes congested, the forwarding rate measured during the first window period will be higher than the link rate and therefore, the next estimation of the fair share will be smaller.

3.3 Simulations

In each of the scenarios used for the simulations[2], there is one flow per source. These flows may be an aggregate of TCP or UDP flows. The data packets are sent from source to destination. The data packets all go in the same direction, from the left to the right of the networks. And, the acknowledgments follow the reverse path. Two way traffic, i.e. traffic with a mix of acknowledgments and data packets, is not considered in this paper. Because the acknowledgments are much smaller and less frequent than the data packets, there is never congestion in the reverse path. In the simulations, CSFQ (MCSFQ) is not used on the path of the acknowledgments. Only the data path is influenced by CSFQ (MCSFQ).

[2] OPNET is the tool used to perform the simulations.

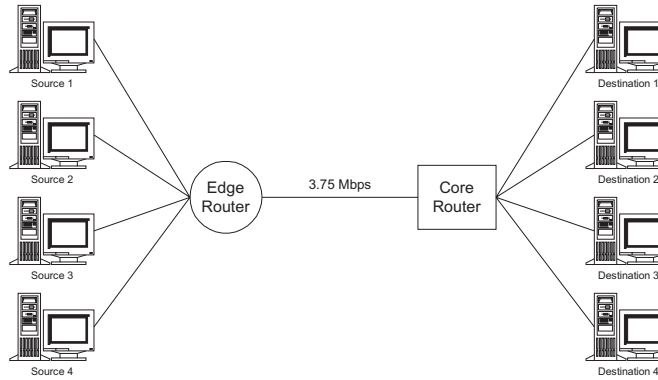

Fig. 4. Single bottleneck scenario

The bandwidths that are indicated on the Fig. 4 and Fig. 8 are the bandwidths for the data path. For the other direction, the link rates are set large enough such that there is never congestion. The simulations are done with a threshold[3] of half the queue size, as recommended in [10]. And, the fair share is initialized at 1 bps.[4]

3.4 Single Bottleneck Scenario

Although the single bottleneck scenario is simple, it already gives a good indication of the behavior of CSFQ (MCSFQ). It indicates whether or not CSFQ (MCSFQ) is able to distribute the bandwidth in a fair way in simple networks.

In the single bottleneck scenario (Fig. 4), there are four sources, an ingress edge router, a core router and four destinations. All packets generated by one source belong to the same flow, so there are four flows in total. All four flows cross the two routers of the network. They first pass through the edge router where they are multiplexed on a single link on which CSFQ (MCSFQ) allocates the bandwidth. Then, the flows cross the core router. Finally they are distributed on the links leading to their destinations. The edge router is the only possible bottleneck. The core router performs CSFQ (MCSFQ) also, but the traffic in excess to a fair bandwidth allocation should already have been dropped by the first router.

In this scenario, the links from the users to the routers have a rate of 100 Mbps with a fixed propagation delay of 2.5 ms. The link joining the two routers is of 3.75 Mbps with a fixed propagation delay of 10 ms in each direction.

[3] The threshold is a value used in case the network is uncongested. The network is supposed to stay that way until the queue occupancy gets above the threshold.

[4] For a discussion concerning the importance of the value chosen for the fair share at initialization refer to [8].

3.5 Uncongested Network Problem

In this subsection, the results, obtained from simulations where the single bottleneck network is uncongested or only congested at times, are presented. The results are obtained from statistics taken at the edge router. All sources send UDP packets except the first source that establishes one TCP connection. The UDP flows are not responsible of a possible congestion. Their rate of 500 Kbps is below one fourth of the link bandwidth. The total rate of the UDP flows will be 1.5 Mbps. That means that 2.25 Mbps are left to the TCP flow. In this situation, the fair share estimation should be around 2.25 Mbps to allow the TCP flow to make use of the bandwidth that is unused by the UDP flows. The window sizes, K_c, are first set to 0.6 seconds.

The graphs on the left of Fig. 5 and Fig. 6 show the throughput of each flow at the edge router with CSFQ, MCSFQ and also with tail drop as the only bandwidth allocation mechanism. The graphs on the right of these figures illustrate the dropping rate of the flows with the same bandwidth distribution mechanisms.

On Fig. 5 (left), it can be seen that the throughput of the UDP flows is around 500 Kbps independently of the bandwidth allocation mechanism. On the other hand, the throughput of the TCP flow varies depending on the bandwidth distribution mechanism used. It is at its peak with tail drop, it is slightly lower

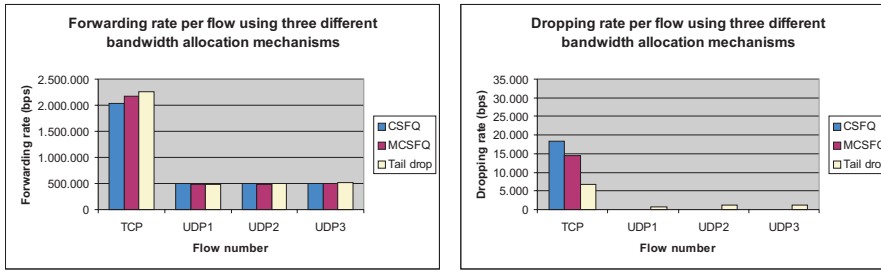

Fig. 5. MCSFQ versus CSFQ : $K_c = 0.6$s

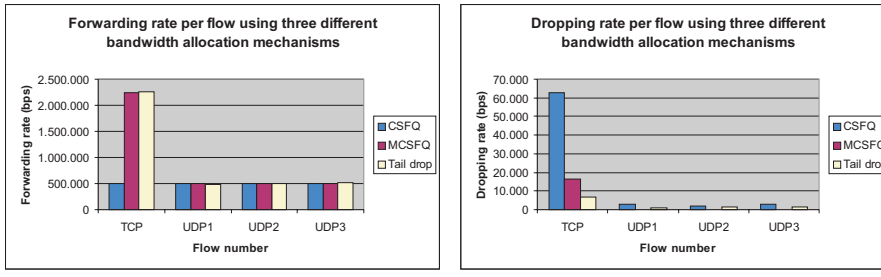

Fig. 6. MCSFQ versus CSFQ : $K_c = 0.1$s

with MCSFQ and it is the smallest with CSFQ. This means that the bandwidth resources are better used with tail drop than with MCSFQ. And, even less resources are used with CSFQ because the estimated fair share is higher with MCSFQ than with CSFQ in uncongested periods. When tail drop is used, UDP packets may be dropped as well as TCP packets (Fig. 5, right), which is not a fair behavior in this situation. With MCSFQ and CSFQ, the fair share estimation is at least equal to the rate of the UDP flows. It follows that no UDP packets are dropped (Fig. 5, right).

When the same simulations are performed with the window sizes set to 0.1 seconds [10] instead of 0.6 seconds, the TCP flow is only able to use the same portion of bandwidth as the one used by each UDP flow (Fig. 6, left), if CSFQ is used. Here, the window size is rather small and because TCP is bursty, its packets carry labels that are above the fair share estimation. Therefore, some TCP packets are dropped even if there is no congestion on the link. On Fig. 6 (right), it can be seen that even UDP packets are dropped that way. This sensitivity of CSFQ to the value of K_c is annoying in practice. Our Modified CSFQ does not suffer from this problem.

The TCP flow is not able to send above the throughput of the UDP flows with CSFQ when the window size is small. As exposed in Sect. 3.1, the TCP flow has difficulties in increasing its sending rate. With a higher window size, the fair share estimation is higher and there are more chances for the TCP flow to be able to increase its sending rate. The MCSFQ bandwidth allocation mechanism is not subject to such an important impact of the window size on the throughput of the flows in uncongested networks (Sect. 3.2).

On Fig. 6 (right), it can be seen that packets from the UDP flows are dropped with tail drop and CSFQ. We cannot say that this is a fair behavior because these flows do not create the congestion. On the contrary, no UDP packet is dropped by MCSFQ. The throughput of TCP is almost the same with MCSFQ and tail drop. Here we can say that MCSFQ is the fairer bandwidth allocation mechanism. We notice that MCSFQ is less sensitive to the window size than CSFQ. If the window size is small the bandwidth is not shared fairly with CSFQ as seen in Fig. 6. And, a high window size does also not provide fairness as shown by [8]. It is interesting to use a small value for K_c to have an estimation of the fair share that is more reactive to the changing traffic patterns. With a small window size, the fair share is estimated more often.

4 Support of Minimum Guarantees

CSFQ was conceived to distribute the bandwidth fairly among the flows sharing the network. It was thought to support flows that have different weights. The bandwidth that each flow receives is then a proportion of its weight. But, CSFQ was not meant, in [10], to be able to provide minimum guarantees to some flows. In this section, we show how to support minimum guaranteed flows in CSFQ and MCSFQ.

A small change in the labeling performed by the edge routers is enough to support minimum throughput guarantees. Some modifications may be done to the buffer acceptance module also [8]. But, these alterations are not mandatory to obtain satisfying results. In addition to the enhancement to CSFQ, an admission control mechanism[5] should be implemented for the support of minimum guaranteed flows to ensure that there are enough resources to provide the guarantees associated to the flows accepted in the network.

Our proposed modification to CSFQ is as follows. First of all, while labeling packets, edge routers have to be able to determine which packets are part of the guarantee associated to the packet's flow and which packets will be treated as best-effort. Bandwidth that is not part of any reservation has to be shared fairly between the flows sending packets in excess of their traffic contract. Therefore, CSFQ will only apply to excess packets. Edge routers will for that purpose mark all guaranteed packets with a label of zero. This indicates that the packet doesn't use any portion of the bandwidth that has to be allocated fairly. But, in excess packets will be marked with a label equal to the estimated excess rate of the flow, as suggested in [6]. A similar principle is also used in [12] for the provision of minimum throughput guarantees by Stateless Prioritized Fair Queueing (SPFQ).

When a packet arrives at a core router, it is checked, to determine the packet's dropping probability (2), if its label is higher than the fair share of the output link on which it has to be transmitted. When the packet is guaranteed, the fair share will never be smaller than the label. The drop probability of the packet is set equal to zero and the packet is not discarded, except if the queue is full and tail drop occurs. But, the drop probabilities of excess packets might be greater than zero.

The changes required to provide flows with minimum throughput guarantees are shown in Fig. 7. In the pseudo code given in Fig. 7, $guar\_rate_i$ is the amount of bandwidth that is guaranteed to $flow_i$ and ER_i is its excess rate. The marking is performed based on a probabilistic determination of packets in and out of the guarantees. A deterministic method may also be considered by using token buckets. The excess rate of the flow is estimated according to (1). Moreover, to obtain a quicker convergence of the fair share estimation, we suggest to estimate the amount of aggregate guarantee. This estimation requires modifications to the fair share estimation function. We refer to the appendix of [8] for informations concerning the implementation of this estimation as well as the deterministic marking.

4.1 Generic Fairness Configuration (GFC) Scenario

The GFC scenario is used to show that flows can benefit from their guarantee and their fair share altogether in networks where the fair share or the RTT varies from one flow to another. This scenario is generic in that the fair share of the links are different and the flows do not have the same path in the network,

[5] Some admission control mechanisms that do not require per flow state are proposed in [11].

```
On receiving packet p
    if (edge router)
        i = classify(p);
        AR_i = estimate_rate(AR_i,p);
        /* Determine if the packet is in or out of the guarantees */
        P_out = max(0,1 - guar_rate_i / AR_i);
        /* Mark the packet */
        if (P_out <= unifrand(0,1))
            p.label = 0;
        else
            p.label = estimate_rate(ER_i,p);
    /* Edge and core routers */
    if (p.label = 0)  /* Packet in the guarantee */
        P_drop = 0;
    else  /* Packet out of the guarantee */
        P_drop = max(0,1-FS/p.label);
    if (P_drop > unifrand(0,1))
        FS = estimate_FS(p,1);
        drop(p);
    else
             if (P_drop > 0)
             p.label = FS;  /* relabel p*/
         FS = estimate_FS(p,0);
         enqueue(p);
```

Fig. 7. Support of minimum guarantees : pseudo-code

leading to different RTTs. Dropping of packets happens in different nodes of the network. Other scenarios and their corresponding simulation results are analysed in [8].

In this scenario, there are 10 sources and 10 destinations (Fig. 8). It follows that 10 flows are considered. These are TCP flows composed of 15 TCP connections each. The flows initiated by the B and X sources are congested on the first link. The flows starting from the C sources are bottlenecked on the second link. And, the flows from the A sources as well as from the Y sources are bottlenecked on the third link. Sources X and Y act as background sources to create congestion at the edge router and at the second core router respectively.

In the Generic Fairness Configuration (GFC) scenario, the first router is an edge router. The second and third routers are classical core routers. They label the packets coming from the sources directly connected to the router. Then, they route the packets to the output links. Finally, the packets are dropped or accepted according to CSFQ, if the router is not connected directly to the packets' destination. The last router is an egress edge router. It does not perform CSFQ. The arriving packets are routed to an output link where they are transmitted to their destination.

The propagation delays on the different links are fixed. The delays of propagation on the links between two routers are 10 ms. Flows A-short, B-short,

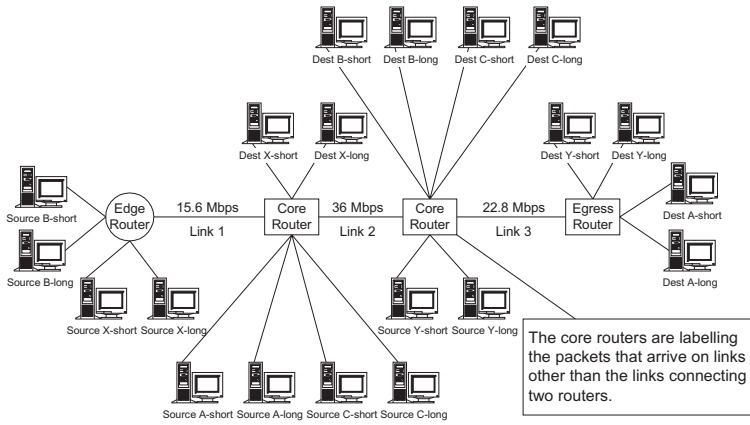

Fig. 8. Generic Fairness Configuration scenario

C-short, X-short and Y-short have a slightly smaller RTT than flows A-long, B-long, C-long, X-long and Y-long. The propagation delay on the links between a source, or destination, of a flow with a small RTT, and a router is 2.5 ms. For flows A-long, B-long, C-long, X-long and Y-long the total delay occurred on the access links has been increased by 32 ms by comparison to the flows with a smaller RTT. This increase is distributed over the two access links crossed by the flows. The window sizes are initialized to 0.6 seconds.

The amount of reserved bandwidth on the second link is 50% of the link rate, in these simulations. Each flow benefits from the same throughput guarantee of 3 Mbps. The rate of link 2 is set to 36 Mbps. 50% of these 36 Mbps are reserved for the TCP flows. There are still 18 Mbps left to share between the A, B and C flows. The excess of the B flows should get 10% of these remaining 18 Mbps. That means that the B flows should be able to use 1.8 Mbps of link 2 in excess of their guarantee. Because there are four flows sharing link 1, the total excess rate on this link should be $1.8 \text{ Mbps} \times 2 = 3.6$ Mbps. The rate of link 1 will be set to $4 \times 3 \text{ Mbps} + 3.6 \text{ Mbps} = 15.6$ Mbps. The excess of the B flows should therefore, in average obtain, 1.8 Mbps on link 1. The B flows are not bottlenecked on the second link. It follows that no packet from these flows should be dropped on link 2. A similar reasoning leads to set the rate of the third link to 22.8 Mbps. The deduction is based on the fact that the A flows should have 30% of the bandwidth of link 2 in addition of their guarantees.

4.2 Support of Minimum Guarantees

The goodput of the flows is considered in Fig. 9. It is compared to the goodput that the flows should have in an ideal situation where the guarantees are provided and the excess bandwidth is shared fairly among the flows. The goodput of a flow is the rate of its data transmission. Therefore, the goodput of a flow is smaller than its throughput. In the goodput, packet headers and retransmissions

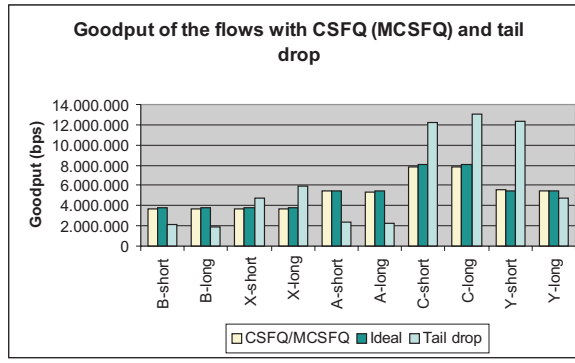

Fig. 9. CSFQ and MCSFQ versus tail drop

of packets are not taken into account. The ideal goodput of Fig. 9 is computed by assuming that no packets are retransmitted which is usually not true. Therefore, it is normal that the goodputs of the flows in the simulations performed with CSFQ (MCSFQ) are slightly below their ideal goodput.

The flows get the same goodput with CSFQ and MCSFQ because the network is always congested and therefore the fair share estimation is the same in both mechanisms. The threshold has also no impact on the bandwidth distribution for the same reason.

When tail drop is used alone, no guarantees are provided to the flows. From Fig. 9, it can be deduced that the A and B flows do not get as much bandwidth as their guarantees. Additionally, the bandwidth is not distributed fairly among the different flows. The RTT has a big impact on the goodput of the flows.

Figure 9 indicates that the flows get their minimum goodput guarantees with CSFQ (MCSFQ). Their excess goodput is above zero. Additionally, the unreserved bandwidth is distributed approximately fairly among the different flows. The flows bottlenecked on the same links approximately get the same amount of bandwidth. The flows with a longer RTT don't get much less bandwidth than the other flows congested on the same link. And, the flows that occur drops at different points in the network, A flows for example, are not too penalized compared to flows that cross less nodes in the network.

5 Conclusion and Further Work

In this paper, we have proposed and evaluated two improvements to CSFQ. First, we have analysed the estimation of the fair share in uncongested networks. We have shown that the estimation used by CSFQ could cause unfairness and underutilization of the network resources. Consequently, we proposed an amelioration to this estimation and validated this improvement by simulations. An important advantage of our Modified CSFQ is that it is much less sensitive to the setting of the window size (K_c) than the original CSFQ.

As a second contribution, we proposed a method to support minimum guaranteed flows with CSFQ and MCSFQ, by enhancing the packet labeling. We have evaluated the performance of these modifications in a complex GFC scenario with TCP traffic. We showed that with these modifications, the TCP flows were able to benefit from their minimum guaranteed bandwidth. Additionally, we noticed that, with CSFQ and MCSFQ, the unreserved bandwidth can be shared fairly among the flows during congestion even when the flows have different RTTs and fair shares.

Several other improvements to CSFQ are possible. First, work should be done to further improve the estimation of the fair share. During uncongested periods, the fair share should be decreased periodically, or only when the congestion is noticed, to avoid tail drops when the network starts to be congested, because tail drops lead to unfair bandwidth distribution. During congestion, another estimation method could also be used. The applicability of the estimation used in the Fair Allocation Derivative Estimation (FADE) [5] should be studied. Second, when tail drops occur with CSFQ, the fair share is decreased by a small percentage [10]. It could be interesting to avoid tail drops instead of reacting to tail drops. The mechanism proposed in [4] could be used for this purpose [8].

Acknowledgments

We would like to thank Olivier Bonaventure for his constructive comments as well as Steve Uhlig, Pierre Reinbold and Bruno Quoitin for their reviews.

References

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. Internet RFC 2475, December 1998.

[2] R. Braden, D. Clark, and S. Shenker. Integrated services in the Internet architecture : an overview. Internet RFC 1633, July 1994.

[3] Z. Cao, Z. Wang, and E. Zegura. Rainbow fair queueing: Fair bandwidth sharing without per-flow state. In *Proceedings INFOCOM '00*, March 2000.

[4] Stefaan De Cnodder, Kenny Pauwels, and Omar Elloumi. A rate based RED mechanism. In *Proc. of the 10th International Workshop on Network and Operating System Support for Digital Audio and Video*, NOSSDAV 2000, Chapel Hill, NC, 26–28 June 2000.

[5] Na Li, Marissa Borrego, and San-qi Li. Achieving per-flow fair rate allocation within diffserv. In *Proceedings of the fifth IEEE Symposium on Computers and Communications*, ISCC 2000, Antibes, France, 3-6 July 2000.

[6] M. Nabeshima, T. Shimizu, and I. Yamasaki. Fair queueing with in/out bit in core stateless networks. In *Proc. of the 8th International Workshop on Quality of Service*, IWQoS 2000, Pittsburgh, PA, 5–7 June 2000.

[7] Kenny Pauwels, Stefaan De Cnodder, and Omar Elloumi. A multi-color marking scheme to achieve fair bandwidth allocation. In *Proc. of the 1st International Workshop on Quality of future Internet Services*, QofIS 2000, Berlin, Germany, 25–26 September 2000.

[8] Cristel Pelsser. Support of fairness and guarantees without per-flow state in routers. Master's thesis, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, June 2001. Available at www.info.fundp.ac.be/\char126cpe/memoire.ps.gz.

[9] I. Stoica, H. Zhang, S. Shenker, R. Yavatkar, D. Stephens, A. Malis, Y. Bernet, Z. Wang, F. Baker, J. Wroclawski, C. Song, and R. Wilder. Per hop behaviors based on dynamic packet states. Internet Engineering Task Force, Work in Progress, draft-stoica-diffserv-dps-00.txt, February 1999.

[10] Ion Stoica, Scott Shenker, and Hui Zhang. Core-stateless fair queueing : Achieving approximately fair bandwidth allocations in high speed networks. In *Proceedings of ACM SIGCOMM'98*, Vancouver, BC, October 1998.

[11] Ion Stoica and Hui Zhang. Providing guaranteed services without per flow management. In *Proceedings of ACM SIGCOMM'99*, Cambridge, MA, August 1999.

[12] N. Venkitaraman, J. Mysore, R. Srikant, and R. Barnes. Stateless prioritized fair queueing. Internet Engineering Task Force, Work in Progress, draft-venkitaraman-spfq-00.txt, July 2000.