# Core-Stateless Proportional Fair Queuing for AF Traffic

Gang Cheng, Kai Xu, Ye Tian, and Nirwan Ansari

Advanced Networking Laboratory, Department of Electrical and Computer Engineering,
New Jersey Institute of Technology, Newark, NJ 07102, USA

***Abstract*— Proportional fair queuing is to ensure that a flow passing through the network only consumes a fair share of the network resource that is proportional to its committed rate or other service level agreement (SLA). It is of great importance in differentiated services (DiffServ) networks as well as other price incentive network services. In this paper, we propose a simple core-stateless proportional fair queuing algorithm (CSPFQ) for the assured forward (AF) traffic in DiffServ networks. We first develop our algorithm based on a fluid model analysis and then extend it to a realizable packet level algorithm. We prove analytically and instantiate through simulations that our algorithm can achieve proportional fair bandwidth allocation among competing flows without requiring routers to estimate flows' fair share rates. Our simulation results also demonstrate that our algorithm outperforms the weighted core-stateless fair queuing (WCSFQ) in terms of proportional fairness.**

## I. Introduction

Since data traffic in the current Internet is inherently bursty, and end-to-end congestion control is not always deployed in all transport or application protocols, some form of active queue management (AQM) at network routers is thus necessary to protect well-behaving flows from irresponsive ones by fairly allocating bandwidth among the flows. Proportional fair queuing is to ensure that a flow passing through the network only consumes its fair share of the network resource that is proportional to its committed rate or other metrics defined in its service level agreement (SLA). Proportionally fair bandwidth allocation is an important mechanism for implementing price incentive QoS services.

Active queue management schemes for fair bandwidth allocation can be classified into two distinct categories, the core-stateful approach and core-stateless approach. The core-stateful approach, such as weighted fair queuing (WFQ) [1] and flow random early drop (FRED) [2], requires the core routers to maintain per-flow state information, and is thus lacking the scalability simply because of the large number of flows at the network core. In contrast, in the core-stateless queue management approach, core routers do not need to maintain per-flow state information. Instead, per-flow information is encoded in the packet headers by the edge routers. Core-stateless queuing schemes such as core-stateless fair queuing (CSFQ) [3] and rainbow fair queuing (RFQ) [4] can also achieve comparable fair bandwidth allocation as that of using core-stateful approaches.

The key idea behind CSFQ [3] is to keep per-flow state at less aggregated edge routers and carry that information in packets to the core. Specifically, edge routers estimate each flow's arrival rate and label the packets of a flow with its corresponding arrival rate. Core routers estimate the flow fair share and probabilistically drop packets with labels (arrival rate) larger than their estimated fair shares of the bandwidth. Extensive simulations show that CSFQ offers considerable improvement of fairness over RED [5] and FIFO. Cao et al. proposed RFQ [4], in which each flow is divided into a set of layers based on its rate, and packets in a flow are marked at edge routers with a layer label (color). Core routers maintain a color threshold and drops layers whose colors exceed the threshold. RFQ outperforms CSFQ in terms of goodput while achieving the comparable fairness of CSFQ. Weighted CSFQ, or WCSFQ, is a natural extension to the original packet labeling scheme by associating each flow with a weight [3]. WCSFQ can be used to realize proportional fair bandwidth allocation.

Differentiated services [6], [7], [8], or DiffServ, is one of the recent proposals to provisioning QoS for IP networks, particularly, the Internet. The goal of the evolving IETF DiffServ framework is to provide a means of offering a spectrum of services in the Internet without the need for per-flow state and signaling in every router. DiffServ differentiates its service priorities by classifying flows into the Expedited Forwarding (EF), the Assured Forwarding (AF), and the best effort forwarding classes through service contracts. Usually, multiple random early drop (MRED) is recommended for the management of queues that handle AF traffic. Although MRED stabilizes the router queue, and eliminates network synchronization by randomization, MRED cannot provide fair bandwidth allocation among flows. In order to guarantee the committed rates for AF flows, packet marking schemes such as in [9] are proposed.

In this paper, we propose a new simple core-stateless proportional fair queue management scheme taht is scalable, effective, and robust in allocating proportionally fair bandwidth among competing AF flows.

## II. Core-Stateless Proportional Fair Queuing for AF Traffic

In this section, we propose a core-stateless proportional fair queuing (CSPFQ) algorithm for handling the AF traffic in DiffServ networks. We start with a fluid model analysis, and then extend it to a packet based algorithm. We define a network as an over-loaded network if there exists a link on which the sum of the committed rates of flows sharing the link is larger than the link capacity. A well-loaded network is referred to as a network in which the sum of the committed rates of flows on any link equals to the link capacity. For any other network that is neither over-loaded nor well-loaded, we call it an under-loaded network. A DiffServ network does not likely result in an over-loaded network since service providers have the control of contracting committed rates to their clients. Hence, we propose our AF queue management under the assumption that networks are either well-loaded or under-loaded.

### A. Fluid Model Algorithm

In a typical DiffServ network, a single FIFO at a core router is shared by many flows belonging to the same AF traffic class, each of which is assigned its own committed rate. It is thus critical that the core router distributes its output bandwidth fairly among the flows and at the same time guarantees their committed rates. In the context of assured forwarding in DiffServ networks, we define the fairness as follows. A router allocates to a flow a share of

[1]This work has been supported in part by the New Jersey Commission on Science and Technology via NJWINS.

 

the excess bandwidth that is proportional to the flow's assigned committed rate. In particular, assume $n$ AF flows are contending for an output link of bandwidth $C$ at the core router. The committed rate, arrival rate, and allocated bandwidth of flow $i$are $r_i$, $\alpha_i$ and $\varphi_i$, respectively. Apparently, in a well-loaded or under-loaded network, we have $\sum_{i=1}^{n} r_i \leq C$. We claim that the bandwidth allocation is fair if the following conditions are met.

$$\varphi_i = \begin{cases} \min(\alpha_i, \lambda r_i), & \text{if } \sum_{i=1}^{n} \alpha_i > C \\ \alpha_i, & \text{if } \sum_{i=1}^{n} \alpha_i \leq C \end{cases}, \quad (1)$$

where $i = 1, 2, ..., n$ and $\lambda$ satisfies

$$\sum_{i=1}^{n} \min(\alpha_i, \lambda r_i) = C. \quad (2)$$

Now, we introduce Theorem 1, which enables us to develop the core-stateless proportional fair queuing (CSPFQ) algorithm using a fluid model.

*Theorem 1:* Assume $\sum_{i=1}^{n} \alpha_i > C$ (i.e., a congested link), and each bit of flow is encoded with a label $\max(\pi\alpha_i - r_i, 0)/r_i$, where $\pi$ is a random variable uniformly distributed in [0,1]. Each flow achieves its fair share rate if the bits with labels larger than $\lambda - 1$ are dropped.

*Proof:* Since we assume that the link is congested and the network is either well-loaded or under-loaded, by (2) we have $\lambda \geq 1$. Therefore,

$$\Pr\{\max(\pi\alpha_i - r_i, 0)/r_i > \lambda - 1\} = \Pr\{\pi\alpha_i > \lambda r_i\}. \quad (3)$$

First, let us consider the case where $\alpha_i < \lambda r_i$. We have

$$\Pr\{\pi\alpha_i > \lambda r_i\} = 0; \quad (4)$$

no bits of flow are dropped, and, according to (1), its allocated bandwidth $\varphi_i$ is equal to its arrival rate and is still $\alpha_i$, which is also its fair share of the bandwidth. Second, we consider the other case where $\alpha_i \geq \lambda r_i$. We have

$$\Pr\{\pi\alpha_i > \lambda r_i\} = \Pr\{\pi > \lambda r_i/\alpha_i\} = 1 - \lambda r_i/\alpha_i. \quad (5)$$

Thus, by dropping the bits with labels larger than $\lambda - 1$, the arrival rate of flow $i$ is decreased to

$$\alpha_i[1 - \Pr\{\max(\pi\alpha_i - r_i, 0)/r_i > \lambda - 1\}] = \varphi_i. \quad (6)$$

Therefore, the arrival rate of flow $i$ is its fair share rate $\varphi_i = \min(\alpha_i, \lambda r_i)$ if each bit is encoded with the label $\max(\pi\alpha_i - r_i, 0)/r_i$, and the bits with labels larger than $\lambda - 1$ are dropped. ■

It seems that computing the fair share rate for each flow is necessary to implement Theorem 1. However, as illustrated in Fig. 1, when congestion occurs, we can fairly allocate bandwidth among flows by iteratively dropping the bits with the largest labels until the rate of aggregated traffic is decreased to the bandwidth of the output link, by which each flow can also achieve its fair share. Note that labels only depend on the committed rates and the arrival rates. We can compute them at the edge routers. Hence, our fluid model CSPFQ queue management algorithm for the AF traffic consists of the following two procedures.

1) Each bit of a flow is assigned a label of $\max(\pi\alpha - r, 0)/r$ by edge routers, where $r$ is the committed rate of the corresponding flow, $\alpha$ is its arrival rate estimated by the router, and $\pi$ is a random variable uniformly distributed in [0,1].
2) At the core router, when congestion occurs, bits with the largest labels are iteratively dropped until the arrival rate of the aggregated traffic is decreased to the router's maximum output rate.

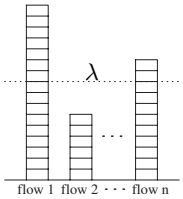


Fig. 1. Bits of each flow are sorted by their labels and piled. The bits with larger labels are above the bits with smaller labels.

### B. Packet Level Realization of CSPFQ

We now extend the fluid model algorithm to a packet level realization. In our algorithm, edge routers label the packets of a flow based on the flow's arrival rate and committed rate. In practice, flow arrival rates are not known to the edge routers, but can be estimated by the edge routers. We adopt the exponentially weighted moving average as presented in [3] to estimate the flow arrival rates. Specifically, upon the arrival of packet $p_k$ of size $l_k$ at time $t_k$, the estimated arrival rate of the flow that this packet belongs to is updated as

$$\alpha_k = \left(1 - e^{-T_k/\omega}\right) l_k/T + e^{-T_k/\omega}\alpha_{k-1}, \quad (7)$$

where $T_k = t_k - t_{k-1}$, $\omega$ is a constant, and $\alpha_{k-1}$ is the estimated arrival rate upon the arrival of the previous packet $p_{k-1}$. Similar to [3], we can directly extend our fluid model queue management algorithm to a packet level implementation. The two procedures of our algorithm become

1) Upon the arrival of a packet, the estimated arrival rate of the corresponding flow is updated at the edge router, and this packet is encoded with a label of $l = \max(\pi\alpha - r, 0)/r$, where $r$ is the committed rate of this flow, $\alpha$ is its estimated arrival rate, and $\pi$ is a random variable uniformly distributed in [0,1].
2) Each core router maintains a threshold, and drops the packets whose labels exceed the threshold. The threshold is a function of the current queue size and the sum of the committed rates of all flows sharing the queue. Specifically, we drop a packet with label $l > 0$ if

$$\sigma s l + q_{current} > \rho q_{\max}, \text{and} \quad (8)$$

$$q_{current} > \beta q_{\max}, \quad (9)$$

where $\sigma$, $\beta$, and $\rho$ are three constants, $s$ is the sum of the committed rates of flows sharing the queue, $q_{current}$ is the current queue size, and $q_{\max}$ is the maximum queue size. Note that, by (9), packets are dropped only when the current queue size is larger than a threshold ($\beta q_{\max}$); otherwise, packets in the queue may be easily drained before the new packets arrive, and the network throughput is not unnecessarily decreased. In our simulations, we set $\beta = 0.2$. $\sigma$ is set to the average round trip time of flows because it takes about a round trip time for a responsive flow such as a TCP flow to throttle its sending rate upon a packet loss. $\rho$ is a constant related to specific networks. In some networks, traffic distortion is high; even a flow containing packets all with small labels can still be very bursty. Hence, in order to guarantee the committed rates of flows, we always leave a portion of the queue to accommodate packets with label 0 so that it is never dropped except when the queue is full. Moreover, $\rho$ is also the dominant parameter that determines the average queue size.

## III. SIMULATIONS

We have implemented our proposed CSPFQ algorithm in the ns-2 network simulator. In this section, we evaluate, through a series of simulations, the performance of our algorithm in terms of proportionally and fairly allocating bandwidth among competing AF flows. To provide a comparison basis, we also run the same set of simulations using the weighted CSFQ, or WCSFQ as introduced in [3]. CSFQ achieves max-min fairness by labeling packets with the corresponding flow's arrival rate and probabilistically dropping the packets whose label is greater than the estimated fair share rate. The dropping probability in CSFQ is $\max(0, 1 - \alpha/r_i)$, where $\alpha$ is the core router's estimation of fair share rate for all contending flows, and $r_i$ is the edge router's estimation of the arrival rate of flow $i$, which is also the label stamped in the packet header (see [3] for details). CSFQ relies on two estimates, the flow arrival rate estimation at the edge routers and the fair share rate estimation at the core routers. WCSFQ is a natural extension of CSFQ by associating each flow with a weight $w_i$. As introduced in [3], WCSFQ achieves fair bandwidth allocation such that all contending flows would have the same value for $r_i/w_i$, and hence it can be used to implement differential services.

As compared to WCSFQ, our CSPFQ is designed with the goal of proportional fairness and is simpler than WCSFQ in that the core router does not need to estimate the flows' fair share rates. Throughout our simulations, all weights in WCSFQ are set to the flows' committed rates. In all the simulations reported here, packet size is set to 1000 bytes; and buffer spaces of various links are set to be the bandwidth-delay-product of the end-to-end paths. Simulations A through D use the network topology as shown in Fig. 2.

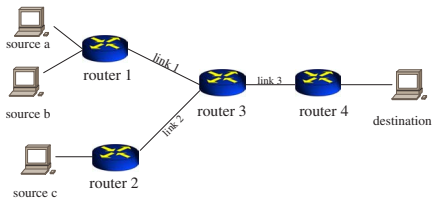


Fig. 2. Three sources sending data to a single destination. Routers 1 and 2 are edge routers and routers 3 and 4 are core routers. Propagation delays for link 1, 2, and 3 are 5ms.

### A. TCP Flows in a Well-loaded Network

In this simulation, all sources are sending TCP bulk data transfer traffic to the destination. The committed rates for flows sent from source a, b, and c are 4Mbps, 1Mbps, and 5Mbps, respectively. The link capacities of the network are provisioned to be 5Mbps, 10Mbps, and 10Mbps, for links 1, 2, and 3, respectively, hence a well-loaded network. The simulation time is 100 seconds. Fig. 3 and Fig. 4 show the bandwidth allocation results by using our CSPFQ and that by using WCSFQ. It is observed that CSPFQ does achieve fair bandwidth allocation among competing AF traffic flows such that the allocated bandwidth for a flow is proportional to the flow's committed rate. WCSFQ with weights set to individual flows' committed rates can also achieve certain proportional fairness but with larger allocation errors.

### B. TCP Flows in an Under-loaded Network

This simulation has the same parameter settings as the previous simulation, except for the link capacities of the network. In this case, the network is provisioned to be under-loaded. Specifically, capacities of links 1, 2 and 3 are 6Mbps, 10Mbps and 12Mbps.

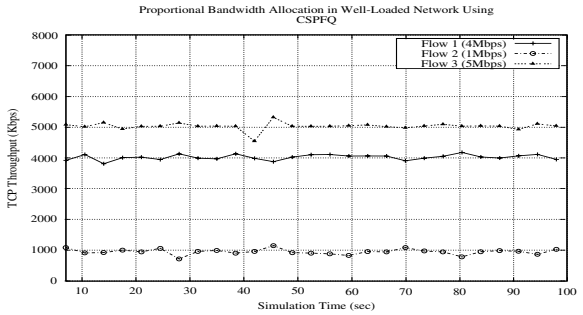


Fig. 3. Bandwidth allocation for TCP flows in a well-loaded network using CSPFQ.

The purpose of this simulation is to evaluate whether our proposed algorithm allocates the excessive bandwidth proportionally to each flow. We also run the simulation twice, once using our CSPFQ, and the other using WCSFQ. The results plotted in Fig. 5 show that CSPFQ does allocate the excessive link bandwidth to each flow according to the proportionality of their committed rates. Throughputs of TCP flows exhibit very little variation throughout the simulation time. As a comparison, Fig. 6 plots the results obtained by running the simulation using WCSFQ. Although WCSFQ can also maintain the rough proportionality among the allocated bandwidths, all TCP flows show quite a large amount of variation in their throughputs.

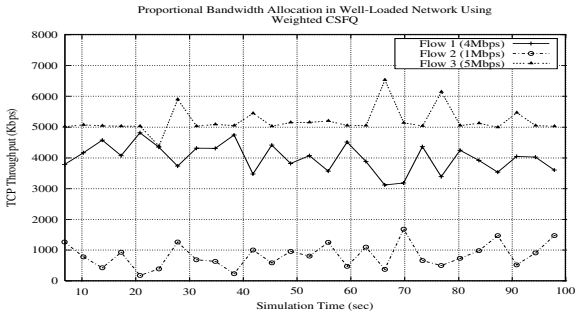


Fig. 4. Bandwidth allocation for TCP flows in a well-loaded network using WCSFQ.

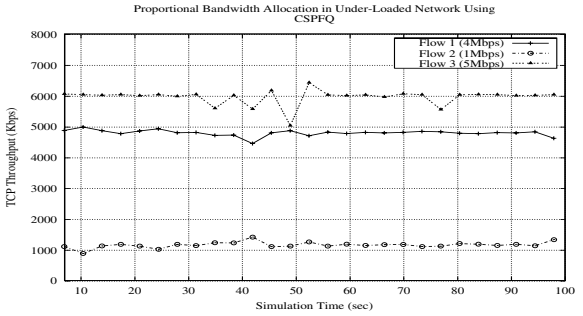


Fig. 5. Bandwidth allocation for TCP flows in an under-loaded network using CSPFQ.

### C. UDP Flows in a Well-loaded Network

In this simulation, we investigate the CSPFQ performance in the environment where flows are not responsive to packet drops. The network provision and the flows' committed rates are the same as those in simulation 1, except that instead of using TCP flows, here we are experimenting with UDP flows. All sources are sending constant

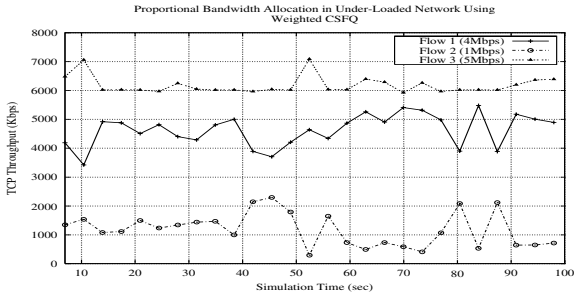


Fig. 6. Bandwidth allocation for TCP flows in an under-loaded network using WCSFQ.

bit rate (CBR) flow of data to the destination using the UDP protocol. The sending rates of all CBR sources are set to be 10Mbps. As shown in Fig. 7 and Fig. 8, both CSPFQ and WCSFQ achieve very satisfactory performance in terms of bandwidth allocation that is fair among the competing flows and proportional to each flow's committed rate. Compared to the results for TCP flows, throughputs of UDP flows appear to be smoother than those of TCP flows. This is because the CBR sources are not responsive to packet drops. Unlike the TCP sources, they do not actively adjust their sending rates.

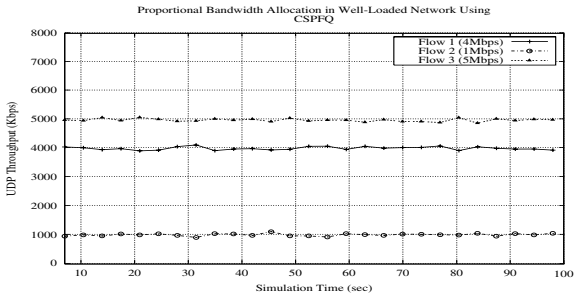


Fig. 7. Bandwidth allocation for CBR flows in a well-loaded network using CSPFQ. Bottleneck is 10Mbps. CBRs are sent at 10Mbps.

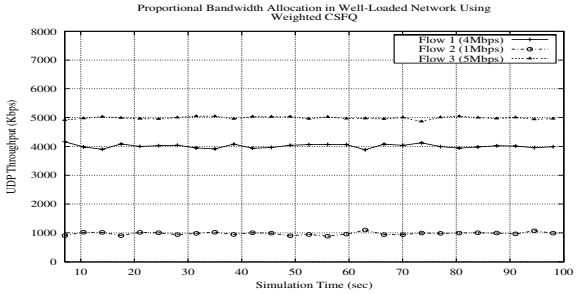


Fig. 8. Bandwidth allocation for CBR flows in a well-loaded network using WCSFQ. Bottleneck is 10Mbps. CBRs are sent at 10Mbps.

### *D. UDP Flows in an Under-loaded Network*

This simulation has the same settings as simulation C, except that the capacity of link 3 is set to 15Mbps to make the network under-loaded. Fig. 9 and Fig. 10 show the results of CSPFQ and WCSFQ, respectively. Similar to the results from simulation C, both algorithms perform fairly well. All bandwidth allocations are proportional to the flows' committed rates.

### *E. Mix of TCP and UDP Flows in a Well-loaded Network*

In the next two simulations, we evaluate the performance of our proposed CSPFQ algorithm in the environment where there

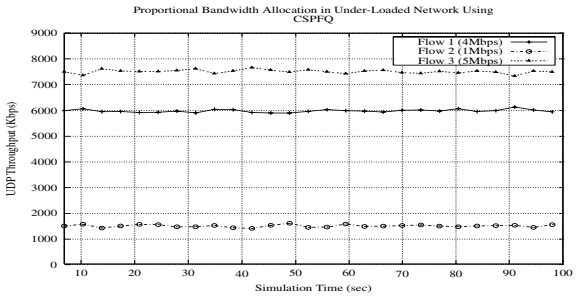


Fig. 9. Bandwidth allocation for CBR flows in an under-loaded network using CSPFQ. Bottleneck is 15Mbps. CBRs are sent at 10Mbps.

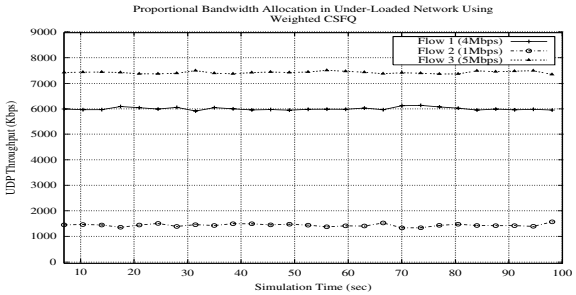


Fig. 10. Bandwidth allocation for CBR flows in an under-loaded network using WCSFQ. Bottleneck is 15Mbps. CBRs are sent at 10Mbps.

are a large number of heterogeneous flows contending for a single bottleneck. Specifically, as depicted in Fig. 11, the network contains four routers, of which routers 1 and 2 are edge routers and routers 3 and 4 are core routers. There are 40 sources sending AF traffic to a single destination. Flows are numbered from 1 to 40. Flow $i$ is assigned a committed rate of $100i$ Kbps, where $i = 1, 2, ...40$, i.e., flow 1 is committed to 100Kbps rate, flow 2 to 200Kbps rate, and so on. Flows 1$\sim$10 and 21$\sim$30 are CBR flows, of which the sources are sending at twice of the corresponding committed rates, i.e., source of flow 1 sends at 200Kbps rate, flow 2 at 400Kbps rate, and so on. Flows 11$\sim$20 and 31$\sim$40 are TCP flows. We group these 40 flows in such a way that flows 1$\sim$10 are aggregated with flows 31$\sim$40 into router 1; flows 11$\sim$20 are aggregated with flows 21$\sim$30 into router 2. Therefore, the aggregated committed rates from link 1 and link 2 are both 41Mbps. The capacity of the bottleneck link, link 3, is provisioned to be 82Mbps, hence a well-loaded network.

Again, we run the simulation twice, using CSPFQ and using WCSFQ, and compare the results. Fig. 12 plots the average throughput of each flow. In the figure, the solid diagonal line represents the ideal proportionally fair bandwidth allocation, and the two dashed diagonal lines represent $\pm 25\%$ of the ideal fairness line.

We observe from the figure that our CSPFQ algorithm can achieve fairly satisfactory result in such an environment. All flows' average throughputs are roughly proportional to their committed rates and allocation errors are within $\pm 25\%$ region. It is clear that irresponsive CBR flows can steal the bandwidth from drop sensitive TCP flows when two types of flows coexist. This is partly attributed to the feedback delay in TCP's congestion control. Overall, CSPFQ can protect the well-behaving TCP flows from the persistent CBR flows and still maintain proportionality in its bandwidth allocation.

On the other hand, WCSFQ, while exhibiting comparable performance in the environments of homogeneous flows, does not result in a satisfactory bandwidth allocation. Under WCSFQ, too much of the bottleneck bandwidth is allocated to the CBR flows and

the throughputs of all TCP flows are suppressed far below their committed rates.

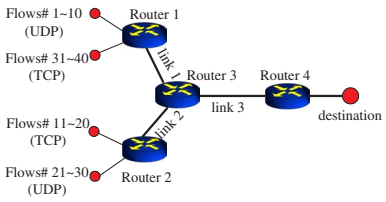


Fig. 11. 40 heterogeneous AF flows compete for a single bottleneck. Each flow has committed rate of $100i$ Kbps, where $i$ is the flow id.

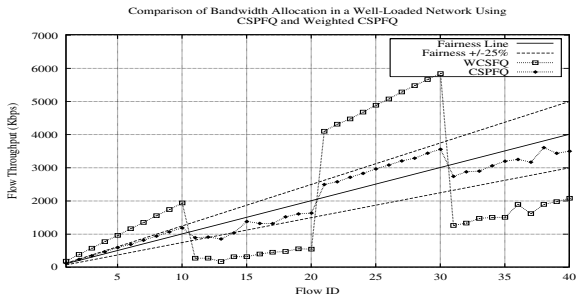


Fig. 12. Comparison of bandwidth allocation of 40 heterogeneous AF flows using CSPFQ and WCSFQ in a well-loaded network. CBR flows send at twice of their committed rates.

### *F. Mix of TCP and UDP Flows in an Under-loaded Network*

Here, we repeat simulation E in an under-loaded network. Instead of 82Mbps bottleneck, we set the capacity of the bottleneck link to 100Mbps, which is 18Mbps larger than the total of the committed rates. Also, different from simulation E, all CBR flows are sending at three times of their committed rates. Fig. 13 shows that our CSPFQ can still maintain the proportionality in fair bandwidth allocations among flows. For WCSFQ, the TCP flows have gained some bandwidth as compared to Fig. 12, but the bandwidth distribution among the flows is certainly not proportional to the flows' committed rates. The proportionality, or the lack of it, is more evident in the following

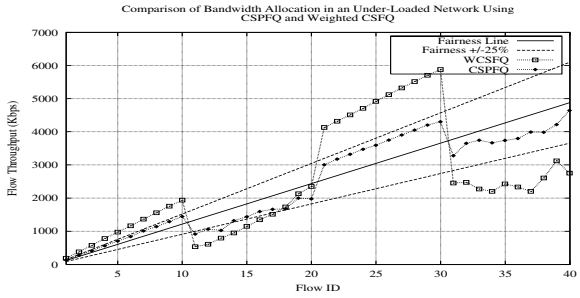


Fig. 13. Comparison of bandwidth allocation of 40 heterogeneous AF flows using CSPFQ and WCSFQ in an under-loaded network. CBR flows send at three times of their committed rates.

figures. Fig. 14 is a combination of plots from Fig. 12 and Fig. 13 for the CSPFQ case. It shows that under CSPFQ, as the network changes from well-loaded to under-loaded, every flow's throughput increases proportionally to the flow's committed rate. Fig. 15 is a combination of plots from Fig. 12 and Fig. 13 representing the results using WCSFQ in well-loaded and under-loaded networks, respectively. The results of using WCSFQ in an environment of heterogeneous flows do not show a clear proportionality.

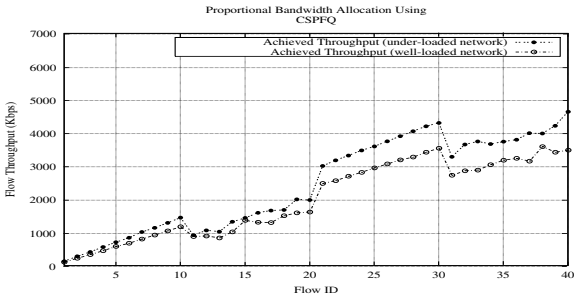


Fig. 14. Comparison of bandwidth allocations using CSPFQ in both well-loaded and under-loaded networks.

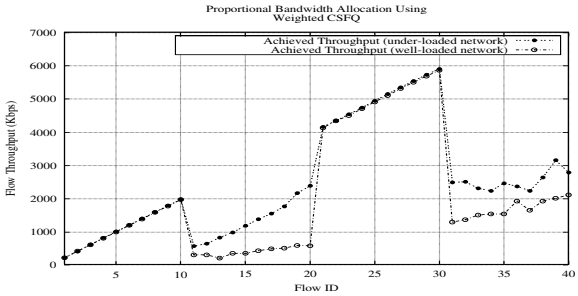


Fig. 15. Comparison of bandwidth allocations using WCSFQ in both well-loaded and under-loaded networks.

## IV. Conclusions

Aiming to provision proportional fair bandwidth allocation for AF flows in DiffServ networks, we have proposed a new simple core-stateless proportional fair queuing scheme, CSPFQ. We have proven analytically that our scheme is fair using the fluid model. We have also shown, through a series of simulations, with comparisons, that the packetized implementation of CSPFQ is effective in achieving proportional fairness and is robust in handling different combinations of flow types. As compared to WCSFQ, our algorithm eliminates the need for core routers to estimate flows' fair shares.

## References

[1] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, 1993.

[2] D. Lin and R. Morris, "Dynamics of Random Early Detection," in *ACM SIGCOMM 1997*, 1997, pp. 127–137.

[3] I. Stoica, S. Shenker, and H. Zhang, "Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High-Speed Networks," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 33–46, 2003.

[4] Z. Cao, Z. Wang, and E. Zegura, "Rainbow Fair Queuing: Fair Bandwidth Sharing Without Per-flow State," in *IEEE INFOCOM 2000*, 2000, pp. 922–931.

[5] S. Floyd and V. Jacoboson, "Random Early Detection For Congestion Avoidance," *IEEE/ACM Transactions on Networking*, pp. 397–413, 1993.

[6] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentialed Services," IETF RFC 2475, 1998.

[7] K. Nichols, S. Blake, K. Baker, and D. Black, "Definitions of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," IETF RFC 2474, 1998.

[8] D. Grossman, "New Terminology and Clarifications for DiffServ," IETF RFC 3260, 2002.

[9] J. Heinanen and R. Guerin, "A Two Rate Tree Color Marker," IETF RFC 2698, 1999.