

Improving Individual Flow Performance with Multiple Queue Fair Queuing

Manfred Georg, Christoph Jechlitschek, and Sergey Gorinsky

Applied Research Laboratory

Department of Computer Science and Engineering

Washington University in St. Louis

St. Louis, Missouri 63130-4899, USA

Email: {mgeorg, chrisj, gorinsky}@arl.wustl.edu

Abstract—Fair Queuing (FQ) algorithms provide isolation between packet flows, allowing max-min fair sharing of a link even when flows misbehave. However, fairness comes at the expense of per-flow state. To keep the memory requirement independent of the flow count, the router can isolate aggregates of flows, rather than individual flows. We investigate the feasibility of protecting individual flows under such aggregate isolation in the context of Multiple Queue Fair Queuing (MQFQ), where the router maintains a fixed number of queues and allows each flow to access multiple queues. MQFQ places packets into the shortest queue associated with their flow. The extra queues protect the flow against congestion caused by a misbehaving flow in a shared queue. However, multiple per-flow queues also enable the misbehaving flow to increase its unfairly acquired fraction of the link capacity. We discuss avoidance of packet reordering within a flow and compare MQFQ with prior schemes for aggregate scheduling.

I. INTRODUCTION

Fair Queuing (FQ) algorithms [1]–[7] dedicate a separate queue to each flow and schedule packets for transmission over a congested link so that every flow receives an average service rate that approximates its max-min fair share of the link capacity [8]. In comparison to the traditional First-In First-Out (FIFO) scheduling of packets, FQ provides a significant degree of isolation between flows and therefore exhibits superior resilience against misbehaving flows. For example, if a User Datagram Protocol (UDP) flow transmits at an unfairly high rate, the excessive transmission does not disrupt well-behaving flows; instead, FQ penalizes the aggressive flow through accumulation and eventual discard of its packets at the router. Fair queuing also improves the fairness properties of end-to-end congestion control. While the sending rate of Transmission Control Protocol (TCP) in a network of FIFO routers is inversely proportional to round-trip time [9], [10] and hence not max-min fair, using FQ with sufficient buffers at bottleneck links enables TCP to transmit at max-min fair rates. Unfortunately, the fairness benefits of FQ come at the expense of maintaining per-flow state at the router.

In this paper, we consider link scheduling with constant memory requirements regardless of the number of flows. Such scheduling disciplines include Stochastic Fair Queuing (SFQ) [11], Stochastic Fair Blue (SFB) [12], and Random Early Detection with Preferential Dropping (RED-PD) [13]. When the number of flows becomes large, these schemes generally treat multiple flows as a single aggregate. For example, SFQ uses a fixed number of queues and serves multiple flows from the same queue. Flows within an aggregate are not isolated from one another and hence share queuing delay and loss characteristics.

We develop Multiple Queue Fair Queuing (MQFQ), an SFQ extension where the router also maintains a fixed number of FIFO queues but allows a flow to access more than one queue. Upon arrival of a packet from a flow, the router places the packet into the shortest of the queues associated with the flow. MQFQ intends to serve different flows from different sets of queues so that a misbehaving flow is unable to congest all queues of another flow. Our investigation shows that two queues per flow are optimal because a higher number of per-flow queues permits a greedy flow to grab a larger fraction of the link capacity. We also experimentally compare MQFQ with SFQ and SFB.

II. RELATED WORK

SFQ maps each flow into one of a fixed number of FIFO queues. When the number of flows is high, a flow shares its queue with other flows. SFQ strives to distribute flows evenly among all queues and thereby support statistically fair sharing in networks with well-behaving traffic. Furthermore, SFQ offers some protection against greedy flows by limiting the impact of excessive transmission on the flows in other queues. In particular, if all k queues of the link are backlogged, the service rate for any flow is capped at the fair share of one queue, i.e., $1/k$ of the link capacity.

SFB extends Blue [14], which itself is an enhancement of Random Early Detection (RED) [15]. Blue maintains

a single FIFO queue but might discard an incoming packet even if the link buffer is not full. The discard probability depends on current and past queuing: while a buffer overflow increases the discard probability, the discard probability decreases whenever the queue empties. SFB extends Blue by adding a Bloom filter to take over calculation of the discard probability. The Bloom filter uses multiple hash functions to assign each flow to a fixed number of bins. Every bin has a fixed size and variable discard probability. When a packet arrives from a flow, SFB increments a counter for each bin associated with the flow. If a bin overflows, SFB discards the packet and increases the discard probability of the bin. SFB discards the incoming packet with probability equal to the minimum among the discard probabilities of the bins associated with the flow. If a bin empties, its discard probability decreases.

A different way to deal with a misbehaving flow is to detect it and limit its rate explicitly [16]–[18]. However, the identify-and-limit approach suffers from the following drawbacks: 1) its effectiveness depends on the traffic pattern: e.g., during coordinated attacks, the number of misbehaving flows that need to be identified and rate-limited might exceed the maximum supported by the router; 2) since identification takes time, rate-limiting kicks in only after some delay; 3) mild cheaters that inflate transmission modestly might evade detection. To ameliorate these problems, identify-and-limit schemes can adopt the technique proposed in this paper. In particular, one can identify and rate-limit greedy flows first and apply our multiple-queue technique to the remaining flows.

III. MULTIPLE QUEUE FAIR QUEUING

Multiple Queue Fair Queuing (MQFQ) is an SFQ enhancement that allows a flow to utilize multiple queues. Instead of a single hash function as in SFQ, MQFQ uses multiple hash functions to determine a set of FIFO queues for a flow. When a packet arrives, MQFQ applies all hash functions to the packet header to compute potential queues. MQFQ puts the packet into the queue with the soonest service. If one queue associated with a flow grows large, the flow uses another of its queues and thereby bypasses the congestion. Since a flow can flood multiple queues, there exists a trade-off between the degree of extra capacity surrendered to a misbehaving flow and the number of flows starved by the misbehaving flow. As we show later, using two queues per flow is most beneficial. Unless explicitly stated otherwise, our subsequent references to MQFQ denote its instance with two hash functions. As in SFQ, MQFQ serves all queues in the round-robin order.

In its extended version [19], our paper analyzes interference between flows under SFQ and MQFQ. Below,

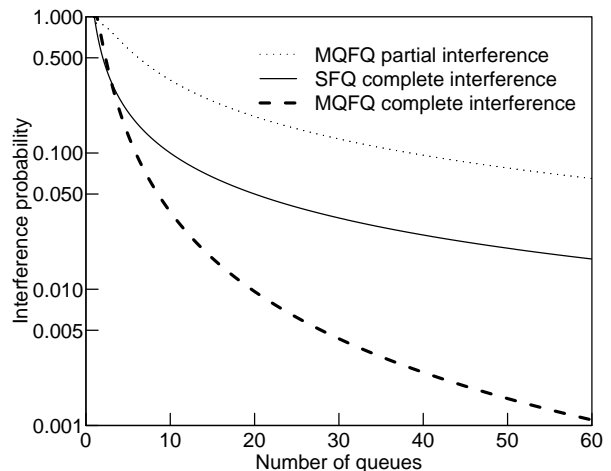


Fig. 1. Probabilities of flow interference under MQFQ and SFQ.

we report the derived probabilities of flow interference. In SFQ with k queues, two flows interfere when hashed into the same queue:

$$P_{\text{SFQ}} = \frac{1}{k}. \quad (1)$$

In MQFQ, two flows interfere either completely when flow x shares all its queues with flow y :

$$P_{\text{MQFQ}}^{\text{comp}} = \frac{4}{k^2} - \frac{3}{k^3} \quad (2)$$

or partially when flow x shares at least one of its queues with flow y :

$$P_{\text{MQFQ}}^{\text{part}} = \frac{4}{k} - \frac{6}{k^2} + \frac{3}{k^3}. \quad (3)$$

Figure 1 shows that all three probabilities decrease when the number of queues grows. While interference under SFQ and partial interference under MQFQ diminish similarly as $O(\frac{1}{k})$, complete interference under MQFQ decreases much faster as $O(\frac{1}{k^2})$.

Although MQFQ might place packets of a flow into different queues, no packet reordering occurs if packets have the same size. If packet sizes are different, reordering is possible but contained within one round of queue traversal. One remedy is to buffer packets for one round to restore their order before sending them into the link. Alternatively, if the router fragments incoming packets into equally-sized cells, applies MQFQ to the cells, and reassembles the packets before sending them into the link, then reordering affects neither the cells nor the reassembled packets.

IV. EVALUATION

We conduct simulations in ns-2 version 2.29 [20] to compare MQFQ with SFQ and SFB experimentally. For SFQ, we modify the ns-2 default implementation by improving the hash functions to avoid their frequent

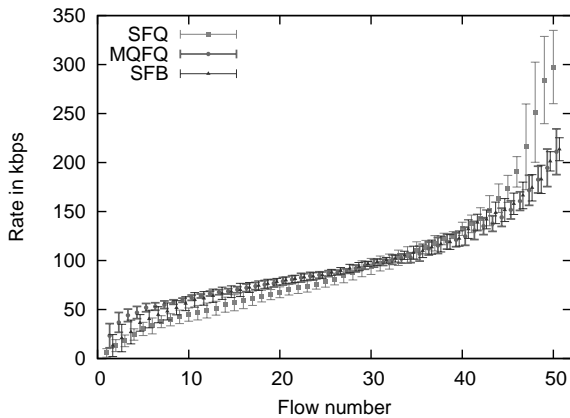


Fig. 2. Responsive traffic from 50 well-behaving TCP flows.

collisions. We implement MQFQ by extending our implementation of SFQ. While all queues share the link buffer space, we allow a queue to grow beyond its fair memory share if free space is at least the queue size plus two packets. The code of our implementations is publicly available [21].

We experiment in a single-bottleneck dumbbell topology. The core bottleneck link has capacity 5 Mbps. The capacity of each access link is 100 Mbps. Round-trip propagation delays are fixed at 60 ms for constant-bitrate (CBR) flows but distributed uniformly between 60 and 80 ms for TCP flows. All experiments use 1,000-byte packets and 100-packet link buffers. We schedule the bottleneck link using one of the evaluated schemes. All other links adhere to FIFO queuing and discard packets only upon buffer overflow. Unless stated otherwise, SFQ and MQFQ use 16 queues. SFB employs two levels of bins with 23 bins on each level. We measure steady-state throughputs between 10 and 50 seconds into each experiment and repeat the experiment 10 times. Plotted results order flows by throughput and depict one standard deviation with error bars. We shift the plots for MQFQ and SFB by 0.2 and 0.4 respectively in order to reduce overlap and improve readability of our graphs.

First, we examine well-behaving settings where the bottleneck link carries data from fifty TCP flows. Figure 2 shows that SFQ, SFB, and MQFQ provide flows with similar distributions of throughput. Because SFQ yields the widest range of individual throughputs, fairness is the worst under SFQ. While slowest flows achieve highest throughput under MQFQ, the graph confirms the intended fairness benefits of MQFQ.

We modify the above scenario by replacing one of the TCP flows with a CBR flow that persistently transmits at an unfairly high rate of 2.5 Mbps. Since the CBR flow always acquires the highest throughput under each of the examined schemes, this flow always appears in Figure 3 as number 50. Under SFQ, the misbehaving flow forces all TCP competitors out from its queue and acquires

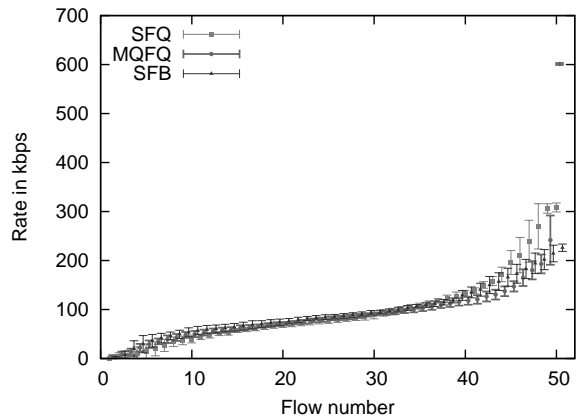


Fig. 3. An unfair CBR flow against 49 well-behaving TCP flows.

more than 300 kbps, i.e., about a one-queue share of the link capacity. Due to uneven distribution of flows among queues, SFQ also gives out unfairly high rates to some TCP flows at the expense of other TCP flows. Whereas SFB restricts the CBR flow most successfully, MQFQ is not as effective and moreover surrenders almost a two-queue share of link capacity to the misbehavior. MQFQ lives up to its intention to protect individual flows: slowest TCP flows receive highest throughput under MQFQ.

The benefits from MQFQ are most apparent when the number of misbehaving flows is large. Figure 4 reports throughputs in experiments where fifty CBR flows transmit at an unfairly high rate about 150 kbps each and thereby overload the 5-Mbps link by 50%. The CBR transmission is randomized, allowing a sending rate to deviate slightly from the 150-kbps average. Throughputs under MQFQ vary the least: from 74 to 138 kbps. The throughput range under SFQ is wider: from 56 to 152 kbps. SFB falters dramatically: the multitude of misbehaving flows overwhelms the Bloom filter, inflates the discard probabilities, and prevents SFB from utilizing the bottleneck link fully. SFB limits individual rates for 27 flows to about 20 kbps. Large error bars for the other flows indicate that SFB limits each of these flows in some but not all of our ten experiments.

Finally, we change the number of queues per flow and denote the respective version of MQFQ by appending the number to its name: SFQ is MQFQ1, regular MQFQ is MQFQ2, etc. As Figure 5 shows for fifty well-behaving TCP flows, 2 queues versus 1 queue per flow yield significant improvement but increasing the number of queues above 2 provides only marginal extra benefits. Furthermore, our other experiments [19] confirm that a misbehaving CBR flow is able to grab almost the entire rate allocated to the queues that the flow can access. Although the misbehavior acquires the extra capacity primary at the expense of fast flows, slow flows do not benefit from enriching the CBR flow either. Hence, we

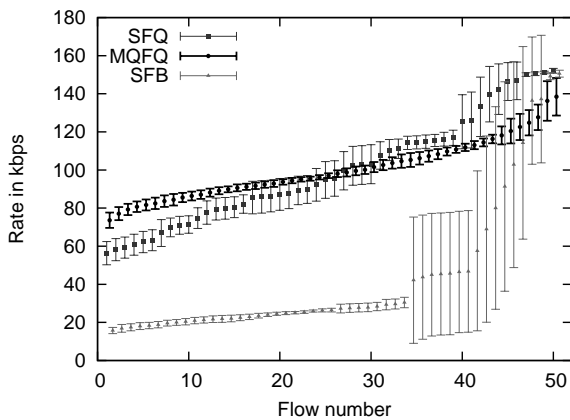


Fig. 4. Link overload by fifty CBR flows.

conclude that two queues per flow constitute the best configuration for MQFQ.

V. CONCLUSION

In this paper, we explored how to protect individual flows from unfair cross traffic in a router that provides isolation only between flow aggregates. In particular, we proposed and evaluated MQFQ, a discipline that employs a fixed number of FIFO queues and multiple hash functions. By applying the hash functions to headers of incoming packets, MQFQ strives to associate each flow with a different subset of the FIFO queues. When a packet arrives from a flow, the router places the packet into the shortest queue associated with the flow. Our investigation showed that two queues per flow are optimal. While packet reordering within a flow might severely undermine TCP performance, we discussed avoidance of such reordering. We also compared MQFQ with SFQ and SFB experimentally in both well-behaving and misbehaving settings. Our experiments confirmed that slowest flows achieve highest throughput under MQFQ. With a single aggressive CBR flow, the protection of weakest flows comes at the price of surrendering more capacity to the misbehavior. However, when the number of misbehaving flows is large, MQFQ balances all individual flow throughputs much more fairly than under SFQ or SFB.

REFERENCES

- [1] J. Nagle, "On Packet Switches With Infinite Storage," *IEEE Transactions on Communications*, vol. 35, no. 4, pp. 435–438, April 1987.
- [2] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queuing Algorithm," in *Proceedings of ACM SIGCOMM*, 1989, pp. 1–12.
- [3] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks – The Multiple Node Case," *IEEE Transactions on Networking*, vol. 2, no. 2, pp. 137–150, 1994.
- [4] S. Golestani, "A Self-Clocked Fair Queuing Scheme for Broadband Applications," in *Proceedings of IEEE INFOCOM*, 1994, pp. 636–646.

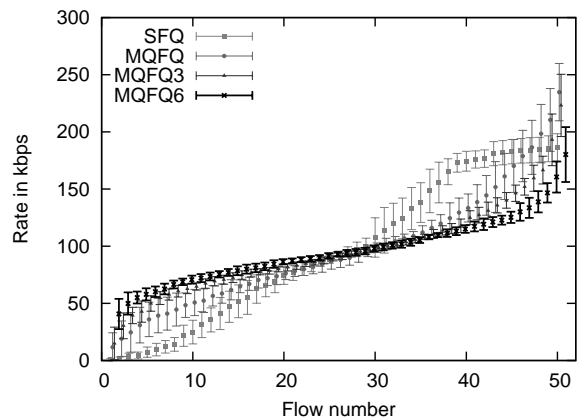


Fig. 5. Fifty well-behaving TCP flows under MQFQ with different numbers of queues per flow.

- [5] M. Shreedhar and G. Varghese, "Efficient Fair Queuing using Deficit Round Robin," in *Proceedings of ACM SIGCOMM*, 1995, pp. 231–242.
- [6] J. C. Bennett and H. Zhang, "WF2Q: Worst-case Fair Weighted Fair Queuing," in *Proceedings of IEEE INFOCOM*, March 1996, pp. 120–128.
- [7] S. Ramabhadran and J. Pasquale, "Stratified Round Robin: A Low Complexity Packet Scheduler with Bandwidth Fairness and Bounded Delay," in *Proceedings of ACM SIGCOMM*, 2003, pp. 239–249.
- [8] D. Bertsekas and R. Gallager, *Data Networks*. Prentice-Hall, 1987.
- [9] J. Mahdavi and S. Floyd, "TCP-Friendly Unicast Rate-Based Flow Control," January 1997, end2end-interest mailing list.
- [10] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," in *Proceedings of ACM SIGCOMM*, September 1998.
- [11] P. E. McKenney, "Stochastic Fairness Queuing," in *Proceedings of IEEE INFOCOM*, June 1990, pp. 733–740.
- [12] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin, "Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness," in *Proceedings of IEEE INFOCOM*, 2001, pp. 22–26.
- [13] R. Mahajan, S. Floyd, and D. Wetherall, "Controlling High-Bandwidth Flows at the Congested Router," in *Proceedings IEEE ICNP 2001*, November 2001.
- [14] W. Feng, K. Shin, D. Kandlur, and D. Saha, "The BLUE Active Queue Management Algorithms," in *Proceedings of IEEE/ACM Transactions on Networking*, August 2002, pp. 513–528.
- [15] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," in *Proceedings of IEEE/ACM Transactions on Networking*, August 1993, pp. 397–413.
- [16] R. Pan, B. Prabhakar, and K. Psounis, "CHOCkE, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation," in *Proceedings of IEEE INFOCOM*, March 2000.
- [17] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting," in *Proceedings of ACM SIGCOMM*, August 2002.
- [18] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker, "Approximate Fairness Through Differential Dropping," *ACM SIGCOMM CCR*, vol. 33, no. 2, pp. 23–39, 2003.
- [19] M. Georg, C. Jechlitschek, and S. Gorinsky, "Improving Individual Flow Performance with Multiple Queue Fair Queuing," Washington University in St. Louis, Tech. Rep. WUCSE-2007-30, May 2007.
- [20] "The Network Simulator – ns-2," <http://www.isi.edu/nsnam/ns/>.
- [21] C. Jechlitschek, "MQFQ Implementation in ns-2 and Supporting Scripts," February 2007, <http://www.cse.wustl.edu/~mgeorg/mqfq/>.