Queue Length Based Fair Queueing in Core-Stateless Networks¹

Mingyu Zhai, Guanqun Gu, and Yuan Yuan

Department of Computer Science and Engineering, Southeast University, Nanjing 210096, China

The Key Laboratory for Computer Networks and Information Integration, Ministry of Education, China
{myzhai, ggq, yyuan}@seu.edu.cn

Abstract. In this paper, we present Queue Length based Fair Queueing (QLFQ), a scheme to approximate fair bandwidth allocation without per-flow state. We have evaluated QLFQ together with CSFQ and RFQ with several different configurations and traffic sources. The simulation results show that QLFQ is able to achieve approximately fair bandwidth sharing in all of these scenarios. The performance of QLFQ is comparable to that of CSFQ, and it performs much better than RFQ.

1 Introduction

Many router mechanisms [1, 2] have been developed to achieve fair bandwidth allocations. Generally, most of these mechanisms need to maintain per-flow state and therefore are more complex and less scalable than simple FIFO queueing when they are used in the interior of a high-speed network. Consequently, design of algorithms that can provide per-flow fair rate without requiring per-flow functionality in the network core routers has become an active area of research.

Recently several frameworks and mechanisms have been proposed to overcome the implementation complexity problem and address the scalability and robustness requirements. Stoica *et al.* propose a new scheme called Core-Stateless Fair Queueing (CSFQ)[3], which approximates fair queueing while limiting flow state to the edge of network and removing it from core routers. Cao *et al.* proposed Rainbow Fair Queueing (RFQ) [4], a similar scheme that avoids fair share rate calculation in core routers and that is better adapted to layered encoding applications. Both schemes remove flow state but still require *computation* to determine dropping thresholds for example.

RFQ is a combination of a layering scheme and a buffer management mechanism. This is the direction we pursue in this paper, where we take it to its simplest configuration. In our approach, in order to determine the dropping threshold, we *only*

---

¹ This work is supported by Chinese NSF (item number: 69896240).

need to know queue occupancy information (i.e. Queue Length) and do few *comparison* operations in core routers (so we call our scheme Queue Length based Fair Queueing (QLFQ)). Compared to RFQ and the other approaches of achieving reasonable fairness, we believe that our scheme has one of the lowest implementation cost in core routers. The simulation results demonstrate that the performance of QLFQ outperforms RFQ and is comparable to CSFQ.

2 Queue Length Based Fair Queueing (QLFQ)

2.1 Network Architecture

The network architecture is the same as that used in CSFQ, RFQ and in Differentiated Services: namely, a network comprised of edge routers and core routers. The edge routers perform packet classification and encode certain state in packet headers, and the core routers use the state for packet discarding.

2.2 Computation of Flow Arrival Rate

At the edge routers, the flow arrival rate must be estimated, in order to assign a label to a packet. To estimate the flow arrival rate, we use the same exponential averaging formula as in [3,4]. This scheme requires an edge router to keep state for each active flow. Specifically, 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 , γ_i , is updated every time a new packet is received:

$$\gamma_i^{\text{new}} = (1 - e^{-T_i^k / K})(l_i^k / T_i^k) + e^{-T_i^k / K} \gamma_i^{\text{old}} \quad (1)$$

where $T_i^k = t_i^k - t_i^{k-1}$ and K is a constant. Using an exponential weight $e^{-T_i^k / K}$ gives more reliable estimation for bursty traffic, even when the packet inter arrival time has significant variance.

2.3 Flow Layering

Flow layering is an important part in both QLFQ and RFQ, by which edge routers partition a flow and assign a number to each layer of the flow. The numbered layers have two purposes. First, they reflect the rate of the flow: the larger the number of layers is, the higher the rate of the flow is; flows with the same rate have the same number of layers. Second, the numbered layers provide a structure for controlled discarding in the network when congestion occurs. That is, when congestion is detected, the core routers will first discard the largest layer and then the second largest layers and so on.

The preliminary requirement of flow layering is to select layer rates for a flow. If a flow has an arrival rate r , and c_i is the rate of layer i , then the flow will be partitioned

into $(j + 1)$ layers, where j is the smallest value satisfying $\sum_{i=0}^j c_i \geq r$. Contrary to RFQ, which uses a *non-linear* layering scheme, QLFQ uses a *linear* layering scheme. More precisely, QLFQ make all layers have *equal* rate. Specifically, if P is the maximum flow rate in the network and N is the total number of layers, the layer rate c is P/N . For example, if a flow has a maximum rate of 16Kbps, and RFQ partitions it into eight layers, then the rate for each layer is 1, 1, 1, 1, 2, 2, 4, 4 (unit: Kbps). Under the same conditions, for QLFQ the layer rate will be 2, 2, 2, 2, 2, 2, 2, 2 (unit: Kbps). Apparently the QLFQ scheme can simplify operations of edge routers and is amenable to hardware implementation.

2.4 Packet Labeling

After estimating the flow arrival rate, each packet is assigned a label, with the constraint that the average rate of packets in each layer is at most c . In QLFQ, to label a packet we use a scheme similar to that of RFQ. But our scheme requires lower computational overhead than RFQ because of the simple flow layering scheme used in QLFQ. Specifically, if the current estimate of the flow arrival rate is r , and j is the smallest value satisfying $j * c \geq r$. Then the current packet is assigned a label $i : 0 \leq i \leq (j-1)$ with probability $1/j$. Whether all packets have fixed size or not, it is easy to see that the probabilistic label assignment will cause the rates for each layer to approach the required rates c . Moreover, the probabilistic label assignment can smooth a burst of packets and avoid temporary buffer overflow.

2.5 Core Router Algorithm

Using QLFQ core routers can achieve approximate fair queueing while maintaining high utilization. In order to do so, the core routers need to monitor queue occupancy and accordingly set a dropping-threshold. When network is congested, the core routers begin to discard layers whose value exceeds the dropping threshold.

The main task of the core router algorithm is to update the dropping threshold according to the current congestion status. The principle behind this core router algorithm is that the queue occupancy reflects the congestion status of the core router, so we can use the queue length information to set the dropping threshold. Namely, the dropping threshold $Drop\_thresh^{new}$ is updated as following:

$$Drop\_thresh^{new} = f(Drop\_thresh^{old}, qlen^{old}) \quad (2)$$

where $Drop\_thresh^{old}$ and $qlen^{old}$ are the dropping threshold and queue length before a new packet arrives respectively.

According to the above principle, in the current implementation of QLFQ we logically divide the queue of an output link into four segments using three queue length thresholds (see Figure 1):

Fig. 1. Queue Partition

Segment A where queue length $qlen$ is less than $Expo\_thresh\_$ - Because the queue is nearly empty in the segment, we drastically increase the dropping threshold and make the aggregate arriving rate match the link capacity as soon as possible.

Segment B where $qlen$ lies between $Expo\_thresh\_$ and $Incr\_thresh\_$ - In the segment, the link is uncongested and the buffer capacity has not been sufficiently utilized, so we smoothly increase the dropping threshold.

Segment C where $qlen$ stays between $Incr\_thresh\_$ and $Decr\_thresh\_$ - With both the link and buffer capacity at full use, we keep the dropping threshold unchanged and make the system work steadily.

Segment D where $qlen$ is larger than $Decr\_thresh\_$ - When the queue length exceeds $Decr\_thresh\_$, the link becomes congested, and we reduce the dropping threshold to get more packets dropped.

Initially, the dropping threshold $drop\_thresh$ is set to the maximum layer value Max_layer . When the current queue is in Segment A and its length $qlen$ has been reduced since the last $drop\_thresh$ update, the new $drop\_thresh$ is computed as:

$$drop\_thresh^{new} = drop\_thresh^{old} + 2^i \quad (3)$$

where i is the number of *continually* arrived packets under the above conditions. And also we track the variation of current queue length $qlen$ so as to prevent $drop\_thresh$ from changing very frequently and ensure stability. If the current queue stays between $Expo\_thresh\_$ and $Incr\_thresh\_$ and its length $qlen$ has been reduced since the last $drop\_thresh$ update, we only increase $drop\_thresh$ linearly using a simple way as following:

$$drop\_thresh^{new} = drop\_thresh^{old} + 1 \quad (4)$$

We decrease $drop\_thresh$ when the current queue length $qlen$ is larger than $Decr\_thresh\_$. But here we discuss it with two aspects: (i) the queue is not full and the current queue length has increased since the last $drop\_thresh$ update - Under the condition, if $drop\_thresh$ equals to Max_layer , we set $drop\_thresh$ to $current\_max\_layer$, which is updated to record the largest layer number *recently* seen when the queue is not in the Segment D. Subsequently, we linearly decrease $drop\_thresh$ as following:

$$drop\_thresh^{new} = drop\_thresh^{old} - 1 \quad (5)$$

(ii) the queue is full – Under this condition, we need reduce *drop_thresh* quickly to alleviate congestion. So we first use a non-linear formula to compute *drop_thresh*:

$$drop\_thresh^{new} = drop\_thresh^{old} - MAX(1, 0.1 * drop\_thresh^{old} * drop\_thresh^{old} / Max\_layer) \quad (6)$$

Secondly, we set *drop_thresh* to *current_max_layer* if *drop_thresh* is still larger than *current_max_layer*. To show the efficiency of expression (6), we give an example. Assuming that the maximum layer number is 10000, and the current dropping threshold is 8000, then the new dropping threshold is 7360. If the current dropping threshold is 1000, then the dropping threshold is reduced to 990. That is, the larger the dropping threshold is, the more it is reduced.

At core routers, both the time and space complexity of QLFQ are constant with respect to the number of competing flows, and thus QLFQ could be implemented in very high speed core routers. In fact, In core routers QLFQ only needs queue occupancy information and conduct few comparison operations to determine the dropping threshold in the steady status. It should be pointed out that the *multiplication* operation in expression (6) is conditioned, and there is little probability to execute it unless the traffic is very bursty (the *division* operation in (6) can be avoided because *Max_layer* is a constant). Our simulations also prove this point.

3 Conclusion

In this paper, we present Queue Length based Fair Queueing (QLFQ), a scheme to approximate fair bandwidth allocation without per-flow state. We believe that, by its simplicity and efficiency, QLFQ is an interesting approach for future high-speed network.

References

1. Parekh, A. A generalized processor sharing approach to flow control - the single node case. In : Proceedings of the IEEE INFOCOM'92, 1992.
2. Lin, D. Dynamics of random early detection. In : Proceedings of ACM SIGCOMM '97,Cannes, France, October 1997, 127-137
3. Stoica, I. Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high-speed networks. In : Proceeding of ACM SIGCOMM'98,1998,118-130
4. Cao, Z. Rainbow fair queueing: Fair bandwidth sharing without per-flow state. In : Proceedings of IEEE Infocom '2000,March 2000