An Approximation to Rate-Equalization Fairness with Logarithmic Complexity for QoS

Jorge A. Cobb Suparn Gupta
Department of Computer Science
The University of Texas at Dallas
Richardson, TX 75083-068875080-3021
Email: {cobb, suparn.gupta}@utdallas.edu

***Abstract*—Rate-guaranteed scheduling protocols ensure that the packets from each of their input flows are forwarded at a rate no less than the rate reserved by the flow. Weighted Fair Queuing (WFQ) is the classical example of this protocol family. Many of these protocols, including WFQ, provide both rate and fairness guarantees. In particular, they distribute unused capacity among among the flows in proportion to the reserved rate of each flow. Thus, flows whose reserved rate is the largest receive a larger share of the unused capacity. In earlier work, we presented a scheduling algorithm that first distributes unused capacity to those flows whose reserved rate is the least. However, the per-packet complexity of this algorithm, known as rate-equalization (EQ) fairness, is linear in the number of flows. In this paper, we present an algorithm that approximates the behavior of EQ fairness but with only logarithmic complexity per packet. Simulation results show that in practical situations the behavior of the approximation algorithm is quite similar to that of pure EQ fairness.**

Keywords: **QoS, Real-Time Traffic, Scheduling Fairness.**

I. Introduction

Real-time applications, such as interactive audio and video, require from the network an assurance that their packets will be forwarded with a lower bound on the forwarding rate and also with a bounded end-to-end delay. Rate-guaranteed schedulers [6], [10], [12] is a family of protocols that is able to provide this assurance.

Iconic examples of this protocol family include Virtual Clock (VC) [22], [24] and Weighted Fair Queuing (WFQ) [17], plus their many variations. One desirable property of a rate-guaranteed scheduler is fairness. That is, a flow should not be "punished" (temporarily denied service) if it exceeds its reserved rate to take advantage of unused bandwidth in the channel. Some protocols, such as Virtual Clock, are unfair, while others, like WFQ, are fair.

Being able to provide fairness is desirable for some applications whose can adapt to changes in the bandwidth provided by the network. Examples of such flows are file transfers and multi-resolution video [16]. The sources of these flows will likely reserve from the network the smallest packet rate necessary to receive a minimum quality of service. The source is then able to detect that additional bandwidth is available due to side-effects observed in the network, such as a smaller end-to-end delay, and increase its packet-generation rate accordingly.

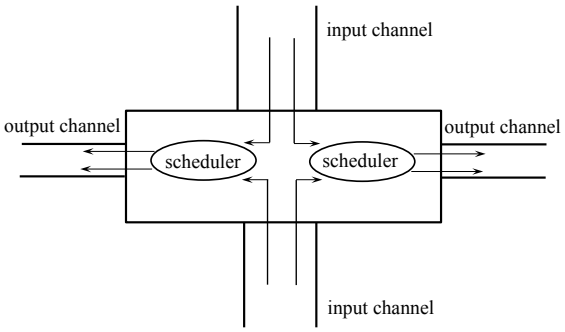


Fig. 1. Schedulers and Channels

Most scheduling protocols that provide both rate and fairness guarantees, such as WFQ and its variants [17] [1] [7] [11] [2], distribute the unused capacity among the flows in proportion to the reserved rate of the flows. Thus, if flow f reserved twice the rate as flow g, then f's share of the unused bandwidth is twice that of g.

In earlier work [9], we presented an alternative form of fairness for rate-guaranteed schedulers: *rate-equalization*. Our protocol firsts distribute unused capacity to flows whose reserved rate is the least. This allocation continues until the flows with least reserved rate and the flows with the next-to-least reserved rate are given the same capacity. This continues, level by level, until, if enough unused capacity is available, all flows will receive the same capacity, and the scheduler will simply behave like Fair Queuing.

The disadvantage of this protocol is its computational complexity, which is in $O(n)$ per packet received, where n is the number of flows. In this paper, we present a protocol that approximates the behavior of rate-equalization, but with a smaller $O(\log(n))$ complexity. We show that this approximation also belongs to the family of rate-guaranteed schedulers, and furthermore, we demonstrate through simulations that its behavior closely resembles that of an exact rate-equalization scheduler.

II. QoS Model

Our QoS model is based on the models of [6] and [10]. A *flow* is a sequence of packets that has quality of service

CR Server

let $B(t)$ be the set of backlogged flows at time t;
for every flow f,
if $f \in B(t)$, then $\psi_{f,CR}(t) = R_f$;
if $f \notin B(t)$, then $\psi_{f,CR}(t) = 0$;

Fig. 2. Constant-rate fluid server

requirements. Each output channel of a computer is equipped with a packet scheduler (see Figure 1). A scheduler receives packets from flows whose path traverses its output channel. Whenever its output channel becomes idle, the scheduler chooses a received packet and forwards it over the output channel. A packet *exits* a scheduler when its last bit packet is transmitted by the output channel. We adopt the following notation for each flow f.

R_f	data rate reserved for flow f.
$p_{f,i}$	i^{th} packet of f, $i \geq 1$.
$L_{f,i}$	length of packet $p_{f,i}$.
L^{max}	maximum packet length of all flows.
$A_{f,i}$	arrival time of $p_{f,i}$.
$E_{f,i}$	exit time of $p_{f,i}$.
C	bandwidth of the output channel.

Consider a constant-rate (CR) fluid server[1] whose input is a set of flows, among them f. Let this server forward the bits of each input flow f at exactly its reserved rate R_f. Such a fluid server is defined in Fig. 2, where $B(t)$ are the *backlogged flows* at time t, i.e., flows with bits remaining to be forwarded, and $\psi_{f,CR}(t)$ is the bit rate given to flow f at time t.

We define $S_{f,i,CR}$ as the *start-time* of packet $p_{f,i}$ at the constant-rate server and $F_{f,i,CR}$ as the *finishing time*. They can be computed recursively as follows, where $S_{f,1,CR} = A_{f,1}$.

$$\begin{aligned} S_{f,i,CR} &= \max(A_{f,i},\ F_{f,i-1,CR}) \quad i > 1 \\ F_{f,i,CR} &= S_{f,i,CR} + \frac{L_{f,i}}{R_f} \quad i \geq 1 \end{aligned} \tag{1}$$

Note that $F_{f,i,CR}$ is the value used by the Virtual Clock scheduling protocol to assign priorities to its packets [22], [24].

Assume that a packet scheduler s forwards the packets of an input flow f at a rate of at least R_f. Then, each packet $p_{f,i}$ exits scheduler s not much later than its finishing time at CR, i.e., $F_{f,i,CR}$. We refer to these packet schedulers as *rate-guaranteed schedulers* [6], [10], [12]. More formally, a scheduler s is a rate-guaranteed scheduler iff, for every input flow f of s and every i, $i \geq 1$,

$$E_{f,i,s} \leq F_{f,i,CR} + \delta_{f,s} \tag{2}$$

for some constant $\delta_{f,s}$. For many protocols, such as Virtual Clock [22], [24] and Weighted Fair Queuing [17] (and their many variations),

$$\delta_{f,s} = L_s^{max}/C_s$$

I.e., packets exit by their finishing time in CR plus at most the time to transmit the largest packet size. In this case, admission control is simple[2], the sum of the reserved rates of all the flows through s must be at most C.

III. Rate-Equalization Fairness

We next overview the motivation for rate-equalizing fairness, which we introduced in [9].

On occasions, the bandwidth of a channel is not fully utilized. This occurs either because existing flows are transmitting at a rate that is below their reserved rate, or because the sum of the reserved rates of the flows is less than the output channel's bandwidth. Under these conditions, it is possible for a flow to temporarily exceed its reserved rate, in an attempt to take advantage of bandwidth unused by other flows.

The manner in which unused bandwidth is distributed among flows varies from one protocol to another. We refer to this distribution of unallocated bandwidth as the *fairness method* of the protocol, and it is the main focus of this paper.

Some protocols, like Virtual Clock (VC) [24][22], do not address fairness. In VC, each packet $p_{f,i}$ is labeled with its virtual finishing time, $F_{f,i,CR}$ (see (1)), and packets are forwarded by the scheduler in order of increasing labels. A consequence of this is that, if a flow exceeds its reserved rate, it may later be denied service by the scheduler, for a duration proportional to the time the flow exceeded its rate [7]. The exit bound in (2) still holds, however, because VC is a rate-guaranteed scheduler.

Other rate-guaranteed schedulers, such as Weighted Fair Queuing (WFQ) [17] and its variants [1] [7] [11] [2], distribute unused bandwidth among flows in proportion to the reserved rate of the flow. Specifically, the effective rate ψ_f that is given to flow f (i.e., the rate at which the scheduler actually forwards the packets of flow f) is

$$\psi_f(t) = \frac{C}{\left(\sum_{g \in B(t)} R_g\right)} \cdot R_f \geq R_f \tag{3}$$

Consider another flow g with lesser reserved bandwidth, e.g., $R_f = 2 \cdot R_g$. From (3), it can be easily shown that

$$(\psi_f - R_f) = 2 \cdot (\psi_g - R_g)$$

Hence, the fairness method of WFQ favors flows with a higher reserved rate. The intuition behind WFQ's fairness method is as follows. If a flow has a reserved rate that is greater than that of other flows, it implies that the owner of the flow is paying a greater price for the network service, and thus, should receive a greater share of the unallocated bandwidth.

In [9], we introduced an alternative fairness method in which the objective is to give every flow the same effective rate, provided enough unused bandwidth is available. This

[1]Fluid servers can forward, during any time interval, an arbitrary number of bits from any subset of its input flows. They are for reference only and cannot be implemented. This is in contrast to packet schedulers which are used in practice and can only forward one packet at a time.

[2]Other protocols have a *rate-independent delay* [10], [25]: where $\delta_{f,s}$ could be negative. This allows for a smaller per-hop delay, but makes the admission control test quite complex. Such protocols are outside the scope of this paper.

results in an effective rate ψ_f that is different from Equation (3). The intuition behind it is the following. Flows whose applications are rate-adaptive could reserve the minimum rate possible to satisfy their QoS requirements, and thus minimize expense. Any additional bandwidth is given to flows that need it the most, i.e., flows with the least reserved rate.

A more detailed description is as follows. First, at all times, the effective rate of any flow f, ψ_f, is at least its reserved rate, R_f. I.e., $R_f \leq \psi_f$. Next, consider another flow g, where $R_f < R_g$. By definition, $R_g \leq \psi_g$. In our method, if enough unallocated bandwidth is available, ψ_f will increase until it becomes equal to R_g, and thus, ψ_f will become equal to ψ_g. Thus, flows with lower reserved rates will "catch up" to flows with larger reserved rates.

Assume that more unallocated bandwidth remains. In this case, the remaining unallocated bandwidth will be distributed equally between f and g, maintaining the relationship $\psi_f = \psi_g$. If there exists another flow h, where $R_h > R_g$, then ψ_f and ψ_g increase equally until they reach R_h (assuming enough bandwidth remains), and hence, $\psi_f = \psi_g = \psi_h$.

To summarize, our fairness method attempts to give all flows the same effective rate. However, in doing so, the requirement of $R_f \leq \psi_f$ for all f must be preserved at all times.

We next describe our fairness method in a more formal way.

IV. Rate-Equalization Server and Scheduler

Packet scheduling algorithms that provide fairness, such as WFQ and some of its variants, describe their fairness method via a virtual fluid server. The packet scheduler then mimics the fluid server as much as possible. The fluid server and the packet scheduler have the same input flows. Both have an output channel, and both of these channels have equal capacity.

What distinguishes the fluid server from the packet scheduler is the manner in which it forwards bits. Once the packet scheduler begins to transmit a packet, the transmission of the packet cannot be preempted. The fluid server, on the other hand, can concurrently forward an arbitrary number of bits from a group of flows. This, of course, is bounded by the capacity (bits/sec.) of the output channel.

A. Fluid Server

In light of our earlier discussion on fairness, we define the rate-equalization (EQ) fluid server as follows. Let $\psi_{f,EQ}(t)$ be the instantaneous bit rate given to flow f by the fluid server. This value is computed as shown in Figure 3. If $\psi_{f,EQ}(t)$ does not change during an interval $[t_1, t_2]$, then the total number of bits of f forwarded during this interval is $\psi_{f,EQ}(t_1) \cdot (t_2 - t_1)$.

The steps shown in Figure 3 are as follows. First, the set $B(t)$ of backlogged flows (i.e., with bits remaining to be forwarded) is determined. Obviously, a non-backlogged flow receives a rate of zero. All other flows are then arranged in increasing order of their reserved rate. Then, the index j is found such that

$$R_{b_j} \leq (C - \textstyle\sum_{k=j+1}^{m} R_{b_k})/j < R_{b_{j+1}}$$

In this manner, if the higher rate flows $b_{j+1}, \ldots, b_m$ receive exactly their reserved rate, then there is enough remaining bandwidth that can be equally shared among the lower rate flows $b_1, b_2, \ldots, b_j$.

EQ Server

let $B(t)$ be the set of backlogged flows at time t;
for every flow f,
  if $f \notin B(t)$, then
    $\psi_{f,EQ}(t) = 0$
  else
    let $b_1, b_2, \ldots, b_m$ be the flows of $B(t)$
      ordered by increasing R;
    let j, $1 \leq j \leq m$, be the largest index such that
      $$R_{b_j} \leq \frac{C - \sum_{k=j+1}^{m} R_{b_k}}{j} < R_{b_{j+1}};$$
    let $R_{EQ} = \dfrac{C - \sum_{k=j+1}^{m} R_{b_k}}{j}$;
    for each k, $1 \leq k \leq j$, $\psi_{b_k,EQ}(t) = R_{EQ}$;
    for each k, $j+1 \leq k \leq m$, $\psi_{b_k,EQ}(t) = R_{b_k}$;

Fig. 3. Rate equalization fluid server

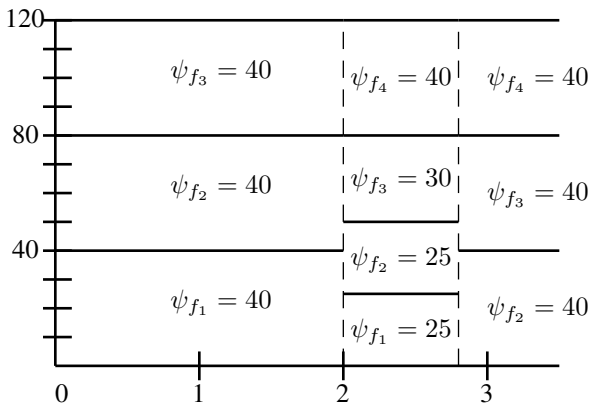


Fig. 4. Fluid server example.

Consider the following example, which is illustrated in Fig. 4. Let $C = 120$ bits/sec.. There are four flows, $f_1 \ldots f_4$, and all packets are 100 bits long. Let $R_{f_1} = 10$ bits/sec., $R_{f_2} = 20$ bits/sec., $R_{f_3} = 30$ bits/sec., and $R_{f_4} = 40$ bits/sec.. Note that $\sum_{i=1}^{4} R_{f_i} = 100$ bits/sec., leaving 20 bits/sec. unallocated.

Assume that at time 0, one packet of f_1 arrives, two packets of f_2 arrive, and two packets of f_3 arrive. At time 2 secs., one packet of f_4 arrives.

At time zero, flows f_1, f_2, and f_3 have bits in their queues. Since $C = 120$, there is enough capacity for all three flows to receive the same effective bandwidth of $\psi = 40$ bits/sec.. At time 2, a packet from f_4 arrives. The total reserved rates of the flows is 100 bits/sec., leaving only 20 bits/sec. to equalize the rates among the flows. Equalizing f_1 to f_2 requires 10 bits/sec.. The remaining 10 bits/sec. are distributed equally between f_1 and f_2, but are not enough to increase their effective rates up R_{f_3}. We thus end with

PEQ Scheduler

upon receiving a packet $p_{f,i}$,
$D_{f,i,PEQ} = \infty$;

if output channel is idle at time t,
model the behavior of EQ up to time t;
let $p_{f,i} \in$ Active(t) iff $t \geq S_{f,i,EQ}$;
for each $p_{f,i} \in$ Active(t),
$D_{f,i,PEQ} = S_{f,i,EQ} + L_{f,i}/R_f$;
let $D_{g,j,PEQ} = \min\{D_{f,i,PEQ} \mid p_{f,i} \in \text{Active}(t)\}$;
forward $p_{g,j}$ to the output channel.

Fig. 5. Packetized rate equalization scheduler

$\psi_{f_1} = \psi_{f_2} = 25$ bits/sec. This leaves $\psi_{f_3} = R_{f_3} = 30$ bits/sec., and $\psi_{f_4} = R_{f_4} = 40$ bits sec.

At time $2\frac{4}{5}$, the last bit of the packet of f_1 exits, so only three flows have a non-empty queue. Again, all remaining flows are given an effective bandwidth of 40 bits/sec. We leave it to the reader to determine the remainder of the example.

B. Packet Scheduler

We next overview the packet scheduler for rate-equalization which we presented in [9], where more details can be found.

In general, the purpose of a fluid server is to guide the packet scheduler in the order it chooses to forward packets. Typically, [5][17][19], for every pair of packets, p_1 and p_2, if p_1 finishes service in the fluid server before p_2 finishes service, then the packet scheduler will forward p_1 before p_2. I.e., the packet scheduler tries to emulate the behavior of the fluid server as much as possible. This emulation, of course, is not perfect, because the packet scheduler can only forward one packet at a time, while the fluid server can forward bits of multiple flows (and hence multiple packets) at once.

For most fluid servers [5][17][19], at the moment a packet arrives, the exit time that this packet will have from the fluid server is unknown. This is because the bit rate at which the packet will be served depends not only on the packets currently in the system, but also on packets that are yet to arrive.

In consequence, when a packet $p_{f,i}$ arrives into a packet scheduler, the scheduler assigns to the packet a *virtual exit time* $T_{f,i}$ (see [17] for details on computing this value), such that, for any other packet $p_{g,j}$, $T_{f,i} \leq T_{g,j}$ iff the exit time of $p_{f,i}$ from the fluid server is at most the exit time of $p_{g,j}$. Packets are then forwarded in order of their virtual exit times. Thus, the packet scheduler forwards packets in the same order in which they are forwarded by the fluid server.

A rate-equalizing fluid server, however, does not have this order-preserving property. That is, if two packets $p_{f,i}$ and $p_{g,j}$ are received, not only can't their exit time from the fluid server be determined, but also their relative exit times cannot be determined. I.e., which of $p_{f,i}$ or $p_{g,j}$ exits first depends on the future arrival of packets.

The reason for not having this property is that the relative effective bandwidth, ψ_g/ψ_f, does not remain constant in rate-equalization. Actually, *not* preserving this ratio is one of the objectives of rate-equalization. Hence, when packets $p_{g,j}$ and $p_{f,i}$ arrive, the scheduler is unable to determine which one will exit the fluid server first.

From the above, the rate equalization packet scheduler (PEQ) cannot assign a virtual exit time T to each packet. Instead, we opted in [9] to assign a real-time deadline, $D_{f,i,PEQ}$, to each packet $p_{f,i}$. The deadline is an *upper bound* on the exit time of $p_{f,i}$ from the fluid server. To obtain this upper bound, we take advantage of the fact that both the fluid server and the packet scheduler have the same input flows and the same output channel capacity. This allows the packet scheduler to keep track of the behavior of the fluid server. The upper bound is as follows.

Let $S_{f,i,EQ}$ be the starting time of $p_{f,i}$ in the fluid server, i.e., when its first bit begins service. Note that scheduler PEQ cannot compute this value when $p_{f,i}$ arrives. However, at time t, where $t \geq S_{f,i,EQ}$, PEQ is aware of this value, because it can keep track of the behavior of the server. Thus, the deadline of $p_{f,i}$ is set to

$$D_{f,i,PEQ} = S_{f,i,EQ} + \frac{L_{f,i}}{R_f}.$$

Then, packets are forwarded in order of this deadline.

Note that since $\psi_f \geq R_f$, the above is a true upper bound on the exit time from the fluid server. Furthermore, since scheduler PEQ cannot compute this value until time $S_{f,i,EQ}$, $p_{f,i}$ is not added to the queue of schedulable packets until this time.

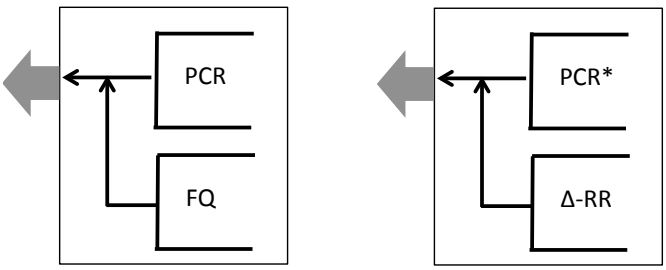
The detailed behavior of the scheduler is shown in Figure 5. More details can be found in [9], including bounds on the difference between a packet's exit time from the scheduler and from the fluid server.

V. Roadblocks to an Efficient Implementation

Recall that our objective is to find an approximation algorithm that will require $O(\log(n))$ processing for receiving and transmitting a packet, where n is the number of flows in the system. Our scheduling protocol resembles WFQ in the sense that we also have a fluid server, and the packet scheduler attempts to emulate it as close as possible.

For many years, the best implementation of WFQ had $O(n)$ complexity. This was due to the overhead of computing the virtual time associated with the arrival time of a packet. The virtual time grows inversely proportional to the number of flows backlogged in the fluid server. The $O(n)$ complexity arises because many flows could become not backlogged in a very short period of time.

Several approximations with $O(\log(n))$ complexity, such as Leap-Forward-VC [20], Time-Shift Scheduling [7], and WFQ+ [2], provided a rough approximation of the virtual time. Other approaches reduced the complexity even further to $O(1)$ by sophisticated variations on the classical round-robin algorithm [13][14][18][23][4][26]. All of these provided



(a) (b)

Fig. 6. Dual-Mode Scheduling

PCR Scheduler

upon receiving a packet $p_{f,i}$,
$D_{f,i,PCR} = F_{f,i,CR}$;

if output channel is idle at time t,
let $p_{f,i} \in$ Active(t) iff $t \geq S_{f,i,CR}$;
if Active$(t) \neq \emptyset$ then
let $D_{g,j,PCR} = \min\{D_{f,i,PCR} \mid p_{f,i} \in \text{Active}(t)\}$;
forward $p_{g,j}$ to the output channel.

Fig. 7. Packetized constant-rate scheduler

the same type of fairness as WFQ, i.e., extra bandwidth is allocated in proportional to the reserved rate.

After many years of only having an $O(n)$ implementation, an $O(\log(n))$ implementation of WFQ was presented in [21]. This required the introduction of a complex search structure that organized the "breaking points" in the virtual time into a search tree. Crucial to making this implementation possible is that the ratio of the effective rates, ψ_g/ψ_f for any pair of backlogged flows, remained the same regardless of the arrival or departure of packets from other flows.

However, the ratio mentioned above is not constant in rate-equalization scheduling. In fact, it can vary significantly. To see this, consider again Figure 3. You can consider the set of backlogged flows to always be divided into two subsets: those flows whose effective rate is their reserved rate, and those flows whose effective rates are all the same due to unused bandwidth. The amount of unused bandwidth in turn depends on how many flows are backlogged, which, like in WFQ, can vary significantly in a very short period of time. This causes many flows to change from one of these two subsets to another, which in turn drastically changes the ration of effective rates.

For these reasons, a technique similar to the one in [21] is not directly applicable. Although we have not proven a lower bound, we speculate that a precise implementation cannot be done in $O(\log(n))$ time. We thus search for an approximation to the fluid server of rate-equalization that runs in $O(\log(n))$ time. Our approximation, presented below, is quite different from that of earlier works mentioned above due to our significantly different method of defining fairness.

VI. Dual-Mode Scheduling

From the above discussion, backlogged flows in the fluid server can be considered to be in one of two disjoint subsets: *enhanced flows*, whose effective rate is greater than their reserved rate and all members have the same effective rate, and *un-enhanced flows*, whose effective rate is simply their reserved rate. This motivates our first packet scheduler design, presented in Section VI-A. Although intuitive, this first attempt is not efficient. We then present our final scheduler design in Section VI-B

A. Flow-Migration Scheduler

Consider Figure 6(a). The service required for an un-enhanced flow f is simply a constant rate R_f. This is provided by a packetized constant rate scheduler (PCR), which is described in more detail in Figure 7. It is similar to the Virtual Clock protocol [22], except that it does not allow flows to exceed their reserved rate. Note that this scheduler is non-work-conserving, i.e., there are times when its queues are non-empty yet it does not have any packets considered 'active', so it remains idle. Enhanced flows, on the other hand, have to be served in an equal manner. This is best accomplished by a fair-queuing (FQ) scheduler, also shown in Figure 6(a).

We thus have two schedulers, one for each type of flow. Priority is given to the PCR scheduler. I.e., when the output channel becomes idle, the PCR scheduler is queried for the next packet to be transmitted. Only if the PCR scheduler is unable to provide a packet (due to its queues being empty or all packets being 'inactive'), then the packet transmitted is chosen from the FQ scheduler. Both of these schedulers can be implemented in $O(\log(n))$ time per packet arrival/departure (FQ using the method of [21]).

The above method should work, provided the membership in the enhanced and un-enhanced flow sets remains constant. However, their membership depends on unallocated bandwidth. An increase in unallocated bandwidth enhances more flows, and a decrease un-enhances some flows.

Unallocated bandwidth comes from two sources: from bandwidth that is not reserved by any flow, and from flows that have temporarily stopped creating packets (empty queues). The former is relatively stable, and changes are predictable (when flows are added or removed). In this case, the appropriate movement of flows between the schedulers can be done before a new flow is accepted or removed from the system. The latter, i.e., queues becoming empty, cannot be predicted, and may cause large changes in flow assignments to the two schedulers. This is particularly true if the flows whose queue becomes empty have a large reserved bandwidth. Thus, moving flows from one scheduler to the other is not efficient, which prompts us to present below our final version of the scheduler.

A-PEQ Scheduler

upon receiving a packet $p_{f,i}$,
    if the queue of f, Q_f, is empty, then
        let $\rho_{min} = \min\{\rho_g \mid Q_g \neq \emptyset\}$;
        $\rho_f = max(\rho_f, \rho_{min})$;
    add $p_{f,i}$ to Q_f;

if output channel is idle at time t,
    model the behavior of PCR^* to dequeue a packet;
    let $p_{f,i}$ be the packet chosen by PCR^*;
    let $\rho_{min} = \min\{\rho_g \mid Q_g \neq \emptyset\}$;
    if $Q_f \neq \emptyset$ then
        dequeue and forward a packet from flow f;
        $\rho_f = \min(\rho_f + 1, \rho_{min} + \Delta)$;
    else
        let g satisfy $\rho_g = \rho_{min}$;
        forward the next packet of flow g;
        $\rho_g = \rho_g + 1$;

Fig. 8. Approximate packetized rate equalization scheduler

B. Static-Flow-Assignment Scheduler

Our final protocol, *Approximate Packetized rate Equalization* (A-PEQ), is shown in detail in Figure 8, with an abstract view in Figure 6(b). For this implementation, we make the simplifying assumption that all packets of all flows have an equal size, L.[3] There are three major differences from the previous scheduler.

First, *all flows take part in both schedulers*. This solves the problem of having to move a large number of flows between the schedulers in a short period of time.

Second, the scheduler PCR* differs from PCR as follows. PCR* assumes that every flow always has packets available (even if its queue is empty). When it chooses a packet from flow f for transmission, it checks the queue of f. If it is empty, then the packet transmitted is instead a packet chosen by Δ-RR. Even though f did not transmit a packet, PCR* updates its state about f as if indeed it had transmitted a packet from f.

Third, instead of FQ, we have a modified round-robin scheduling, which we denote Δ-RR. Each flow f has a round number ρ_f in Δ-RR. When Δ-RR is asked to forward a packet, it chooses it as follows.

- If Δ-RR is called because PCR* is unable to transmit a packet, then Δ-RR chooses a packet from the backlogged flow *with the least round-number*, and increases the flow's round number by one.
- If PCR* is able to transmit a packet from a flow f, then the round-number of f is increased by one, *even though* Δ-RR *did not output a packet*.

The motivation for the above choices is as follows. Consider two flows f and g, where f has a large reserved rate (always un-enhanced in the fluid server) and g a small reserved rate (always enhanced in the fluid server). All un-enhanced flows, such as f, transmit packets from PCR* at a high rate, so their round numbers in Δ-RR are higher than those of other flows. The slower flows, such as g, are served in round-robin order, and thus receive the same bandwidth.

Consider now two slow flows g and h, with g having a greater reserved rate than h. Note that through their respective packet transmissions at PCR*, the round number of g grows faster than h's. Nonetheless, both flows receive about the same behavior from Δ-RR. This is because the unused bandwidth at PCR* causes Δ-RR to serve the slowest flows, such as h, first, which allows these flows to reach the same round numbers as other flows, such as g.

One final detail remains. Assume the round number of flow f, due to its large reserved rate, grows much larger than that of other flows. Then, assume enough bandwidth becomes available to make f an enhanced flow in the fluid server. However, due its large round number, f will not receive service in Δ-RR for a long time. To avoid this, we place a bound, Δ, on the difference between the round number of any flow and the minimum round number of any backlogged flow, as indicated in Figure 8.

The bound Δ is a tunable parameter of the system. If it is too large, enhanced flows may not receive their due bandwidth, and if it is too small, bandwidth may be wasted on un-enhanced flows.

VII. Performance Bounds

In this section, we briefly outline some of the upper bounds on the performance of the A-PEQ scheduler. More in-depth discussions and the proofs may be found in [15].

We first note that A-PEQ is a rate-guaranteed scheduler. However it's upper bound on the exit time is slightly greater than that of Relation (2), as follows.

Theorem 1: For every packet $p_{f,i}$ in the A-PEQ scheduler,

$$E_{f,i,\text{A-PQE}} \leq F_{f,i,CR} + L/R_f$$

■

The reason for the term $\frac{L}{R_f}$, as opposed to the smaller term $\frac{L}{C}$, comes from the PCR* scheduler, due to the following. PCR* chooses f, and if it finds f's queue, then control is passed to Δ-RR, but at this very moment a packet from f arrives. Thus, the packet has missed its scheduling opportunity in PCR*.

Next, the complexity of A-PEQ is as desired.

Theorem 2: Let n be the number of input flows to an A-PEQ scheduler. Then, the time complexity of processing a received packet and the time complexity of selecting a packet for transmission is packet $O(\log(n))$. ■

For PCR*, an $O(\log(n))$ time implementation is possible using well-known techniques, such as maintaining only one finishing time, F_f, for each flow f, as opposed to maintaining one value per packet. Also, maintaining a queue of flows that will become active at some time t can be done using the methods discussed in [2]. Implementing Δ-RR is obviously

[3] We will investigate eliminating this restriction in future work.

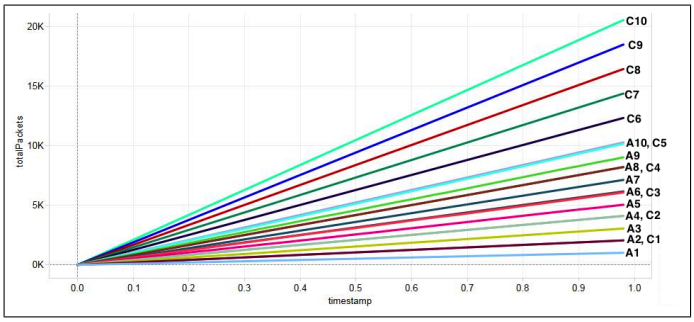


Fig. 9. Scenario UA for WFQ: Flows in categories *A* and *C* only.

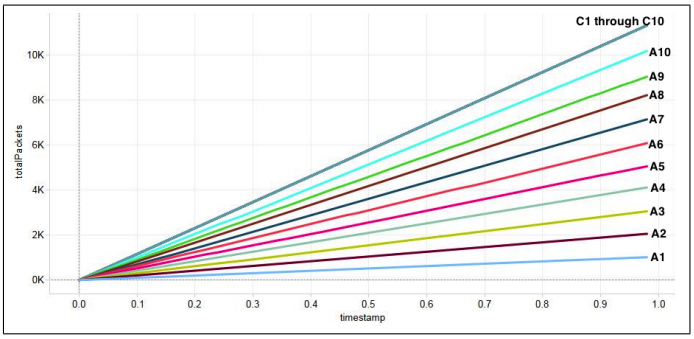


Fig. 10. Scenario UA for PEQ: Flows in categories *A* and *C* only.

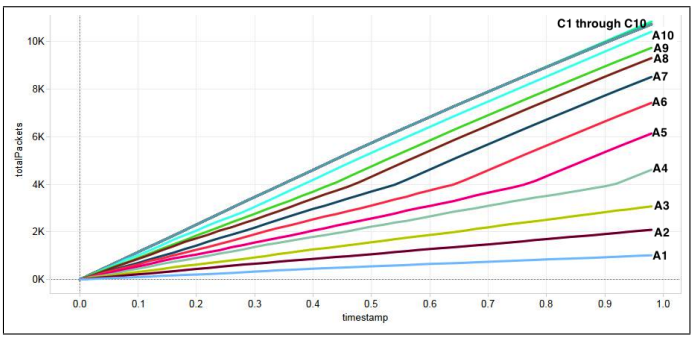


Fig. 11. Scenario UA for A-PEQ: Flows in categories *A* and *C* only.

$O(\log(n))$, since it only needs to maintain the smallest round number among the backlogged flows.

Finally, note that, contrary to schedulers like WFQ and PEQ, A-PEQ does not simulate the behavior of the virtual fluid server. Thus, it is difficult, if not impossible, to provide an upper bound on the difference in the exit time of a packet from the EQ fluid server and the A-PEQ packet scheduler. This is why we provide simulations in Section VIII, which evaluate how closely A-PEQ approximates our initial $O(n)$ packet scheduler, PEQ. Nonetheless, to argue that A-PEQ does provide the desired fairness, we have the following.

Theorem 3: Assume that starting from a time t, for each flow in an A-PEQ scheduler, either its queue is always empty or always non-empty. Let $b_1, b_2, \ldots, b_m$ be the set of backlogged flows. Let j be as defined in Figure 8. Hence, flows $b_1, \ldots, b_j$ are permanently enhanced in the fluid server, while flows $b_{j+1}, \ldots, b_m$ are permanently un-enhanced. Let $P(t_1, t_2, b_k)$ be the number of bits from flow b_k transmitted by the A-PEQ scheduler during time interval $[t_1, t_2]$. Finally, let

$$\Delta > \frac{R_{max}}{R_{min}}$$

where R_{max} and R_{min} are the maximum and minimum reserved rates among the backlogged flows. Then,

- For every k, $1 \leq k \leq j$, as t' increases, $P(t, t', b_k)$ converges to R_{EQ}, where R_{EQ} is as defined in Figure 3.
- For every k, $j+1 \leq k \leq m$, as t' increases, $P(t, t', b_k)$ converges to R_{b_k}.

■

VIII. Simulation Results

We simulate the behavior of a single scheduling node, with 30 input flows divided into three groups of ten flows each. In each group, the rate reserved by the flows vary from 1 Mbyte/sec. up to 10 Mbyte/sec. in increments of 1 Mbyte/sec.. Hence, the total rate reserved by the flows is $3 \cdot \sum_{x=1}^{10} x = 3 \cdot 55 = 165$ Mbytes/sec.. The three groups are as follows:

- Flows A_1 through A_{10} generate packets at their reserved rates. E.g., flow A_1 generates packets at 1 Mbyte/sec., flow A_2 gererates packets at rate of 2 Mbytes/sec., and flow A_{10} generates packets at a rate of 10 Mbytes/sec..
- Flows B_1 through B_{10} generate packets at *half* their reserved rate. E.g., flow B_1 generates packets at a rate of 1/2 Mbyte/sec., and flow B_{10} generates packets at a rate of 5 Mbytes/sec..
- Flows C_1 through C_{10} correspond to group *C*. These are greedy flows, whose queues never become empty. Note that only flows in *C* can take advantage of unused bandwidth, since flows in *A* and *B* create packets at the same rate (or lower) than their reserved rate.

There are two sources of unused bandwidth in a scheduler: unallocated (UA) bandwidth, and allocated but unused (UU) bandwidth. Thus, we simulate three general cases: (a) UA, which is achieved by including only flows in groups *A* and *C* and thus leaves 1/3 of the channel bandwidth unallocated, (b) UU, which is achieved by including all three flow groups and leaves 1/6 of the bandwidth unused (B only uses half of its reserved bandwith), and (c) UU + UA, which is achieved by including only flows B and C and leaves 1/3 + 1/6 of the channel bandwidth unallocated or unused.

For groups *A* and *B*, we simulate either a constant packet arrival rate or a poisson arrival rate. We thus have a total of six different scenarios. However, both poisson and constant rate yielded very similar plots. Due to space limitations we will show only the poisson arrival rate.

Figures 9, 10, and 11 show the UA scenario, i.e., the input to the scheduler is only flows in categories *A* and *C*. The figure plots the total number of packets transmitted over time. Due to the absence of flows from *B*, this leaves 55 Mbytes/sec. to be distributed among the greedy flows.

For WFQ (Figure 9), the flows from *A* reside at the bottom, since they do not take advantage of the bandwidth. The flows

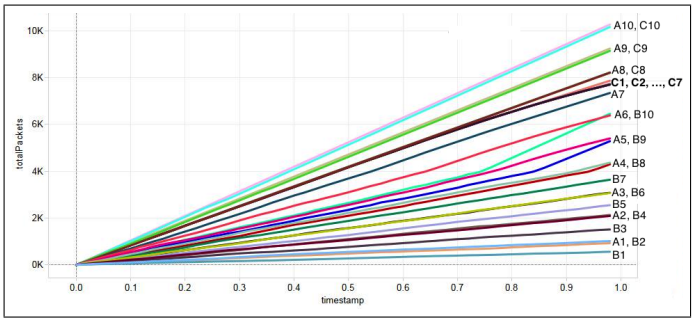


Fig. 12. Scenario UU for A-PEQ: Flows in all categories, *A*. *B* and *C*.

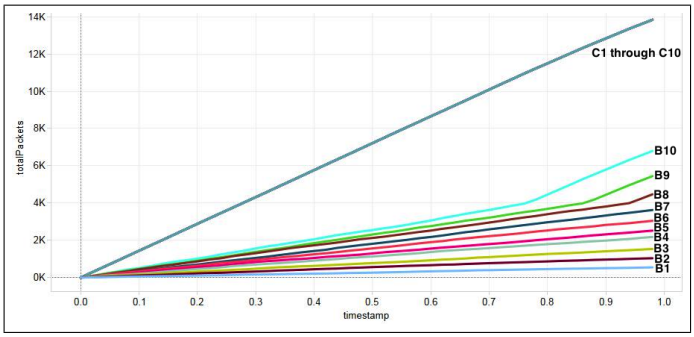


Fig. 13. Scenario UU+UA for A-PEQ: Flows in categories *B* and *C* only.

from *C* receive bandwidth in proportion to their reserved rates. E.g., flow C_5, whose reserved rate is 5 Mbytes/sec., receives only about half the bandwith than flow C_{10}, whose reserved rate is 10 Mbytes/sec..

For PEQ (Figure 10), the 55 Mbytes/sec. are distributed over *all* flows in C, giving each of them an effective bandwidth of about 11 Mbytes/sec., as shown in the figure (only flow 11 is visible since all lines of flows 11 through 20 overlap). For A-PEQ (Figure 11), a similar behavior is shown. Thus, A-PEQ approximates PEQ quite closely.

Due to lack of space, for cases UU and UU+UA we only show the plots for A-PEQ since they are similar to those of PEQ. Case UU is shown in Figure 12, containing all flows in A, B, and C. The unused bandwidth by B is about 27.5 Mbytes/sec. This is enough to make flows C_1 through C_7 have the same bandwith, about 7.5 Mbytes/sec., but no additional bandwidth is left over for flows C_8 through C_9.

Finally, case UU+UA is shown in Figure 13. Here, we only have flows from B (half-rate) and C (greedy). Flows in B, consume only a total of 27 Mbytes/sec., leaving 138 Mbytes/sec. for flows in C. I.e., each flow in C is forward at a rate of about 14 Mbytes/sec., as desired.

IX. Future Work and Concluding Remarks

The first item that needs to be addressed in future work is to relax the restriction that all flows must have the same packet size. This affects the Δ-RR scheduler, but leaves the PCR* scheduler as is. Another item that needs to be addressed are bounds on the value of Δ. A value as chosen for Theorem 3 appears appropriate, the effects of different values for Δ must be studied in detail.

Finally, scheduling packets over multiple parallel links between neighboring computers has been studied for guaranteed-rate schedulers [8] [3]. We also plan to investigate the impact of rate-equalization over parallel links.

References

[1] J. C.R. Bennett and H. Zhang. Hierarchical packet fair queueing algorithms. *IEEE/ACM Tran. on Networking*, 5(5):675–689, Oct. 1997.

[2] Jon C.R. Bennett and Hui Zhang. WF2Q: worst-case fair weighted fair queueing. In *IEEE INFOCOM Conference*, 1996.

[3] J.M. Blanquer and B. Ozden. Fair queuing for aggregated multiple links. In *Proc. of the ACM SIGCOMM Conference*, 2001.

[4] B. Caprita, J. Nieh, and W. C. Chan. Group round robin: Improving the fairness and complexity of packet scheduling. In *Proc. Symp. on Architecture for Netw. Commun. Syst.*, page 2940, October 2005.

[5] Jorge Cobb. Universal timestamp scheduling for real-time networks. *Computer Networks*, 31:2341–2360, 1999. Elsevier.

[6] Jorge Cobb and Mohamed Gouda. Flow theory. *IEEE/ACM Transactions on Networking*, 5(5):661–674, October 1997.

[7] Jorge Cobb, Mohamed Gouda, and Amal-El Nahas. Time-shift scheduling: Fair scheduling of flows in high-speed networks. *IEEE/ACM Transactions on Networking*, 6(3):274–285, June 1998.

[8] Jorge Cobb and Miaohua Lin. End-to-end delay guarantees for multiple-channel schedulers. In *Proc. IEEE Int. Workshop on QoS*, May 2002. T*Journal of High-Speed Networks*, 12(1 and 2):61–86, November 2002.

[9] Jorge A. Cobb. Rate equalization: A new approach to fairness in deterministic quality of service. *37th Annual IEEE Conference on Local Computer Networks*, 0:50–57, 2011.

[10] N. Figueira and J. Pasquale. Leave-in-time: A new service discipline for real-time communications in a packet-switching data network. In *Proc. of the ACM SIGCOMM Conference*, 1995.

[11] S.J. Golestani. A self-clocked fair queueing scheme for broadband applications. In *IEEE INFOCOM Conference*, 1994.

[12] P. Goyal, S. Lam, and H. Vin. Determining end-to-end delay bounds in heterogeneous networks. In *Proc.of the NOSSDAV Workshop*, 1995.

[13] C. Guo. Srr: an o(1) time-complexity packet scheduler for flows in multiservice packet networks. *IEEE/ACM ToN*, 12(6), Dec. 2004.

[14] C. Guo. G-3: An o(1) time complexity packet scheduler that provides bounded end-to-end delay. In *Proc. of the IEEE INFOCOM Conf.*, 2007.

[15] Suparn Gupta. An approximation to rate-equalization fairness with logarithmic complexity for qos. Master's thesis, The University of Texas at Dallas, expected Spring, 2014.

[16] Partho P. Mishra and Hemant Kanakia. A hop by hop rate-based congestion control scheme. *ACM SIGCOMM*, October 1992.

[17] A. K. J. Parekh and R. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single node case. *IEEE/ACM Transactions on Networking*, 1(3), June 1993.

[18] S. Ramabhadran and J. Pasquale. Stratified round robin: A low complexity packet scheduler with bandwidth fairness and bounded delay. *ACM Comput. Commun. Review*, 33(4):239250, October 2003.

[19] D. Stidialis and A. Varma. Rate proportional servers: A design methodology for fair queuing algorithms. *IEEE/ACM Transactions on Networking*, April 1998.

[20] Subhash Suri, George Varghese, and Girish Chandranmenon. Leap forward virtual clock: A new fair queuing scheme with guaranteed delays and throughput fairness. In *In Proceedings of INFOCOM'97*, 1997.

[21] Paolo Valente. Exact gps simulation with logarithmic complexity, and its application to an optimally fair scheduler. *ACM SIGCOMM*, 2004.

[22] G. Xie and S. Lam. Delay guarantee of the virtual clock server. *IEEE/ACM Trans. on Net.*, pages 683–689, December 1995.

[23] X. Yuan and Z. Duan. Frr: A proportional and worst-case fair round robin scheduler. In *Proc. IEEE INFOCOM*, page 831842, March 2005.

[24] L. Zhang. Virtual clock: A new traffic control algorithm for packet-switched networks. *ACM Trans. on Comp. Syst.*, 9(2), May 1991.

[25] Shin K.G. Zheng Q. On the ability of establishing real-time channels in point-to-point packet-switched networks. *IEEE Transactions on Communications*, 42(2-4), 1994.

[26] L. Zhong, J. Xu, and X. Wang. Vwqgrr: A novel packet scheduler. In *Proc. of the IEEE ICN*, page 2228, April 2007.