AN APPROXIMATION TO RATE-EQUALIZATION
FAIRNESS WITH LOGARITHMIC COMPLEXITY

By

Suparn Gupta

APPROVED BY SUPERVISORY COMMITTEE:

Cong Liu

Jorge Arturo Cobb

Kamil Sarac

AN APPROXIMATION TO RATE-EQUALIZATION FAIRNESS

WITH LOGARITHMIC COMPLEXITY

by

SUPARN GUPTA, MS

THESIS

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

May 2014

ACKNOWLEDGMENTS

There are a number of people without whom, I would have never been able to complete this thesis and be at a point where I stand at present.

Firstly, I would like to express my deepest gratitude towards my thesis advisor, Dr. Jorge A. Cobb who has the substance of genius. His persistent guidance and advice made this journey a smooth ride with a lot of learning. Without his involvement, this thesis would have never been possible.

In addition, I am highly indebted to my approval committee, Dr. Cong Liu and Dr. Kamil Sarac for giving their valuable time, advice and criticism to my thesis. Their suggestions have been an important contribution.

Lastly, I would like to thank my mother Dr. Urmil Gupta for her never ending encouragement!

AN APPROXIMATION TO RATE-EQUALIZATION FAIRNESS WITH LOGARITHMIC COMPLEXITY

Suparn Gupta, Masters

The University of Texas at Dallas, 2014

Supervising Professor: Jorge Arturo Cobb

ABSTRACT

Scheduling protocols that provide both rate and fairness guarantees, such as Weighted Fair Queuing (WFQ) distributes the unused bandwith among flows in proportion to their reserved rate. Flows with higher reservation rate thus receive a larger share of bandwith than those with lesser reserved rates.

In an earlier work [9], a new approach to allocate unused bandwidth known as rate-equalization (REQ) was proposed, in which the unused bandwidth is first given to the flows with least reserved rates. However, this algorithm has a per-packet complexity of *O(n)* where *n* is the number of flows. The present study proposes a new algorithm, which is an approximation of REQ fairness with logarithmic per-packet complexity. The simulation results endorse than in different practical scenarios, the behavior of the new algorithm is quite similar to that of pure REQ fairness.

CONTENTS

ACKNOWLEDGMENTS iv

ABSTRACT v

List of Figures viii

List of Tables x

Chapter 1. Introduction 1

1.1 Scheduling Protocols 2

1.1.1 Weighted Fair Queuing (WFQ) 3

1.1.2 Worst-case Fair Weighted Fair Queueing (WF^2Q) 5

1.1.3 Virtual Clock (VC) 6

Chapter 2. QoS Model 8

2.1 Rate Equalization 12

2.2 Rate-Equalization Server and Scheduler 14

2.2.1 Fluid Server 14

Chapter 3. Logarithmic Rate Equalization Scheduler 17

3.1 Packet Scheduler 17

3.2 Roadblocks to Efficient Implementation 19

3.3 Dual-Mode Scheduling 20

3.3.1 Flow-Migration Scheduler 21

3.3.2 Static-Flow-Assignment Scheduler 23

3.4 Performance Bounds 25

Chapter 4. Simulations and Results 27

4.1 Simulation Environment 28

4.1.1 Overview of scenarios 29

4.2 Simulation Results 30

4.2.1 UA, Poisson 30

4.2.2 UA, Constant Rate 34

4.2.3 UU, Poisson 38

4.2.4 UU, Constant Rate 42

4.2.5 UA + UU, Poisson 46

4.2.6 UA + UU, Constant Rate 50

Chapter 5. Conclusions and Future work 54

Bibliography 55

LIST OF FIGURES

2.0.1 Schedulers and Channels 9

2.0.2 Constant-rate fluid server 11

2.2.2 Fluid server example. 15

2.2.1 Rate equalization fluid server. 15

3.1.1 Packetized rate equalization scheduler 19

3.3.1 Dual-Mode Scheduling 21

3.3.2 Packetized constant-rate scheduler 21

3.3.3 Approximate packetized rate equalization scheduler 23

4.2.1 UA, Poisson, WFQ 30

4.2.2 UA, Poisson, EQ 31

4.2.3 UA, Poisson, PEQ 32

4.2.4 UA, Poisson, A-PEQ 33

4.2.5 UA, CR, WFQ 34

4.2.6 UA, CR, EQ 35

4.2.7 UA, CR, PEQ 36

4.2.8 UA, CR, A-PEQ 37

4.2.9 UU, Poisson, WFQ 38

4.2.10 UU, Poisson, EQ 39

4.2.11 UU, Poisson, PEQ 40

4.2.12 UU, Poisson, A-PEQ 41

4.2.13 UU, CR, WFQ 42
4.2.14 UU, CR, EQ 43
4.2.15 UU, CR, PEQ 44
4.2.16 UU, CR, A-PEQ 45
4.2.17 UA + UU, Poisson, WFQ. 46
4.2.18 UA + UU, Poisson, EQ 47
4.2.19 UA + UU, Poisson, PEQ 48
4.2.20 UA + UU, Poisson, A-PEQ 49
4.2.21 UA + UU, CR, WFQ 50
4.2.22 UA + UU, CR, EQ 51
4.2.23 UA + UU, CR, PEQ 52
4.2.24 UA + UU, CR, A-PEQ 53

LIST OF TABLES

1 Flow categories in simulation 28

CHAPTER 1

INTRODUCTION

In a network of various interconnected nodes which provides QoS, a certain amount of bandwidth is reserved for each flow. A scheduling protocol is implemented to make decisions about the packet transmission from each backlogged flow. Selection of such protocol is a crucial decision because of the complexity and execution cost. Any such scheduling protocol can be analyzed on the basis of different kinds of guarantees it provides. Two such metrics of analysis are rate and fairness guarantees.

Rate Guarantee: A flow requests for a certain amount of bandwidth (rate) in the network before it begins transmitting packets. If the network has enough bandwidth available, it grants the reservation request. Otherwise, the request is denied. A scheduling protocol which provides rate guarantee assures that a flow gets atleast its reserved bandwidth at all times while it is active. A flow may receive more bandwidth than what it originally requested for, if bandwidth gets available. Such a guarantee thus ensures that any flow gets atleast a certain amount of service in the scheduler.

Fairness: If the network is not congested, it might be possible that a part of bandwidth is unused. It may happen for two reasons:

(1) There is some unallocated bandwidth available in the network.

(2) A set of flows are not fully utilizing their share of allocated bandwidth by transmitting packets at a slower rate than their service rate.

To use the resources efficiently, a scheduler thus distributes the unused bandwidth among the flows fairly while maintaining firewalls between them so that the allocation is followed strictly by all the flows. Fairness means a flow should not be “punished” (temporarily denied service) if it exceeds

its reserved rate to take advantage of unused bandwidth in the channel. The way in which each protocol distributes the unused bandwidth is different.

If a flow gets empty, the scheduler starts serving the packets from the next non empty flow. Such a scheduler will be work-conserving since it doesn't let resources go to waste. In certain scenarios, a scheduler may decide not to distribute the unused bandwidth. Such a scheduler is non-work conserving. In the next few sections, a number of well known scheduling protocols are described.

1.1 SCHEDULING PROTOCOLS

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) [21], [23] and Weighted Fair Queuing (WFQ) [16], plus their many variations. One desirable property of a rate-guaranteed scheduler is fairness. 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 [15]. 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. Speaking generally of scheduling protocols, rate guarantee and fairness can be considered as two extreme ends between which various rate guaranteed scheduling algorithms fit in.

Prior to begining the discussion about scheduling protocols, it is important to establish a formal notion of a flow. A flow :

(1) is a stream of packets where each packet follows the same path in the network.

(2) avails a certain minimum amount of service in the network.

Each packet is uniquely bound to a particular flow which can be identified by analysing the headers in the packet. At any given time t, if a flow has packets in its queue, it is known as a *backlogged* flow. Otherwise, a flow with empty queue at any given time t is *idle*. Consider a set of all the flows represented by W. Then at any given time t, the set of backlogged flows given by B satisfies $B \subseteq W$.

In the following sections, consider an output channel with a capacity C and a set of flows W in the fluid server such that each flow is represented by $f_i \, \varepsilon \, W$, *where* $1 \leq i \leq n$. Let B represent the set of backlogged flows in the fluid server.

1.1.1 Weighted Fair Queuing (WFQ)

A Generalized Processor Sharing (GPS) discipline provides rate and fairness guarantees. However, since it models a fluid server, it assumes that the scheduler can serve all the flows. However, a real scehduler has to transmit a packet from one flow at any given time and thus, it is impossible to practically realize a GPS system. Several packetized approximations which follow the footsteps of GPS have been proposed in past many years. Out of such known algorithms, Weighted Fair Queuing is well known to provide both rate and fairness guarantees. It assures that *(a)* each flow gets atleast the amount of bandwidth it reserved and *(b)* the unused bandwith is allocated among the flows in proportion of their reserved rates. Thus the flows with higher reserved rates get the larger share of unused bandwidth than those with smaller reserved rates.

Each flow $f_i \, \varepsilon \, B$ is assigned a weight w_i and gets the bandwidth c_i allocated as per the following rule:

$$c_i = \frac{w_i}{\sum_x w_x}, \quad f_x \, \varepsilon \, B \tag{1.1.1}$$

There are two schedulers (servers) in the system:

(1) Fake or bit-by-bit scheduler, which forwards a few bits of each flow at a time (i.e. fractions of a packet)

(2) Real scheduler, which

- assigns timestamps to packets.
- sends out packets in non-decreasing order of timestamp.

The timestamp is the "virtual" finishing time of the packet at the fake server. The virtual time *V(t)* at real time t is the "bit number" or "round-number" in the fake bit-by-bit server at real time *t*. In case of fair queuing, *V(t)* is increased by 1 every time one bit is forwarded from all the flows in the fake server. In case of WFQ, *V(t)* is increased by 1 every time when for each flow $f_i \; \varepsilon \; B$, w_i bits are forwarded from all the flows in the fake server. If there are fewer number of flows in the server, the *virtual time* will run faster. Also, for each tick in *virtual* time, the number of bits forwarded for flows with higher weights is more than those with lower weights.

Let $F_{f,i}$ be the virtual time when the *i-th* packet of flow f exits the bit-by-bit server, $A_{f,i}$ be the virtual time when the *i-th* packet of flow f exits the bit-by-bit server. Then following relation can be established.

$$F_{f,i} = max(V(A_{f,i}),\ F_{f,i-1}) + \frac{L_{f,i}}{w_f} \tag{1.1.2}$$

where $L_{f,i}$ is the length of *i-th* packet of flow f

In [16], it has been proven that if the rate admission control is maintained using the leaky bucket, end to end delay guarantee can be achieved. A packet gets out of the real server by the time it gets out of the fake server plus a constant. Thus,

$$Finishing\ time = F_{f,i} + \frac{L_{max}}{C} \tag{1.1.3}$$

where L_{max} is the maximum length of the packet.

It was long known that the WFQ can be implemented with a per-packet complexity of *O(n)*, until Valente [20] proposed a way to implement WFQ in *O(log(n))* per-packet complexity. He uses a complex scheme to organize the breaking points in a search tree to optimize searches.

1.1.2 Worst-case Fair Weighted Fair Queueing (WF^2Q)

In [16], Parekh stated a number of relationships between a GPS and a packetized approximation of GPS, which he called as PGPS (identical to WFQ). Two main claims were as follows:

(1) A packet gets out of a PGPS scheduler no later than it gets out of GPS scheduler with a delay of atmost one packet transmission time.

(2) PGPS cannot lag behind the GPS scheduler by more than one packet transmission.

From above two claims, one may deduce that a PGPS scheduler offers a service which is almost identical to a GPS scheduler with a variation of atmost one packet transmission time, which was once a popular belief until Bennet et al.[2] proved that there can be significant discrepances between a PGPS and a GPS scheduler. They claim that the PGPS scheduler can be far ahead of GPS in terms of service given to a session leading to an unfair behavior of the protocol towards other sessions.

In PGPS or WFQ approximation of GPS, packets are transmitted according to the order of their timestamps. It might happen that a packet may get transmitted in the PGPS before it exits the GPS scheduler. In this case, the next packet of the session might start receiving service even before the first packet finishes in GPS. This may lead WFQ to run ahead of GPS thus favoring a particular flow. In [2], a modified policy for selecting packets for transmission has been proposed which is called WF^2Q. According to this policy, when a WFQ scheduler selects the next packet for transmission, it choses a packet among those, which have already started receiving service in corresponding GPS system. Also, it has been established that with WF^2Q policy, the system offers service almost identical to GPS with a maximum difference of one packet transmission. Also, WF^2Q scheduler is proven to be work conserving.

1.1.3 Virtual Clock (VC)

Another packet service discipline has been proposed in by Zhang [23] which is known as Virtual Clock algorithm. The basic concept of this algorithm is somewhat similar to Time Division Multiplexing (TDM), which maintains a perfect firewall among various sessions. In TDM, each session can transmit data only in a particular time slot. However, this turns out to be inefficient on resource utilization as a time slot may get wasted if the session doesn't have any packet to transmit. Virtual Clock preserves the firewall property of a TDM and ensures work conserving nature of the scheduler.

In VC algorithm, two clocks are implemented, a real clock and a virtual clock. Whenever, a packet is received at the queuing node, it is timestamped with a virtual clock value. The packets are served in non-decreasing order of virtual clock timestamps of each packet. Suppose that a session f has a reserved bandwidth of R_f. Then for each such flow a value known as $Vtick_f$ [23]is calculated according to the following relationship:

$$Vtick_f = \frac{1}{R_f} \tag{1.1.4}$$

Now consider the *i-th* packet of flow f represented by $p_{f,i}$. Using 1.1.4, the Virtual Clock value $VC_{f,i}$ for $p_{f,i}$ can be calculated using the following equation:

$$VC_{f,i} = VC_{f,i-1} + Vtick_f \tag{1.1.5}$$

One important point to note here is that, flows which generate packets at a slower rate may accumulate credits over the time and thus create unfairness in the system towards other flows. This is taken care of by periodically synchronizing the Virtual Clock values with the real time so that VC values don't fall far behind of real time. In[21], it has been proven that Virtual Clock Algorithm has delay bound which is identical to that of WFQ. However, VC is not a fair scheduling policy.

Flows which transmit data at a speed higher than their reserved rates may get extra bandwidth for sometime if its available. But in later time, since their VC values will be far ahead of slower flows, they will get punished. This behavior has been modified in a scheme proposed in [19]. This scheme gives an end-to-end delay guarantee similar to WFQ and also exhibits throughput fairness.

CHAPTER 2

QOS MODEL

In all the previously discussed algorithms, the rate guarantee is always ensured by providing atleast the reserved rate of service to a session. The way fairness is ensured varies with different protocols. Algorithms, which follow the GPS discipline exhibit fairness by distributing the unused bandwidth in proportion to the reserved rates of flows. Algorithms like Leap Forward Virtual Clock [19] maintain fairness by moving misbehaved sessions to a separate queue while keeping the well-behaved flows together.

The QoS model presented here is based on the models of [6] and [10]. A *network* is a set of computers connected via point-to-point communication channels. A flow is a sequence of packets, all of which originate at the same source computer and are destined to the same computer. All packets of a flow must traverse the network along a fixed path.

Each flow is characterized by its data rate and an upper bound on its end-to-end delay. Before a source introduces a new flow to the network, enough network resources are reserved to ensure the flow's delay bound is not violated. If enough resources are not available, the flow is rejected.

Each output channel of a computer is equipped with a packet scheduler (see Figure 2.0.1). 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.

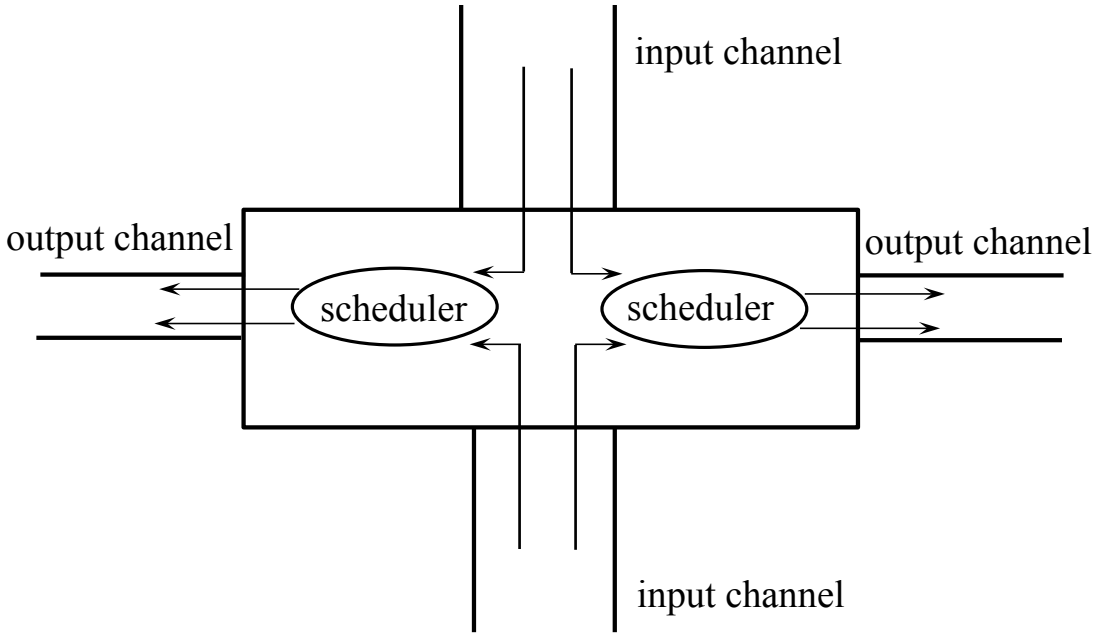


FIGURE 2.0.1. Schedulers and Channels

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_f^{max} maximum packet length of f.

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.

[1]Fluid servers can forward, during any time interval, an arbitray 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.

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

FIGURE 2.0.2. Constant-rate fluid server

Let us 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 $S_{f,1,CR} = A_{f,1}$.

$$\begin{aligned} S_{f,i,CR} &= \max(A_{f,i},\ F_{f,i-1,CR}) \quad i > 1 \\ F_{f,i,CR} &= S_{f,i,CR} + \frac{L_{f,i}}{R_f} \quad i \geq 1 \end{aligned} \tag{2.0.6}$$

Note that $F_{f,i,CR}$ is the value used by the Virtual Clock scheduling protocol to assign priorities to its packets [21, 23].

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., $F_{f,i,CR}$. These packet schedulers are referred to 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, $i \geq 1$,

$$E_{f,i,s} \leq F_{f,i,CR} + \delta_{f,s} \tag{2.0.7}$$

for some constant $\delta_{f,s}$. For many protocols, such as Virtual Clock [21, 23] and Weighted Fair Queuing [16] (and their many variations),

$$\delta_{f,s} = L_s^{max}/C_s \tag{2.0.8}$$

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.

2.1 RATE EQUALIZATION

In this section, the Rate-Equalization fairness which was presented in [9] is discussed. 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 that is not being used by other flows. The source of the flow can determine if there is unused bandwidth by receiving implicit feedback from the network, in particular, that its packets are experiencing an end-to-end delay that is smaller than expected.

The manner in which unallocated bandwidth is distributed among flows varies from one scheduling 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) [23][21], do not address fairness. In VC, each packet $p_{f,i}$ is labeled with its virtual finishing time, $F_{f,i,CR}$ (see (2.0.6)), 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 [8]. This is potentially unbounded. The exit bound in Equation (2.0.7) still holds, however, because VC is a rate-guaranteed scheduler.

Other rate-guaranteed protocols, such as Weighted Fair Queuing (WFQ) [16] and its variants [1] [8] [11] [2], distribute the unallocated bandwidth among flows in proportion to their reserved rate. Specifically, the effective rate ψ_f that is given to flow f (i.e., the rate at which the scheduler

[2]Other protocols have a *rate-independent delay* [10, 24]: 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.

actually forwards the packets of flow f) is

$$\psi_f(t) = \frac{C}{\left(\sum_{g \in B(t)} R_g\right)} \cdot R_f \geq R_f \tag{2.1.1}$$

where $B(t)$ are the backlogged flows at time t (i.e., those flows with a non-empty queue) and C is the capacity of the output channel of the scheduler.

Consider another flow g with lesser reserved bandwidth, e.g., $R_f = 2 \cdot R_g$. From (2.1.1), it can be easily shown that

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

That is, the manner in which unused bandwidth is allocated to backlogged flows is proportional to the reserved rate of the flow. 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 user who generates 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 presented an alternative fairness method in which the overall objective is to give every flow the same effective rate, provided enough unallocated bandwidth is available. This will result in a value for the effective rate ψ_f that is different from the one given in Equation (2.1.1). The intuition behind it is the following. Every flow will be guaranteed its reserved rate R_f. However, 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 those flows that need it the most, i.e., those with the least reserved rate.

A more detaied 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 by introducing a rate equalization fluid server, which will then be emulated as close as possible by a packet scheduler.

2.2 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.

2.2.1 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 2.2.1. 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 2.2.1 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

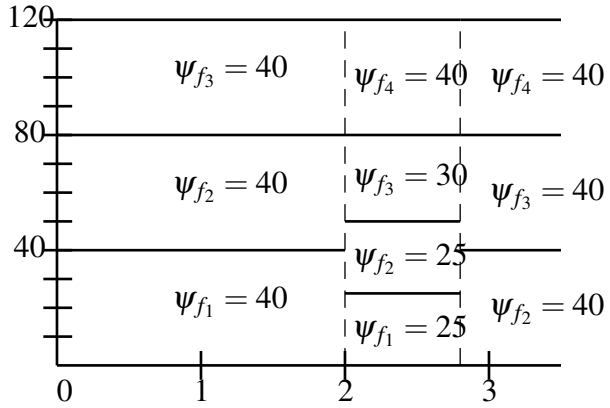


FIGURE 2.2.2. Fluid server example.

rate of zero. All other flows are then arranged in increasing order of their reserved rate. Then, the index j is found such that

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

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

---

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**
      $R_{b_j} \leq \dfrac{C - \sum_{k=j+1}^{m} R_{b_k}}{j} < R_{b_{j+1}}$**;**
    let $R_{EQ} = \dfrac{C - \sum_{k=j+1}^{m} R_{b_k}}{j}$**;**
    for each k**,** $1 \leq k \leq j$**,**
      $\psi_{f,EQ}(t) = R_{EQ}$**;**
    for each k**,** $j+1 \leq k \leq m$**,**
      $\psi_{f,EQ}(t) = R_{b_k}$**;**

FIGURE 2.2.1. Rate equalization fluid server

Consider the following example, which is illustrated in Fig. 2.2.2. 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 $\psi_{f_1} = \psi_{f_2} = 25$ 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. In the similar fashion one can compute the remainder of the example.

CHAPTER 3

LOGARITHMIC RATE EQUALIZATION 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][16][18], 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.

3.1 PACKET SCHEDULER

For most fluid servers [5][16][18], 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 [16] 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

$$D_{f,i,PEQ} = S_{f,i,EQ} + \frac{L_{f,i}}{R_f}. \tag{3.1.1}$$

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 3.1. More detais can be found in [9], including bounds on the difference between a packet's exit time from the scheduler and from the fluid server.

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**$\{D_{f,i,PEQ} \mid p_{f,i} \in$ **Active**$(t)\}$**;**
forward $p_{g,j}$ **to the output channel.**

FIGURE 3.1.1. Packetized rate equalization scheduler

3.2 ROADBLOCKS TO 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 [19], Time-Shift Scheduling [8], 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][17][22][4][25]. All of these provided 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 [20]. 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.

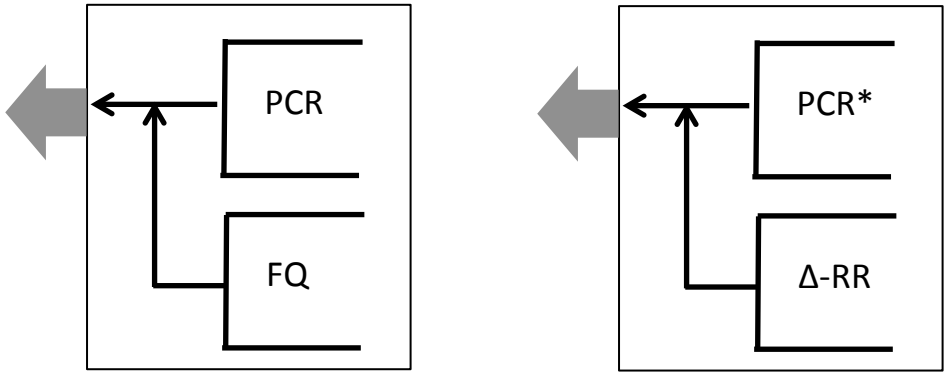
However, the ratio mentioned above is not constant in rate-equalization scheduling. In fact, it can vary significantly. To see this, consider again Figure 2.2.1. You can consider the set of backlogged flows to always be divided into two subsets: those flows whose effective rate is their reserved rate, 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 signnificantly 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 [20] 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.

3.3 DUAL-MODE SCHEDULING

From the above discussion, backlogged flows in the fluid server can be cosidered 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 *unenhanced flows*, whose effective rate is simply their reserved rate. This motivates our first packet scheduler design, presented in Section 3.3.1. Although intuitive, this first attempt is not efficient. We then present our final scheduler design in Section 3.3.2

3.3.1 Flow-Migration Scheduler



(a) (b)

FIGURE 3.3.1. Dual-Mode Scheduling

PCR Scheduler

Let $B(t)$ **be the set of backlogged flows at time** t upon receiving a packet $p_{f,i}$,
$D_{f,i,PCR} = F_{f,i,CR}$;

if output channel is idle at time t,
let $p_{f,i} \in \text{Active}(t)$ iff $t \geq S_{f,i,CR}$;
if $\text{Active}(t) \neq \emptyset$ 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.

FIGURE 3.3.2. Packetized constant-rate scheduler

Consider Figure 3.3.1(a). The service required for an unenhanced 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 3.3.1. It is similat to the Virtual Clock protocol [21], 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 3.3.1(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 [20]).

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

Unallocated bandwidth comes from two sources: from bandwidth that is not reserved by any flow, and from flows that have temporarilly 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.

3.3.2 Static-Flow-Assignment Scheduler

A-PEQ Scheduler

upon receiving a packet $p_{f,i}$**,**
  add $p_{f,i}$ **to the queue of** f**,** 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 $\rho_{min} = \mathbf{min}\{\rho_g \mid Q_g \neq \emptyset\}$**;**
  if $Q_f \neq \emptyset$ **then**
    dequeue and forward a packet from flow f**;**
    $\rho_f = \mathbf{min}(\rho_f + 1, \rho_{min} + \Delta)$**;**
  else
    let g **satisfy** $\rho_g = \rho_{min}$**;**
    forward the next packet of flow g**;**
    $\rho_g = \rho_g + 1$**;**

FIGURE 3.3.3. Approximate packetized rate equalization scheduler

Our final protocol, *Approximate Packetized rate Equalization* (A-PEQ), is shown in detail in Figure 3.3.2, with an abstract view in Figure 3.3.1(b). For this implementation, we make the simplifying assumption that all packets of all flows have an equal size, L.[1] 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. Eventhough f did not transmit a packet, PCR* updates its state about f as if indeed it had transmitted a packet from f.

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

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 unenhanced in the fluid server) and g a small reserved rate (always enhanced in the fluid server). All unenhanced 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 3.3.2.

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

3.4 PERFORMANCE BOUNDS

In this section, we outline some of the upper bounds on the performance of the A-PEQ scheduler. 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 1.1.3, as follows.

THEOREM 1. *For every packet $p_{f,i}$ in the A-PEQ scheduler,*

$$E_{f,i,A-PEQ} \leq F_{f,i,CR} + \frac{L}{R_f} \tag{3.4.1}$$

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 $O(log(n))$, 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 Chapter 4, 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 $b_1, b_2, ..., b_m$ be the set of backlogged flows. Let j be as defined in 3.3.2. Hence, flows $b_1,\ b_2,\ .....b_j$ are permanently enhanced in the fluid server, while flows $b_{j+1}, ..., b_m$ 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*

$$\Delta > \frac{R_{max}}{R_{min}} \tag{3.4.2}$$

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 REQ, where REQ is as defined in 2.2.1.
- For every k, $j+1 \leq k \leq m$, as t' increases, $P(t, t', b_k)$ converges to R_{b_k} .

CHAPTER 4

SIMULATIONS AND RESULTS

Our goal is to design a scheduler which provides fairness of EQ scheduler in logarithmic time. In the previous chapter, we looked at the details of A-PEQ scheduler. We proved that A-PEQ scheduler is a rate-guaranteed which assures ateast the reserved amount of bandwidth to each flow with a slightly higher value of upper bound. We claimed that it can be implemented in logarithmic time, since both PCR^* and $\Delta - RR$ can itself be implemented with logarithmic complexity using exiting techniques. We also gave a formal proof that A-PEQ scheduler provides the identical or similar fairness as EQ scheduler provided the state of flows doesn't change in the system.

In case of schedulers like WFQ, there exists a corresponding fluid server. If such cases, it is possible to compare the fairness of approximation algorithms with the fairness providedd by the ideal server. However, $A-PEQ$ doesn't emulate any such fluid server. To prove that despite of lack of a fluid server, $A-PEQ$ does offer fairness which is similar to EQ scheduler, we run simulations to compare the behavior of different schedulers. Fairness is contingent upon various factors like:

- Traffic patterns
- Number of flows in the system
- Reservations by each flow

There are other factors as well which might affect the behavior of the scheduling policy like packet drop probabilities, queue lengths etc. However, for the sake of simplicity we didn't consider them during simulations. To see the performance in possible practical scenarios, we run a number of simulations in which same traffic patterns were pushed through WFQ, EQ, PEQ and finally through $A-PEQ$. In the following section, we describe the simulation environment, traffic patterns and assumptions used in the simulator.

4.1 SIMULATION ENVIRONMENT

We simulate the behavior of a single scheduling node, with 30 input flows. The total output channel capacity of the node is 165 $Mbytes/sec.$. Each flow falls into exactly one of the following groups:

Group Name	Description
A	Generate packets at an average rate equal to their reserved rates
B	Generate packets at an average rate which is equal to half of their reserved rates
C	Generate packets at a very high rate such that their queues are always non-empty

TABLE 1. Flow categories in simulation

In all the simulation scenarios, each of the above mentioned groups is:

- either empty
- or non empty with exactly 10 flows

All the flow have their reserved rates in the scheduling node, which are described below:

- Flows A_1 through A_{10} correspond to group A and have a reserved rate of 1, 2, ..., 10 $Mbytes/sec.$ respectively. 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} correspond to group B and have a reserved rate of 1, 2, ..., 10 $Mbytes/sec.$ respectively. E.g., flow B_1 generates packets at 0.5 $Mbyte/sec.$, flow B_2 gererates packets at rate of 1 $Mbytes/sec.$, and flow B_{10} generates packets at a rate of 5 $Mbytes/sec.$.
- Flows C_1 through C_{10} correspond to group C and have a reserved rate of 1, 2, ..., 10 $Mbytes/sec.$ respectively. 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.
- All packets, regardless of flow groups are considered to be of constant size, 1000 $bytes$.

4.1.1 Overview of scenarios

In the scheduling node, unused bandwidth may exist because of two possible reasons described below

Unallocated (UA) bandwidth: Consider a system with only group *C* flows. Then the total reserved bandwith will be 55 $Mbytes/sec.$ which will result is 110 $Mbytes/sec.$ of unallocated bandwidth. We refer to this scenario as **UA**.

Allocated but unused (UU) bandwidth: Consider a system with all the three groups of flows. Then the total reserved bandwith will be 165 $Mbytes/sec.$ which will result is 0 $Mbytes/sec.$ of unallocated bandwidth. However, since flows in group *B* generate packets at a rate half of their reserved rates, it gives nearly 27.5 $Mbytes/sec.$ of unused bandwidth on average which can be used by other flows. We refer to this scenario as **U**U.

Thus, we simulate three general cases:

(1) **UA**, which is achieved by including only flows in groups A and C and thus leaves 1/3 of the channel bandwidth unallocated,

(2) **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

(3) **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.

Since we assume that flows in group *C* always have non-empty queues, their packet arrival distribution doesn't affect the behavior of the system. However, for groups A and B, we simulate two kinds of arrival distributions:

- constant packet arrival rate
- poisson arrival rate.

We thus have a total of six different scenarios each for four different scheduling algorithms. In the next sections, we present the results of simulations.

4.2 SIMULATION RESULTS

4.2.1 UA, Poisson

In this scenario, two groups A and C are considered and flows in group A have a poisson arrival process. In the plots, flows with id $1, 2, ..., 10$ correspond to flows A_1 through A_{10} and flows with id $11, 12, ..., 20$ correspond to flows C_1 through C_{10}.

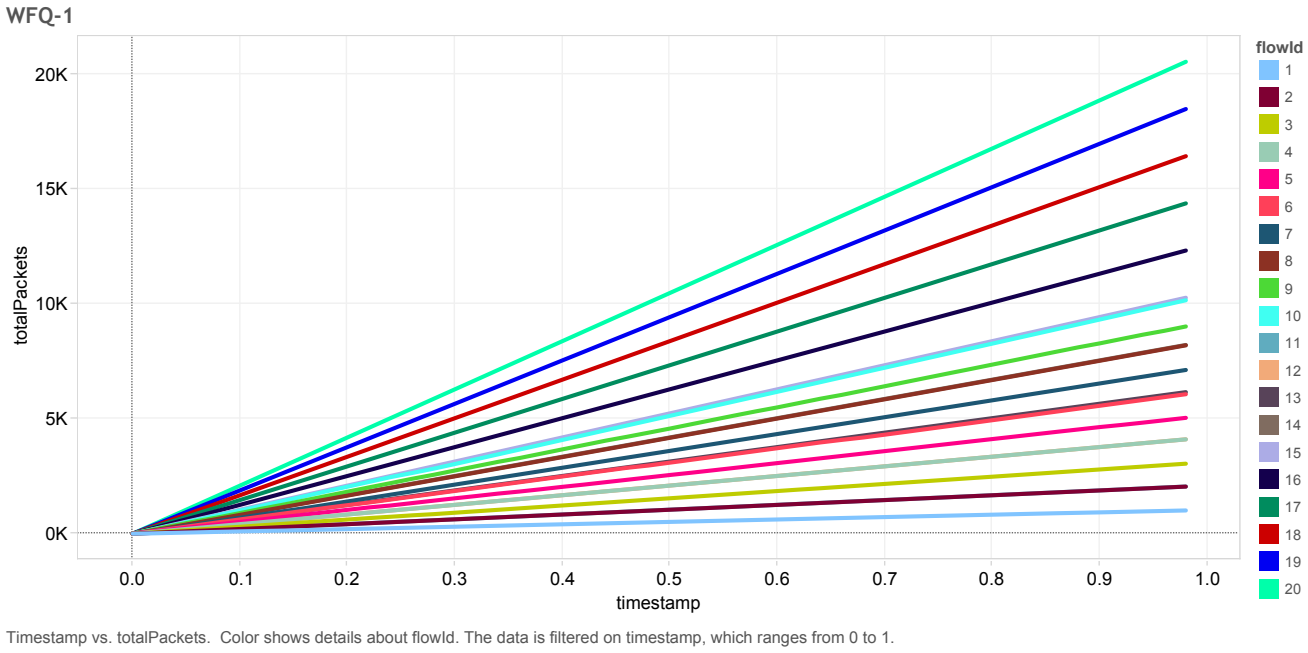


FIGURE 4.2.1. UA, Poisson, WFQ

Figure 4.2.1 shows the behavior of WFQ. Since flows A_1 through A_{10} produce the packets at an average rate equal to their reserved rates, they are not able to utilize the extra bandwidth. However, since C_1 through C_{10} are greedy and always have non-empty queues, all of them get nearly twice the bandwidth of the corresponding flows in group A, e.g. C_{20} is able to transmit twice the number of packets than A_{10} even though both of them have a reserved rate of 10 $Mbytes/sec$..

Figure 4.2.2 shows the behavior of pure EQ scheduler. Since flows A_1 through A_{10} produce the packets at an average rate equal to their reserved rates, they are not able to utilize the extra bandwidth. However, since C_1 through C_{10} are greedy and always have non-empty queues, each of the group C flows get a bandwidth of nearly 11 $Mbytes/sec.$ and thus converge together at a point above the plot of flow A_{10}.

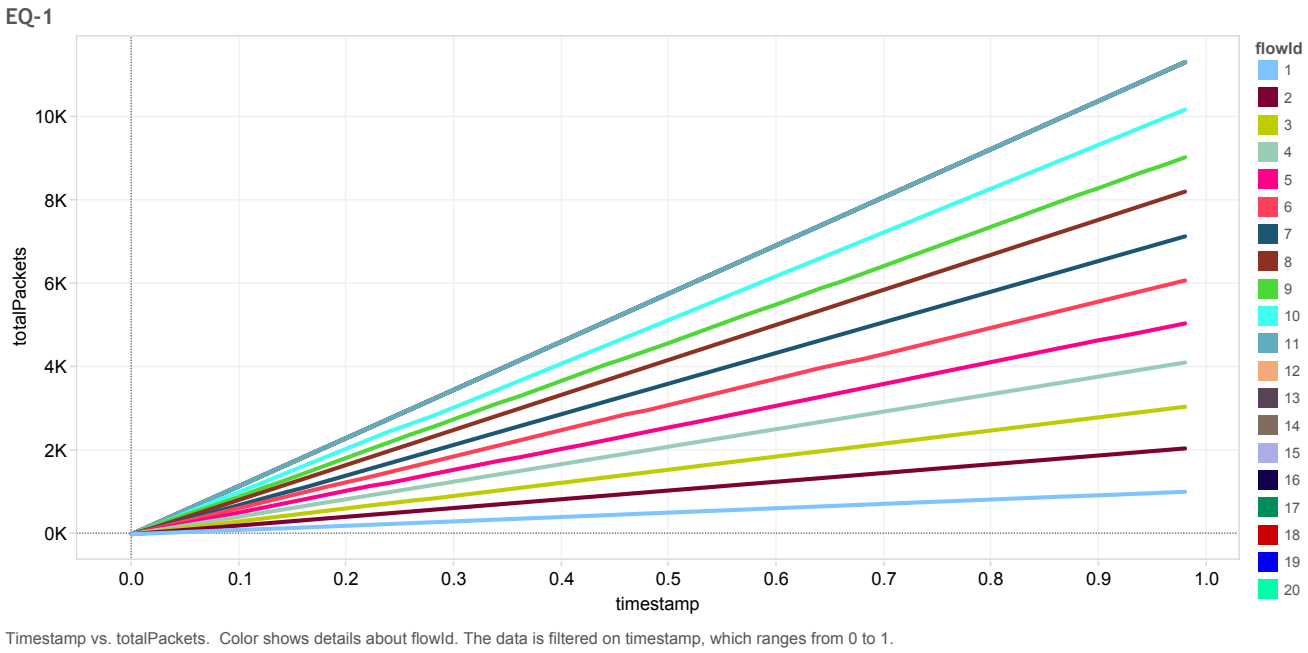


FIGURE 4.2.2. UA, Poisson, EQ

Figure 4.2.3 shows the behavior of PEQ scheduler. As in the case of EQ, C_1 through C_{10} are greedy and always have non-empty queues, each of the group C flows get a bandwidth of nearly 11 $Mbytes/sec.$ on average and thus converge together at a point above the plot of flow A_{10}. The plot is nearly identical to that of EQ scheduler.

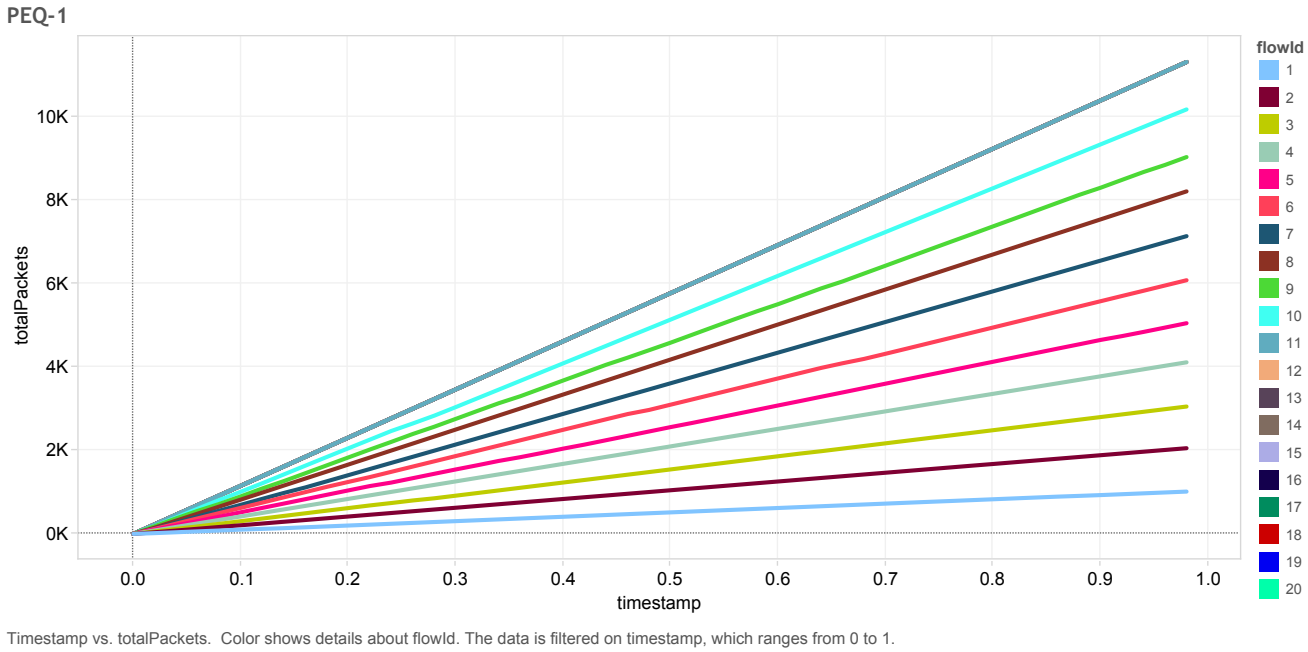


FIGURE 4.2.3. UA, Poisson, PEQ

Figure 4.2.4 shows the behavior of A-PEQ scheduler. As in the case of PEQ, C_1 through C_{10} are greedy and always have non-empty queues, each of the group C flows get a bandwidth of nearly 11 $Mbytes/sec.$ on average. The plot for the greedy flows is nearly identical to that of EQ and PEQ. Thus we can argue that $A-PEQ$ is as fair as EQ and PEQ in this case.

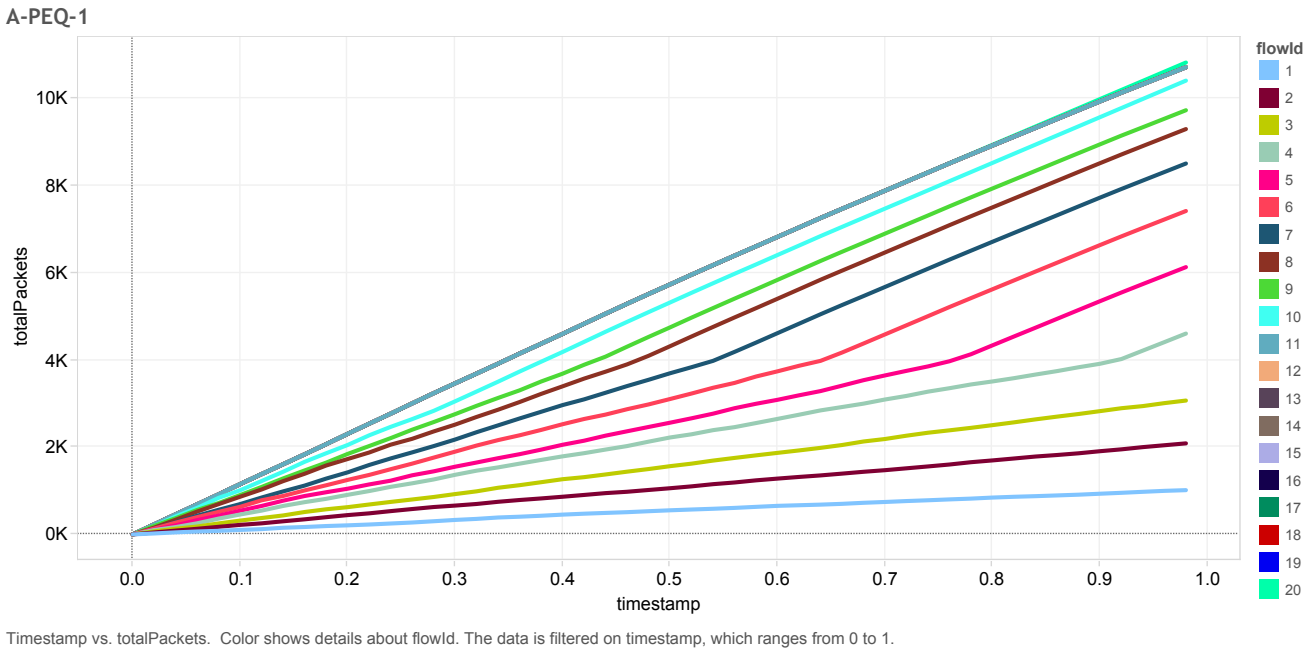


FIGURE 4.2.4. UA, Poisson, A-PEQ

4.2.2 UA, Constant Rate

In this scenario, two groups A and C are considered and flows in group A have a constant rate arrival process. In the plots, flows with id 1, 2, ..., 10 correspond to flows A_1 through A_{10} and flows with id 11, 12, ..., 20 correspond to flows C_1 through C_{10}.

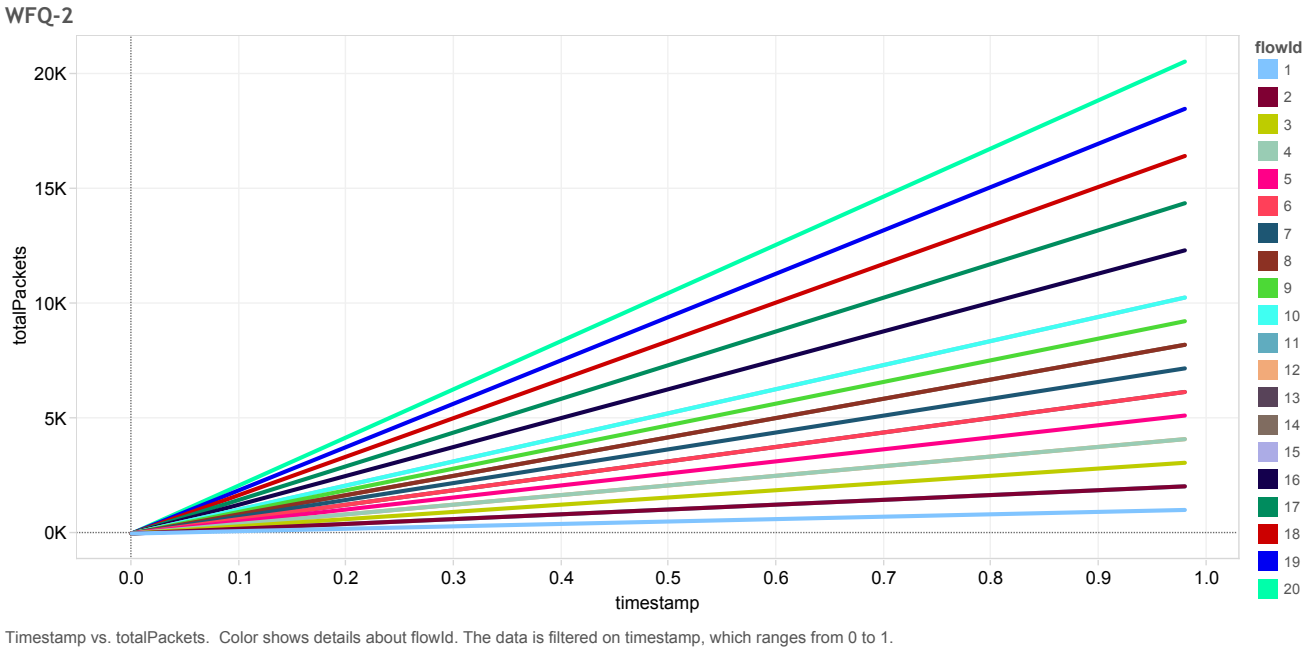


FIGURE 4.2.5. UA, CR, WFQ

Figure 4.2.5 shows the behavior of WFQ. Since flows A_1 through A_{10} produce the packets at a constant rate equal to their reserved rates and they are still not able to utilize the extra bandwidth. However, since C_1 through C_{10} are greedy and always have non-empty queues, all of them get nearly twice the bandwidth of the corresponding flows in group A, e.g. C_{20} is able to transmit twice the number of packets than A_{10} even though both of them have a reserved rate of 10 $Mbytes/sec$..

Figure 4.2.6 shows the behavior of pure EQ scheduler. Since flows A_1 through A_{10} produce the packets at a constant rate equal to their reserved rates, they are not able to utilize the extra bandwidth. However, since C_1 through C_{10} are greedy and always have non-empty queues, each of the group C flows get a bandwidth of nearly 11 $Mbytes/sec.$ and thus converge together at a point above the plot of flow A_{10}.

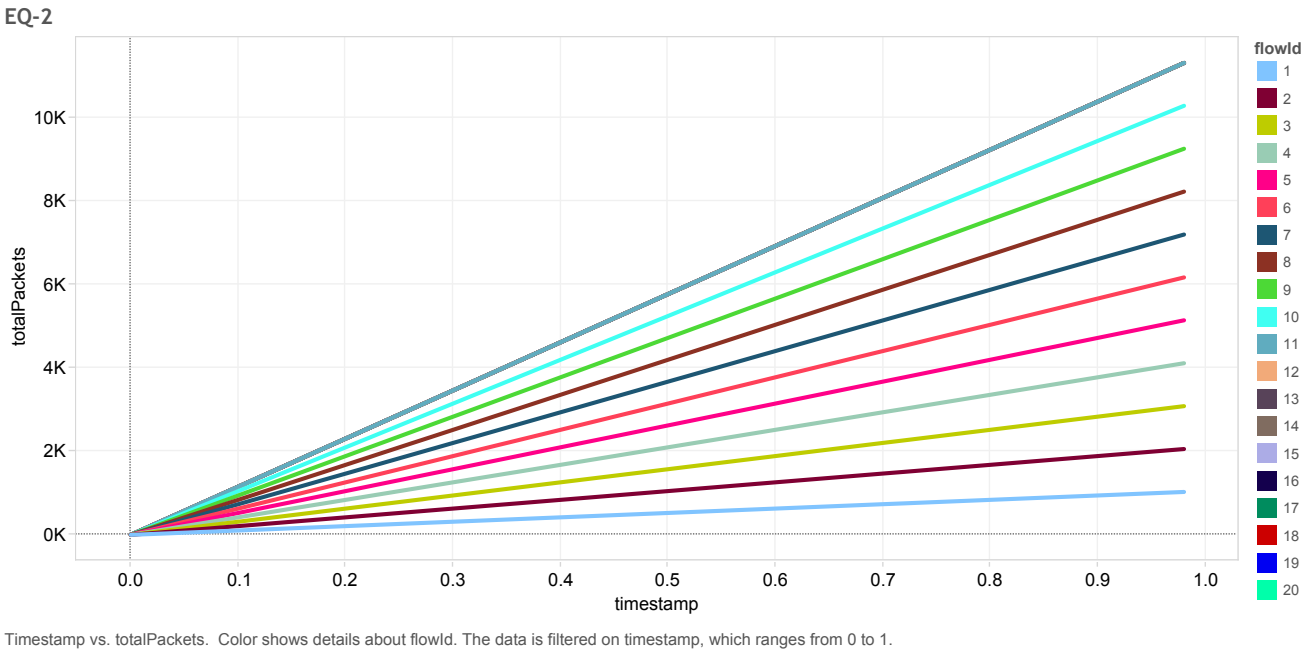


FIGURE 4.2.6. UA, CR, EQ

Figure 4.2.7 shows the behavior of PEQ scheduler. C_1 through C_{10} are greedy and always have non-empty queues, each of the group C flows get a bandwidth of nearly 11 $Mbytes/sec.$ on average and thus converge together at a point above the plot of flow A_{10}. The plot is nearly identical to that of EQ scheduler and is more uniform.

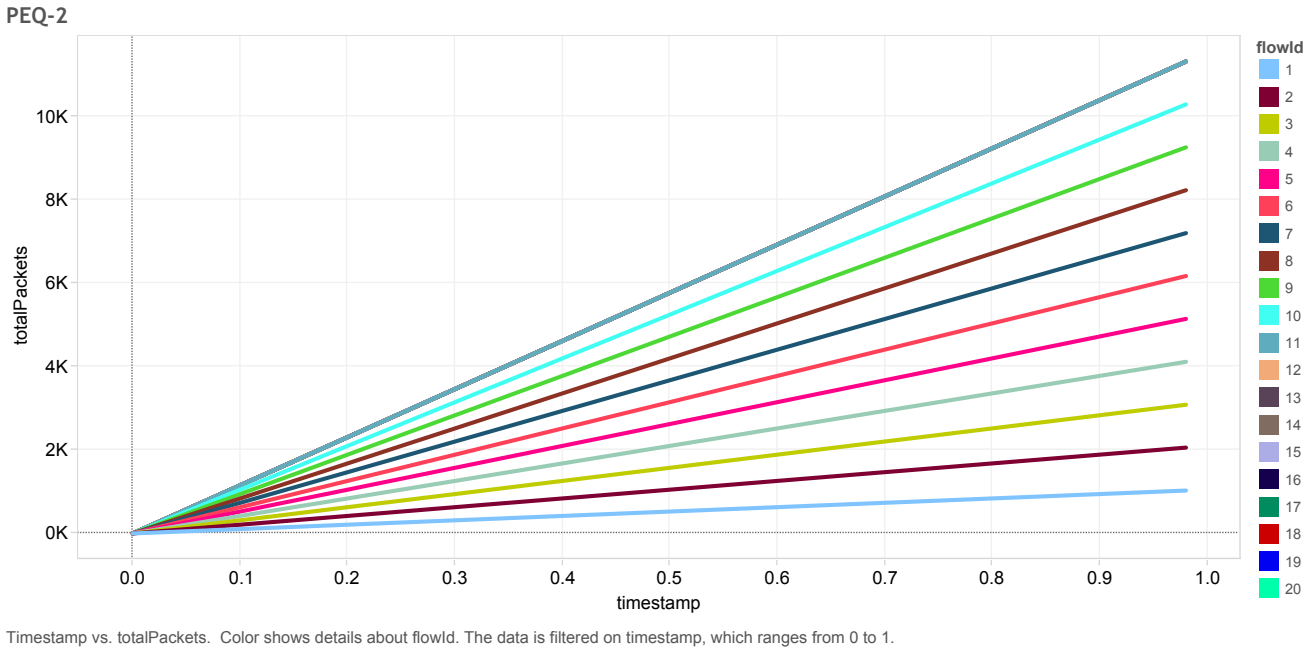


FIGURE 4.2.7. UA, CR, PEQ

Figure 4.2.8 shows the behavior of A-PEQ scheduler. As in the case of PEQ, C_1 through C_{10} are greedy and always have non-empty queues, each of the group C flows get a bandwidth of nearly 11 $Mbytes/sec.$ on average. The plot for the greedy flows is nearly identical to that of EQ and PEQ. For the flows in group A, the plots are more uniform than in case of poisson arrivals since the arrival rate is constant.

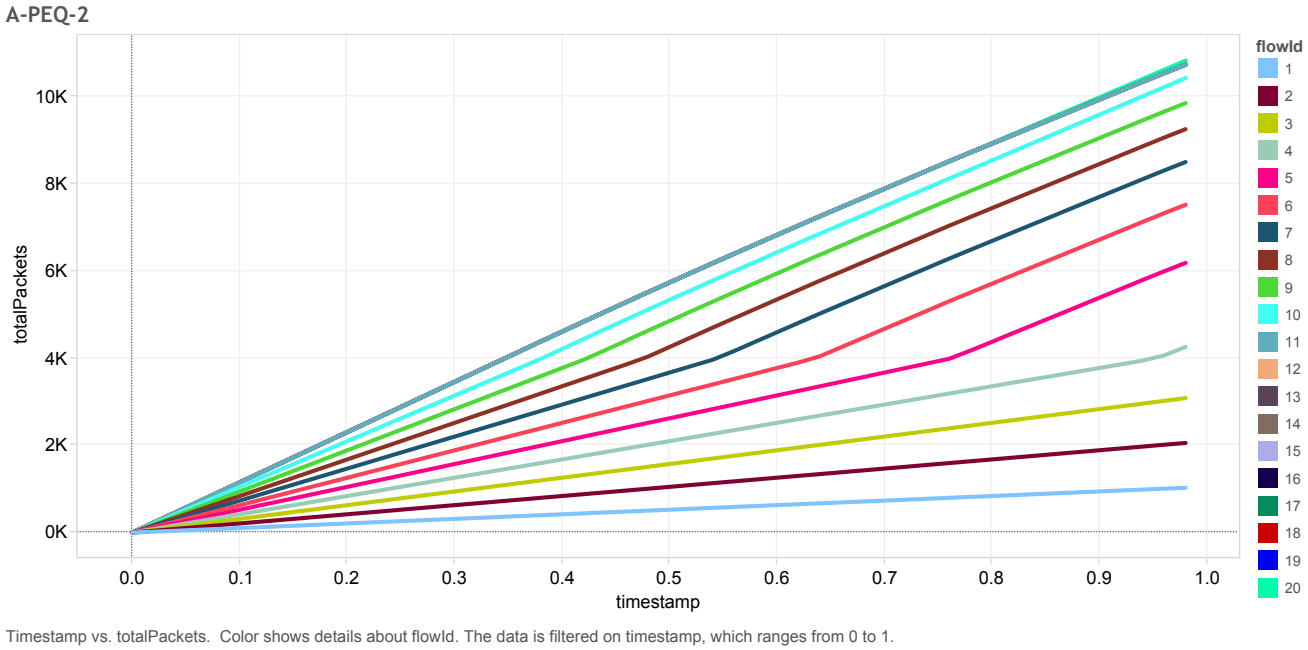


FIGURE 4.2.8. UA, CR, A-PEQ

4.2.3 UU, Poisson

In this scenario, all the three groups A, B and C are considered and flows in group A and B have a poisson arrival process. In the plots, flows with id 1, 2, ..., 10 correspond to flows A_1 through A_{10} and flows with id 11, 12, ..., 20 correspond to flows B_1 through B_{10} and flows with id 21, 22, ..., 30 correspond to flows C_1 through C_{10}.

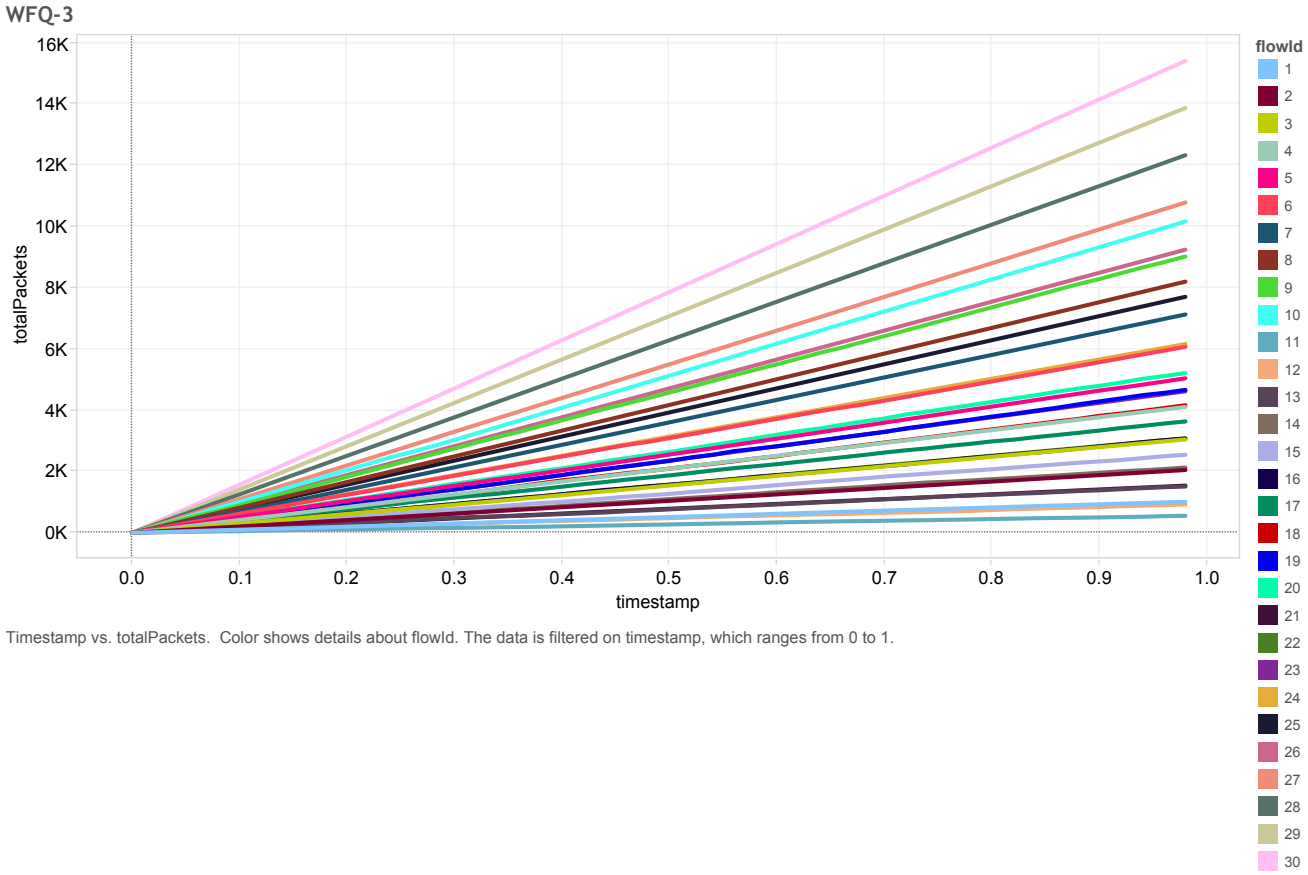


FIGURE 4.2.9. UU, Poisson, WFQ

Figure 4.2.9 shows the behavior of WFQ. The only unused bandwidth available is because of group B flows producing data at a slower speed. This gives roughly 27.5 $Mbytes/sec.$ of bandwidth which can be distributed among the flows of group C. For example, flow C_1 gets an average bandwidth of 1.5 $Mbytes/sec$.

Figure 4.2.10 shows the behavior of pure EQ scheduler. In this case too, only group C flows can use the extra bandwidth. However, the distribution is quite different from that of WFQ. Since there is not enough bandwidth to give all the group C flows an equal share (27.5 $Mbytes/sec.$ can bring the first 7 flows together), flows from C_1 through C_7 tend to come together.

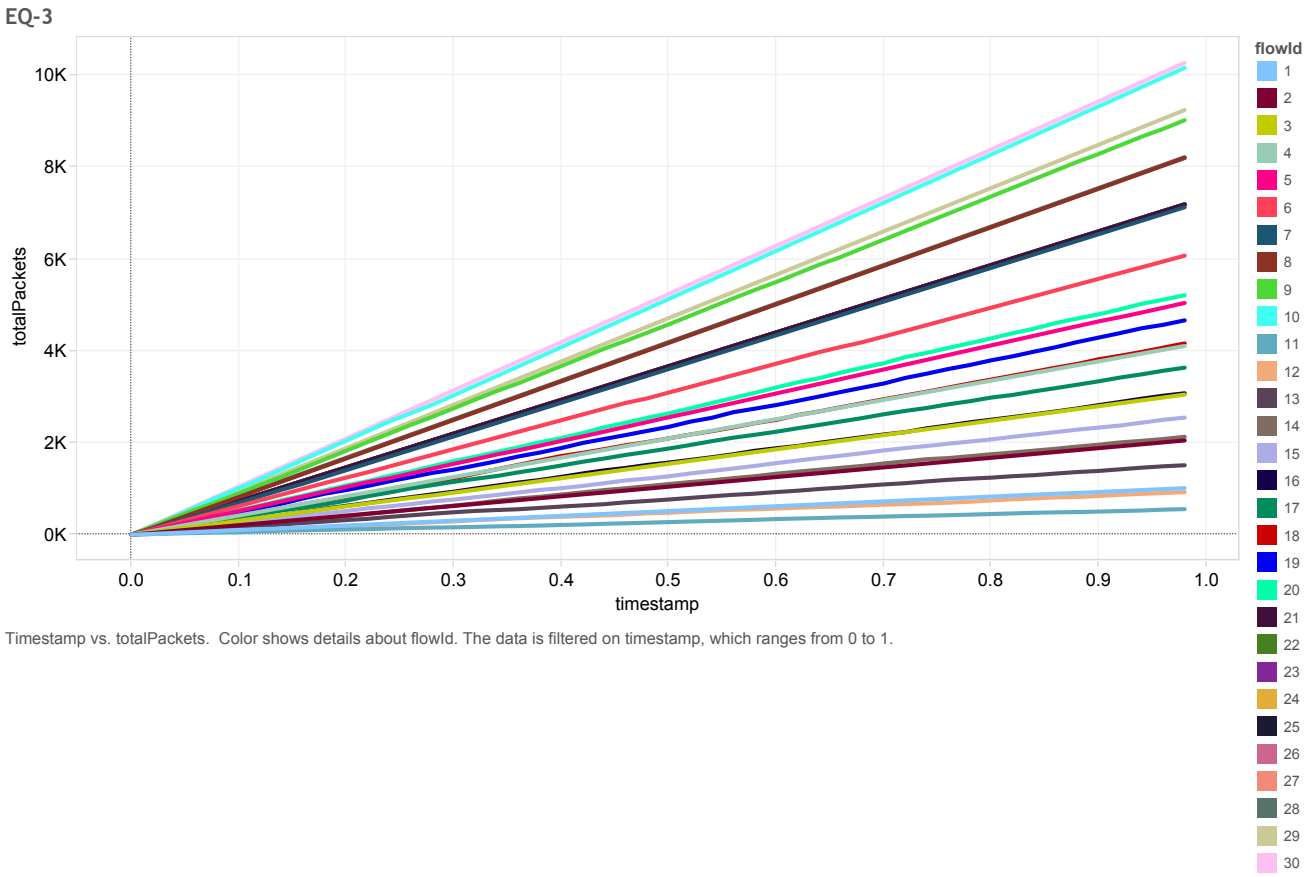


FIGURE 4.2.10. UU, Poisson, EQ

Figure 4.2.11 shows the behavior of PEQ scheduler. All the greedy flows cannot be brought together which is why, flows C_1 through C_7 converge below C_8.

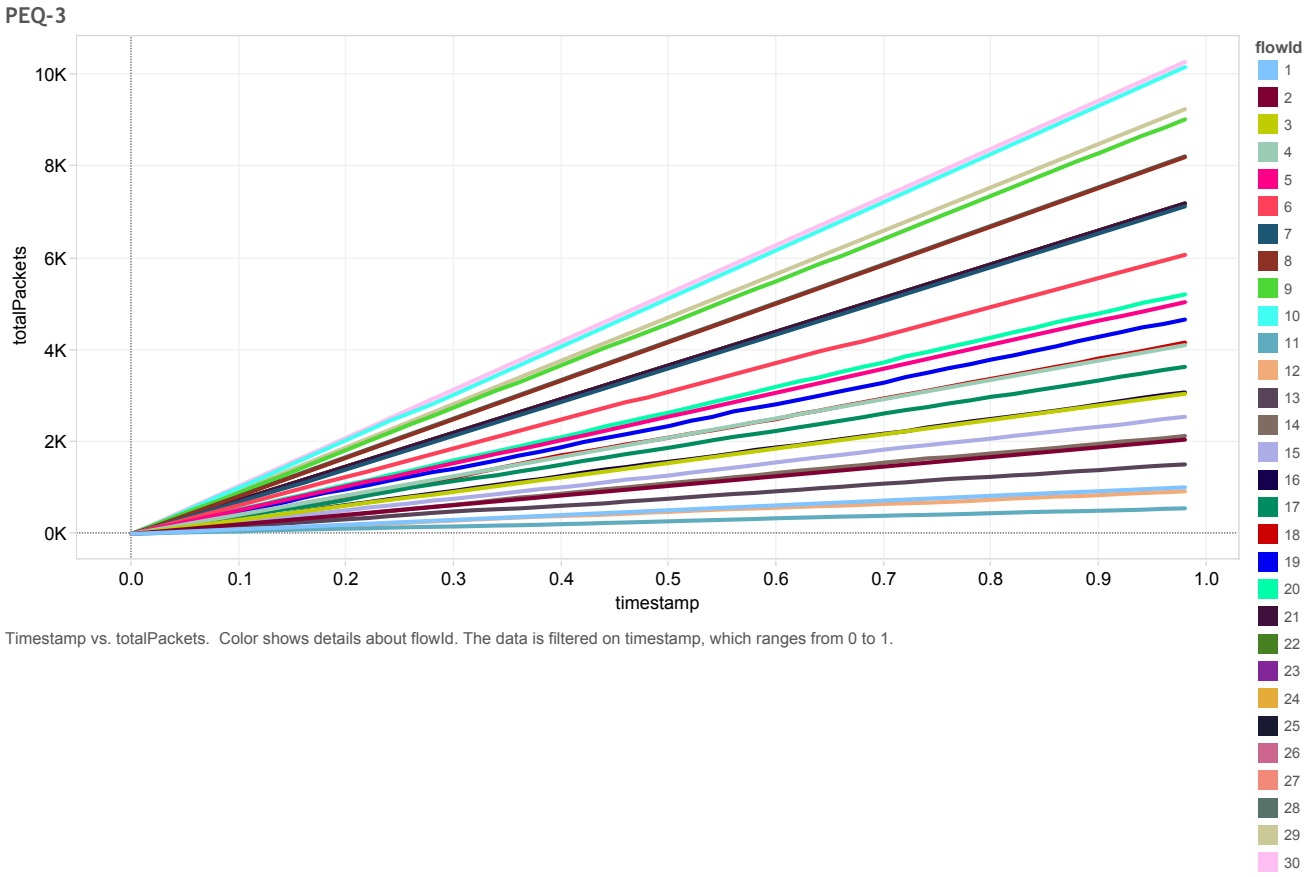


FIGURE 4.2.11. UU, Poisson, PEQ

Figure 4.2.12 shows the behavior of A-PEQ scheduler. Here we can see that as the time goes on, the flows C_1 through C_7 tend to drift apart from the flow C_8 and resembles the EQ and PEQ scheduler in corresponding scenarios.

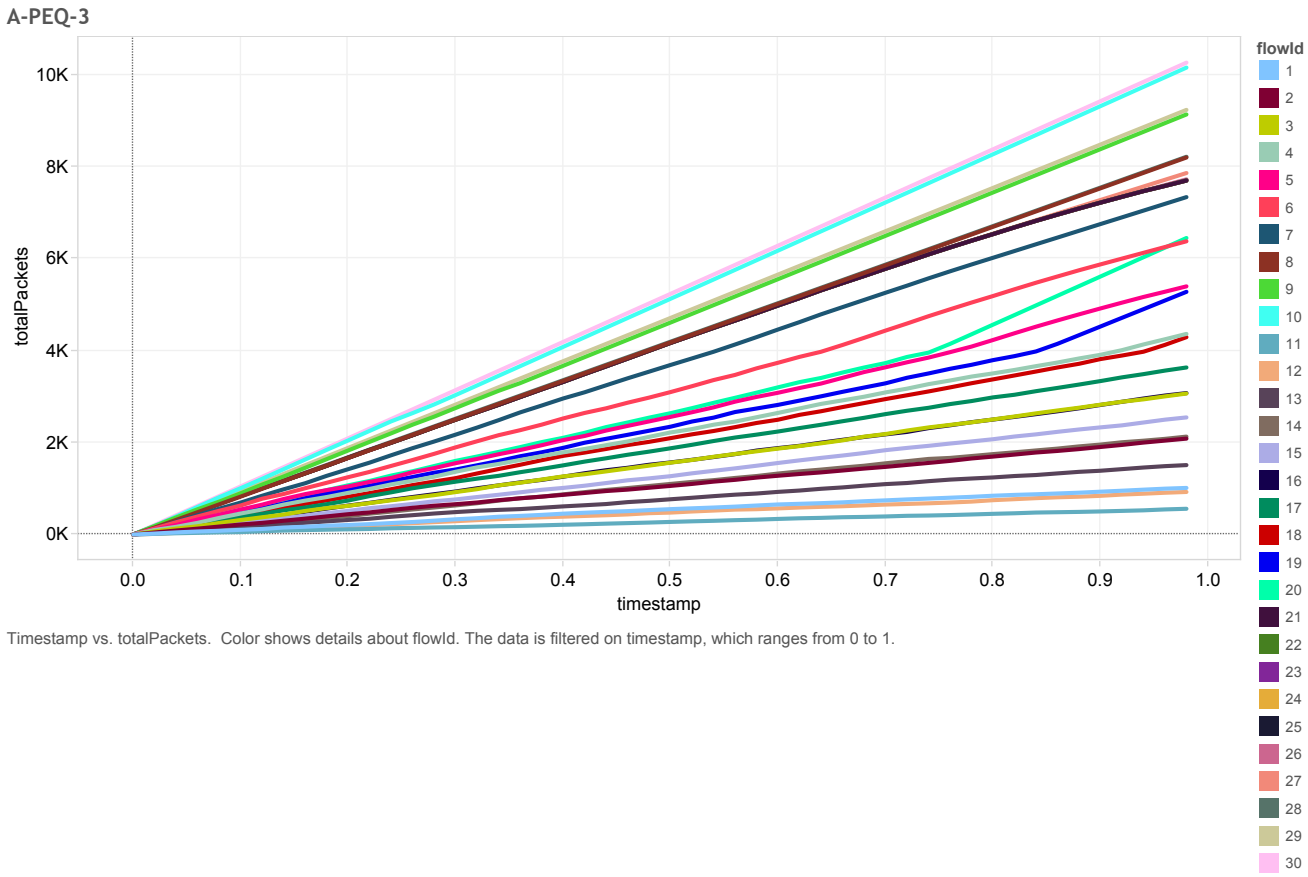


FIGURE 4.2.12. UU, Poisson, A-PEQ

4.2.4 UU, Constant Rate

In this scenario, all the three groups A, B and C are considered and flows in group A and B have a constant rate arrival process. In the plots, flows with id 1, 2, ..., 10 correspond to flows A_1 through A_{10} and flows with id 11, 12, ..., 20 correspond to flows B_1 through B_{10} and flows with id 21, 22, ..., 30 correspond to flows C_1 through C_{10}.

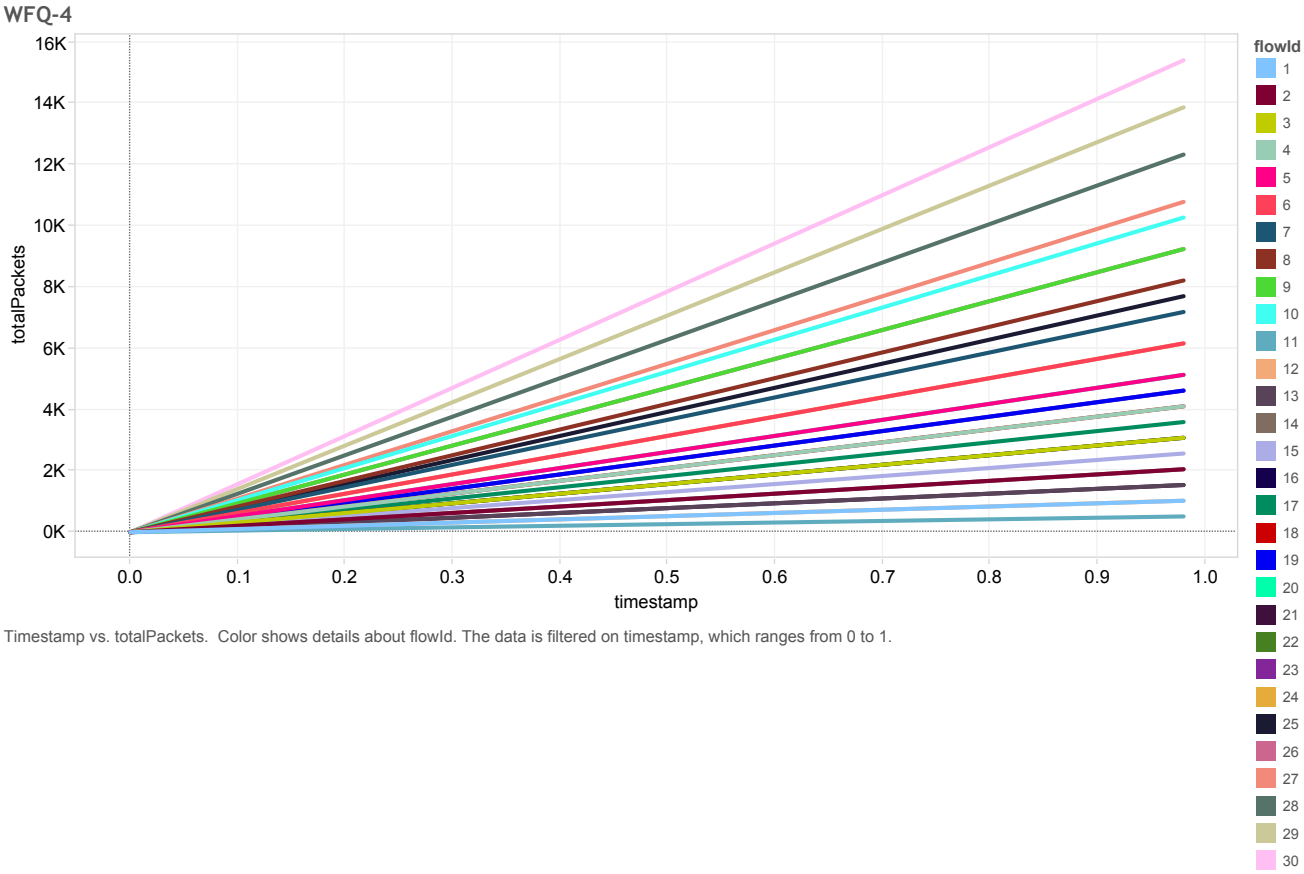


FIGURE 4.2.13. UU, CR, WFQ

Figure 4.2.13 shows the behavior of WFQ. The only unused bandwidth available is because of group B flows producing data at a slower speed. This gives roughly 27.5 $Mbytes/sec.$ of bandwidth which can be distributed among the flows of group C. For example, flow C_1 gets an average bandwidth of 1.5 $Mbytes/sec$. Also, with a constant arrival rate, the plots tend to be more uniform.

Figure 4.2.14 shows the behavior of pure EQ scheduler. In this case too, only group C flows can use the extra bandwidth. However, the distribution is quite different from that of WFQ. Since there is not enough bandwidth to give all the group C flows an equal share (27.5 $Mbytes/sec.$ can bring the first 7 flows together), flows from C_1 through C_7 tend to come together below the flow C_8.

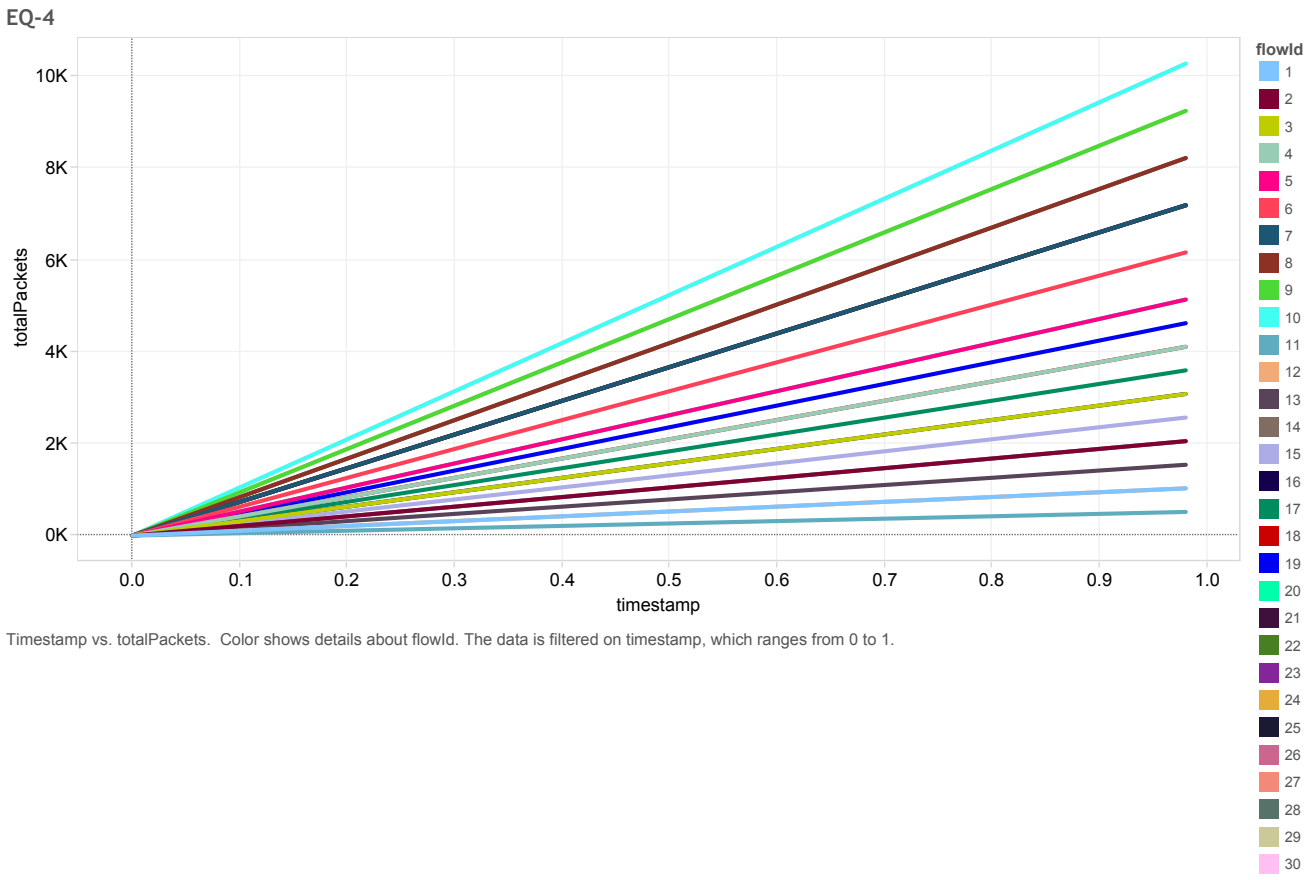


FIGURE 4.2.14. UU, CR, EQ

Figure 4.2.15 shows the behavior of PEQ scheduler. All the greedy flows cannot be brought together which is why, flows C_1 through C_7 converge below C_8. Also, we can see nearly perfect overlapping due to the constant rate arrival process.

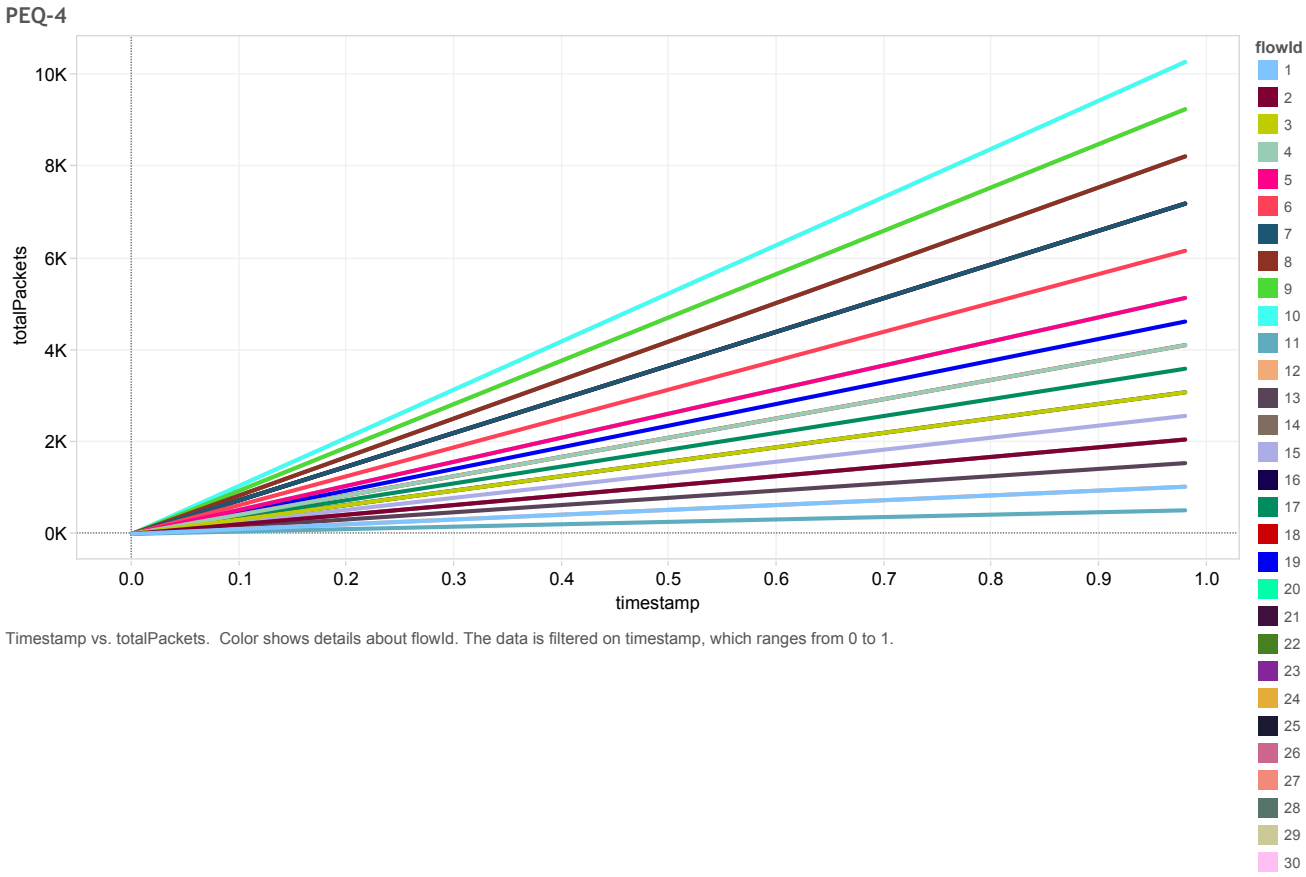


FIGURE 4.2.15. UU, CR, PEQ

Figure 4.2.16 shows the behavior of A-PEQ scheduler. As with the poisson arrivals, we can see that with time, the flows C_1 through C_7 tend to drift apart from the flow C_8 and resembles the EQ and PEQ scheduler in corresponding scenarios. For the flows in groups A and B, due to constant arrival rate, perfect overlapping can be noticed.

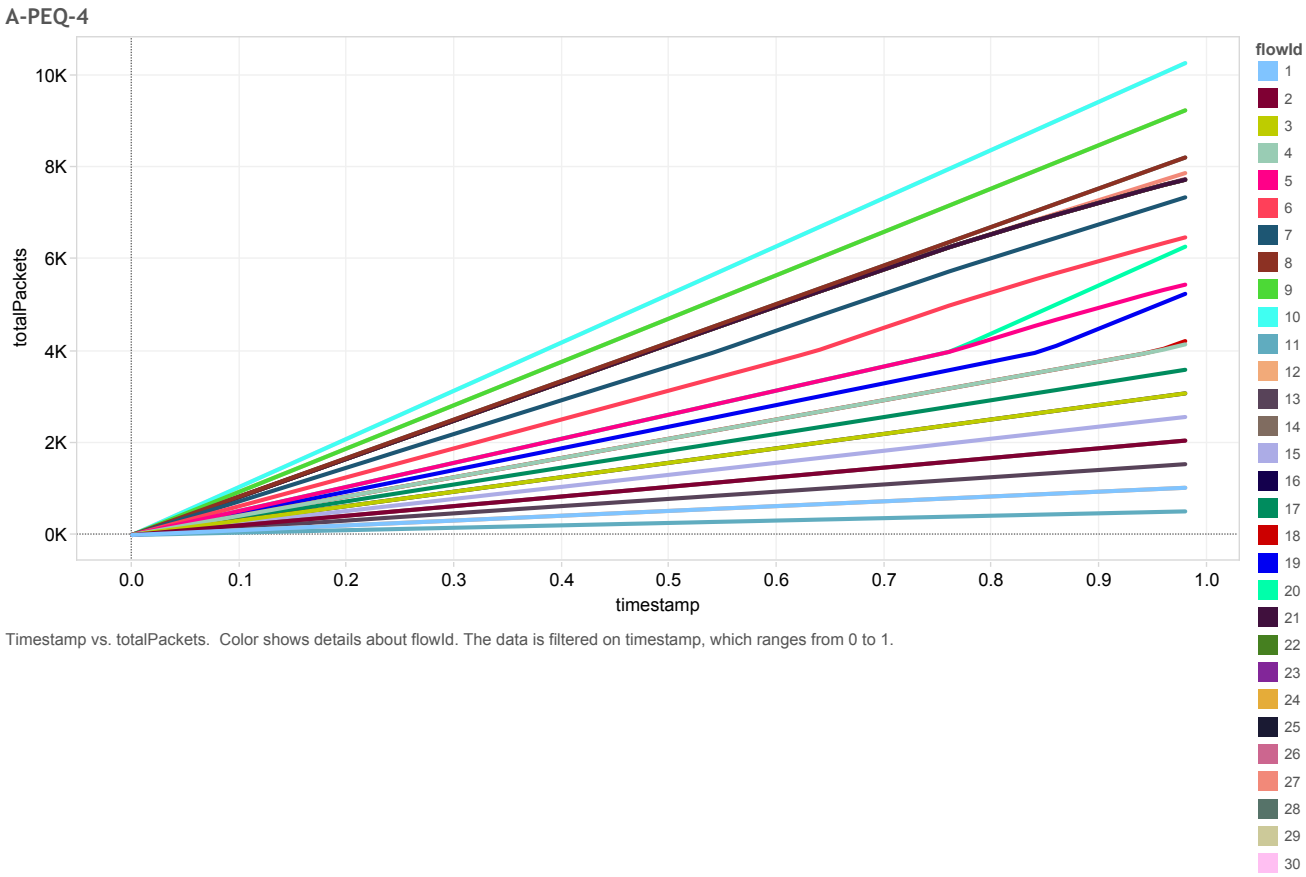


FIGURE 4.2.16. UU, CR, A-PEQ

4.2.5 UA + UU, Poisson

In this scenario, two groups B and C are considered and flows in group C have a poisson arrival process. In the plots, flows with id $1, 2, ..., 10$ correspond to flows B_1 through B_{10} and flows with id $11, 12, ..., 20$ correspond to flows C_1 through C_{10}.

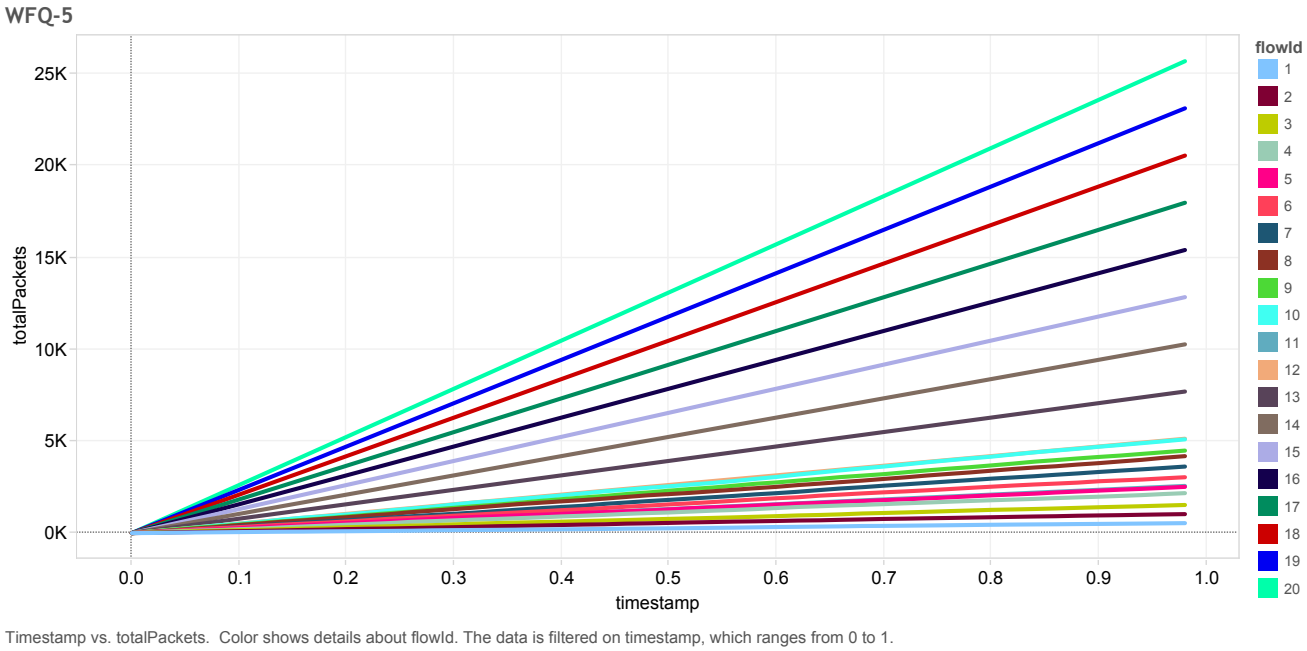


FIGURE 4.2.17. UA + UU, Poisson, WFQ

Figure 4.2.17 shows the behavior of WFQ. Since there is unused bandwidth because of unallocation and flows in group B generating traffic at half of their reserved rate, flows in group C get nearly 2.5 times of their reserved rates.

Figure 4.2.18 shows the behavior of pure EQ scheduler. A total of 137.5 $Mbytes/sec$. (110 + 27.5) of bandwidth can be allocated to group C flows. Thus, all the flows in this group can be brought together and given an average bandwidth of nearly 13.7 $Mbytes/sec$. The highest bandwidth used by any group B flow is nearly 5 $Mbytes/sec$. which is why we see a huge gap between the plots of two flows.

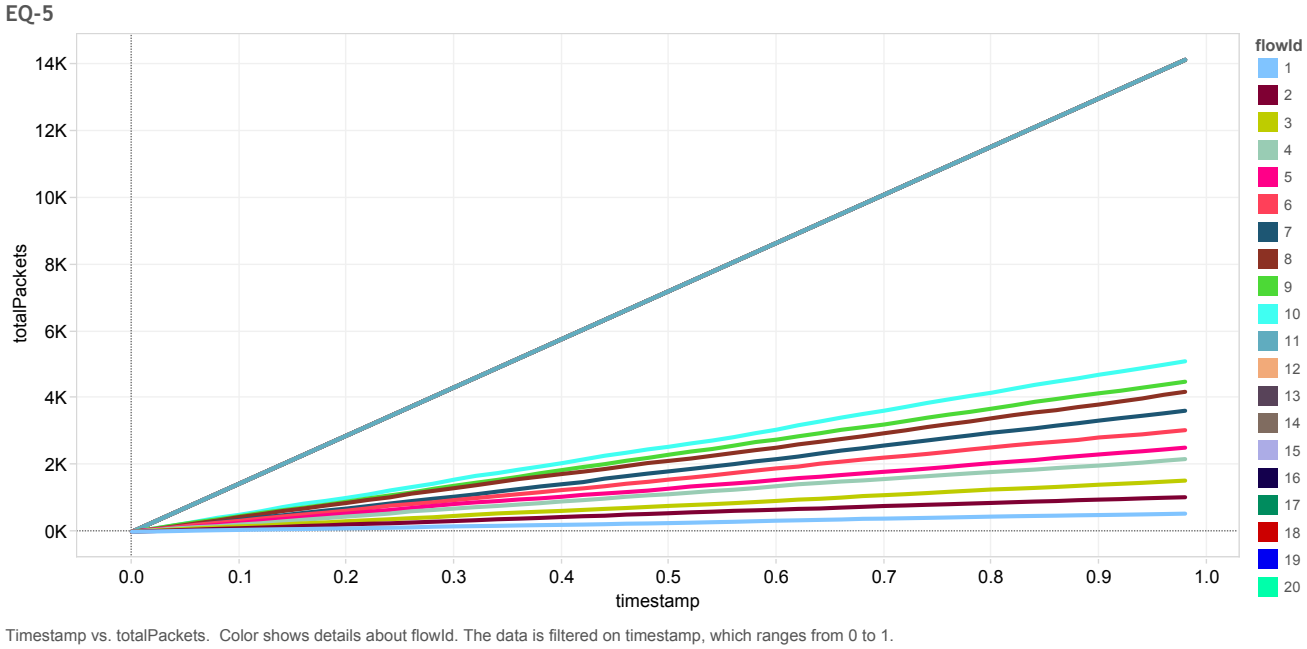


FIGURE 4.2.18. UA + UU, Poisson, EQ

Figure 4.2.19 shows the behavior of PEQ scheduler. The plot is identical to that of EQ scheduler. Each flow in group C gets an average bandwidth of 13.7 $Mbytes/sec$. For example, number of packets sent by flow C_{10} over a time of 1 second are:

$$\begin{aligned} count &= \frac{13.7 \times 1024 \times 1024}{1000} \\ &\backsimeq 14300 \; packets \end{aligned}$$

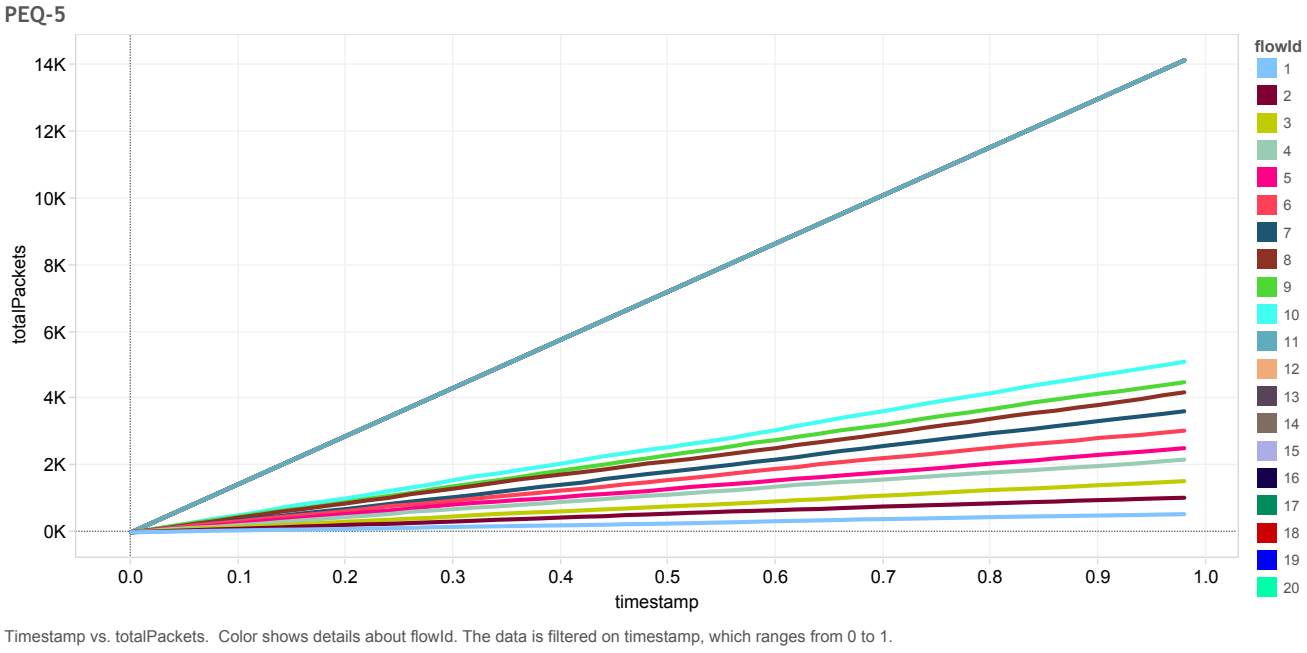


FIGURE 4.2.19. UA + UU, Poisson, PEQ

Figure 4.2.20 shows the behavior of A-PEQ scheduler. Flows in group C group together way above the group B flows. Considering flow C_1, number of packets sent over time period of 1 *second* is nearly 14200. Thus its average bandwidth can be calculated as under:

$$\begin{aligned} \textit{Average BW} &= \frac{14000 \times 1000}{1024 \times 1024} \\ &\backsimeq 13.35\ \textit{Mbytes/sec.} \end{aligned}$$

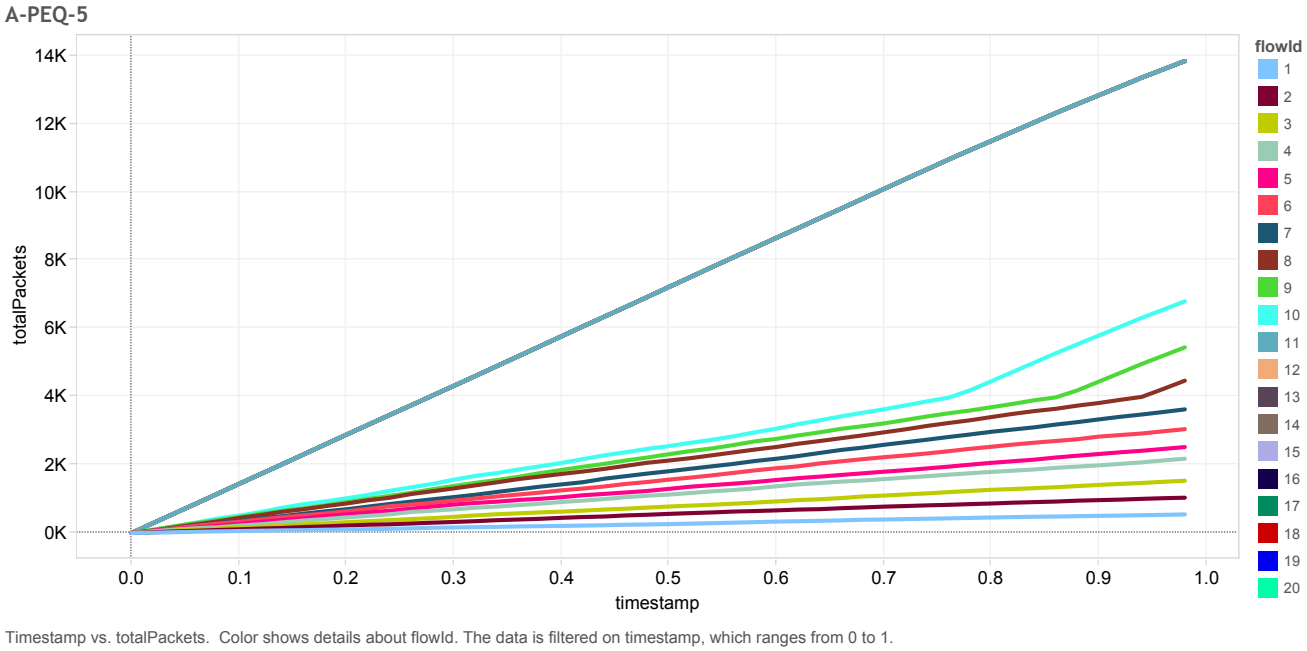


FIGURE 4.2.20. UA + UU, Poisson, A-PEQ

4.2.6 UA + UU, Constant Rate

In this scenario, two groups B and C are considered and flows in group C have a poisson arrival process. In the plots, flows with id $1, 2, ..., 10$ correspond to flows B_1 through B_{10} and flows with id $11, 12, ..., 20$ correspond to flows C_1 through C_{10}.

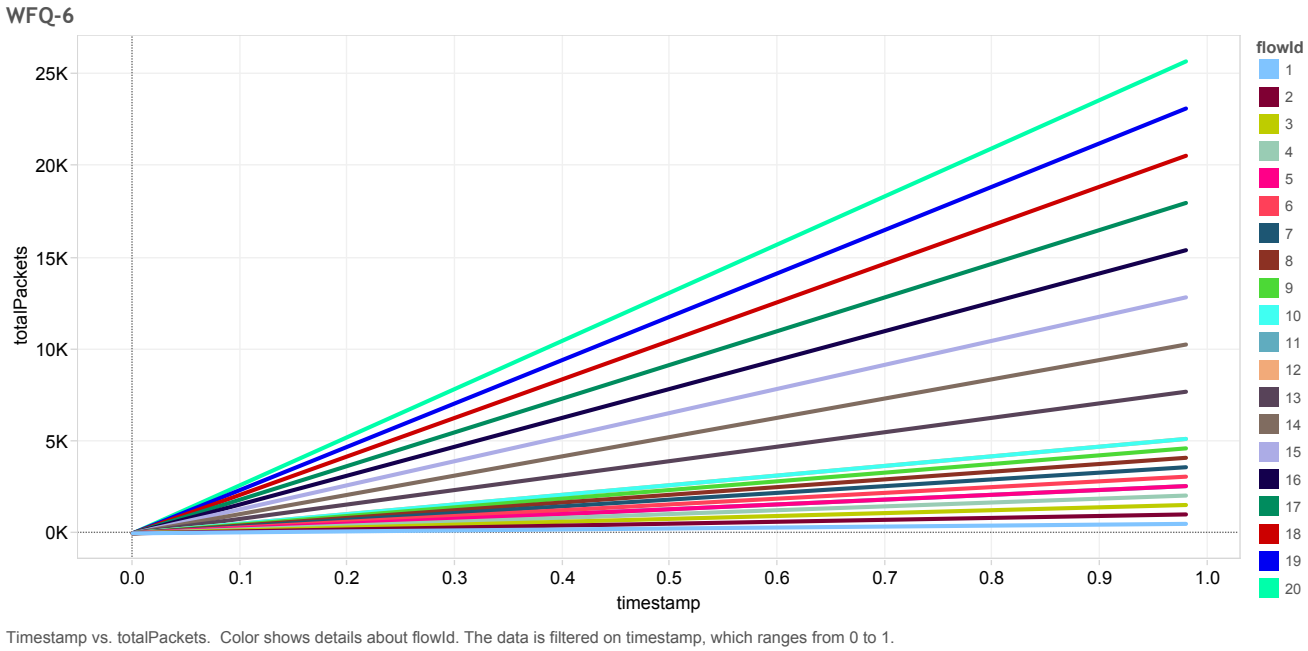


FIGURE 4.2.21. UA + UU, CR, WFQ

Figure 4.2.21 shows the behavior of WFQ. Since there is unused bandwidth because of unallocation and flows in group B generating traffic at half of their reserved rate, flows in group C get nearly 2.5 times of their reserved rates. For example, flow C_{10} transmits nearly 25000 *packets* over a period of 1 *second* instead of 10000 *packets* if it was transmitting at its average rate.

Figure 4.2.22 shows the behavior of pure EQ scheduler. Like the poisson arrivals total of 137.5 $Mbytes/sec$. (110 + 27.5) of bandwidth can be allocated to group C flows. Thus, all the flows in this group can be brought together and given an average bandwidth of nearly 13.7 $Mbytes/sec$. The highest bandwidth used by any group B flow is nearly 5 $Mbytes/sec$.

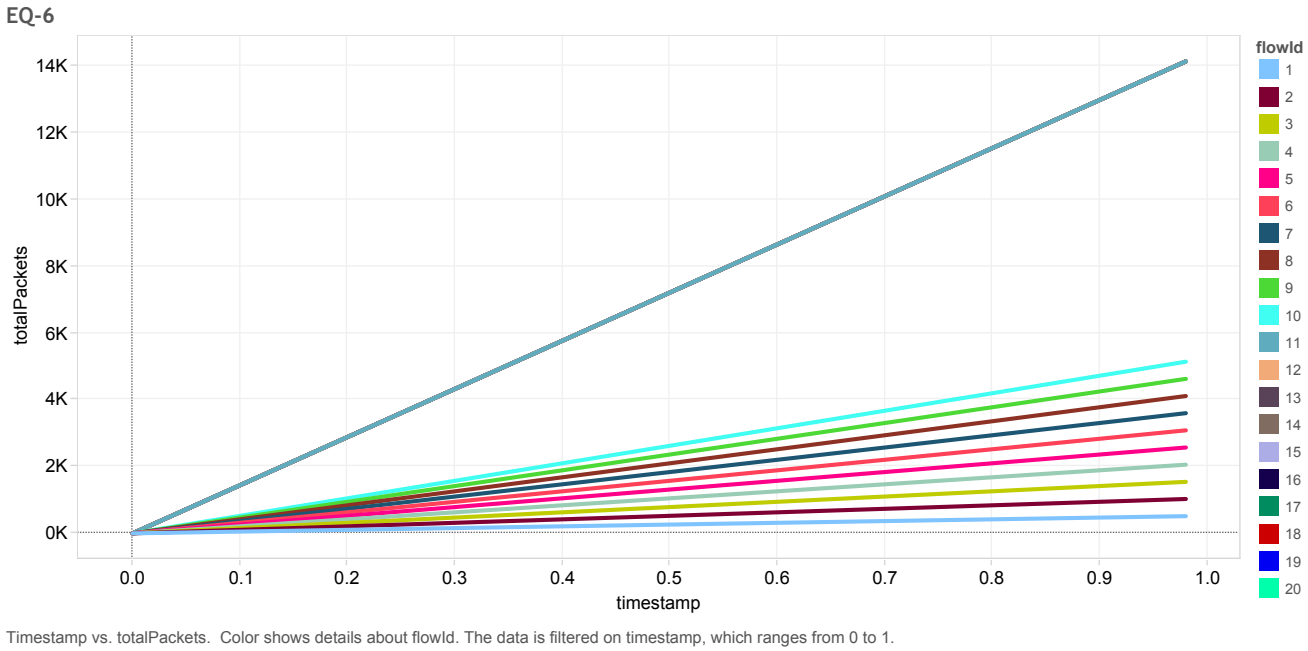


FIGURE 4.2.22. UA + UU, CR, EQ

Figure 4.2.23 shows the behavior of PEQ scheduler. Each flow in group C gets an average bandwidth of 13.7 $Mbytes/sec$. For example, number of packets sent by flow C_1 over a time of 1 second are:

$$\begin{aligned} count &= \frac{13.7 \times 1024 \times 1024}{1000} \\ &\backsimeq 14300 \; packets \end{aligned}$$

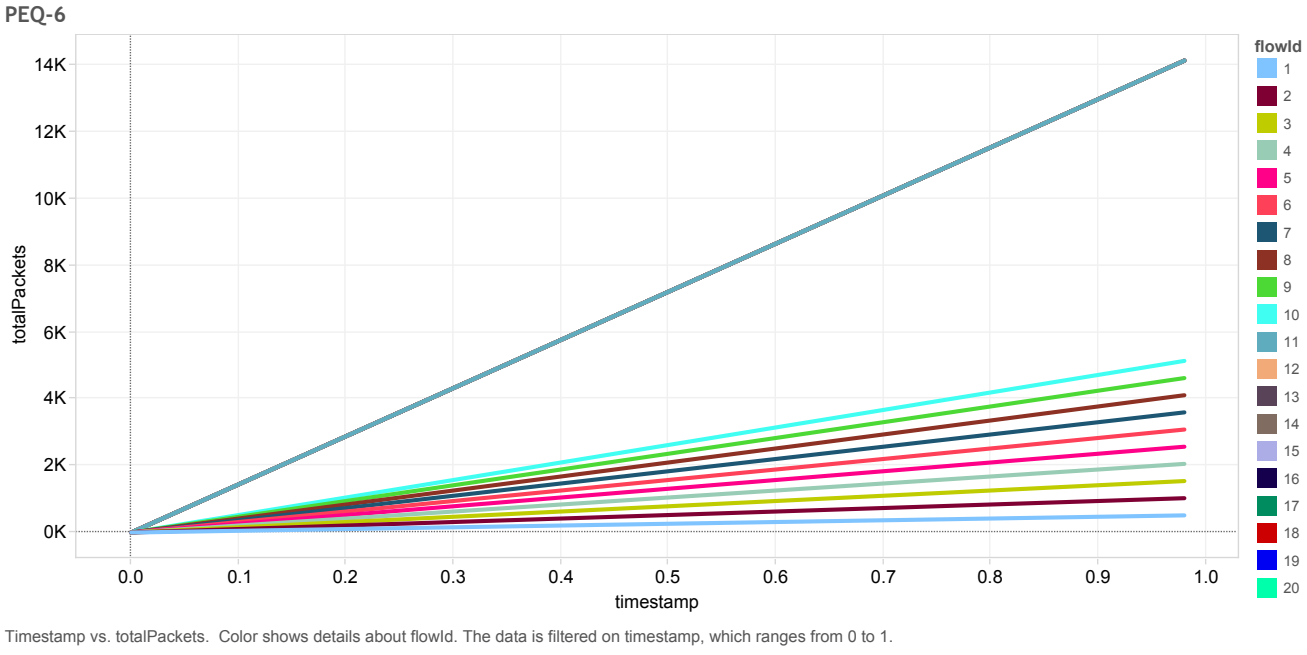


FIGURE 4.2.23. UA + UU, CR, PEQ

Figure 4.2.24 shows the behavior of A-PEQ scheduler. Flows in group C group together way above the group B flows. Considering flow C_{10}, number of packets sent over time period of 1 *second* is nearly 14200. Thus its average bandwidth can be calculated as under:

$$\begin{aligned} \textit{Average BW} &= \frac{14000 \times 1000}{1024 \times 1024} \\ &\backsimeq 13.35\ \textit{Mbytes/sec.} \end{aligned}$$

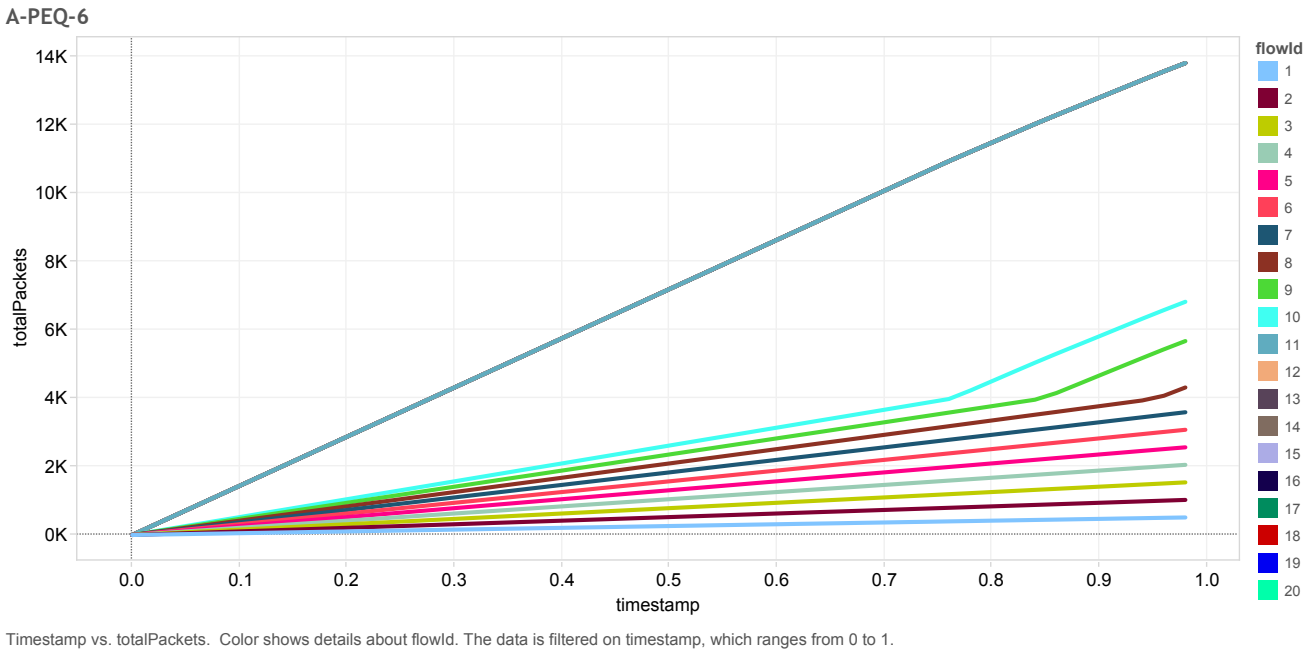


FIGURE 4.2.24. UA + UU, CR, A-PEQ

REMARK. An important point to note is that, the plots for both poisson and constant rate arrival process turn out to be quite similar. This is due to the fact that over a period of time, the expected number of arrivals in poisson arrival process and the constant rate arrival process should be equal. The instantaneous snapshots may vary. However, out plots are averaged over an interval of 0.02 seconds which is why, the two kinds of plots look quite identical.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

We started off with an aim to create a scheduler, which is rate-guaranteed, provides approximately EQ fairness and has a logarithmic complexity. Using theorems 1 and 2, we argued that A-PEQ scheduler is rate guaranteed and also, it has a logarithmic complexity. With the help of theorem 3 and simulations, we claimed that $A-PEQ$ offers approximately same fairness as EQ scheduler does. Thus $A-PEQ$ scheduler satisfies all the three design criteria.

Out of several areas that are still not explored, one thing is the tuneable parameter Δ. The effects of changing the value of Δ have not been studied during the course of this thesis due to time limitation. In the future work, bounds on this parameter need to be formally stated. A value as chosen in theorem 3 appears appropriate, the effects of different values for must be studied in detail.

Another 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 $\Delta-RR$ scheduler, but leaves the $PCR*$ scheduler as it is.

Finally, scheduling packets over multiple parallel links between neighboring computers has been studied for rate-guaranteed schedulers in Cobb and Lin [7]Blanquer and Ozden [3]. We also plan to investigate the impact of rate-equalization over parallel links.

BIBLIOGRAPHY

[1] J. C.R. Bennett and H. Zhang. Hierarchical packet fair queueing algorithms. *IEEE/ACM Transactions on Networking*, 5(5):675–689, October 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.*, pages 29–40, 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 and Miaohua Lin. End-to-end delay guarantees for multiple-channel schedulers. In *Proc. IEEE International Workshop on Quality of Service (IWQoS)*, Miami, May 2002.

[8] 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.

[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. doi: http://doi.ieeecomputersociety.org/10.1109/LCN.2011.6115513.

[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 Transactions on Networking*, 12(6), December 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] Partho P. Mishra and Hemant Kanakia. A hop by hop rate-based congestion control scheme. *SIGCOMM Comput. Commun. Rev.*, 22:112–123, October 1992. ISSN 0146-4833. doi: http://doi.acm.org/10.1145/144191.144254. URL `http://doi.acm.org/10.1145/144191.144254`.

[16] 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.

[17] S. Ramabhadran and J. Pasquale. Stratified round robin: A low com- plexity packet scheduler with bandwidth fairness and bounded delay. *ACM Comput. Commun. Review*, 33(4):239–250, October 2003.

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

[19] 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.

[20] Paolo Valente. Exact gps simulation with logarithmic complexity, and its application to an optimally fair scheduler. *SIGCOMM Comput. Commun. Rev.*, 34:269–280, August 2004. ISSN 0146-4833. doi: http://doi.acm.org/10.1145/1030194.1015497. URL `http://doi.acm.org/10.1145/1030194.1015497`.

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

[22] X. Yuan and Z. Duan. Frr: A proportional and worst-case fair round robin scheduler. In *Proc. IEEE INFOCOM*, pages 831–842, March 2005.

[23] Lixia Zhang. Virtual clock: A new traffic control algorithm for packet-switched networks. *ACM Transactions on Computer Systems*, 9(2):101–124, May 1991.

[24] 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.

[25] L. Zhong, J. Xu, and X. Wang. Vwqgrr: A novel packet scheduler. In *Proc. of the IEEE ICN*, pages 22–28, April 2007.