FAIR QUEUING AND SCHEDULING

Zhu Li, Yang Haoran, and Liu Zhongkun[,†]

E-mail:

Abstract

Is giving all control of congestion to the TCP layer really the only option? As the Internet has evolved, so have situations in which we may not want routers handling all traffic on a first-come, first-served basis. Traffic with delay bounds so-called real-time traffic, often involving either voice or video is likely to perform much better with preferential service. Even without real-time traffic, we might be interested in guaranteeing that each of several customers gets an agreed-upon fraction of bandwidth, regardless of what the other customers are receiving. If we trust only to TCP Renos bandwidth-allocation mechanisms, and if one customer has one connection and another has ten, then the bandwidth received may also be in the ratio of 1:10. This may make the first customer quite unhappy. The fundamental mechanism for achieving these kinds of traffic-management goals in a shared network is through queuing; that is, in deciding how the routers prioritize traffic. In this report we carry out several experiments to compare different scheduling algorithms.

Introduction

Queuing disciplines provide tools for meeting administratively imposed constraints on traffic. Two senders, for example, might be required to share an outbound link equally, or in the proportion 60%-40%, even if one participant would prefer to use 100% of the bandwidth. Alternatively, a sender might be required not to send in bursts of more than 10 packets at

a time. Closely allied to the idea of queuing is scheduling: deciding what packets get sent when. Scheduling may take the form of sending someone elses packets right now, or it may take the form of delaying packets that are arriving too fast.

you may wish to allocate bandwidth according to administratively determined percentages. For example, you may wish to give each of three departments an equal share of download (or upload) capacity, or you may wish to guarantee them shares of 55%, 35% and 10%. If you want any unused capacity to be divided among the non-idle users, fair queuing is the tool of choice, though in some contexts it may require cooperation from your ISP. If the users are more like customers receiving only the bandwidth they pay for, you might want to enforce flat caps even if some bandwidth thus goes unused; token-bucket filtering would then be the way to go. If bandwidth allocations are not only by department (or customer) but also by workgroup (or customer-specific subcategory), then hierarchical queuing offers the necessary control. In general, network management divides into managing the hardware and managing the traffic; the tools in this chapter address this latter component. These tools can be used internally by ISPs and at the customer/ISP interconnection, but traffic management often makes good economic sense even when entirely contained within a single organization.

Related Work

Fair queuing is an important alternative to FIFO and priority. Where FIFO and its variants have a single input class and put all the incoming traffic into a single physical queue, fair queuing maintains a separate logical FIFO subqueue for each input class; we will refer to these as the per-class subqueues. Division into classes can be fine-grained eg a separate class for each TCP connection or coarse-grained eg a separate class for each arrival interface, or a separate class for each designated internal subnet.

Suppose for a moment that all packets are the same size; this makes fair queuing much

easier to visualize. In this (special) case sometimes called Nagle fair queuing. the router simply services the per-class subqueues in round-robin fashion, sending one packet from each in turn. If a per-class subqueue is empty, it is simply skipped over.

Fair queuing was extended to streams of variable-sized packets. A straightforward extension of fair queuing is weighted fair queuing (WFQ), where instead of giving each class an equal share, we assign each class a different percentage. For example, we might assign bandwidth percentages of 10%, 30% and 60% to three different departments. If all three subqueues are active, each gets the listed percentage. If the 60% subqueue is idle, then the others get 25% and 75% respectively, preserving the 1:3 ratio of their allocations. If the 10% subqueue is idle, then the other two subqueues get 33.3% and 66.7%.

There are another two another algorithms which achieves the state-of-art:*Bit by Bit Round Robin* and*The GPS model*. The fisrt is a straightforward extension of the round-robin idea and the second is a straightforward extension of the round-robin idea.

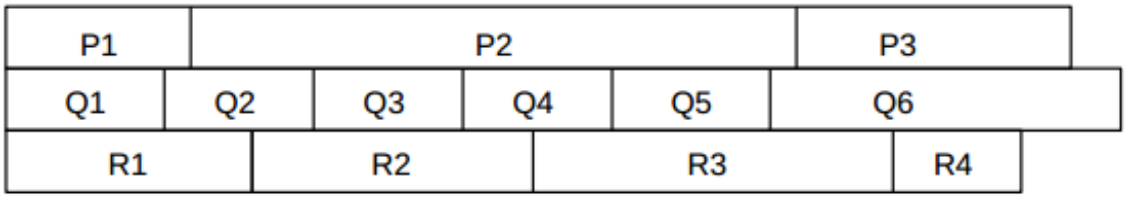
Outline

In our work, we first consider three queues share the same bandwidth. At last, we extend our algorithm that can realise fair queuing even if the three queues request different bandwidths.

Our Alogrithm

To make our algorithm easy to understand, we start with a simplification: let us assume that each per-class subqueue is always active, where a subqueue is active if it is nonempty whenever the router looks at it. If each subqueue is always active for the equal-sized-packets case, then packets are transmitted in order of increasing (or at least nondecreasing) cumulative data sent by each subqueue. In other words, every subqueue gets to send its first packet, and only then do we go on to begin transmitting second packets, and so on. Still assuming each subqueue is always active, we can handle different-sized packets by the same

idea. For packet P, let C_P be the cumulative number of bytes that will have been sent by Ps subqueue as of the end of P. Then we simply need to send packets in nondecreasing order of C_p. A completely equivalent strategy, better suited for generalization to the case where not all subqueues are always active, is to send each packet in nondecreasing order of virtual finishing times, calculated for each packet with the assumption that only that packets subqueue is active. The virtual finishing time F_P of packet P is equal to C_P divided by the output bandwidth. We use finishing times rather than starting times because if one packet is very large, shorter packets in other subqueues that would finish sooner should be sent first.



[1] Packet transmission order: Q1, P1, R1, Q2, Q3, R2, Q4, Q5, P2, R3, R4, P3, Q6

Scheme 1: Variable-packet-sized fair queuing with all subqueues active

This ensures that, in the long run, each subqueue gets an equal share of bandwidth.

A completely equivalent strategy, better suited for generalization to the case where not all subqueues are always active, is to send each packet in nondecreasing order of virtual finishing times, calculated for each packet with the assumption that only that packets subqueue is active. The virtual finishing time F_P of packet P is equal to C_P divided by the output bandwidth. We use finishing times rather than starting times because if one packet is very large, shorter packets in other subqueues that would finish sooner should be sent first.

Definition

1. t: the real time of the system and events(packets arrive and leave) occur as the time proceeds.

2. F_p: the packet p's virtual finishing time. F_{prev}: the last packet's virtual finishing subqueue when a new packet enters into this queue.

Procedure

1. When a packet arrives, calculate its virtual finish time and the formula is as follows:

$$F_p = \min(F_{prev}, t) + packet\ size \tag{1}$$

2. When one packet is transferred, then we choose the packet which possess the minimum virtual time. The packet with the minimum virtual finishing time is transmitted.

Program

```
Shared variables
    const N                 // Nb of queues
    queues[1..N]            // queues
    lastVirFinish[1..N] // last virtual finish instant


receive(packet)
     queueNum = chooseQueue(packet)
     queues[queueNum].enqueue(packet)
     updateTime(packet, queueNum)



updateTime(packet, queueNum)
    // virStart is the virtual start of service
    virStart = max( now(), lastVirFinish[queueNum] )
    packet.virFinish = packet.size + virStart
    lastVirFinish[queueNum] = packet.virFinish


send()
```

```
    queueNum = selectQueue()
    packet = queues[queueNum].dequeue()
    return packet



selectQueue()
    it = 1
    minVirFinish =\infty
    while (it <= N) do
      queue = queues[it]
      if( not queue.empty and queue.head.virFinish < minVirFinish )
        minVirFinish = queue.head.virFinish
        queueNum = it
      it = it + 1
    return queueNum
```

The algorithm above is a fair queuing algorithm in the premise that three queues have the same bandwidth, In our experiment, we will see the average waiting time is merely equal. And this algorithm has very good flexibility. That's because to solve the problem in which different queues possess different percentage of bandwidth, we can simply change the equation:

$$F_p = \min(F_{prev}, t) + packet\ size \tag{2}$$

to

$$F_p = \min(F_{prev}, t) + (packet\ size)/\alpha_i \tag{3}$$

where α_i is the percentage of bandwidth that the subqueue i should get.

Experiment

Our methods

1. WFQ(weighted fair queuing) The detailed explanation of this algorithm is above

2. MWFQ(modified weighted fair queuing) This algorithm can realise fair queuing even the three queues is allocated different percentage of bandwidth whether the arrival rate is the same or not.

Assumption

1.Arrival obeys Poisson distribution.

2.Service time is subject to exponential distribution.

Results

We set the system's process rate is 100/s

Table 1: WFQ $\lambda = 30, 30, 30$,bandwidth=$1/3, 1/3, 1/3$

	AveWaittime	AvequeueLen
queue1	0.0475	1.518
queue2	0.0482	1.536
queue3	0.0478	1.525

Table 2: WFQ $\lambda = 20, 30, 40$,bandwidth=$1/3, 1/3, 1/3$

	AveWaittime	AvequeueLen
queue1	0.0108	0.303
queue2	0.0445	1.424
queue3	0.2408	7.982

Table 3: MWFQ $\lambda = 30, 30, 30$,bandwidth=$0.25, 0.35, 0.40$

	AveWaittime	AvequeueLen
queue1	0.265	6.559
queue2	0.0368	0.1.2317
queue3	0.01998	0.7371

Table 4: MWFQ $\lambda = 20, 30, 40$,bandwidth=$0.25, 0.35, 0.40$

	AveWaittime	AvequeueLen
queue1	0.0337	0.789
queue2	0.0367	1.227
queue3	0.0672	2.657

Table 5: MWFQ $\lambda = 20, 30, 40$,bandwidth=$0.40, 0.35, 0.25$

	AveWaittime	AvequeueLen
queue1	0.0037	0.1187
queue2	0.0023	0.7949
queue3	2067.19	41335.35

Table 6: MWFQ $\lambda = 30, 30, 30$,bandwidth=$0.32, 0.33, 0.34$

	AveWaittime	AvequeueLen
queue1	0.056	1.750
queue2	0.047	1.508
queue3	0.041	1.356

Table 7: MWFQ $\lambda = 28, 28, 28$,bandwidth=$0.30, 0.34, 0.36$

	AveWaittime	AvequeueLen
queue1	0.06	1.79
queue2	0.033	1.069
queue3	0.026	0.886

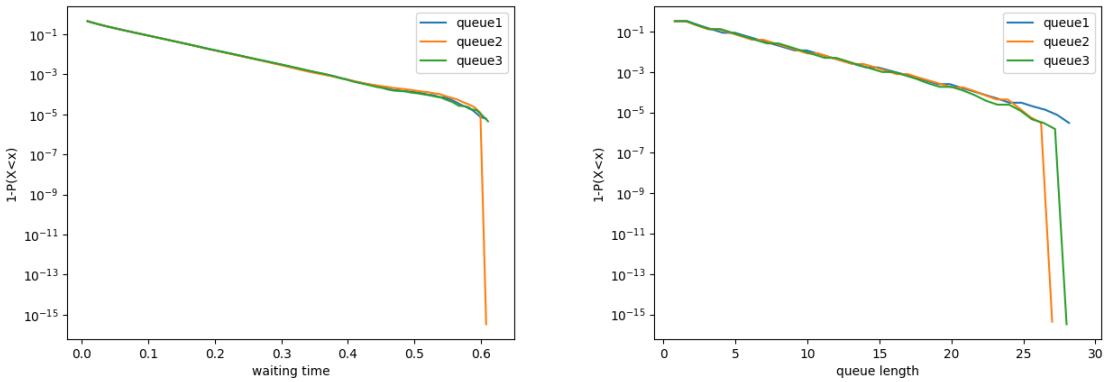


Figure 1: FIFO $\lambda = 30, 30, 30, Bandwidth = 1/3, 1/3, 1/3$

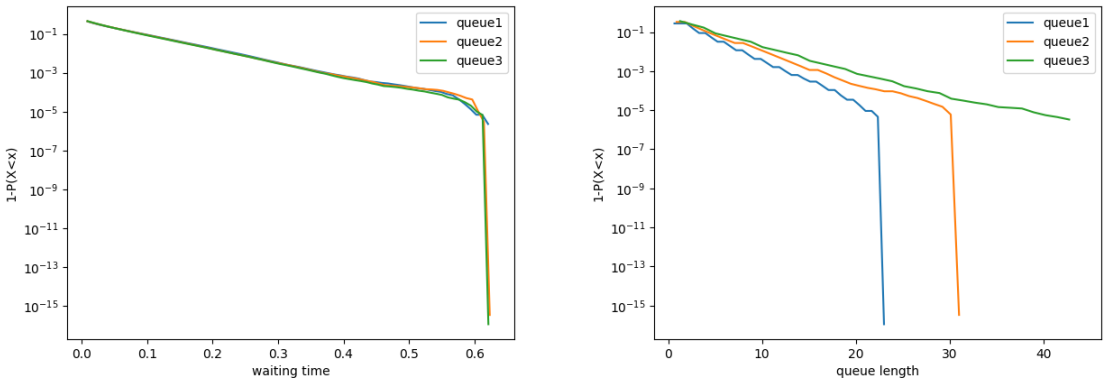


Figure 2: FIFO $\lambda = 20, 30, 40, Bandwidth = 1/3, 1/3, 1/3$

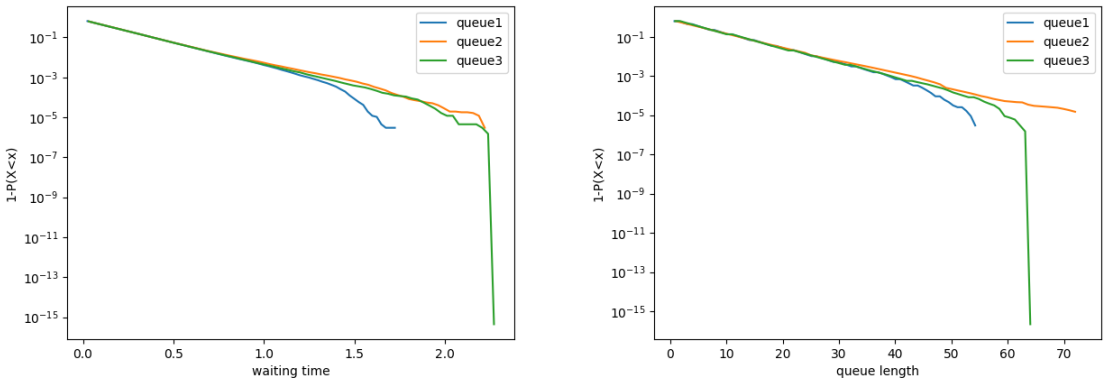


Figure 3: WFQ $\lambda = 30, 30, 30, Bandwidth = 1/3, 1/3, 1/3$

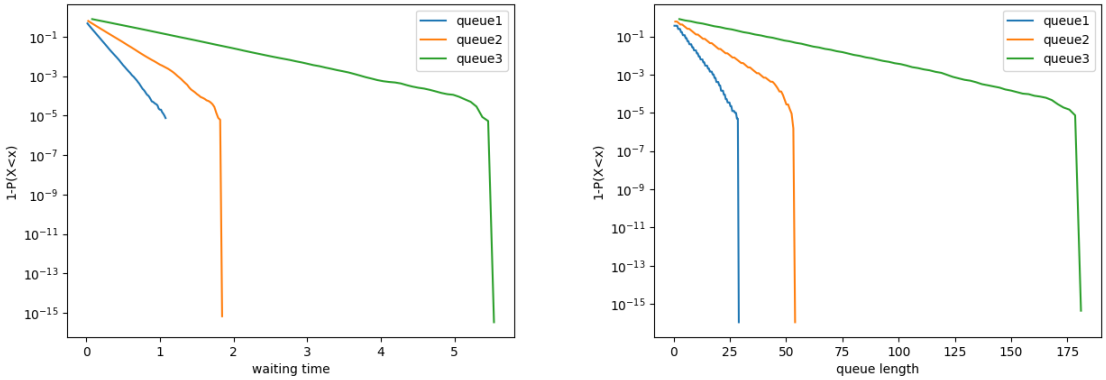


Figure 4: WFQ $\lambda = 20, 30, 40, Bandwidth = 1/3, 1/3, 1/3$

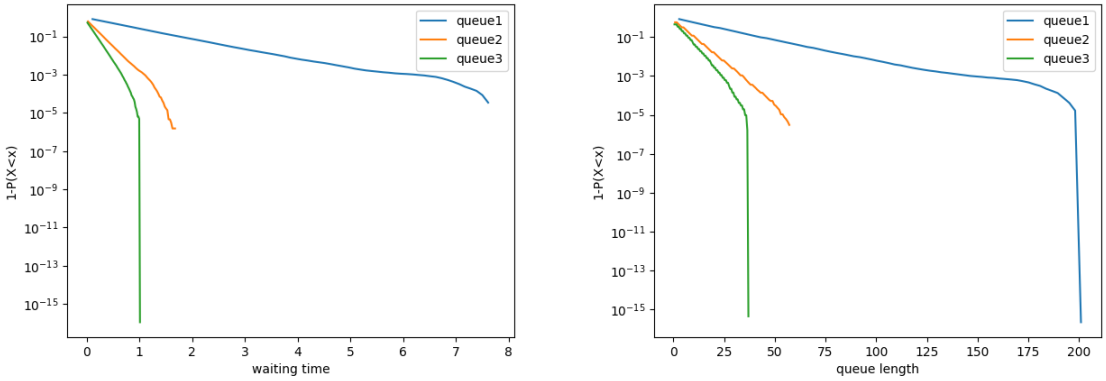


Figure 5: MWFQ $\lambda = 30, 30, 30, Bandwidth = 0.25, 0.35, 0.45$

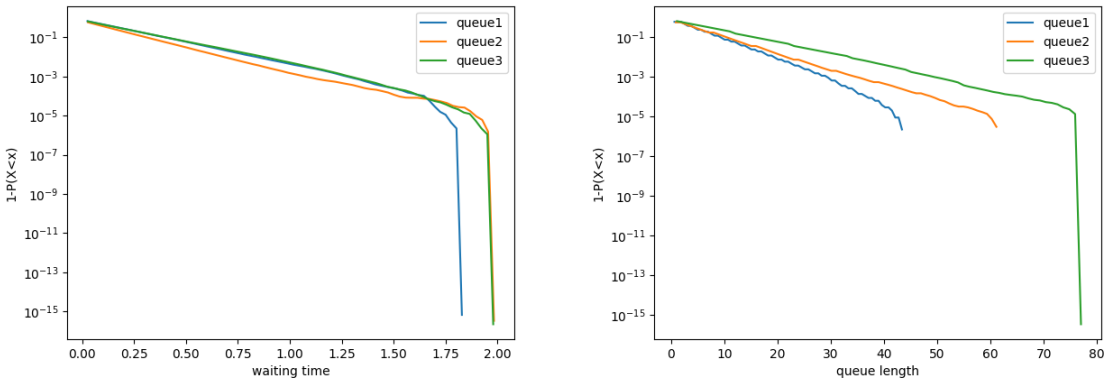


Figure 6: MWFQ $\lambda = 20, 30, 40, Bandwidth = 0.25, 0.35, 0.40$

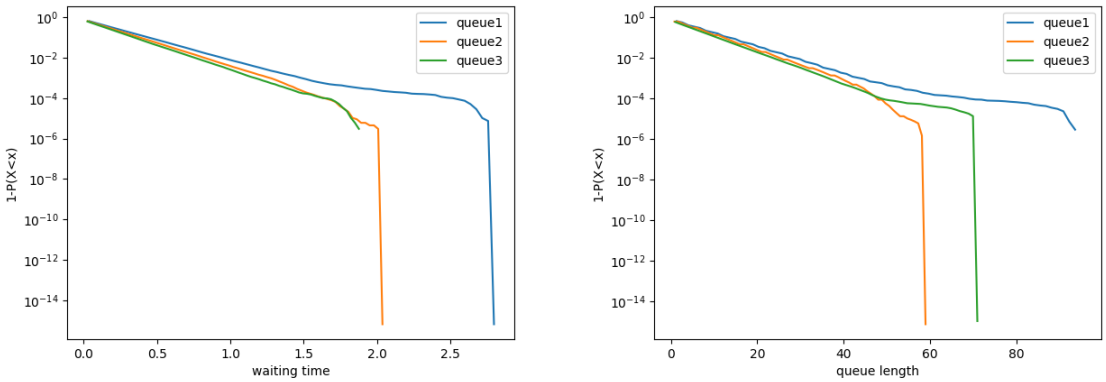


Figure 7: MWFQ $\lambda = 30, 30, 30, Bandwidth = 0.32, 0.33, 0.34$

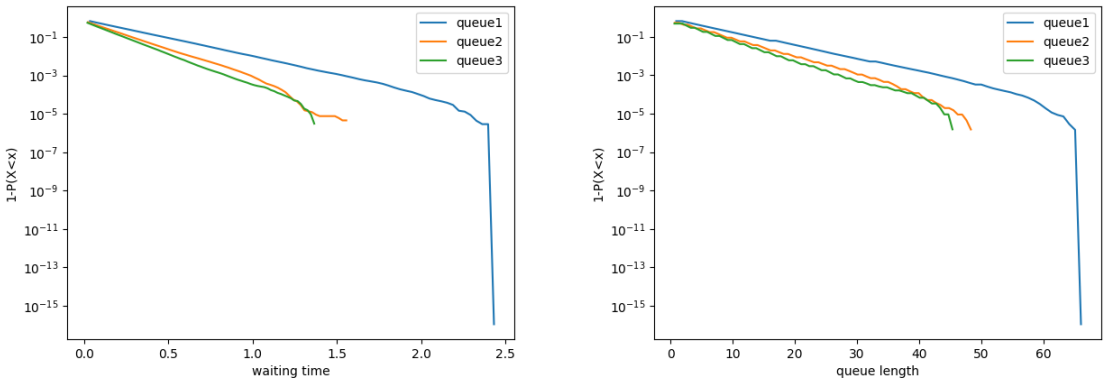


Figure 8: MWFQ $\lambda = 28, 28, 28, Bandwidth = 0.30, 0.34, 0.36$

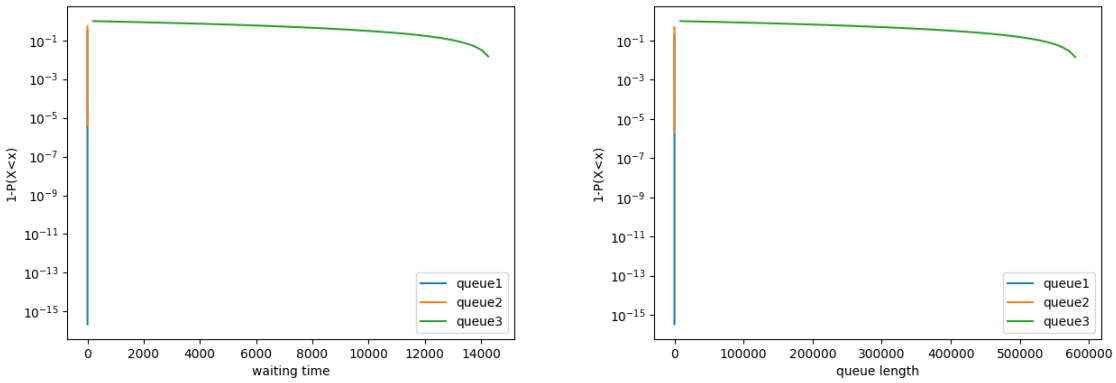


Figure 9: MWFQ $\lambda = 20, 30, 40, Bandwidth = 0.40, 0.35, 0.25$

Analysis

Figure1: We can tell that when the arrival rates of the three queues are equal, the FIFO algorithm is relatively fair because the distribution of Waiting time and queue length are almost coincident.

Figure2: When the arrival rate of three queues are not equal, one amazing phenomenon is that the distribution of waiting time is still almost coincident while the distribution of queue length is not coincident. This phenomenon can be explained as follows: For example, the arrival rate of three queues is 1:2:3, we can imagine that in the unit time one packets arrive in queue1, two packets arrive in queue2, three packets arrive in queue3, we regards them as one packet. This can tell why the distribution of waiting time is almost coincident.

Figure3: From this figure, we can tell that our algorithm WFQ works well when the arrival rates of three queues is equal.

Figure4: It is clear that the wait time of queue1 and queue2 is dramatically smaller than queue3, that's because queue3's arrival rate is 40 while the service rate is 33, so it will be congestion.

Figure5: This figure illustrates the trend when three queues get different percentage of bandwidth under the premise of the same arrival rate. The service rate queue1 get is smaller than queue1's arrival rate, so the waiting time is relatively high.

Figure6: The service rate three queues get is nearly equal to the arrival rate respectively, so the distribution of waiting time and queue length are almost coincident .

Figure7 and Figure8: The service time is higher than the arrival rate, The curve is relatively flat.

Conclusion

In this paper, Firstly, we realize **Weighted Fair queuing** algorithm, which can ensure fairness on the condition of three different queues share the same bandwidth despite different

arrival rate. And then, we modify this algorithm to accommodate the situation that different queues can possess different bandwidth. According the above analysis, this algorithm is relatively fair than the FIFO algorithm, because of the complexity of maths formula, we do not give the strict prove, but it is close to the GPS algorithm.