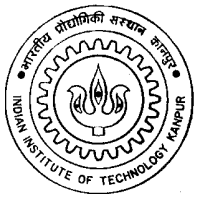
Packet Scheduling Algorithms to Support QOS in Networks

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by

Srikar B S



to the

Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

October, 1999

Certificate

This is to certify that the work contained in the thesis entitled "*Packet Scheduling Algorithms to Support QOS in Networks*", by *Srikar B S*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

October, 1999

(Dr. Dheeraj Sanghi)
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

Acknowledgements

I would like to express my gratitude to Dr.Dheeraj Sanghi for his invaluable guidance and inspiration throughout my thesis work. In addition to his support and flexibility, I would like to thank him for his patience during periods of slow progress. I am very happy to have had the opportunity to learn from and interact with Dr.Sanghi and other professors of the department. I am grateful to them for improving my knowledge, my primary motive in coming to IITK.

It was great to be in the company of all my class mates. It was a pleasure studying, discussing, arguing, travelling, etc., with friends, especially Uma, Gopi, Prasanna, Sridhar, Girish, Hiran, Subhash, Prasad, Vasu, ... Kanpur being in a strategic position it was a wonderful oppurtunity to visit places like the Himalayas and Varnasi, Gopi always being ready in initiating the plans for the trips. I was very lucky to have the company of Vasu and Prashant during times when I felt lonely towards the later stage of my work. Thanks to the juniors who were understanding when I was eating into their lab space and resources.

I would like to thank all my Kannada Sangha friends. The faculty members and their family provided a homely atmosphere and respite from the hall mess with delicious food. We had a wonderful time in the company of Arvind, Karthik, Katti, Suresh, Puttaraju, Frederick, Anand, Narayan, Shyam and others.

Finally, thanks to my parents, whose blessings are always with me, grand parents, brothers, sis-in-law, and all my close relatives for their love and care.

Abstract

Traditionally packet switching networks have supported only best-effort traffic. Newer applications need communication services that allow end clients to transport data with performance guarantees given in terms of delay, delay variation, bandwidth, and loss rate. The choice of the packet scheduling algorithm to be used at switching nodes is very crucial to provide the quality of service

We have conducted a literature survey covering various scheduling disciplines that can be used to provide performance guarantees to clients. In this thesis, we have compared select scheduling disciplines in an experimental study using simulation. Weighed fair queueing, Class based queueing, and Rate-controlled static priority queueing, are the scheduling disciplines chosen for the study. The traffic types used in the study are CBR, VBR, ABR and UBR flows. For real-time flows used in the experiments, traffic source models for audio and video traffic were used as candidates for CBR and VBR flows. An extension of RCSP for using a new traffic model has been designed and experimented. We have also studied the use of RCSP for scheduling real-time flows in a link-sharing paradigm that uses CBQ.

Several experiments were conducted using the simulation test-bed for comparing the performance of the scheduling disciplines when serving various traffic mixes of CBR, VBR, ABR, and UBR flows. The conclusions from these experiments have been presented.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

There have been vast improvements in computer and communication technologies. Computers using faster processors along with high speed, high bandwidth networks introduce opportunities for a whole new range of network applications.

Traditional computer networks have supported applications like remote terminals, file transfer, and exchange of text messages(e-mail). In today's networks there is a need to support multimedia applications like video conferencing, Internet telephony and broadcast audio and video. For these applications, data is encoded and transmitted as packets, and delays in packet arrivals at the receiver introduce gaps into the decoded output, adversely affecting the quality. Hence, in addition to bandwidth, these applications require guarantees on packet delay and delay variations. If a packet reaches too late, it is as good as lost.

Another issue is that of jitter control. Jitter is used to define the distortion introduced into a traffic pattern along the path to the destination. In the literature, two definitions of jitter are used, rate jitter and delay jitter. Rate jitter is defined to be the maximum number of packets received in a averaging interval. Delay jitter is defined to be maximum difference between the delays experienced by any two packets on the same connection. While delay jitter captures the maximum delay variation, the rate jitter parameter captures the maximum length of a burst of data that may need to be stored before being decoded at the receiver.

The requirements for real-time traffic are summarized as follows

- Minimum bandwidth guarantee
- Maximum bound on packet delay

- Maximum bound on rate jitter or delay jitter
- Maximum bound on packet losses

While a circuit switched network can provide performance guarantees, it is very inflexible and wasteful of network resources. Packet switching on the other-hand provides a flexible environment for sharing of network resources. Traditionally, data traffic was treated as best-effort traffic, hence was easily accommodated into packet switching networks. Real-time traffic requirements along with the need to utilize network resources effectively provides new challenges in the design of packet switching networks.

The approach to provide real-time services in packet switching networks can be briefly outlined as follows. When a real-time channel is to be established a request is sent to the underlying network about the requirements of the channel. The network cannot make these guarantees unless it has some idea on characteristics of the traffic arrival pattern. While it difficult to give an exact characterization of the traffic, traffic specifications can provide some information on the nature of the traffic. The specifications include an average packet arrival rate over an averaging interval. In addition to average rate some specifications give a peak rate, or a maximum burst size. The network uses an connection admission control procedure to determine if it can indeed satisfy the requirements of the channel without affecting the performance guarantees promised to existing real-time channels. The network then commits necessary resources to the real-time channel.

In a packet switching network, packets from different connections will interact with each other at each switch; without proper control, these interactions may adversely affect the performance experienced by clients. The scheduling disciplines at the switching nodes, which control the order in which packets are serviced, determine how packets from different connections interact with each other.

Using simple FIFO scheduling at routers would not be suitable for providing performance guarantees to flows as it would lead to very inefficient use of network resources. New scheduling disciplines have been designed with an aim at providing different quality of service to different connections. While these disciplines may have been individually tested upon to experimentally verify their requirements, our goal in this thesis is to provide a common platform for comparing the performance of the scheduling disciplines. We use a simulation test-bed in which these scheduling disciplines are implemented for scheduling packets at routers. Modeled audio and

video sources are used for generating real-time traffic. We study the performance of the scheduling disciplines in various scenarios when servicing different traffic mixes.

A survey of various scheduling disciplines is presented in Chapter 2 . In the Chapter 3 we discuss about the network simulator that has been used for studying the scheduling disciplines. We present the implementations of existing scheduling disciplines and the ones that have been added, and discuss modeling of video sources. In Chapter 4 we present results of the experiments that have been conducted to compare the performance of the scheduling disciplines. We conclude with Chapter 5.

Chapter 2

Scheduling Disciplines

An extensive survey of scheduling disciplines was conducted. In this chapter we present various scheduling disciplines which can be used to provide performance guarantees to flows.

2.1 VirtualClock

Lixia Zhang [23] proposed the VirtualClock algorithm designed for data traffic control in high speed networks. The algorithm maintains the statistical multiplexing flexibility of packet switching while ensuring each data flow its reserved average throughput rate at the same time. End-to-end delay guarantees can be provided to sources that are Leaky Bucket [18] constrained.

The algorithm supports diverse throughput requirements from various applications. It monitors flows, provides measurement input to other network control functions, and builds firewalls among flows such that their interaction at the queue does not affect individual flows.

The basic idea behind the algorithm was inspired by the Time Division Multiplexing (TDM) system. In a TDM system, channels can transmit during specific slots of time, which are assigned to them based on a fixed bandwidth allocation. In such a system, if a flow has no data to send during its slot the capacity is wasted. The algorithm uses a virtual clock instead of the real clock, and tries to retain the advantage of statistical multiplexing in packet switching, but make possible average rate allocations with firewalls between flows.

Using the idea of TDM, the algorithm maintains a VirtualClock for each data flow. The VirtualClock, $VirtualClock_i$, assigned to $flow_i$ ticks at every packet arrival. A flow that is assigned a rate, AR_i, ticks at a rate proportional to $1/AR_i$, based on the packet size, provided that the sources packets arrive according to the specified rate. Packets are stamped with the $VirtualClock_i$ value, and transmitted in the order of the assigned time stamps. This system is work conserving as it merely orders packets based on time stamps, and picks packets for service as long as resources are available.

A flow's VirtualClock acts as a meter driven by packet arrivals, and can be used to monitor flows. If the $VirtualClock_i$ is growing faster than the real clock then the flow is sending packets faster than the specified rate, and if $VirtualClock_i$ is less than the real clock the flow is sending packets slower than the specified rate. Flows can be monitored at specified time intervals(AI_i) to check if they are sending too fast. While choosing a very small value for the interval would mean complete lack of flexibility for a flow to send at momentarily higher rates, choosing a larger value would allow a flow to send too many packets before the interval expires. To overcome the difficulty in arriving at an appropriate value for a time interval it is chosen to monitor a flow once every $AIR_i(AI_i \times AR_i)$ number of packets have arrived.

A flow exceeding its specified rate can manage to send as many as AIR_i packets without being considered as misbehaving. Once it has sent AIR_i packets, if $VirtualClock_i$ is greater then real time then the flow would be noted as misbehaving. As packets are ordered according to time stamps, the VirtualClock of a misbehaving flow will run too fast and its packets will find themselves at the end of the service queue. In addition, some control measures need be taken. For example, a certain number of control messages can be sent to each misbehaving flow requesting it to slow down, but if no change is observed in the behaviour the switch may simply choose to stop giving priority treatment to these flows by removing them from the flow table.

If after sending AIR_i packets, $VirtualClock_i$ is going too slow, it is reinitialized to the real clock value. This is needed to prevent flows from accumulating too many credits, having a potential to send big bursts, and posing danger to other flows.

Simulation experiments had shown that flows which remain idle for some time can send a burst and cause an increase in queueing delays for other flows. This can happen before AIR_i number of packets are received from a flow. To overcome this scenario an auxiliary virtual clock $auxVC$ is used. There is one auxiliary virtual clock for each flow. The $auxVC$ is used to time stamp the packets, whereas, the VirtualClock

of individual flows continue to meter individual flows. The difference between the two is that while a slow VirtualClock is reset to real time only after receiving AIR_i packets, the $auxVC$ is maintained to be above real time upon receiving every packet. With respect to monitoring of flows, VirtualClock, as has been discussed, allows some flexibility for flows to send packets at higher rates for short durations by accumulation of credits. However, using $auxVC$ to timestamp packets would ensure that while flows can still accumulate credits, such accumulations will not affect the queueing delays of other flows.

The mechanism used to provide priorities to flows is by giving a lead of a certain amount P to its VirtualClock. Hence, packets will be stamped with a smaller $VirtualClock_i$ value and will be chosen prior to other packets.

The algorithm can be summarized as follows:

- Upon receiving the first packet from $flow_i$, $VirtualClock_i \leftarrow$ real time.
- Upon receiving each packet from $flow_i$,
  1. $auxVC \leftarrow \max(\text{real time}, auxVC)$.
  2. $VirtualClock_i \leftarrow VirtualClock_i + Vtick$, $auxVC \leftarrow auxVC + Vtick$
  3. Stamp the packet with $auxVC$ value.
- Transmit packets by the order of increasing time stamp values.
- Upon receiving each set of AIR_i packets from $flow_i$

  if ($VirtualClock_i -$ real time) $>$ threshold, control actions should be taken.

  if ($VirtualClock_i <$ real time), $VirtualClock_i \leftarrow$ real time.

The VirtualClock algorithm provides a mechanism for allocating bandwidth to flows. Its design ensures that flows do not suffer due to misbehaving sources. Leaky Bucket constrained sources can be guaranteed a worst-case end-to-end delay.

2.2 Weighted Fair Queueing

With uniform processor sharing, at a given instant, there is a set of N non-empty FIFO queues waiting to be served. During any time interval the server serves all

N packets at the head of these queues simultaneously, each at a rate of $1/N$th of the link speed. Generalized Processor Sharing(GPS) is a generalization of uniform processor sharing which allows different connections to have different service shares. Fluid Fair Queueing(FFQ) is an alternative name for GPS. To understand how FFQ works one can image the link as a pipe in which different flows are assigned specific fractions of the pipe's cross sectional area. The total cross sectional area is the link's bandwidth r. The fraction of area reserved for a flow i depends on the weight, ϕ_i, assigned to the flow. The fraction of the area changes dynamically as the set of flows currently in transit change. The area is shared among the flows currently in transit in proportion to their assigned weights. The generalized processor sharing scheme is work conserving. A session in transit is said to be backlogged at time t if it has a packets queued for service. Let $S_i(\tau, t)$ be the session i traffic served in the interval $(\tau, t]$. Under the GPS scheme we have

$$\frac{S_i(\tau, t)}{S_j(\tau, t)} \geq \frac{\phi_i}{\phi_j}, j = 1, 2, \cdots, N \tag{1}$$

for any session i that is continuously backlogged in the interval $(\tau, t]$.

Summing over all sessions j:

$$S_i(\tau, t) \sum_j \phi_j \geq (t - \tau) r \phi_i \tag{2}$$

hence a session i is guaranteed a rate of

$$g_i = \frac{\phi_i}{\sum_j \phi_j} r \tag{3}$$

If we define r_i to be the average rate of session i, then as long as $r_i \leq g_i$, the session can be guaranteed a throughput g_i. Session i's backlog will be cleared at a rate $\geq g_i$. The delay of a session i bit arriving at a time t can be bounded as a function of session i's queue length. These throughput and delay bound guarantees are independent of the queues and arrivals of other sessions. Assigning real numbers as needed in choosing the values of ϕ_i gives good flexibility of resource allocation.

Though GPS has these attractive properties, it is, however, not amenable to implementation. GPS works on the assumptions that flows can be infinitely divisible, and that multiple flows can be simultaneously serviced. Both these assumptions do not hold in practice where only one session can be serviced at a time, and an entire packet is to be served before serving another packet. Demers, Shenker and

Keshav first proposed an approximation of GPS called Weighted Fair Queueing(WFQ) which is practically implementable. The same scheme was further studied by Parekh and Gallager [16], [17], under the name of Packet-by-packet GPS(PGPS), in the context of integrated services networks. Their main contribution was in combining the mechanism with Leaky Bucket admission control in order to provide performance guarantees in terms of both throughput and delay. The PGPS scheme is flexible in the range of throughput and delay guarantees it can offer to flows while maintaining the work conserving nature of the GPS scheme.

The PGPS scheme is based on the time the packets finish service under the GPS scheme. Let F_p be the time when a packet p finishes service under the GPS scheme. The PGPS is an approximation of the GPS scheme that services packets in the order of increasing F_p. However, it is possible that by the time the server is free to pick the next packet for service, the packet that would have the next smallest F_p under the GPS scheme may not have arrived. Consequently, the scheme cannot be both work conserving, and serve in increasing order of F_p. According to the PGPS scheme, the server picks the next packet to finish service under GPS if no packets were to arrive after it.

Despite this shortcoming the PGPS approximation has been shown [16] to be very close in behaviour to the GPS scheme. The following points have been proved.

1. For all packets waiting to be served at any time τ, the order in which the packets will finish service under the GPS and the PGPS scheme will be the same.

2. Let $\hat{F}_p$ be the time at which packet finishes service under PGPS. Then, $\hat{F}_p$ - $F_p \leq \frac{L_{max}}{r}$, where L_{max} is the maximum packet length. The service completion time for a packet under the PGPS scheme never lags behind that in case of the GPS scheme by a value more than the time it takes to service a maximum size packet.

3. Let $\hat{S}_i(t,\tau)$ be the amount of session i traffic in bits served under PGPS. For all times τ and all sessions i, $S_i(0,\tau) - \hat{S}_i(0,\tau) \leq L_{max}$. For the amount of traffic served under the two schemes, the PGPS scheme never lags behind the GPS scheme by a value more than the maximum size packet. Consequently, the backlog in case of PGPS will never exceed the backlog in case of GPS by a value more than the maximum size packet.

2.2.1 VirtualClock versus WFQ

Both the VirtualClock algorithm and the WFQ algorithm use time stamping to order the packets to be served. While the VirtualClock algorithm assigns time stamps to packets based on a static TDM system, the WFQ algorithm assigns time stamps based on the GPS scheme. When using the VirtualClock algorithm, the entire arrival history of a session is summarized by the variable $auxVC$. In such a setup, once a connection misbehaves, it may be punished regardless of whether such a misbehaviour affects the performance of other connections.

Consider an example where two sessions, S1 and S2, are being served, and both are allocated equal rates. Let S2 be idle for an initial duration when S1 sends at twice the allocated rate. In the case of using the VirtualClock algorithm, since only packets of S1 are served, the $auxVC$ of S1 ticks at twice the rate of real clock while the the $auxVC$ of S2 ticks at the rate of the real clock. When packets of S2 start arriving, $auxVC$ of S1 will be ahead of real clock, and S1's packets will be punished as higher values of the time stamps assigned to them will put them behind in the service queue. Hence, S1 being served at a higher rate did not affect S2, nevertheless, it is punished for utilizing unused bandwidth. In case of WFQ, the finish time of a packet, on its arrival, is calculated based only on the sessions that currently have a backlog. Hence, when S2 is idle, S1 can occupy the complete bandwidth, but once S2 starts sending, the bandwidth is shared equally among the two sessions.

2.2.2 WF^{2}Q

GPS based schemes have been used in the context of feedback based congestion control. A source constantly samples feedback from receiver in order to check for symptoms of network congestion. The source controls the rate at it which it admits packets into the network by reacting appropriately to these symptoms. Misbehaving sources might try to take advantage of network resources by sending packets disregarding, or otherwise simply ignoring symptoms of congestion. Fair allocation of bandwidth at queueing points would ensure that such misbehaving sources do not hog up the network resources. The GPS discipline offers fair allocation of bandwidth and protection from misbehaving sources. Robust congestion control algorithms can be built based on the more accurate measurement and protection provided by a GPS like servicing discipline.

It has been shown that delay of any packet in the PGPS scheme as compared to its delay in the GPS discipline is no greater than the transmission time of one packet. In terms of service rate, the PGPS discipline does not fall behind the corresponding GPS discipline by more than one max packet size. PGPS was considered as the best way of approximating the GPS scheme. It was later found by Bennet and Zhang [1], contrary to popular belief, that large discrepancies can occur between behaviour of the GPS scheme and the PGPS scheme. They found that while no flow can lag behind too much, but particular flows can be significantly ahead.

An Example

To understand difference in the behaviour of the two systems, we can consider the example in Figure 2.1. At time 0, there are six sessions which have backlogs to be serviced by the queueing discipline. Assume that the link serves unit size packets at a rate of 1bps. Session 1 is guaranteed a rate of 0.5, and the remaining 5 sessions are guaranteed a rate of 0.1. Let the arrival pattern of the packets be as shown in Figure 2.1. Session 1 arrival pattern shows 6 back to back packets, and the remaining sessions have one packet in their respective queues.

According to the GPS scheme, for the given arrival pattern, packets of Session 1 finish service at times 2, 4, 6, 8, 10 and 11, and packets of the remaining sessions finish service at time 10. PGPS picks packets for service in the order of their finishing time in the GPS discipline. The initial 5 packets of Session 1, having a finishing time less than or equal to the finishing time of packets from other sessions, are chosen by PGPS to be served back-to-back. However, the 6th packet, having a finishing time after the other packets, is sent following the packets of the other sessions. There is a conflict in serving the packets of other sessions as they have the same finish times. Resolving this contention by giving higher priority to sessions with a lower session number gives a service order as shown in Figure 2.2. It can be noticed that service completion time for packets of session 1 under the PGPS discipline is far ahead when compared to that of the GPS discipline. In addition to this, a big gap results between the end of transmission of the 5th packet and the begin of transmission of the 6th packet.

In a simple feedback based scheme two back-to-back packets are sent and the difference in the arrival times of their acknowledgements is used to gauge the bandwidth available to a session. Keshav [14] proposed a packet-pair algorithm that uses such

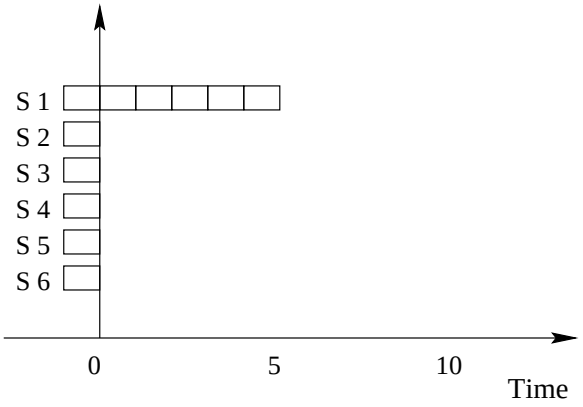


Figure 2.1: Packet Arrivals

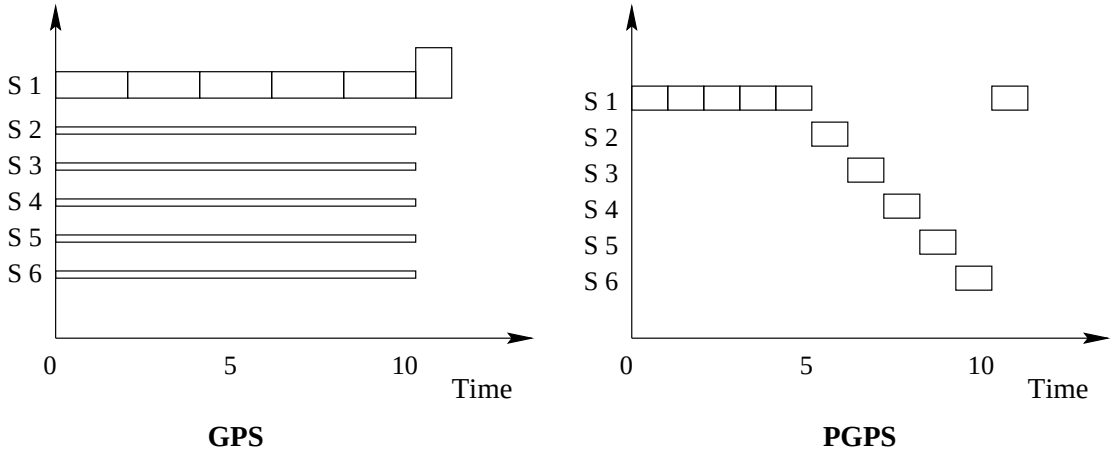


Figure 2.2: Service Order when using GPS and PGPS

this technique. This kind of an algorithm assumes that switches implement a packet service algorithm like GPS. However, if the PGPS approximation of GPS is employed, a session's service rate may oscillate between zero and full link speed. This is likely to cause instability of the source control algorithm. In the previous example, if the packet-pair were among the first five packets, they would be serviced one immediately after another(difference in the service completion time is 1). On the other hand if the 5th and 6th packets constituted the packet pair there would be a large difference of 6 time units between the service times of the packets.

The New Approach

Bennet and Zhang [1] proposed a new approximation to the GPS service discipline called Worst-case Fair Weighted Fair Queueing(WF^2Q). This service discipline shares both the bounded delay and the worst case fairness properties of the GPS discipline. They noted that the problem in PGPS is due to the fact that service of a packet can start earlier than its start time in the GPS. While PGPS selects the next packet to service among all available packets, the WF^2Q scheme selects the next packets to service among a subset of the available packets. When picking the next packet for service, WF^2Q considers only the set of packets whose service would have started in the corresponding GPS scheme. Among the packets in this set it chooses that packet which will finish service first under GPS as the next packet to be serviced. The service order for the sessions with arrival pattern as shown in Figure 2.1 will be as shown in Figure 2.3.

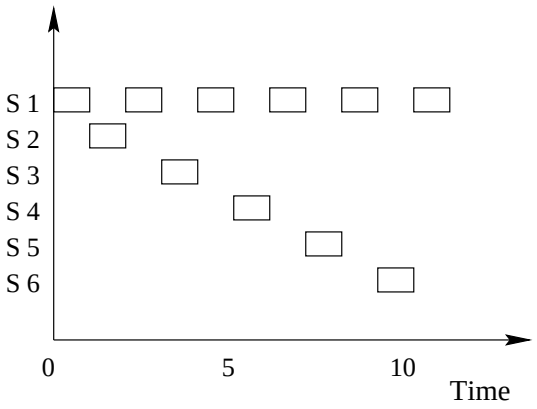


Figure 2.3: WF^2Q Service Order

Many useful properties of PGPS are retained by WF^2Q. Like in the case of PGPS, the worst-case delay bounds for packets in fluid GPS and WF^2Q system differ by no more than the time to service a single packet of maximum size. For any session, the service rate in terms of the bits served by WF^2Q does not lag behind the fluid GPS system by any value greater than the maximum packet size. Consequently, the backlog of any session will not exceed its backlog in GPS by a value greater the maximum packet size. In addition to these properties, while PGPS can be quite ahead of the GPS system, WF^2Q cannot go ahead of GPS by more than a fraction of the maximum packet size. Since the service provided can be neither far behind nor too far ahead, WF^2Q provides a service almost identical with GPS system.

To summarize, in this section we have presented the ideal GPS scheme and its two practical approximations WFQ and WF^{2}Q. Though both schemes have the same worst case delay for packets, WF^{2}Q approximates GPS more closely than WFQ. These schemes can be used to allocate bandwidth to flows, and along with leaky bucket admission control they can also be used to provide end-to-end delay guarantees.

2.3 Deadline Based Scheduling

In a scheme presented for real-time channel establishment in wide area networks, a scheduling discipline based on Earliest Due Date principle was proposed by Ferrari and Verma [6]. A real-time channel between two nodes is characterized by parameters representing the performance requirements of the flow. The proposed scheme provides for establishment of real-time channels giving guarantees to flows in terms of bounds on the minimum data rate, maximum packet delay, and a maximum packet loss rate.

The solution proposed works in conjunction with a channel establishment procedure, scheduling policies, and flow control policies. Clients specify their traffic characteristics using the parameters $(X_{min}, X_{ave}, I, P_{max})$. X_{min} is minimum packet inter-arrival time. X_{ave} is the average packet inter-arrival time over an average interval I. P_{max} is maximum packet size. Channels can be either deterministic or statistical in nature. A deterministic channel expects an absolute source to destination delay bound D, whereas a statistical channel expects the source to destination delay bound of D to be satisfied with a probability greater than Z.

Channel Establishment

A fast channel establishment procedure has been proposed which requires one round trip time of a packet. The goal of the procedure is to break up the end-to-end delay bound D_i of channel i into local delay bounds $d_{i,j}$ at each intermediate node j. The local bounds are computed so that, if node j can assure that no packet will be delayed locally beyond bound $d_{i,j}$, the end-to-end delay bound D_i can be met. A suggested value for the local delay bound is included by each node in the establishment request during its forward trip. The destination does the final allocation of the local delay bounds. It may relax the local delay bounds ensuring that the end-to-end delay bound is still satisfied. A sufficiency check for the buffer space, committed by intermediate

nodes, is also conducted by the destination. Relaxation of the committed buffer space may also be possible. These relaxations can take effect when the node finally commits the resources during the establishment message's return trip. Tests are conducted by the nodes during the request message's forward trip, if it can, indeed, support the requested connection. These tests are to check whether the required bandwidth, delay bounds and buffer space can be provided by the node without jeopardizing the performance guarantees given to already established channels. If tests fail, message can be sent back either to the sender that may choose to wait or try another path, or to an intermediate node that can forward the message through another path. On the return trip, after an unsuccessful test, the nodes that have tentatively reserved the resources can free them for use by further requests.

Scheduling

The scheduling mechanism proposed as a part of the solution is deadline based scheduling. Each arriving packet of a real-time channel is assigned a deadline. The packets are scheduled in the order of increasing deadline. In case of a conflict, priority is give to deterministic over statistical channels. Packets belonging to best effort traffic and other tasks of node have lower priority. Local tasks are pre-emptible by real-time packets. Three queues are maintained at the scheduler. The first queue contains the packets belonging to deterministic channels, the second queue contains packets belonging to the statistical channels. All other types of packets and node activities are in the third queue. Each packet that arrives is given a deadline, which is the time by which it is to be serviced. Let $d_{i,n}$ be the local delay bound assigned to channel i in node n. A packet arriving at time t_0 will be assigned a node deadline equal to $t_0 + d_{i,n}$. The deadline may be reduced if it overlaps with the deadline of some other deterministic packet. The queues are ordered on increasing values of deadlines. When the scheduler is free to select the next packet, it compares the deadline of a packet at the head of the statistical queue with the beginning time(deadline - service time) of the packet at the head of the deterministic queue. If the service begin time of the deterministic packet is less when compared to the service end time of the statistical packet, the deterministic packet is selected for service. If this is not the case, a similar check is done comparing the packet at the head of the statistical queue and the packet at the head of the third queue.

Rate control

In a rate based flow control scheme the sender controls its packet sending rate based on its knowledge of the characteristics of the receiver and those of the channel path. The receiver needs to check whether it will be able to accept packets at the rate declared by the receiver. A malicious user could circumvent this rate based flow control by sending at a higher rate than the declared $1/X_{min}$. This may also be due to some failure at the sending host. Such a scenario may prevent the satisfaction of delay bounds guaranteed to other clients of the real-time service. One solution to this problem is to provide rate control by increasing the deadlines of offending packets to delay them on heavily loaded nodes. If buffer space is limited such packets may also be dropped at the node. The deadline assigned to the offending packet would equal the deadline that the packet would have if it had obeyed the $(X_{min}, X_{ave}, I, P_{max})$ characteristics declared at connection establishment. As a consequence of this rate control scheme, an intermediate node can assume that the clients are obeying the promised traffic specifications even when two packets sent at an interval longer than or equal to X_{min} by the client come close together due to network load fluctuations.

A server employing these mechanisms of rate control, scheduling and admission control will ensure that no packet along a channel will spend more than its delay bound in the node, provided that the channel does not send packets faster than its specified rate. These local delay bounds make sure that packets can be provided corresponding end-to-end delay bounds. It is also ensured that the network nodes can provide buffer space to prevent any packet losses.

2.3.1 Controlling Delay Jitter

The server used for providing real-time service is called a bounded delay server. This server can be extended as suggested in [19], to provide delay jitter guarantees. The new scheme attempts at providing jitter bound to a real-time connection which is equal to the delay bound at the last node in the channel's path. This is achieved by faithfully preserving the original arrival pattern at the last bounded delay node on the channel's path as it was at the entrance of the network. Each node performs two functions, the first function is to reconstruct and preserve the original arrival pattern of the channel, the second function is to ensure not to disturb the pattern too much so that it is possible to be reconstructed at the next downstream node.

Such a reconstruction of arrival pattern can be made by breaking the server into a set of regulators, one per channel and a bounded delay server. The function of the regulator is to absorb delay jitter introduced by the upstream neighbour. Bounding of delay jitter requires the clients to specify the end-to-end delay jitter bound J for the channels packets, in addition to the end-to-end delay and buffer space requirements.

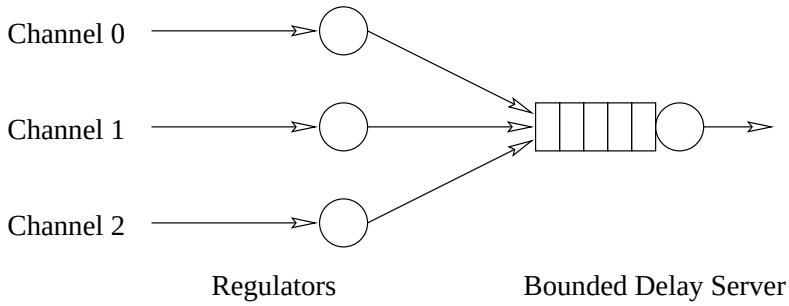


Figure 2.4: The node model for jitter control

The channel establishment in this case is similar to the one previously discussed. Every node offers a suggested value for the local delay bound $d_{i,n}$ and the local delay jitter bound $j_{i,n}$. Usually this value will be equal to the local delay bound. A test is performed during the forward trip of the channel establishment message to obtain the value of the least possible jitter a node can provide. The space requirement check at nodes in this case is different from the previous scheme in that in addition to checking if there is enough buffer space in order to provide the local delay bound, there has to be buffer space to absorb jitter introduced by the upstream neighbour and reconstruct the original arrival pattern. The next check is whether the final node can provide a delay jitter $j_{i,N} \leq J_i$.

The scheduling policy used in the jitter scheme is same as earlier. A new rate control algorithm is used at the newly introduced regulators. The bounded delay server performs an additional task of including in each packet's header the difference between the instant the packet is served and the instant it was supposed to be served (packet's deadline). If a packet arrives at t_0 after experiencing the maximum possible delay at the previous node. The packet is assigned a deadline of $t_0 + d_{i,n}$ and an eligibility time of $t_0 + d_{i,n} - J_{i,n}$. Packets not following the traffic characterization are treated as earlier. The time difference in the packet header is added to the time a packet is to be held by the regulator before being handed to the scheduler. This ensures that the channel will behave as if it experienced a constant delay $d_{i,n}$ at the previous node. As a consequence of the regulators, the pattern entering the scheduler

at every node is the same as the original pattern of packet arrivals in case it has been satisfying the traffic characterization.

The jitter scheme proposed gives a delay bound same as in the previous case but gives the channel a view of constant delay network element with a delay equal to the end-to-end delay bound D_i and the delay jitter bound equal to the delay bound of the last node on the channel's path.

2.4 Time-frame Based Strategies

In this section we consider two scheduling disciplines based on time framing strategies. Both these techniques use the notion of a time frame. Starting from a reference point in time, common to all nodes, the time axis is divided into fixed size periods of length T, each such period is called a frame.

2.4.1 Stop-and-Go Queueing

Golestani [9] proposed a framework based on the stop-and-go queueing to provide loss-free, bounded-delay transmission for real-time channels. The strategy uses the concept of time frames along with a packet admission policy and a service discipline, stop-and-go queueing, at the switching nodes.

The admission policy used in the framework needs a flow to be (r, T) smooth. A flow is said to be (r, T) smooth if in each frame of length T the received packets have a collective length not exceeding $r.T$ bits. Packets violating this limit are not admitted till the end of the current frame. If packets of fixed size Γ are used, a flow is (r, T) smooth if the number of packets that are received is bounded by $\frac{1}{\Gamma} r.T$.

Time frames are viewed as traveling with packets from one end of the link to the other. To each frame as viewed at the beginning of a link called a departing frame there corresponds a frame at the receiving end called arriving frame. Accordingly, if τ_l is the sum of propagation delay of the link and processing delay at the receiving end, arriving frames at the receiving end will be lagging behind their corresponding departing frames at the transmitting end by τ_l.

Assume the case where arriving frames at the receiving end are synchronized with departing frames, i.e., the delay parameter τ_l for each link l is a constant multiple of the frame size T. In this case stop-and-go queueing is based on one simple rule.

Packets arriving in a frame f are always delayed until the beginning of the frame following f. Along with the given admission policy it ensures the following

- All packets arriving in f will be serviced before the subsequent frame ends.
- Each packet flow maintains its (r, T) smoothness throughout the path to the destination.
- For a link l with capacity C_l, a buffer space of $2C_l.T$ will be enough to prevent buffer overflow.

In practical cases where the delay parameter τ_l may not be a multiple of T, an additional delay θ_l is introduced into each link to make the total delay the next highest multiple of T, thereby synchronizing the departing and arriving frames.

Stop-and-go queueing does not correspond to any specific service discipline for serving packets of different flows within a frame. However, using a service discipline like FIFO would simplify the realization of the framework. Stop-and-go queueing used in conjunction with FIFO service discipline is called delayed FIFO queueing.

We now consider queueing delays experienced by the packets. If τ is the sum of the propagation and processing delays of the flow path, end-to-end delay of packets can be represented as $\tau+Q+d$. Here, $Q+d$ corresponds to the total queueing delay where Q represents the constant factor of the delay and d represents a variable factor. For a path with H links traversed by a flow, holding a packet till the beginning of the frame that comes after current arrival frame at each node leads to a constant delay factor of $H.T$ with a possible variation d bounded as $-T < d < T$. Additionally, delays of $\theta_l (< T)$ introduced in each node for synchronization leads to another constant delay factor, bounded by $H.T$. The total constant end-to-end queueing delay factor Q is bounded as $H.T \leq Q \leq 2H.T$.

There is a trade-off involved between queueing delays and flexibility of bandwidth allocation. If all packets have a fixed length Γ; the incremental step of bandwidth allocation Δr, is

$$\Delta r = \frac{\Gamma}{T} bits/sec.$$

For a given connection which traverses H hops, the queueing delay can be expressed as

$$Q = \alpha H.T$$

where α is some constant between 1 and 2 that depends on the flow path. It follows that

$$\Delta r.Q = \alpha H.\Gamma, 1 \leq \alpha < 2.$$

This equation clearly states that for a fixed source destination route and a fixed packet size, Δr and Q cannot simultaneously be decreased and a reduction in one would lead to a proportional increase in the other. While it is desirable to have less queueing delay for some connections and hence a small value of frame size T, larger value of T would allow smaller steps of bandwidth allocation and hence greater flexibility in the allocation. To overcome this trade off a strategy based on multiple frame types of different sizes has been designed.

Multiple frame sizes

A generalization of the framing strategy allows the use of more than one frame size. Consider G different frame sizes $T_1, T_2, \ldots, T_G$, where T_g is a multiple of T_{g+1}, $T_g = K_g.T_{g+1}$. Connections are set up as type g connections for some $g = 1, 2, \ldots, G$ associated with a frame size T_g. In this case, the admission policy would require that each type g connection, k, with an allocated transmission rate r_k to be (r_k, T_g)-smooth. With the current framework, stop-and-go queueing works on the following rules:

1. A type g packet arriving in type g frame f, is delayed until the beginning of the type g frame $f + 1$.

2. Any type g packet has non preemptive priority over frames of type g' where $g' < g$.

Assume that the aggregate transmission rate allocated over each link l to all type g connections, denoted by C_l^g satisfies the following

$$\sum_{g=g_0}^{G} C_l^g \leq C_l - \frac{\Gamma_{max}}{T_{g_0}}, g_0 = 2, \ldots, G$$

where Γ_{max} is the maximum length of packets in the network. It then follows that:

1. Any type g packet that has arrived at some link l during a type g frame f, will receive service before the type g frame following f expires.

2. The packet stream of each connection which on entry is (r_k, T_g) smooth, will remain so throughout the path to the destination.

3. A buffer space of $\sum_{g=1}^{G} 2C_l^g.T_p$ per link l is sufficient to eliminate any buffer overflow. This value is always less than or equal to $2C_l.T_1$.

The end-to-end delay bound for a type g connection is the same as if the stop-and-go queueing was practiced on a single-frame basis with frame size equal to T_g. The end-to-delay for a packet can be expressed as $\tau + Q_g + d_g$ where Q_g is the constant queueing delay factor and d_g is the end-to-end delay jitter. For a flow path traversing H links, these delay factors are bounded as follows

$$\begin{aligned} H.T_g \leq \quad & Q_g \quad < 2H.T_g \\ -T_g < \quad & d_g \quad < T_g \end{aligned}$$

For a constant packet size Γ, these bounds lead to the conclusion that

$$\Delta r_g.Q_g = \alpha.H.\Gamma, 1 \leq \alpha < 2.$$

Though this is similar to what was obtained using a single frame type, the coupling between the queueing delay and the incremental step of bandwidth allocation now applies separately for connections corresponding to different frame types. It is now simultaneously possible to assign small queueing delays for connections with a small frame size, and allocate bandwidth in fine segments for other connections with larger frame size.

Along with provision for flows with performance guarantees and flexibility in resource allocation, best-effort traffic can be accommodated to improve network efficiency. The design also provides for easy implementation in high-speed networks where low processing delays are desirable.

2.4.2 Hierarchical Round Robin Based Scheduling

Kalmanek, Kanakia, and Keshav [13] proposed a queue service discipline based on round-robin scheduling to provide rate and jitter guarantees to connections. Users request an average service rate, defined over an averaging interval. The concept of jitter used in this service discipline is a short term average rate defined over an averaging interval. The interval used in the definition of jitter, *jitter averaging interval*, is

different from the one used in the definition of service rate, and it is typically smaller. Jitter bounds the number of packets that can transmitted in the jitter averaging interval.

The definition of jitter used in the service discipline helps in smoothing the output stream by bounding the short term average rate. In transporting video or voice traffic, variance in inter-packet gaps at the receiver is handled by using buffers to smooth out the traffic. In such cases, the main factor affecting cost would be the length of the burst that has to be stored before decoding data. This burst length is captured in the definition of jitter. While ensuring the average service rate to a stream means that the decoder gets input data at a minimum data rate, jitter bounds the maximum burst that needs to be buffered. The goals of the scheme are to provide rate guarantees, an upper bound on jitter and the lowest possible delay and packet loss to best-effort traffic.

The design is presented for a scheduler used on an ATM kind of a network where data is transmitted in fixed size units called cells. First, consider the working of an ordinary round-robin server for fixed-size cells. Data from each connection is stored in separate data queues. A list of connection identifiers currently in need of service are maintained in a service list. On cell arrival, the connection identifier value, CID, is added to the service list, ensuring that it is not already there. The server takes a CID from the service list and serves it for some fixed time quanta. The quantum can be different for different connections. If the data queue corresponding to the CID is non-empty, the CID is added to the tail of the service list before the server goes on to the next CID in the list.

A hierarchical round robin based scheduler is based on refinements of the basic round-robin server. Transmission on an output link occurs in frames of fixed size. A frame is measured in time slots, a time slot corresponds to the service time of one cell. The first refinement would be to make the round-robin server to start service through the list of CID's once per frame. Once the connections service quantum is served, the CID is returned to the end of the service list. It should be ensured that it is not serviced again in the current frame. One way to ensure this is to maintain two lists, as shown in Figure 2.5, *current list* which is the list of CID's currently under service, and *next list*, where the CID's are added. At the end of each frame the two lists are swapped. The second refinement is to maintain a hierarchy of service lists at the server, each having a different frame length. The topmost list in the hierarchy has the shortest frame length, and serves connections allocated the highest rate. Best-effort

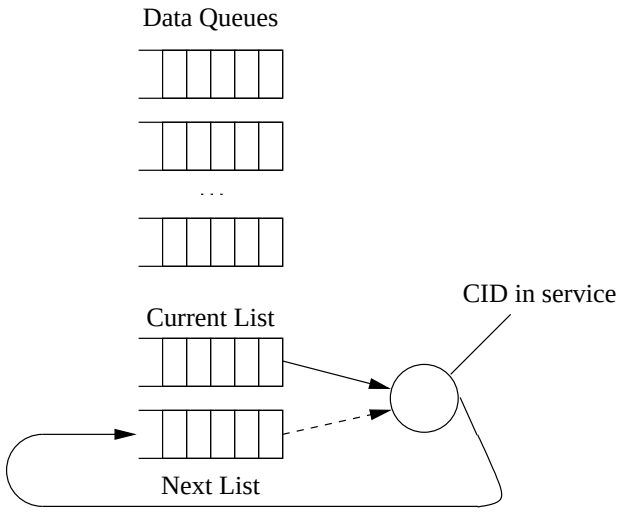


Figure 2.5: A Frame based Round Robin Server

traffic is at the bottom in the hierarchy. A list at level i has some number of time slots, n_i, associated with it. On the completion of service of the n_i slots associated with a list, another round is started through the service list. Some fraction of slots associated with each list are allocated to lists lower in the hierarchy. For example, consider a hierarchy of two lists as shown in Figure 2.6, with $n_1 = 5$ and $n_2 = 6$. Of

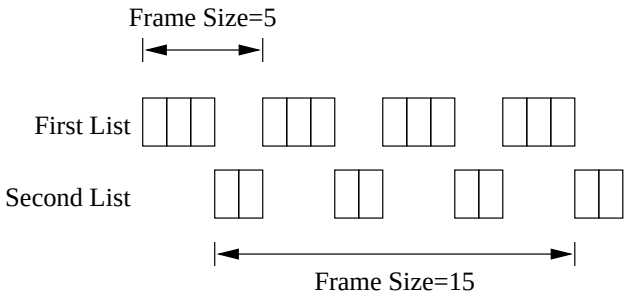


Figure 2.6: A hierarchy of two lists with different frame sizes

the 5 slots associated with the first list let 2 slots be allocated to lists lower in the hierarchy. This means that 60% bandwidth can be allocated to connections of the first list and 40% bandwidth can allocated to connections of the second list. The frame size associated with a list is the time between the start of two consecutive rounds of service of the list. Frame size for the first list is 5 time slots, and for the second list it is 15 time slots as it takes these many time slots, after the start of service of one round and before starting the next round, to service the 6 slots corresponding to this list.

Service lists are interleaved as follows: Let FT_i be the frame time interval at which the list i's service starts. After service of $b_i < n_i$ cells at level i, the $n_i - b_i$ cells are served at level $i+1$. Each connection j can have a service quanta $a_j \geq 1$, number of cells served from this connection when picked for service. Before a frame expires, if no CID is left to be served then the best-effort server at the bottom of the hierarchy fills the rest of the frame with best-effort traffic cells.

The algorithm guarantees a bandwidth a_j/FT_i cells/sec to a connection j at level i. The jitter bound is computed as follows. In the worst case a_j cells are served at the end of one frame followed by another a_j cells at the beginning of the next frame. For the interval during which the cells are sent, the jitter is bounded by $2 * a_j$ cells during one frame time. For computing delay bound at each intermediate node it can be noted that in the worst case a cell which arrives just after connection j is served may not receive service till the end of the next frame. Hence, delay is bounded by $2 * FT_i$. In addition to providing rate guarantee, jitter bound, and delay bound, having different frame sizes corresponding to different levels of the hierarchy allows different steps of bandwidth allocation. Larger frame sizes allow smaller steps of bandwidth allocation.

2.5 Rate-Controlled Static Priority Queueing

A service discipline by name rate-controlled static priority queueing was designed by Zhang and Ferrari [22] to provide guarantees for throughput, delay, delay jitter in a connection-oriented packet switching network. The key idea in the RCSP server is to separate rate control and delay control function in its design.

An RCSP server is composed of two components, a Rate controller and a Static Priority scheduler which are responsible for allocating bandwidth and delays to different connections respectively. This method of subdivision of the server facilitates the decoupling of delay and bandwidth allocation. The combination not only simplifies the admission control but is also simple to implement.

The RCSP server expects each real time connection that needs a particular type of service to give a specification for the traffic produced by the source. The network needs to allocate resources on a per connection basis. One such specification is the (X_{min}, X_{ave}, I, P) traffic characterization. X_{min} is the minimum packet inter-arrival time, X_{ave} is the average packet inter-arrival time over an averaging interval I. P is maximum packet size for a given connection.

Conceptually an RCSP server has two components. A rate controller and a static priority scheduler as shown in Figure 2.7. The rate controller shapes the input traffic to the desired traffic pattern. This is achieved by assigning eligibility times to each real time packet received. The scheduler's main functionality is to order the transmission of eligible packets handed over by the rate controller. The rate controller acts like a set of regulators, one per connection. The regulators controls the interactions between switches and eliminates jitter. There are two kinds of regulators, rate jitter controlling regulator and the delay jitter controlling regulator. While the rate jitter controlling regulator partially reconstructs the received traffic pattern at each switch, the delay jitter controlling regulator does a complete reconstruction of the pattern at each switch.

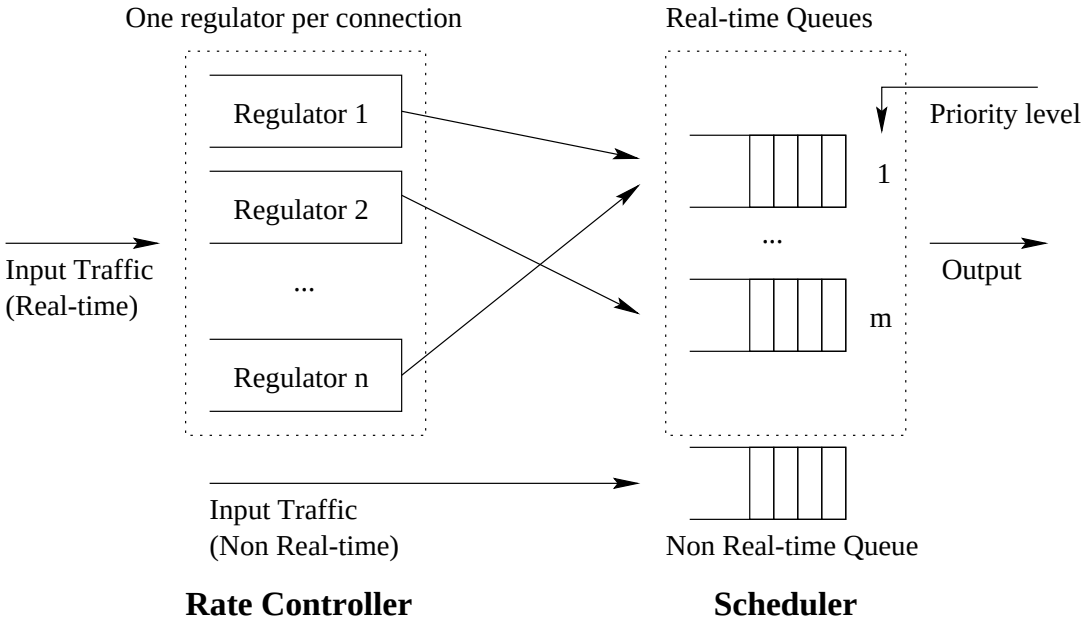


Figure 2.7: Rate-Controlled Static Priority Queueing

The rate jitter controlling regulator assigns an eligibility time to each packet based on the packets arriving earlier at the switch. Let ET_j^k be the eligibility time of the kth packet at switch j. Eligibility time of a packet is calculated as follows.

$$\begin{aligned} ET_j^1 &= AT_j^1 \\ ET_j^k &= \max(ET_j^{k-1} + X_{min} + \tau_j^k, AT_j^k), k > 1 \end{aligned}$$

AT_j^k is the arrival time of the kth packet at switch j. τ_j is the minimum number such that the average packet inter-arrival time in any period I does not exceed X_{ave}. This

method of assigning the eligibility time implies that a packet becomes eligible only after it arrives and that a connection packet eligibility times satisfies the given traffic characterization, (X_{min}, X_{ave}, I).

The delay jitter controlling regulator assigns eligibility times based on the eligibility time of the same packet at the upstream switch. Eligibility time of a packet is calculated as follows.

$$\begin{aligned} ET_0^k &= AT_0^k \\ ET_j^k &= ET_{j-1}^k + d_{j-1} + \pi_{j-1,j} \end{aligned}$$

d_j is the delay bound or maximum waiting time at the scheduler of switch $j-1$. $\pi_{j-1,j}$ is the propagation delay between switch $j-1$ and switch j. Thus a packet is never eligible before its arrival. Another implication is that $ET_j^{k+1} - ET_j^k = AT_0^{k+1} - AT_0^k$, i.e., the traffic pattern of the connection is fully reconstructed at the output of the regulator of every switch. Provided the input traffic obeys the traffic specification at the entrance of the network, it will obey the traffic specification at the output of the regulator of each switch.

Both regulators enforce the traffic specification, for each connection so that traffic going into the scheduler always satisfies the traffic specification.

Scheduler algorithm services all packets in the order of priority. It is a non-preemptive static priority scheduler. It has a number of prioritized real-time queues and a non real-time packet queue. Packets at priority level 1 have the highest priority. A connection is assigned a priority at connection establishment time. The scheduler services packets in FCFS order from the highest non-empty priority queue. Non real-time packets are serviced when there are no real-time packets waiting at the scheduler to be serviced. No order is specified for the service of non real-time packets. By limiting the number of connections at each priority level using an admissions control procedure the scheduler can provide delay bounds to the packets depending on the priority level to which they belong.

The residence time of a packet in an RCSP switch has two components: the holding time in the regulator and the waiting time at the scheduler. A delay bound on waiting time is associated with each priority level. Let $d_1, d_2, \ldots, d_n (d_1 < d_2 < \cdots < d_n)$ be the delay bounds associated with each of the n priority levels, respectively. Assume the jth connection among i_k connections traversing switch at priority level k has the traffic specification $(X_{min_{k,j}}, X_{ave_{k,j}}, I_{k,j}, P_{k,j})$. Also assume that the link speed is l, and the size of the largest packet that can be transmitted over the link is P_{max}. Then

we have the following results. Firstly, if

$$\sum_{k=1}^{m}\sum_{j=1}^{i_k}\lceil\frac{d_m}{X_{min_{k,j}}}\rceil P_{k,j} + P_{max} \leq d_m l, \tag{4}$$

the waiting time of an eligible packet at level m is bounded by d_m.

Let a connection with traffic specification (X_{min}, X_{ave}, I, P) passing through two RCSP switches connected in cascade be assigned a priority level with delay bound d at the first switch. If connection admission control is satisfied at both switches, then the waiting time in the first switch plus the holding time in the second switch is less than d if a rate jitter controlling regulator is used, and is equal to d if a delay jitter controlling regulator is used at the second switch.

Consider a connection passing through k switches connected in cascade where end to end propagation delay is Π. Assume that the connection is assigned to the priority levels with delay bounds $d_{i_1}, \ldots, d_{i_k}$ at each switch, respectively. If admission control conditions are satisfied at each switch, then the end to end delay for any connection is bounded by $\sum_{j=1}^{k} d_{i_j} + \Pi$ for both types of regulators. If a delay jitter regulator is used then delay jitter for any packet is bounded by d_{i_k}. The amount of buffer space equal to $(\lceil\frac{d_{i_{j-1}}}{X_{min}}\rceil + \lceil\frac{d_{i_j}}{X_{min}}\rceil)P$ is needed by connection i at switch j to prevent packet loss $(j = 1, \ldots, k; d_{i_0} = 0)$.

RCSP provides end-to-end delay guarantees to flows, in addition it provides rate jitter and delay jitter control through the use of rate jitter and delay jitter controlling regulators. While it gives complete flexibility in rate allocation, it also provides for varying levels of delay bounds based on the priority assigned to a connection. There is no coupling between rate and delay allocations. Hence, one connection can have a low bandwidth but a strict delay bound along with another connection having high bandwidth but a more relaxed delay bound. Many scheduling disciplines use sorted priority queues in their design. As a sorted priority queue has an $O(logN)$ insertion operation, these scheduling disciplines are unsuitable for high speed networks. RCSP does not use sorted priority queues, and is suitable for implementation in high speed networks.

2.6 Link-sharing

Sally Floyd [8] proposed mechanisms for link-sharing and resource management in packet networks, and presented algorithms for hierarchical link-sharing. By link-sharing is meant the a method to allow the bandwidth on a link to be shared among different traffic entities, where a traffic entity can be an individual flow or an aggregation of flows. Hierarchical link-sharing allows multiple agencies, protocol families, or traffic types to share bandwidth on a link in a controlled fashion. Instead of using separate mechanisms for link-sharing and real time services, the approach suggested is to view the link-sharing and real-time service requirements as simultaneous and complementary. Both requirements can be implemented with a unified set of mechanisms.

Because of the decentralized nature of the Internet, composed of multiple administrative domains with a wide range of resource limitations, control of Internet resources involves local decisions on usage as well as considerations of end to end requirements. Link-sharing goal is to allow control on distribution of bandwidth on local links in response to local needs and allow isolation of real-time and best-effort traffic. This isolation allows the use of packet scheduling algorithms that provide priority based scheduling designed to meet end to end requirements of real-time traffic.

One requirement for link-sharing is to allow a link to be shared by multiple organizations. Each organization is given a guaranteed bandwidth share on the link and unused bandwidth is available for other organizations to share. In an organization there might be a need to share bandwidth among multiple protocol families. The links bandwidth would also required to be shared between different traffic types, for example audio traffic, video traffic, FTP traffic, etc. These requirements lead to the need to support a hierarchical link-sharing structure as shown in Figure 2.8. Another goal of link-sharing can be to enforce explicit mechanisms at the gateway to prevent starvation of lower priority traffic.

The method suggested to achieve goals of link-sharing is to conceptually break the scheduler into a general scheduler and a link-sharing scheduler. In the absence of congestion the general scheduler could use any scheduler mechanism to schedule the packets. On congestion the link-sharing scheduler would come into play to rate limit the overlimit classes of traffic. Link-sharing mechanisms take minimum action to ensure that classes receive the allocated share of bandwidth over relevant time intervals.

One approach to link-sharing is based on the hierarchical class-based resource management proposed initially by Van Jacobson [3]. This approach is called class-based queueing(CBQ). An implementation of CBQ is discussed in a Section 3.1.

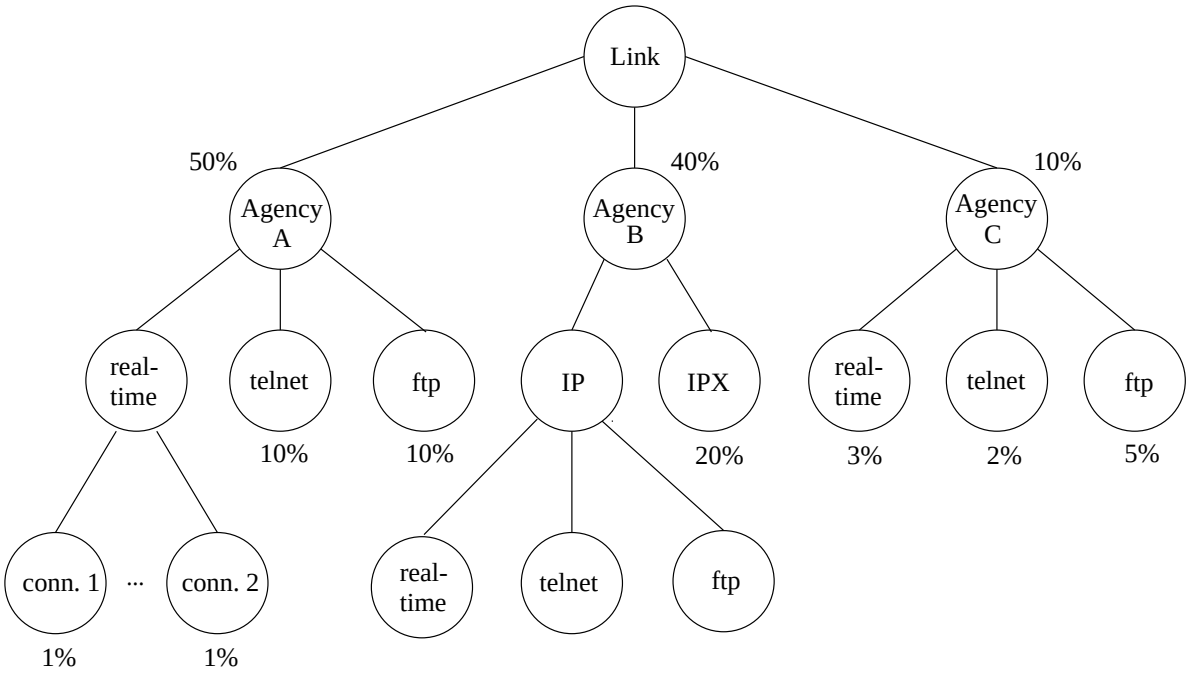


Figure 2.8: A hierarchical Link-sharing Structure

The goal of link-sharing is to provide a single set of mechanisms for link-sharing which should be implemented such that they carefully coordinate with any additional mechanisms for providing real-time services. A link-sharing structure specifies the desired policy in terms of division of bandwidth in times of congestion. The link-sharing structure and the respective bandwidth allocations could be either static or dynamic. The part of the structure specifying the division of bandwidth among organizations or protocol families would be static in nature and would be set up at network administration time. The structure for link-sharing among different traffic types or traffic of end connections would be of a dynamic nature. The general scheduler deployed should support some kind of a priority based scheduling scheme to cater to real-time traffic. A hierarchical link-sharing structure would could be like shown in Figure 2.8. Bandwidth is first divided among multiple agencies. An agency that has different protocol families being used like, IP and IPX may require to divide the bandwidth among different traffic protocols families. Bandwidth of a particular agency would be shared among different traffic types. Bandwidth reserved for a particular traffic type

can be shared among multiple connections. In the example, Agency A should receive 50% of the bandwidth over relevant time intervals. If agency A's real-time class does not have enough data to send then the excess bandwidth then the excess bandwidth would be allocated to other subclasses of Agency A.

The main goals of links sharing are as follows.

1. Each interior or leaf class should receive roughly its allocated bandwidth over appropriate time intervals.

2. If all leaf and interior classes with sufficient demand have received their allocated bandwidth share, the distribution of any excess bandwidth should not be arbitrary but follow some set of reasonable guidelines.

2.6.1 Link-sharing Guidelines

A formal set of guidelines for link-sharing and approximations to the formal set of link-sharing guidelines have been defined. The following definitions will help understand the guidelines.

regulated and unregulated classes A class is *regulated* if packets from that class are being scheduled by the link-sharing scheduler at the gateway; a class is said to be *unregulated* if traffic from the class is being scheduled by the general scheduler.

classifier, estimator The *classifier* classifies packets arriving at the gateway to appropriate class for the output link. The *estimator* estimates the bandwidth used by each class over appropriate time interval, to determine whether or not a class has been receiving its link-sharing bandwidth.

overlimit, underlimit, at-limit A class is said to be *overlimit* if it has recently used more than its allocated bandwidth, *underlimit* if it has used less than a specified fraction of its link-sharing bandwidth and *at-limit* otherwise.

satisfied, unsatisfied A leaf class is defined as *unsatisfied* with the link-sharing behaviour if it is underlimit and has a persistent backlog, and *satisfied* otherwise. A non-leaf class is defined as *unsatisfied* with the link-sharing behaviour if it is underlimit and has some descendant class with a persistent backlog.

levels All leaf classes in the link-sharing structure are defined to be at *level* 1, and each interior class has a level one greater than the highest level of any of its children.

The formal link-sharing guidelines state that a class can continue unregulated if one of the following conditions hold.

1. The class is not overlimit, or
2. The class has a not-overlimit ancestor at level i, and there are no unsatisfied classes in the link-sharing structure at levels lower than i.

Otherwise the class will be regulated by the link-sharing scheduler. These link-sharing guide lines are used to decide if a class is allowed to be scheduled by the general scheduler, unregulated, or whether the class should have its bandwidth regulated by by the link-sharing scheduler. The division of bandwidth among unregulated classes is determined by the general scheduler. These guidelines are used to determine when a class is using more than the bandwidth allocated to it, causing some other class to be in an unsatisfied state. The link-sharing guidelines do not specify about how often the scheduler should check whether a class needs to be regulated. This can be decided upon in implementations based on the requirements. With formal link-sharing guidelines the decision whether or not to regulate a class, depends not only on the limit status of the parent classes, but also on the 'satisfied' status of other classes in the link-sharing structure. Though it is possible that these formal link-sharing guidelines could be efficiently implemented, however, several approximations to the formal link-sharing guidelines that lend themselves more readily to efficient implementations have been suggested. The approximations suggested are Ancestors-Only link-sharing and Top-Level link-sharing.

Chapter 3

Simulator and Implementations

We have conducted a simulation study of selected scheduling disciplines. Our goal was to compare the performance of the scheduling disciplines in serving traffic mixes consisting of different kinds of traffic flows. The simulation framework that we have used is built on *ns*, the object oriented network simulator, developed at Lawrence Berkeley Network Laboratory. *ns* [5] began as a variant of REAL Network simulator in 1989 and its development is now an ongoing collaboration with the VINT project.

ns is an object oriented simulator written in C++ with an OTcl interpreter as its front-end. OTcl [20], developed at MIT, is an extension to Tcl/Tk for object oriented programming. The simulator supports a class hierarchy in C++ and a similar class hierarchy within the interpreter. In the user's perspective there is a one-to-one correspondence between a class in the interpreted hierarchy and one in the compiled hierarchy. Users create new simulator objects through the interpreter and these objects are instantiated within the interpreter and are closely mirrored by corresponding objects in the compiled hierarchy.

The simulator has many features desirable for our study. Among the features useful to us are the following:

- Classification of packets. Classifiers are provided to match a packet against some logical criteria and perform actions based on the match results. In particular, a hash classifier is used to classify a packet as a member of a particular flow. The packet header supports a flow ID field. Packets are assigned to flows based on the flow ID. This facilitates differential treatment to flows.

- An extensible queue management and packet scheduling framework. A base

class provides the basic functionality to implement a queueing discipline with virtual functions for implementing enque and deque operations at a node. This aids the addition of new queueing disciplines by implementing the enque operation where a decision is to be taken on what to do with a packet that is to be added to the queue, and the deque operation where the next packet to be transmitted is chosen.

- In addition to simple FIFO scheduling, some other scheduling disciplines come as a part of the simulator. The most interesting among these is the CBQ including a priority based round robin scheduler. There are a few more scheduling disciplines namely Fair Queueing, Stochastic Fair Queueing, and Deficit Round-robin scheduling.

- Trace and monitoring support. Number of ways are provided for collecting output or trace data on a simulation. Two distinct functional capabilities are supported. The first, called traces, record information on each individual packet as it arrives, departs or is dropped at a link or queue. The other type, called monitors, records counts of various interesting quantities such as packet and byte arrivals, departures, etc. Monitors can monitor counts associated with the all packets at a node or on a per-flow basis.

- An extensible traffic generation framework. A base class provides the basic functionality for traffic generation. A virtual function is provided to return the size of the next packet to be sent by a particular transport agent and the next time this function should be called. Some simple traffic generators come with the simulator which includes a constant bit rate traffic generator for generating traffic according to a deterministic rate. On/off Traffic generation according to exponential and Pareto distributions is also supported.

3.1 CBQ

In this section we discuss CBQ as implemented in *ns*. CBQ is a hierarchical, class based resource management framework which supports a link-sharing structure and is built on the guidelines as mentioned in Section 2.6. The implementation is based on the conceptual idea of a general scheduler to schedule packets from unregulated classes and a link-sharing scheduler to schedule packets from regulated classes.

3.1.1 Estimator

In order to decide when class is overlimit and may need to be regulated, an *estimator* is used. On transmission of a packet from a class the estimator is used to recompute the limit status of this class and all its ancestor classes. The estimator uses an exponential weighted moving average(EWMA). The estimator looks at the recent inter-packet departure times giving a decaying weight to the most distant packets, it computes the mean inter-packet departure time. Let d be the discrepancy between the actual inter-departure time and the allocated inter-departure time for the class taking into consideration the packet that has just been sent. The exponential weighted moving average, avg of the d variable is computed using the following equation.

$$avg \leftarrow (1 - w)avg + w \times d.$$

Here w determines the time constant of the estimator, how quickly avg catches up to reflect a sudden change in the sending rate of a flow. A lower value for w means it takes more time for the avg to reflect a change in the rate. Note that negative values of d and avg indicate the class is exceeding its link-sharing bandwidth and non-negative values indicate otherwise. Another consideration is the control on the maximum burst that can be sent from a class which has been idle for a long time. By setting a maximum value as a limit for the values that avg can take, a limit can be set on the maximum burst that can be sent by a class which has been idle for a long period. Accordingly, a limit on the minimum value of avg controls to what extent the limit status of a class should be influenced by previous bandwidth the class has received in excess of its allocated share. While avg indicates the limit status of a class, it is actually reflected through a time-to-send field in the class. The time-to-send field indicates the next time the gateway is allowed to send a packet from that class. If avg is positive, it means that the class is under limit and the time-to-send field is set to 0. If avg is negative then time-to-send field is set to a value x, such that if the gateway waits x seconds before sending a packet from the class then the class will no longer be over its limit. x is computed using the following equation.

$$x = -avg\frac{1 - w}{w} + f(s, b) \tag{1}$$

where s is the size of the packet just transmitted from the class which is allocated a link-sharing bandwidth b. $f(s, b)$ is the inter-departure time between successive packets if the class is sending precisely at the link-sharing bandwidth b.

3.1.2 General Scheduler

The general scheduler schedules packets from unregulated classes. A separate queue is maintained for each class associated with an output link. Packets from higher priority classes are scheduled first. Within classes of the same priority the general scheduler uses a variant of weighted round-robin scheduling with weights proportional to bandwidth allocated to the classes. Use of weighted round-robin ensures that higher priority classes receive their allocated bandwidth over fairly small intervals of time. Weighted round-robin scheduling ensures that the bandwidth is distributed to unregulated classes of the same priority in proportion to bandwidth allocated to the classes. The use of priority general scheduler with weighted round-robin within levels results in excess bandwidth being distributed to the higher priority classes, with the distribution being proportional to the relative link sharing allocations of those classes.

Before the general scheduler schedules a packet it checks the limit status of the class. It examines the time-to-send field and behaves as follows.

1. If time-to-send is 0 then the class is at-limit or underlimit and hence allowed to transmit the packet.

2. If time-to-send is positive and less than the current time, it may be overlimit but is allowed to transmit the packet.

3. If time-to-send is greater than current time then the class is overlimit, nevertheless, it may be allowed to transmit a packet if permitted by the link-sharing guidelines.

3.1.3 Link-sharing scheduler

The purpose of the link-sharing scheduler is to schedule packets from classes that are overlimit and found necessary to be regulated according to the link-sharing guidelines. The link-sharing scheduler sets the time-to-send field for a regulated class to $f(s, b) = s/b$ seconds ahead of current time. Where s is the size of the packet just transmitted and b is the bandwidth share, in bytes per second, that is allocated to the class. This results in the class being considered as overlimit by the general scheduler until the time in the time-to-send field. At that time it becomes unregulated and the general scheduler schedules a packet from the class. If the class is still calculated to be overlimit after the packet is sent, time-to-send field is again set to s/b seconds

ahead of current time. This method of working means that a class is never restricted by the link-sharing scheduler to less than its allocated bandwidth, regardless of the excess bandwidth used by the class in the past. Even for a regulated class the exact scheduling of packets is still determined by the general scheduler. This means that the priority-based general scheduler is likely to send a packet from a high priority regulated class soon after the time indicated in the time-to-send field. A low priority regulated class could be delayed somewhat longer before sending a packet from the class. However, if a class is delayed too often then the class may no longer remain overlimit and hence need not be regulated by the link-sharing scheduler.

3.2 Implementation of WFQ

Our implementation of the WFQ is based on the virtual time concept to track progress of the GPS leading to a practical implementation of PGPS as suggested in [16].

An *event* denotes each arrival and departure at the server. Let t_j be the time at which the jth event occurs. A busy period is when the server has packets in service or ready to be served. Let the starting time of the first busy period be $t_1 = 0$. For each $j = 2, 3, \ldots$, set of sessions busy in the interval (t_{j-1}, t_j) is fixed, and we may denote this set as B_j. Virtual time $V(t)$ is defined to be zero all times that the server is idle. For any busy session $V(t)$ is calculated as follows.

$$\begin{aligned} V(0) &= 0 \\ V(t_{j-1}, \tau) &= V(t_{j-1}) + \frac{\tau}{\sum_{i \in B_j} \phi_i} \\ & \quad \tau \leq t_j - t_{j-1}, j = 2, 3, \ldots \end{aligned}$$

where ϕ_i as defined in Section 2.2 is the weight associated with a session i. These equations imply that V increases at a rate $\frac{1}{\sum_{i \in B_j} \phi_i}$ and each backlogged session receives service at a rate $\frac{\phi_i}{\sum_{i \in B_j} \phi_i}$. V is increasing at the marginal rate at which backlogged sessions receive service. When the kth packet belonging to session i of length L_i^k arrives at time a_i^k, its virtual starting time S_i^k and virtual finishing time F_i^k of the packet are calculated using the equations

$$\begin{aligned} S_i^k &= \max\{F_i^{k-1}, V(a_i^k)\} \\ F_i^k &= S_i^k + \frac{L_i^k}{C\phi_i} \end{aligned}$$

where C is the link speed. The advantages of using virtual time approach is, firstly virtual time is to be updated only on occurrence of events as defined earlier. Secondly the virtual time finishing time of a packet can be calculated at packet arrival. This is possible because virtual time tracks the progress by changing its rate according to the number of active sessions and its weights. When more sessions are active it progresses slowly but when only few sessions are active it picks up speed. For each packet, on arrival, its virtual time finishing time is calculated and the packet is stamped with this virtual time. Packet are serviced in the order of increasing time stamps.

In the design of a buffering policy to be used with WFQ it would be desirable to allow complete utilization of the buffer space and also allow flows to be able to occupy buffer space proportional to their weights. This does not seem to be easy to achieve. On packet arrival, if there is going to be a buffer overflow then a packet is to picked for removal. It might be desirable to pick the packet belonging to a session that has been exceeding a quota, calculated based on its weight and maximum buffer size available. But this packet may have already influenced the virtual time calculations done after its arrival and these calculations need to be redone. Since this would be a costly affair, we use a different approach. We maintain a variable *threshold*. When the total number of packets is to exceed the *threshold* and the arriving packet belongs to a session that is exceeding its quota then we drop the packet. Doing this we get minimum utilization of *threshold* value for buffer space. On exceeding *threshold* only packets belonging to sessions exceeding their quota are dropped. Packets belonging to a session that is not exceeding its quota are dropped only when the complete buffer space is occupied.

3.3 Implementation of RCSP

It has been shown in Section 2.5 that the RCSP server has two components, the rate controller and the scheduler. The rate controller contains a set of regulators, one per flow. However, this conceptual decomposition into multiple regulators does not imply there must be multiple physical regulators.

On packet arrival its eligibility time is calculated, and it is the purpose of the regulator to hold the packets till they become eligible. Holding packets till they become eligible means the maintenance of a set of timers. The timers are managed using a calendar queue mechanism [2].

The implementation of the regulator is as shown in Figure 3.1. The calendar queue consists of a clock and a calendar. The calendar is a circular list of pointers indexed by time. Each entry of the list points to a linked list of packets and the priority level they belong to. The clocks ticks at a fixed time interval *tick*. Upon every tick, all packets from the list pointed to by the current time entry in the circular list and corresponding to the current time are handed to the scheduler. The priority level of the packet is used to decide the corresponding priority level in the scheduler at which the packet is to be queued.

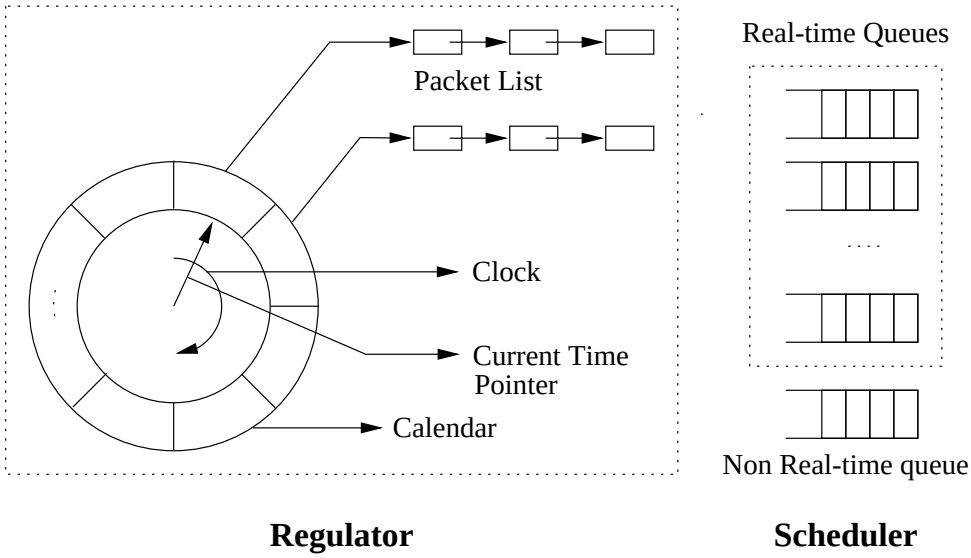


Figure 3.1: RCSP using Calendar Queue

Upon arrival of each packet, the eligibility time of the packet, ET is calculated; if $\lfloor \frac{ET}{tick} \rfloor$ is equal to the current clock time, the packet is appended at the corresponding real-time queue of the scheduler; otherwise, the packet is appended at the calendar queue entry corresponding to $\lfloor \frac{ET}{tick} \rfloor$. Since the calendar is maintained as a circular list a modulo function on the list size is used to find the entry in the list corresponding to a given time. On retrieving packets corresponding to a tick it is to be checked if the packets eligibility time actually corresponds to the current time or some time in future which also indexes to the same entry in the calendar. The data structures and the calendar queue mechanism facilitate a fast implementation for the RCSP server mechanism.

3.3.1 Support for ABR and UBR Traffic

The RCSP policy is readily usable for real-time traffic where the traffic is often specified either with parameters like average rate, peak rate and averaging intervals. There is often need to make a minimum bandwidth reservation for some classes of traffic. Neither is there a particular traffic arrival pattern nor is it needed to be specified, as such flows may not need any delay or delay jitter guarantees. Such traffic is classified as ABR traffic. The RCSP mechanisms are not directly suitable for accommodating such traffic. We have made provision for accommodating ABR traffic by making some changes to the originally proposed scheme. It would work as usual for real-time traffic whose traffic is characterized.

The treatment of real-time traffic is to regulate the traffic as per the traffic characterization and schedule the packets according to priority assigned to the flows. The regulator mechanisms not only ensure that the traffic obeys the given traffic characteristics, but also ensure a minimum rate at which packets from a flow become eligible and ready to be scheduled by the scheduler. For ABR traffic, we would like the latter capability of the regulator in providing a minimum rate. As there is no characterization for ABR traffic we need no regulatory mechanisms. The solution provided is to have one regulator for each ABR flow as before with the peak rate and average rate set to the minimum bandwidth guaranteed to the flow. However, All ABR packets are also queued in an alternative ABR queue and are always eligible to be scheduled. The result is that packets become eligible at a minimum rate and receive priority treatment like other real-time flows. When the server is idle it picks packets from the ABR queue for scheduling. Depending on the arrival rate of the ABR traffic, it can fill the gap between the link capacity and the bandwidth occupied by real-time flows. It will be a policy issue as to how the alternate queue of ABR packets are treated as compared to best effort traffic.

In the RCSP policy the UBR or best-effort traffic is maintained in FIFO queues. Once packets exceed the buffer capacity packets are dropped. In our implementation we maintain the FIFO ordering for best-effort traffic, however we have chosen to have a fair treatment of flows with respect to buffer space allocation. When a packet arrives and the buffers are full we choose the best-effort queue with the largest number of packets in the queue as the flow whose packet is to be dropped. The packet closest to the tail of the queue from the chosen flow is dropped from the queue.

3.3.2 Support for a Different Traffic Model

As discussed in Section 2.5 each real-time flow scheduled by the server has a specification $(X_{min}, X_{ave}, I, S_{max})$. X_{min} is the minimum inter-arrival time, X_{ave} is the average inter-arrival time over an averaging interval I, and S_{max} is the maximum packet size for a flow. The rate-controller part of the RCSP server is designed to support flows that have the the above mentioned traffic model.

We have suggested an implementation to support a different traffic model. The new model is based on an average rate and a maximum burst size. Packets are still regulated not to exceed a maximum rate. The essential difference is that instead of an explicit averaging interval over which the average rate is to be maintained, we instead ensure that the long time average rate is not exceeded and allow bursts according to the specified burst size. We use a token bucket filter for regulating the traffic according to this model.

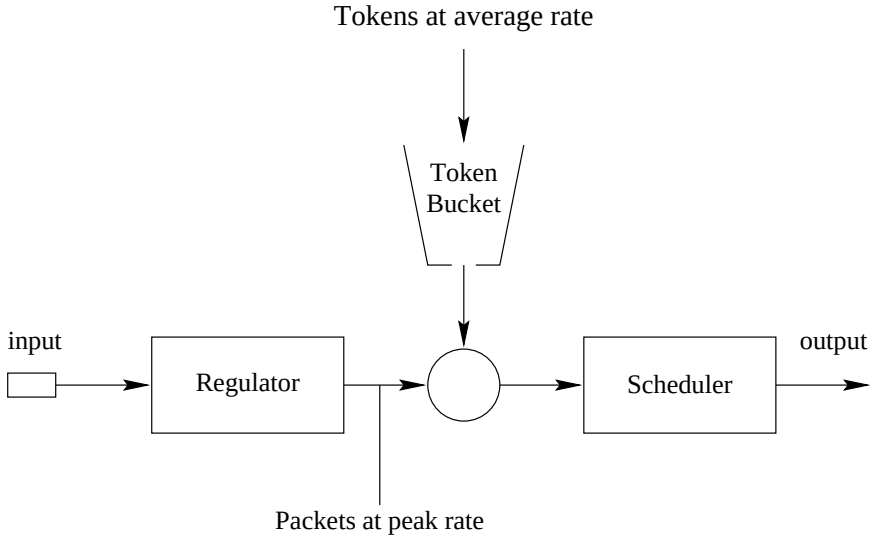


Figure 3.2: Token Bucket filter as regulator

As in the previous case we have a rate controller with a regulator for each flow and a static priority scheduler. To support the new model we introduce an additional token bucket filter as a part of the regulator. The packets have to get through the token bucket stage before being eligible for scheduling. The rate controller is designed as shown in the Figure 3.2. First the eligibility time is calculated for packets using the equation $ET^k = \max(ET^{k-1} + X_{min}, AT^k)$. ET^k is the eligibility time of the kth packet of a flow and AT^k is the arrival time of the kth packet. Packets are held in

a regulator till they become eligible. Eligible packets are enqueued into another wait queue. Here packets are regulated by a token bucket filter. Tokens are added to the bucket at the average rate and the maximum bucket size is given by the maximum burst size. If there are tokens in the bucket for the particular flow then the packets become eligible for scheduling. If the bucket is empty then the packets wait for their turn to come. This happens when more tokens are added to the bucket.

The time intervals at which tokens are added to a flows bucket ensures that the long term average rate of a flow is maintained at the output. The initial regulation ensures that the flows max rate as specified by X_{min} is not exceeded. An interesting consequence is that the bounds on the delay and delay jitter, and the buffer space required for a flow all remain as with the previous design. This is because bounds and the buffer space requirement are based on the peak rate and average rate does not affect their values.

3.4 Guaranteed Service Using CBQ and RCSP

Existing CBQ implementation using a Weighted round-robin scheduler as the general scheduler has many useful properties, but certain desirable features, not supported in its current form are, nevertheless, present in other scheduling disciplines. The existing implementation of the CBQ does not provide flexible support for guaranteed service.

The support for guaranteed service that we are discussing here is from a draft paper [7] which discusses the implementation of guaranteed service using CBQ.

3.4.1 CBQ and Guaranteed Service

The router that is to support a guaranteed service is given the traffic specification and service specification. The traffic specification consists of a token bucket with a bucket depth b and bucket rate r. The source should be policed at the edge of the network by such a token bucket. The service specification consists of a rate R for $R \geq r$. The flow's service at the router is characterized by the bandwidth R and a buffer size B, where B is derived from the given traffic specification. The router ensures that for a flow following the given specifications, the per packet queueing delay will be less than $b/R + C/R + D$. Here b/R is attributed to a bucket b of packets arriving instantaneously and effectively transmitted at rate R. The C/R and

D terms include the queueing delay due to transmission delay of the packet currently in service and delay due to other real-time packets scheduled to be sent before the packet in question.

The real-time classes are proposed to be among the priority 1 classes. Consider the worst-case delay of a packet for connection i at a particular gateway. Assume there are at most k priority-1 classes with the ith class having a bandwidth of ρ_i. Let the link capacity of the output link be ρ bytes/sec. Assume, in addition, that at most half the band-width is allocated to priority-1 classes. Though this is not a necessary condition it is used for simplifying the analysis. For a class-C_i packet that arrives at the gateway with no class-C_i backlog and no deficit is guaranteed to receive its allocated share of bandwidth in each round and a packet may wait at most

$$\frac{M/2 + kL}{\rho} \geq \frac{2kL}{\rho} secs$$

Here M is the total number of bytes belonging to priority-1 class which are serviced in each round of the round-robin scheme, and L is the maximum packet size in bytes.

To summarize, the worst case delay for all priority-1 classes is the same, however, it is not a function of the bandwidth allocated to the class. This worst-case delay that has been discussed is based on the assumption that not more than half the bandwidth is allocated to the classes at priority 1.

3.4.2 Incorporating an RCSP class

In this section we present an alternative approach based on RCSP scheduling that fits itself into the link-sharing paradigm that is supported by CBQ. To review the highlights of the RCSP scheme discussed in Section 2.5, it supports multiple priority levels corresponding to different local delay bounds, rate jitter control or delay jitter control is provided and the amount of buffer space needed to support a connection with a given traffic specification is bounded. It can support traffic specifications based on averaging interval as well as that based on bucket size as discussed in Section 3.3.2. Rate allocation is flexible and is not coupled with allocation of queueing delays. The big advantage in the CBQ scheme is in the provision for allocation of link-bandwidth to classes without any knowledge of the arrival pattern, and at the same an independent priority service is possible.

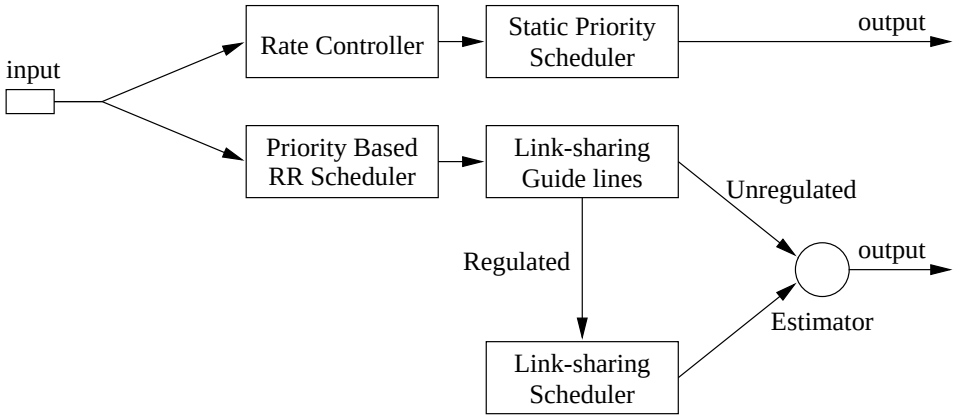


Figure 3.3: Extending CBQ to use RCSP

In an attempt to take advantage of the flexible mechanisms for allocation of resources to real-time flows by the RCSP scheduler and also provide some of the advantages of CBQ, we have designed and implemented an extension that is simple as it uses the existing implementations effectively. Our suggestion is that in the link sharing structure, all real-time classes are assigned to a special class called RCSP class which is in some sense an exempt class. All the non-real-time flows form a part of the link-sharing structure as in the normal case of CBQ. This design is shown in Figure 3.3. Packets belonging to the real-time classes that are assigned to the RCSP class are queued at the RCSP server on arrival, other packets enter FIFO-queues and are served by a round-robin scheduler according to the CBQ scheme. When the output link is free to service the next packet, a check is made to see if there are any eligible packets waiting at the RCSP server's priority scheduler. Packets waiting at this scheduler are given the first priority for service. If there are no packets from the RCSP server waiting to be serviced then a packet is chosen for service by the priority based weighted round-robin server. If the class to which the packet belongs is underlimit then it gets serviced. If it is overlimit then a check is made according to the link-sharing guidelines if it can be serviced. The class may need to regulated by the link-sharing scheduler, else it is serviced next. On servicing each packet, the estimator updates the limit status of the class. The only changes necessary are to include a check on packet arrival and departure. On arrival it is to be checked as to which scheduler the packet it is to be enqueued. When the link is free to send the next packet, a check is to be made to decide whether the RCSP server has a packet ready for service, else CBQ server takes the opportunity.

Though the RCSP class is getting the exempt status, it is to be noted that the regulators, forming the rate controller part of the RCSP server, ensures rate control of flows to maintain the allocated average rate over some time intervals. The time intervals depends on the traffic specification to be used. Since the RCSP class gets the topmost priority, the behaviour with respect to real-time flows is same as in the case of a stand alone RCSP server. If there are n real-time flows and each flow i is assigned an average rate r_i then the RCSP class is effectively allocated a bandwidth of $r_{RCSP} = \sum_{i=1}^{n} r_i$. The remaining link bandwidth r_{CBQ} can be allocated among the other classes. Due to a regulatory mechanism inbuilt in the RCSP server it fits into the link-sharing paradigm as a highest priority class which is never regulated by the link-sharing scheduler. However there is a guarantee that over sufficient intervals of time it remains underlimit or at-limit.

In the perspective of the RCSP server, the new scheme fits in as having a CBQ server serving the non-real-time packets instead of a simple FIFO queue that is shown in Figure 2.7. In the perspective of the CBQ server, the RCSP flows form an exempt class with an RCSP server taking responsibility of the class, and the rest of the flows are served by the combination of general-scheduler and link-sharing scheduler. This design seems to be fine in view of supporting a class structure with allocations among different traffic types whether aggregate classes or classes for individual connections, however, it is still to be investigated in the scenario of also trying to share bandwidth among different agencies or protocol families.

With simple modifications to existing implementations a mechanism for achieving flexibility of allocating resources to real-time flows and flexibility of link-sharing among non-real-time flows has been designed and implemented. In the next chapter we provide experimental results to support our claim.

3.5 Modeling Video Traffic

The simulator, *ns*, provides support for some traffic source models. Existing traffic generators support Constant bit rate sources and simple variable bit rate sources based on an on/off traffic model following exponential and Pareto distributions. We wanted to study scheduling disciplines with respect to how well they work for various kind of traffic categories. Variable bit rate traffic is one of the important categories, this is traffic of a bursty nature. Traffic from video sources is expected to be a

substantial portion of the traffic carried by emerging broadband integrated networks. Compressed video traffic transmitted on the network is generally variable bit rate coded video.

We have implemented as part of the simulator a VBR traffic generator in which we support statistical source models for VBR coded video. Video compression techniques are based on scene changes in the video. A video sequence consists of scenes, a scene in turn consists of multiple frames. At the start of a scene the complete frame is needed to be transmitted but subsequent frames may have lot of frame content common to the initial frame. It would be enough to transmit the difference between the frames rather than the complete frame. Hence, while high bit rate is needed for the first frame of a scene, lower bit rates may suffice for subsequent frames in the scene. In order to model a source we require to model scene lengths, the size of the first frame of a scene and the size of frames within scenes.

As discussed in [12] a single model based on a few physically meaningful parameters and applicable to all kinds of sequences is not possible. Three statistical distributions are used as candidates for describing scene length, scene change frame size and intermediate frame size. The three distributions that have been chosen are the Gamma distribution, the Weibull Distribution, and the Pareto distribution. We assume frames to be equally spaced, at a rate of 30 frames per second. Transmission from the source is such that packets carrying the frame contents are evenly spaced during the interframe interval, packets are not transmitted in bursts in some subinterval of the interframe interval. The particular distribution to be used for modeling each of parameters, the parameters pertaining to the individual distributions, and a scale parameter are all values that can be chosen for a particular VBR source. Here the scale parameter decides the size of a typical scene change frame, the purpose of such a parameter is to be able to choose different resolutions for the video signal that is transmitted. The maximum bit rate requirements depend on the resolution and the frame rate. Though the frame rate is fixed at 30 frames per second we can vary the resolution parameter.

Chapter 4

Experiments and Results

We used the simulation tool described in Chapter 3 to compare the performance of scheduling disciplines in serving traffic mixes of various kinds of traffic types with their respective performance requirements. The scheduling disciplines that were used for the simulation study are WFQ, CBQ and RCSP. The traffic types that constitute the traffic mixes are outlined as follows:

1. Constant bit rate(CBR) traffic. Packets arrive at constant time intervals. We provide minimum bandwidth guarantees to these flows and are interested in providing bounds on the packet delays and losses.

2. Variable bit rate(VBR) traffic. Packet arrival rate is not constant and typically bursty in nature. We provide minimum bandwidth guarantees to these flows and are interested in providing bounds on the packet delays and losses.

3. Available bit rate(ABR) traffic. No packet arrival pattern is specified. We provide minimum bandwidth guarantee to these flows, no delay guarantees need to be provided

4. Unspecified bit rate(UBR) traffic. No packet arrival pattern is specified. Neither do we provide any bandwidth guarantees, nor any delay guarantees to these flows.

For our experiments, we used the network topology shown in Figure 4.1. The traffic corresponding to each flow is generated at a separate node. All flows have a common sink and traverse a common bottleneck link. The scheduling disciplines to be

tested upon are employed at the router for scheduling the packets to be transmitted on the bottleneck link in order to provide quality of service to flows. By providing a guarantee on minimum data rates and by bounding the packet delays on each link along the path of a flow we can ensure a bound on the end-to-end delays and a bound on packet losses. It would be interesting to study the performance of the scheduling disciplines at the bottleneck link for two reasons. Firstly, the load on the link would be high. If the load is less, i.e., the available bandwidth is sufficiently larger than the demand, then it would not be very difficult for the scheduling disciplines to provide loss-less service with minimal packet delays. Secondly, interactions of packets from various flows at the router along with high load at the output link would make it difficult for the scheduling disciplines to provide required quality of service to the flows. This would help to bring out the difference in the performance of the scheduling disciplines.

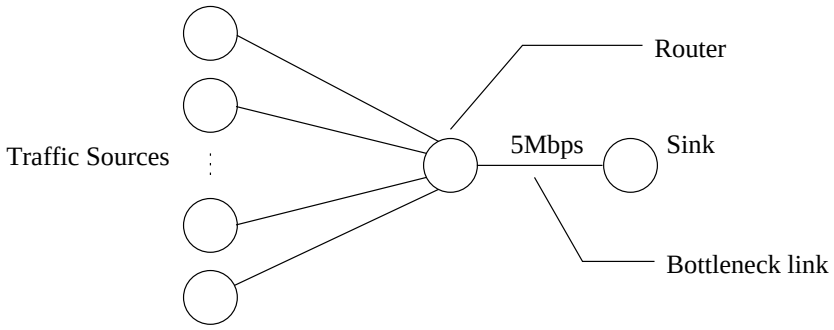


Figure 4.1: Topology of the Network used

For CBR flows we have chosen a data rate of 64 Kbps, the typical bandwidth required for audio traffic. The CBR traffic generator generates packets of size 210 bytes at a rate of 64 Kbps. For VBR flows we use traffic generated by video sources which are modeled according to the distributions discussed in Section 3.5. Gamma distributions are used for modeling scene length, initial frame size, and subsequent frame size. The parameters that can be chosen for a gamma distribution are order, and scale.

- For scene length in number of frames, we set $order = 30$ and $scale = 3.5$ giving scene lengths in the range of about 50 to 200 frames. At a frame rate of 25 frames/s this gives scene durations ranging from 2 to 8 seconds.
- For initial frame size in kilobytes, we set $order = 30$ and $scale = 2.9$ giving frame sizes in the range of about 50 KB to 150 KB.

- For subsequent frame size in kilobytes, we set $order = 15$ and $scale = 2.0$ giving frame sizes in the range of about 10 KB to 60 KB.

These frames sizes correspond to VBR scale value of 1. We used three different scales for VBR sources giving us VBR types VBR-1, VBR-2, and VBR-3 corresponding to scale values of 0.0367, 0.726 and 0.1445 respectively. These VBR sources generate packets of size 300 bytes at an average rate of approximately 194 Kbps, 415 Kbps, and 852 Kbps. For ABR flows, as the only criterion to be checked was whether they get their minimum guaranteed bandwidth, we only had to ensure an arrival rate higher than the allocated rate. There are two kinds of ABR flows which are allocated minimum rates of 150 Kbps, and 225 Kbps. Even in case of ABR traffic, we have used the same sources used for modeling video traffic. We have chosen scale values of 0.5 and 1.0 for these flows which ensures that their arrival rates are higher than their allocated rates.

Table 4.1 shows the parameter values used for the different flow types. The CBR flows are given the highest priority followed by VBR, and ABR flows. The third column shows the buffer space, in number of packets, allocated to flows. As CBR and VBR flows have higher priority, packets of these flows need not wait for long before being picked for service. ABR packets on the other hand would have to wait in the queues for a longer before receiving service. Hence, CBR and VBR flows are allocated smaller buffer space as compared to ABR flows. The 4th and 5th columns

Flow Type	Priority	Buffer Space (pkts)	Bandwidth (%)	X_{avg}(ms)	X_{min}(ms)	I
CBR	1	10	1.280	26.250	26.250	-
VBR-1	2	20	4.661	10.290	1.765	0.4
VBR-2	2	40	9.968	4.815	0.882	0.8
VBR-3	2	80	20.444	2.348	0.441	1.2
ABR-1	3	150	3.000	15.630	15.630	-
ABR-2	3	150	4.500	10.470	10.470	-

Table 4.1: Parameters used for CBR, VBR and ABR flows

show the average bandwidth allocated to flows. For CBQ it is expressed as percentage of the link bandwidth, while for RCSP it is expressed as average packet inter-arrival time(X_{avg}). The values shown reflect an average rate of 64 Kbps for CBR flows, and 20% more than the estimated average rates for VBR flows. While the CBR flows have no peaks, we found it necessary to allocate some additional bandwidth to VBR

flows to handle peaks sustained over some duration of time. In our experiments we found that allocating an additional bandwidth of 20% to VBR flows gives acceptable queueing delays. The column corresponding to X_{min} gives the minimum packet inter-arrival time used by RCSP which indicates the maximum peak rate allowed. For VBR flows, values are set for an estimated peaks of about 6 times the average rate. The last column gives the averaging interval over which the average data rate is to be maintained, while allowing higher rates within the interval. Averaging interval is not relevant for CBR and ABR flows as we set average rate equal to peak rate,

The CBR and VBR flows have a QOS loss parameter of 1 percent with a maximum queueing delay of 5 ms. Packets experiencing a delay of more than 5 ms are considered as good as lost.

A single seed is used to generate a set of seeds which in-turn are used to initialize the random number generators employed by VBR traffic generators. This seed is made a settable parameter so that a particular value can be chosen for a test. Using different seeds for different tests ensures that we take into account how the variations in packet arrival patterns for VBR traffic are handled by the scheduling disciplines. Making the random seed a settable parameter allows us to study how the same traffic arrival pattern is handled by different scheduling disciplines.

4.1 WFQ

Experiments were conducted to test the performance of WFQ for different traffic mixes. Weights were assigned to flows proportional to the estimated bandwidth required. We considered traffic mixes at different estimated link loads. We discuss the results of some of these experiments.

Table 4.2 indicates the results obtained when serving 12 flows of type VBR-2 for a duration of 40 s. The estimated bandwidth required to serve all the flows is 98% of the link bandwidth. The 3rd column shows the number of packets received in the duration for which the tests were run, where the estimated value is 6922 packets. The last column shows the number of packets delayed beyond 5 ms as a percentage of the packets forwarded. None of the flows QOS guarantee is violated. Table 4.3 indicates 2 sets of results obtained when serving mixtures of VBR-2 and VBR-3 kind of flows, and VBR-1 and VBR-3 kind of flows for a duration of 40s. The estimated bandwidth required to serve all the flows in both cases is 90% of the link bandwidth.

ID	Type	Packets rcvd.	Late(%)
0	VBR-2	6809	0.338
1	VBR-2	6915	0.405
2	VBR-2	6734	0.460
3	VBR-2	6751	0.548
4	VBR-2	6831	0.542
5	VBR-2	6793	0.545
6	VBR-2	6818	0.572
7	VBR-2	6843	0.482
8	VBR-2	6823	0.454
9	VBR-2	6850	0.453
10	VBR-2	6819	0.543
11	VBR-2	6831	0.512

Table 4.2: WFQ: Performance for VBR-2 flows

The estimated number of packets to be received for VBR-1 and VBR-3 flows are 3237 and 14197 respectively. When serving a mixture of VBR-2 and VBR-3 flows, QOS guarantee of 2 VBR-2 flows is violated. When serving a mixture of VBR-1 and VBR-3 flows, QOS guarantee of 3 VBR-1 flows is violated.

ID	Type	Packets rcvd.	Late(%)
0	VBR-2	6691	0
1	VBR-2	6935	0
2	VBR-2	6751	0.193
3	VBR-2	6809	1.043
4	VBR-2	6773	6.304
5	VBR-3	13989	0
6	VBR-3	13928	0
7	VBR-3	13929	0

ID	Type	Packets rcvd.	Late(%)
0	VBR-1	3204	0.218
1	VBR-1	3186	0.691
2	VBR-1	3204	0.936
3	VBR-1	3249	2.678
4	VBR-1	3205	5.367
5	VBR-1	3250	11.354
6	VBR-3	14309	0
7	VBR-3	14153	0
8	VBR-3	14156	0
9	VBR-3	14147	0

Table 4.3: WFQ: Performance for a mixture of 2 kinds of VBR flows

From these results and from other experiments we conducted, we found that when serving homogeneous traffic, even under high link loads, WFQ gives good performance. However, when serving a mixture of 2 kinds of flows, the QOS guarantee of a few flows is violated even for a link load of about 90%. The violations occur for flows that are reserved lesser bandwidth.

The main reason for this behaviour of WFQ is that flows which are given higher

weights are scheduled in a manner that they suffer lesser delays. It is not easy to reserve resources for flows requiring high bandwidth but high delay along with other flows requiring low bandwidth but lower delays. This also implies that it is not flexible enough to accommodate ABR traffic along with the CBR and VBR traffic, where ABR traffic is to be guaranteed bandwidth without regard to delays. For example, if it is required to reserve 15% bandwidth to an ABR flow and 5% bandwidth to a CBR flow, then CBR packets will experience longer delays than ABR packets. To summarize, WFQ is not flexible enough to accommodate traffic mixes which include CBR, VBR and ABR flows.

4.2 Comparison between CBQ and RCSP

Next, we discuss the experiments conducted to compare the performance of RCSP and CBQ scheduling disciplines under various link loads. For these experiments we have 2 ABR-1 flows and 2 ABR-2 flows reserved a total of 15% of the link bandwidth. We know the estimated data rates for all the CBR and VBR flow types. With these estimated data rates we generate all permutations of flow mixes which would require bandwidths in the range of 75% to 90% for CBR and VBR flows, assuming a minimum of 1 CBR flow, 1 VBR-1 flow, 1 VBR-2 flow and 2 VBR-3 flows. For example, with estimated total bandwidth requirement in the range of 93% to 94%, there are 26 permutations of flow mixes.

Table 4.4 compares the performance of the CBQ and RCSP for one of these flow mixes, falling in the 93-94% range, with 6 CBR flows, 3 VBR-1 flows, 1 VBR-2 flow, and 2 VBR-3 flows in addition to 4 ABR flows. The simulation was run for a duration of 40 s. In this duration, the estimates of the number of packets to be received are 1524, 3237, 6922, and 14197 packets for CBR, VBR-1, VBR-2, and VBR-3 flows. The table compares the percentage of packets delayed beyond 5 ms, and the minimum and maximum delays experienced by packets. Only data pertaining to CBR and VBR flows is shown. It can be observed that when using CBQ one flow of type VBR-1 has more than one percent of its packets delayed beyond 5ms. This is an instance where QOS guarantee of one of the flows is violated. An interesting observation about CBR flows is that packet delays under CBQ is less compared to packet delays under RCSP. However, delay jitter, the difference between maximum and minimum delay, is less under RCSP compared to CBQ.

ID	Type	Packets received	CBQ			RCSP		
			Late (%)	Delay(ms)		Late (%)	Delay(ms)	
				Min	Max		Min	Max
0	CBR	1524	0	0	0.054	0	0.568	0.597
1	CBR	1524	0	0	0.082	0	0.583	0.613
2	CBR	1524	0	0	0.077	0	0.599	0.629
3	CBR	1524	0	0	0.071	0	0.615	0.646
4	CBR	1524	0	0	0.069	0	0.632	0.662
5	CBR	1524	0	0	0.061	0	0.648	0.679
6	VBR-1	3182	0.094	0	6.117	0	0	3.209
7	VBR-1	3217	0	0	2.458	0	0	2.996
8	VBR-1	3201	1.562	0	9.590	0	0	3.218
9	VBR-2	6854	0.569	0	7.654	0	0	3.390
10	VBR-3	14132	0.021	0	5.511	0.446	0	8.251
11	VBR-3	13947	0	0	3.077	0.151	0	6.454
12	VBR-3	14151	0.106	0	6.450	0	0	3.601

Table 4.4: Results of an experiment comparing CBQ and RCSP

Next, we consider the number of times the QOS guarantee of a particular type of flow is violated at various link loads. Table 4.5 summarizes the number of instances where the QOS guarantee of flows are violated under CBQ and RCSP scheduling. Each row of the table corresponds to a different range of estimated link load. The entries in the table give counts of the number of instances where the QOS guarantee of a flow type was violated under CBQ and RCSP. For example, the row corresponding to 93-94% link load summarizes the results for 26 flow mixes involving 256 CBR flows, 96 VBR-1 flows, 47 VBR-2 flows, 56 VBR-3 flows. The violations of QOS guarantees occurs only for one instance corresponding to a VBR-1 flow in case of CBQ and one instance corresponding to a VBR-3 flow in case of RCSP. Since CBR flows have the highest priority, their QOS guarantees are never violated. In the range of 90-97% link load, QOS guarantees of VBR flows are violated on 10 instances when CBQ is used and on 2 instances when RCSP is used. Though RCSP does perform better, in both the cases, only in a very few of the instances the QOS guarantees violated. Under higher link loads, in the range 97-100%, RCSP clearly performs better than CBQ. QOS guarantees are violated more in CBQ regime. Under overload conditions when the estimated bandwidth is more than the output link can support, RCSP clearly gives priority treatment to VBR flows as compared to ABR flows, while CBQ tries to follow the link sharing guidelines even as it gives higher priority treatment to VBR flows. Hence, RCSP still does well with respect to VBR flows; ABR flows suffer

Link Load	CBQ					RCSP				
	V-1	V-2	V-3	A-1	A-2	V-1	V-2	V-3	A-1	A-2
90-91	1	0	0	0	0	0	0	0	0	0
91-92	1	0	0	0	0	0	0	0	0	0
92-93	1	0	0	0	0	0	0	0	0	0
93-94	1	0	0	0	0	0	0	1	0	0
94-95	0	1	1	0	0	0	0	0	0	0
95-96	2	0	0	0	0	0	0	0	0	0
96-97	1	1	0	0	0	0	0	1	0	0
97-98	4	0	4	0	0	0	0	0	0	0
98-99	2	1	7	0	0	0	0	1	0	0
99-100	7	5	10	0	2	0	0	2	2	2
100-101	4	4	17	30	31	0	0	1	28	26
101-102	2	6	18	64	63	0	0	0	64	64
102-103	8	11	28	88	88	0	0	1	88	88
103-104	7	11	32	76	76	1	0	2	76	76
104-105	13	12	44	90	90	0	0	0	90	90

Table 4.5: QOS violations when using RCSP and CBQ

losses. In case of CBQ the packet losses are distributed among all flows and hence QOS guarantees of VBR flows are also violated quite often.

4.2.1 In Absence of ABR flows

In the traffic mixes considered thus far, the presence of ABR flows gives the scheduling disciplines some flexibility in accommodating real-time flows. For ABR flows, packet delays are not of concern, and the bandwidth reserved for ABR flows is claimed by real-time flows when their demand peaks. This may be followed by a duration where the ABR flows reclaim unused bandwidth. Though the ABR flows may not get their reserved bandwidth over short time intervals the reservations are maintained over longer time intervals. Hence, even under high link loads, QOS guarantee could be give to CBR and VBR flows without violating the minimum bandwidth guarantee to ABR flows. We now look at the experiments conducted to compare the performance when trying to serve flow mixes containing only VBR and CBR flows.

Table 4.6 compares the number of instances of QOS violations when using CBQ and RCSP. We consider permutations of flow mixes with estimates of bandwidth required in the range of 85-100% of link bandwidth, where each flow mix has a minimum of 1 CBR flow, 1 VBR-1 flow, 2 VBR-2 flows, and 2 VBR-2 flows. It can

seen from the table that while with CBQ we have less number of QOS violations when when link load is under 90%, with RCSP we have less number of QOS violations even when the link load goes up to 96%. To summarize, when serving only real-time flows, RCSP performs better than CBQ and makes provision for better resource utilization.

Link Load	CBQ			RCSP		
	V-1	V-2	V-3	V-1	V-2	V-3
85-86	0	0	1	0	0	2
86-87	0	0	0	0	1	0
87-88	0	0	1	0	0	0
88-89	0	0	1	0	0	0
89-90	2	1	0	0	0	2
90-91	2	4	4	0	0	0
91-92	1	4	11	0	0	4
92-93	2	3	11	0	0	2
93-94	6	4	28	1	0	3
94-95	10	17	32	0	1	1
95-96	27	13	39	0	0	1
96-97	48	32	62	1	2	16
97-98	54	66	78	10	12	19
98-99	141	107	113	34	37	50
99-100	173	145	111	99	76	80

Table 4.6: QOS violations when using RCSP and CBQ, with no ABR flows

4.3 RCSP with a Token Bucket Regulator

The experiments described in the previous section were repeated using RCSP with a Token Bucket Regulator(RCSP-TB). Table 4.7 shows the performance of RCSP-TB when serving one of the flow mixes with a required estimated bandwidth in the range of 93-94% of the link bandwidth. This table shows the results for the same set of flows for which the the results when using CBQ and RCSP was presented in Table 4.4. The bucket size used for VBR flows is 25 packets. The table presents the percentage of packets delayed beyond 5 ms, and the minimum and maximum delays experienced by packets. It can be observed for CBR flows that the maximum delay is almost the same as in the case of RCSP, but the delay jitter is larger. For VBR flows, while the delays are comparable to those of RCSP, the number of packets delayed beyond 5ms is lesser.

ID	Type	Packets received	RCSP-TB		
			Late (%)	Delay(ms)	
				Min	Max
0	CBR	1524	0	0.358	0.593
1	CBR	1524	0	0.370	0.609
2	CBR	1524	0	0.383	0.625
3	CBR	1524	0	0.396	0.641
4	CBR	1524	0	0.409	0.658
5	CBR	1524	0	0.422	0.675
6	VBR-1	3182	0	0	2.596
7	VBR-1	3217	0	0	2.620
8	VBR-1	3201	0	0	2.702
9	VBR-2	6854	0	0	2.758
10	VBR-3	14132	0	0	3.165
11	VBR-3	13947	0	0	4.305
12	VBR-3	14151	0.014	0	5.120

Table 4.7: Results of an experiment using RCSP-TB

Table 4.8 shows the number of instances when the QOS guarantee of different flow types was violated under different ranges of link loads for flow mixes containing CBR, VBR, and ABR flows. Table 4.9 shows the number of instances of violation of the QOS guarantee with flow mixes containing only CBR and VBR flows. Comparing these results with the results shown in Table 4.5 and Table 4.6 we can see that as compared to CBQ or RCSP, the number of instances of QOS guarantee violations is lesser, showing that RCSP-TB performs better.

The reason for the better performance when using RCSP-TB is due to increased flexibility. RCSP uses an averaging interval; the amount of data that can be sent over any such interval is limited based on the average data rate assigned to a flow. When using a Token Bucket regulator there is no strict averaging interval over which the average data rate is to be maintained, whereas, the number of tokens in the bucket determines how peak data rates can be handled. Tokens get accumulated at times when the packet arrival rate is below the average rate assigned to it. The tokens can then be used during peak transmission. This gives more flexibility in handling peaks. The limit on the maximum bucket size, however, prevents flows from accumulating too many tokens to send bursts for very long intervals.

Link Load	RCSP-TB				
	V-1	V-2	V-3	A-1	A-2
90-91	0	0	0	0	0
91-92	0	0	0	0	0
92-93	0	0	0	0	0
93-94	0	0	0	0	0
94-95	0	0	0	0	0
95-96	0	0	0	0	0
96-97	0	0	0	0	0
97-98	0	0	0	0	0
98-99	0	0	0	0	0
99-100	0	0	0	0	2
100-101	0	0	0	26	28
101-102	0	0	0	64	64
102-103	0	0	0	88	88
103-104	0	0	0	76	76
104-105	0	0	0	90	90

Table 4.8: QOS violations when using RCSP-TB

Link Load	RCSP-TB		
	V-1	V-2	V-3
85-86	0	0	1
86-87	0	0	0
87-88	0	0	0
88-89	0	0	0
89-90	0	0	0
90-91	0	0	0
91-92	0	0	0
92-93	0	0	1
93-94	0	0	0
94-95	0	0	0
95-96	0	0	0
96-97	0	1	4
97-98	9	11	7
98-99	35	35	31
99-100	101	75	65

Table 4.9: QOS violations when using RCSP-TB, with no ABR flows

4.4 Using RCSP in the Link Sharing model

As explained in Section 3.4.2, an RCSP scheduler serving packets from an exempt class can be used to service real-time flows along with CBQ which provides a flexible set of mechanisms for allocating resources to flows. Experiments were conducted to check the treatment of real-time flows by the RCSP scheduler that has been fit into the link-sharing paradigm supported by CBQ. The effect of the RCSP scheduler on non-realtime flows was also studied.

The experiments described in Section 4.2 were repeated using the combined RCSP and CBQ scheme. The behaviour of scheduling scheme for CBR and VBR flows was exactly the same as when using RCSP for scheduling. The reason for this is that RCSP, being an exempt class, always gets the first chance to decide the packet to be scheduled next. Hence, the behaviour of RCSP used along with CBQ would be same as when RCSP is used independently.

Figure 4.2 shows the link-sharing structure used for an experiment to check the treatment of real-time flows and the effect of including RCSP on treatment of non-realtime flows. The RCSP class serves 3 CBR flows, 6 VBR-1 flows, 2 VBR-2 flows, and 1 VBR-3 flow and requires an estimated bandwidth of 60%. The remaining bandwidth is shared among two non-realtime classes, class A and class B. 28% percent link bandwidth is shared among two class A flows, F1 and F2, where F1 is given a higher priority than F2. 12% link bandwidth is shared among two class B flows, F3 and F4, where F3 has higher priority than F4. We use CBR sources for generating traffic for each of these flows at a rate equal to the output link bandwidth of 5 Mbps.

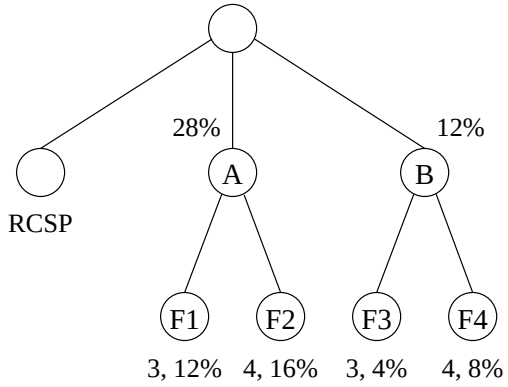


Figure 4.2: Link-sharing structure for the simulation

Table 4.10 shows the results for a simulation run of 40 s. It compares CBQ using the conventional Weighted Round Robin(WRR) scheduler with CBQ using an RCSP scheduler with respect to the treatment of real-time flows. The columns shows the results corresponding to percentage of packets delayed beyond 5 ms, and minimum and maximum queueing delays. It can be observed that with WRR, packets of CBR flows experience lesser packet delays, but with RCSP there is lesser delay jitter. For VBR flows, packet delays are lesser in case of RCSP as compared to WRR. In case of

ID	Type	Packets received	CBQ/WRR			CBQ/RCSP		
			Late (%)	Delay(ms)		Late (%)	Delay(ms)	
				Min	Max		Min	Max
0	CBR	1524	0	0	0.049	0	0.250	0.281
1	CBR	1524	0	0	0.054	0	0.261	0.292
2	CBR	1524	0	0	0.049	0	0.271	0.303
3	VBR-1	3171	0	0	3.170	0	0	1.192
4	VBR-1	3231	0	0	4.764	0	0	1.378
5	VBR-1	3199	1.282	0	21.188	0	0	1.839
6	VBR-1	3162	0.095	0	5.551	0	0	1.198
7	VBR-1	3208	0.031	0	5.462	0	0	1.207
8	VBR-1	3229	0.031	0	5.680	0	0	1.411
9	VBR-2	6741	0	0	4.950	0	0	1.328
10	VBR-2	6758	0	0	4.688	0	0	1.187
11	VBR-3	13953	1.003	0	9.812	0.315	0	9.703

Table 4.10: Treatment of real-time flows by CBQ/WRR and CBQ/RCSP

using WRR the QOS guarantees of flow 5 and flow 11 are violated, whereas, in case of RCSP scheduler none of the flows QOS guarantees are violated. From this we see that using an RCSP scheduler gives more desirable results with respect to real-time flows.

Figure 4.3 compares the treatment of the four flows F1, F2, F3, and F4. The graphs show the bandwidth obtained by each flow along the y-axis against time on the x-axis. The source corresponding to each of the flows is made to stop transmitting for short durations in order to see how the bandwidth unutilized by the idle flow is shared among the active flows. First flow F3 stops transmitting for some duration, followed by F1, F2, and then F4. In both sets of graphs the following observations can be made. Firstly, when all sources are transmitting, each flow gets at least the links sharing bandwidth allocated according to the link-sharing structure shown in Figure 4.2. Some flows get more than their allocated bandwidth. This happens

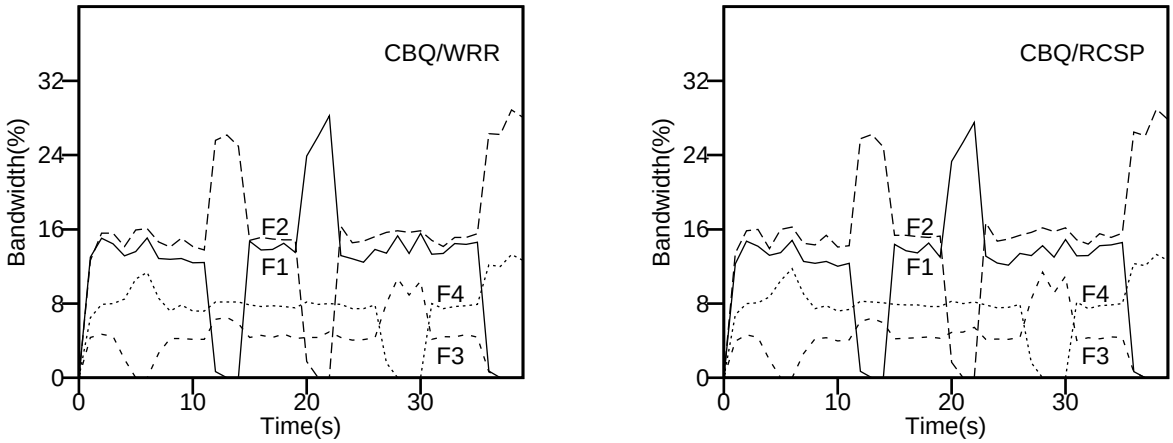


Figure 4.3: Treatment of non-realtime flows by CBQ/WRR and CBQ/RCSP

because the requirement of real-time flows may temporarily reduce below the 60% mark. The excess bandwidth is to be allocated to flows F1 and F3 which have a higher priority, and the bandwidth is to be shared in proportion to their link-sharing bandwidth. Since F1 is allocated a higher bandwidth it gets a larger part of the extra bandwidth. Secondly, when each class stops transmitting, excess bandwidth is used by the other class belonging to the same parent class. Thus, both class A and class B receive their allocated link-sharing bandwidth. However, when flow F1 stops transmitting, F2, belonging to the same parent class, does not get all the excess bandwidth, F3 eats into its share. When the data was sampled at smaller time intervals, it was observed that the bandwidth obtained by flow F2 toggles above and below class A's allocated share of 28%. As soon as class A exceeds the mark it is regulated by the link-sharing scheduler. Changing a parameter to increase the duration of the flow's history that affects the estimation of bandwidth occupied makes F2 get a bandwidth share closer to the 28% mark. Finally, the most interesting observation is that the graphs indicating the link-sharing among the non-realtime flows both when using the WRR scheduler and the RCSP scheduler are very similar. Hence, we get higher priority treatment of real-time flows by not compromising on link-sharing guidelines used in the treatment of non-realtime flows.

Chapter 5

Conclusions

In this thesis, we have studied packet scheduling algorithms used at queueing points (routers) in a network. These scheduling algorithms are important for providing quality-of-service to various flows in terms of delay, delay variation, bandwidth and loss rate. A survey of various scheduling disciplines has been presented.

We have used a simulation test-bed for our study of some select scheduling disciplines. The aim was to provide a common platform for comparing the performance of scheduling disciplines in serving different traffic mixes with their respective QOS requirements. We used the simulation, *ns* [5], developed at Lawrence Berkeley Network Laboratory. This came with implementation of CBQ. We extended this package by implementing weighted fair queueing(WFQ), and rate controlled static priority(RCSP) queueing disciplines were added to the simulator. CBR traffic generators were used for modeling audio sources, and Variable bit rate modeled video sources were implemented for generating VBR traffic.

As an extension the RCSP scheduling discipline, provision has been made for supporting traffic specifications using burst size and average rate. For this, a new regulator, based on a token bucket filter, was designed and implemented. In an attempt to utilize the flexibility of link-sharing along with the advantages of RCSP in servicing real-time flows, we have suggested a mechanism for incorporating RCSP into the link-sharing paradigm.

Several experiments were conducted to compare the performance of the scheduling disciplines in handling traffic mixes consisting of CBR, VBR, ABR, and UBR traffic types. Below we summarize the results of the experiments.

- WFQ performs well when serving homogeneous traffic (several flows with similar specifications). Its performance is acceptable when serving a mix of 2 or 3 types of VBR traffic. It is unsuitable for serving traffic mixes containing CBR, VBR, and ABR traffic types.
- Both CBQ and RCSP give good results when serving traffic mixes containing all traffic types. However, RCSP performs better under higher network loads.
- When serving traffic mixes containing only CBR and VBR flows RCSP does better than CBQ.
- RCSP using a token bucket based regulator performs slightly better than normal RCSP due to the increased flexibility in handling temporary peaks in packet arrival rates.
- RCSP incorporated into the link-sharing framework gives the same results as when used independently with respect to treatment of real-time flows.
- Incorporating RCSP into the link-sharing framework doesn't significantly affect the serving of non-realtime flows as specified by the link-sharing guidelines.

5.1 Future Work

The audio traffic used in the tests was from a simplistic constant bit rate traffic generator. Some effort has been made to model video traffic generation. If we are able to use actual audio and video traffic samples for the study, it would give us a more realistic idea about how the scheduling disciplines would perform under practical situations. The simulation study we did consider a select set of the scheduling disciplines that have been proposed in the literature, implementing some more like stop-and-go queueing and deadline based scheduling and studying them is also desirable.

Bibliography

[1] Bennet, J. C., and Zhang, H. WF^2Q: Worst-case Fair Weighted Fair Queueing. In *Proceedings of IEEE INFOCOM* (1996).

[2] Brown, R. A Fast O(1) Priority Queue Implementation for the Simulation Event Set Problem. *Communications of the ACM 31*, 10 (October 1988), 1220–1227.

[3] Clark, D., and Jacobson, V. Flexible and Efficient Resource Management for Datagram Networks. unpublished manuscript, April 1991.

[4] Demers, A., Keshav, S., and Shenker, S. Analysis and Simulation of a Fair Queueing Algorithm. *Journal of Internetworking Research and Experience* (October 1990), 3–26.

[5] Fall, K., and Varadhan, K. *ns* Notes and Documentation. URL http://www-mash.cs.berkeley.edu/ns/nsDoc.ps.gz, July 1999.

[6] Ferrari, D., and Verma, D. A Scheme for Real-Time Channel Establishment in Wide-Area Networks. *IEEE Journal on Selected Areas in Communications 8*, 3 (1990), 368–379.

[7] Floyd, S. Notes on CBQ and Guaranteed Service. Draft Document. URL http://www.aciri.org/floyd/papers/guaranteed.ps, July 1995.

[8] Floyd, S., and Jacobson, V. Link-Sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking 3*, 4 (August 1995).

[9] Golestani, S. J. A Stop-and-go Queueing Framework for Congestion Management. In *Proceedings of ACM SIGCOMM'90* (Philadelphia, Pennsylvania, September 1990), pp. 8–18.

[10] Golestani, S. J. A Self-clocked Fair Queueing Scheme for Broadband Applications. In *Proceedings of IEEE INFOCOM* (Toronto, CA, June 1994), pp. 636–646.

[11] Greis, M. Tutorial for the Network Simulator "ns". URL http://titan.cs.uni-bonn.de/~greis/ns/ns.html.

[12] Heyman, D. P., and Lakshman, T. V. Source Models for VBR Broadcast-Video Traffic. *IEEE/ACM Transactions on Networking 4*, 1 (February 1996), 40–48.

[13] Kalmanek, C., Kanakia, H., and Keshav, S. Rate Controlled Servers for Very High-Speed Networks. In *IEEE Global Telecommunications Conference* (San Diego, CA, December 1990), pp. 300.3.1–300.3.9.

[14] Keshav, S. A Control-theoretic Approach to Flow Control. In *Proceedings of ACM SIGCOMM'91* (Zurich, Switzerland, September 1991), pp. 3–15.

[15] McCanne, S., and Floyd, S. *ns* - Network Simulator(version 2), Manual Page. URL http://www-mash.cs.berkeley.edu/ns/ns-man.html.

[16] Parekh, A. K., and Gallager, R. G. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: Single-Node Case. *IEEE/ACM Transactions on Networking 1*, 3 (June 1993), 344–357.

[17] Parekh, A. K., and Gallager, R. G. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case. *IEEE/ACM Transactions on Networking 2*, 2 (April 1994), 137–150.

[18] Turner, J. New Directions in Communications, or Which Way to the Information Age? *IEEE Communications Magazine 24*, 10 (October 1986), 8–15.

[19] Verma, D. C., Zhang, H., and Ferrari, D. Delay Jitter Control for Real-Time Communication in Packet Switching Networks. In *Proceedings of Tricomm '91* (46, Chapel Hill, North Carolina, April 1991, April 1991), pp. 35–46.

[20] Wetherall, D. OTcl - MIT Object Tcl, The FAQ & Manual (Version 0.96). http://www.neosoft.com/neoscript/commands/otcl/doc/tutorial.html, September 1995.

[21] Zhang, H. Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks. In *Proceedings of the IEEE* (October 1995), vol. 83, pp. 1374–1395.

[22] Zhang, H., and Ferrari, D. Rate-Controlled Static-Priority Queueing. In *Proceedings of IEEE INFOCOM* (Sanfrancisco, CA, April 1993), pp. 227–236.

[23] Zhang, L. Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks. In *Proceedings of ACM SIGCOMM'90* (Philadelphia, PA, September 1990), pp. 19–29.