IMPLEMENTATION AND PERFORMANCE SIMULATION OF VIRTUALCLOCK SCHEDULING ALGORITHM IN IP NETWORKS

by

Nazy Alborz

B.A., Shahid Beheshti University, Tehran, Iran, 1998.

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

IN THE SCHOOL OF ENGINEERING SCIENCE

SIMON FRASER UNIVERSITY

April 2002

Approval

Name:	Nazy Alborz
Degree:	Master of Applied Science
Title of thesis:	Implementation and Performance Simulation of VirtualClock Scheduling Algorithm in IP Networks

Examining Committee:

Chair: Dr. Mehrdad Saif

____
Dr. Ljiljana Trajkovic, Senior Supervisor
School of Engineering Science

____
Dr. Stephen Hardy, Supervisor
School of Engineering Science

____
Dr. Joseph Peters, Internal Examiner
School of Computing Science

Date approved: ___________________________

Abstract

In today's high-speed packet networks that support various applications with different service requirements, congestion control is an important issue. One of the methods for preventing congestion is packet scheduling [14]. Packet scheduling in network routers can provide guaranteed performance in terms of delay, delay jitter, packet loss, and throughput.

The main objective of this thesis is to implement a model for the VirtualClock scheduling mechanism, perform a simulation based performance analysis of the VirtualClock algorithm, and compare it to three commonly used scheduling mechanisms: WFQ, Custom Queuing, and Priority Queuing. The VirtualClock algorithm monitors the average transmission rate of packet data flows. It also provides each flow with a guaranteed throughput and a low queuing delay.

We implement a scheduler model for VirtualClock and incorporate it into the IP layer output queues of an IP router using OPNET simulation tool. We measure the performance of the algorithms in terms of fairness, end-to-end delay, and amount of packet loss from different traffic flows during various time periods. We also simulate a network running several Internet applications: HTTP, FTP, IP Telephony, and videoconferencing and we observe the impact of scheduling algorithms on the performance of these applications. Our simulation results indicate similarities of VirtualClock to WFQ and to Custom Queuing. They also illustrate the differences between VirtualClock and Priority Queuing.

Acknowledgments

I would like to thank my senior supervisor, Dr. Ljiljana Trajkovic for her support, guidance, and knowledge and for giving me the opportunity to be a member of the Communication Network Laboratory. I want to acknowledge all the members of this lab for their help and for being both great friends and colleagues.

My special thanks to Dr. Stephen Hardy and Dr. Joseph Peters for their valuable comments and for being on my supervisory committee.

I would also like to thank my friend Maryam Keyvani for being a wonderful friend, as well as my parents, my brother, and my fiancé for their great support and endless love during my study period. Without you all it would be impossible.

Dedication

To my parents, for standing by me with their support and guidance throughout my graduate studies.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

1.1 Introduction

The development of communication networks enables users to transfer information in the form of voice, video, electronic mail, and computer files. The following networks illustrate the evolution steps of communication networks:

- Telephone networks
- Computer networks
- Cable television networks, and
- Wireless networks.

Although these networks are quite different, they are now able to provide services that have previously been limited to specific networks. The tendency of different networks towards converging into a single network is achieved by using digital technology. This convergence tendency does not imply that a single network technology will emerge as a substitute for all the other technologies. Instead, the new converged network enables both traditional and new communication services to be delivered over various network infrastructures. Therefore, a major challenge for the practitioners and researchers is how to interconnect these network infrastructures in a way that is extensible and secure and that provides a wide range of quality of services needed to support a variety of information delivery [18, 34].

1.2 Quality of Service (QoS)

The concept of Quality of Service (QoS) has been dramatically changed during the development of communication networks. Starting from the early days of computer networks, transmitting packets from their sources to their destinations had been the most significant goal of a network. The reliable access to the network had been a major concern in terms of QoS. Today, the rapid evolution of networks has brought up the issue of ever increasing demand for bandwidth and of simultaneous support for different types of services in the same telecommunication network. Thus, QoS has become a key factor in the deployment of today's networks and services.

Although QoS has recently been a hot issue among networking researchers, there are still ambiguities in the way they understand and define it. In general, QoS means providing consistent and predictable data delivery service in order to satisfy different application requirements [28]. QoS can be observed from two different perspectives: network users and network providers. Each of them has different QoS objectives. What a network user requires is access to a large bandwidth with the lowest possible price. On the other hand, the goal of network providers is to maximize network efficiency while meeting the specific QoS requirements of network users at the same time [4, 21]. Our main focus is on the QoS from the network providers' point of view. Common QoS parameters used for characterizing the network performance are:

- Bandwidth (throughput): number of bits or bytes transmitted over the network in a specific time period.
- Delay: the time it takes for the data packet to traverse from its source to its destination. It consists of three components: propagation delay, transmission delay, and queuing delay.
- Delay jitter: the variation in delay encountered by a data packet. This is the difference between the maximum and the minimum possible packet delay.
- Loss probability: the chance of a packet being lost in the network. There are a number of situations that may result in the loss, such as buffer overflow in the network switching nodes or a call set-up request denial.
- Utilization: the ratio of busy time to the total elapsed time in a given period. It can be measured in each of the network elements like sources, switches, and links.

QoS is the ability of the network applications and elements like hosts or routers to give some level of assurance to their traffic, by satisfying their service requirements [28].

QoS requires the cooperation of all network layers from top-to-bottom, as well as every network element from end-to-end. QoS is not able to create more bandwidth than what is already provided by the network. However, it manages the existing network bandwidth according to network users' service requirements.

There are two technologies for providing QoS [28]:

- Resource reservation (integrated services): provides guaranteed service to network traffic. This is achieved by allocating network resources to user applications according to their requested QoS and network management policy. In this framework, resource requirements are signaled from the source node and the network reserves resources according to the signal. An example is the Resource Reservation Protocol (RSVP).
- Prioritization (differentiated services): network traffic is classified to various categories, marked and prioritized. This enables the network to give preferential treatment to the applications with more stringent QoS requirements. Network resources are allocated according to network management policy and the class the traffic belongs to.

QoS is implemented in the networks through three mechanisms. A QoS–enabled network does not need all these mechanisms together. However, they can be combined in such a way to provide the requested service.

1.2.1 End-to-end mechanisms

These mechanisms operate on both ends of a connection. They can either control or adapt the behavior of a certain connection. The control mechanisms are able to control the quality of the connection. An example of control mechanisms is Call Admission Control (CAC). A network using CAC has a lower congestion probability, because this mechanism preserves the integrity of the traffic already in the network by avoiding the admission of other traffics.

The adaptation mechanisms adapt the behavior of a selected session in a connection based on certain parameters, such as congestion control mechanisms. These mechanisms reduce the traffic rate of the session in reaction to packet loss.

1.2.2 Edge mechanisms

These mechanisms operate on the user-network interface and are divided into two categories: policing and shaping mechanisms. These two categories are similar, except that the shaping mechanisms work at the network side, while policing mechanisms work on the user side.

Shaping: These mechanisms adapt the user traffic generation rate to the traffic rate parameter that has previously been negotiated between the user and the network. If a source starts generating traffic with a rate more than what it has specified in its contract, the shaper stores the incoming traffic and sends it to the network in such a way to follow the specified values in the traffic contract. These functions do not discard any packets that are conforming to the traffic contract.

Policing: These mechanisms, which operate on the user side of a user-network interface, check if the traffic generated by the user conforms to its contract with the network. If the traffic is nonconforming, the mechanism acts in such a way to make the traffic conformant. Two actions are done by the mechanism when source traffic exceeds its negotiated traffic rate:

- Packet dropping: in which the nonconforming packets are dropped.

- Packet marking: in which the nonconforming packets are marked by the network, so that in case of congestion they will be the first packets to be dropped.

1.2.3 Core mechanisms

These mechanisms operate on the network switching nodes like switches and routers and are classified into the following categories:

Buffering: When the incoming traffic to the switching nodes is larger than the output link capacity, packets are temporarily stored in a storage called buffer. There are two architectures for buffering: shared buffer and per-flow buffer. In shared buffer architecture, a common physical memory is shared by the packets being buffered. As soon as the output link becomes free, they are removed from the buffer by scheduler. A per-flow architecture allocates a specific portion of memory to incoming packets from each flow. These memory portions in the network switching elements are often called queues. A flow can be identified in different ways. In this document there are six criteria for identifying a single flow, which will be explained later on. The advantage of per-flow architecture is that it prevents greedy sources from affecting the other sources in the network. On the other hand a shared buffer architecture is very simple to be implemented. An advantage of buffering is that it increases the network throughput, because logically the larger the buffer, the higher the amount of data carried by the network.

Queue management: Queue managements are the mechanisms that choose which packet has to be dropped in case of buffer overflow. Each queue in a network switching node has a queue management mechanism associated with it. If a node has more than one queue there will be several instances of the queue management that operate independently on each queue [29]. Drop tail, Random Early Detection (RED), and Weighted Random Early Detection (WRED) are examples of queue management mechanisms [5].

- Drop-tail: is the simplest queue management technique. As shown in Fig. 1.1, this scheme, packets are dropped from the end of the queue when there is no more space in the queue buffer. Drop Tail operates on each queue independently.

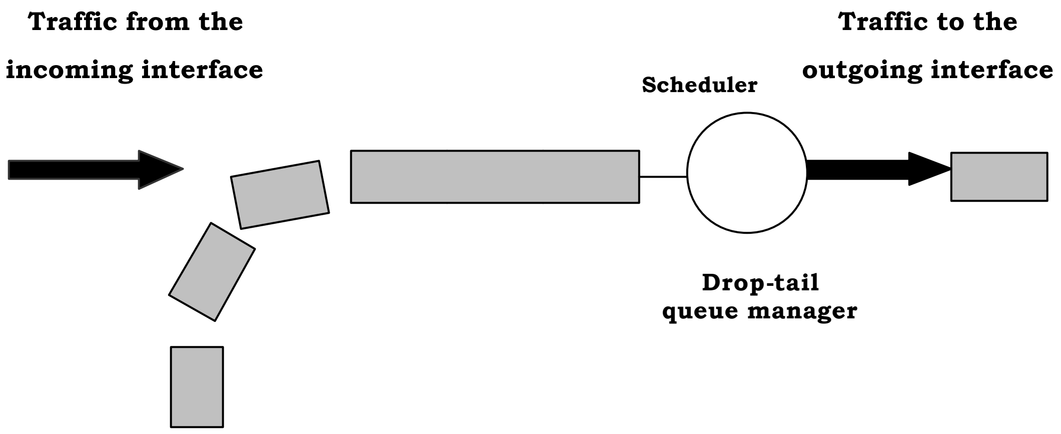

Traffic from the incoming interface
Scheduler
Traffic to the outgoing interface
Drop-tail queue manager

Fig. 1.1 Drop-tail queue management.

- RED: is an advanced queue management technique. As shown in Fig. 1.2, his scheme starts dropping packets when the number of stored packets in the queue buffer exceeds a certain threshold. The probability of loss increases with the increase of number of queued packets. When the number of queued packets reaches the maximum queue size, RED behaves like Drop-tail and discards all the incoming packets [5]. By dropping packets before the queue buffer gets full and congestion happens, RED indicates to the source to decrease its transmission rate. RED takes advantage of the TCP congestion control mechanism. Assuming the packet source is using TCP, it will decrease its transmission rate until all the packets reach their destinations. Thus, RED can be used as a way to cause sources to back off traffic using TCP [5, 13].

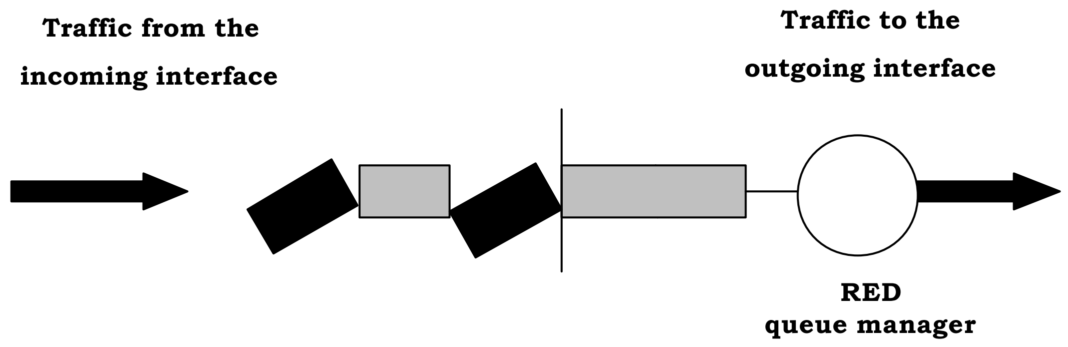

Traffic from the incoming interface
Traffic to the outgoing interface
RED queue manager

Fig 1.2 RED queue management.

- WRED: this scheme is a special form of RED. It combines the capabilities of RED together with the ability to differentiate the drop probability among various traffic flows stored in the same queue. The queue threshold in this scheme is different for different traffic classes. When the number of packets in a queue exceeds this threshold, incoming packets to the queue are dropped according to the drop probability of their classes.

Scheduling: The scheduler selects the next packet among the packets waiting in the switch buffers and sends it to the output link. The scheduler's task is easy in a shared buffer architecture, since the scheduler only selects the packet with the largest queue waiting time.

FIFO is the most common scheduling algorithm used for shared buffer architecture. However, this task becomes more complicated in a per-flow architecture [29]. The scheduler can use a variety of algorithms in order to select the next eligible packet from different queues. Since the focus of this thesis is on scheduling algorithms, they will be discussed in more detail in the next chapter.

1.2.4 Combination of QoS mechanism

In order to provide certain QoS to network applications, any combination of the above mentioned mechanisms can be deployed in networks. What should be taken into consideration is that although not all mechanisms have to be present to provide a certain level of QoS, these mechanisms are not independent. For example using a certain mechanism like “per-flow buffering” might prevent the usage of a mechanism like “FIFO scheduling”. Moreover, it is possible for the network to provide the same service using a different combination of above mechanisms. How the users’ required services are provided by the network is usually transparent to the users.

1.3 The Internet and its architecture

Internet that is based on a packet switched networking technology with layered infrastructure started in 1969 as a research project called “ARPANET” [33]. At the beginning, it only connected four computers while today over tens of billions of computers around the word are connected, exchanging messages and resources through the Internet.

1.3.1 Internet architecture

Fig. 1.3 shows a simplified network hierarchy for Internet architecture.

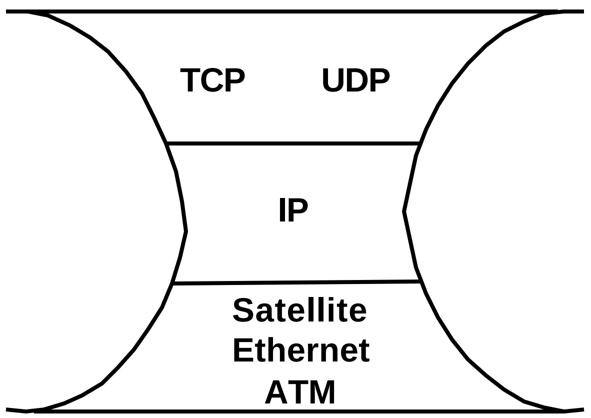


Fig 1.3 Internet architecture.

The Internet protocol (IP) at the network layer enables the Internet to interconnect heterogeneous sub-networks running on different technologies such as Ethernet, Asynchronous Transfer Mode (ATM), and Token Ring, in order to form a network of networks [10]. IP hides the heterogeneity of the underlying layers, thus it can support different applications through a common transport layer. There are two standard interfaces to the transport layer: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). IP relies on TCP/UDP to provide reliable data delivery. This "reliability" can only assure data delivery. Neither IP nor its high-level protocols can ensure delivery time or provide any guarantees for data throughput. Consequently, they can make no guarantees about when data will arrive, or how much it can deliver [28].

Current network applications such as Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), remote terminal (Telnet), remote login (rlogin), Simple Network Management Protocol (SNMP), and e-mail are now running over TCP/UDP. More and more multimedia applications which are emerging everyday are using TCP/UDP protocols to achieve end-to-end packet delivery.

1.3.2 Next generation of the Internet

Today's Internet only provides best-effort service, in which traffic is processed as quickly as possible with no guarantees for the traffic delivery time [35]. With the increase of multimedia applications running on the Internet with stringent performance guarantee requirements, such as bounded delay and minimum throughput, the Internet traffic is changing from best-effort to QoS sensitive. Since the Internet is still not ready for this change, supporting the wide range of QoS for current and future Internet applications is a major challenge. A key issue in QoS is how to manage or control the network's shared resources in terms of bandwidth on the links or buffers in the switching nodes.

1.4 Thesis organization

The thesis is organized as follows. Chapter 2 provides background on packet scheduling. It introduces a general scheduler model, discusses the packet scheduling requirements and represents the two main categories of packet scheduling algorithms. Some of the scheduling algorithms from both categories are explained in this chapter with more focus on the VirtualClock scheduling mechanism. In Chapter 3 we describe the implementation of the VirtualClock scheduling algorithm in OPNET simulation tool. We introduce the OPNET software environments and VirtualClock implementation embedded in each of these environments. Chapter 4 shows the validity and functionality of the implemented VirtualClock model through OPNET simulation of two network scenarios. In Chapter 5 we conduct two series of simulation experiments on two network scenarios in order to compare various performance aspects of the VirtualClock scheduling algorithm with several other scheduling mechanisms like FIFO, WFQ, CQ, and PQ. We use the OPNET simulation tool to simulate these network scenarios. The first scenario compares the performance of VirtualClock in comparison to other schedulers for the time periods during which different sources show various traffic behaviors. In the second scenario, we compare and analyze the effect of scheduling algorithms on the performance of several common Internet applications: HTTP, FTP, IP Telephony, and videoconferencing. We conclude the thesis with Chapter 6 and address the possible future research.

Chapter 2
Scheduling

2.1 Introduction

The next generation of the Internet supports two types of applications: best-effort and guaranteed-service applications [18]. The best-effort applications, which are already common to the Internet, are content to accept whatever performance the network gives them. For example although a file transfer application would prefer to encounter zero end-to-end delay and infinite bandwidth, it can still adapt to the available network resources. These applications are called best-effort because the network promises to deliver their data without any guarantees on performance bounds.

Beside these applications, the Internet is expected to carry traffic from applications that require performance bounds in the future. For example, an application that contains voice as a 64 Kbps data stream will be no longer usable if the network provides a bandwidth less than 64kbps [18].

The performance received by a connection depends principally on the scheduling mechanism. These mechanisms are carried out by the switching nodes located along the path between the source and the destination of a connection. Scheduling mechanisms are implemented at output interfaces of switches or routers. At each output queue of an interface a scheduling mechanism is used to choose which packet to transmit to the outgoing link. “The scheduler can allocate different queuing delays and different bandwidths to different connections. This is done by the scheduler’s choice of service order and by serving a certain number of packets from a particular connection [18].” The scheduler can also allocate different loss rates to connections by assigning a certain amount of buffer space to them. Furthermore, it is able to allocate resources, which are desired properties in the networks fairly among best-effort connections. Thus, the next generation of Internet needs scheduling disciplines in order to support:

- per-connection delay, bandwidth, and loss bounds needed for guaranteed-service applications [7]
- fair resource allocation needed for best-effort applications.

2.2 Scheduler model

In packet switched networks, scheduling is done in the network’s intermediate nodes like routers and switches. Each intermediate node consists of the following components [29]: Input buffers, output buffers, and switching fabrics. Fig. 2.1 shows the architecture of a network switch.

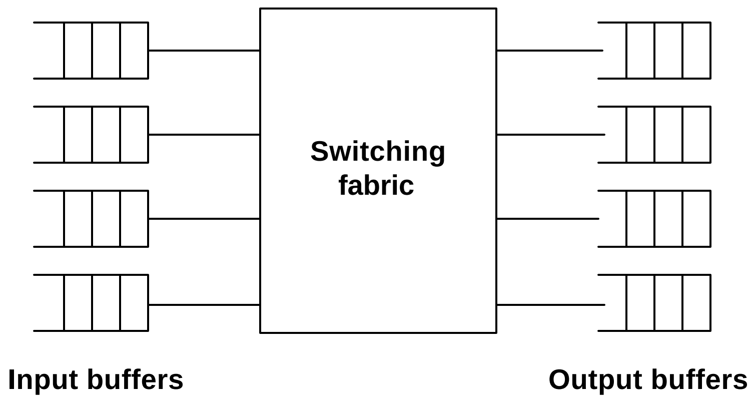


Fig 2.1 General architecture of a network switch.

Input interface is a switch component that receives packets from other intermediate nodes and selects the best output link in order to send the packet. The output interface is in charge of selecting the best packet in order to send it to the outgoing link. The switching fabric is a component that transfers packets from the input link to the output link.

Buffering can be done at any of the above mentioned components in a packet network's intermediate nodes:

- Input buffers: some switches have small input buffer spaces so that they only hold packets for the time period they are being forwarded to the switch fabric. For the other switches, all the buffers are located at the inputs [18].
- Output buffers: these are the most commonly used buffers. They store packets as they wait for their turn to be sent to the outgoing link [18, 29].
- Mixed input and output buffers: these buffers are usually employed for the case in which the switching fabric is not fast enough. So, in order to prevent packet loss due to the low forwarding speed of the fabric, a small amount of memory resides on the input interfaces. However, the main buffering point is still located at the output interfaces [29].
- Switching fabric's buffers: this can be the case of a shared buffer for all the output interfaces like FIFO scheduling.

Scheduling algorithms are located on the output interface of the network switches. Each interface has its own instance of the scheduler, i.e., different scheduling algorithms can be used on different interfaces.

The architecture of a scheduler is shown in Fig. 2.2. Each scheduler consists of two main components: classifier and scheduler. The classifier is in charge of allocating packets into different queues according to the scheduling classifier scheme. The scheduler selects the next best packet from the queues according to its appropriate scheduling algorithm.

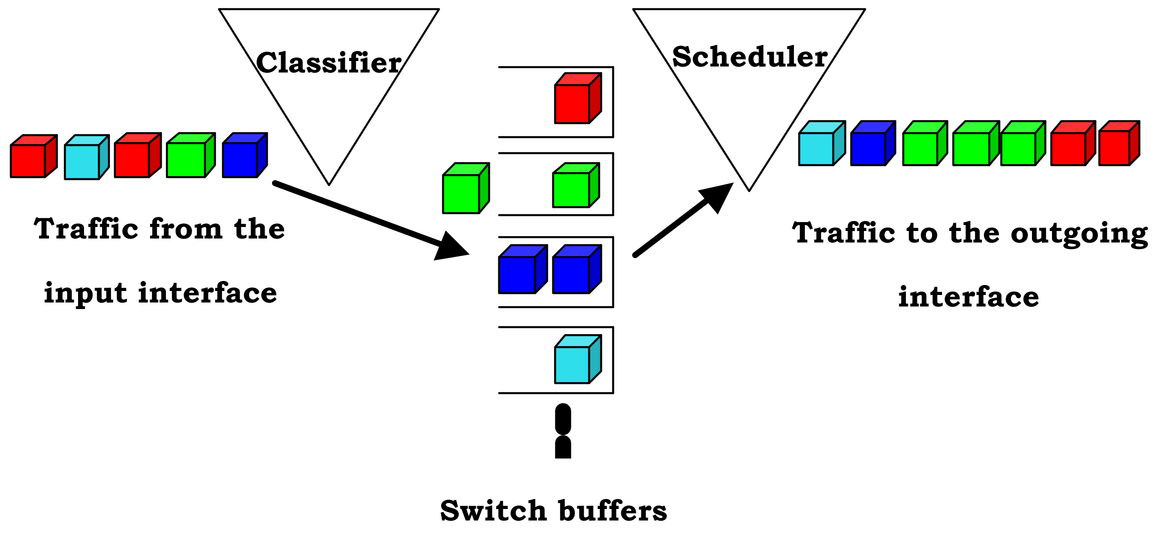

Classifier
Scheduler
Traffic from the input interface
Traffic to the outgoing interface
Switch buffers

Fig. 2.2 The model of a scheduling algorithm.

2.3 Scheduling requirements

In the design of scheduling schemes, different trade-offs can be considered in terms of the following five requirements [12]. Depending on the specific situation, some of these requirements may be more important than others and the decision for the best choice is made given the particular situation.

- Complexity: the scheduling schemes are different in terms of both control and hardware implementation complexity. The complexity of the algorithm is important because the router needs to pick up a packet to send out to the output link every time a packet departs. The frequency of the packet departure depends on the speed of the output link and it can be once every few microseconds. Thus, a scheduling scheme should require only a few simple operations. It also has to be implemented inexpensively in hardware [18].

- Fairness: Since a scheduling scheme assigns a share of network resources in terms of output link bandwidth and buffer capacity, fairness is a property needed for supporting best-effort applications [18].
- Isolation (protection): is the property of not permitting the misbehaving users to affect the well-behaving users. Misbehavers are the users that send packets at a rate faster than their fair shares [37].
- Efficiency: to guarantee certain performance requirements a CAC policy is needed in order to limit the number of guaranteed-service connections [18]. “A service discipline is more efficient than another one if it can meet the same performance guarantees under a heavier load of guaranteed-service traffic [37].”
- Performance: as discussed before the main goal of a scheduling algorithm is to guarantee performance bounds for connections in terms of throughput, delay and loss. The network and its users agree upon certain traffic parameters. The users should not exceed the specified bounds in the agreement and the network guarantees to provide the connection’s service requirements [18]. Guaranteeing the performance bounds is a difficult problem in

today's networks, since all the schedulers along the connection's path have to take part in providing it [18].

2.4 Classification of scheduling algorithms

Packet scheduling mechanisms are classified into two categories: work-conserving and non-work-conserving [37]. A work-conserving scheduler is idle when there are no packets waiting in the router's queues. In a non-work-conserving scheduler, each packet is assigned a time when it has to be sent to the output interface. The scheduler remains idle and no packet will be transmitted until the next packet is eligible for transmission [18, 37].

One of the attributes of work-conserving algorithms is that they have the minimum average total queuing delay. The average of total queuing delay is calculated on all the flows that have to be served. In addition, the average total queuing delay value is equal for all work-conserving algorithms. In other words, although different schedulers use different algorithms to choose the next best packet to transmit, the overall average queuing delay is always the same. This shows that some algorithms serve some flows faster at the expense of other ones [29].

A certain question arises at this point: why do we need to use non-work-conserving algorithms and waste bandwidth by leaving the link idle until the packets eligibility times arrive? The answer is that non-work-conserving algorithms make the traffic flow arriving at the switches more predictable by keeping the link idle for short periods of time [18]. These algorithms are usually the best choice for networks with real time traffic since they can provide specific bounds on delay and delay jitter to the flows. This is achieved by delaying packets to meet special delay requirements.

2.5 Scheduling algorithms

One of the most important functions of scheduling algorithms is to select packets for transmission to outgoing links. Beside the classical First In First Out (FIFO) algorithm and the simple priority based queuing, a large number of new scheduling algorithms have been proposed during the past decade. All these algorithms are the variants of two fundamental disciplines, Generalized Processor Sharing (GPS) and Earliest-Deadline-First (EDF) [3].

GPS divides the resources among different traffic flows according to their requirements. EDF assigns a deadline to each packet. It attempts to achieve the required QoS of the flows by serving the flows' packets in the increasing order of their deadlines [3]. The packets deadlines are associated with their maximum tolerable delay and are calculated by adding their arrival times to their maximum tolerable delays. Delay earliest due date and jitter earliest due date are examples of such algorithms.

This section focuses on the first category of the scheduling algorithms (GPS) with more emphasis on the VirtualClock algorithm as a special case. We also review some of the other GPS algorithms and outline their properties.

2.5.1 FIFO

First In First Out (FIFO) or First Come First Serve (FCFS) is one of the simplest scheduling algorithms. In this mechanism, packets are served in the order in which they arrive to the switching node. FIFO is a work-conserving algorithm and has a very low complexity, so it is one of the most commonly implemented algorithms in the networks. There are some limitations for FIFO as follows.

- It is not able to provide fairness in resource allocation to different flows. Nevertheless, this limitation is not very important for best-effort applications.
- It cannot provide any performance guarantees in terms of delay, delay jitter, or throughput to the real time applications. Thus, multimedia applications do not work well with FIFO schedulers [29]. One way to provide a delay bound is to limit the buffer size, so that the packets are guaranteed to be sent in less than the time it takes to serve a full queue. A disadvantage of this solution is that it increases the packet loss probability, which is a consequence of the high buffer overflow probability.

2.5.2 Generalized Processor Sharing (GPS) algorithm

Generalized Processor Sharing (GPS) is an ideal scheduling algorithm [25]. In this algorithm, packets from each flow are classified into different logical queues. GPS serves non-empty queues in turn and skips the empty queues. "It sends an infinitesimally small amount of data from each queue, so that in any finite time interval it visits all the queues at least once [18]." There can be a service weigh associated with each queue. Queues receive service according to their associated weights.

Following are the variables used in a GPS algorithm:

- f_i: The share of bandwidth reserved by flow$_i$
- $W_i\ (t_1,\ t_2)$: amount of traffic served from flow$_i$ during the time period (t_1, t_2).

A connection flow is defined as backlogged if it has packets that are either receiving service or waiting for service in its queue [18]. Thus, GPS serves each backlogged connection with minimum rate equal to its reserved rate at each instant. The extra bandwidth not used from other connection flows is distributed among all the backlogged connections in proportion to their reservation. In other words, during an arbitrary interval (t_1, t_2), for any pair of backlogged connection flows i and j the following equation holds [18]:

$$W_i\ (t_1,\ t_2) /\ W_j\ (t_1,\ t_2) = f_i / f_j \tag{2.1}$$

or

$$W_i\ (t_1,\ t_2)\ /\ f_i = Constant. \tag{2.2}$$

Because GPS posses the properties of ideal fairness and complete isolation, a lot of research studies have been done on it. However, GPS is not implementable because serving an infinitesimal amount of data from each non-empty queue is not possible. Thus, various emulations of GPS have been proposed in the literature. The following sub-sections describe some of these emulations.

2.5.3 VirtualClock

The idea behind the VirtualClock algorithm was derived from Time Division Multiplexing (TDM) systems. A TDM system eliminates interference among users because individual user channels (flows) can transmit only during specific time slots. The disadvantage of a TDM system is that users are limited to constant data transmission rates and the channel capacity is wasted whenever a slot is given to a flow that has no data to send at that moment. The purpose of the VirtualClock algorithm is to maintain the guaranteed throughput and firewalls of a TDM system, while still achieving the statistical multiplexing properties of packet switched networks.

The algorithm makes the statistical data flow resemble a TDM channel by assigning each data flow a virtual clock. Each virtual clock advances one tick at every packet arrival from a specific flow. The tick step is the mean packet inter-arrival time that has been specified by the flow. Thus, each virtual clock carries the expected arrival time of the packet. If a flow sends packets according to its specified average rate, its virtual clock follows real time. The algorithm stamps the packets with their own "virtual clock" values and transmits the packets in the ascending order of these stamps. Nevertheless, there is a major difference between a TDM system and a network controlled by a VirtualClock scheduling algorithm. The difference is that unlike TDM, a VirtualClock controlled network can support data flows with distinct throughput rates. The network reservation protocol determines how large a share of bandwidth each flow needs on average. Then, according to the flow's reserved transmission rate, the VirtualClock algorithm determines which packet should be forwarded next in case there is more than one packet waiting [38].

Choosing flow parameters:

We consider the following parameters for each flow$_i$ entering a switch in a network:

- AR_i, average transmission rate (packets/sec)
- IR_i, packet inter-arrival time (sec)
- AI_i, average observation interval (sec).

Choosing the *AI* value for each flow is very important. *AI* value of a flow is chosen such as:

$$\text{total transmitted data (over } AI) / AI = AR \tag{2.3}$$

and the range of possible values for *AI* is: $1/AR \leq AI \leq$ total flow duration.

The value for *AI* should be small enough to give the network sufficient control, nevertheless it should be large enough to tolerate variations in packet arrival pattern [38]. For example if the lower bound value is chosen for *AI*, the source is obliged to send packet at a constant rate. On the other hand, choosing an upper bound value (total transmission time) for AI, allows the source to send packet in any arbitrary manner.

VirtualClock Algorithm:

The algorithm uses two control variables for each flow, Virtual Clock *(VC)* and auxiliary Virtual Clock *(auxVC)* [38]. The following two functions are performed by the VirtualClock algorithm:

Data forwarding:

- When the first packet is received from flow$_i$, VC_i and $auxVC_i$ are both set to the real time.
- Upon receiving each packet from flow$_i$,

a) $Vtick_i \leftarrow 1/AR_i$

b) $auxVC_i \leftarrow \max$ (real time, $auxVC_i$)

c) $auxVC_i \leftarrow auxVC_i + Vtick_i$

$VC_i \leftarrow VC_i + Vtick_i$

- Stamp the packet with $auxVC_i$ value.
- Insert the packet in its outgoing queue.
- Serve the packets according to their increasing stamp values.

Flow monitoring: The algorithm calculates a control variable: $AIR_i = AR_i \times AI_i$ for each flow$_i$ (in packets). Upon receiving AIR_i packets from $flow_i$, the following conditions are checked:

- If (VC_i - real time) > T (a control threshold), then the source of flow$_i$ is warned
- If (VC_i < real time), let VC_i = real time.

Thus, the VC variable plays the role of a flow meter, and it is increased according to the flow's negotiated packet arrival rate. Hence, the difference between a flow's VC and the real time shows how closely a flow is following its specified rate [38].

By introducing a second parameter called the auxiliary Virtual Clock, the algorithm prevents flows from accumulating credits. Consider the case when a source sends a burst of packets after remaining idle for a while. In this situation, although the *VC* value might fall behind the real time, the use of the auxiliary Virtual Clock will cause the packet's *auxVC* stamp to be updated with the real time. Thus, the traffic burst will be interleaved with packets from other flows. Therefore, the *auxVC* is used to order packets from distinct flows. By serving packets in the order of their *auxVC* values, the algorithm assures that flows use the bandwidth according to their specified packet arrival rates. Thus, although nonconforming flows can use free bandwidth, they cannot affect conforming flows.

VirtualClock functionalities:

Firewall protection: The VirtualClock algorithm provides firewall protection between distinct flows by serving packets in the order of their *VC* values. In the networks using a VirtualClock scheduler, if one or more flow sources exceed their specified average packet generation rates; these nonconforming sources will not affect other conforming sources. Because, the more nonconforming the flow, the worse service it gets from the VirtualClock scheduler. When a flow source generates packets at a rate higher than it is expected, the *VC* value of the flow's appropriate queue advances beyond the real time; thus the flow's packets will be placed at the end of the service queue. In a VirtualClock controlled network, although nonconforming sources can use the idle network resources, they cannot degrade the service to other conforming flows.

Flow prioritization: The VirtualClock algorithm is able to provide priority service to flows that require guaranteed performance. The VirtualClock can divide the incoming flows into two categories: guaranteed service and best effort service. Prioritization is done by replacing the real time by real time – *P* in the VirtualClock algorithm for guaranteed service flows. *P* is the priority value. The chosen *P* should be large enough to separate priority flows from non-priority flows. However, the algorithm prevents the priority queues from making unfair use of network resources and affecting other flows. If so, their *VC* value of such priorities will run ahead of the real time and ultimately they lose their priority.

The priority value for best effort traffic flows is set to -∞, so they will be stamped by *VC* value of ∞. Thus, these packets will be located at the end of service queues and will receive low priority service.

Flow monitoring: The VirtualClock is able to monitor the flows average throughput rate using the *VC* Variable. Each flow is monitored periodically by comparing its *VC* value with the real time, as mentioned earlier. Thus, the algorithm can deliver measurement information to other network control functions. It also can provide feedback to flow sources when the flow's actual packet rate significantly exceeds the negotiated rate. The flow can either be checked:

- every *AI* time period, or
- after receiving every *AIR = (AR ´ AI)* packets from the flow.

An advantage of using the second option is that the scheduler can react to traffic changes faster, if the *AI* value is large. The above options are similar in cases where the flow is transmitting packets at its specified traffic rate.

2.5.4 Weighted Round Robin (Custom Queuing)

Weighted Round Robin (WRR) is a simple emulation of GPS. The difference between GPS and WRR is that WRR serves a certain amount of data instead of sending an infinitesimal amount of data from the queues [18]. The served data can be in the form of packets or bytes.

In this algorithm each queue has a weight that allows sending a certain amount of data from each nonempty queue. The weight is usually a percentage of the whole bandwidth. "This algorithm is a closer approximation for GPS when all connection flows have equal weights and all the packets have the same size (in case of packet WRR) [18, 29]." When different traffic flows have different weights, the WRR algorithm serves the flows in proportion to their weights. In cases where there are different packet sizes for different flows in order to achieve a normalized set of weights for the flows, the WRR algorithm divides each flow's weight by the average packet size of that flow.

There exist two problems that cause WRR not to emulate GPS correctly [18]:

- In practice the source's packet sizes may not be predictable, so a WRR algorithm cannot allocate bandwidth fairly to different flows.
- At time scales shorter than a round trip time, the algorithm is not a fair algorithm since some flows may get more service than the others. WRR tends to be fair only at larger time scales.

A limitation of the WRR algorithm is that its performance depends on the packet arrival pattern. For example, when a packet arrives to a queue just after the queue has been served, it has to wait in the queue for a whole round time before getting served, no matter how important the packet's flow is.

2.5.5 Deficit Round Robin

Deficit Round Robin (DRR) is a modification of WRR. The improvement to this algorithm is that it can handle variable packet sizes without knowing the average packet size of the flows. DRR services the queues in a round robin order. Each queue is allowed to send a certain amount of bytes in each round. There are two variables associated with each queue in this algorithm: *Quantum* and *Deficit Counter* [29, 31]. *Quantum* represents the number of bytes that each queue can send on its turn. The *Deficit Counter* variable is used to keep track of the credit each queue possesses for sending traffic and is initialized to zero.

The scheduler checks each queue in turn and adds the queue's *Quantum* to its *Deficit Counter* variable. DRR tries to send the number of bytes equal to the queue's *Deficit Counter* from the queue. If the packet size in bytes is smaller than the *Quantum* value of the packet's queue, the packet is served and the *Deficit Counter* is reduced by the packet size in bytes. If the packet size is larger that the *Quantum* value, the scheduler moves to the next queue without serving the packet [31]. The queue of the unserved packet keeps the credit it obtained from not sending a packet on the previous round. This credit is used by the queue on the next service rounds.

The importance of the DRR algorithm is that it is easily implemented. However, like WRR, it is unfair at time scales smaller than a round trip time.

2.5.6 Packet by packet Generalized Processor Sharing and weighted Fair Queuing

Packet by packet Generalized Processor Sharing (PGPS) and Weighted Fair Queuing algorithms are both approximations of GPS. The difference between these algorithms and GPS is that unlike GPS they don't service an infinitesimal amount of data from each queue. Another improvement which has been done to GPS in these algorithms is that, in the case of flows' variable packet sizes, they do not need to know the average packet size in advance [25, 26]. WFQ is essentially the same as PGPS, but they were independently developed. Thus, we only focus on explaining WFQ.

WFQ was developed by Demers, Keshav, and Shenker in 1989. The idea behind the algorithm is that for each packet, WFQ computes the time at which service to the packet would be finished, deploying a GPS scheduler. Then the WFQ scheduler services the packets in the increasing order of their finish times [18]. "In other words WFQ simulates GPS on the side and uses the results of this simulation to determine the packets' service order [18]."

Computation of *finish time*:

For computing packet *finish times,* consider the following variables and notations:

- *R(t), round number* at time *t*. *Round number* is the number of rounds a bit-by-bit round robin scheduler has completed at a given time.
- *P(i, k, t)*, the size of *k* th packet that arrives to the *i* th queue at time *t*.
- *F (i, k, t)*, the *finish time* of the *k* th packet that arrives to the *i* th queue at time *t*.
- *W(i),* the weight of *i* th connection.
- *Active queue* is a queue in which the largest finish number of a packet either in that queue or the last served from the queue is larger than the current *round number*.

The length of a round, i.e., the time it takes to serve one bit from each active queue, is logically proportional to the number of active queues. This *finish time* for packets arriving at both *active* and *inactive* queues is calculated as follows.

The finish time of packets arriving to an *inactive* connection is the sum of the current *round number* and the packet size (bits)/queue weight, which is the time it takes a bit-by-bit round robin scheduler to finish the service of the packet. For *active* queues the finish time is the sum of the largest finish time of a packet in its queue or last served from the queue and the arriving packet size (bits)/queue weight [5, 18]. In other words, combining the above statements *finish time* for packet *i* is calculated as:

$$F\,(i,\,k,\,t) = \max\,\{F\,(i,\,k\text{-}1,\,t),\,R\,(t)\} + P\,(i,\,k,\,t)/\,W\,(i). \qquad (2.4)$$

A problem with this algorithm is "iterated deletion" [18]. The iterated deletion problem happens when a queue becomes *inactive* and is deleted from the list of *active* queues, perhaps causing other queues to become *inactive*. A solution to this problem is to compute the list of deletions at any given time, which is a difficult task. A WFQ scheduler updates the *round number* on every packet arrival and departure, which is a complex computation happening once every few microseconds. This is one of the major problems with the implementation of WFQ in high-speed networks [18].

Another problem is that WFQ has to associate each data flow to a separate queue, which brings up the scalability problem. A solution to this problem is using hashing techniques.

WFQ also needs to keep track of the per-connection scheduler state, which causes implementation complexity and is expensive for schedulers that support a large number of flows [18].

Despite all these problems many manufacturers such as Cisco, Inc. and FORE Systems, Inc. (manufacturer of ATM switches) have been using variants of WFQ in their routers and switches as of 1996 [5, 18]. Two variants of WFQ are Self-Clocked Fair Queuing (SCFQ) and Start-time Fair Queuing.

2.5.7 Self-Clocked Fair Queuing

As discussed above a major problem with WFQ was the complexity and cost for computation of *round numbers* upon each packet arrival. SCFQ which was proposed by Golestani in 1994, was a solution for speeding up the computation of *finish numbers* [11]. Instead of using the *round number* in this algorithm, the *finish time* of the packet currently receiving service is used to update the queue's *finish time* upon arrival of a packet to an empty queue. So, *finish time* for packet *i* is computed as [18]:

$$F(i, k, t) = \max\{F(i, k\text{-}1, t), CF\} + P(i, k, t)/W(i) \,. \qquad (2.5)$$

Finish time of a packet is set to the maximum of *CF* (finish time of the packet presently receiving service) and the finish time of last packet in the queue, plus the time it takes to finish service to the packet. Although SCFQ does not have the computational complexity of WFQ, it is an unfair algorithm over short periods of time. SCFQ has looser delay bounds than WFQ and consequently causes greater unfairness over shorter time scales. Hence, there is a trade-off between the lower computational cost and the fairness of SCFQ and WFQ [18, 29].

2.5.8 Start-time Fair Queuing

This algorithm is a variant of SCFQ. It provides lower implementation complexity than WFQ, without having loss, delay bounds, and unfairness properties of SCFQ. In this scheme in addition to *finish time* another parameter called *start time* is calculated for each packet entering a queue. For packets arriving to an *inactive* queue the *start time* will be the current *round number* and for packets arriving to an *active* queue the *start time* is the *finish number* of its previous packet [18].

If there exist some packets to send, the round number is set to the *start time* of the packet that is currently receiving service; otherwise the round number will be set to the maximum of the *finish numbers* of the packets that have been sent until that time. The packet's *finish time* is computed by adding up its *start time* and its packet size (bits)/queue weight [18].

2.5.9 Priority Queuing

Priority Queuing is one of the first solutions for providing different services to different flows. It allows prioritizing traffic flows.

The algorithm assigns packets from traffic flows to different queues with various priorities. The Priority Queuing (PQ) algorithm functions as follows.

The highest priority queues receive service until they have packets. Thus, the first high priority queue has the entire bandwidth of the output link available. The second priority queue has the entire link bandwidth decreased by the amount used by the first priority queue and so on. Thus, the traffic in each priority queue is influenced by the queues with higher priorities [5, 29].

PQ is useful for assuring that mission critical traffic gets priority treatment. "For example, Cisco uses PQ to ensure that important Oracle based sales reporting data gets to its destination ahead of other less critical traffic [5]."

This algorithm is not fair because the lowest priority flows can starve if highest priority flows have large amounts of traffic. This is a work-conserving mechanism and functions well when a network has a small amount of high priority traffic. In addition, there might be situations in which the received service is much better than a required service delivered by a high priority flow. In these cases the algorithm is not able to degrade the service to the high priority flow in order to improve service to other flows. Thus, Priority Queuing is not a good choice for today's networks, which carry both best effort and guaranteed service traffic.

2.6 Theoretical comparison of scheduling algorithms

In this section we compare several well-known scheduling algorithms. Table 2.1 compares general performance aspects of these algorithms. In the second column, value "n", used for computing implementation complexity of the algorithms, is the number of active queues at each time instance. Active queues are the queues that either have packets stored in them or are in the process of packet transmission. In the fourth column of Table 2.1, by algorithm "fairness" we mean a fair throughput comparable to what ideal GPS algorithm provides. In the last column of the table, a "small" delay bound means a delay bound that is a small additive constant larger than the bound GPS algorithm provides [32].

Algorithm	**Implementation complexity**	**Category**	**Fairness**	**Delay bounds**
GPS	Impractical	Work-conserving	Fair	0
VirtualClock	O(log n)	Work-conserving	Fair in if flow arrival rate is known	Small
CQ	O(1)	Work-conserving	Fair in case of identical	Small

			packet sizes	
DRR	O(1)	Work-conserving	Fair	Large
WFQ	O(n)	Work-conserving	Fair	Small

Table 2.1 Comparison of scheduling algorithms.

Table 2.2 shows the scheduling algorithms which best serve particular classes of traffic and their corresponding application types in today's Internet.

Algorithm	VirtualClock	CQ	WFQ	PQ
Traffic	Consistent and predictable streams	Constant packet size: ATM cells	All: CBR and unpredictable with constant and variable packet sizes	Unpredictable mission critical
Application	Voice over IP	No Internet application	All: Best effort and guaranteed service	Control messages

Table 2.2 Appropriate scheduling algorithm for various applications and traffic types.

2.7 Scheduling algorithms in OPNET

The OPNET simulation tool has some scheduling algorithms implemented in its IP routers: FIFO, Priority Queuing (PQ), Custom Queuing (CQ), and Weighted Fair Queuing (WFQ). These algorithms are also the most common algorithms used by Cisco, Inc. in their router products. The following describes the features of these algorithms and their implementation in the OPNET simulation tool.

2.7.1 FIFO

This algorithm is uses one common buffer space in which packets are stored in case of output link congestion. The only configurable parameter for this scheme is the *Maximum Queue Size* (pkts). When this limit is exceeded, the packets will be dropped from the buffer.

2.7.2 Priority Queuing

In OPNET's PQ implementation there exist four priority queues: High, Normal, Medium, and Low. Each packet is assigned to one of these priority queues based on an assigned priority. Packets that are not classified by these assigned proprieties will fall into the Normal queue. Fig. 2.3 shows a general architecture of the implemented PQ in OPNET.

2.7.3 Custom Queuing

OPNET's Custom Queuing model is based on the byte by byte WRR algorithm. This algorithm works by cycling through the queues in round-robin fashion and sending the number of bytes according to the portion of allocated bandwidth from each queue.

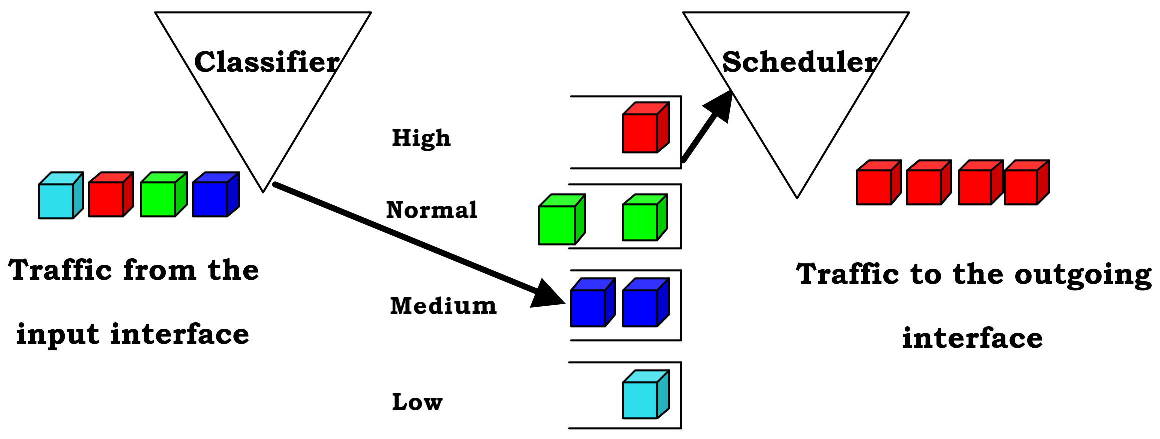

Classifier
Scheduler
High
Normal
Medium
Low
Traffic from the input interface
Traffic to the outgoing interface

Fig. 2.3 Priority queuing scheduler in OPNET.

There is a configurable variable (*Byte Count*) associated with each queue. This variable specifies how many bytes of data should be delivered from the current queue before the system moves into the next queue. When a particular queue is being processed, packets are sent until the number of sent bytes exceeds the *Byte Count*, or until the queue is empty. There is a limitation of 16 on the maximum number of definable CQ queues on each IP router interface. Fig. 2.4 shows how the queues are served employing the CQ algorithm in OPNET.

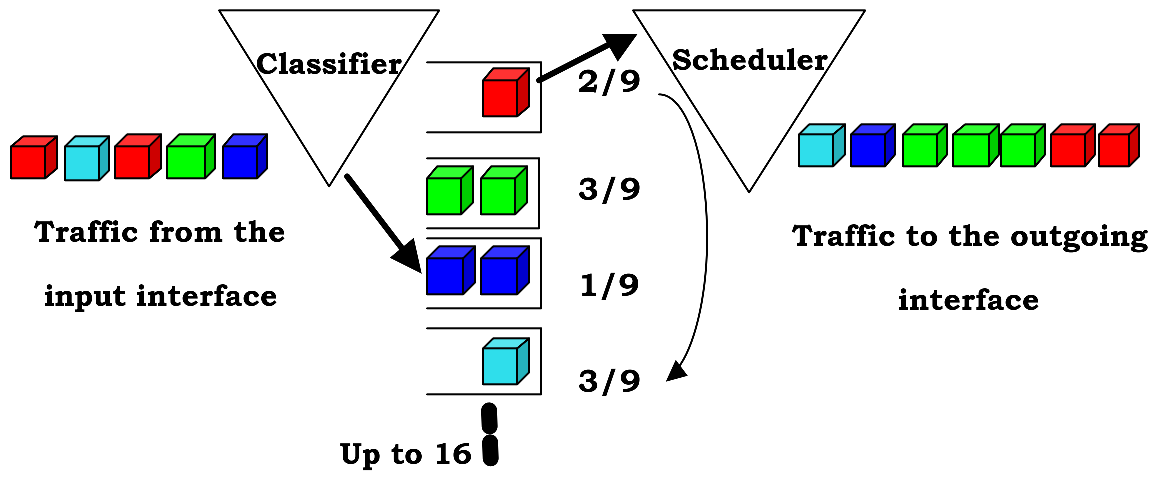

Classifier
Scheduler
2/9
3/9
1/9
3/9
Traffic from the input interface
Traffic to the outgoing interface
Up to 16

Fig. 2.4 Custom Queuing scheduler in OPNET.

2.7.4 Weighted Fair Queuing

WFQ model in OPNET is based on the Weighted Fair Queuing algorithm explained earlier in this chapter. In this model, the queue whose head packet has the lowest *Finish time* will be served first by the WFQ scheduler. The packets' *Finish times* are calculated using the WFQ equation. In addition, there is a configurable variable (weight) that defines the number of packets served from each queue upon their turn. Fig. 2.5 shows the function of the WFQ scheduler and the proportion of the sent packets based on the queue weighs.

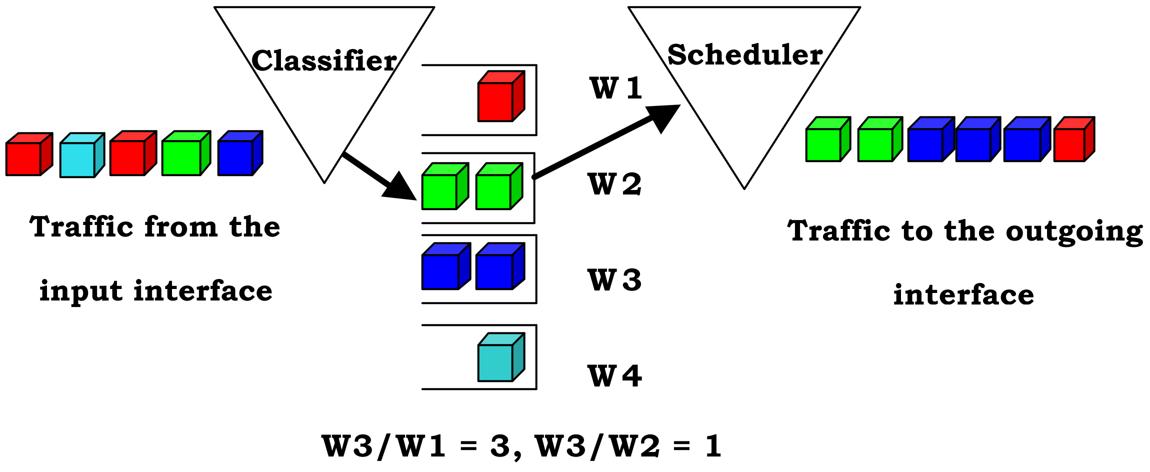

Classifier
Scheduler
W1
W2
W3
W4
Traffic from the input interface
Traffic to the outgoing interface
W3/W1 = 3, W3/W2 = 1

Fig. 2.5 WFQ scheduler in OPNET.

.

Chapter 3
Implementation of VirtualClock Algorithm

3.1 Introduction

In this chapter, we describe the implementation of the data forwarding function of the VirtualClock scheduling algorithm. To implement the algorithm, we use the OPNET simulation tool. OPNET is an intelligent network management software, developed by OPNET Technologies, Inc. founded in 1986. Since then eight versions of the software have been released.

The VirtualClock algorithm model is implemented in the Internet layer of OPNET's IP routers and is created using the state transition diagram model, coded in embedded C. The VirtualClock model has been contributed to the OPNET Model Depot site, where users can download the model in order to control congestion in their simulated IP networks.

3.2 OPNET environment

OPNET provides an environment that supports modeling of communication networks and distributed systems [22, 23]. The OPNET environment contains tools for all phases of a study, including design, simulation, data collection, and data analysis.

There are three layers for the hierarchical structure of an OPNET model: Network layer, Node layer, and Process layer. Each of these layers has an editor incorporated with them in the OPNET environment.

3.2.1 Project editor

The project editor is used to construct and edit the topology of communication network models. The interconnection and position of network nodes are adjustable in this editor. It also provides operations to support the simulation and analysis of these network models. This editor is the highest modeling level in OPNET in the sense that it uses the objects that are defined in the other modeling editors.

3.2.2 Node editor

The node editor is used to define the structure and behavior of nodes used in the network domain (such as clients, servers, switches, routers, bridges, and firewalls). Each network node is made up of several modules. Each of these modules defines one aspect of node behavior such as data generation, data storage, data forwarding, etc. These modules are connected together via packet streams or statistic wires. In addition to the node structure, this editor defines the interface of a node model, which determines which aspects of the node model are visible and definable by the user.

3.2.3 Process editor

The process editor is used to specify the behavior of process models, which define the functionality of the modules used by node models. In addition to the behavior of a process, this editor defines the model's interfaces, which determine what characteristics of the process model are visible and adjustable by users.

Process models are defined by finite state machines, which are composed of two main components: states and transitions. States refer to an object that corresponds to one of the situations that the process may find itself in. The process is always exactly in one state at a time. The process can move between states upon receiving some interrupts. The interrupts fulfill the conditions that make the process move from one state to another. The interrupts may be originated either from the process itself or from another process, called a parent process to the invoked process (child process).

The operation of each state is defined in a distinct block written in embedded C or C++ code. These blocks are called executives. The executives of a state are split into two sections, called enter and exit executives. The enter executives are executed when a process enters a state and the exit executive are performed while the process leaves a state to enter another state. States are divided into two categories: forced states and unforced states that differ in execution timing. In unforced states there is a pause between the enter and exit executives. Once the execution of an enter executive is finished, the process returns the control to the process that has invoked it. The invoked process is suspended until the it is invoked again. At this point the exit executive of the blocked state is executed. In the forced states, the exit executives are executed by a process immediately after completion of the enter executives. For this reason the exit executives of forced states are usually left blank.

Fig. 3.1 shows the execution flow through an unforced state. In this figure, step (1) shows the time when the process completes the enter

Executive immediately after it enters the unforced state. The state blocks at this point until a new invocation takes place.

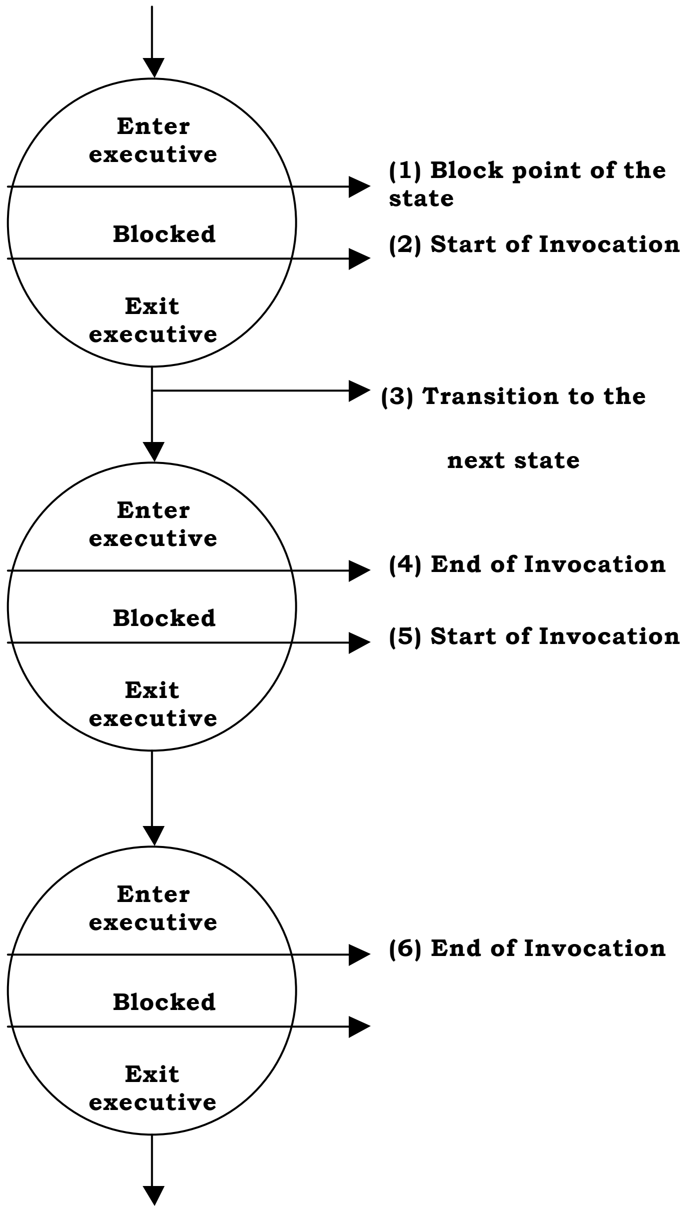


Fig. 3.1 The execution flow of the unforced states.

Step (2) is when a new invocation to the unforced state occurs. At this point the exit executives of the unforced state is executed and the process proceeds to the next state. Step (3) shows the transition to the next state. This state can either be a new state or the same unforced state depending on the transition conditions. Step (4) is when the execution of the Enter executives of the next state is finished and the process is blocked again and ….

3.2.4 Network-wide database and Common Attribute objects

This database contains some node model attributes that are common to more than one object in the network model. Node models that use this database include a unique object in their network models, which is called the *Common Attribute Object*. These objects serve the following two functions:

- They define the *Value Combination* for the structured attributes that are commonly used by some node models like routers, workstations, etc. Each *value Combination* that is referenced by a specific name is a set of sub-attribute values. It means that users can specify that an object of their network uses a specific *value Combination.* This can be done by selecting the *Value Combination*'s unique name as the attribute value of a node used in their network.
- They parse the structured attribute and store the information into the network-wide attribute database.

3.3 VirtualClock process model

The algorithm is implemented in OPNET's *ip_output_iface* process model [1, 2]. This process model is a child process to the IP layer process model (*ip_rte_v4*) of all IP routers. Fig.3.2 shows the internal structure of an IP router node model in OPNET. The *ip_rte_v4* process model is marked by a black circle in Fig. 3.2. The *ip_output_iface* child process is invoked whenever there is a scheduling algorithm selected by the user for the IP router. This selection has been done by choosing the appropriate scheduling mechanism as a value to the *Queuing Scheme* attribute of each of the interfaces in the IP router objects.

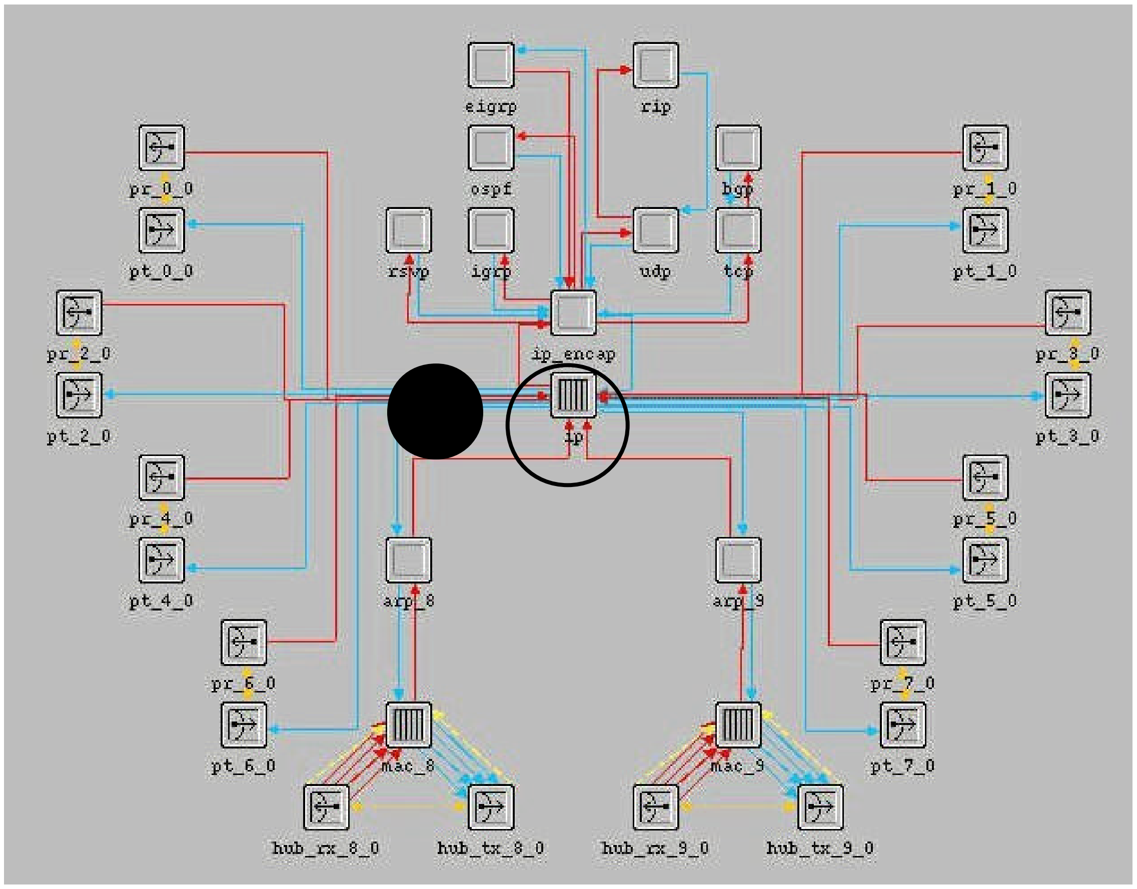


Fig. 3.2 Internal hierarchy structure of the IP router node model.

The *ip_output_iface* process model is in charge of assigning separate queues to various data flows entering the router and scheduling packets out of the queues. The scheduling is performed based on the

VirtualClock algorithm or one of the other scheduling mechanisms currently implemented in OPNET.

3.3.1 State transition diagram

We expanded and modified the state transition diagram of the already existing OPNET process model *ip_output_iface*. Fig. 3.3 shows the original *ip_output_iface* state transition diagram. As mentioned above, this process is in charge of performing other scheduling algorithms like FIFO, Weighted Fair Queuing (WFQ), Priority Queuing (PQ), and Custom Queuing (CQ).

Congestion avoidance mechanisms like Random Early Detection (RED) and Weighted Random Early Detection (WRED) are also handled by this child process. In addition to those, this process is Resource Reservation Protocol (RSVP) aware. It means that the configuration of the queues can be managed upon a RSVP request.

The state transition diagram of the VirtualClock algorithm consists of four states: *init, enqueue, dequeue,* and *idle*, as shown in Fig. 3.4 The init, *enqueue*, and *dequeue* states are all forced states and the *idle* state is an unforced state. These states are incorporated within the state transition diagram of the *ip-output_iface* process model. The state transition diagram functions as follows:

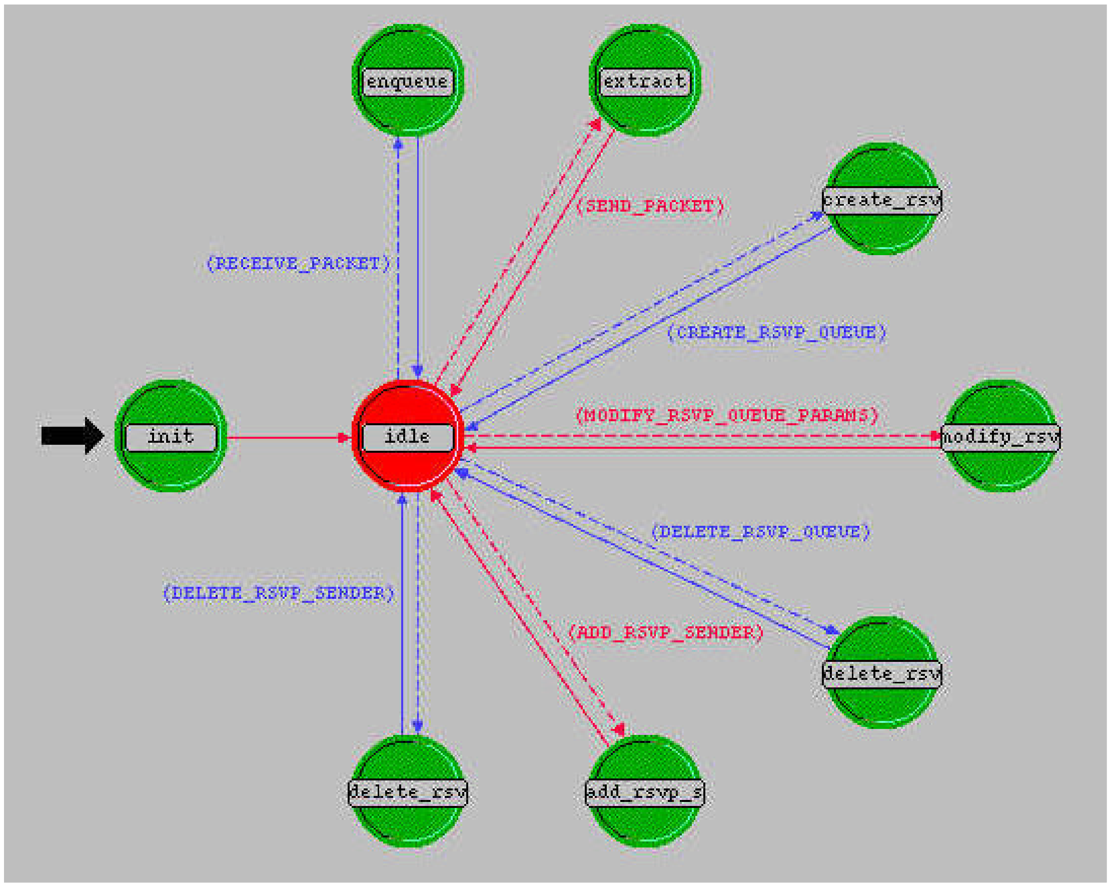


Fig. 3.3 State transition diagram of *ip_output_iface* process model.

When a packet arrives to the ip layer process model, the scheduling process model is invoked. At the invocation time, the process enters the *init* sates in which some initializations for the variables and structures are done. Since the *idle* state has no transition conditions, the process enters the *idle* state immediately after the executives in the *init* state are executed. The *idle* state is an unforced state and the enter executive of the *idle* sates is blank, so after the process has entered this state, it remains idle until it receives an interrupt.

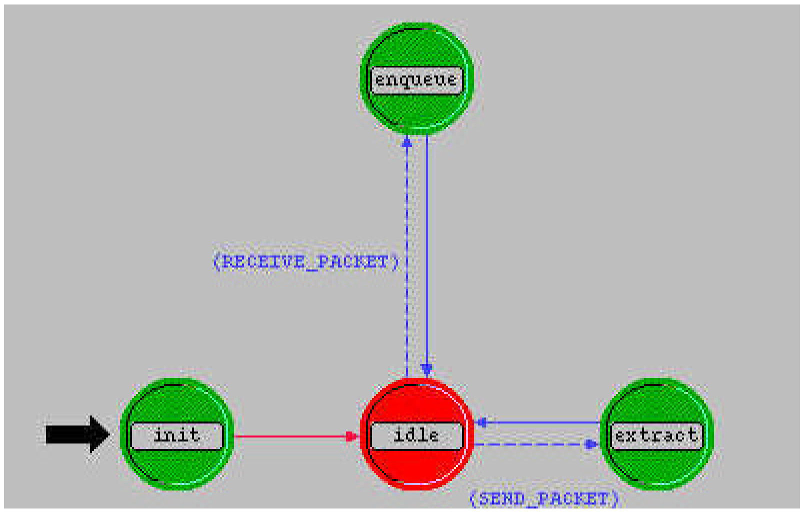


Fig. 3.4 State transition diagram of the VirtualClock process model.

There are two interrupts that cause the process to move from *init* state to either the *enqueu* or *dequeue* states. Depending on the origin of the interrupt one of the outgoing conditions of the idle state is satisfied, which will take the process to the next state. The interrupt may originate from the *ip_rte_v4* process model upon arrival of a packet from the upper layer. This will cause the RECEIVE_PACKET condition to be satisfied and takes the process to the *euqueue* state. In the *enqueue* state the packet is assigned to one of the existing queues in the router's output interfaces according to the classification table that is defined in the *IP QoS configuration* object. The process then returns to the *idle* state. The interrupt might also originate from the process itself when it is time to send a packet to the outgoing link. This interrupt is called a self-interrupt and when it happens the SEND_PACKET condition is satisfied, taking the process to the *dequeue* state. So, the process enters the *dequeue* state when the last packet has just been transmitted to the outgoing link. This state is in charge of choosing the queue from which the next packet has to be transmitted according to the VirualClock algorithm. The details of the VirtualClock implementation are described step by step below as functions done in the *enqueue* and *dequeue* states.

3.3.2 Enqueue state

When a packet arrives from an upper layer process, it enters the *enqueue* state. The function of the *enqueue* state is as follows:

Step 1. Get the incoming packet.

Step 2. Determine the queue to which the packet belongs according to the flow associated with the incoming packets. The flow recognition criteria are: packet source address, destination address, incoming port number, outgoing port number, and required Type of Service (ToS).

Step 3. Check whether the packet is the first packet of its flow. This is performed by setting a boolean flag for each queue. The flag is set to true upon arrival of the very first packet, and is checked every time a packet enters the *enqueue* state. If the packet is the first packet of the flow, VC_i and $auxVC_i$ of the queue corresponding to $flow_i$ are initialized with the real time.

Step 4. Get the AR_i of $flow_i$ and calculate $Vtick_i$ for the packet's queue.

Step 5. Advance VC_i and $auxVC_i$ by $Vtick_i$ and stamp the packet with $auxVC_i$. The stamping is implemented by using OPNET's data type called Interface Control Information (*ICI*). *ICI* contains fields for user-defined parameters to be shared by multiple entities in the network. After advancing the *auxVC* for each packet, the *auxVC* value is saved in the *a_Virtualcl_Clock_Stamp* field in the *ICI* named *ip_arp_req_v4*. Then, the *ICI* is associated with the packet, and remains with it as long as the packet is waiting in the queue. The *a_Virtual_Clock_Stamp* field is accessed in the *dequeue* state in order to choose the packet with the lowest *auxVC* value to be sent to the output interface.

Step 6. If there are no packets waiting in other queues, the packet is sent out immediately. Otherwise, it will remain in its associated queue and the process control returns to the *idle* state.

3.3.3 Dequeue state

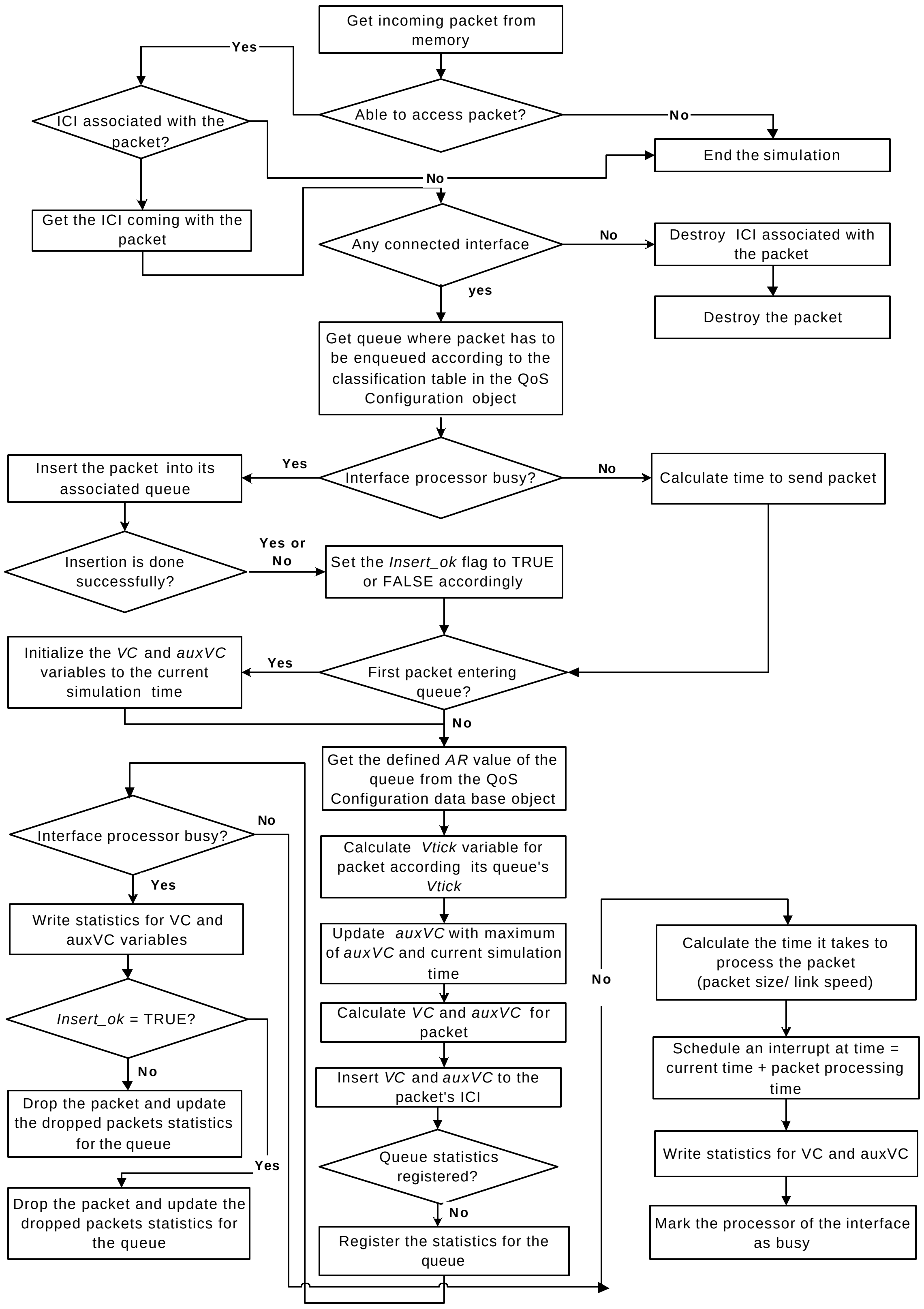
The packet enters the *dequeue* state when it is time for it to be dequeued. The operations conducted in the *dequeue* state are:

Step 1. Send the packet to the network interface.

Step 2. Get the *ICI* named *ip_arp_req_v4* associated with the packet, and read its *a_Virtual_Clock_Stamp* field.

Step 3. If there are no packets in other queues, return to the *idle* state. Otherwise, choose the next packet to be dequeued. In order to select the correct packet to be serviced, check all the queues by looking at the *auxVC* stamp value of the packets located at the head of the queues. The packet with the lowest *auxVC* stamp value is chosen as the next packet to be serviced. In case there is more than one packet with an identical stamp value, the priority is given to the packet from the queue with the lowest index.

Step 4. Schedule the time at which the selected packet should be serviced, and return to the *idle* state. This time is calculated by dividing the packet size (bits) by the link rate (bits/sec).



Get incoming packet from memory
Able to access packet?
Yes
No
ICI associated with the packet?
End the simulation
No
Get the ICI coming with the packet
Any connected interface
No
Destroy ICI associated with the packet
yes
Destroy the packet
Get queue where packet has to be enqueued according to the classification table in the QoS Configuration object
Insert the packet into its associated queue
Yes
Interface processor busy?
No
Calculate time to send packet
Insertion is done successfully?
Yes or No
Set the Insert_ok flag to TRUE or FALSE accordingly
Initialize the VC and auxVC variables to the current simulation time
Yes
First packet entering queue?
No
Get the defined AR value of the queue from the QoS Configuration data base object
Interface processor busy?
No
Yes
Calculate Vtick variable for packet according its queue's Vtick
Write statistics for VC and auxVC variables
Update auxVC with maximum of auxVC and current simulation time
No
Calculate the time it takes to process the packet (packet size/ link speed)
Insert_ok = TRUE?
Calculate VC and auxVC for packet
No
Insert VC and auxVC to the packet's ICI
Schedule an interrupt at time = current time + packet processing time
Drop the packet and update the dropped packets statistics for the queue
Yes
Queue statistics registered?
Write statistics for VC and auxVC
Drop the packet and update the dropped packets statistics for the queue
No
Register the statistics for the queue
Mark the processor of the interface as busy

Fig. 3.5 *Enqueue* state flowchart of VirtualClock state transition diagram.

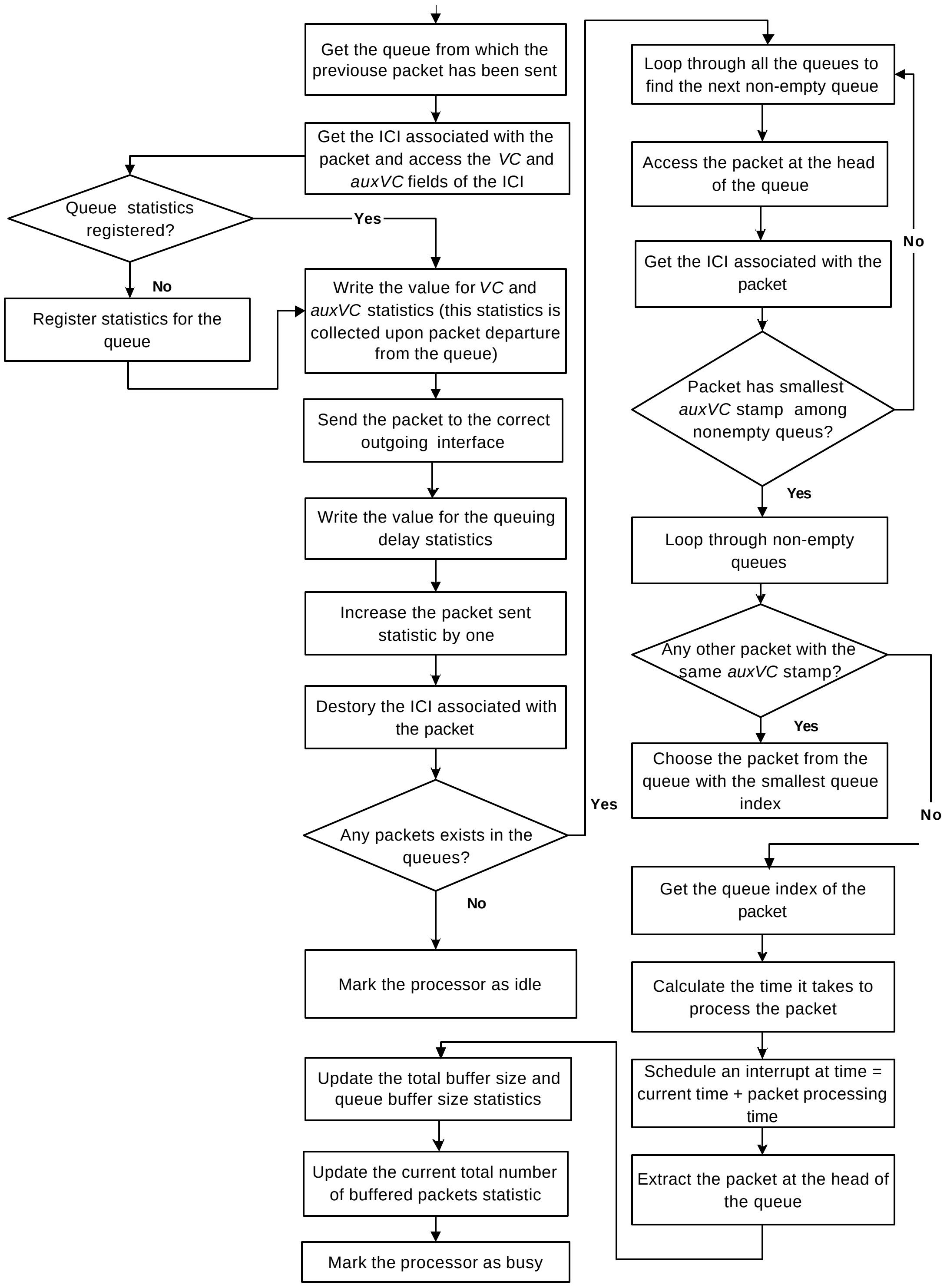

Get the queue from which the previouse packet has been sent
Get the ICI associated with the packet and access the *VC* and *auxVC* fields of the ICI
Queue statistics registered?
Yes
No
Register statistics for the queue
Write the value for *VC* and *auxVC* statistics (this statistics is collected upon packet departure from the queue)
Send the packet to the correct outgoing interface
Write the value for the queuing delay statistics
Increase the packet sent statistic by one
Destory the ICI associated with the packet
Any packets exists in the queues?
Yes
No
Mark the processor as idle
Update the total buffer size and queue buffer size statistics
Update the current total number of buffered packets statistic
Mark the processor as busy
Loop through all the queues to find the next non-empty queue
Access the packet at the head of the queue
Get the ICI associated with the packet
Packet has smallest *auxVC* stamp among nonempty queus?
No
Yes
Loop through non-empty queues
Any other packet with the same *auxVC* stamp?
Yes
No
Choose the packet from the queue with the smallest queue index
Get the queue index of the packet
Calculate the time it takes to process the packet
Schedule an interrupt at time = current time + packet processing time
Extract the packet at the head of the queue

Fig. 3.6 *Dequeue* state flowchart of VirtualClock state transition diagram.

Figs. 3.5 and 3.6 show the flowcharts for the *enqueue* and *dequeue* states of VirtualClock state transition diagram, respectively.

3.4 *IP QoS Configuration* object

IP QoS Configuration is an OPNET *Common Attribute* object that allows the definition of a queuing profile for any of the QoS schemes used by IP nodes in OPNET. The advantage of using this global object is that once you define the scheme you can simply refer to the scheme on individual IP objects [23].

3.4.1 Definition of the VC Profiles structured attribute

We added a new attribute (*VC Profiles*) for the VirtualClock algorithm in *qos_attribute_definer* process model. This process model is used to define the functionality of OPNET's *IP QoS Configuration* object node model.

By assigning values to the sub-attributes of the *VC Profiles* attribute, users will be able to define their own VirtualClock queuing profiles for the outgoing interface. This attribute has three sub-attributes: *Profile Name, Buffer Capacity,* and *Queues Configuration.* These attributes define the queuing parameters of each interface.

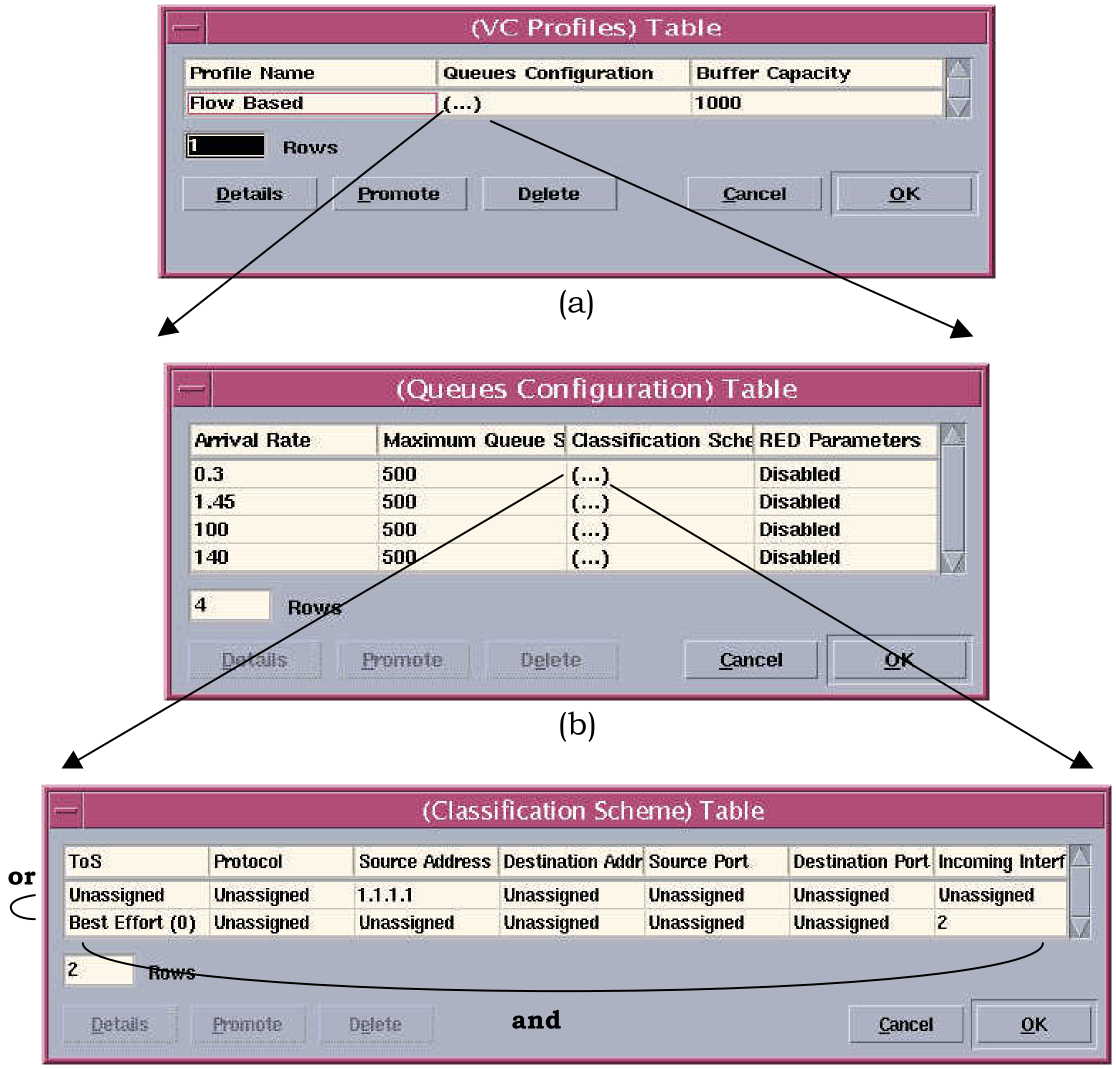
Profile Name is the name of the queuing management profile. The queuing attributes of each interface are identified by a profile name. *Buffer Capacity* shows the size of buffer for each interface. When the value for this sub-attribute is reached, the interface enters the state of congestion; otherwise the queues can still store packets.

Queues Configuration sub-attribute enables the users to define an optional number of queues on each interface. This sub-attribute, which is a compound attribute itself, is shown as a table with each row representing a separate queue. The following attributes apply to each queue: *Arrival Rate, Classification Scheme, and Maximum Queue Size.*

Arrival rate is the expected packet arrival rate of the flow entering the queue. *Maximum Queue Size* is the maximum allowable number of packets per queue. This is used when the interface is congested. It means that when the total number of buffered packets in all the queues is reached if the total number of packets in each queue exceeds this value, packets will be dropped from that queue.

The *Classification Scheme* sub-attribute defines the criteria for the packet in order to enqueue it. There are six distinct criteria: ToS, protocol, source IP address, destination IP address, source port number, destination port number, and the incoming interface from which the packet entered the router.

Fig. 3.7 shows the structure and the hierarchy of the *VC Profiles* attribute and its sub-attributes through an example. As an example we have created a profile for VirtualClock algorithm called *Flow Based* shown in Fig. 3.7 (a). Fig. 3.7 (b) shows that this profile has four queues with different *Arrival Rates* and the same *Maximum Queue Size* of 500 (packets). The classification criteria for these queues are represented in Fig. 3.7. (c). This figure shows that the packets originated from source with an IP address of 1.1.1.1, or the packets that have entered the router from Interface = 0 and have the ToS = Best Effort (0) are all allocated to Q0.



Profile Name	Queues Configuration	Buffer Capacity
Flow Based	(...)	1000

1 Rows

Arrival Rate	Maximum Queue S	Classification Sche	RED Parameters
0.3	500	(...)	Disabled
1.45	500	(...)	Disabled
100	500	(...)	Disabled
140	500	(...)	Disabled

4 Rows

ToS	Protocol	Source Address	Destination Addr	Source Port	Destination Port	Incoming Interf
Unassigned	Unassigned	1.1.1.1	Unassigned	Unassigned	Unassigned	Unassigned
Best Effort (0)	Unassigned	Unassigned	Unassigned	Unassigned	Unassigned	2

2 Rows

(c)

Fig. 3.7 (a) *VC Profiles*, (b) *Queues Configuration,* and (c) *Classification Schemes* tables of the *IP QoS Configuration* object.

3.4.2 Introducing the VC Profiles to the global database

In order to make the *VC Profiles* structured attribute and its sub-attributes recognizable to the network-wide database, we have implemented a function called *attr_def_VC_profiles_info_parse( )*. This function is implemented in the *qos_attribute_definer* process model, where we previously defined the *VC Profiles* attribute and its sub-attributes. The function parses the information defined in the compound attribute *VC Profiles*. It also creates instances for the data structure *QM Information*, allocates memory to these instances and stores the parsed information in them. The *QM Information* structure contains general queuing information for each interface. As mentioned before, the queuing attributes of each interface is identified by a profile name. Each profile contains many queues and each queue can have a list of queue classification criteria. The *QM Information* structure contains the following fields:

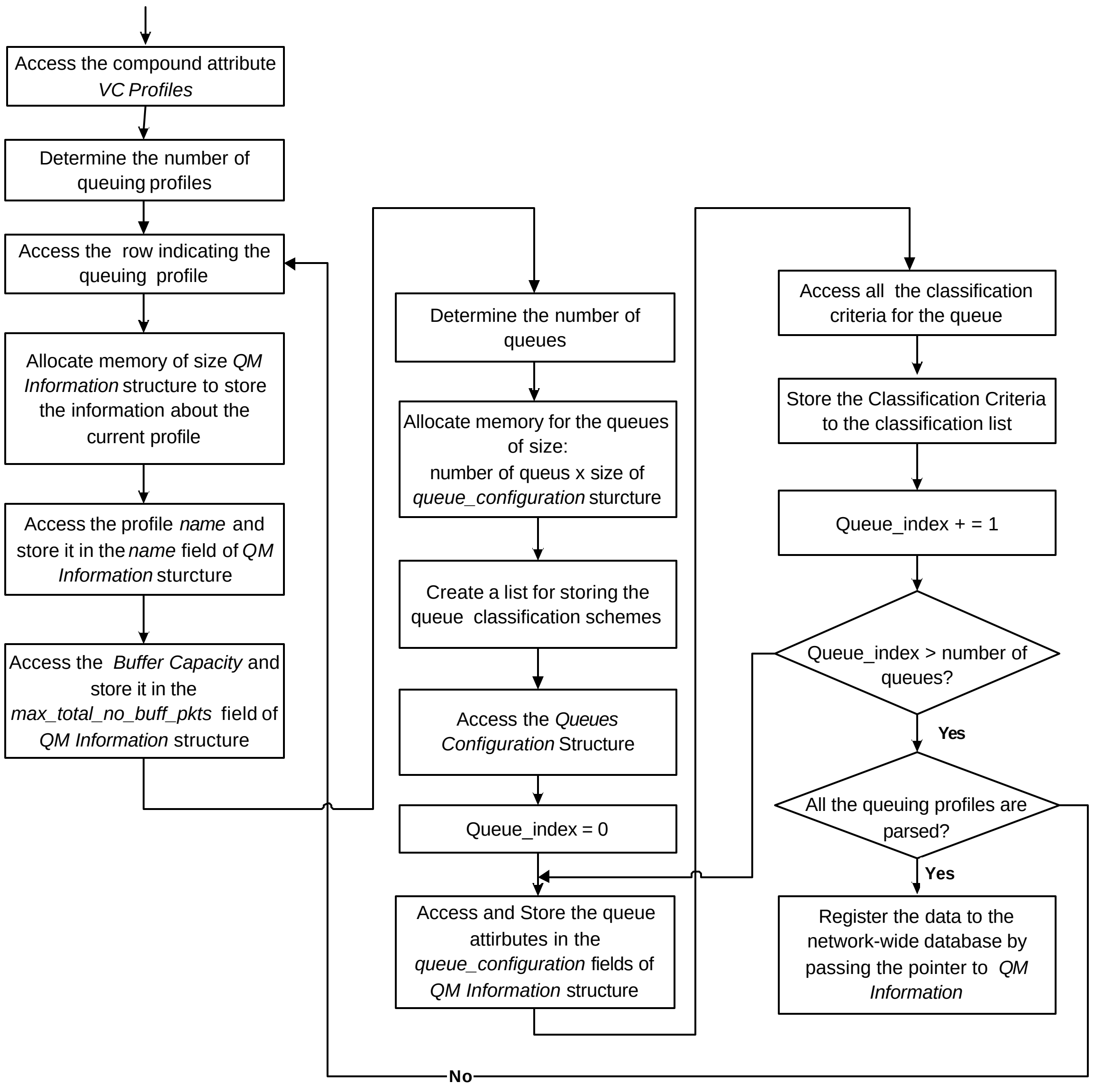
- *name:* name of the queuing profile interface
- *no_queues:* number of queues in the interface
- *max_total_no_buff_pkts*: maximum number of buffered packets
- *queue_configuration*: which is a structure itself and stores the queues' parameters
- *classification_list_ptr*: a pointer to the structure that stores criteria to classify packets.

The information parsed from the *VC Profiles* table is stored in the first three fields of the *QM Information* Structure. Parsed information from the Queues Configuration and Classification Scheme tables are stored in *queue_configuration* and *classification_list_ptr* fields of the structure, respectively. After storing the parsed information into the allocated memory, the *attr_def_VC_profiles_info_parse( )* function registers the information into the attribute database in order to provide access from network objects to the information. Flow chart shown in Fig. 3.8 represents the step-by-step operations of this function.

3.4.3 Configuration of the IP objects

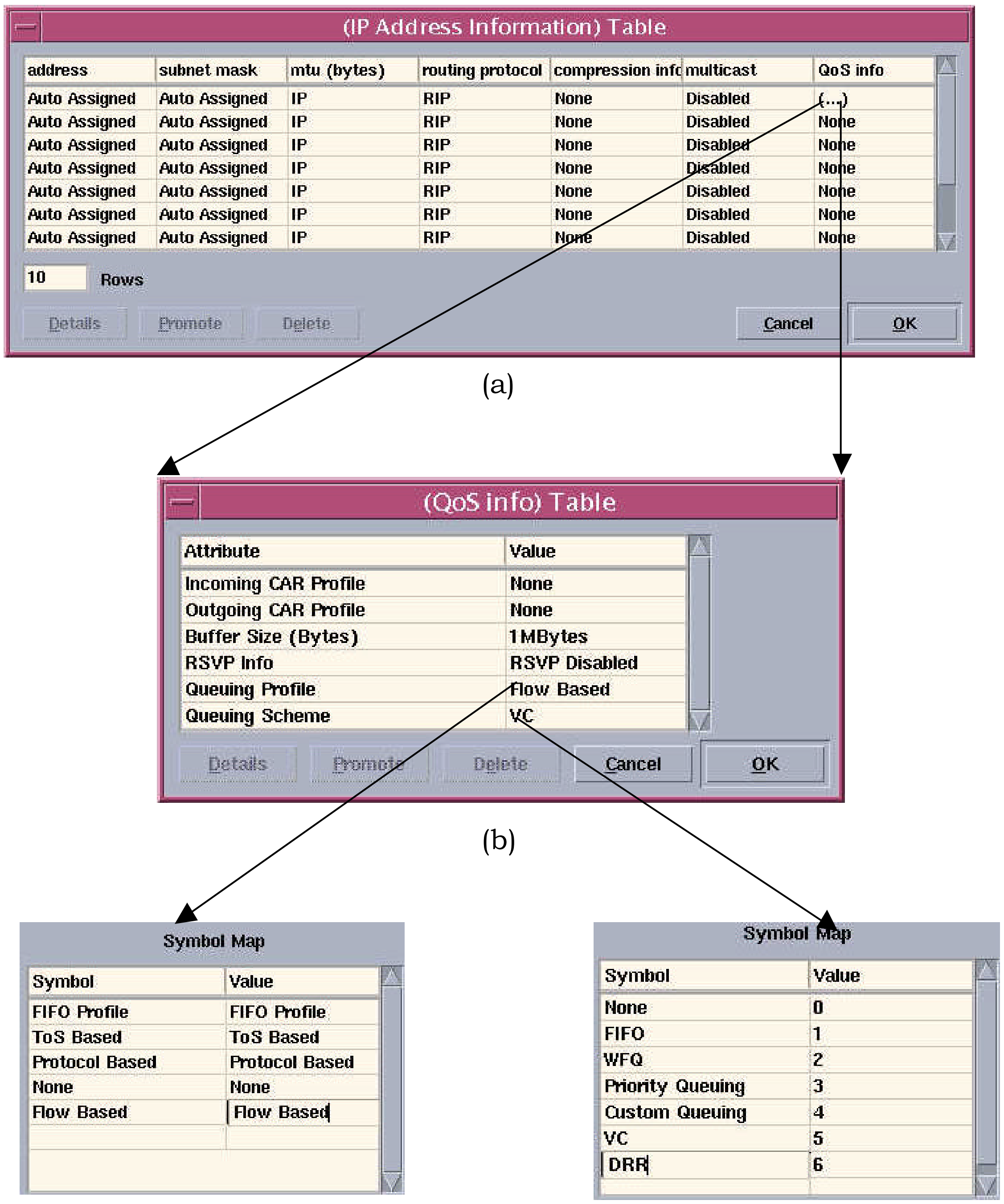
After defining the common attributes in the global database, we should create an interface to these attributes in all the OPNET IP objects that desire to use the VirtualClock as a scheduling technique on their output interfaces. Thus, we have defined the attribute values shown in Fig. 3.9 in the IP layer process model of the IP objects. These attribute values are defined for two sub-attributes of the *IP Address Information* compound attribute of *the ip_rte_v4* process model. The *IP Address Information* attribute allows the configuration of the object's output interfaces.

Fig. 3.9 (a) depicts the *Ip Address Information* compound attribute. Each row in the table is related to one interface and each column in the table defines the attributes of the interface. As revealed in Fig. 3.9 (b) the *QoS info* sub-attribute of each interface shows the QoS related parameters such and management scheme used for that certain interface. This sub-attribute includes: scheduling mechanism (FIFO, WFQ, Priority Queuing, Custom Queuing, and **VirtualClock**), queuing profile, queue management mechanism (RED, WRED), bandwidth management mechanism (Committed Access Rate), and if RSVP is supported on the interface, the attributes of RSVP. For the queuing profile filed, one of the profiles that have previously been defined for each queuing scheme in the *QoS Configuration* Object can be chosen.



Access the compound attribute *VC Profiles*
Determine the number of queuing profiles
Access the row indicating the queuing profile
Allocate memory of size *QM Information* structure to store the information about the current profile
Access the profile *name* and store it in the *name* field of *QM Information* sturcture
Access the *Buffer Capacity* and store it in the *max_total_no_buff_pkts* field of *QM Information* structure
Determine the number of queues
Allocate memory for the queues of size: number of queus x size of *queue_configuration* sturcture
Create a list for storing the queue classification schemes
Access the *Queues Configuration* Structure
Queue_index = 0
Access and Store the queue attirbutes in the *queue_configuration* fields of *QM Information* structure
Access all the classification criteria for the queue
Store the Classification Criteria to the classification list
Queue_index + = 1
Queue_index > number of queues?
Yes
All the queuing profiles are parsed?
Yes
Register the data to the network-wide database by passing the pointer to *QM Information*
No

Fig. 3.8 Flowchart of the *attr_def_VC_profiles_info_parse( )* function.



address	subnet mask	mtu (bytes)	routing protocol	compression info	multicast	QoS info
Auto Assigned	Auto Assigned	IP	RIP	None	Disabled	(...)
Auto Assigned	Auto Assigned	IP	RIP	None	Disabled	None
Auto Assigned	Auto Assigned	IP	RIP	None	Disabled	None
Auto Assigned	Auto Assigned	IP	RIP	None	Disabled	None
Auto Assigned	Auto Assigned	IP	RIP	None	Disabled	None
Auto Assigned	Auto Assigned	IP	RIP	None	Disabled	None
Auto Assigned	Auto Assigned	IP	RIP	None	Disabled	None

Attribute	Value
Incoming CAR Profile	None
Outgoing CAR Profile	None
Buffer Size (Bytes)	1MBytes
RSVP Info	RSVP Disabled
Queuing Profile	Flow Based
Queuing Scheme	VC

Symbol	Value
FIFO Profile	FIFO Profile
ToS Based	ToS Based
Protocol Based	Protocol Based
None	None
Flow Based	Flow Based

Symbol	Value
None	0
FIFO	1
WFQ	2
Priority Queuing	3
Custom Queuing	4
VC	5
DRR	6

(c) (d)

Fig. 3.9 (a) IP *Address Information*, (b) *QoS Information,* (c) *Queuing Profile*, and (d) *Queuing Scheme* tables of the IP router objects.

As illustrated in Fig. 3.9 (d), We have added *VC* as a possible value for the *Queuing Scheme* sub-attribute and also *Flow Based* as a sample *VC Profile* value for the queuing profile sub-attribute, shown in Fig. 3.9 (c).

Chapter 4
VirtualClock Model Verification

In this Chapter we will show the correctness and the functionality of the implemented VirtualClock model by running simulations. For this purpose, we use OPNET software tool version 7.0.L. We introduce two scenarios: a simple scenario in which there are two sources creating conforming traffic with a constant packet generation rate. In this scenario, we keep track of the packets' arrival and departure times into and from the *ip_output_iface* process model. We also examine the packets' *Virtual Clock* and *aux Virtual Clock* stamps to see how the VirtualClock algorithm selects and forwards packets according to the stamps. In the second scenario, which is a more complex one, we examine the functionality of the VirtualClock algorithm during conforming and nonconforming periods of time.

4.1 Model verification

In order to evaluate the performance of the VirtualClock process model, we have created the simple network model in the OPNET's network editor. The network model is shown in Fig. 4.1.

This is an ideal scenario in which both sources are generating packets according to their specified packet rates. The Ethernet network consists of two clients sending traffic to associated servers via switches and routers. Clients 1 and 2 generate 1,024 byte IP packets at a constant rate of 10 and 5 packets/sec, respectively.

All the nodes in the network are connected with 10BaseT links with a 10 Mbps data rate. The only link in the network that has a lower capacity is the link between Routers 1 and 2, chosen to be a DS0 link with a 64 Kbps data rate. The bottleneck link is positioned immediately before the output interface of Router 1 where the VirtualClock scheduling algorithm is implemented. Hence, we can observe the order in which the packets are dequeued by the VirtualClock algorithm.

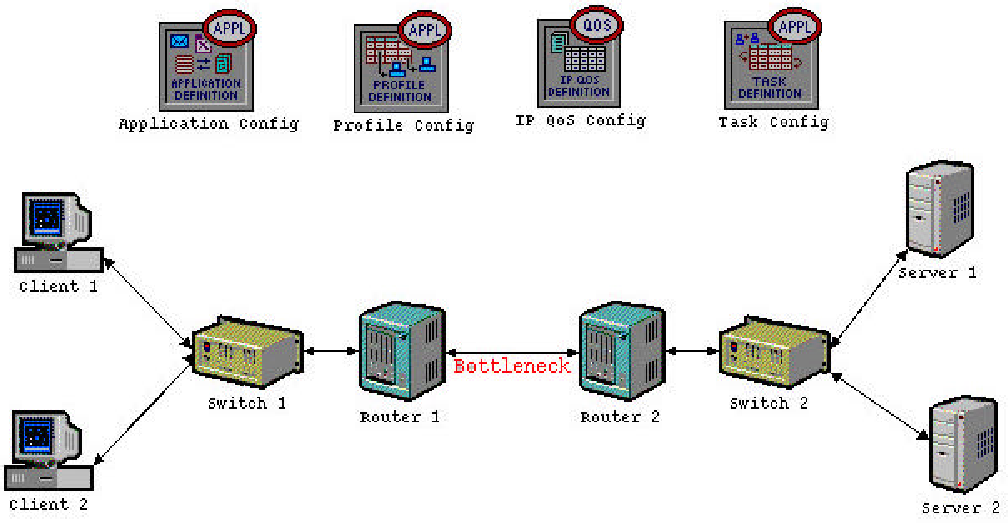


Figure 4.1 Network model for performance verification of the VirtualClock algorithm.

Incoming packets from Clients 1 and 2 are destined for Servers 1 and 2, respectively. The packets are sorted into two distinct queues and ordered out of the queues according to their specified packet rates.

In the *VC Profile* of the *IP QoS Configuration* object, we defined a new queuing profile named *Flow Based*. This profile has two rows. Each row represents a queue with the following parameter settings:

- *Arrival Rate*$_0$ = *10, Queue Size*$_0$ = *500*

- *Arrival Rate$_1$ = 5, Queue Size$_1$ = 500.*

The following figures are simulation results, obtained from OPNET, to show the validity of the algorithm. Incoming packets from Client 1 are recognized as a flow and are assigned to the first queue (Q0).

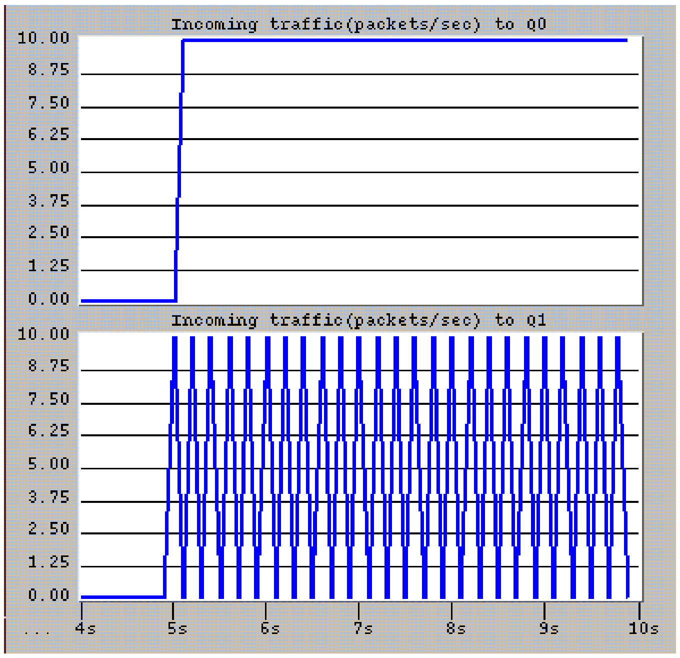


Figure 4.2 Incoming traffic to queues: Q0 (top) and Q1 (bottom), in (packets/sec) vs. time.

Packets from the second flow, coming from Client 2, are assigned to the second queue (Q1). The incoming traffic to queues Q0 and Q1 is shown in Fig. 4.2.

Fig. 4.3 illustrates the *VC* stamp values of the packets in queues Q0 and Q1, respectively. The stamp values are calculated upon packet arrivals to the *enqueue* state of the scheduling process. As expected, when a packet arrives to a particular queue, the queue's *VC* increases by *1/AR* of the particular queue (0.1 for Q0, and 0.2 for Q1), and the arriving packet is stamped with the value *VC*.

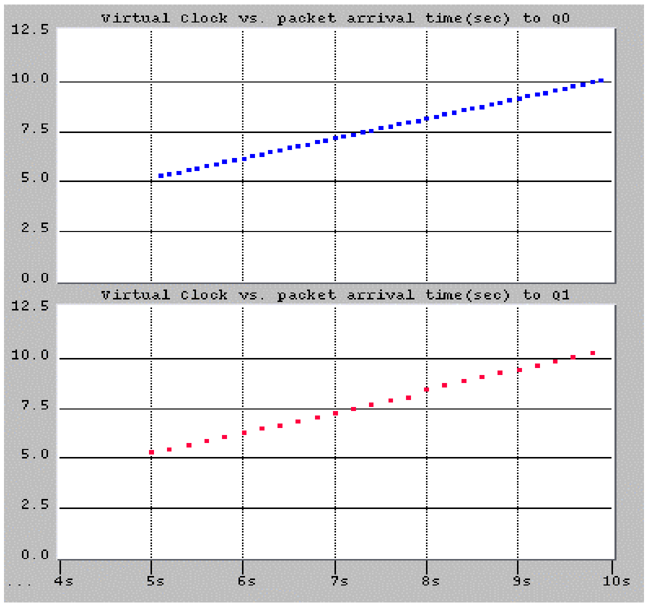


Fig. 4.3 Virtual Clock (*VC*) stamp vs. packet arrival time (sec) for queues: Q0 (top) and Q1 (bottom).

Fig. 4.4 shows the auxiliary Virtual Clock (*auxVC*) stamp values of the packets in queues Q0 and Q1 that are sent to the outgoing interface. We expect the VirtualClock algorithm to be a fair algorithm that assigns the bandwidth fairly to the flows according to their negotiated packet generation rates. Because the specified arrival rate of the first flow is twice that of the second flow, we expect that the scheduling algorithm will send two packets from Q0 for each packet sent from Q1, as illustrated in Figure 4.5.

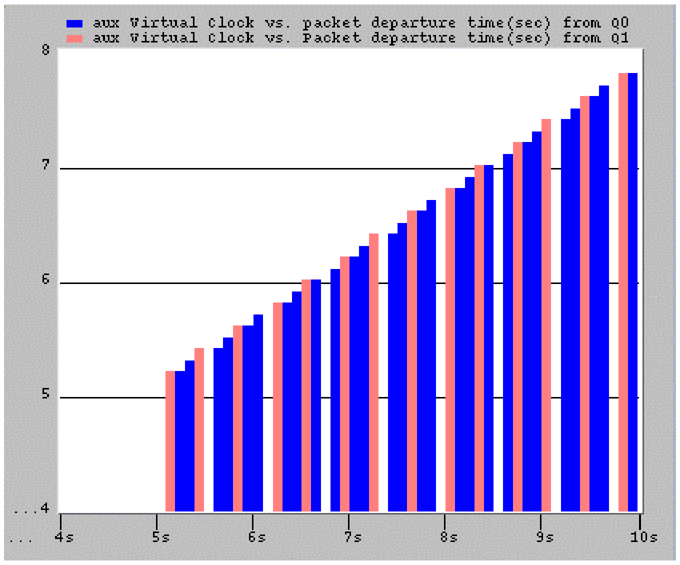


Fig. 4.4 Auxiliary Virtual Clock (*auxVC*) vs. packet departure time (sec) for queues Q0 (dark) and Q1 (light).

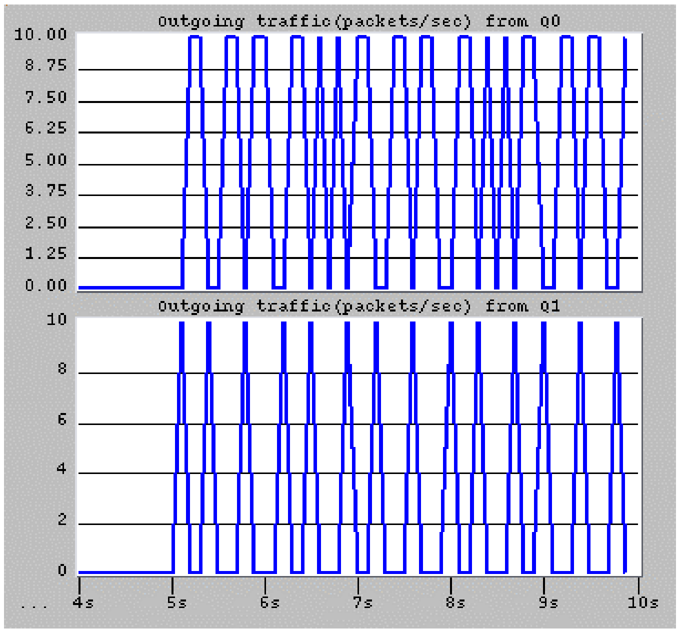


Fig. 4.5 Outgoing traffic from queues: Q0 (top) and Q1 (bottom), in (packets/sec) vs. time (sec).

Time	**Action**	**State**	***auxVC***	**Queue**
5	Q1 packet arrives and scheduled for t = 5.12	enqueue	5.2	Q1
5.1	Q0 packet arrives and queued	enqueue	5.2	Q0
5.12	Q1 packet is sent	dequeue	5.2	Q1
5.12	Q0 packet is selected and scheduled for t = 5.24	dequeue	5.2	Q0
5.2	Q0 packet arrives and queued	enqueue	5.3	Q0
5.2	Q1 packet arrives and queued	enqueue	5.2	Q1
5.24	Q0 packet is sent	dequeue	5.2	Q0
5.24	Q0 packet is selected and scheduled for t = 5.36	dequeue	5.3	Q0
5.3	Q0 packet arrives and queued	enqueue	5.4	Q0
5.36	Q0 packet is sent	dequeue	5.3	Q0
5.36	Q1 packet is selected and scheduled for t = 5.48	dequeue	5.4	Q1
5.4	Q0 packet arrives and queued 5.48	enqueue	5.5	Q0
5.4	Q1 packet arrives and queued	enqueue	5.6	Q1
5.48	Q1 packet is sent	dequeue	5.4	Q1
5.48	Q0 packet arrives and scheduled for t = 5.6	enqueue	5.4	Q0
5.5	Q0 packet arrives and queued	enqueue	5.6	Q0
5.6	Q0 packet arrives and queued	enqueue	5.7	Q0
5.6	Q1 packet arrives and queued	enqueue	5.8	Q1
5.6	Q0 packet is sent	dequeue	5.4	Q0
5.6	Q0 packet is selected and scheduled at t = 5.72	dequeue	5.5	Q0
5.7	Q0 packet arrives and queued	enqueue	5.8	Q0
5.72	Q0 packet is sent	dequeue	5.5	Q0
5.72	Q0 packet is selected and scheduled at t = 5.8	dequeue	5.6	Q0
5.8	Q0 packet arrives and queued	enqueue	5.9	Q0

Table 4.1 Timetable for state transition diagram of VirtualClock.

Pkt #	**Pkt arrival time**	**Expected *VC* upon pkt arrival**	***VC* upon pkt arrival**	**Expected *auxVC* upon pkt arrival**	**Pkt departure time**	**Expected *auxVC* upon pkt departure**
1	5.1	5.0 + 0.1 = 5.2	5.2	5.0 + 0.1 = 5.2 5.2 >5.1 (real time)	5.2	5.2
2	5.2	5.2+ 0.1 = 5.3	5.3	5.2 + 0.1 = 5.3 5.3 > 5.2 (real time)	5.4	5.3
3	5.3	5.3 + 0.1 = 5.4	5.4	5.3 + 0.1 = 5.4 5.4 > 5.3 (real time)	5.5	5.4
4	5.4	5.4 + 0.1 = 5.5	5.5	5.4 + 0.1 = 5.5 5.5 > 5.4 (real time)	5.6	5.5
5	5.5	5.5 + 0.1 = 5.6	5.6	5.5 + 0.1 = 5.6 5.6 > 5.5 (real time)	5.6	5.6
6	5.6	5.6 + 0.1 = 5.7	5.7	5.6 + 0.1 = 5.7 5.7 > 5.6 (real time)	5.8	5.7
7	5.7	5.7 + 0.1 = 5.8	5.8	5.7 + 0.1 = 5.8 5.8 > 5.7 (real time)	5.9	5.8
8	5.8	5.8 + 0.1 = 5.9	5.9	5.8 + 0.1 = 5.9 5.9 > 5.8 (real time)	6.1	5.9
9	5.9	5.9 + 0.1 = 6.0	6	5.9 + 0.1 = 6.0 6.0> 5.9 (real time)	6.2	6.0
10	6.0	6.0 + 0.1 = 6.1	6.1	6.0 + 0.1 = 6.1 6.1> 6.0 (real time)	6.4	6.1

Table 4.2 Verification table for Q0, *AR* = 10 (packets/sec).

Pkt #	**Pkt arrival time**	**Expected *VC* upon pkt arrival**	***VC* upon pkt arrival**	**Expected *auxVC* upon pkt arrival**	**Pkt departure time**	**Expected *auxVC* upon pkt departure**
1	5.0	5.0 + 0.2 = 5.2	5.2	5.0 + 0.2= 5.2 5.2 >5.0 (real time)	5.1	5.2
2	5.2	5.2+ 0.2 = 5.4	5.4	5.2 + 0.2= 5.4 5.4> 5.2 (real time)	5.4	5.4
3	5.4	5.4 + 0.2 = 5.6	5.6	5.4 + 0.1 = 5.6 5.6> 5.4 (real time)	5.8	5.6
4	5.6	5.6 + 0.2 = 5.8	5.8	5.6 + 0.2 = 5.8 5.8 > 5.6 (real time)	6.2	5.8
5	5.8	5.8 + 0.2 = 6.0	6.0	5.8 + 0.2 = 6.2 6.0 > 5.8 (real time)	6.5	6.0
6	6.0	6.0 + 0.2 = 6.2	6.2	6.0 + 0.2 = 6.2 6.2 > 6.0 (real time)	6.9	6.2
7	6.2	6.2 + 0.2 = 6.4	6.4	6.2 + 0.2 = 6.4 6.4 > 6.2 (real time)	7.2	6.4
8	6.4	6.4 + 0.2 = 6.6	5.6	6.4 + 0.2 = 6.6 6.6 > 6.4 (real time)	7.6	6.6
9	6.6	6.6 + 0.2 = 6.8	6.8	6.6 + 0.2 = 6.8 6.8 > 6.6 (real time)	8.0	6.8
10	6.8	6.8 + 0.2 = 7	7.0	6.8 + 0.2 = 7.0 7.0 > 6.8 (real time)	8.3	7.0

Table 4.3 Verification table for Q1, *AR* = 5 (packets/sec).

Tables 4.1 shows the timetable of VirtualClock state transition diagram. Tables 4.2 and 4.3 are used to verify the values of *VC* and *auxVC* variables for queues:Q0 and Q1.

4.2 Functionality test

We use OPNET simulation tool to evaluate the performance of the VirtualClock algorithm in the simulation scenario similar to the scenario described in Fig. 4.1. The VirtualClock scheduler model is used inside the IP router node.

In our scenario, we consider two conforming and one nonconforming sources. The first conforming source has constant packet generation rate. The third source which is also conforming, generates traffic with self-similar characteristics, which fluctuates around its specified average packet arrival rate. The nonconforming source generates packets with a constant rate. For a short period of time, it conforms to its negotiated packet generation rate. After this short period, the source decreases its rate for a while in order to gather credits for sending a burst of packets at a rate that is four times its expected rate. This behavior for the nonconforming source is chosen so that we could examine the reaction of the VirtualClock algorithm in situations when a misbehaving source wants to use the credits gained for sending a burst of traffic.

The network topology is similar to the topology shown in Fig. 4.1, with an additional source and destination. In this scenario, we use three Ethernet clients to send traffic to three Ethernet servers. All the nodes are connected with 10BaseT links, except the DS0 link between Routers 1 and 2. The specified packet arrival rates of Clients 1, 2, and 3 are 4, 2, and 2 packets/sec, respectively. These rates are assigned in the *VC Profile* of the *IP QoS Configuration* object. In the *VC Profile*, we defined a queuing profile *Flow Based1* that contains three rows. Each row represents a queue with the following parameter settings:

- *Arrival Rate*$_0$ *= 4, Queue Size*$_0$ *= 500*
- *Arrival Rate*$_1$ *= 2, Queue Size*$_1$ *= 500*
- *Arrival Rate*$_2$ *= 2, Queue Size*$_2$ *= 500.*

Client 1 starts sending packets at time 20 sec. Using an exponential process, It generates IP packets of size 1,024 bytes at a constant rate of 4 packets/sec, and stops at 555 sec. Client 2 also generates 1,024 bytes IP packets. It begins generating traffic at a constant rate of 2 packets/sec at 20 sec. At time = 142.5 sec, it reduces the traffic rate and keeps on sending packets at a rate of 0.5 packets/sec for 250 sec. At 392.5 sec, it increases its rate and continues sending packets at a rate of 8 packets/sec until time 455 sec. Client 3 is an OPNET *ethernet_rpg_wkstn_adv* source node model. This source is chosen from OPNET's Raw Packet Generator (RPG) model [24]. RPG is a traffic source model that is used to generate self-similar traffic [30].

The reason for choosing a self-similar traffic generator in our scenario is that we wanted to make our network congested with a traffic that behaves like today's genuine networks traffic. Measurements conducted on Bellcore Ethernet traffic first indicated the self-similar characteristic of the traffic. The measurements showed that traffic pattern seemed similar over the large time scales (hours and minutes) and small time scales (seconds and milliseconds) [15, 19]. Later on, the self-similar behavior has been observed in the traffic of a number of other network applications like HTTP traffic, video traffic, and Motion Pictures Experts

Group (MPEG) traffic [15]. The self-similar traffic model is used to capture the fractal properties of Internet traffic.

Client 3 starts at 5 sec and sends traffic with the following specifications:

- *Average arrival rate* = 2 (packets/sec)
- *Hurst parameter* = 0.9
- *Fractal onset time scale* = 0.1.

Fig. 4.6 shows the incoming traffic from Clients 1, 2, and 3 to Router 1's queues Q0, Q1, and Q2, respectively. Fig. 4.7 depicts the Virtual Clock (*VC*) stamp values of the packets in queues Q0, Q1, and Q2. The slope of the (*VC*) graph for Qi is calculated as: $\text{slope}_i = Vtick_i \,/\, (1/\, AR_i)$.

If a flow is conforming to its expected packet generation rate, the values of *Vtick* and $1/AR$ are identical. Hence, its *VC* slope is equal to 1. As it can be seen in Fig. 4.7 (top), flow_0 adheres to its specified packet arrival rate. Thus, the slope of the Virtual Clock graph is 1. Fig. 4.7 (middle) indicates that the slope of the graph changes at instances when packet arrival rate of the flow changes. For the periods during which traffic has a lower arrival rate (142.5 sec - 392.5 sec), Virtual Clock line has slope < 1. For higher arrival rate periods (392.5 sec - 455 sec), we observe a steeper line with slope > 1. Fig. 4.7 (bottom) shows that, for a flow fluctuating around its specified arrival rate, the Virtual Clock graph is close to a line with slope 1.

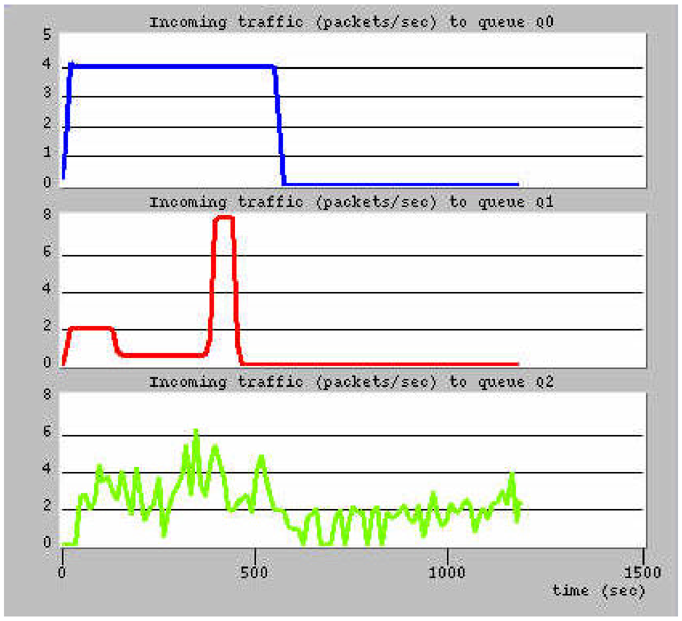


Fig. 4.6 Incoming traffic to queues: Q0 (top), Q1 (middle), and Q2 (bottom) in (packets/sec) vs. time.

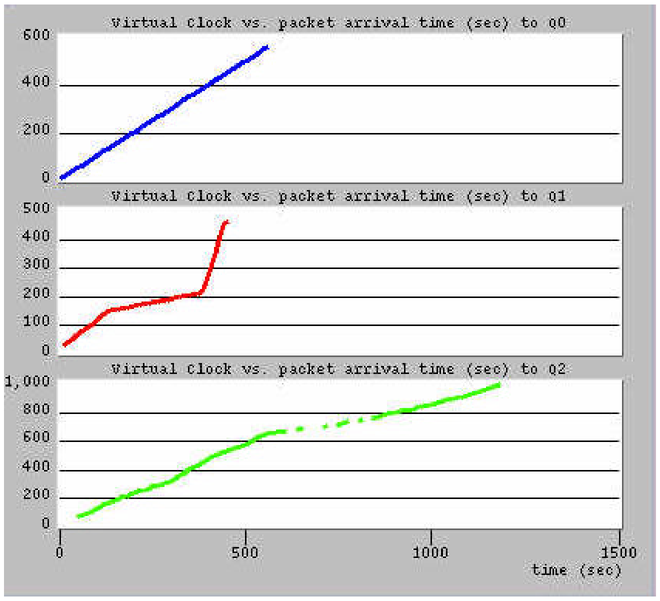


Fig. 4.7 Virtual Clock (*VC*) stamps vs. packet arrival time (sec) for queues: Q0 (top), Q1 (middle), and Q2 (bottom).

Fig. 4.8 displays the auxiliary Virtual Clock stamp values (*auxVC*) of packets in queues Q0, Q1, and Q2. The role of this variable is to prevent the flows from gathering credits by not sending traffic for a period of time and then suddenly sending a burst of traffic. This is achieved by upgrading the auxiliary Virtual Clock value to the larger value between *auxVC* and real time.

Figs. 4.7 and 4.8, show that the Virtual Clock and auxiliary Virtual Clock graphs of packets coming to Q0 (top) and Q2 (bottom) have the same slope, because the flows have been conforming to their expected sending rate. In contrast, the traffic to Q1 reduces its rate at 142.5 sec. This is shown as the difference between the slopes of the Virtual Clock and auxiliary Virtual Clock in Figs. 4.7 and 4.8 (bottom) from 142.5 sec to 392 sec.

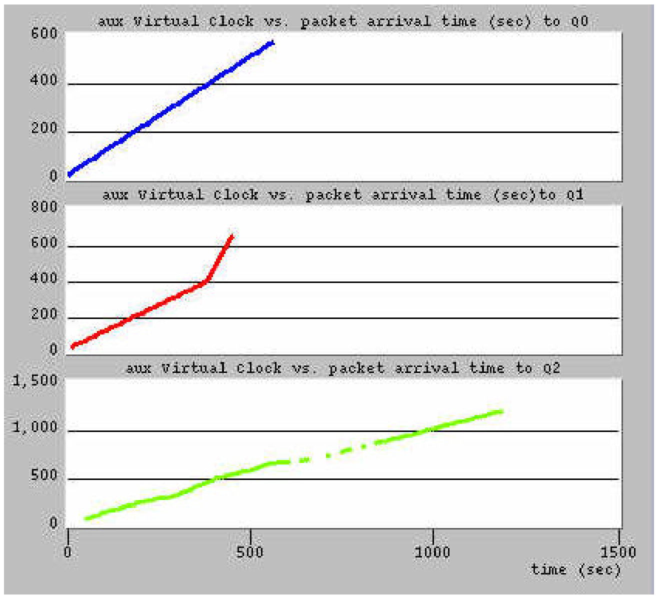


Fig. 4.8 Auxiliary Virtual Clock (*auxVC*) stamps vs. packet arrival time (sec) for queues: Q0 (top), Q1 (middle), and Q2 (bottom).

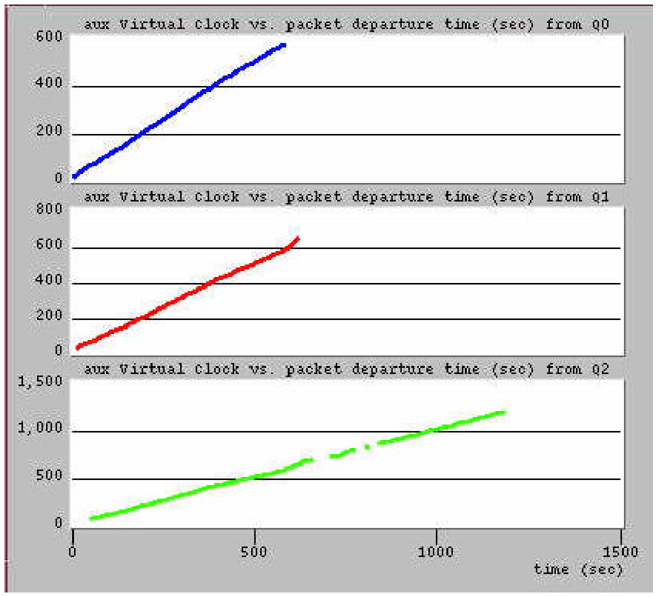


Fig. 4.9 Auxiliary Virtual Clock (*auxVC*) stamps vs. packet departure time (sec) for queues: Q0 (top), Q1 (middle), and Q2 (bottom).

Fig. 4.9 shows the auxiliary Virtual Clock (*auxVC*) stamp values of the packets in queues Q0, Q1, and Q2 that are being sent to the outgoing interface.

Fig. 4.10 shows the outgoing traffic from queues Q0, Q1, and Q2 of Router 1, after being scheduled by the VirtualClock algorithm. In case the total link traffic does not exceed the link's capacity, the algorithm allocates bandwidth to flows according to their specified packet arrival rates. If there is no available bandwidth, the traffic from a flow that is exceeding its specified arrival rate will be queued and serviced with the flow's specified arrival rate. The packets are dropped when the queue becomes full.

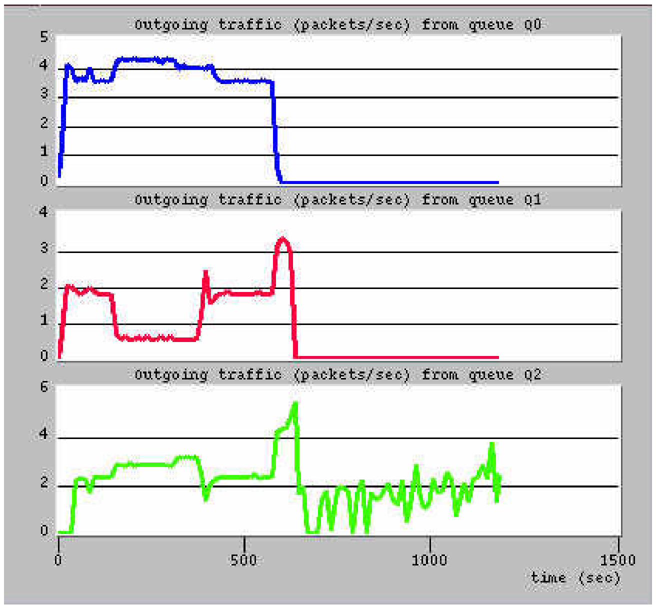


Fig. 4.10 Outgoing traffic from queues: Q0 (top), Q1 (middle), Q2 (bottom) in (packets/sec) vs. time (sec).

It can be seen from Figs. 4.6 and 4.10 (top) that since Client 1 is sending traffic according to its specified arrival rate of 4 packets/sec, the queue Q0 is served with the same rate.

As seen from Figs. 4.6 and 4.10 (middle), traffic from Client 2 is serviced with its arrival rate while it conforms to its specified packet rate (2 packets/sec until 142.5 sec and 0.5 packets/sec from 142.5 sec to 392.5 sec). When the client starts sending packets with arrival rate of 8 packets/sec, which is four times the expected rate, the VirtualClock schedules packets with the flow's expected packet rate (2 packets/sec). The total bandwidth is used by the three sources until 555 sec, when Client 1 stops sending traffic. At that time, part of the bandwidth will be freed and the traffic from Q1 will be serviced at a higher rate.

As seen in Figs. 4.6 and 4.10 (bottom), Client 3 generates self-similar traffic. The Virtual Clock algorithm forwards packets at a rate of 2 packets/sec, which is the average traffic generation rate of the source.

Chapter 5
Simulation Results

In this section we describe a series of simulation experiments. These simulation experiments are conducted in order to compare the functionality of the VirtualClock scheduling algorithm with several other scheduling mechanisms, such as First In First Out (FIFO), Weighted Fair Queuing (WFQ), Custom Queuing (CQ), and Priority Queuing (PQ). The selected mechanisms are commonly used as congestion management techniques in today's IP routers. These routers are mainly produced by manufacturers like Cisco, Inc. and Juniper Networks, which are the leaders in network backbone router manufacturing. WFQ is Cisco's premier queuing mechanism. The other two algorithms (CQ and PQ) are also being used in its routers [5]. "For example Cisco Uses PQ to ensure that important Oracle-Based sales reporting data gets serviced ahead of other less critical traffic [5]." CQ is the queuing mechanism that Juniper Networks uses in its products [16, 17].

Our simulation experiments are conducted using two different scenarios for analysis and comparison of various performance aspects of the VirtualClock scheduling mechanism. The first scenario emulates an ideal situation in which sources follow their expected packet generation rates, except for the periods during which they do not conform to these rates. We look at the functionality of the scheduling mechanisms during the whole simulation time, specifically the nonconformance periods. In the second scenario we emulate a network, running traffic from real Internet applications: Hypertext Transfer Protocol (HTTP), File Transfer

Protocol (FTP), IP Telephony, and videoconferencing. We will study the impact of VirtualClock and other scheduling mechanisms on the performance of these applications during congestion periods. Following we describe the above mentioned simulation Scenarios 1 and 2 and explain the performance results driven by running simulations using the OPNET simulation tool.

5.1 Simulation Scenario 1

The purpose of this scenario is to compare the functionality of the VirtualClock scheduling mechanism with the above mentioned scheduling mechanisms. We look at the performance of these algorithms in comparison to VirtualClock for the time periods during which: 1) all sources conform to their specified traffic generation rate, 2) some sources decrease their traffic generation rate in order to collect credit for the periods with high traffic generation rate, and 3) sources try to use the collected credit for sending traffic with a rate larger than their expected traffic rate.

We compare the fairness of the algorithms in terms of allocating bandwidth to these sources at the congested point in the network and also the amount of buffer usage, number of dropped packets, and queuing delay in the allocated queues to each source.

This scenario is identical to the functionality test scenario in Chapter 4, page 67 of this document. As described before, there are three Ethernet clients 1, 2, and 3 sending traffic to three Ethernet servers 1, 2, and 3 respectively. Source 1 is non-conforming and sources 2 and 3 are conforming to their expected traffic generation rates. The network model for this scenario is shown in Fig. 5.1.

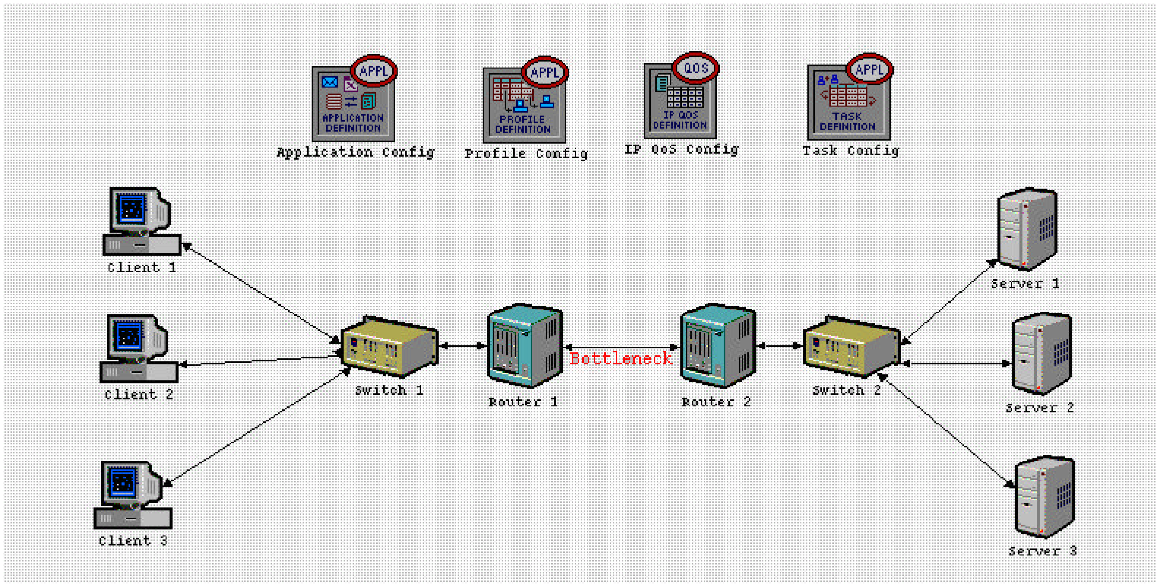
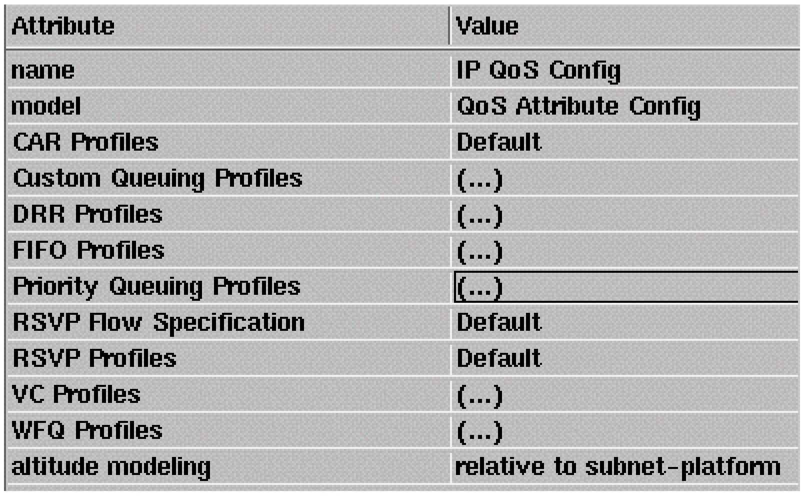


Fig. 5.1 Network Model for Scenario 1.

All nodes in the network are connected with 10BaseT links with the capacity of 10Mbps. The link between Routers 1 and 2 is the network bottleneck and is a DS0 link with 64Kbps data rate. Thus, Router 1 does not have the ability to forward all the traffic that arrives to the outgoing link. For this reason the scheduling mechanisms are performed at the beginning point of the bottleneck link (Router 1). These mechanisms are defined as attributes of the *IP QoS Configuration* Object, which has to be used in our network if we want to perform scheduling in one of the routers. Fig. 5.2 shows the attributes of this object. In case of congestion, by employing any of the above scheduling algorithms, the received traffic from Clients 1, 2, and 3 are categorized and stored in Router 1's output queues: Q0, Q1, and Q2, respectively. Fig. 5.3 shows the Incoming traffic to Q0, Q1, and Q2. All the traffic generation parameters of the sources are the same as those in the network of the functionality scenario and are specified in page 68 and 69.



Attribute	Value
name	IP QoS Config
model	QoS Attribute Config
CAR Profiles	Default
Custom Queuing Profiles	(...)
DRR Profiles	(...)
FIFO Profiles	(...)
Priority Queuing Profiles	(...)
RSVP Flow Specification	Default
RSVP Profiles	Default
VC Profiles	(...)
WFQ Profiles	(...)
altitude modeling	relative to subnet-platform

Fig. 5.2 Attributes of the *IP QoS Configuration* object.

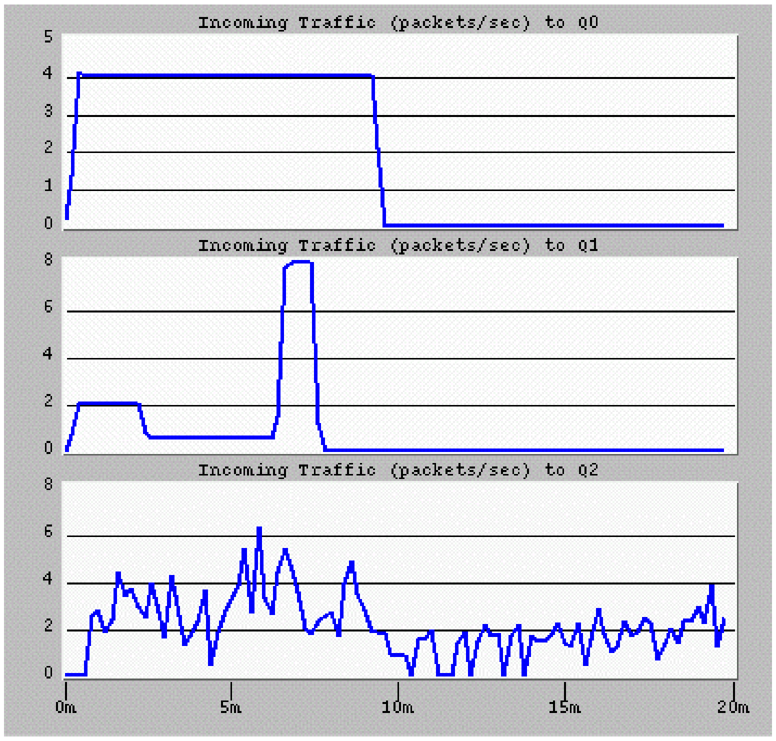


Fig. 5.3 Incoming traffic to queues: Q0 (top), Q1 (middle), and Q2 (bottom) in (packets/sec) vs. time (min).

In the following sub-sections, we repeat the simulation with the same network configuration and traffic generation parameters for the mentioned schedulers and compare the simulation results with the results of the VirtualClock scheduler.

In order to configure the VirtualClock queue parameters we use the queuing profile *Flow Based1*, defined in the *VC Profile* attribute of the *IP QoS Configuratio*n object and stated on pages 67 and 68 of this document.

5.1.1 VirtualClock vs. WFQ

In this section, first we explain the configuration of the WFQ algorithm in OPNET. Then we show the simulation results of WFQ compared with VirtualClock.

WFQ queue parameters are defined as sub-attributes of the *WFQ Profile* attribute of the *IP QoS Configuration* Object. In order to set appropriate values for the WFQ queue parameters, we define a new profile *Flow Based* in the WFQ Profile. This profile has three rows, and each row represents a queue with the following queue parameter settings:

- *$Weight_0$ = 4, $Queue\ Size_0$ = 500*
- *$Weight_1$ = 2, $Queue\ Size_1$ = 500*
- *$Weight_2$ = 2, $Queue\ Size_2$ = 500.*

WFQ assigns a weight to each flow, which determines the percentage of the link bandwidth assigned to each flow. Since the average arrival rate to queues Q0, Q1, and Q2 are 4, 2, and 2 respectively, and the packet sizes are equal for all the flows, we assign queue weights in proportion to the packet arrival rate to these queues. Thus, the bottleneck link bandwidth allocations to queues Q0, Q1, and Q2 are 50%, 25%, and 25%. The maximum length of each queue is defined by *Queue Size* attribute. When a queue is longer than the *Queue Size*, all the additional packets are dropped.

Fig. 5.4 shows the outgoing traffic from queues Q0, Q1, and Q2 of Router 1 after being scheduled by the VirtualClock (dark graph) and WFQ (light graph) algorithms.

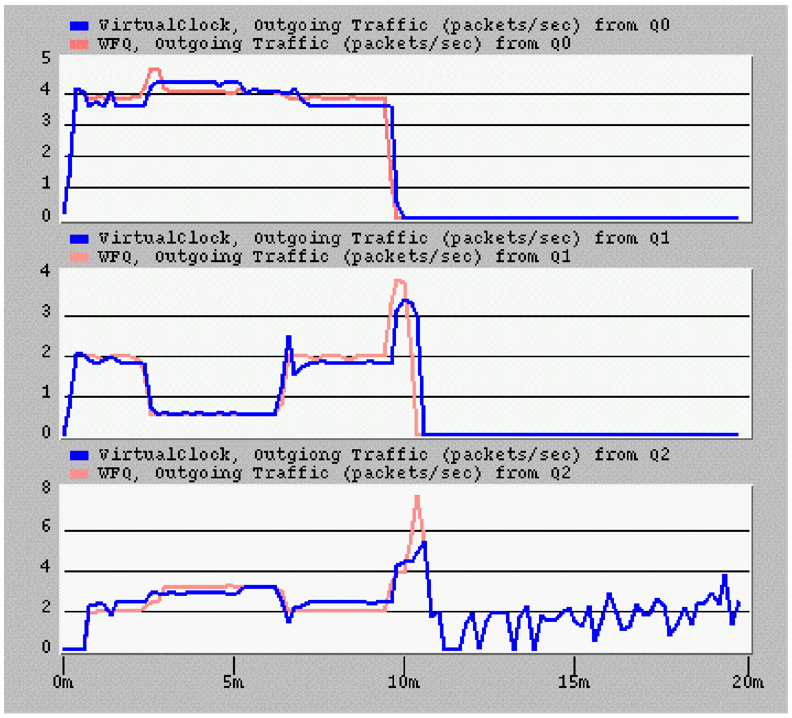


Fig. 5.4 VirtualClock vs. WFQ, outgoing traffic from queues: Q0 (top), Q1 (middle), and Q2 (bottom) in (packets/sec) vs. time (min).

In this scenario, if all three clients send traffic according to their specified traffic generation rate, the total capacity of the bottleneck link will be fully utilized. Otherwise, if a fair scheduling algorithm is being used at the congestion point, only nonconforming sources will be punished.

Since Client 1 is conforming to its specified traffic generation rate (4 packets/sec), it can be seen from the Fig. 5.4 (top) that both VirtualClock and WFQ serve the traffic from Q0 with the same rate. The middle graph in this figure shows that traffic from Q1 is serviced by both VirtualClock and WFQ with its arrival rate, while it conforms to its expected average arrival rate (from the beginning of the simulation until time 6 minutes and 32 seconds). This graph also indicates how VirtualClock and WFQ treat the flows fairly. This is done by reducing the traffic burst from Client 2 to its specified rate when it tries to make use of its collected share of bandwidth during time = 2 minutes and 22 seconds to time= 6 minutes and 32 sec in order to send a burst of traffic. The bottom graph shows that although Client 3 is fluctuating around its average traffic rate, both VirtualClock and WFQ serve Q3 with a constant rate equal to its specified rate (2 packets/sec).

WFQ scheduling is a well-known algorithm for its high degree of fairness [25, 26]. By comparing the VirtualClock and WFQ outgoing traffic from queues Q0, Q1, and Q2, it can be concluded that VirtualClock can also be considered as fair as the WFQ algorithm, since these graphs for each queue are tightly following each other, and have exactly the same behavior during sources' conforming and nonconforming periods.

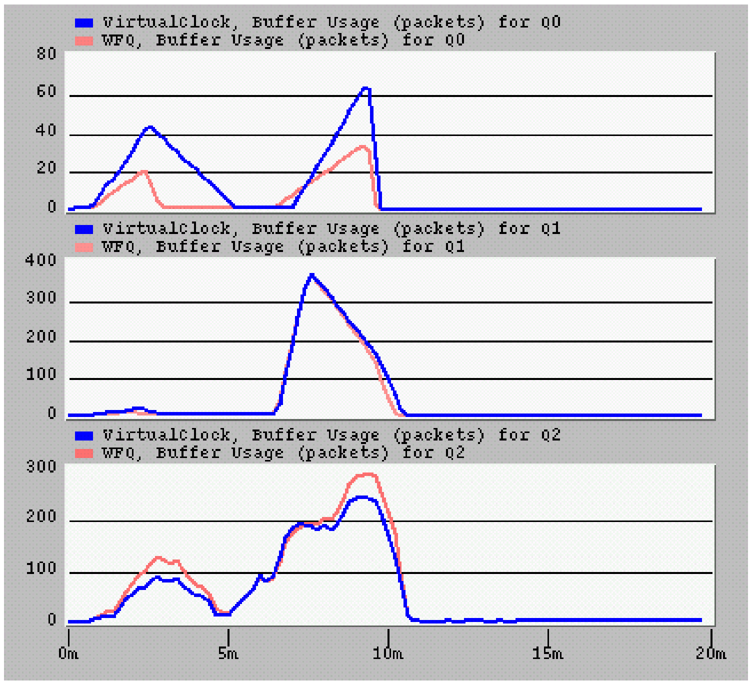


Fig. 5.5 VirtualClock vs. WFQ, buffer usage for Q0 (top), Q1 (middle), and Q2 (bottom) in (packets) vs. time (min).

Fig. 5.5 shows the amount of buffer usage for VirtualClock and WFQ queues: Q0, Q1, and Q2. The graphs: (top), (middle), and (bottom) show that the maximum buffer usage for all three queues occur during the bursty period of Client 2. By comparing the amount of the buffer usage in these queues, it can be seen that the buffer usage in Q0 (top graph) in comparison to the other queues is the lowest. The reason for this is that the incoming traffic to this queue has a constant rate, that is similar to the outgoing traffic rate. Thus, the packets from this flow do not need to get buffered in Q0. Although the average incoming traffic is equal to the average outgoing traffic from this queue, Q2 has a higher number of buffer usage. The reason is that since, the actual incoming traffic rate fluctuate around its average value, packets need to be buffered in Q2 to be serviced by the scheduler with a constant rate.

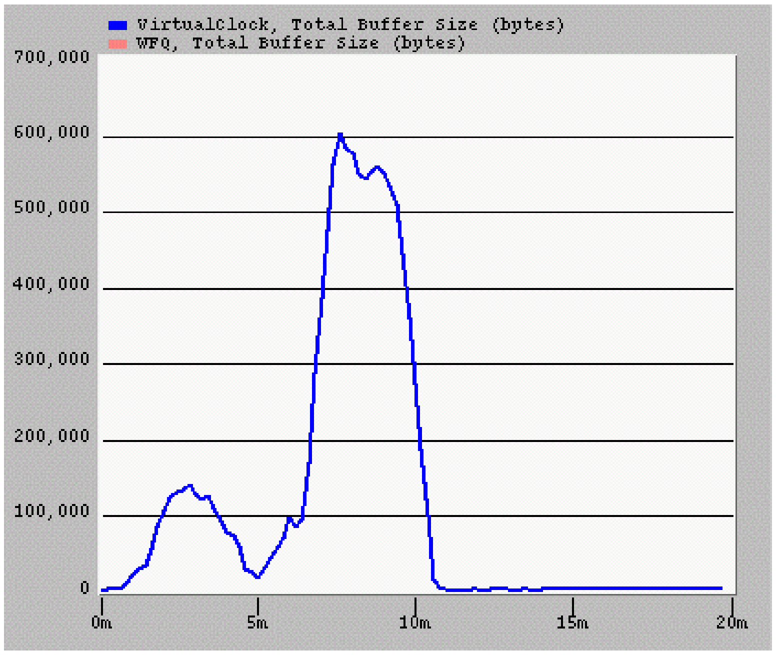


Fig. 5.6 VirtualClock vs. WFQ, total buffer usage in (bytes) vs. time (min).

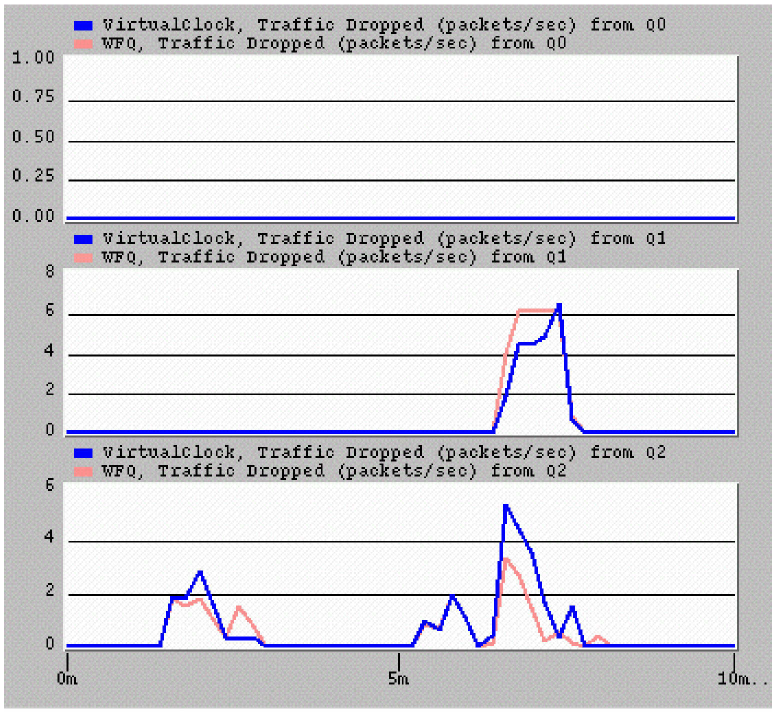


Fig. 5.7 VirtualClock vs. WFQ, traffic dropped from queues: Q0 (top), Q1 (middle), and Q2 (bottom) in (packets/sec) vs. time (min).

Fig. 5.6 shows the total buffer usage for VirtualClock and WFQ queues. It can be seen from the graph that all of the queues for these two algorithms are using exactly the same amount of buffer. By looking at Figs. 5.5 and 5.6 and comparing the amount of buffer usage (total amount and individually for each queue) for VirtualClock and WFQ queues, it can be concluded that both algorithms have the same behavior for buffer usage. Comparison of dropped packets from queues: Q0, Q1, and Q2 is shown in Fig. 5.7.

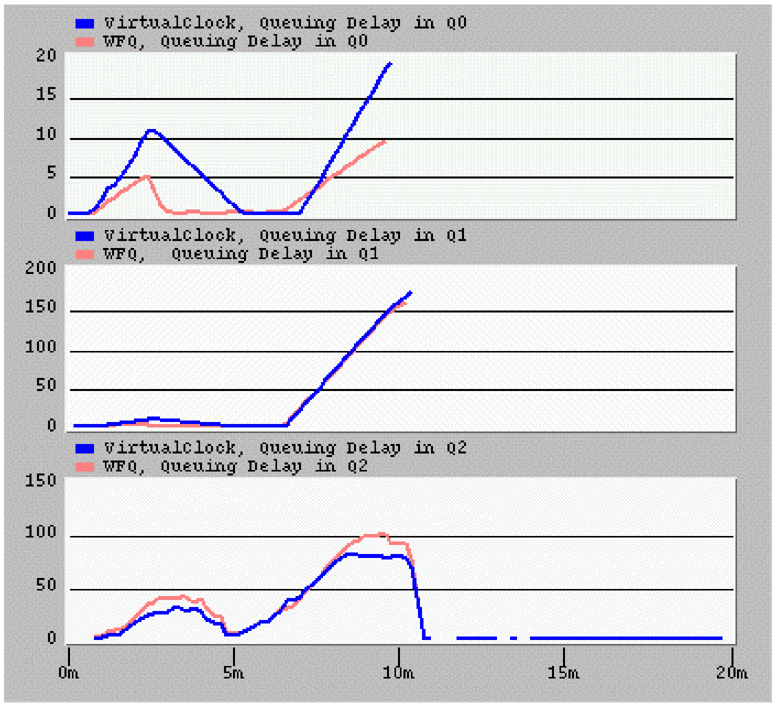


Fig. 5.8 VirtualClock vs. WFQ, queuing delay in queues: Q0 (top), Q1 (middle), and Q2 (bottom) in (sec) vs. time (min).

VirtualClock does not directly reduce queuing delay; however, it indirectly contributes to the queuing delay reduction. This is done by servicing individual traffic flows with their specified traffic generation rates so that traffic from each source will experience minimal queuing delay if it is transmitting with its expected rate [38] Fig. 5.8 (top), (middle), and (bottom) show the queuing delay in VirtualClock and WFQ queues for traffic from Clients 1, 2, and 3, respectively. The graphs show that, the highest delay among three queues is encountered by Q1 during the bursty period. On the other hand, the lowest delay is encountered by Q0, which stores the conforming flow. With a traffic rate that varies around its expected value, Q2 experiences higher delay during times when this variation is higher and lower delay during lower variation periods.

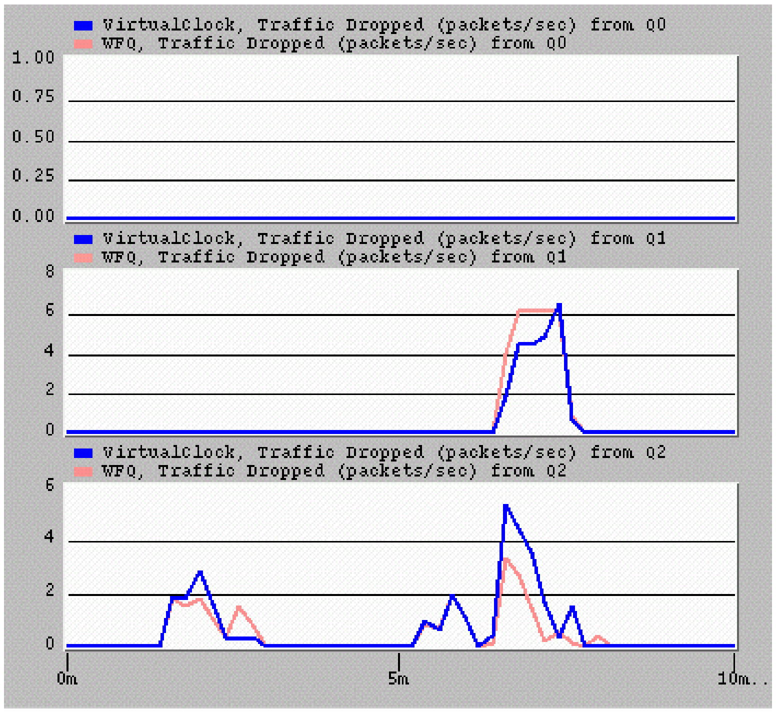


Fig. 5.9 VirtualClock vs. WFQ, traffic dropped from queues: Q0 (top), Q1 (middle), and Q2 (bottom) in (packets/sec) vs. time (min).

In order to evaluate the traffic loss behavior of the VirtualClock algorithm in comparison to WFQ, we reduce the value of the *Queue Size* attribute to 30 (packets) for the three queues in both algorithms and perform the simulation again. The reason for initially selecting Queue Size = 500 was that we could completely capture the packet forwarding behavior of the algorithms without losing any traffic in the queues. Fig. 5.9 shows the dropped traffic from VirtualClock and WFQ algorithms from the queues. The top graph shows that since Client 1 is a conforming source and sends traffic with a constant rate, the traffic entering Q0 is sent out of the queue with its incoming rate, thus no loss happens in this queue. The middle graph shows that the higher number of loss happens during Client 2's burst period and the amount of loss is similar for both of the algorithms. And the bottom figure shows that since the queue buffers are mostly being deployed during periods with higher variation around the average rate value, consequently these periods posse higher traffic loss.

From Figs. 5.2 to 5.9 we can observe that the functionality of VirtualClock in terms of fairness, resource (buffer) usage, and loss is very close to the WFQ algorithm.

5.1.2 VirtualClock vs. Custom Queuing

In this section we repeat Scenario 1 using Custom Queuing as the scheduling mechanism in Router 1. Following that we first describe the configuration of Custom Queuing algorithm in OPNET, then compare the driven simulation results from Custom Queuing with those of VirtualClock algorithm.

As mentioned earlier, Custom Queuing assigns a certain percentage of the bandwidth to each queue at a potential congestion point of the network. This particular percentage of the bandwidth can be indirectly specified in terms of the *Byte Count* variable defined for each queue. The scheduler cycles through the queues in a round-robin order. At each queue's turn packets are sent until the *Byte Count* value is exceeded, or the queue becomes empty. Once the *Byte Count* value is exceeded the packet that is currently being transmitted will be completely sent.

Similar to WFQ and VirtualClock, Custom Queuing queue parameters are defined as sub-attributes of the *CQ Profile* attribute of the *IP QoS Configuration* object. Custom Queuing queue parameters are identified by defining a *Flow Based Profil*e in the *CQ Profile*. Each row of this profile shows a queue with the following queue parameter settings:

- $ByteCount_0 = 2048$, $Queue\ Size_0 = 500$
- $ByteCount_1 = 1024$, $Queue\ Size_1 = 500$
- $ByteCount_2 = 1024$, $Queue\ Size_2 = 500$.

We assign values to the queue *Byte Count* variables in proportion to the expected packet arrival rate of the flows entering these queues. Following are the steps to be taken for determining the *Byte Count* value [5]:

- **Step 1:** For each queue, divide the desired percentage of bandwidth allocation to each queue by the packet size (byte).
- **Step 2:** Normalize the calculated numbers in step 1 by dividing them by their lowest number.

- **Step 3:** Round up the calculated values in step 2 to the next whole number. If the ratio values have a fraction, an additional packet should be sent. This step calculates the actual *Packet Count*.
- **Step 4:** Multiply the calculated *Packet Counts* for each queue by that queues packet size. This step converts the *Packet Count* numbers to *Byte Counts*.

In order to calculate the bandwidth distribution that the above *Byte Count* ratio allocates do the following next two steps:

- **Step 5:** calculate the total number of bytes sent after the algorithms serves each queue once by adding up the *Byte Counts* for each queue.
- **Step 6:** Calculate the percentage of number of bytes sent from each queue, and finally,
- **Step 7:** If this number is not close enough to the desired bandwidth, multiply the normalized values calculated in step 2 by the best value and do the other steps until getting close enough to the desired bandwidth percentage.

The average expected arrival rate to queues Q0, Q1, and Q2 are 4, 2, and 2 and all the packets are of the same size (1024 bytes). Thus, the bandwidth percentage for queues Q0, Q1, and Q2 is 50%, 25%, and 25%, respectively. The *Byte Count* values for the three queues are calculated by following the above steps. The number of serviced packets from queues Q0, Q1, and Q2 are 2, 1, and 1 in each round.

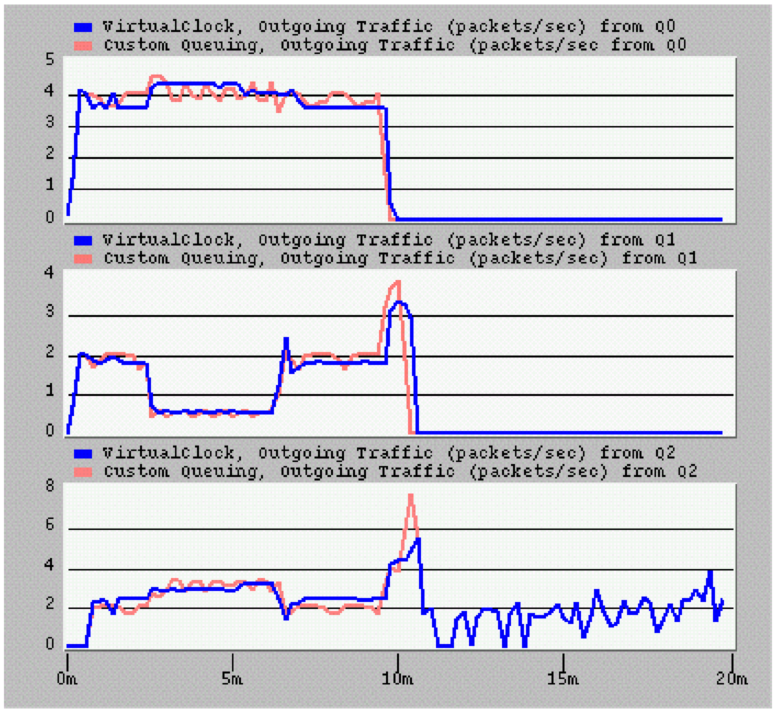


Fig 5.10 VirtualClock vs. Custom Queuing, outgoing traffic from queues: Q0 (top), Q1 (middle), and Q2 (bottom) in (packets/sec) vs. time (min).

Fig. 5.10 shows the outgoing traffic from queues Q0, Q1, and Q2 of Router 1 after being scheduled by the VirtualClock and Custom Queuing algorithms. The graphs (top), (middle), and (bottom) show that although the average outgoing traffic during different periods (conforming and nonconforming) is the same for both algorithms (as described in section 5.1), the outgoing traffic from the Custom Queuing algorithm has a different pattern.

Graphs show that the traffic is sent from the VirtualClock queues with a constant rate equal to the specified traffic rate of the flows entering the queues. On the other hand, the traffic from Custom Queuing queues is sent out with a rate, which oscillates around this specified rate.

Therefore, the graphs illustrate that VirtualClock treats the conforming flows more fairly than Custom Queuing.

Buffer usage individually for Q0, Q1, and Q2 and totally for VirtualClock and Custom Queuing queues is shown in Figs. 5.11 and 5.12, respectively. Fig. 5.13 shows the queuing delay that the traffic from each of the three sources encounter in queues Q0, Q1, and Q2. Dropped packets from each of these queues are illustrated in Fig. 5.14. For measuring the traffic dropped statistic, we set the *Queue Size* = 30, the same as stated in section 5.1.

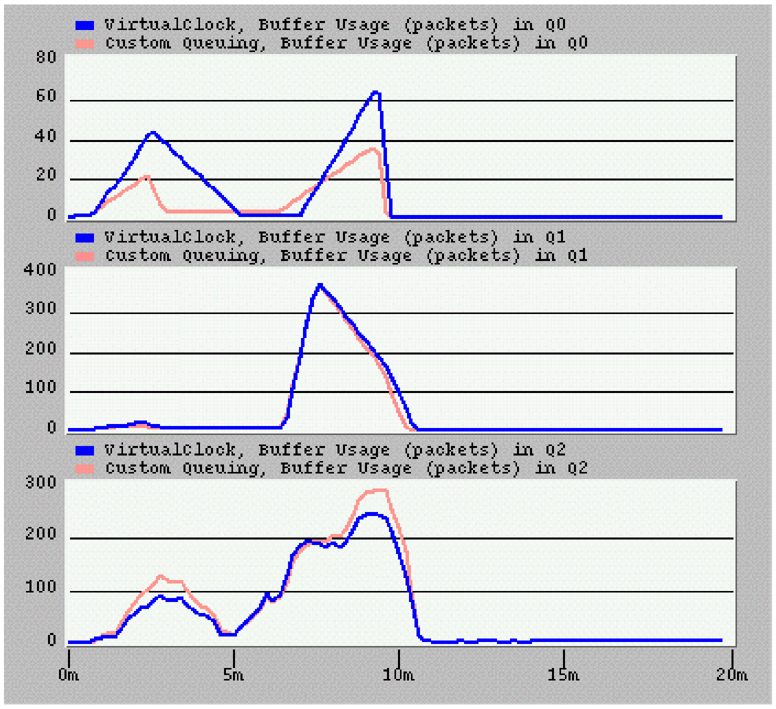


Fig. 5.11 VirtualClock vs. Custom Queuing, buffer Usage for queues: Q0 (top), Q1 (middle), and Q2 (bottom) in (packets) vs. time (min).

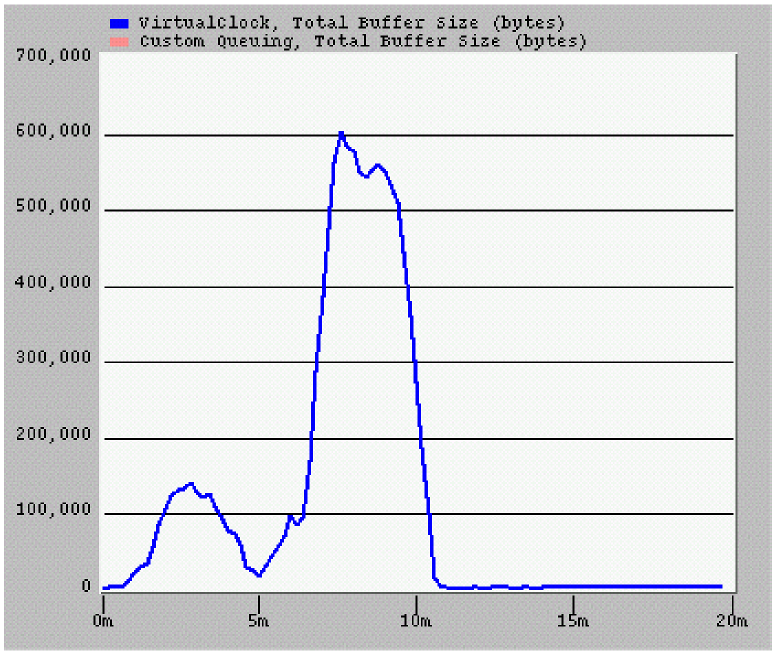


Fig. 5.12 VirtualClock vs. Custom Queuing, total buffer usage for queues in (bytes) vs. time (min).

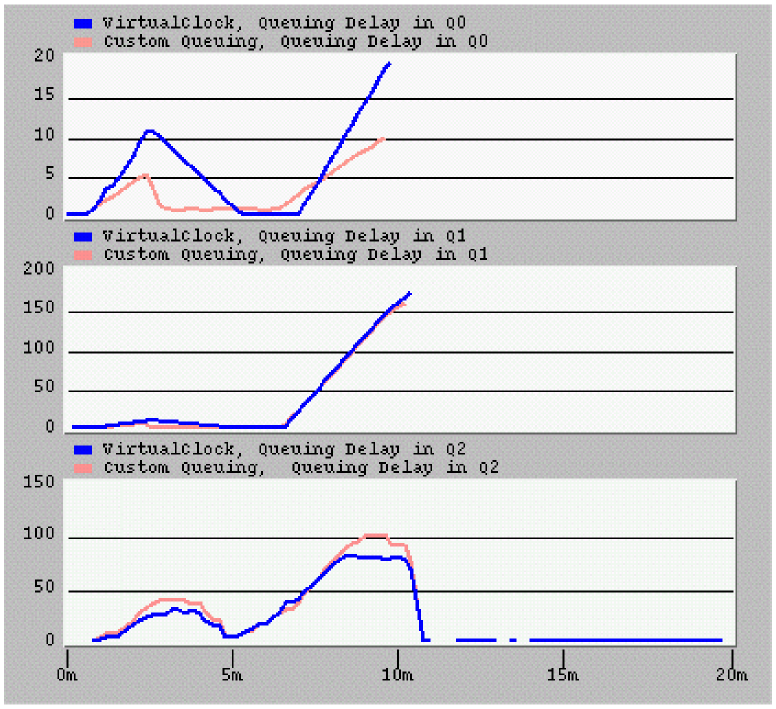


Fig. 5.13 VirtualClock vs. Custom Queuing, queuing delay in queues: Q0 (top), Q1 (middle), and Q2 (bottom) in (sec) vs. time (min).

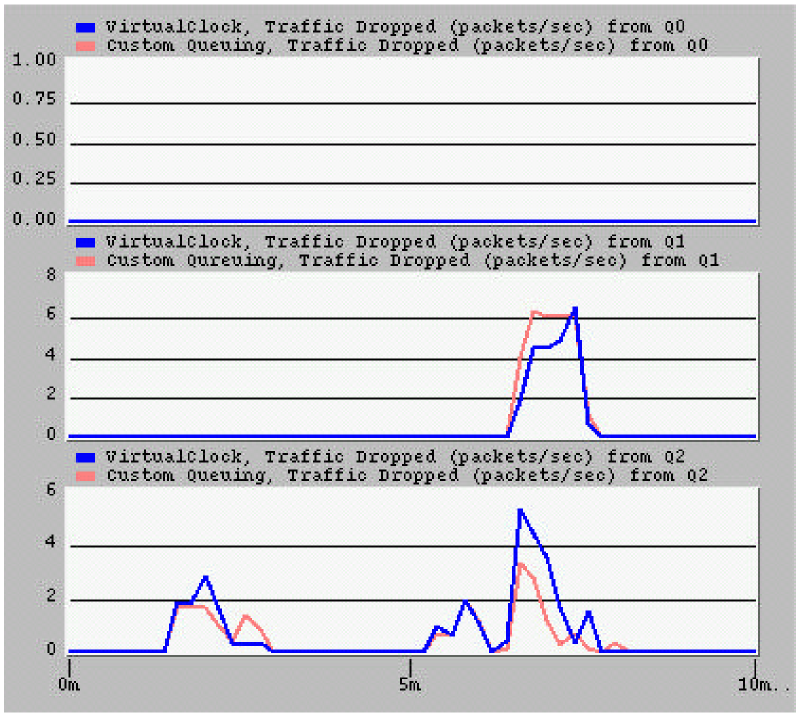


Fig. 5.14 VirtualClock vs. Custom Queuing, traffic dropped from queues: Q0 (top), Q1 (middle), and Q2 (bottom) in (packets/sec) vs. time (min).

Comparing the performance of VirtualClock and Custom Queuing in our simulation scenario by looking at Figs. 5.10 to 5.14, we can conclude that VirtualClock behaves more fairly than Custom Queuing. The constant pattern of traffic being scheduled by VirtualClock from different queues in comparison to the oscillating pattern of traffic from Custom Queuing proves this. On the other hand the functionality of these algorithms in terms of Buffer usage, queuing delay, and packet loss is very similar. However, WFQ and Custom Queuing behave more similarly in terms of the mentioned performance measurements.

5.1.3 VirtualClock vs. Priority Queuing

We replicate Scenario 1, employing the Priority Queuing scheduling algorithm at the output queues of Router 1 and re-running the simulations. In this section, we will explain the allocation of priority queues to flows from different sources. Then we observe the performance similarities and dissimilarities between VirtualClock and Priority Queuing.

In Chapter 2.5.2, we described the basic idea behind the Priority Queuing (PQ) mechanism. Using this mechanism, traffic can be prioritized into 4 priority classes: High, Medium, Normal and Low. For PQ implementation in OPNET, there is one queue associated with each priority group. Unclassified packets assigned into the normal queue. The traffic from higher priority queues will get preferential service over lower priority queues.

In this scenario we allocate traffic from Clients 1, 2, and 3 to low, medium, and high priority queues respectively. Similar to VirtualClock, WFQ, and Custom Queuing, PQ queue parameters are defined as sub-attributes of *PQ Profile* attribute of the *IP QoS Configuration* Object. The *Flow Based Profile* defines the three priority queues and their priorities. The priority of the queues is assigned by their location order and increases by increasing the queue number, meaning that Q2 has the highest and Q0 has the lowest priority.

- *Low Priority, Queue* $Size_0 = 500$
- *Medium Priority, Queue* $Size_1 = 500$
- *High priority, Queue* $Size_2 = 500$.

Fig 5.15 shows the outgoing traffic from VirtualClock and PQ queues. The bottom graph shows the traffic sent from Q2, the PQ's highest priority queue. The sent and received traffic of this queue using PQ are exactly the same, since the highest priority queue has access to the total link bandwidth. On the other hand, while the incoming traffic to this queue has variation around its specified traffic rate, VirtualClock regulates the outgoing traffic with a constant rate equal to the flow's specified rate.

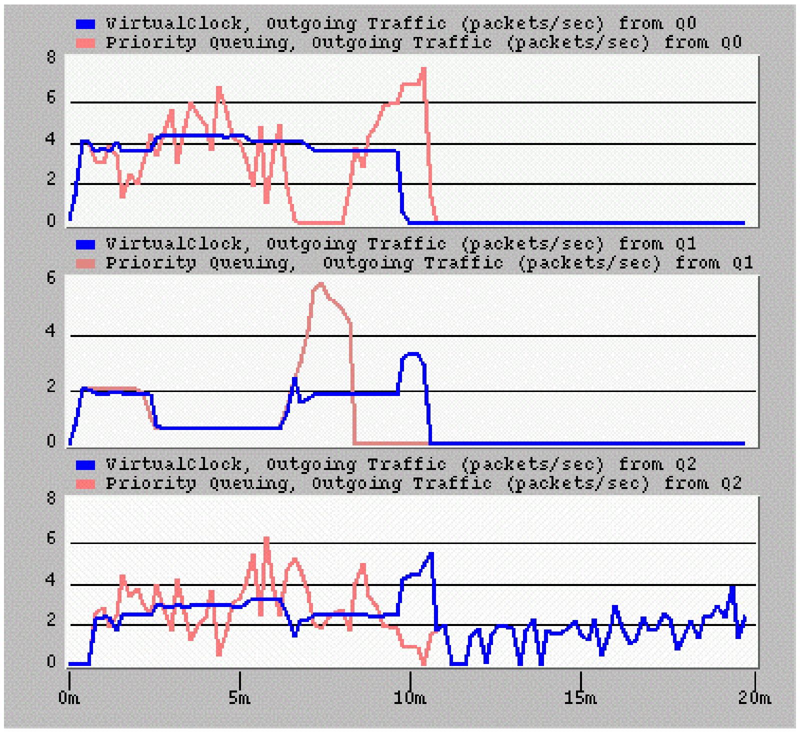


Fig 5.15 VirtualClock vs. Priority Queuing, outgoing traffic from queues: Q0 (top), Q1 (middle), and Q2 (bottom) in (packets/sec) vs. time (min).

The middle graph shows the outgoing traffic from Q1 (PQ's medium priority queue). Only during the traffic's nonconformance period the outgoing traffic is forwarded with a lower rate than incoming traffic to this queue. The medium priority queue can use the rest of the bandwidth left unused by the high priority queue. Therefore, the transmit rate during the nonconformance period is the maximum rate the flow can transmit until the total link bandwidth is occupied. The difference between the traffic sent from VirtualClock and Priority Queuing is also observable during this period. The top graph illustrates the outgoing traffic from Q3, the low priority queue of PQ scheduler. We can observe that the PQ scheduler assigns the remaining link bandwidth, not utilized by other priority queues, to Q0. Because during client 2's nonconformance period the total bandwidth is used by the high and medium priority queue, Q0 will not send any traffic. By comparing the Outgoing traffic emplying the two algorithms, it can be concluded that the VirtualClock is still transmitting traffic from Q0 with the flow's negotiated rate; however PQ can only transmit according to the remaining link bandwidth. It might also starve during certain periods.

Figs. 5.16 and 5.17 show the buffer usage in VirtualClock and PQ queues. The bottom graphs of this figures show that, since PQ forwards the packets to the outgoing link immediately after they arrive to Q2, these packets do not use the buffer and encounter no delay in the queue. The middle graphs show that the buffer is only occupied in Q1 during the traffic's nonconformance period. Thus the traffic is also being delayed in Q1 during this period. And finally the low priority queue (Q0) has a high amount of buffer occupancy during the simulation, particularly while client 1 is sending a burst of traffic. The buffer usage by this queue with PQ algorithm is almost five times the value of the

VirtualClock algorithm. Consequently, the traffic entering Q0 has a long queuing delay. Observe Fig. 5.17 (bottom graph).

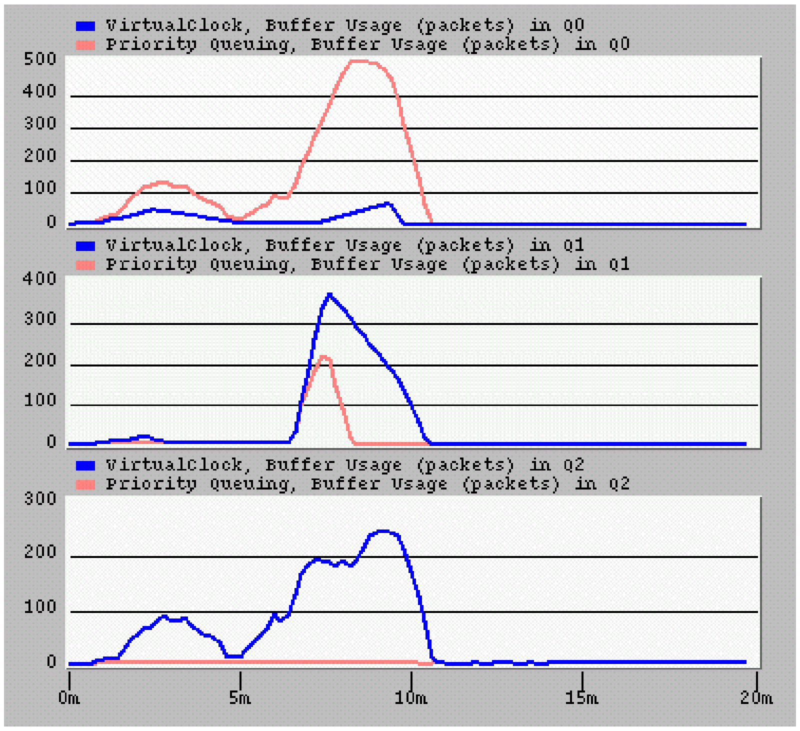


Fig. 5.16 VirtualClock vs. Priority Queuing, buffer Usage for queues: Q0 (top), Q1 (middle), and Q2 (bottom) in (packets) vs. time (min).

The traffic loss behavior of the PQ algorithm in comparison to VirtualClock is shown in Fig. 5.18 Similar to sections 5.1 and 5.2, we reduce the Queue size value of all the queues to 30 packets and repeat the simulation for observing the queues' loss behavior. It is observed from the bottom figure that since Q2's buffer occupancy is zero, no traffic is lost due to the buffer overflow in this queue. The middle graphs shows packet loss for Q1 only during its flow's nonconformance period and the top graph shows highest loss behavior during the simulation.

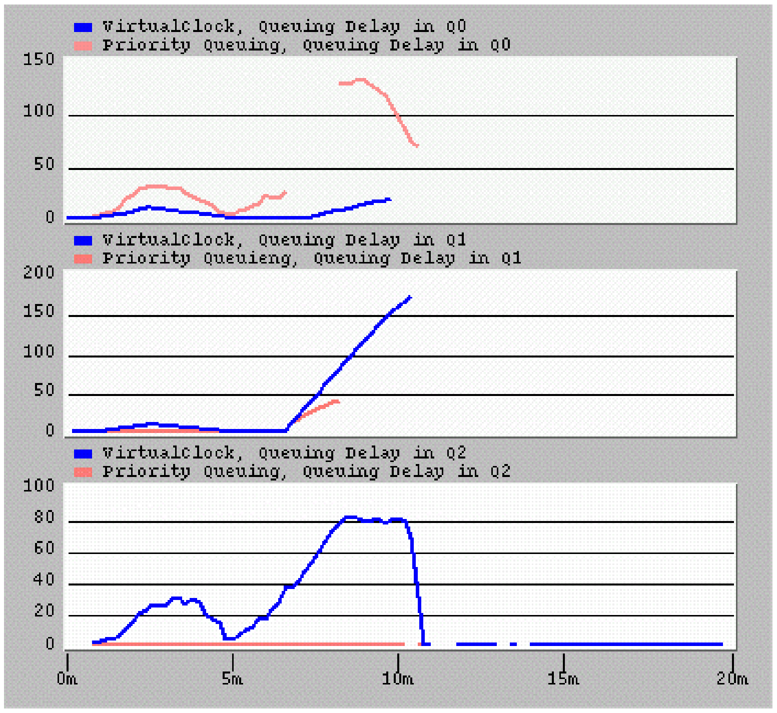


Fig. 5.17 VirtualClock vs. Priority Queuing, queuing delay in queues: Q0 (top), Q1 (middle), and Q2 (bottom) in (sec) vs. time (min).

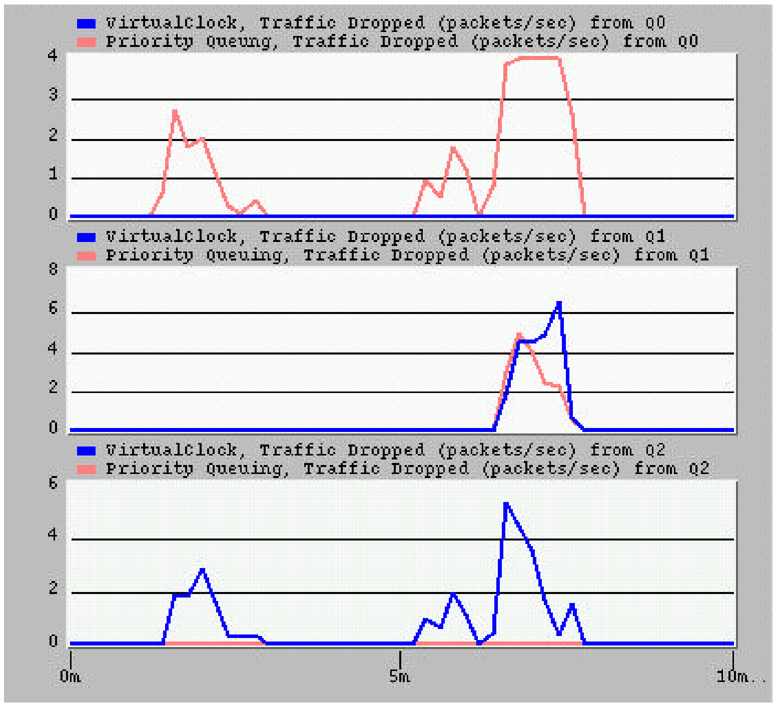


Fig. 5.18 VirtualClock vs. Priority Queuing, traffic dropped from queues: Q0 (top), Q1 (middle), and Q2 (bottom) in (packets/sec) vs. time (min).

5.2 Simulation Scenario 2

In this scenario we compare the effect of VirtualClock with the WFQ, Custom Queuing, and Priority Queuing scheduling mechanisms on the performance of four common Internet applications: Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), IP Telephony, and videoconferencing. Following there is a brief description about these protocols:

- **HTTP** is an application level data transfer protocol that forms the foundation of the Web. It is the protocol language that enables Web browsers (clients) to access files on Web servers. Thus, HTTP is implemented in two programs: the client program and the server program. The client program and server programs talk to each other by exchanging HTTP messages. HTTP defines the structure of these messages and how the client and server exchange the messages. In an HTTP session, Web clients (i.e., browsers) request Web pages from web servers and servers transfer Web pages to clients. HTTP communication usually takes place over TCP/IP connections.

- **FTP** is a protocol for transferring files from one host to another host. In an FTP session, the user is sitting in front of the local host and wants to transfer files to or from a remote host. In order for the user to access the remote account, the user has to provide a user identification and a password. After providing this authorization information, the user can transfer files from the local file system to the remote file system and vice versa [27]. In contrast to HTTP that uses one TCP connection, FTP uses two TCP connections: a control connection to send control information and a data connection to transfer files. When the user requests a file transfer (either to or from the remote host), FTP opens aTCP connection on server port. FTP sends one file over this connection and then closes the connection. If, during the same session, the user wants to transfer another file, FTP opens another data TCP connection.

- **IP Telephony** refers to services that transport audio or facsimile traffic over the Internet, rather than the public switched telephone network, employing the Internet Protocol. When an Internet telephone call originates, the analog voice signal is converted to a digital format and the signal is compressed into the Internet protocol (IP) packets for transmission over the Internet. A reverse process is performed at the receiving end. IP telephony, also called Voice Over IP (VoIP), requires certain bounds on packet end-to-end delay and delay jitter to ensure a voice quality similar to the conventional phone calls. Delay is an important factor that affects the quality of the conversation. Humans can tolerate about 250 msec of delay before it is noticed. Today audio quality has been observed to become quite poor in a pure best-effort network, especially in congested periods because it exceeds this delay. So, in order to improve the delay problem, QoS routers are used in the networks to allocate sufficient bandwidth to voice traffic by giving it a high service priority. Also voice compression standards like G.729 (8:1) and G.723 (10:1) are used to minimize the bandwidth required for voice. Without compression, a voice signal requires 64 Kbps of bandwidth to maintain good quality.

G.723, for instance, is the maximum compression rate and requires only 5.3 Kbps (plus an added 7.8 Kbps for IP overhead) [8, 20]. Table 5.1 shows the IP telephony QoS requirements.

Quality	**Packet loss (%)**	**Max delay jitter (ms)**
Perfect	0	0
Good	3	75
Medium	10	125
Poor	25	225

Table 5.1 IP Telephony QoS requirements.

- **Videoconferencing** is a technology that allows the transmission of digital voice conversations and video pictures over the Internet. The QoS requirements for transmitting voice and video are much different than for data. The required bandwidth for data signals depends on the amount of tolerable delay in receiving the file. However, voice and video need more synchronization to move pieces of the information at a constant rate so it can be heard and seen in real time. Also bandwidth requirements for voice and video are more critical than for data. As mentioned before, without compression, a good quality voice signal requires 64 Kbps of bandwidth. However, video signals require a much larger bandwidth without compression. The amount of required bandwidth for video depends on expected image resolution, frame rate, and compression technique. Image resolution is the image's clarity and is based on the number of available pixels that represent the image. The frame rate is how often the image is captured and sent out to the network. Among compression techniques, JPEG (Joint Photographic Experts Group) and MPEG (Moving Pictures Experts Group) are two ISO/ITU (International Telecommunication Union) standards for lossy compression. JPEG compresses each frame independently in a video sequence, providing compression ratio of 100:1 that decreases bandwidth requirements by a factor of 100. MPEG compresses each frame dependent on previous frames. It means that the information will not be repeated in the current frame if the frame includes similar information from the previous one. Thus, the compression ratio is improved to 300:1 in this technique.

In this scenario we have an HTTP Client downloading and uploading from an HTTP Server and an FTP Client that transfers and receives files from an FTP Server. In addition we have two voice parties talking through a voice session and two videoconferencing parties connected with a videoconferencing session. The topology in this scenario is similar to Scenario 1 i.e., HTTP client, FTP Client, Voice Party1, and Videoconferencing Party1 are connected to an Ethernet switch. The HTTP and FTP Servers together with Voice Party2 and Videoconferencing Party2 are connected to another Ethernet switch. Each of the switches is connected to an IP router. All the links in the network are 10BaseT (10Mbps) except the link between the IP routers, which is a DS1 link of capacity 1.544 Mbps. Because unlike Scenario 1, a bi-directional traffic is flowing through the link between router A and router B (bottleneck link), the scheduling mechanisms are being employed at both of the routers. Fig. 5.19 shows the network model for Scenario 2. We will describe the traffic specification for each of the application sessions next.

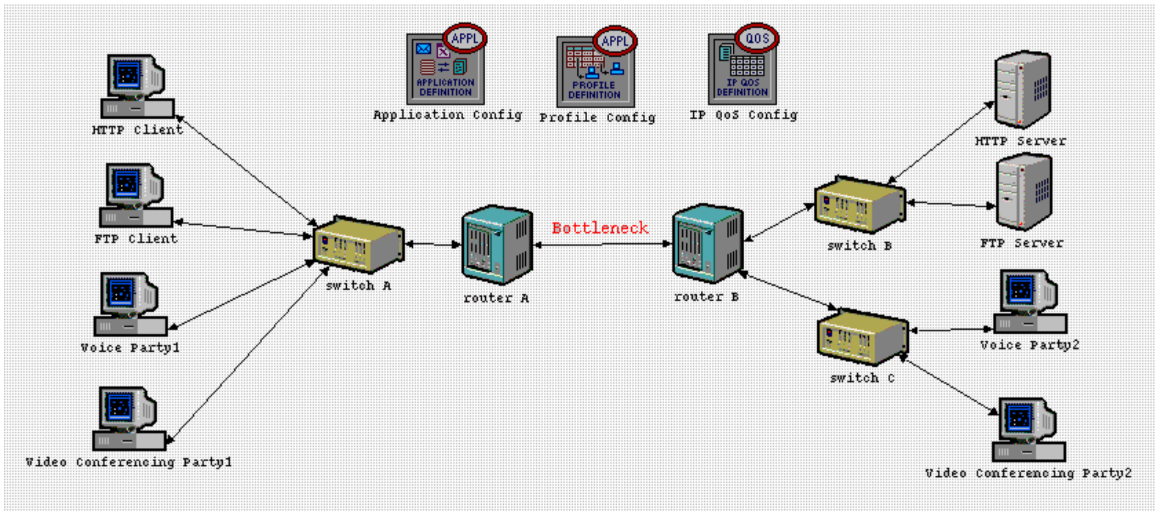
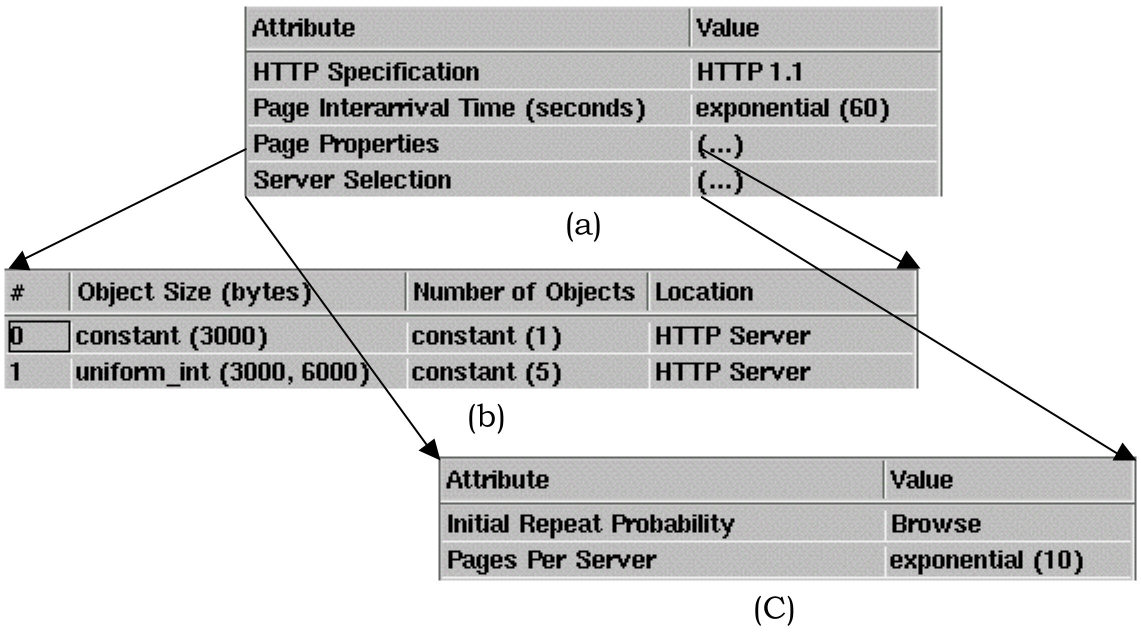


Fig. 5.19 Network model for Scenario 2.



Attribute	Value
HTTP Specification	HTTP 1.1
Page Interarrival Time (seconds)	exponential (60)
Page Properties	(...)
Server Selection	(...)

(a)

#	Object Size (bytes)	Number of Objects	Location
0	constant (3000)	constant (1)	HTTP Server
1	uniform_int (3000, 6000)	constant (5)	HTTP Server

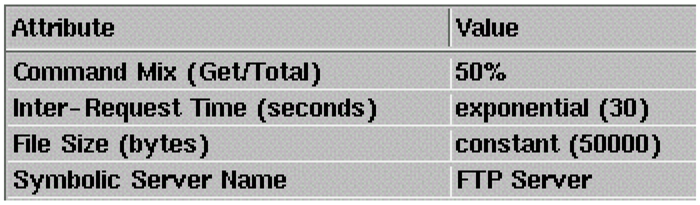
(b)

Attribute	Value
Initial Repeat Probability	Browse
Pages Per Server	exponential (10)

(C)

Tables 5.2 Traffic specification for HTTP application: (a) HTTP table, (b) Page Properties table, and (c) Server Selection table.

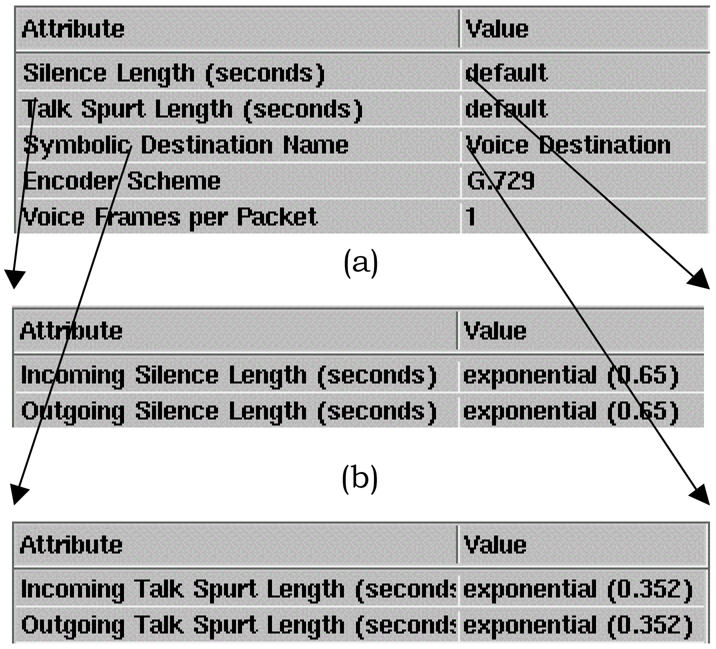
The traffic specification for HTTP application is defined in the above tables. In the top table, the HTTP Specification attribute defines the version of HTTP [9]. The Page Interarrival Time value shows the distribution name and argument to be used for generating random outcomes for page Interarrival times. The Page Properties table (b) specifies the page properties. Each page contains some objects and each object is represented by a row in this table. The first row represents the page itself and the subsequent rows represent the objects within the page. For each object the first column shows the distribution and arguments for size of each object in bytes and the second column shows the number of objects. Table (c) defines the Server Selection table. Since HTML pages have links to other pages either on the same server or on the remote server, this affects the server selection. Different values for the Initial Repeat Probability attribute can emulate different user behaviors. The browsing value for this attribute means a user goes to a site and access many links on that site before moving to another site. The second row in the table defines the number of pages per each server [23].



Attribute	Value
Command Mix (Get/Total)	50%
Inter-Request Time (seconds)	exponential (30)
File Size (bytes)	constant (50000)
Symbolic Server Name	FTP Server

Table 5.3 FTP application table.

Table 5.3 shows the parameters for the FTP application. The client downloads one file per FTP session. The Command Mix (Get/Total) attribute denotes the percentage of file get command to the total FTP commands. The remaining percentage is file put transactions. The second and third row show the distribution name and argument used for generating time between file transfers and file size.



Attribute	Value
Silence Length (seconds)	default
Talk Spurt Length (seconds)	default
Symbolic Destination Name	Voice Destination
Encoder Scheme	G.729
Voice Frames per Packet	1

(a)

Attribute	Value
Incoming Silence Length (seconds)	exponential (0.65)
Outgoing Silence Length (seconds)	exponential (0.65)

(b)

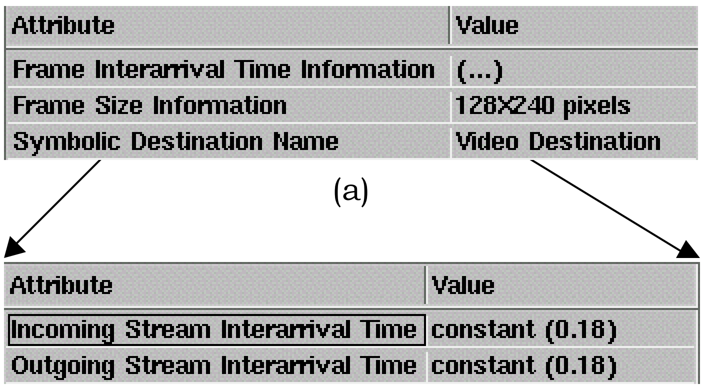
Attribute	Value
Incoming Talk Spurt Length (second:	exponential (0.352)
Outgoing Talk Spurt Length (second:	exponential (0.352)

(c)

Tables 5.4 Traffic specification for IP Telephony application: (a) HTTP table, (b) Spurt Length, and (c) Talk Spurt Length.

The traffic specification for IP Telephony application is defined in tables 5.4 (b) and (c). These tables specify the distribution and arguments to be used for random outcome generation for Silence and Talk Spurt Length, respectively. The Silence Length table specifies the time spent by the called party (incoming) and the calling party (outgoing) in a silence mode during a speech-silence cycle. The Talk Spurt Length table (c) specified the time spent by the called (incoming) and calling party (outgoing) in the speech mode. The fourth row in table 5.4 (a) shows the encoding scheme to be used by called and calling parties [23]. Coding techniques for telephony and voice packet are standardized by the ITU-T (International Telecommunication Union-Telecommunication Standardization Sector) in its G-series recommendations. We choose

G.729, an encoder scheme where voice is coded into 8-kbps streams. This scheme has an algorithmic delay of less than 16 ms. [6]. The last row in table (a) determines the number of encoded voice frames grouped into a voice packet before being sent by the application to the lower layers.



Attribute	Value
Frame Interarrival Time Information	(...)
Frame Size Information	128X240 pixels
Symbolic Destination Name	Video Destination

(a)

Attribute	Value
Incoming Stream Interarrival Time	constant (0.18)
Outgoing Stream Interarrival Time	constant (0.18)

(b)

Tables 5.5 Traffic specification for videoconferencing application: (a) videoconferencing table, and (b) Frame Interarrival Time Information table.

Tables 5.5 define the traffic specifications for the videoconferencing application. The Frame Interarrival Time Information table (b) shows the frame rate in frames/sec for the incoming and outgoing video streams. The second row in table (a) shows the frame size of the incoming and outgoing traffic streams [23].

We will replicate the above simulation scenario with the same network configuration and traffic generation parameters for the applications, with different scheduling mechanisms: VirtualClock, WFQ, CQ, and Priority Queuing. In each sub-section, we will compare the performance of these applications using the above scheduling mechanisms with the application performance using VirtualClock. The queue allocation for all the algorithms in both router A and B is the same i.e., HTTP, FTP, IP Telephony, and videoconferencing packets are assigned to Q0, Q1, Q2, and Q3, respectively. Next we will define the queue parameters for the VirtualClock queues.

Initially we run the simulation for the network with the VirtualClock algorithm deployed at the routers. The parameters for VirtualClock queues, defined in *VC Profile,* are as follows:

- *$ArrivalRate_0$ = 0.3, Queue $Size_0$ = 500*
- *$ArrivalRate_1$ = 1.5, Queue $Size_1$ = 500*
- *$ArrivalRate_2$ = 110, Queue $Size_2$ = 500*
- *$ArrivalRate_3$ = 140, Queue $Size_3$= 500.*

The *ArrivalRate* value for each queue is equal to the measured average Packet arrival rate (packets/sec) from the applications to that queue. It has to be taken into consideration that since VirtualClock algorithm services the flows in a packet by packet basis, the Arrival Rate should be measured in packets/sec.

5.2.1 VirtualClock vs. WFQ

In this section we will show the simulation results of WFQ in comparison with VirtualClock for Scenario 2. The WFQ queue configuration is defined below with each row representing a queue having the following queue parameter settings:

- *$Weight_0$ = 0.02, Queue $Size_0$ = 500*
- *$Weight_1$ = 0.58, Queue $Size_1$ = 500*

- *$Weight_2$ = 1.94, Queue $Size_2$ = 500*
- *$Weight_3$ = 97.45, Queue $Size_3$ = 500.*

For calculating the associated weight to each queue, the percentage of the link bandwidth consumption by each application flow has to be determined. This is done by measuring the average traffic rate (bytes/sec) from each flow. The average packet arrival rate to queues, Q0, Q1, Q2, and Q3 is 0.3, 1,5, 100, and 140 (packets/sec) with the average packet size (bytes) 154, 767.2, 37, and 1325, respectively. Therefore, the above queue weights show the percentage of required bandwidth by each queue.

Fig. 5.19 shows the HTTP page response time and average page response by VirtualClock and WFQ algorithms. The page response time is measured from the time the client requests a page from the server until the time the page is downloaded to the client's computer. By comparing the amount of this variable for VirtualClock and WFQ, it can be seen that WFQ provides a lower average page response time for HTTP than VirtualClock. The top graph shows that, although the page response time difference using the two algorithms is very high for a short period at the beginning of the simulation, for the rest of the simulation this difference disappears.

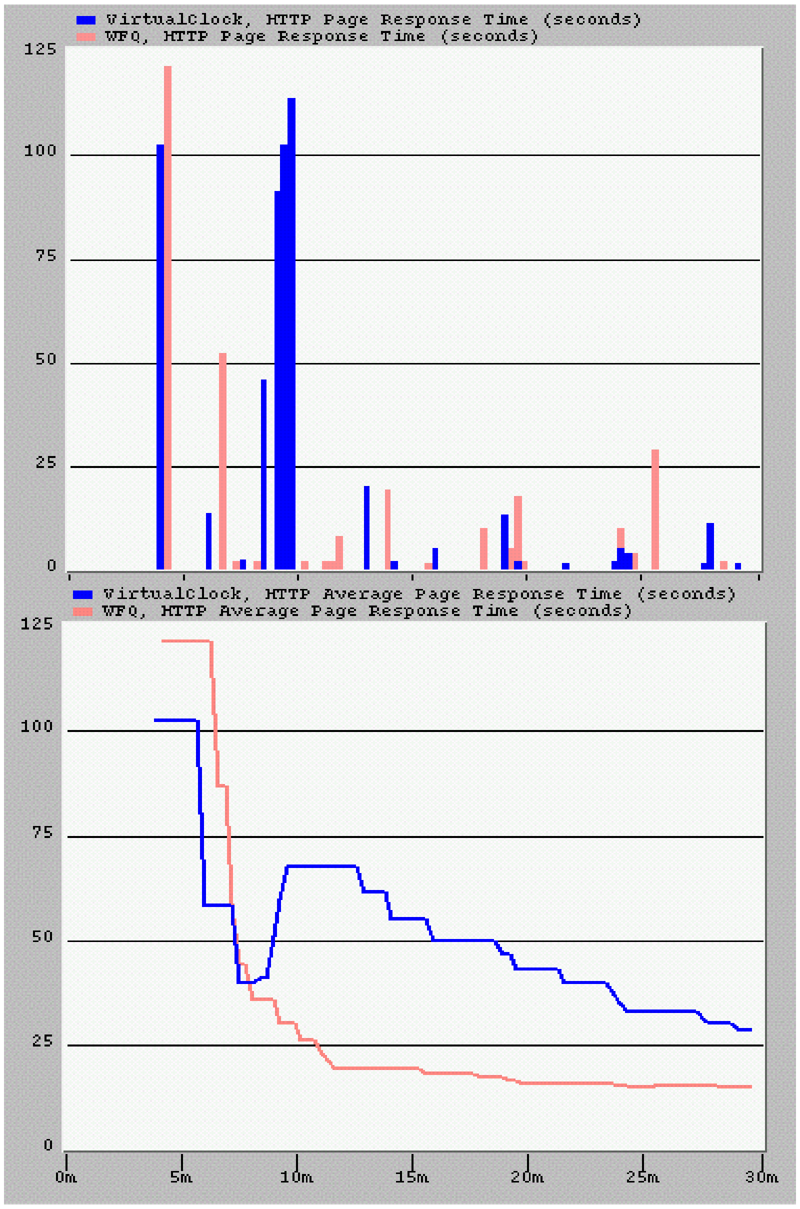


Fig 5.19 VirtualClock vs. WFQ, HTTP page response time (top) and average page response time (bottom) in (sec) vs. time (min).

Fig. 5.20 shows the FTP download response time and average download response time using VirtualClock and WFQ algorithms. Download time is measured from the time a client application sends a request to the server to the time it receives the requested file. The bottom graph shows that VirtualClock has a lower average FTP download response time than WFQ. However, the top graph shows that both algorithms provide the same download time during the simulation. The difference in FTP average download response time between the two algorithms (seen in the top graph) is caused by different delays encountered by security packets at the beginning of the simulation.

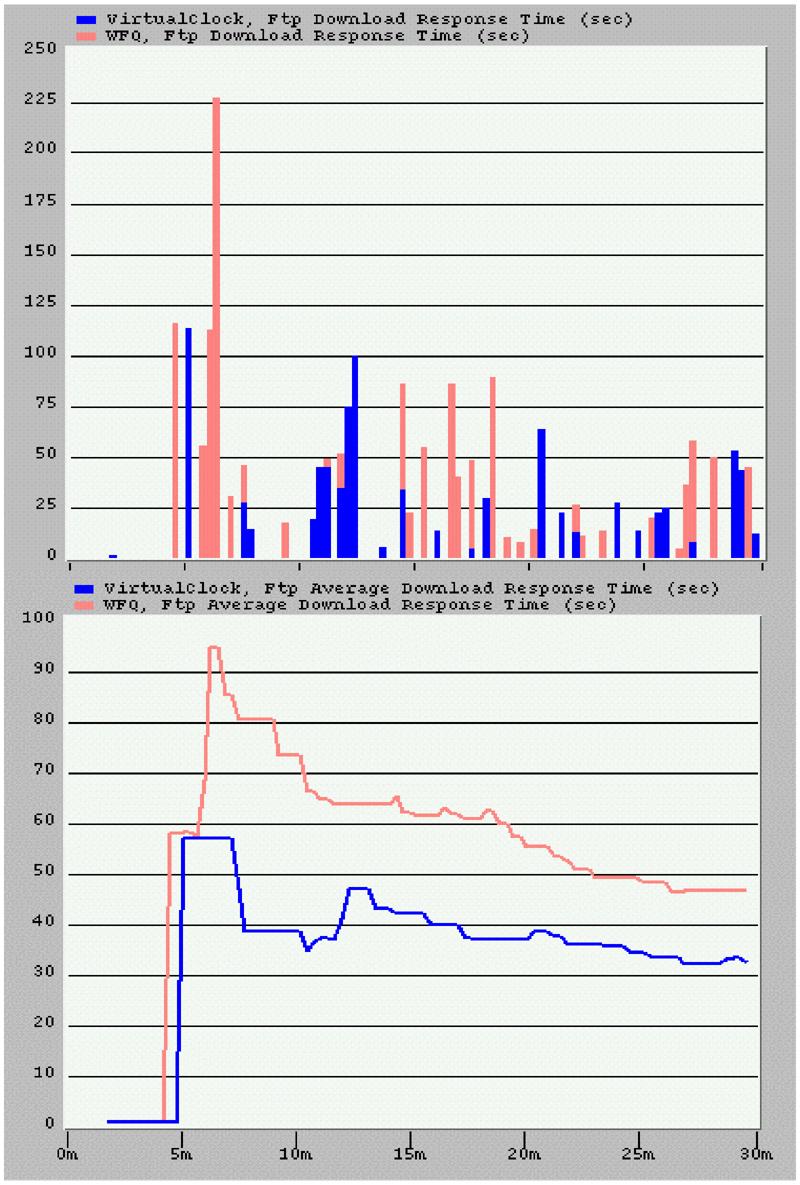


Fig 5.20 VirtualClock vs. WFQ, FTP downloads response time (top) and average download response time (bottom) in (sec) vs. time (min).

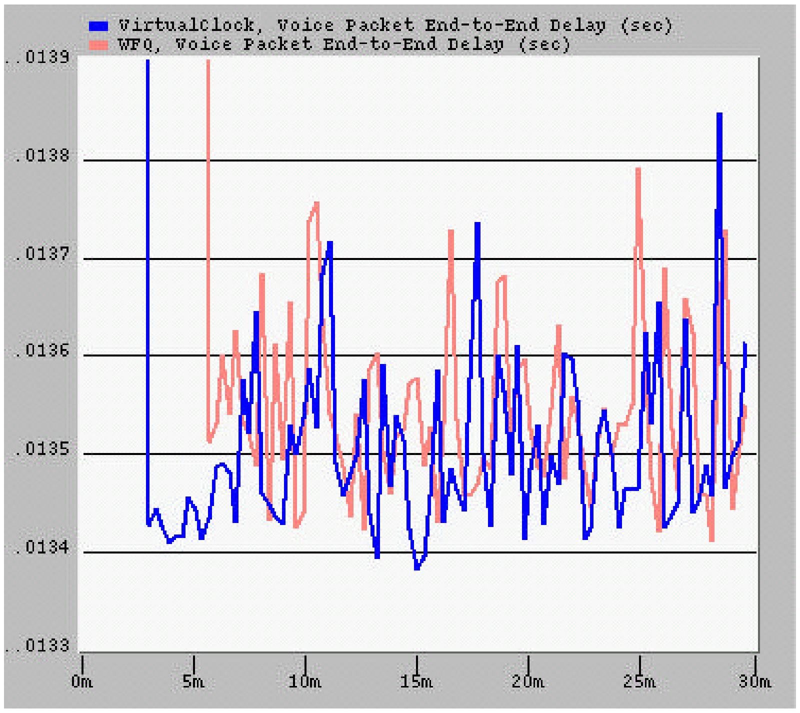


Fig 5.21 VirtualClock vs. WFQ, IP Telephony, voice packet end-to-end delay in (sec) vs. time (min).

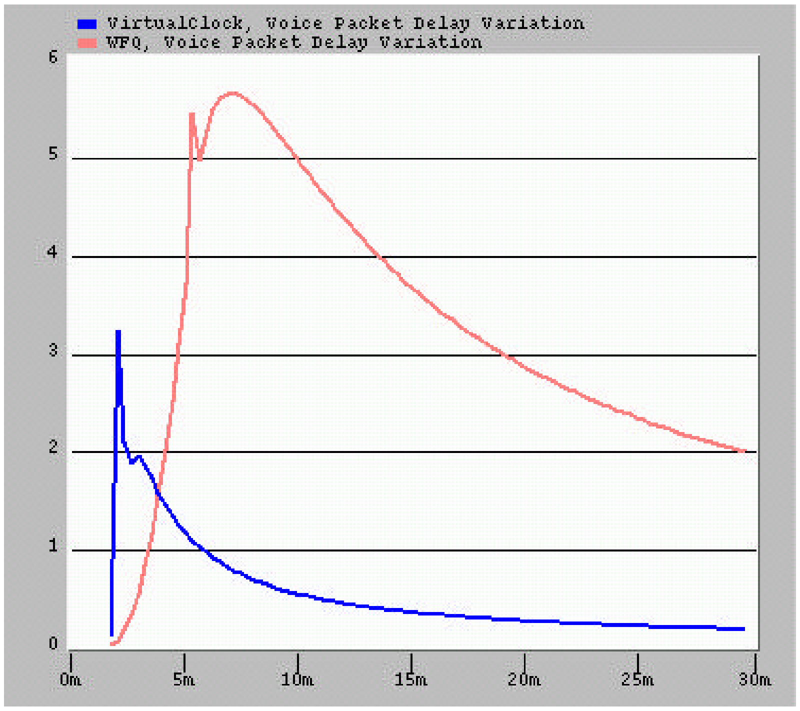


Fig 5.22 VirtualClock vs. WFQ, IP Telephony, voice packet delay variation (sec) vs. time (min).

The voice packet end-to-end delay from the IP Telephony application is shown in Fig. 5.21. It can be seen that the effect of both VirtualClock and WFQ on the voice packets end-to-end delays is very similar. Fig. 5.22 compares the delay variation encountered by voice packets using the two algorithms. It shows that the VirtualClock causes a much lower delay variation for voice packets. However, delay variation which is an essential factor for a good quality voice connection has a very high value for both of the algorithms due to the high congestion in the network. The highest tolerable delay variation for the poorest quality voice is 225 msec.

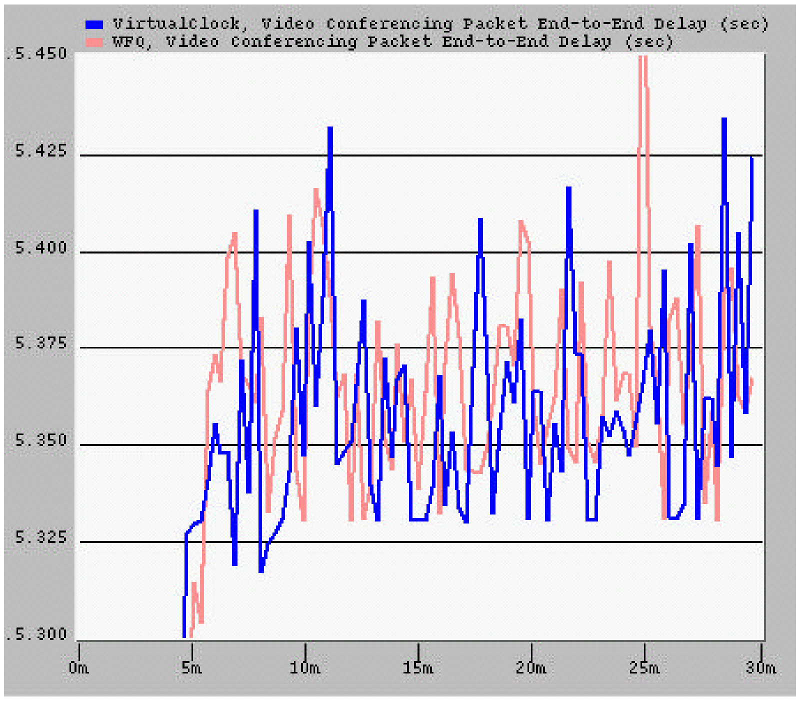


Fig 5.23 VirtualClock vs. WFQ, Videoconferencing packet end-to-end delay in (sec) vs. time (min).

In a videoconferencing session if the packets arrive late but their lateness is even and predictable by the receiving terminal the quality of the conference can still be retained. The important factor is variation in delay that results in an uneven and unpredictable quality within a video conference [36]. However, excessive delay increases the chances of people talking over one another because they do not realize that the person at the other end has started speaking too. Fig. 5.23 shows the videoconferencing end-to-end delay for VirtualClock and WFQ algorithms. The graph shows that the end-to-end delay value is similar for both WFQ and VirtualClock. It can also be observed that the amount of delay variation is low for both of the algorithms, although this delay is high.

5.2.2 VirtualClock vs. Custom Queuing

We repeat Scenario 2 with the Custom Queuing mechanism at the routers and look at the performance of the applications using VirtualClock in comparison to Custom Queuing. What follows are the queue Custom Queuing parameter settings. The *Byte Count* values for the Custom Queuing queues are calculated by measuring the average bandwidth requirement (bits or bytes) and average packet size (bytes) for each application flow and following the described steps in section 5.1.2. The computed queue parameter setting for the Custom Queuing queues are as follows:

- $ByteCount_0 = 154$, $Queue\ Size_0 = 500$
- $ByteCount_1 = 7020$, $Queue\ Size_1 = 500$
- $ByteCount_2 = 14907$, $Queue\ Size_2 = 500$
- $ByteCount_3 = 719475$, $Queue\ Size_2 = 500$.

HTTP page response time and average page response time is shown in Fig. 2.24 It can be seen that the average page response time for both VirtualClock and Custom Queuing is relatively similar. Fig. 5.25 illustrates the FTP download and average download response time. The bottom graphs show that VirtualClock provides a lower average file download time than Custom Queuing.

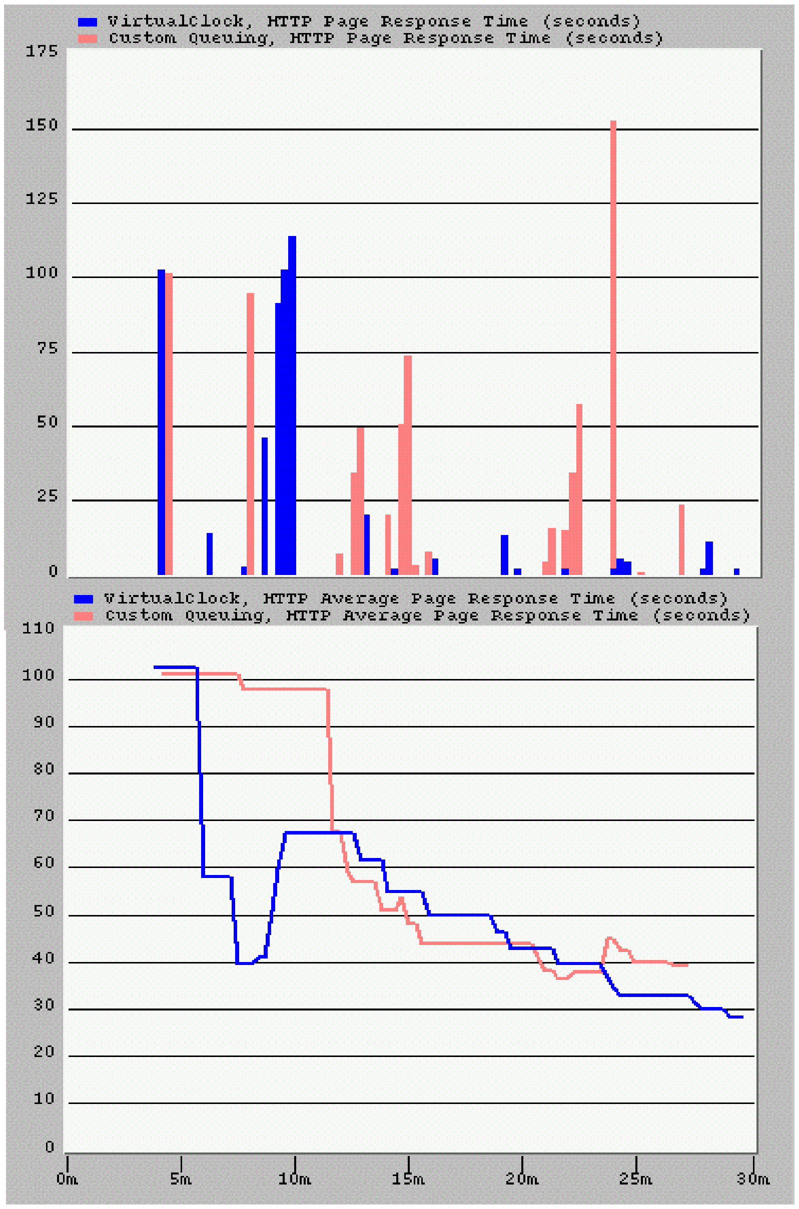


Fig 5.24 VirtualClock vs. Custom Queuing, HTTP page response time (top) and average page response time (bottom) in (sec) vs. time (min).

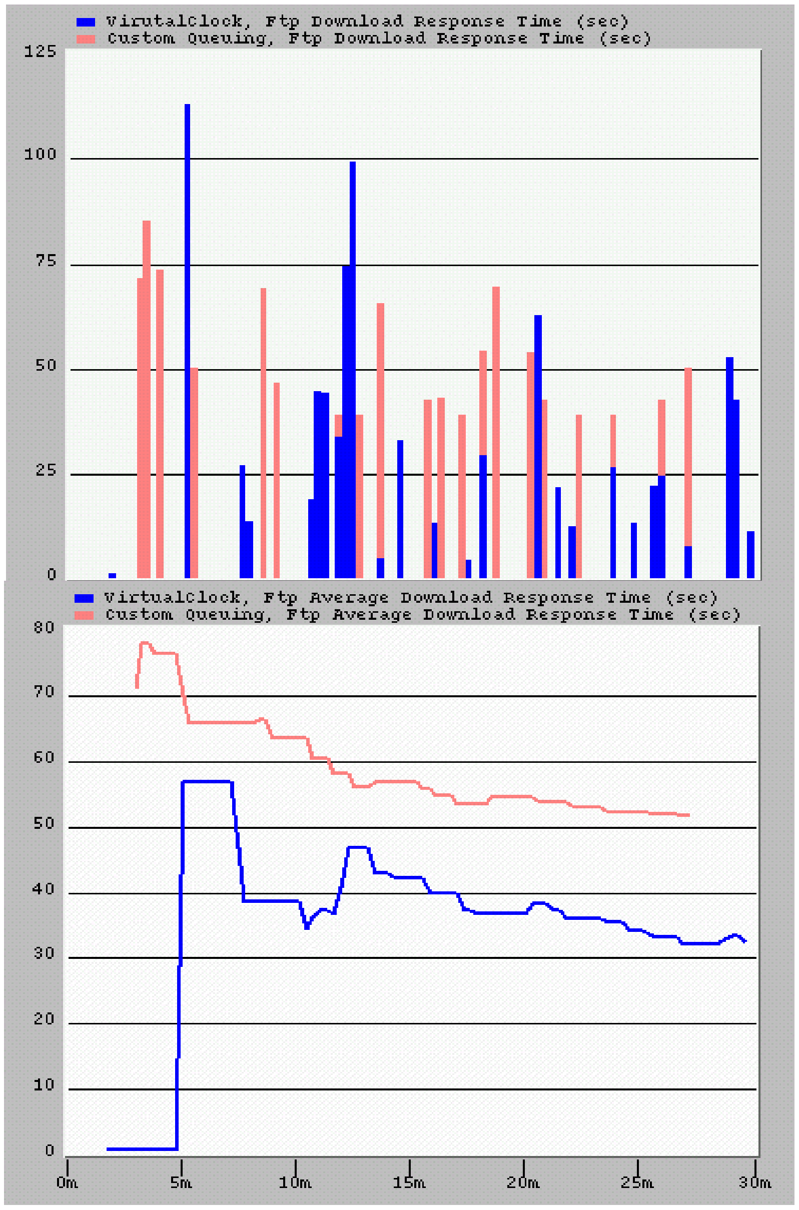


Fig 5.25 VirtualClock vs. Custom Queuing, FTP download response time (top) and average download response time (bottom) in (sec) vs. time (min).

The performance of voice is shown in Figs. 5.26 and 5.27, in terms of voice packet end-to-end delay and packet delay variation. It can be seen there that VirtualClock has a considerably lower value for both packet end-to-end delay and packet delay variation than Custom Queuing.

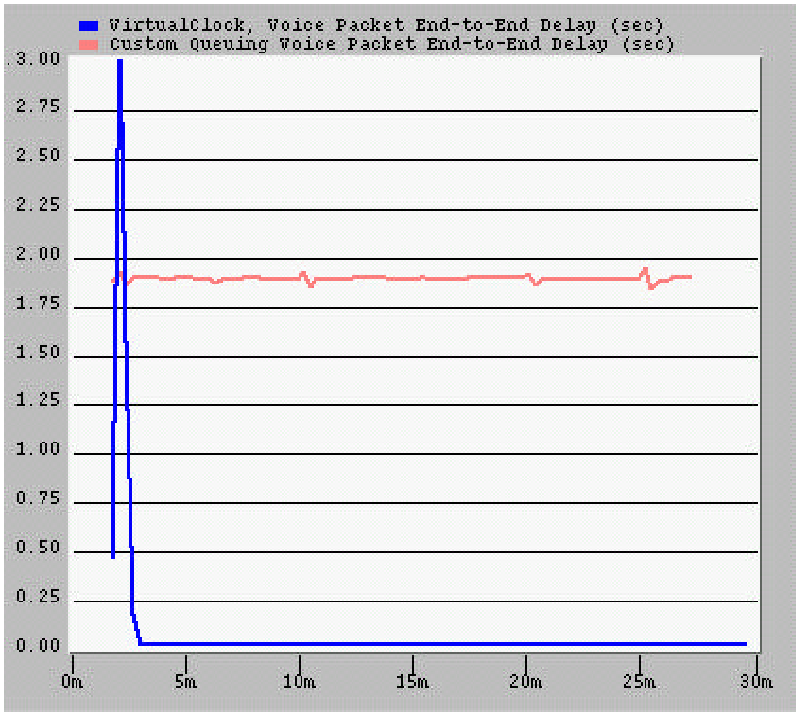


Fig. 5.26 VirtualClock vs. Custom Queuing, IP Telephony, voice packet end-to-end delay in (sec) vs. time (min).

The videoconferencing application packet end-to-end delay for VirtualClock and Custom Queuing is shown in Fig. 28. The figure shows that the end-to-end delay encountered by videoconferencing packets using Custom Queuing is almost half of that value using VirtualClock. Although Custom Queuing provides a poor performance for IP Telephony, it supplies a better-quality functionality for a videoconferencing application. Therefore, it can be concluded that a higher quality for videoconferencing is achieved by paying the price of gaining a worse quality for IP Telephony.

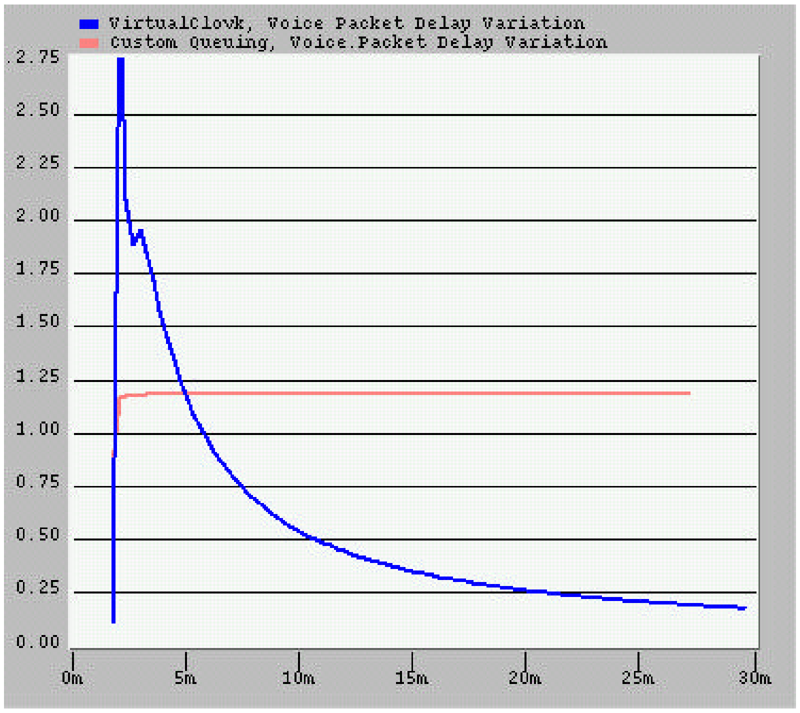


Fig 5.27 VirtualClock vs. Custom Queuing, IP Telephony, voice packet delay variation (sec) vs. time (min).

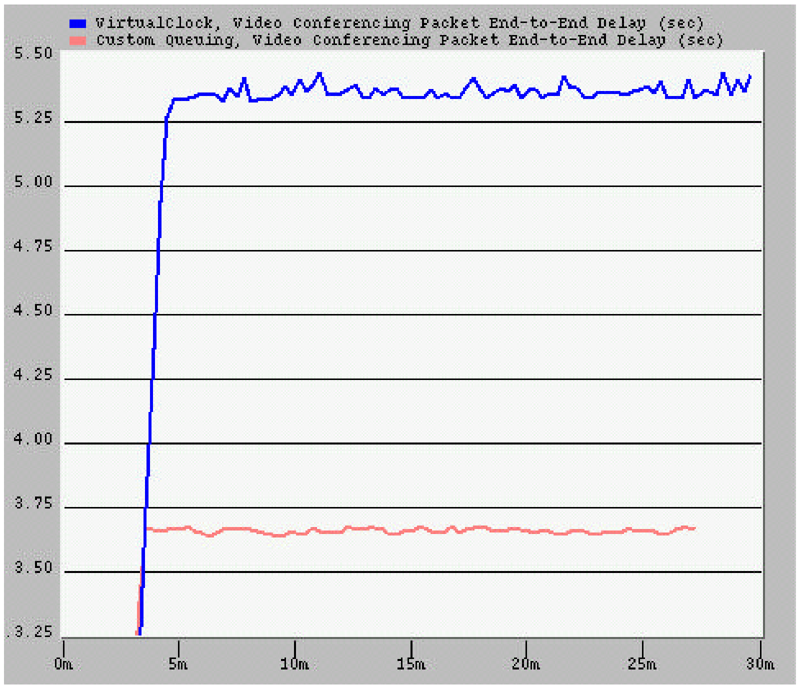


Fig 5.28 VirtualClock vs. Custom Queuing, Videoconferencing packet end-to-end delay in (sec) vs. time (min).

As seen in Figs.5.26, 5.27, and 5.28 VirtualClock and CQ are providing different end-to-end delay and delay jitter for IP Telephony and videoconferencing applications. Since voice and videoconferencing end-to-end delay provided by VirtualClock and WFQ are identical and WFQ is proven to be a fair algorithm, it can be concluded that CQ is not treating thesis applications fairly. As discussed earlier, CQ behaves unfair when traffic flows have variable packet sizes and average packet sizes of the flows are unpredictable.

In our simulation scenario, the average packet size of a flow entering a CQ queue is one of the main parameters for calculating the *Byte Count* value associated with that queue. Since voice and videoconferencing packet sizes are of different sizes, the solution we choose to estimate the average packet size of those flows is to monitor the incoming packets to thesis queues, collect the statistic for the size of the packets, plot the average value of these statistics in a graph, and choose the best guess for the average packet size. Thus, the chosen average packet size is not a precise value and explains the observed dissimilarities in the performance of voice and videoconferencing applications using VirtualClock and CQ. We also see that since the average packet size of the flows is not unknown for in our simulation scenario, CQ behaves unfair.

5.2.3 VirtualClock vs. Priority Queuing

In this section we look at the performance dissimilarities of the applications when employing VirtualClock and Priority Queuing at the output queues of the routers. Priority Queuing enables us to give absolute preferential service to high priority applications instead equally treating the applications according to their required bandwidth (VirtualClock). We allocate the traffic from FTP, voice, videoconferencing, and HTTP applications to High, Medium, Normal, and Low priority queues defined in section 2.5.2, respectively. In the real networks, both HTTP, and FTP are considered as applications with non-critical performance requirements. On the other hand, both IP Telephony and videoconferencing are known as mission critical applications. In this scenario since we are also interested in observing and comparing the effect of Priority Queuing in comparison to VirtualClock on low volume traffic such as HTTP and HTTP, we assign FTP to the highest priority queue. Otherwise the total link bandwidth would be consumed by high volume, mission critical applications (voice, and videoconferencing).

The queue allocation to the applications is defined in *PQ Profile* attribute of *IP QoS Configuration* object as follows:

- *Low Priority, Queue* $Size_0$ *= 500 (HTTP)*
- *Medium Priority, Queue* $Size_1$ *= 500 (Videoconferencing)*
- *Normal priority, Queue* $Size_2$ *= 500 (IP Telephony)*
- *High priority, Queue* $Size_3$ *= 500 (FTP).*

Because the HTTP application is assigned to the lowest priority queue, and the other three applications utilize all the bandwidth in the bottleneck link, no HTTP traffic can be transferred through this link. Thus, the HTTP traffic isn't serviced and will be starved in this network.

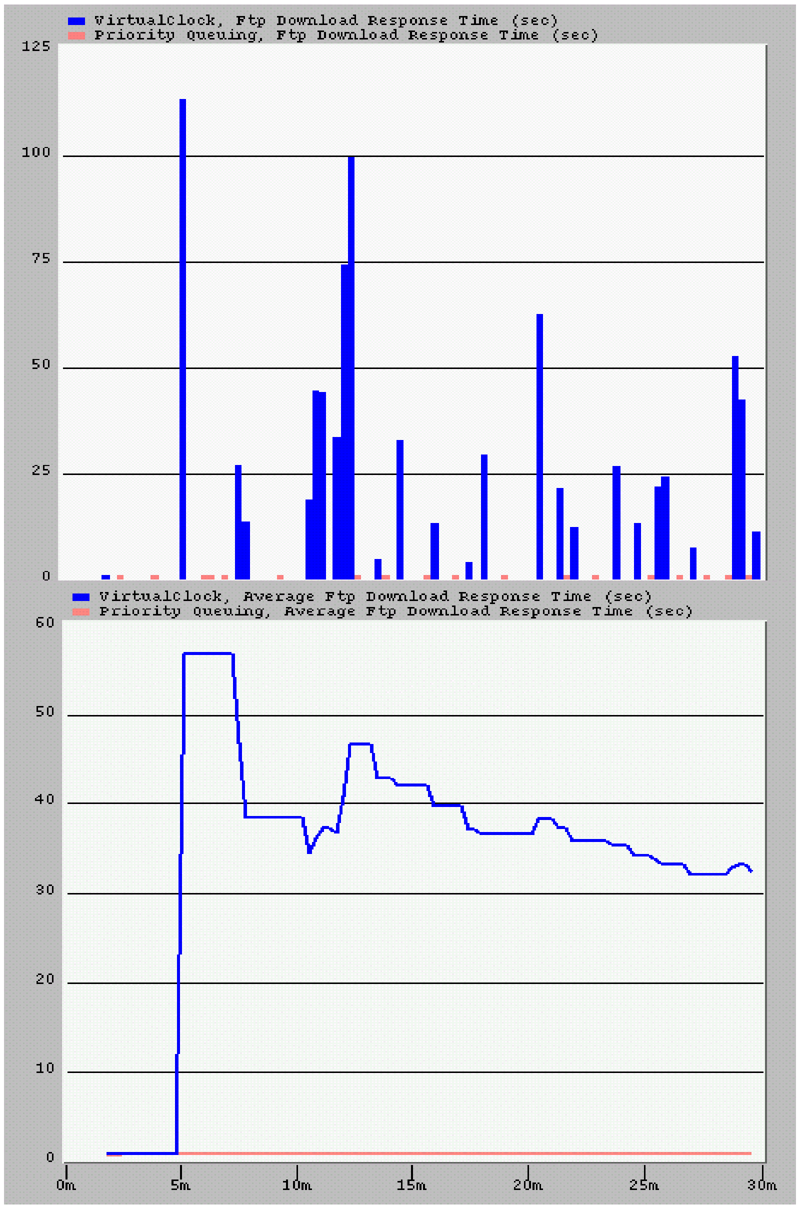


Fig 5.29 VirtualClock vs. Priority Queuing, FTP download response time (top) and average download response time (bottom) in (sec) vs. time (min).

Fig. 5.29 shows the FTP download response time and average download response time of VirtualClock vs. Priority Queuing. It can be verified that since the FTP has highest priority among the other applications, it is encountering a very low FTP download response time (average of 0.55 sec.) in contrast to the high file download time of the VirtualClock.

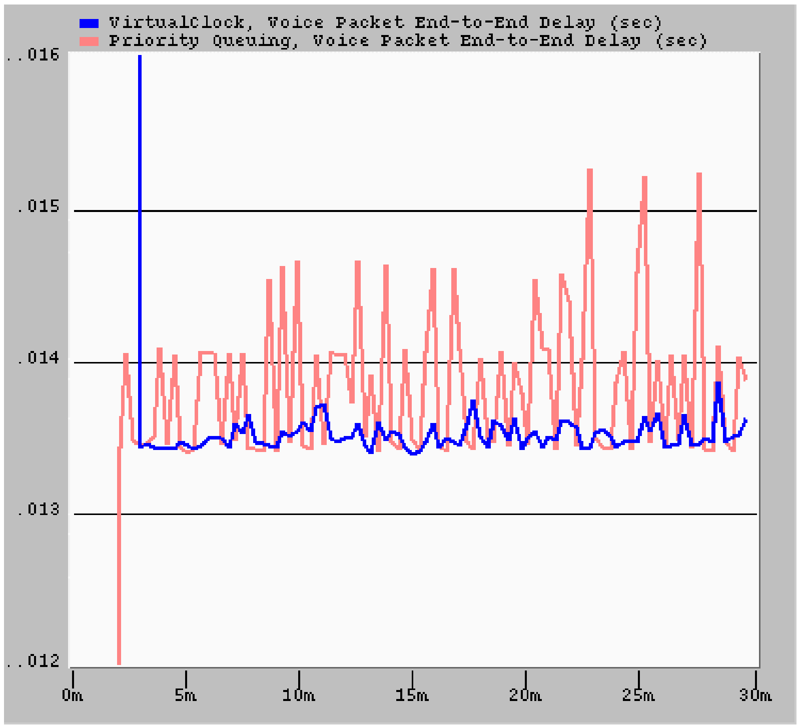


Fig. 5.30 VirtualClock vs. Priority Queuing, IP Telephony, voice packet end-to-end delay in (sec) vs. time (min).

Fig. 5.30 shows the end-to-end delay of voice packets using VirtualClock and PQ. It can be seen that the average delay value encountered by voice packets for both algorithms is very similar. However, the figure shows that delay variation, using Priority Queuing is higher than using VirtualClock.

Figure 5.31 shows the packet end-to-end delay of a videoconferencing application. It can be seen from the graphs that the two algorithms load the same end-to-end delay for videoconferencing packets. It can be concluded from the driven simulation results of Scenario 2 that VirtualClock behaves very similarly to WFQ in providing service to applications with various priorities and different performance requirements.

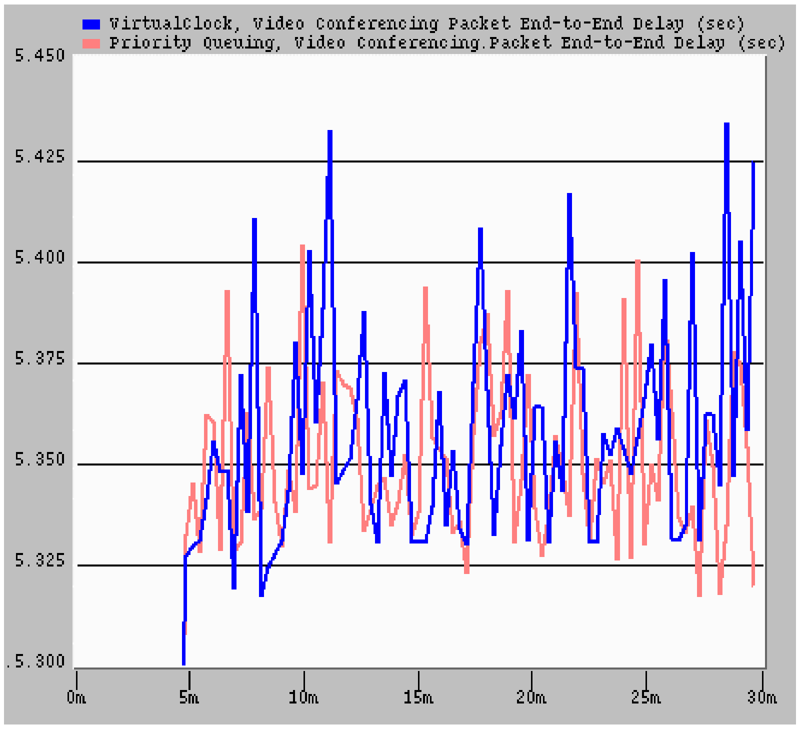


Fig 5.31 VirtualClock vs. Custom Queuing, Videoconferencing packet end-to-end delay in (sec) vs. time (min).

It can also be observed that VirtualClock provides similar service performances as Custom Queuing to the low volume applications (HTTP and FTP). However, it services the higher volume, more mission critical applications differently than Custom Queuing. VirtualClock provides better service for voice application in terms of packet end-to-end delay and delay variation, but poorer service for videoconferencing. It has to be taken into consideration that the performance of the algorithms is being evaluated by choosing the algorithm specific queue parameters, which leads to the same network bandwidth allocation to the queues.

Chapter 6
Conclusion

With more and more multimedia applications currently running on the Internet, Internet is expected to support a wide range of applications in the future. The applications with different QoS requirements in terms of bandwidth, delay, delay jitter, and traffic loss. Traffic scheduling in network switching nodes is used as a means of avoiding congestion in order to provide QoS to different traffic classes.

In this thesis we described the implementation of the VirtualClock scheduling mechanism employing the OPNET simulation tool. The algorithm is implemented in the output queues of the IP router node objects in OPNET. We show how we incorporate the algorithm in the IP layer of the router's network hierarchy so that it can communicate with the upper and lower network layers of this object. We verified the correctness and the functionality of the VirtualClock model by conducting simulations on two network scenarios. In the first network scenario, we measured the arrival and departure times of packets entering and leaving the router's queues. We verified that the measured values match the scheduled packet departure times by the VirtualClock scheduler. In the second network scenario, we examined the functionality of the model during the sources' conforming and nonconforming periods. We verified the algorithm by examining the conformance of the two main variables of the algorithm measured during simulation with their expected calculated values during both conforming and non-conforming traffic periods.

Finally we compared the performance of the VirtualClock algorithm with several other scheduling algorithms: WFQ, CQ, and PQ. These algorithms are used in current IP routers, which are manufactured by Cisco system Inc. and Juniper networks. The comparison is performed through two sets of simulation scenarios. Our driven simulation results from the first scenario showed that the VirtualClock algorithm performs closely to WFQ and CQ during both conforming and nonconforming periods. However, the VirtualClock algorithm behaves differently than PQ. VirtualClock allocates bandwidth fairly to traffic from different flows according to their specified traffic generation rates, but PQ provides preferential priority to selected traffic queues at the price of starvation of other low priority queues. We also compare the effect of VirtualClock with the above algorithms on the performance of four Internet applications: HTTP, FTP, IP Telephony, and Video Conferencing. It can be indicated from the results that applications with a predictable, consistent traffic generation rate like voice have a rich performance under the effect of VirtualClock and CQ and WFQ has a similar effect on the mentioned Internet applications. On the other hand, PQ has a better influence on the performance of unpredictable, mission critical applications that need to get preferential service over the other applications like videoconferencing.

References

[1] N. Alborz, M. Keyvani, M. Nikolic, and Lj. Trajkovic, “Simulation of packet data networks using OPNET,” *OPNETWORK ‘00*, Washington, DC, Aug. 2000.

[2] N. Alborz and Lj. Trajkovic, “Implementation of VirtualClock scheduling algorithm in OPNET,” *OPNETWORK ‘01*, Washington, DC, Aug. 2001.

[3] M. Andrews, “Probabilistic end-to-end delay bounds for Earliest Deadline First scheduling,” in *Proc. of INFOCOM ‘00*, Mar. 2000.

[4] G. Armitage, *Quality of Service in IP Networks: Foundation for a Multi-Service Internet.* Indianapolis, IN: Macmillan Technical Publishing, 2000.

[5] Cisco Systems, Inc. documentation on QoS: *http://www.cisco.com/warp/public/732/Tech/qos* (accessed Apr. 2002).

[6] Cisco Systems, Inc. documentation on Voice Over IP: *http://www.cisco.com/univercd/cc/td/doc/product/access/acs_mod/1700/1750/1750voip/intro.htm* (accessed Feb. 2002).

[7] A. Demers, S. Keshav, and S. Shenkar, “Analysis and simulation of a fair queuing algorithm,” in *Proc. of ACM SIGCOMM ‘89*, pp. 3-12, Sept. 1989.

[8] P. Fasano, “QoS for IP Telephony,” Presentation at ISIT'99, Vancouver, June 1999.

[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext Transfer Protocol (HTTP/1.1),” RFC 2616, June. 1999: *http://www.faqs.org/rfcs/rfc2616.html* (accessed Feb. 2002).

[10] N. Giroux and S. Ganti, *QoS in ATM Networks: State-of-the-Art Traffic Management.* Upper Saddle River, NJ: Prentice-Hall Inc., 1999.

[11] S. J. Golestani, “A Self-Clocked fair queuing scheme for broadband applications,” in *Proc. of IEEE INFOCOM ‘94*, pp. 636-646, Apr. 1994.

[12] R. Guering and V. Peris, “Quality-of-Service in packet networks: basic mechanisms and directions,” Computer Networks, vol. 31, no. 3, pp. 169-189, Feb. 1999.

[13] V. Jacobson, “Congestion avoidance and control,” in *Proc. of the ACM SIGCOMM ‘98*, pp. 314-329, Aug. 1998.

[14] R. Jain, "Congestion control in computer networks: issues and trends," *IEEE Network*, vol. 4, no. 3, pp. 24-30, May 1990.

[15] M. Z. Jiang, "Analysis of Wireless Data Network Traffic," M.A.Sc. Thesis, Engineering Science Department, Simon Fraser University, Apr. 2000.

[16] Juniper Networks, solutions and technology application note, Applying Class of Service: *http://www.juniper.net/techcenter/app_note/350004.html* (accessed Feb. 2002).

[17] Juniper Networks, products and services datasheet, FPCs for the M160 Router: *http://www.juniper.net/products/dsheet/pdfs/100042.pdf* (accessed Mar. 2001).

[18] S. Keshav, *An Engineering Approach to Computer Networking.* Reading, MA: Addison Wesley, 1998.

[19] W. Leland. M. Taqqu, W. Willinger, and D. Wilson, "On the self-similarity of Ethernet traffic (extended version)," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1-15, Feb. 1994.

[20] Nortel Networks, technology overview series for small and medium businesses, IP Telephony Basics: *http://www.nortelnetworks.com/solutions/smb/nbrc/collateral/ip_tel_basics.pdf* (accessed Feb. 2002).

[21] A. Oodan, K. E. Ward, and A.W. Mullee, *Quality of Service in Telecommunications*. London: The Institution of Electrical Engineers, 1997.

[22] OPNET Technologies, Inc., Washington DC, OPNET documentation V.7.0.L.

[23] OPNET Technologies, Inc., Washington DC, OPNET documentations on Configuring Applications and Profiles, The Custom Application Model, Standard Network Applications, and Simulation Methodology for the Analysis of QoS, July 2000.

[24] OPNET Technologies, Inc., Washington DC, OPNET documentation on RPG model description, Feb. 2001.

[25] A. K. Parekh and R. G. Gallager, "Generalized processor sharing approach to flow control in integrated services networks: the single node case", *IEEE/ACM Transactions on Networking,* vol. 1, no. 3, pp. 344-357, June 1993.

[26] A. K. Parekh and R. G. Gallager, "Generalized processor sharing approach to flow control in integrated services networks: the multiple node case", *IEEE/ACM Transactions on Networking,* vol. 2, no. 2, pp. 137-150, Apr. 1994.

[27] J. Postel and J. Reynolds, “File Transfer Protocol (FTP),” RFC 959, Oct. 1985: *http://www.faqs.org/rfcs/rfc959.html* (accessed Feb. 2002).

[28] QoSforum.com, IP QoS FAQ: *http://www.qosforum.com/docs/faq* (accessed Dec. 2001).

[29] F. Risso, “Quality of Service on packet switched networks,” Ph.D Thesis, Dept. of Computer Science, Torino Polytechnic, Jan. 2000.

[30] B. Ryo, “A tutorial on fractal traffic generators in OPNET for Internet simulation,” *OPNETWORK ‘00*, Washington DC, Aug. 2000.

[31] M. Shreedhar and G. Varghese, “Efficient fair queuing using Deficit Round Robin,” *IEEE/ACM Transitions on Ne*tworking, vol. 4, no. 3, pp. 375- 385, June. 1996.

[32] S. Suri, G. Varghese, and G. Chandranmenon, “Leap Forward Virtual Clock: A new fair queuing scheme with guaranteed delays and throughput fairness,” in *Proc. of IEEE INFOCOM ‘97*, pp. 557-562, Apr. 1997.

[33] A. S. Tanenbum, *Computer Networks, Third edition*. Upper Saddle River, NJ: Prentice-Hall Inc., 1996, pp. 374-396.

[34] J. Walrand and P. Varaiya, *High-performance Communication Networks, Second edition*. San Francisco, CA: Morgan Kaufman Publishers Inc., 2000, pp. 26-32 and pp. 261-293.

[35] X. Xiao and L.M. Ni, “Internet QoS: a big picture,” *IEEE Network*, vol. 13, no. 2, pp. 8-18, Mar. 1999.

[36] G. G. Xie and S. Lam, “Delay guarantee of VirtualClock server,” *IEEE/ACM Transitions on Networking*, vol. 3, no. 6, pp. 683-689, Dec. 1995.

[37] H. Zhang, “Service disciplines for guaranteed performance service in packet-switching networks,” in *Proc. of the IEEE*, vol. 83, no. 10, Oct. 1995.

[38] L. Zhang, “VirtualClock: a new traffic control algorithm for packet switching networks,” in *Proc. of ACM SIGCOM ‘90*, Sept. 1990.