Implementation of VirtualClock Scheduling Algorithm in OPNET

Nazy Alborz and Ljiljana Trajkovic
School of Engineering Science
Simon Fraser University
Vancouver, British Columbia
Canada V5A 1S6
E-mail: {nalborz, ljilja}@cs.sfu.ca
http://www.ensc.sfu.ca/research/cnl

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. Packet scheduling in routers can provide guaranteed performance in terms of delay, delay jitter, packet loss, and throughput.

In this paper, we describe the OPNET model of an IP router with a scheduling algorithm called VirtualClock. The VirtualClock algorithm monitors the average transmission rates of packet data flows. It also provides each flow with a guaranteed throughput and a low queuing delay. We have incorporated the VirtualClock algorithm into the OPNET process model *ip_output_iface*. This process model executes scheduling algorithms in the network layer of IP objects.

1 Introduction

One of the most significant promises in today's integrated services packet-switched networks (such as Internet) is providing Quality of Service (QoS) guarantees [1]. These networks are able to integrate applications with a wide range of traffic characteristics. These applications range from video conferencing with stringent QoS requirements, to best effort applications with no required guarantees. In network switching nodes, packets with different QoS requirements interact with one another when they are multiplexed at the same output port. If there is no control over these interactions, they will degrade the performance of the network [2].

Packet scheduling algorithms in network routers and switches can provide guaranteed QoS. Scheduling algorithms not only allow packets from various traffic streams to be statistically multiplexed, but also provide a protection between these streams. The three main functions of packet scheduling disciplines are to determine:

- which packets among different service classes get transmitted
- when these packets get transmitted, and
- which packets get discarded in case of an overflow in switch buffers.

These operations affect throughput, delay, and loss rate performance parameters.

Packet scheduling mechanisms are classified into two categories: work-conserving and non-work-conserving [3]. 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 time when the next packet is eligible.

In this paper we describe the implementation of a work-conserving scheduling algorithm called VirtualClock [4]. The algorithm is similar to the Weighted Fair Queuing (WFQ) algorithm [1] that is currently implemented in OPNET [5]. The difference between the WFQ and VirtualClock algorithms is that the VirtualClock simplifies the calculation of finish times. Finish time is calculated and used by the scheduler when choosing the next packet to be dequeued [6]. Other advantages of the VirtualClock algorithm are:

- it provides per connection bandwidth allocation
- it guarantees protection between traffic flows.

This paper is organized as follows. In Section 2, we introduce the VirtualClock [4] scheduling algorithm, its role, and its functionality. In Section 3, we describe the implementation of the algorithm in the IP layer process module of an IP-router node model. The validity of the model through OPNET simulations of a simple network is discussed in Section 4. Section 5 presents the simulation results from a network with various traffic sources, which employs VirtualClock algorithm. We conclude with Section 6.

2 VirtualClock Algorithm

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 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 the case that there is more than one packet waiting [4].

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). The *AI* value should be chosen as: total transmitted data / *AR*. Its range is: $1/AR \leq AI \leq$ total flow duration.

2.1 Inside the VirtualClock Algorithm

The algorithm uses two control variables for each flow, Virtual Clock *(VC)* and auxiliary Virtual Clock *(auxVC)* [4].

The following two functions are performed by 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 [4].

By introducing a second parameter called 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 usage 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 the flows use the bandwidth according to their specified packet arrival rates. Thus, although nonconforming flows can use free bandwidth, they cannot affect conforming flows.

3 VirtualClock Implementation

In this section we describe our implementation of the data forwarding function of a VirtualClock algorithm in an IP router. The algorithm is implemented in the *ip_output_iface* process model, which is a child process of the IP layer process model *ip_rte_v4*.

The *ip_output_iface* process 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: FIFO, Weighted Fair Queuing, Custom Queuing, and Priority Queuing [5].

In our implementation, we have used the modified version of the external files: *oms_qm.c* and *ip_qos_support.c*, *ICI* file: *ip_arp_v4. ici*, and header file: *oms_qm.h.*

3.1 VirtualClock Process Model

We expanded and modified the state transition diagram of the already existing OPNET process model *ip_output_iface.* The state transition diagram of the VirtualClock algorithm consists of four states: *init, enqueue, dequeue, and idle*, as shown in Figure 1.

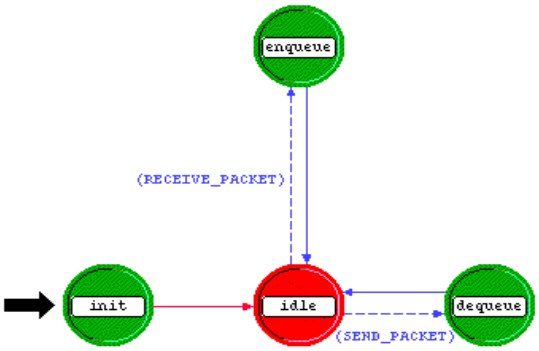

Figure 1: State transition diagram of the VirtualClock algorithm.

3.1.1 *enqueue* State

When a packet arrives from an upper layer process, it enters the *enque*ue state. The *enqueue* state functions as follows:

Step 1. Get the incoming packet.
Step 2. Determine the queue to which the packet belongs according to the flow of the incoming packet. 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_Virtual_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.

3.1.2 *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 that there is more than one packet with 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 (in bits) by the link rate (in bits/sec).

3.2 QoS Configuration Object

A new profile (*VC Profile*) for the VirtualClock algorithm was also added in *qos_attribute_definer* process model of OPNET's *QoS Configuration Object* [7]. This enables users to choose the suitable scheduling mechanism.

Users can define a queuing profile with an optional number of queues, and can configure the queue attributes: *Arrival Rate* (expected packet arrival rate of the flow entering the queue), *Maximum Queue Size*, and *Classification Scheme* (a scheme for categorizing packets to flows and assigning them to queues).

4 Model Verification

In order to evaluate the performance of the VirtualClock model, we have created the simple network model.

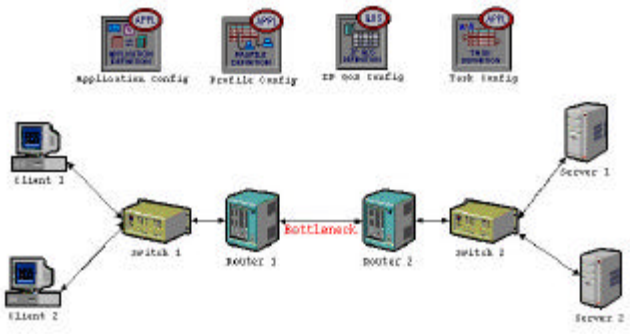

Figure 2: Simple network model for performance verification of the VirtualClock algorithm.

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 bytes 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, which was 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.

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.*

Incoming packets from Client 1 are recognized as a flow and are assigned to the first queue (Q0). Packets from the second flow, coming from Client 2, are assigned to the second queue (Q1). The following graphs are obtained as outputs of OPNET simulations. Incoming traffic to queues Q0 and Q1 is shown in Figure 3.

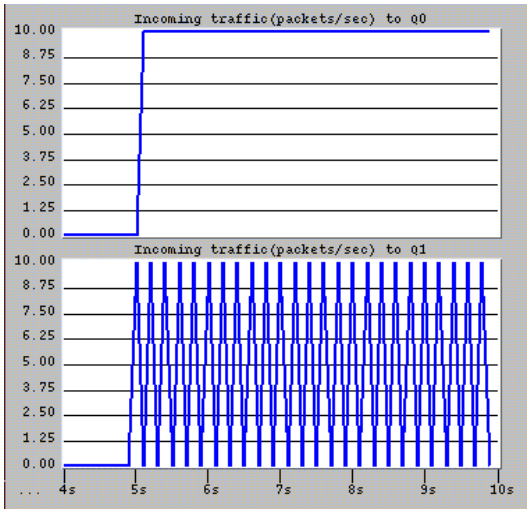

Figure 3: Incoming traffic to queues: Q0 (top) and Q1 (bottom), in (packets/sec) vs. time.

Figure 4 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*.

Figure 5 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 5.

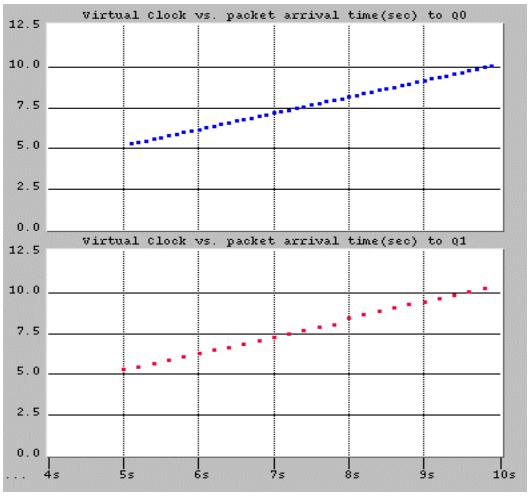

Figure 4: Virtual Clock (*VC*) stamp vs. packet arrival time (sec) for queues: Q0 (top) and Q1 (bottom).

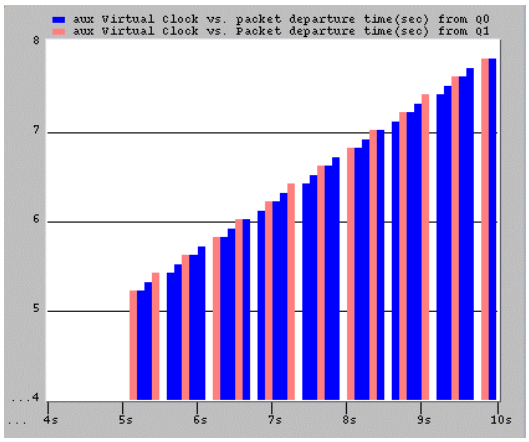

Figure 5: Auxiliary Virtual Clock (*auxVC*) vs. packet departure time (sec) for queues Q0 and Q1.

The outgoing traffic from queues Q0 and Q1, scheduled by the VirtualClock algorithm, is shown in Figure 6.

5 Simulation Experiments

In order to evaluate the performance of the VirtualClock algorithm we conducted simulations using OPNET version 7.0.B.

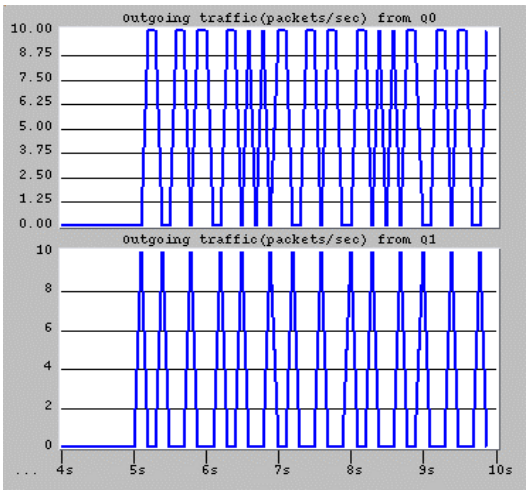

Figure 6: Outgoing traffic from queues: Q0 (top) and Q1 (bottom), in (packets/sec) vs. time.

In this scenario, we consider two conforming and one nonconforming sources. The first conforming source has constant packet generation rate. The second conforming source 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 Figure 2, 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 4 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*$_2$ *= 4, Queue Size*$_2$ *= 500.*

Client 1 starts sending packets at time 20 sec. It generates 1,024 bytes IP packets 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 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 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 [8]. RPG is a traffic source model that is used to generate self-similar traffic [9]. 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* = 4 (packets / sec)
- *Hurst parameter* = 0.9
- *Fractal onset time scale* = 0.1.

Figures 7 shows the incoming traffic from Clients 1, 2, and 3 to Router 1's queues Q0, Q1, and Q3, respectively. Figures 8 shows 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: 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 the Figure 8 (top), flow$_0$ adheres to its specified packet arrival rate. Thus, the slope of the Virtual Clock graph is 1. Figure 8 (middle) shows that the slope of the graph changes at instances when packet arrival rate of the flow changes. For the periods during which traffic has 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. Figure 8 (bottom) shows that for a flow fluctuating around its specified arrival rate, the Virtual Clock graph is close to a line with slope 1.

Figure 9 shows 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.

By comparing Figures 8 and 9, we see 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 Figures 8 and 9 (bottom) from 142.5 sec to 392 sec.

Figure 10 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.

Figure 11 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.

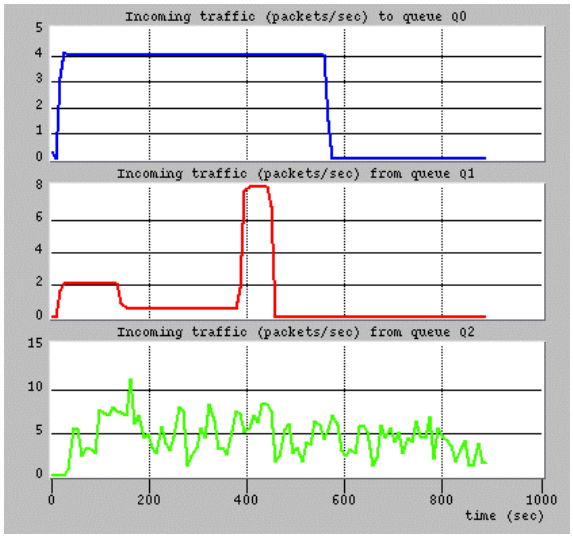

Figure 7: Incoming traffic to queues: Q0 (top), Q1 (middle), and Q2 (bottom) in (packets/sec) vs. time.

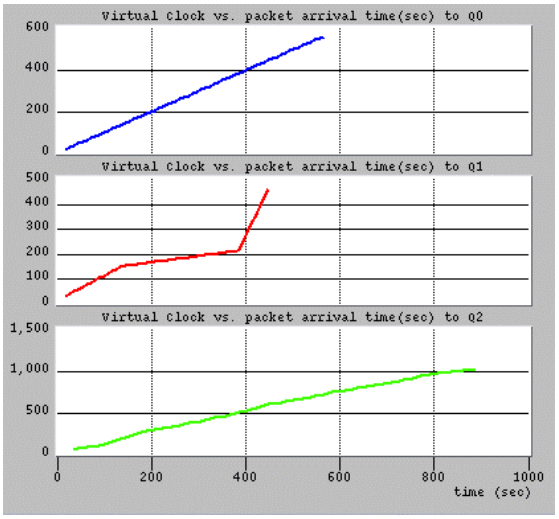

Figure 8: Virtual Clock (*VC*) stamp vs. packet arrival time (sec) for queues: Q0 (top), Q1 (middle), and Q2 (bottom).

It can be seen from Figures 7 and 11 (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 Figures 7 and 11 (middle), traffic from Client 2 is serviced with its arrival rate (2 packets/sec) while it conforms to its specified packet rate (until 142.5 sec). When 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.

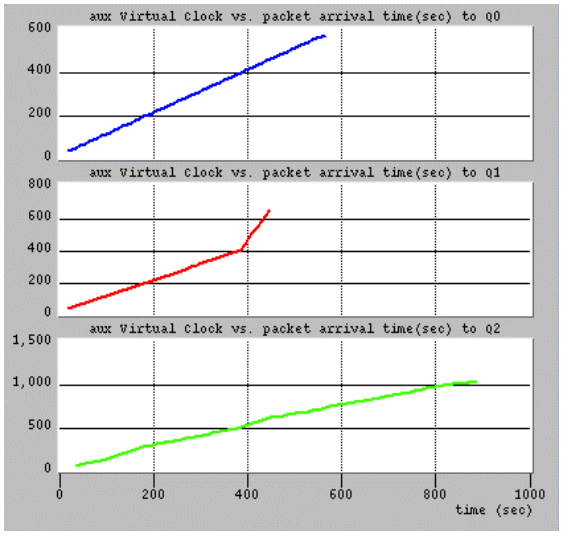

Figure 9: Auxiliary Virtual Clock (*auxVC*) stamp vs. packet arrival time (sec) for queues: Q0 (top), Q1 (middle), and Q2 (bottom).

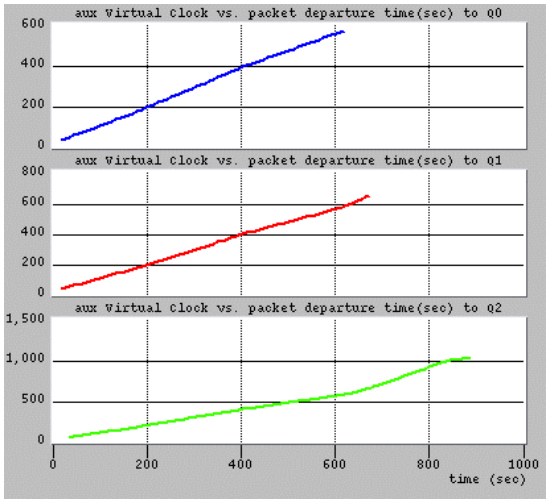

Figure 10: Auxiliary Virtual Clock (*auxVC*) stamp vs. packet departure time (sec) for queues: Q0 (top), Q1 (middle), and Q2 (bottom).

As seen in Figures 7 and 11 (bottom), Client 3 generates self-similar traffic. The Virtual Clock algorithm forwards packets at rate of 4 packets/sec, which is the average traffic generation rate of the source.

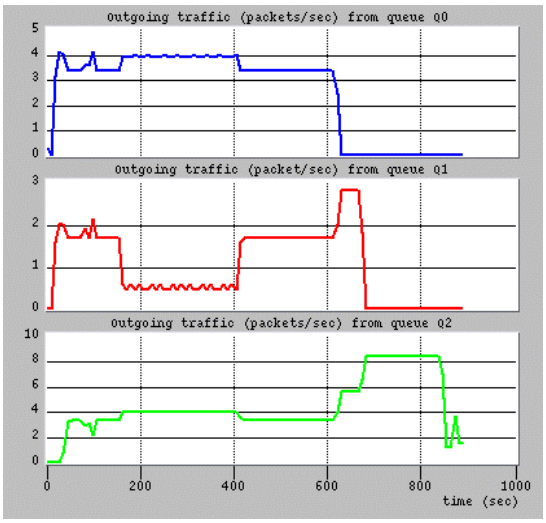

Figure 11: Outgoing traffic from queues: Q0 (top), Q1 (middle), and Q2 (bottom) in (packets/sec) vs. time.

We have compared the functionality of the VC algorithm with three algorithms already implemented in OPNET: WFQ, PQ, and CQ. We used the same scenario as in Section 5 and we repeated simulation experiment while employing different scheduling algorithms in Router 1. The outgoing traffic from queues Q1, Q2, and Q3 is shown in Figure 12. The graphs show the OPNET simulation results from VC, WFQ, PQ, and CQ algorithms.

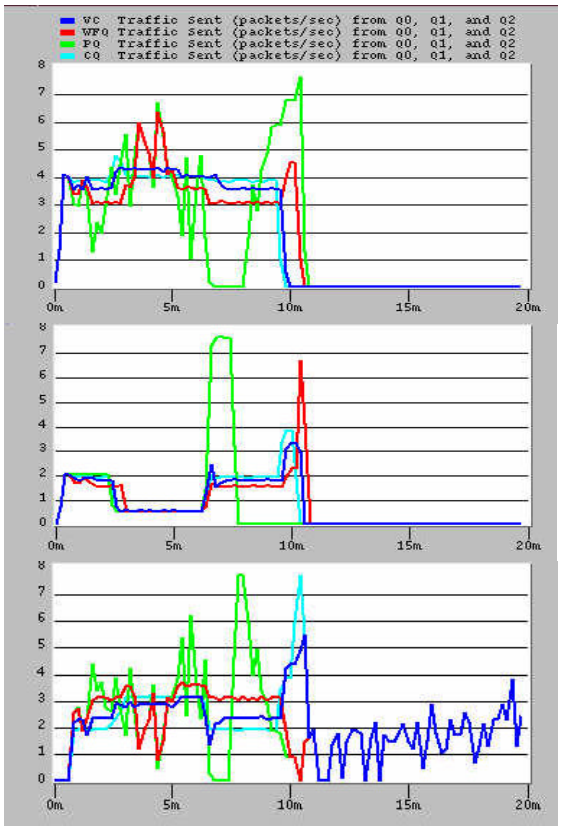

Figure 12: Outgoing traffic after being scheduled by scheduling algorithms: VC, WFQ, PQ, and CQ. Shown are queues: Q0 (top), Q1 (middle), and Q2 (bottom) in (packets/sec) vs. time.

6 Conclusion

In this paper we described the OPNET implementation of the data forwarding function of the VirtualClock scheduling algorithm. We described the modification in the OPNET's *ip_output_iface* scheduling process model used in the IP layer of all IP objects. We verified the correctness of the algorithm by simulating a simple network. We also used more complex simulation scenarios to illustrate the functionality of the algorithm.

Acknowledgment

The authors acknowledge with thanks the support of OPNET Technologies, Inc. This research was funded by the Canadian Foundation for Innovation Grant 910-99.

References

[1] Cisco Systems, Inc. documentation on QoS: http://www.cisco.com/warp/public/732/Tech/quality.shtml.

[2] J. Walrand and P. Varaiya, *High-performance Communication Networks, Second edition*. San Francisco, CA: Morgan Kaufman Publishers Inc., 2000, pp. 369 - 372.

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

[4] L. Zhang, "VirtualClock: a new traffic control algorithm for packet switching networks," in *Proc. ACM SIGCOM*, Sept. 1990.

[5] 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.

[6] S. Keshav, *An Engineering Approach to Computer Networking*. Reading, MA: Addison Wesley, 1998, pp. 209 - 263.

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

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

[9] B. Ryo, "A tutorial on fractal traffic generators in OPNET for Internet simulation," *OPNETWORK2000*, Washington DC, Aug. 2000.