ENSC 835-3: NETWORK PROTOCOLS AND PERFORMANCE
CMPT 885-3: SPECIAL TOPICS: HIGH-PERFORMANCE NETWORKS

Smart Queuing: An Adaptive Approach

Spring 2002

FINAL PROJECT

Tedi Susanto (tsusanto@cs.sfu.ca)
Jason Sze (jszea@cs.sfu.ca)
www.sfu.ca/~tsusanto/885
www.sfu.ca/~jszea/885

Abstract

Congestion control and Quality of Service (QoS) provision are important issues in today's high-speed networks. Packet scheduling can provide users with different QoS as well as ensure that the network is running efficiently. There are many packet scheduling or queuing algorithms, each has its own advantages and disadvantages. We investigated the performance of several different queuing mechanisms (FIFO, PQ, WFQ, CQ) using OPNET network simulation tool. A simple set of traffic parameters is used to determine which mechanism will optimize the network performance in terms of delay. From this characterization, we introduced a *smart* queuing mechanism, one that adapts to the current traffic situation by dynamically changing between the different algorithms.

1 Introduction

The explosive growth of the Internet has caused increasing demand on the packet-switched networks. Among many of the issues that have to be addressed, congestion control is of primary importance. An optimal congestion control mechanism would ensure that the network is running at its fullest capacity, efficiently regulate the flow of traffic, as well as provide the promised Quality of Service (QoS) to its users. This is not an easy task, especially given the constantly changing and chaotic [6] nature of the Internet. The existence of multiple users with different QoS and access rates, who are running different applications, such as email, file transfer, and video conferencing, leads to one dynamic and complex system. There are three ways to provide QoS and congestion control [1]:

- end-to-end mechanisms, such as call acceptance control, that operate at the two ends of a connection,
- edge mechanisms, such as shaping and policing, that operate at user-network interface, and
- core mechanisms, such as buffering, queue management, and scheduling, that operate at network switching nodes like routers and switches.

In this project, we investigated several packet scheduling/queuing algorithms that are widely used in network switching nodes to provide QoS. In addition, these algorithms also allow for statistically multiplexing packets from various traffic streams, and provide protection between streams. The three main functions of packet scheduling are to determine: (1) which packets get transmitted, (2) when these packets get transmitted, and (3) which packet get discarded in case of buffer overflow [1]. The performance of these algorithms can be measured using QoS parameters such as throughput, delay, delay jitter, and loss rate.

The OPNET model library provides several queuing mechanisms that are commonly used in network routers and switches. These include First In First Out (FIFO), Priority Queuing (PQ), Weighted Fair Queuing (WFQ), and Custom Queuing (CQ). These algorithms attempt to strike a balance between complexity and fairness, and there is no

one single standard as to which one is the most optimal. Some manufacturers implement more than one mechanism [3] to allow network operators to select one that is most appropriate. However, sometimes selecting which one to use may not be that simple. Each one has its own benefits and limitations, and the best one can be highly dependent on the current traffic flow and network condition.

Our project is to simulate a *smart* queuing algorithm in OPNET. This mechanism will constantly sample a set of parameters relating to the current traffic condition and switch to the best algorithm depending on the situation. This adaptive approach ensures that the routers operate at the optimal point even when the network condition changes.

This report is organized as follow. Section 2 provides background information on the different queuing algorithms. Section 3 describes the simulation design, and section 4 gives the results and discussion. Conclusion is provided in Section 5.

2 Background

There are many different queuing mechanisms used in IP routers. Some of the most common types of queuing mechanisms employed are:

i) First-in-first-out (FIFO)
ii) Priority queuing (PQ)
iii) Weighted fair queuing (WFQ)
iv) Custom queuing (CQ)

First-in-first-out (FIFO) is the most simplistic type of queuing and is synonymous to first-come-first-serve (FCFS). All incoming packets are placed in a single queue and are served in the same order as they were received. This type of queuing requires very little computation and its behaviour is very predictable (i.e. packet delay is a direct function of the size of the FIFO queue). However, due to its simplistic nature, there are many undesirable properties related to this queuing method. Since all packets are inserted into the same queue, it is impossible to offer different services for different packet classes. Also, if an incoming flow suddenly becomes bursty, then it is possible for the entire buffer space to be filled by this single flow and other flows will not be serviced until the buffer is emptied. Figure 1 illustrates this mechanism.

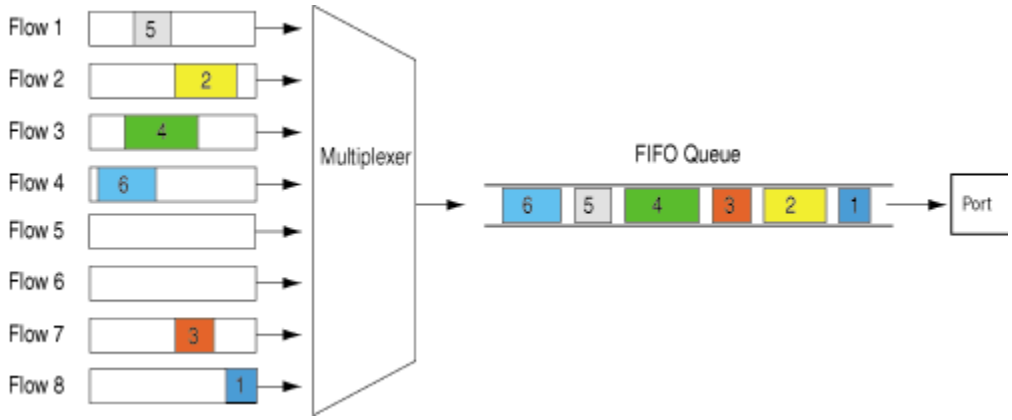

Figure 1. FIFO queue [3]

Priority queuing (PQ), as shown in Figure 2, provides a simple way of offering different services to different classes of packets. Its operation involves classifying each incoming packet into different priorities and placing them into separate queues accordingly. Packets of higher priority are transmitted on the output port before lower ones. This is a great way of providing differentiated service, but it has some shortcomings. An example is if there is a large continuous flow of high priority traffic into the queue, then low priority packets will experience excessive delay, and perhaps even to the extent of service starvation.

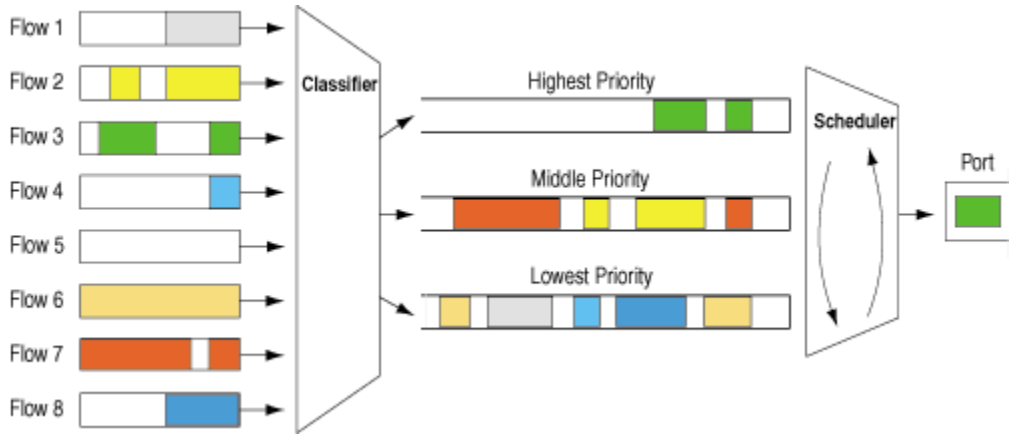

Figure 2. Priority queuing [3]

Fair queuing (FQ) is a class of queuing mechanism with the purpose of allowing fair access for each incoming flow and to prevent a bursty flow from consuming all of the output bandwidth. FQ contains a queue for each distinct flow and packets from each flow are inserted into its respective queue. The system then services each queue one packet at a time in a round-robin fashion. Weighted fair queuing (WFQ) is a variation of Fair Queuing (FQ) in that it supports flows with different bandwidth requirements. It does this by assigning each queue with different weights that corresponds to the proportion of the allocated output bandwidth. In WFQ, as described in [3], each incoming packet is time stamped with a finish time in addition to being placed into its corresponding flow queue . Unlike FQ, selection of which packet to be serviced is now

based on this time stamp on each packet. Packets are serviced by examining their finish times and ones with earlier finish times are transmitted before later ones. It is possible for a later packet to have a finish time stamp that is smaller than an earlier packet. Currently, WFQ is only implemented in the software level, so application in high-speed routers is limited. In addition, depending on the implementation platform, WFQ has been shown to perform unfairly [5]. Figure 3 illustrates the WFQ mechanism.

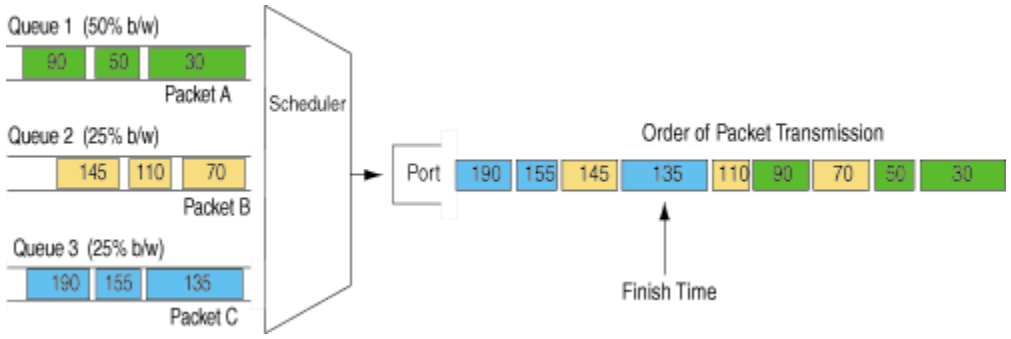

Figure 3. WFQ [3]

Lastly, Custom Queuing (CQ) (a.k.a. Class-based Queuing) is designed to address the limitations present in PQ and WFQ. In CQ, as described in [3], each packet is classified as belonging to a particular service class and is placed in the queue for that class. Each service class is assigned a weight that corresponds to the percentage of the output bandwidth allocated to it. Packets from each queue are transmitted based on the weight assigned to their queues. One benefit of CQ is that it can be implemented in hardware. It provides differentiated service, as well as guaranteed output bandwidth for each service class (even for low-priority traffic). This method is shown in Figure 4.

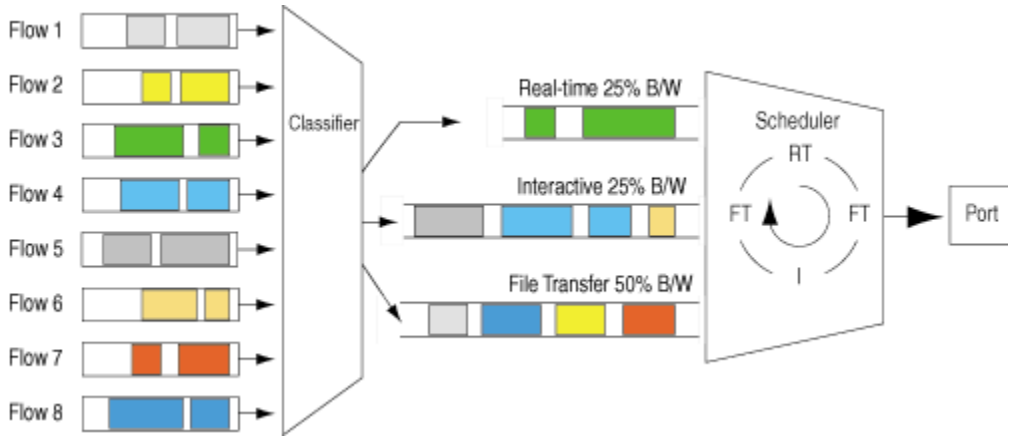

Figure 4. Custom queuing [3]

3 Simulation Design

3.1 Smart Queuing

The idea behind smart queuing is to take advantage of all the benefits offered by different queuing mechanisms but at the same time to avoid their limitations. There are cases where one queuing schemes perform better than others and vice versa. And as the traffic condition changes, the mechanism that was originally optimal may no longer be the best. Smart queuing addresses this change in traffic by dynamically switching to the best queuing at that time. Of course the *best* could be highly subjective, dependent on the network operator's policies and preferences. However, in our project we have adopted best in terms of network performance such as fairness in accessing the network resources given that users have different QoS and packet delay.

In this project, the set of parameters that we maintained to characterize the current traffic pattern are the number of users (source addresses), the Type of Service (ToS) associated with each packet, and the transfer rates. In our simulation, we have assumed that each user has only one connection with one ToS and are sending packets at a constant rate. We use this information together with a set of simple rules to determine which queuing is the best, in terms of fairness and delay. It should also be noted that the advantages of smart queuing can only be fully realized when all the routers in the packet path implement this mechanism.

3.2 Smart Router Implementation

The OPNET model library provides standard router model which can be configured to use FIFO, PQ, WFQ, and CQ. Within this router is an *ip* node representing the IP layer which implements the *ip_dispatch* process model. This process model implements IP routing functions, and fragmentation and reassembly. It routes IP packets arriving on any interface to the appropriate output interface based on their destination address, using either dynamic routing protocol such as RIP or OSPF, or static routing tables [7]. If the router is configured to use a queuing scheme, this process spawns a child process *ip_output_iface* for each interface that uses a queuing scheme. Note that a router may use different queuing schemes for different interfaces/ports. The *ip_output_iface* process creates a specific queue management structure *qm_info* which enqueue and dequeue packets according the queuing mechanism selected by the user. This process also generates statistics for the interface during simulation. Figure 5 summarizes this hierarchy as implemented in OPNET.

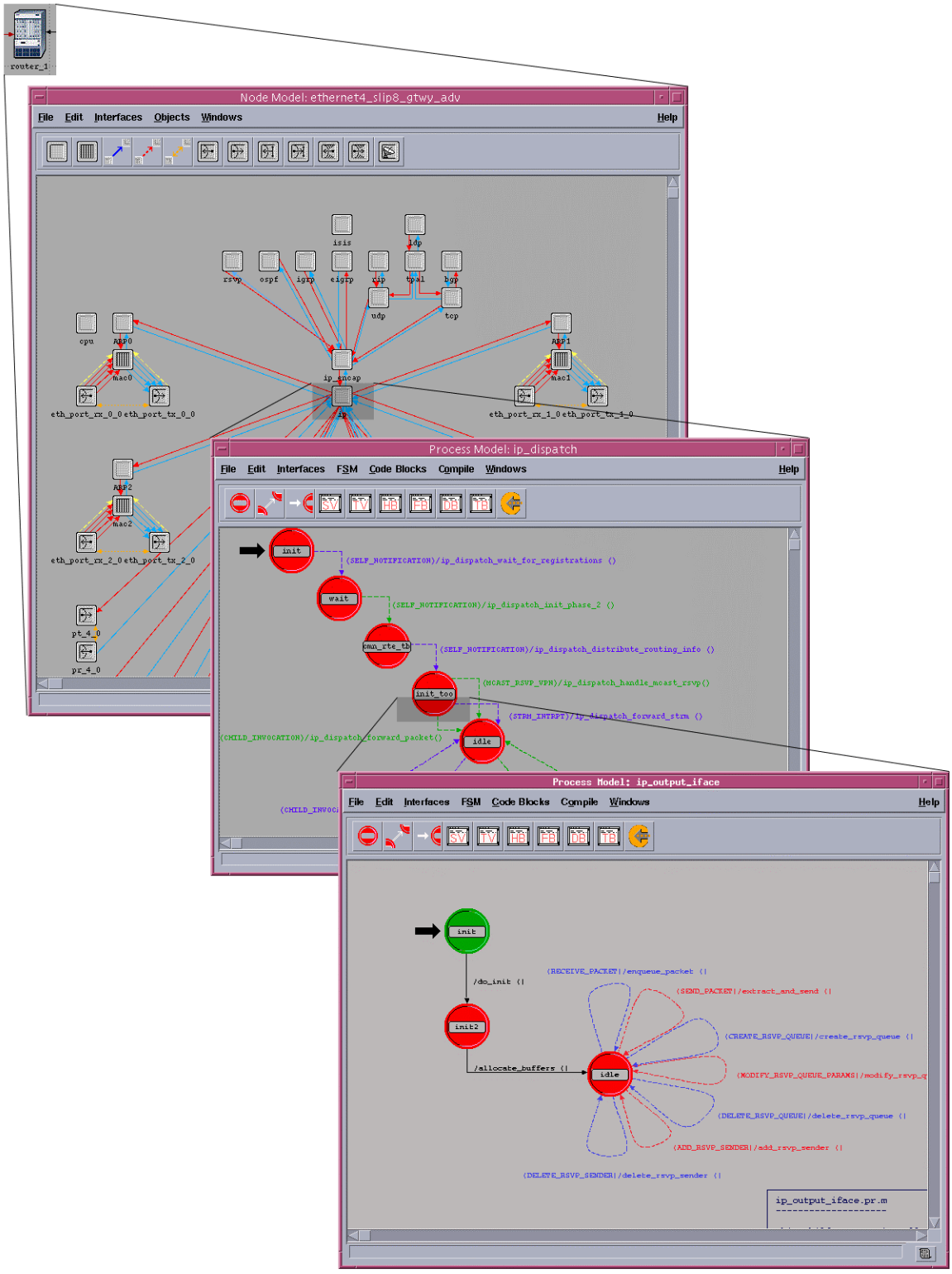

Figure 5. Supporting QoS in OPNET

In a typical simulation, packets are routed from any input interface to the output interface by the *ip_dispatch* process. The *ip_output_iface* is aware of the outgoing link capacity and will perform queuing accordingly. For smart queuing, we modified the *ip_output_iface* process to include multiple queuing management (qm) structures; that is, all four schemes are available to be used. In addition, the process also performs the following functions:

- collecting incoming packet stream statistics,
- performing switching decision,
- buffering packets during transistion of queuing schemes, and
- passing packets to the appropriate queue management structure.

Figure 6 shows a diagram describing this process.

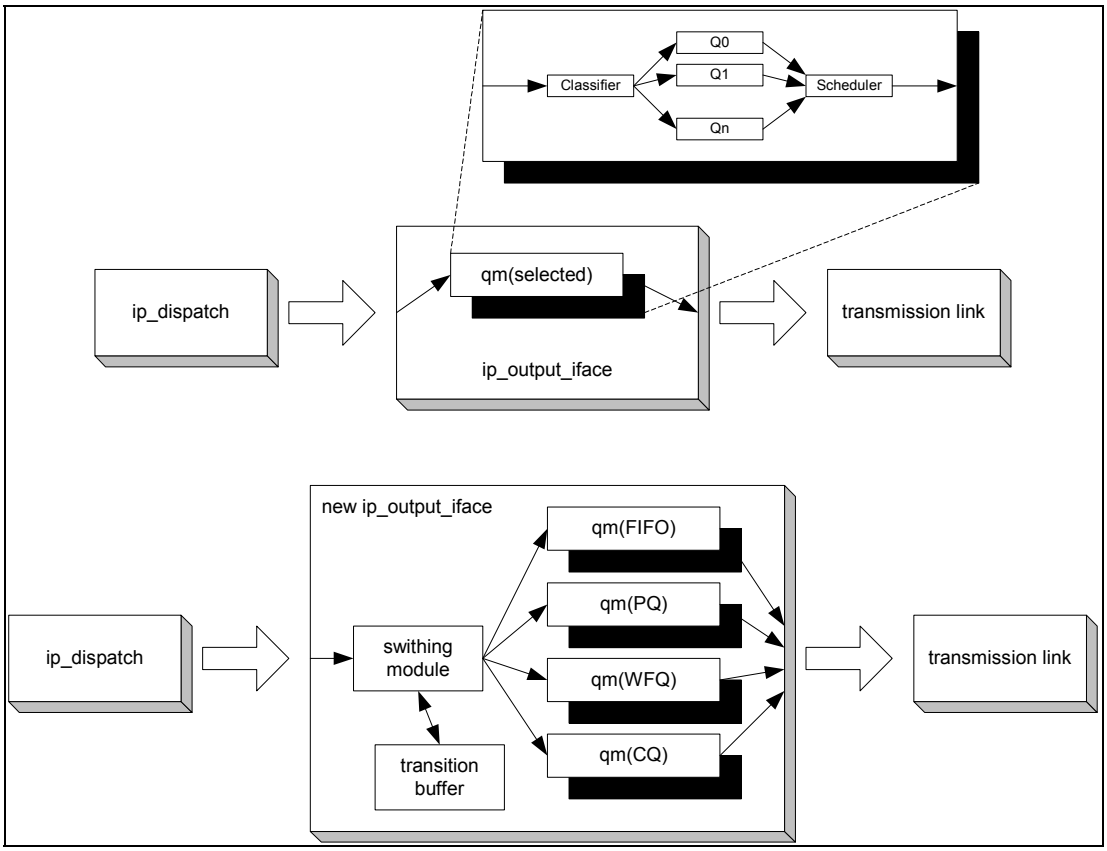

Figure 6. Smart queuing process flow

The different qm structures are independent and they are not aware of each other. Hence, a synchronization method is required during switching so that the next queuing method waits until the current method is finished with packets in its queues, before transmitting any packets. Thus, a packet counter is associated with each qm to keep track of the number of packets within the qm. New packets are kept in the transistion buffer until the current qm is finished; and once this occurs, the contents of the buffer is flushed to the next qm. As of the writing of this report, we were still unable to correctly implement the

transistion buffer. Instead, in our simulation, we dropped the new incoming packets until the current qm is completed.

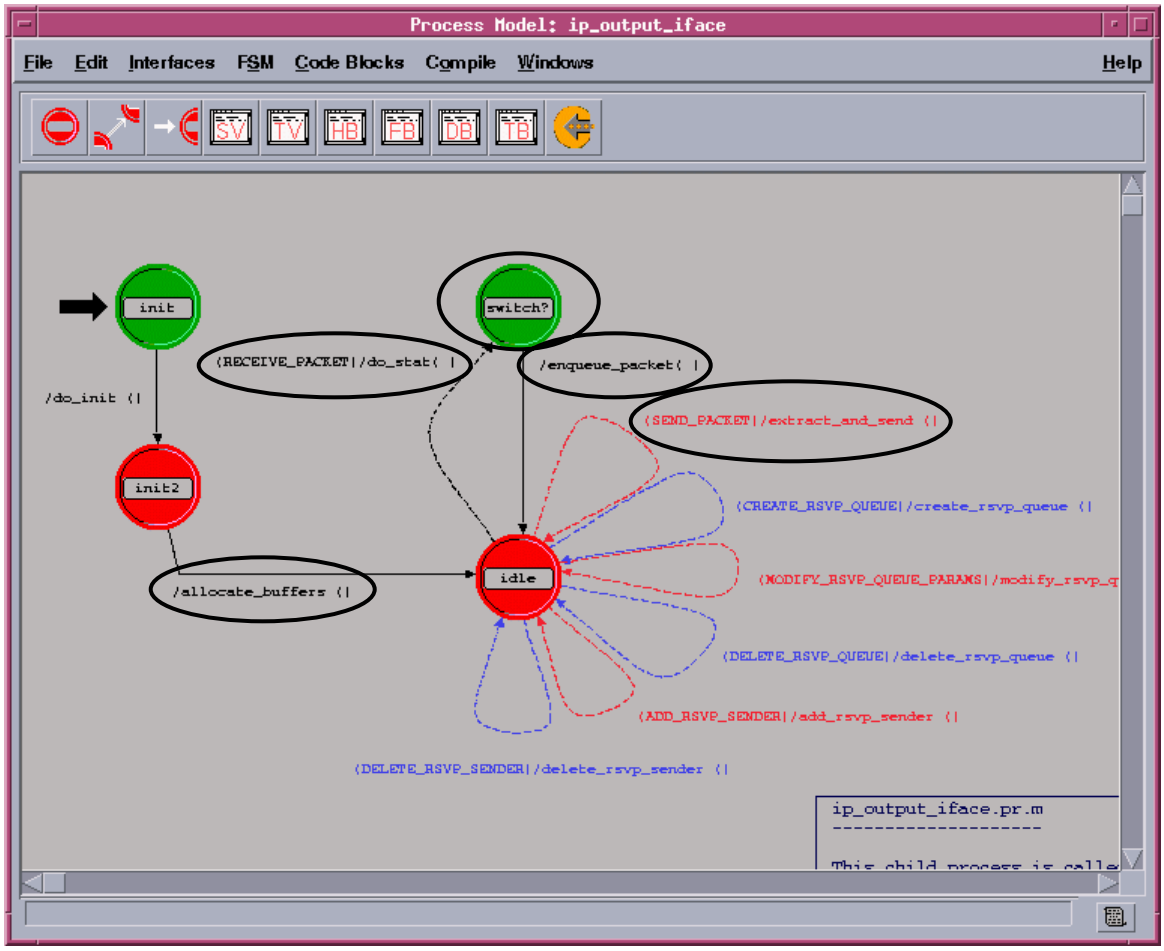

Figure 7. Smart queuing state transistion diagram

Figure 7 shows the resulting state transition diagram of the modified *ip_output_iface* process model. In the `allocate_buffers()`, the 4 qm structures are initialized and statistic handles are registered. Whenever a `RECEIVE_PACKET` interrupt is received (from *ip_dispatch*), the `do_stat()` procedure is called, where the swithing module collects the traffic information and records it to a table (see Table 1).

source_address	*ToS*	*last_pkt_time*	*allow_rate*	*active*	*misbehave*
192.0.0.1	2	30.15 sec	25.0 pkt/s	yes	no
192.0.0.2	0	12.50 sec	20.0 pkt/s	no	no
…					

Table 1. Sample traffic parameters

If the packet *source_address* is not in the table, an entry is added. The *last_pkt_time* is set to the current simulation time, the *allow_rate* is fixed according to the *ToS*, and the user status is set to active. If the packet *source_address* is in the table, the transfer rate is

calculated according to $\frac{1}{current\_sim\_time - last\_pkt\_time}$. If this value is greater than *allow_rate*, the status is set to misbehave and *last_pkt_time* is updated, otherwise only the *last_pkt_time* is updated. Two user-configureable parameters are used to specify the behaviour of the switching module. The first is *activity_timeout*, which specify the duration that must elapse before a user is considered inactive. Thus, for each user, if the value (*current_sim_time–last_pkt_time*) is greater that *activity_timeout*, that user is set to be inactive. The second parameter is *switching_sensitivity*, which will be described next.

From the above table, the swithing module then determine which queuing to use by considering all active users and according to the following rules:

- If there is a misbehaving user, use WFQ
- Else if there is more than one ToS, use CQ
- Else use FIFO

These rules were selected primarily for validating our model. Within OPNET, WFQ and CQ implementation are very similar [7], both provide differentiated service using weights and byte count respectively. Therefore, these rules essentially dictates that if there is no need for differentiated service, use FIFO to take advantage of lower queuing overhead and faster processing time.

If the current queuing is different from what is recommended according to the rules, we have to switch. In order not to switch whenever there is a slight change in traffic condition, the *switching_sensitivity* packet counter is used. As an example, say we are currently using FIFO because there are two well-behaving users at ToS 2. Then a routing information packet is sent to neighbouring routers using ToS 0 periodically. This will trigger a switch that is unnecessary. The *switching_sensitivity* is used to avoid this by resisting the switch for a certain number of packets. For example, if it is set to 20, then there must be a total of 20 incoming packets that continuously cause a switch recommendation before actual switching is carried out. This value is related to *activity_timeout*, since the total time taken to process this number of packets should be greater than *activity_timeout*, to prevent switching caused by periodic packets.

We have also modified the `enqueue_packet()` and the `extract_and_send()` procedures so that packets are enqueued to and dequeued from the appropriate qm. In addition, the QoS Configuration Object is updated to include the modified *qos_attribute_definer* process model. This allows user to configure the smart queuing parameters from the project workspace, as shown in Figure 8.

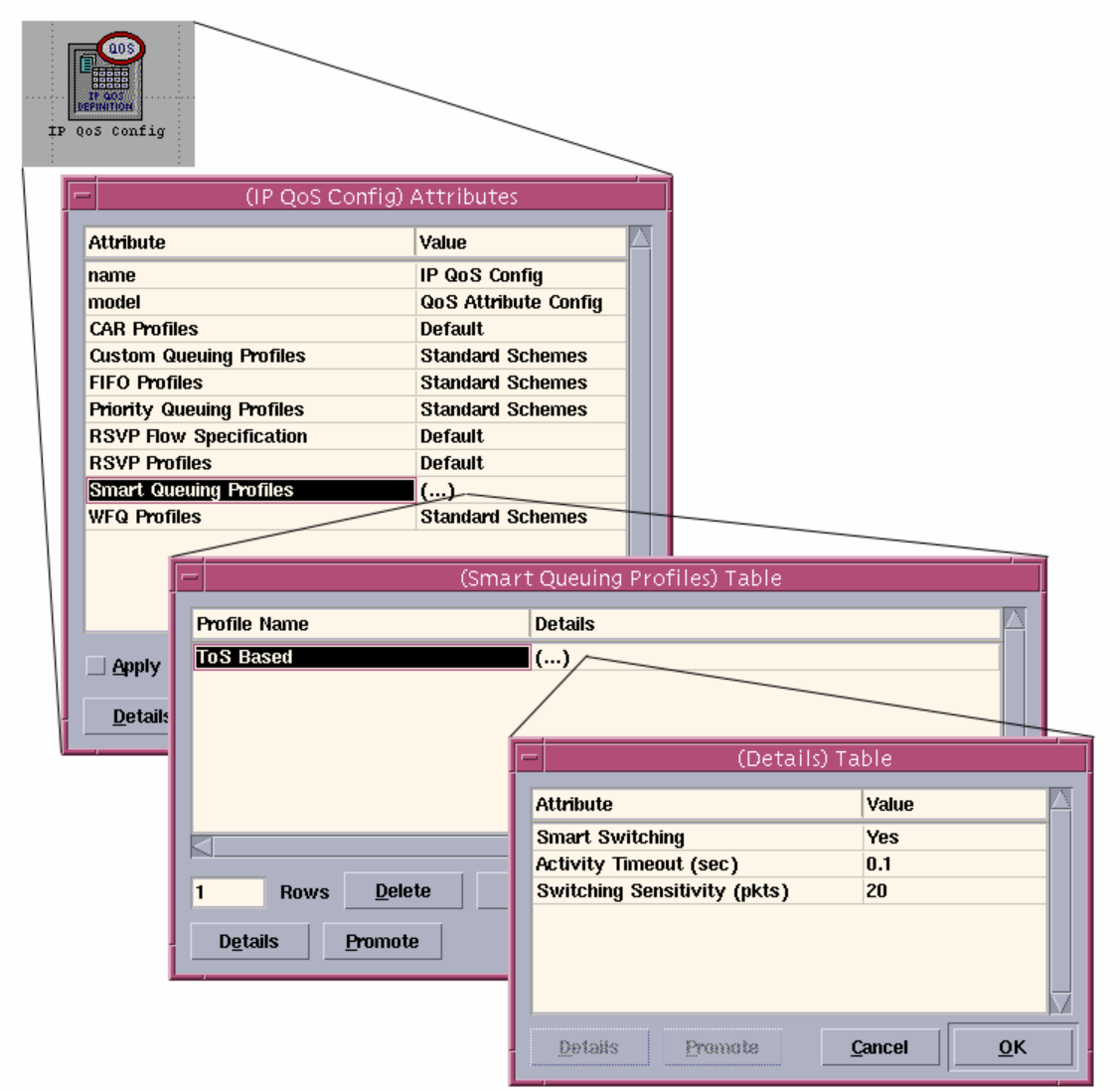

Figure 8. User-configurable parameters for smart queuing

3.3 Network Topology and Simulation Settings

In creating our network for simulation, we used standard OPNET objects from the Internet Toolbox pallete. Figure 9 shows the topology created for the simulation. Four Ethernet workstations are the source of our IP traffic. These are connected to a switch, which is then connected to an IP router via 100baseT connections. The destination of our traffic are four servers that are connected, via 100baseT, to a different router on the other side of the network. A bottleneck is created between the two routers using 56 kbps link in order to simulate congestion.

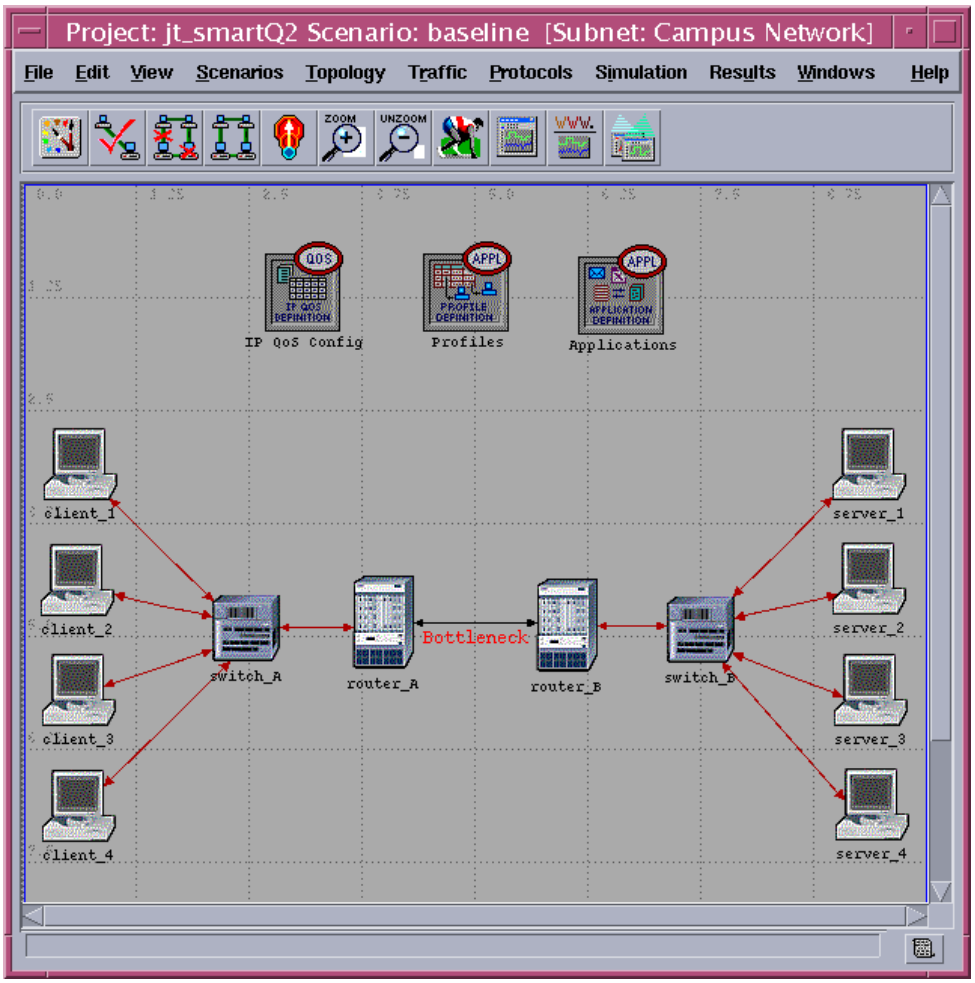

Figure 9. Network topology

There are two main types of protocol associated with IP networks: TCP (Transmission Control Protocol) and UDP (User Data Protocol). To simplify our simulation, we have used constant bit rate (CBR) UDP traffic. These were generated using standard OPNET video conferencing application. The packet size is selected to be 165 bytes, resulting in line capacity of approximately 40 pkt/s when using 56 kbps link. In order to simulate different traffic conditions, the four clients are active at different times. Table 2 summarizes the behaviours of the four clients.

	Traffic Rates (pkt/s)	ToS	Start Time (sec)	Finish Time (sec)
Client_1	25	2	15	120
Client_2	25	2	30	45
Client_3	20	0	60	75
Client_4	30	2	90	105

Table 2. Network clients simulation settings

The classification scheme is set to ToS Based, and we have set the allowable rates for ToS 0 and 2 to be 22 pkt/s and 27 pkt/s respectively. Thus Client_4 traffic will be

considered to be non-conforming. Figure 10 shows the graph of traffic sent by the four clients. Using smart queuing and the described switching rules for this scenario, we expect that FIFO would be used when Client_2 is active, CQ would be used when Client_3 is active, and WFQ would be used when Client_4 is active. Both the weights and byte count for WFQ and CQ profiles are set to 20 and 25 for ToS 0 and ToS 2 respectively.

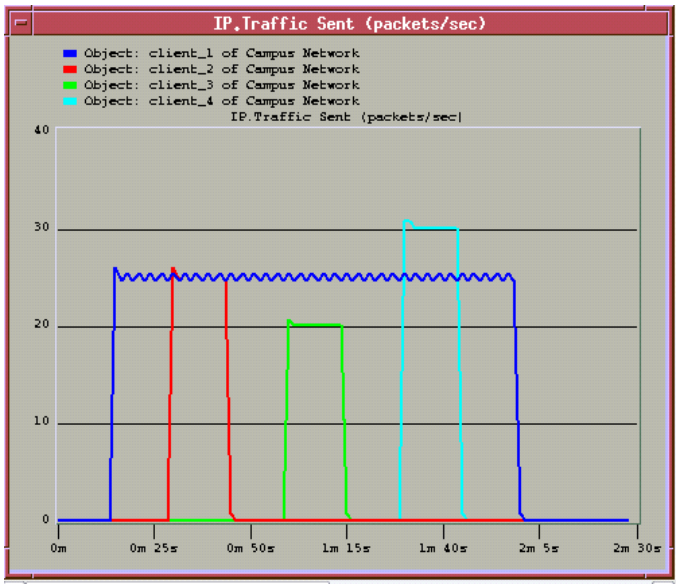

Figure 10. Clients packet send rates

The *activity_timeout* is set to 0.1 seconds, as this value should be larger than the packet interarrival time of the slowest sender, ie. larger than 0.05 seconds. The *switching_sensitivity* is set to 20 packets. This value can be approximated by calculating the total number packets received during a timeout period, ie. according to $number\_of\_users \times average\_send\_rates \times activity\_timeout = 4 \times 25 \times 0.1 = 10$ packets. As a comparison, we also run this simulation independently using only FIFO queuing and only WFQ queuing. However, we will not attempt to provide quantitative comparison between smart queuing and these methods as the transition buffer is yet to be implemented.

4 Discussion

4.1 Model Verification

We will begin by showing that our implementation of smart queuing is in working order. Below is a graph of the "packet send rate" for the 3 queuing mechanisms that smart queuing used while running the experimental setup as described in section 3.3.

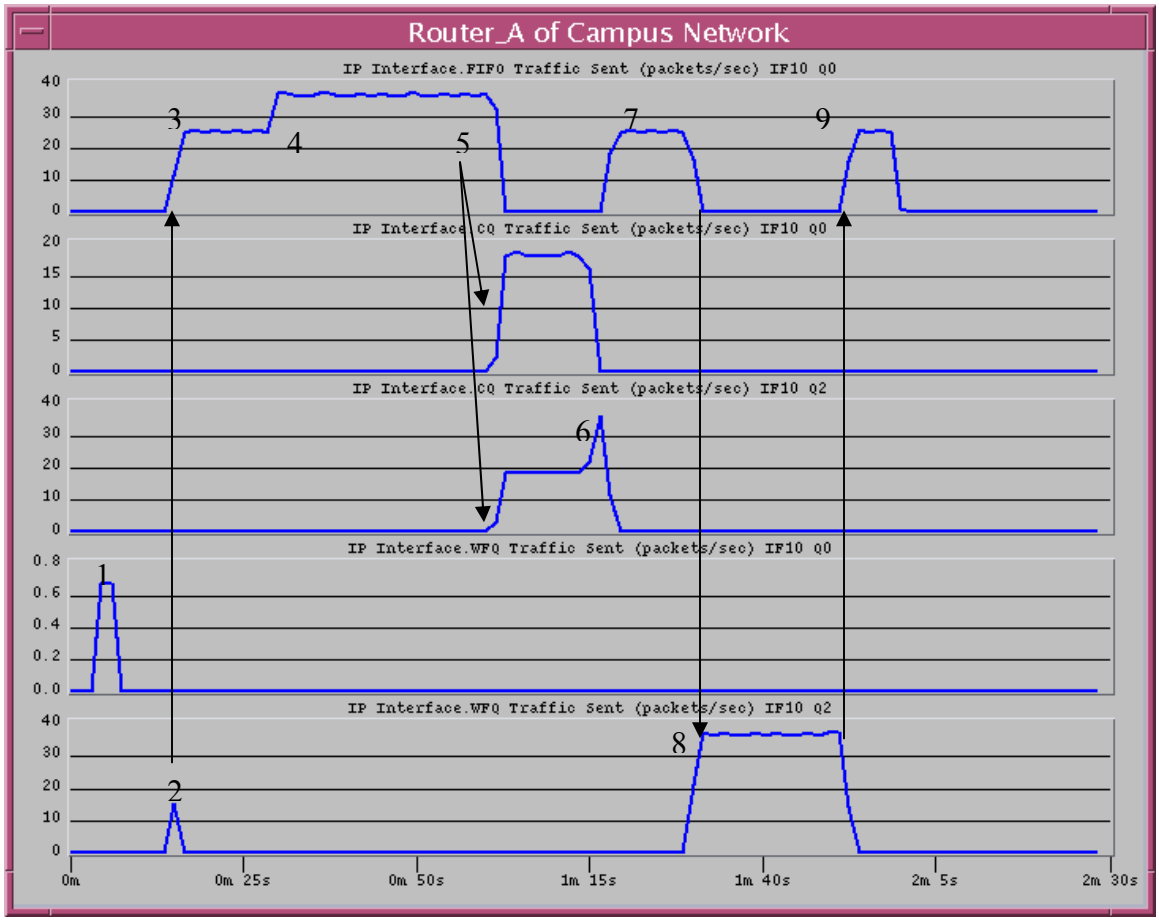

Figure 11. Packet sent rate of the 3 queuing mechanisms (FIFO, CQ, WFQ) smart queuing used inside router A.

(1) At first, smart queuing starts with WFQ as the method to use. Initially, packets are sent among the various elements in the network (routers, switches, clients) to establish connection. These handshaking packets are received by router A and are handled using WFQ as anticipated.

(2) After handshaking is done, router A begins receiving packets containing actual data from client 1. Smart queue samples the incoming packets to determine if a change in queuing method is necessary. WFQ is still used by smart queue even though it observes that there is only one non-misbehaving client present and FIFO should be used instead. This is due to the resistance as defined by the *switching_sensitivity* parameter, and smart queuing switches to using FIFO only after 20 packets have gone by.

(3) All incoming packets from client 1 are handled using FIFO. This can be verified by noticing that the packet sent rate by FIFO plateaus at 25 packets/second, which is equal to the rate at which packets are generated by client 1.

(4) Client 2 begins sending packets into the network. Since client 2 is of the same ToS as client 1 and is also non-misbehaving, smart queuing decides that we should keep using FIFO.

(5) At this point, client 2 stops sending packets, but there are still outstanding packets left in the buffer, so FIFO is still sending at the maximum rate. Client 3 then begins sending packets. Client 3 is also non-misbehaving, but has a different ToS than client 1. Under this condition, smart queuing decides that we should switch to using CQ. Under CQ, a separate subqueue is created for each distinct ToS. Therefore, packets belonging to client 1 (higher ToS) are inserted and sent from subqueue Q2, and packets belonging to client 3 (lower ToS) are inserted and sent from subqueue Q0.

(6) Client 3 has stopped sending packets and all its outstanding packets have been sent. Under this condition, only client 1 is still active and smart queuing dictates that we should change back to FIFO. Before we can do so however, there are still outstanding packets from client 1 under CQ that has not been cleared. Since there are no more client 3 packets that needs to be send, all of the output stream can be given to client 1's packets, so the sent rate of client 1's subqueue increases, causing the peak as shown.

(7) Only client 1 is active, so smart queuing switches back to FIFO.

(8) Client 4 begins sending packets into the network. Since client 4 is misbehaving, smart queue dictates that WFQ should be used. Both client 1 and client 4 have the same ToS, therefore, their packets are handled by the same subqueue.

(9) Client 4 stops sending packets. Now, only non-misbehaving client 1 remains, and smart queuing selects FIFO again.

Due to the fact that we were unable to successfully implement a transition buffer to hold incoming packets during switching between different queuing mechanisms, we were forced to drop incoming packets. For all queuing mechanisms that use multiple subqueues, all forced packet drop statistics are associated with subqueue Q0. From the graph below (Figure 12), one can see packet drop occurred at the same instance as when we switched among various queuing schemes (Figure 11).

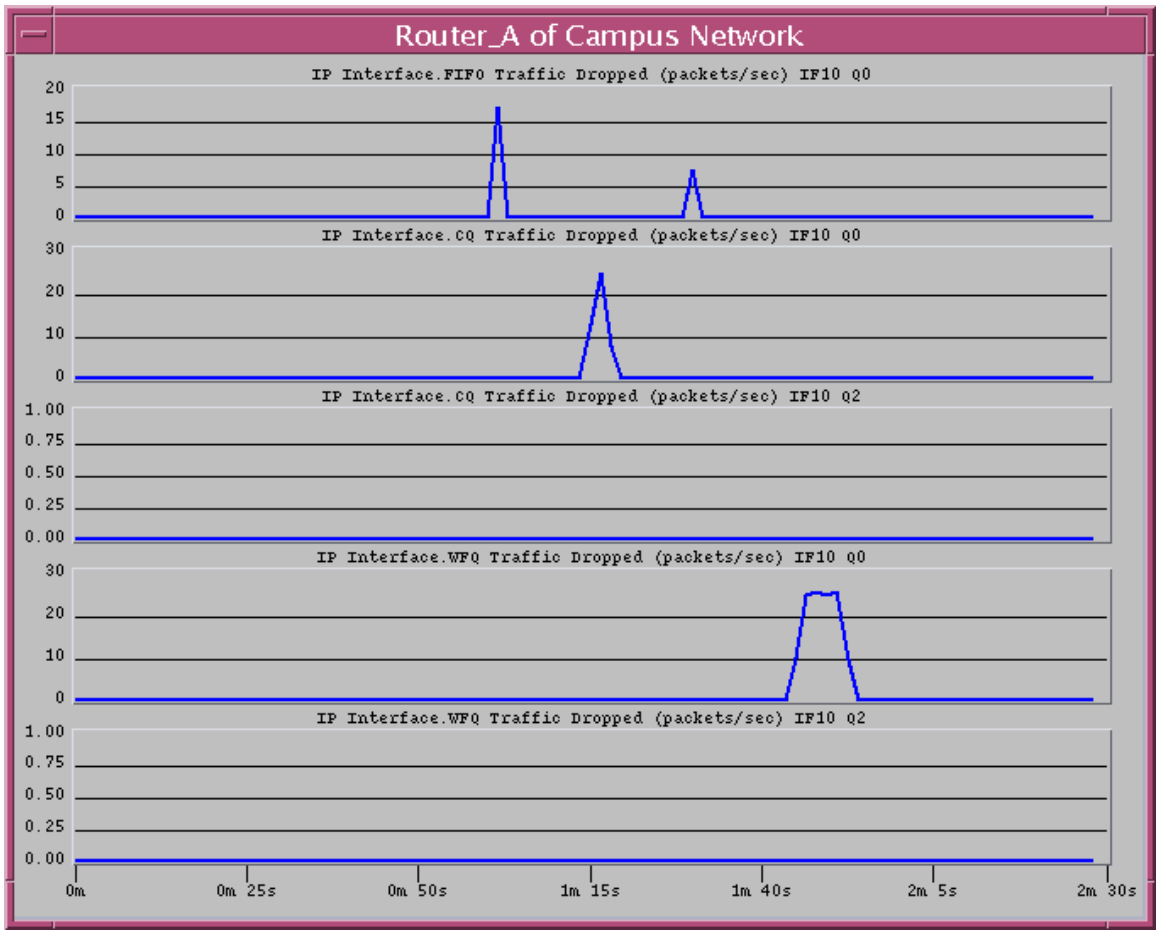

Figure 12. Packet dropped rate experienced by various queues during simulation.

4.2 Client End-to-End Delay

Figure 13 below is a graph showing the end-to-end delay experienced by client 1 of our network.

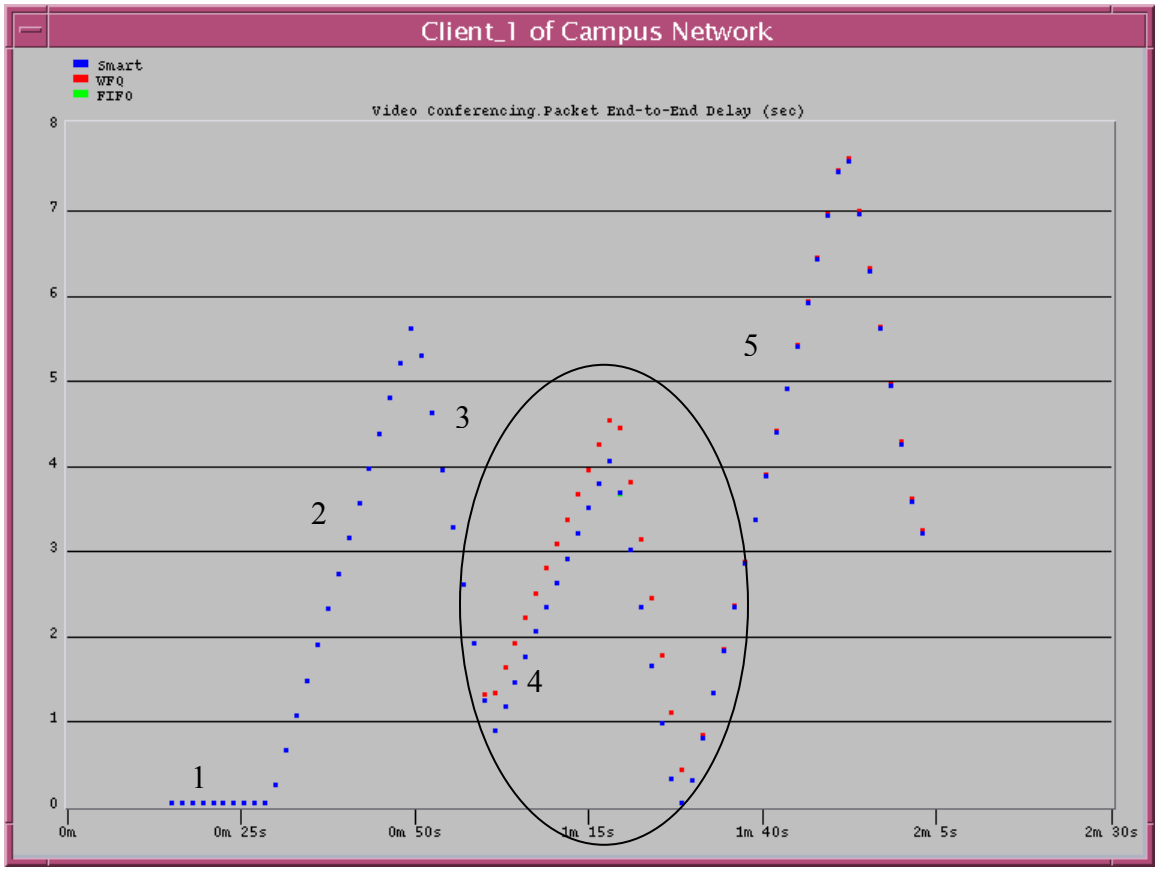

Figure 13. End-to-end delay experienced by client 1

In the above graph, the curve described by the red dots is the end-to-end delay experienced by client 1 while using only WFQ for the entire duration of the simulation. The curve described by the blue dots is the end-to-end delay experienced by client 1 while using our smart/adaptive queuing approach. The curve described by the green dots (overlaid by the blue curve) is the end-to-end delay while using only FIFO.

It is of interest to note that the end-to-end delay experienced by client 1 is identical whether router A is using either FIFO only, WFQ only, or smart queuing only, except during the time when client 1 and client 3 are active. The reason for this similarity can be explained as follows:

During time period (1), only client 1 is sending packets. Under smart queuing, router A selects FIFO queuing scheme. So the delay between smart queuing and FIFO should be the same (as illustrated in Figure 13). In terms of WFQ, there is only one client with only

one type of ToS active in the network at this particular time. So under ToS based WFQ, only one subqueue is initiated and established to handle the queuing of incoming packets. This is identical to the FIFO case, so the delay experienced should be the same.

During time period (2), both clients 1 and 2 are sending packets. Since both clients 1 and 2 are non-misbehaving and of the same ToS type, smart queuing also selects FIFO as the queuing method of choice. Therefore, the delay between smart queuing and FIFO should be the same. Similarly with WFQ, there are only packets of one ToS type, so packets from both client 1 and 2 are inserted into the same subqueue. Therefore, the delay should be identical to FIFO. The introduction of client 2's packets now competes with the packets of client 1 for bandwidth on the output stream, so the delay experienced by client 1 increases until client 2 no longer sends packets (3), at which point the delay for client 1 starts to decrease.

A difference in delay time can be seen during time period (4) when both clients 1 and 3 are sending packets. At this time, both clients 1 and 3 are non-misbehaving and are of different ToS types. Under this condition, smart queuing selects CQ as the method to use. Since CQ and WFQ are both ToS based, two subqueues are allocated and used, one for each ToS type. This should result in no delay difference between CQ and WFQ. However, the graph shows that using smart queuing results in a lesser delay when compared to WFQ. Our explanation to this phenomenon is as follows. Since smart queuing drops incoming packets while in the transition to a different queuing scheme, once the new scheme is established, the queue is empty. So subsequent packets do not need to wait for the backlog of previous packets to be sent first, but can themselves be sent right away. So the overall packet delay is of the same slope, but is of lesser value.

Another interesting point to note is the smaller delay experienced by client 1 when using FIFO as compared to WFQ. This can be explained by the fact that WFQ requires more computational processing than FIFO. With WFQ, two subqueues are allocated and supported, one for each ToS type. Packets are directed to their appropriate subqueues depending on their ToS. The decision to select which subqueue to dequeue from depends on the weights of each subqueues. It is much simpler with FIFO, where only one subqueue is maintained.

Lastly, during time period (5), both clients 1 and 4 are sending their packets. Since client 4 is a misbehaving user of the same ToS type as client 1, smart queuing selects WFQ as the method to use. This justifies the identical delay between smart queuing and WFQ. In the case of similarity between WFQ and FIFO, since both clients 1 and 4 are of the same ToS type, packets from both clients are placed in the same subqueue under WFQ. This is identical to the one subqueue in FIFO, so identical delay is expected.

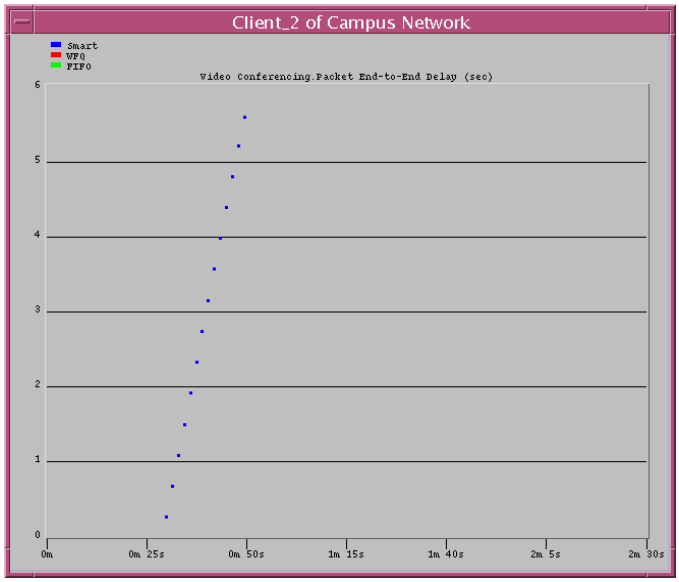

Figure 14. End-to-end delay experienced by client 2

Client 2 is only active for 15 seconds between simulation time 0:30 to 0:45. During this time, both clients 1 and 2 are sending their packets. Since clients 1 and 2 are both non-misbehaving and of the same ToS class, smart queuing selects FIFO as the queuing scheme. Therefore, delay times are identical between smart queuing and FIFO. The identical delay between smart queuing and WFQ can be explained using the same explanation as for client 1 during time period (2). In fact, the delay experienced by clients 1 and 2 during time period (2) is identical.

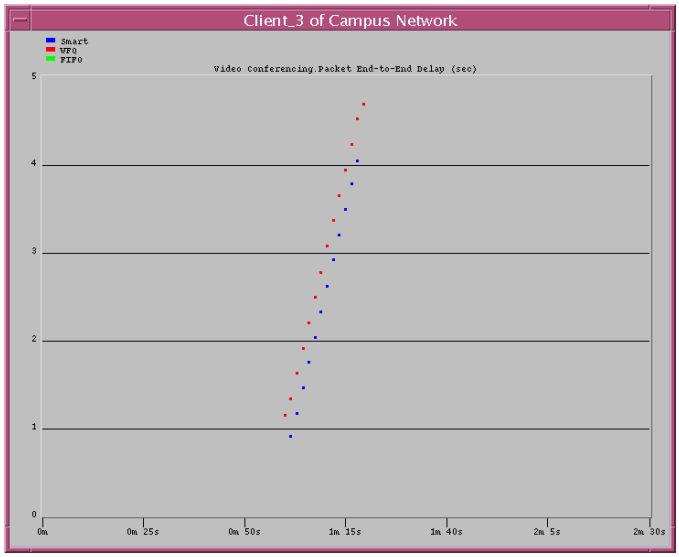

Figure 15. End-to-end delay experienced by client 3

From Figure 15, the delay experienced by client 3 is similar to client 1 during time period (4) of client 1 (see Figure 13). Although they are of different ToS, the assigned weights are proportional to their send rates. This results in similar delay for both clients. With

regards to the difference between smart queuing and FIFO only, the same explanation for client 1 can be applied.

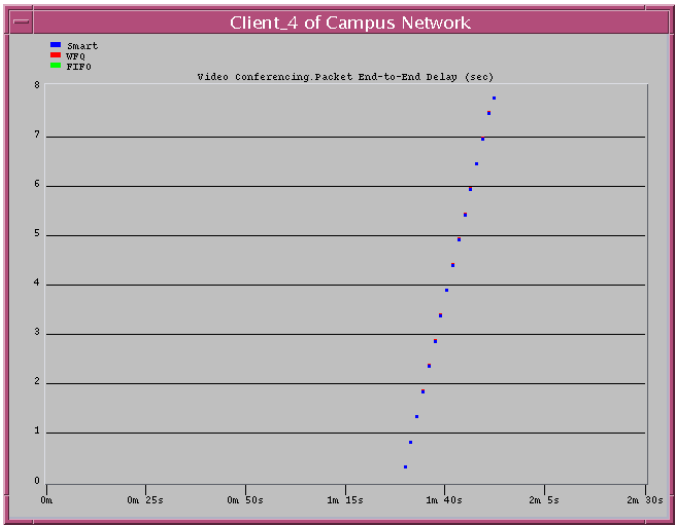

Figure 16. End-to-end delay experienced by client 4

The delay experienced by client 4 is identical to client 1 during time period (5), and can be rationalized using the exact explanation as for client 1.

5 Conclusion

Current network elements only support fixed queuing schemes. We believed that the option to dynamically change among several queuing schemes would allow for better performance in terms of a reduction in end-to-end delay for all clients. So far, we have implemented the ability for changing queuing scheme on the fly and have shown that it works in OPNET.

Due to time constraints and the fact that this is our alternate project[1], we have not yet found a scenario that shows smart queuing provides better performance than fixed queuing. Furthermore, we have made many simplifying assumptions in our project. Among many of the outstanding issues that should be further considered are: the effect of bursty, variable rate and TCP traffic; the effect of active queue management (RED) [2,4]; incorporating other queuing schemes, such as VirtualClock [1]; better traffic characterization parameters; and better switching conditions (rules, lookup tables, online simulation).

[1] Our original project focused on UMTS (Universal Mobile Telecommunication System), but because we do not have access to the UMTS specialized model in OPNET, we selected this project instead.

The algorithm for smart queuing is also quite complex, and arguably could be computationally as expensive as WFQ. It requires continuous sampling of traffic condition as well as switching and synchronization management between different queuing mechanisms. However, smart queuing is not an actual queuing mechanism, rather it is more of a control function. From this point, we could separate the control from the actual queuing, allowing smart queuing to be implemented in software and still take advantage of the fast switching hardware.

6 References

[1] N. Alborz and L. Trajkovic, "Implementation of VirtualClock Scheduling Algorithm in OPNET", *Proceedings of OPNETWORK 2001*, Washington DC, Aug. 2001.

[2] Chengyu Zhu, Oliver W.W.Yang, James Aweya, Michel Ouellette, and Delfin Y. Montuno, "A Comparison of Active Queue Management Algorithms Using OPNET Modeler", *Proceedings of OPNETWORK 2001*, Washington DC, Aug. 2001.

[3] Chuck Semeria, "Supporting Differentiated Service Classes: Queue Scheduling Disciplines", White Paper, Juniper Networks. www.juniper.net/techcenter/techpapers/200020.html

[4] Costin Iancu, Anurag Acharya, "A Comparison of Feedback Based and Fair Queuing Mechanisms for Handling Unresponsive Traffic", *Proceedings of ISCC' 2001 - Sixth IEEE Symposium on Computers and Communications*, Hammamet, Tunisia, July 3-5, 2001.

[5] Goncalo Quadros, Antonio Alves, Edmundo Monteiro, Fernando Boavaida, "How Unfair Can Weighted Fair Queuing Be?", *Proceedings of ISCC'2000 – Fifth IEEE Symposium on Computers and Communications*, Antibes, France, July 4-6, 2000.

[6] A. Veres and M. Boda, "The chaotic nature of TCP congestion control," *Proceedings of IEEE INFOCOM*, pp. 1715-1723, TelAviv, Israel, 2000.

[7] OPNET Technologies, Inc., Washington DC, OPNET documentation V.8.0.C.