# Simulating the Performance of prioritized scheduling with buffer management for[1] differentiated services architecture using NS2

Technical Report 2009-563

Manar Alalfi

School of computing- Queen's University

Kingston, Ontario, Canada

alalfi@cs.queensu.ca

## 1. Introduction

In traditional Internet, all connections get the same treatment in the network. This is in contrast with other networking concepts, such as the ATM (Asynchronous Transfer Mode), that can offer quality of service requirements to connections at the price of much higher signaling and processing related to the acceptance of new connections and maintaining the guarantees of ongoing connections. Moreover, since network resources are limited, offering guarantees on performance measures requires to reject new connections if resources are not available. This is in contrast with the best effort nature of today's the Internet where no admission control is performed. Yet, it has been recognized it is important to differentiate between connection classes and to be able to allocate resources to connections according to their class. Thus a subscriber that is willing to pay more could benefit of smaller delays and larger throughputs. This is in particular of interest for real time applications over the Internet (voice, video). For that reason, the Diffserv has been introduced. It is based on marking packets at the edge of the network according to the performance level that the network wishes to provide them; then according to the marks, the packets are treated differently at the network's nodes. A common way to differentiate packets is by using RED buffer or RIO buffer which is based on RED and using different parameters for different packets.

In this report a different buffer management scheme[1] will be described and implemented for the aim of providing different level of services for different class of traffic and that to support differentiated services Internet , the improvement of this scheme over other schemes is in minimizing the packet loss ratio and or delay for high-priority classes without starving low-priority classes, a number of test comparisons will be provided using different number and or type of traffic, to compare the scheme performance over RIO buffer management with FIFO scheduling.

## 2. Buffer Management Models:

The buffer management Model that we want to use for the aim of comparison with the proposed Model on (1) is RIO which is based on RED,(Random Early Drop ) which is a buffer management scheme proposed by Floyd and Jacobson. It is used in router to prevent congestion caused by TCP packets. In normal drop-tail router, packet gets dropped when queue is full. In RED router, it will monitor average queue length. When the average length exceeds some threshold, it will either drop or mark packets with a certain probability. RED with in/out (RIO) is proposed by Clark and Fang as an extension of RED. First, packets are classified as IN or OUT depending on whether they conform to some profile. Then the router will use the same mechanism as in RED but has two sets of parameters, one for in packets and one for out packets. OUT packets are more likely to be dropped then IN packets, thus it can provide different levels of drop precedence.

## 1. RED

As mentioned above, the probability of dropping is a function of average queue length. The curve is shown as follow:

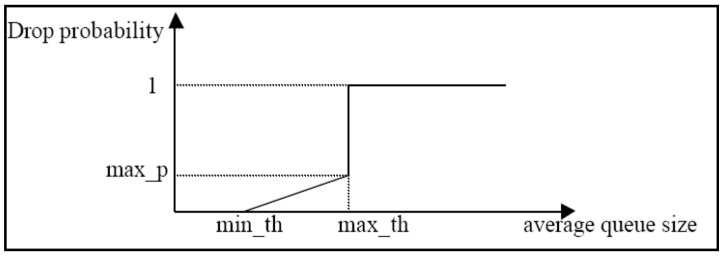

**Figure1: RED Queue Parameters**

There are 3 basic parameters to represent the probability function, called min threshold (min_th), max threshold (max_th), and max probability (max_p). The probability curve is then fixed by these three parameters. There is one more parameter in RED queue, which represent the choice of "time constant" (roughly speaking) for the averaging for the average queue size, called w_q.The choice of this value is a tradeoff. If w_q is too low, then the estimated average queue size is probably responding too slowly to transient congestion. If w_q is too high, then the estimated average queue size too closely tracks the instantaneous queue size. simulation in NS, suggested max_th is approximately 3 times of min_th. And max_p should not exceed 0.1. It also recommends setting w_q to at least 0.001. The ns-1 and ns-2 simulators set w_q to a default of 0.002.

## 2. RIO

RIO has two sets of RED parameters. Define max_in, min_in and p_max_in to be min threshold, max threshold, and max probability for IN packets respectively. Similarly, we have max_out, min_out and p_max_out for OUT packets. The curves are shown below.

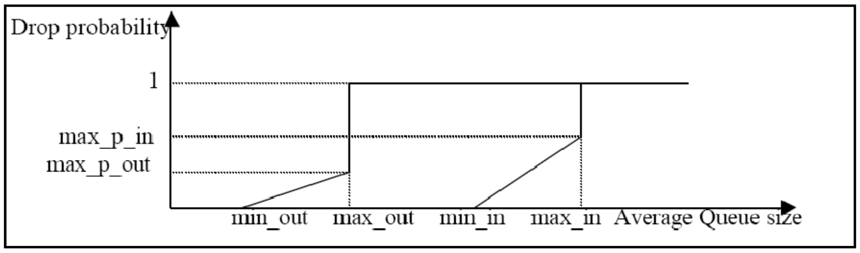

**Figure2 RIO Queue parameters**

## 3. Model description

The scheme that is presented in (1), is a versatile service and buffering scheme for different classes in a differentiated service Internet , in the proposed scheme there are three buffers, one assigned for higher priority class (HPC) traffic, the second one is for lower priority class (LPC) traffic and the third is shared by the both classes, where priority of the shared buffer occupancy given to HPC, this scheme is proposed for the purposes of reducing starvation for LPC traffic when it's sharing a bottleneck link with HPC, and that by reserving a dedicated buffer for it, and to handle short term congestion when the HPC traffic can use it's assigned buffer, where under normal network load the HPC buffer is set aside and HPC traffic can share a buffer with other traffic classes , the proposed model can be extended simply to support a wide range of traffic classes, which is more suitable for differentiated services over the internet.

## 4. Project scope

The scope of this project is to implement and test the scheduling and buffer management scheme that is represented in the paper (1), then to compare it's performance to that for RIO( Red with In and Out packets) with the following test cases :

- By using 2 source – CBR and VBR, with VBR traffic treated as out, and then by using another sources TCP Reno,TCP Sack, with TCP Sack treated as out traffic
- With N VBR sources , with traffic policing marking out packets, and then with N TCP Sack instead.

## 5. Model analysis

- **Buffer Management**

The scheme propose tow buffers with sizes and parameters determines the performance of the respective classes , the first buffer is assigned to HPC traffic and the second to LPC traffic, the two classes also have a shared buffer with higher priority occupancy given to HPC, the buffering strategy is shown in finger (3) which can also be described as follows:

- LPC traffic always use its dedicated buffer as long as it has a room, when it becomes full, the LPC packet will be inserted on Shared buffer if it has a room otherwise it will be dropped

- For HPC traffic, shared buffer is always checked first for a room, if there is a space the arrived HPP will be en queued in the shared Queue otherwise it will be en queued at HP Queue if there is a space available , other wise it will be dropped.

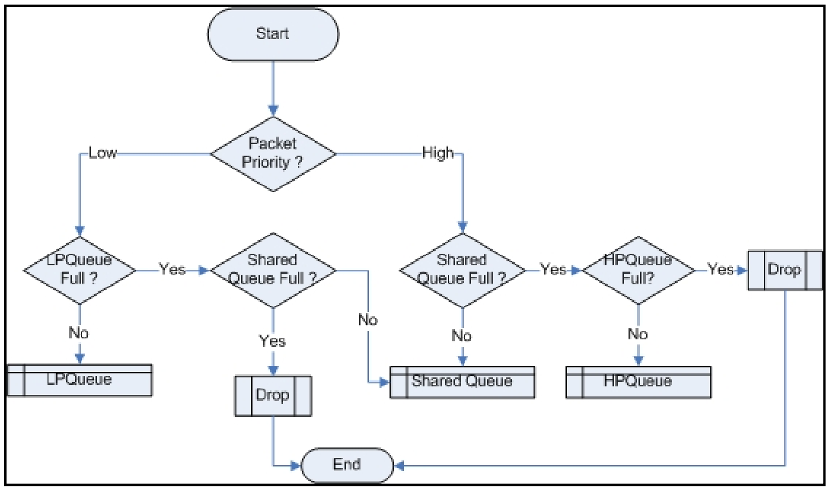

**Figure 3 : Enque Method in HPLP Class**

- **De queue Method ( Processing with FIFO Scheduling)**

  - When LPC packet is being served, and a room on LP Queue becomes available , the Shared queue will be searched for LPC packet , if there is one it will be removed from shared Queue and queued in LP Queue as shown in figure (4)

  - when ever a room becomes available for a new packet in the shared buffer, the space will be occupied by a HPC packet, so a HPC packet will be de queued from HP Queue and inserted in Shared Queue, this will be happened in two cases, the first when a packet is being served from the shared Queue and the second when a LPC packet is being removed from the Shared buffer this is shown on figure 2.
  - There are many scheduling mechanisms that can be used, FIFO , Round Robin , WRR, HPC service priority, LPC service priority, the one that is used in the implementation is FIFO, and that because this scheme performance will be compared with RIO which uses FIFO scheduling, so packets will be processed according to there sequence numbers from the different queue , the queue that have on its top the lowest sequence number packet will be served first.

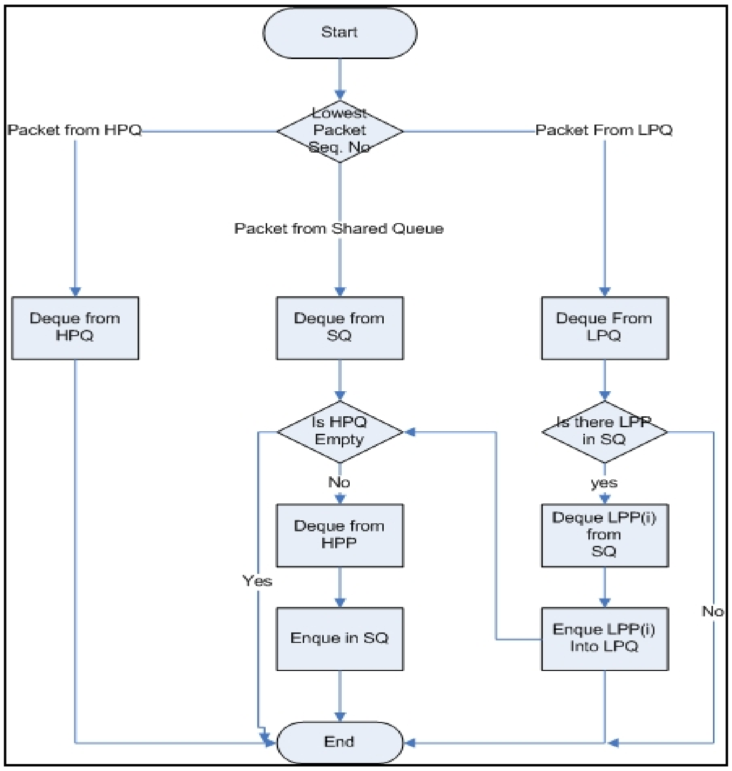

**Figure 4 : De queue Method ( Processing with FIFO Scheduling)**

## 6. Model implementation

- **Adding a New Module**

This section outlines the process of creating and adding new classes to the ns software hierarchy.

Adding a new module to ns consists of three steps:

o **Determining the need:**

The Diffserv functionality is captured in a Queue object; it is an alternative to other queue types such as DropTail, CBQ, and RED . A Diffserv queue requires:

- . to implement a multipurpose service and buffering scheme for different classes in a differentiated services Internet

- . to implement buffering priorities, for HPC and LPC that aims to low HPC packet loss ratio without starving LPC traffic

- **Determining the class hierarchy positioning:**

Since this new class implements the generic Queue functionality and extends it with new methods and attributes, it is placed beneath Queue in the object hierarchy, as shown in the following figure:

Connector Queue

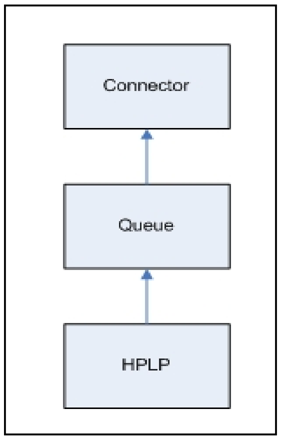

**Figure 5 : hierarchy positioning**

- **Writing code:**

Writing the code for the new module requires three or four steps, depending on the class:

***Step 1: Creating the header file ("HPLPF-queue.h")***

This file includes class specifications, as well as other definitions needed by the new class

***Step 2: Creating the main C++ file ("HPLPF-queue.cc")***

This file includes implementations of each of the new class's methods. To incorporate the new class into ns and make it accessible through Tcl scripts, the class must be linked to the ns class hierarchy. The following  code is used in "HPLP-queue" to add  HPLPF*Class* to the class hierarchy:

```
static class HPLPFQueueClass : public TclClass
{
public:
   HPLPFQueueClass() : TclClass("Queue/HPLPF") {}
        TclObject* create(int, const char*const*)
        {
              return (new HPLPFQueue);
      }
} class_high_proiority_Low_Proiority;
```

***Step 3: Modifying “Makefile”***

The third step is to add a reference to the new module in “Makefile,” so that when the *make* command is invoked the compiler generates a binary version of the new code and includes it in the ns compilation. The reference is added to the *object files* section of “Makefile”:

HPLPF-queue.o

- **Model integration**

For the created model to be used in any simulation scenario  it should be integrated into a complete model that is composed of a ( source(s) – router – destination ) in  TCL file , the new Queue now can be used in place of  RIO/RED , so it can be used  like any other predefined queues.

## 7. NS2 Simulation:

- **For RIO Queue :**

 To run ns simulation in ns, there are two steps need to be done. First, you need to define RIO queue and set parameters in tcl code. You also need a packets classifier to mark IN and OUT packet for you. Here can use token bucket policing and tagging.

**Set RIO Parameters:**

```
#out packets are droped before any in packets

Queue/RED/RIO set in_thresh_ 10 #min_in
Queue/RED/RIO set in_maxthresh_ 20 #max_in
Queue/RED/RIO set out_thresh_ 3 #min_out
Queue/RED/RIO set out_maxthresh_ 9 #max_out
Queue/RED/RIO set in_linterm_ 50 #1/max_p_in
Queue/RED/RIO set linterm_ 200 #1/max_p_out
Queue/RED/RIO set priority_method_ 0
Queue/RED/RIO set debug_ false
$ns duplex-link <node1> <node2> <b/w> <delay> RED/RIO
```

If priority_method_ is set to 1, flow with fid_ 0 will always get priority over other flows.
(Do not set it to 1 in simulating RIO)

o  Token bucket policing and tagging

```
set link1 [$ns link $n1 $n3]
set tcm1 [$ns maketbtagger Fid]
$ns attach-tagger $link1 $tcm1
set fcl1 [$tcm1 classifier]; # flow classifier
$fcl1 set-flowrate 0 100000 10000 1
```

“$fcl1 set-flowrate 0 100000 10000 1” set token bucket for flow with fid_ 0, the maximum rate of IN packets is 100000bps, the size of bucket is 10000, and initially there are 1 token in the bucket.

## 8. simulation scenarios

- **Simulation scenario 1 :**

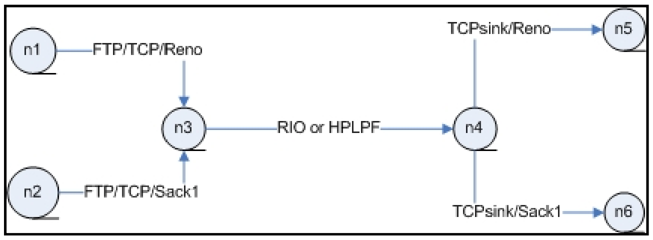

**Figure6: Network Topology For Scenario 1**

- 2 sources TCP Reno and TCP Sack , where TCP Reno is treated as IN , and SAKE as Out Traffic ( RIO queue ) , and in the proposed model Reno as HPC traffic and SAKE as LPC Traffic , the

bottleneck link ( that use RIO or HPLPF ) queues was tested , regarding to queue size, link bandwidth , loss rate , and traffic throughput, the results was drawn using xgraph tool in NS2 where the following graphs were generated :

**Scenario parameters and results analysis:**

- In all simulation scenarios the simulations were run for 50 sec, and the queue size for the bottleneck link was chosen to be 20 packet.

To distinguish the two flows from each other the IN flow ( TCP RENO ) is shaped by token bucket policer to have a rate that is 10 times greater than that for the Out flow ( TCP Sack ), we can see from figure ( 7 ), that the two flows using RIO Queue are not distinguished from each other and that because the flow rate ratio chosen by the policer was not sufficient to give each flow it's class of service , the flow rate ratio, and the bucket size were chosen in the first test to be for Reno flow 5 time than that for sack , For HPLPF the two flows were completely distinguished from each other for the same Rate, bucket size ratio

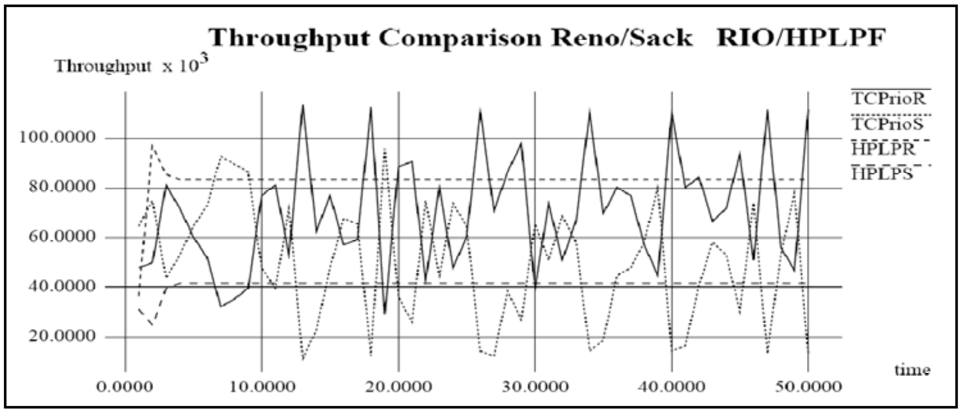

**Figure 7 : Throughput Comparison ( Rate for Reno 5 times Sack)**

A second test were done for the token bucket provides a rate and bucket size ratio for the two flows to be 1/10. Reno flows will given a rate and a bucket size which is 10 times given for sack flow, we can see from figure 8 that those policer parameters were able to provide the level of service required by the two flows , so they are completely distinguished from each other, where the throughput for the two flows using HPLPF remains the same for the two tests. We can observe from figure 7 and 8 , that while the two flows are distinguished from each other by using HPLPF Queue , they are more fair share than those which use RIO, which means that HPC packets don't starve LPC packet in the same way like RIO, and that was one of the objectives for designing HPLPF Queue.

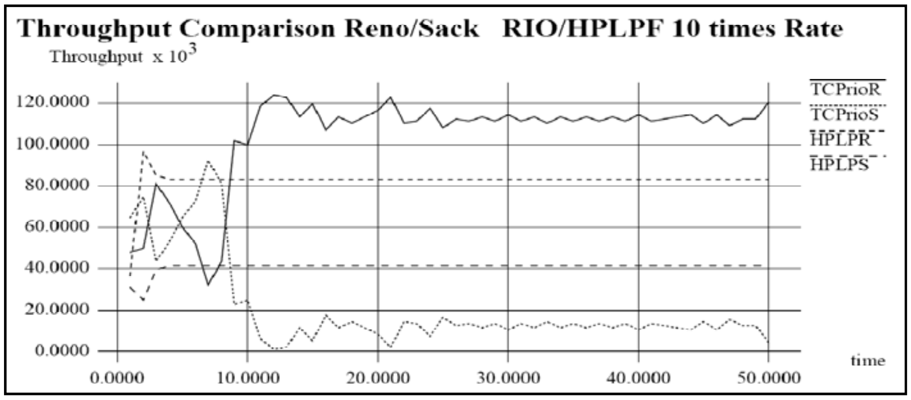

**Figure 8: Throughput Comparison ( Rate for Reno 5 times Sack)**

- We can see from figure ( 9 ) that The amount of packet loss with HPLPF was decreased with comparison of that for RIO, and that proves it's performance, regarding not starving LPC packets when sharing the same bottleneck link with HPC packets, while decreasing the amount of total packet dropped especially for HPC flow, figure (9 ) a shows the result of loss rate for test 1 ( mentioned above) , and figure 9(b) for test 2, the amount of packet loss in the two cases for HPLPF Queue doesn't changes, where it decreased in the case of RIO for test 2 as each traffic is successfully given it's level of service by the token bucket policer.

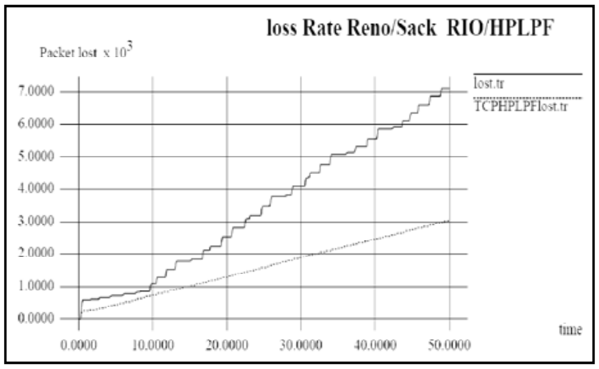

**Figure 9(a): loss Rate with flow rate for Rno 5 times Sack**

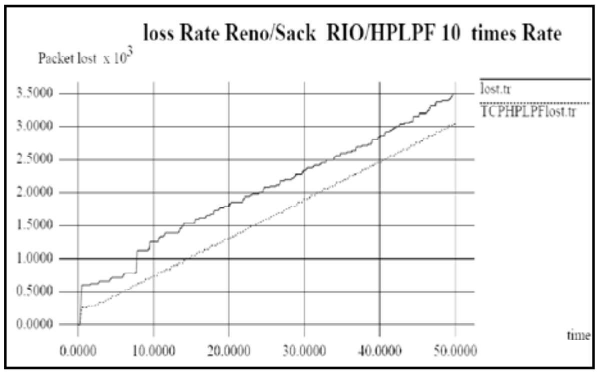

**Figure 9(b): loss Rate with flow rate for Rno 10 times Sack**

- Regarding the queue size, as I mentioned before it was chosen to be 20 packet, we can see that it get to be filled for many times in RIO, ie. Reach it's maximum size, when that happened the packets begin to be dropped and the queue size shrink as some of the packets are dropped and other are being served.

According to HPLPF queue it size remains some how more stable than that for RIO, which reflects that the total amount of loss is less than that for RIO, the HPLPF queue don't get empty for all the simulation scenario time , and that is shown in figure 10.

In figure 11 the queue size is generated for test two we can see that the amount of drop decreased for RIO , and that is expressed by the number of times the queue size dropped fast to 0, and that for the reason listed above, for HPLPF the queue size remains the same.

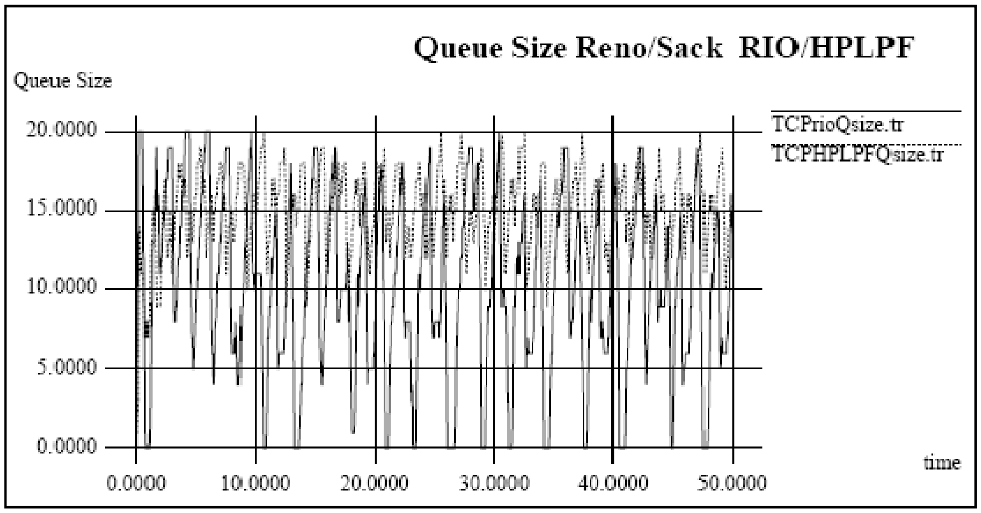

**Figure 10 : Queue size for Reno/Sack RIO/HPLPF for a rate ratio of 5:1**

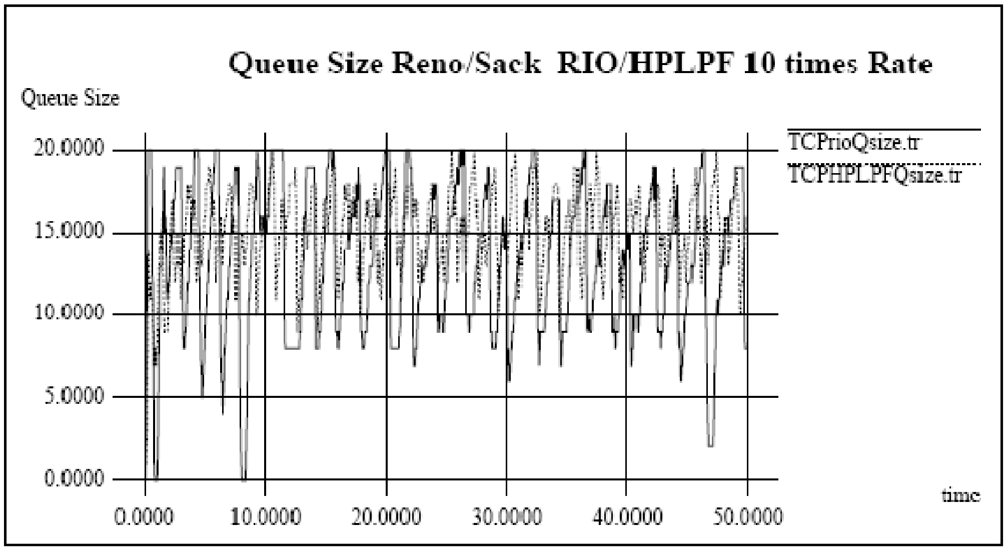

**Figure 11: Queue size for Reno/Sack RIO/HPLPF for a rate ratio of 10:1**

- **Simulation Senario2:**

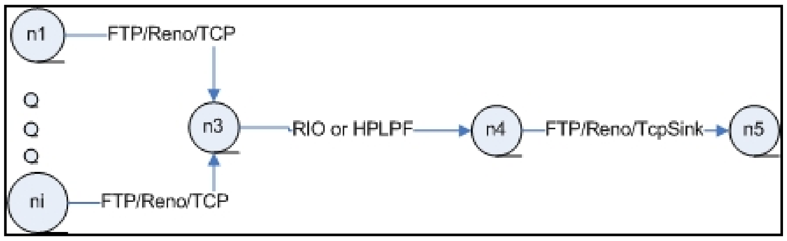

**Figure 12 Network topology for scenario 2**

- Using N TCP Sack with traffic policing ( Token bucket Policer) that shapes the flow Rate for the N TCP flows and marking them as IN or Out for RIO, High , low priority for HPLPF Queue , the generated graphs was as follows :

  - The simulation was run using 5 Sack flows for 50 , where the policer set the rates with different amounts according to the following formula

```
set fillRate [ expr { pow(1000,$i) }]
$fcl($i) set-flowrate $i  fillRate  [expr $fillRate/10] 1

Where each flow rate increases  by 10 amount of it's previous one,
and  that also the same for the bucket size.
```

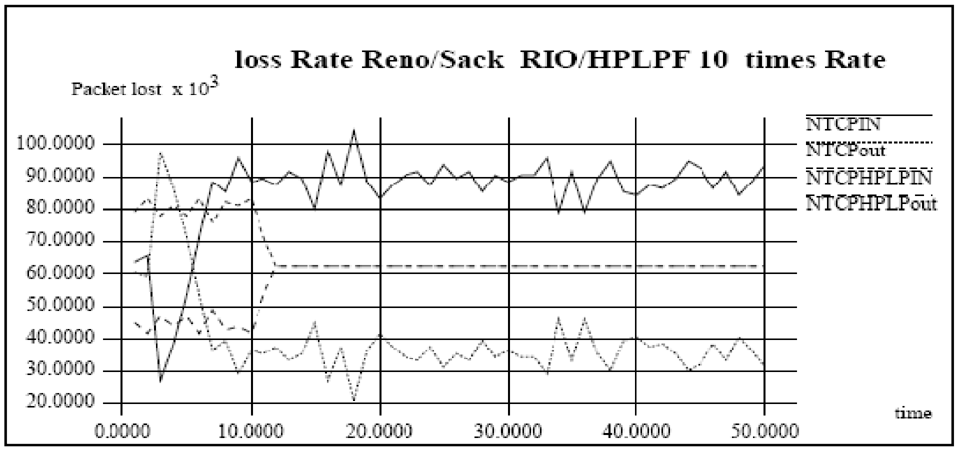

**Figure 13 : loss rate N sack RIO/HPLPF each flow is 10 times greator rate than the previous Sack ( N=6)**

We can see from figure 13 that the two flows in RIO are completely distinguished from each other, but for HPLPF they start to be distinguished then they have the same throughput.

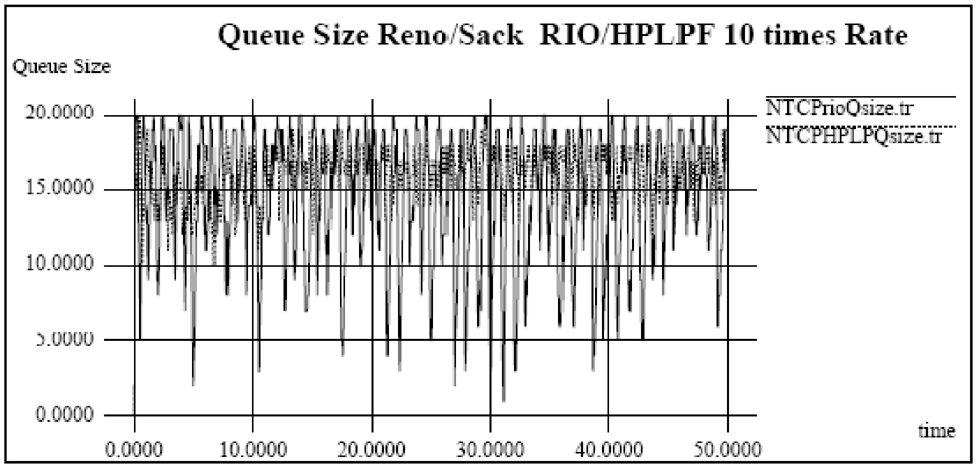

**Figure 14 : Queue N sack RIO/HPLPF each flow is 10 times greater rate than the previous Sack ( N=6)**

According to Queue size we can see from figure 14 that the frequencies of decreasing increased which reflects that the amount of loss will be increased and that is shown in figure 15, and that for the both queues, we can see that the amount of loss start to be less for HPLPF then they nearly converges.

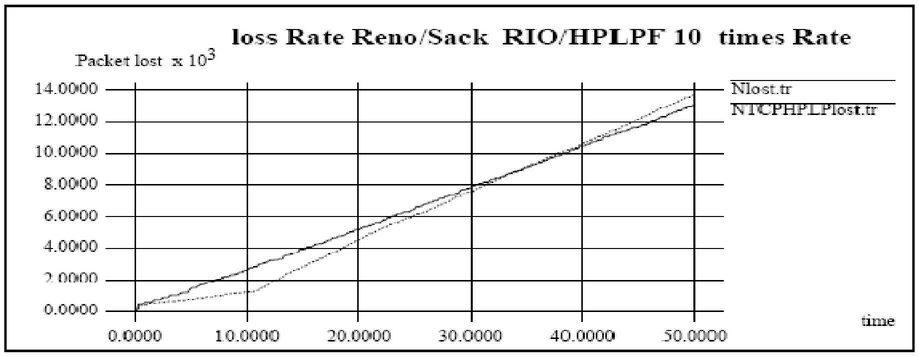

**Figure 15: loss Rate for scenario 2**

- **Simulation Scenario 3 :**

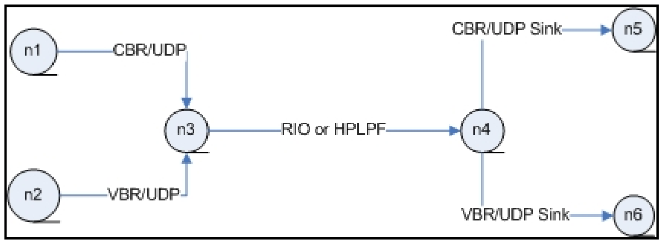

**Figure 16: Network Topology for scenario 3**

- 2 sources CBR and VBR , where CBR is treated as IN , and VBR as Out Traffic ( RIO queue ) , and in the proposed model CBR as HPC traffic and VBR as LPC Traffic , the bottleneck link ( that use RIO or HPLPF ) queues was tested , regarding to queue size, link bandwidth , loss rate , and traffic throughput, the results was drawn using xgraph tool in NS2 where the following graphs where generated :

First of all regarding throughput we can see that the two flows in the two queue management scheme are distinguished from each other with CBR IN for RIO, and HPC for HPLPF, and VBR as Out for RIO and LPC for HPLPF queue, the throughput for the two queue where nearly the same as UDP traffic don't employ any congestion control mechanism , so the fair share that is considered an improvement for HPLPF over RIO will not improve the performance as the two queues will treat the two flows by the same way, under the same parameters and that could be Shown from figure 17 .

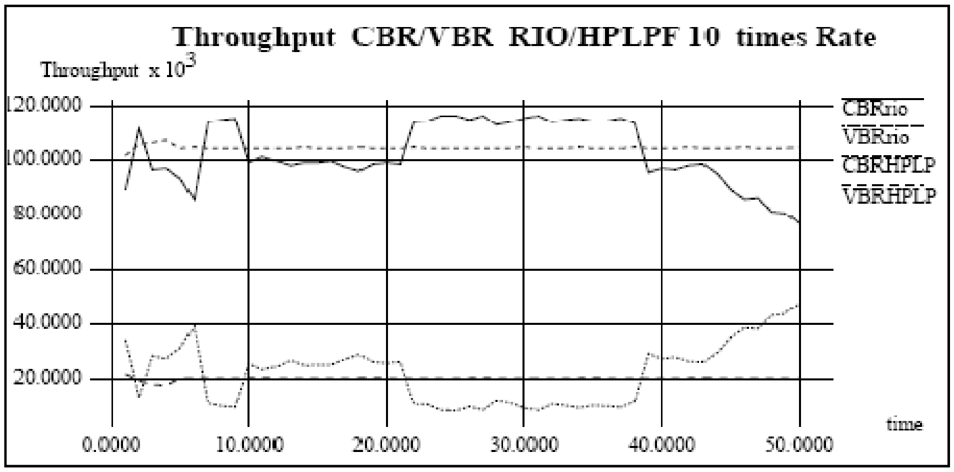

**Figure 17: throughput CBR/VBR RIO/HPLPF for a rate with 10:1**

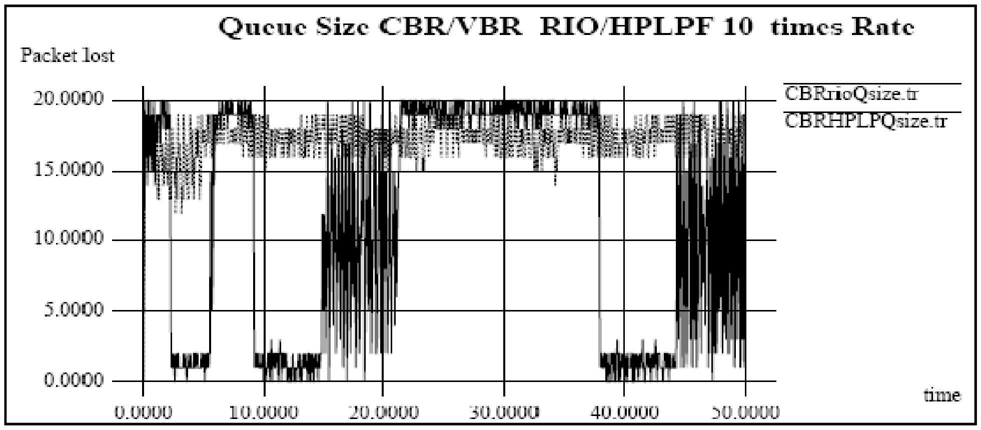

**Figure 18: Queue Size CBR/VBR RIO/HPLPF for a rate of 10:1**

Regarding the loss rate it's nearly the same for the two queues as can be seen from figure 19 , comparing the amount of loss for UDP traffic and that for TCP , UDP traffic have a much higher loss rate than that with TCP sources, as TCP employs congestion control mechanisms that will decrease the amount of packet loss.

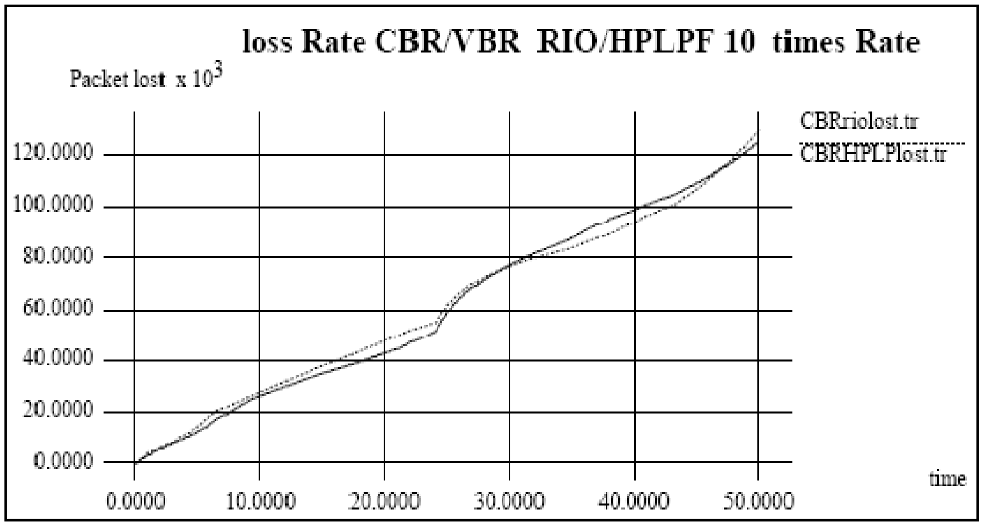

**Figure 19: loss rate CBR/VBR RIO/HPLPF for a rate of 10:1**

- **Simulation Senario4:**

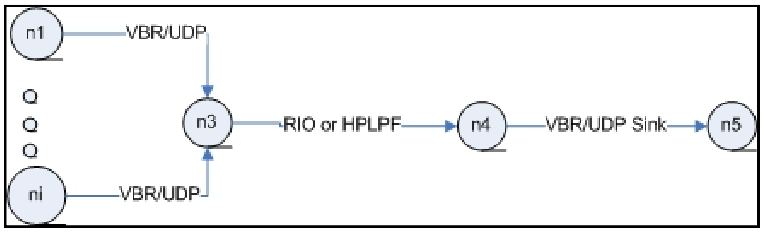

**Figure 20: Network Topology for NVBR scenario 4**

- Using N VBR with traffic policing ( Token bucket Policer ) that shapes the flow Rate for the N VBR flows and marking them as IN or Out for RIO, High , low priority for HPLPF Queue , the generated graphs was as follows :

  Regarding throughput we can see from figure 21, that the throughput decreased for the two type of queues, and that is logical as the amount of flows increased with the same queue size , so the total throughput will decrease, as there will be much more packet loss as we can see from figure 22, where the amount of loss for the tow queues are nearly the same , we can also see that the amount of throughput for the LPC packet was improved over the Out flow for RIO which proves the performance of HPLPF over RIO in providing less starvation for LPC flows.

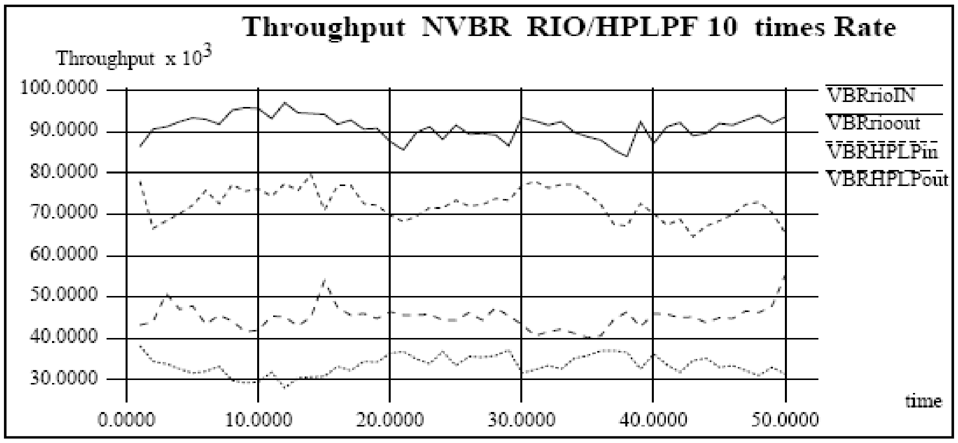

**Figure 21: Throughput NVBR RIO/HPLPF for a rate ratio each flow is 10 times its previous**

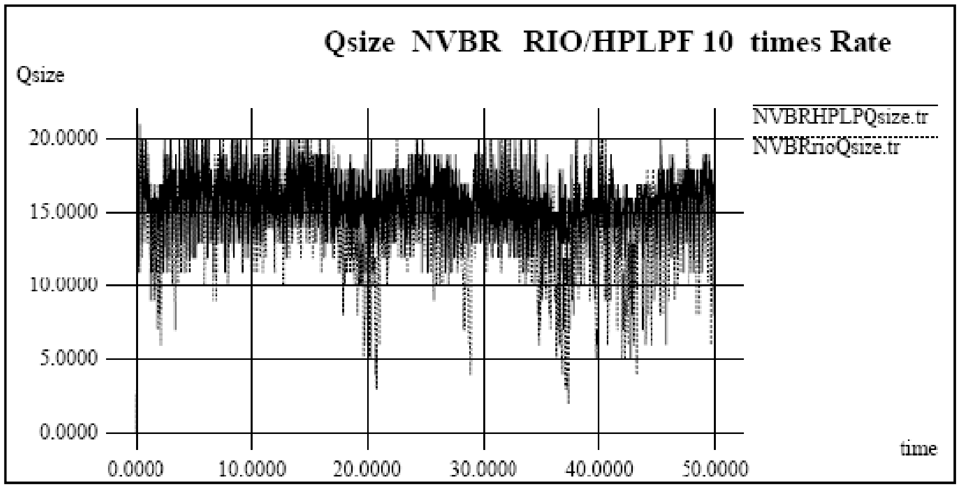

**Figure 22: Queue size NVBR RIO/HPLPF for a rate ratio each flow is 10 times its previous**

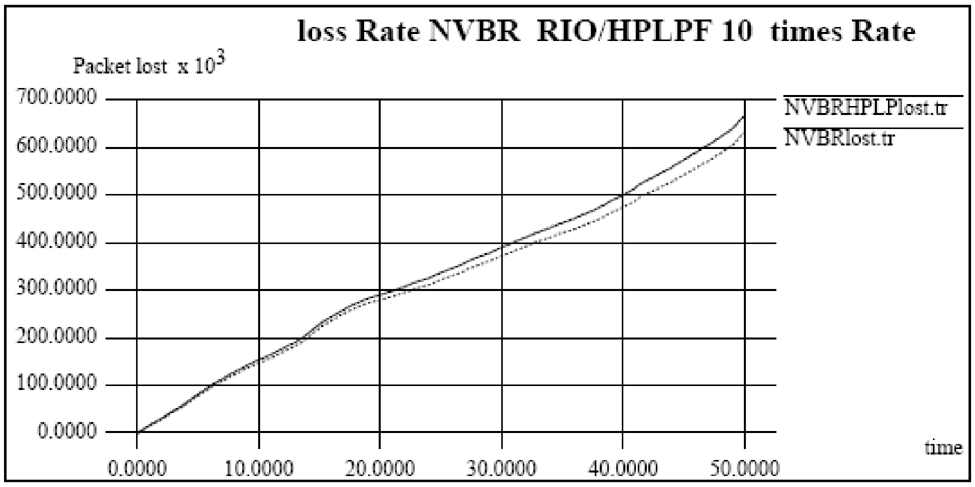

**Figure 23: loss rate NVBR RIO/HPLPF for a rate ratio each flow is 10 times its previous**

## Conclusions and Recommendation

In this Report an implementation for the scheme presented in paper (1) , was done for two type of scheduling FIFO, and RR , then a testing for the implemented scheme were done by comparing the performance of it, with that for RIO queue for different test cases ( different number and type of traffic) , the testing shows that the proposed model was able to achieve a relative differentiation , and improvement in performance in terms of throughput , for the two class of traffic HPC and LPC traffic with decreasing the loss rate comparing it with RIO, at the same time HPC traffic was not starving LPC traffic, the implementation could be extended simply to cover more classes with different scheduling mechanisms to be more efficient and scalable for wide range of traffic classes in the internet, the testing was done for FIFO scheduling, and I think that the performance will be improved by using other scheduling mechanisms such as RR which also being implemented but not tested yet, so the implemented scheme with RR scheduling can be tested with the diffserv model that was developed by Nortel Network (3) which they develop a diffserv model with a RIO queue in the core node that uses different scheduling mechanisms ,those include RR and WRR, the integration of the implemented scheme in this report with that developed in 3 is straightforward with simple modification for the overall diffserv model.

## References:

1. Ahmed E. Kamal a,*, Hossam S. Hassanein b,1, "Performance Evaluation of Prioritized Scheduling with Buffer Management for Differentiated Services Architectures", http://www.cs.queensu.ca/~trl/papers/docs/2004/kh_2004c.pdf
2. Eitan Altman, Tania Jimenez, NS Simulator for Beginners, http://www-sop.inria.fr/maestro/personnel/Eitan.Altman/COURS-NS/n3.pdf
3. Peter Pieda,Jeremy Ethridge, A Network Simulator Differentiated Services Implementation http://www-sop.inria.fr/maestro/personnel/Eitan.Altman/COURS-NS/DOC/DSnortel.pdf
4. Jae Chung **and** Mark Claypool , NS by Example, http://nile.wpi.edu/NS/
5. Marc Greis's tutorial

## Appendix A :

### 1. Sample of the Trace file for HPLPF Queue implementation

```
packet priority  1
inserting HPP int Shared Buffer
 minS = 0.000000
 min = 0.000000 minS= 0.000000 minH = 1000000.000000
minL=1000000.000000  1
Deque from SQ buffer HPQ is Empty
 packet priority  0
inserting LPP int LPQ Buffer
 minL = 10.000000
min = 10.000000 minS= 1000000.000000 minH = 1000000.000000
minL=10.000000  3
Deque from LPP
 packet priority  1
inserting HPP int Shared Buffer
 minS = 3.000000
 min = 3.000000 minS= 3.000000 minH = 1000000.000000
minL=1000000.000000  1
Deque from SQ buffer HPQ is Empty
min = 1000000.000000 minS= 1000000.000000 minH = 1000000.000000
minL=1000000.000000  1
Deque from SQ buffer HPQ is Empty
 packet priority  1
inserting HPP int Shared Buffer
 minS = 6.000000
 min = 6.000000 minS= 6.000000 minH = 1000000.000000
minL=1000000.000000  1
Deque from SQ buffer HPQ is Empty
 packet priority  0
inserting LPP int LPQ Buffer
 minL = 20.000000
min = 20.000000 minS= 1000000.000000 minH = 1000000.000000
minL=20.000000  3
Deque from LPP
 packet priority  1
inserting HPP int Shared Buffer
 minS = 9.000000
 min = 9.000000 minS= 9.000000 minH = 1000000.000000
minL=1000000.000000  1
Deque from SQ buffer HPQ is Empty
min = 1000000.000000 minS= 1000000.000000 minH = 1000000.000000
minL=1000000.000000  1
Deque from SQ buffer HPQ is Empty
 packet priority  0
inserting LPP int LPQ Buffer
 minL = 30.000000
min = 30.000000 minS= 1000000.000000 minH = 1000000.000000
minL=30.000000  3
Deque from LPP
 packet priority  1
inserting HPP int Shared Buffer
 minS = 13.000000
 min = 13.000000 minS= 13.000000 minH = 1000000.000000
minL=1000000.000000  1
Deque from SQ buffer HPQ is Empty
min = 1000000.000000 minS= 1000000.000000 minH = 1000000.000000
minL=1000000.000000  1
Deque from SQ buffer HPQ is Empty
 packet priority  1
```