Projek Ilmiah Tahap Akhir I & II
Session 2003/2004

Implementation of RIO in UMJaNetSim

Name: Chan Chin We
Matrix no: WEK010029
Supervisor: Mr. Phang Keat Keong
Moderator: Mr. Ang Tan Fong

Abstract

As the complexity and diversity of network have grown, network simulator has proven to be an important tool in the design, analysis, testing and performance estimation of network. Besides that, network simulator can also perform a variety of functionalities such as network planning, network monitoring, protocol analyzer and so forth.

In this project, I will do some research on the recent TCP congestion control and avoidance mechanisms and implement one of the mechanisms into the Java Network Simulator (JaNetSim). The current version of JaNetSim has been implemented RED congestion avoidance mechanism. Therefore, this project will focus on the RIO (RED with in and out profile) queue mechanism.

RIO, initially proposed in the context of Differentiated Services networks by distinguish between two classes of packets – in-profile and out-of-profile, which is based on the marking scheme. Besides that, various marking scheme have also been proposed. In this project, I will focus on the earliest making algorithm, which is calculating the average queue sizes for in-profile packets and total packets. Traditionally, packet marking is performed by ingress routers according to a user-profile specified in term of bandwidth.

Acknowledgment

First and foremost, I would like to express my deepest gratitude to my supervisor, Mr. Phang Keat Keong for his guidance, advices and encouragements throughout the entire project. Thanks to Mr. Ang Tan Fung and Mr. Lim for their useful and informative feedbacks to this project.

I please to thank my project group members, Andrew Chiam, Au Yee Boon, Chee Wai Hong, Chia Kai Yan, Lim Lee Wen, Malini, and Tang Geck Hiang for their cooperation and sharing the crucial thought throughout the project. Besides that, special thanks also to the other fellow course mates for sharing their knowledge and ideas with me.

Last but not the least, thanks to my lovely family for their supports and encourages.

Table of Content

List of Figures

List of Tables

Chapter 1
Introduction

Chapter 1: Introduction

1.1 Today's of Internet

The Internet is enabling a global revolution in scholarly communication. For the first time in history, we have an infrastructure that can deliver information any time, any place. This could change our society and global culture as dramatically and unpredictably; as did once the printing press and the automobile.

The Internet began in 1969 as a local network between universities in the United States and ever since has quietly doubled in size each year. By the 1980s, it already spanned the globe, though it remained little known outside of universities. This suddenly changed in 1993, when a new software program called Mosaic let users display text and images from the Internet and explore a "World Wide Web(WWW)" by selecting objects on the screen with a mouse. Millions of people began to access the Web from home. The Internet has continued to double in size every year or so, but the mathematical results of this process have become astonishing; up to 10% of the world's population could soon be online.

As the Internet grew, its organization and management were delegated to the Internet Advisory Board, or (IAB) (refer Figure 1.1). Originally, the IAB consisted of a number of subsidiary organizations, but their main function was to coordinate the Internet task forces. In 1989, the task forces were placed into two major groups within the IAB: Internet Research Task Force (IRTF) and the Internet Engineering Task Force (IETF). The IRTF is responsible for researching problems concerning the TCP/IP (Transmission Control Protocol/Internetworking Protocol) network community and the Internet. The IETF itself is chartered to identify problems and

coordinate problem solving in the areas of Internet management, engineering and operations[1].

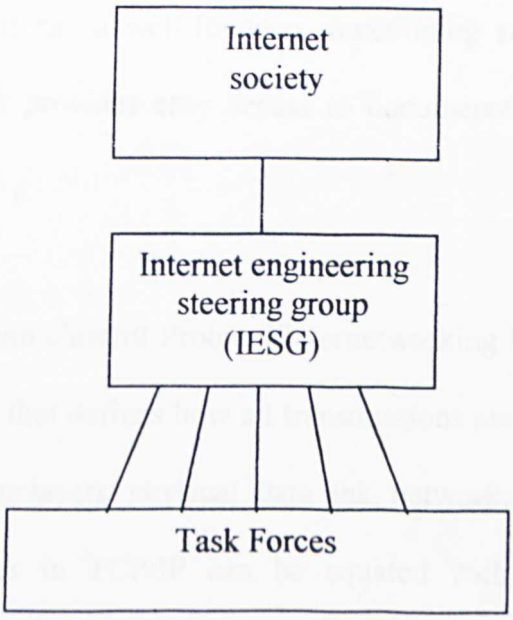


Figure 1.1 The Internet Organization

It is important to understand that the Internet is not a new kind of physical network. It is, instead, a method on interconnecting physical networks and a set of conventions for using networks that allow the computers they reach to interact. While network hardware plays only minor role in the overall design, understanding the Internet technology requires one to distinguish between low-level mechanisms provided by the hardware itself and the higher-level facilities that the TCP/IP protocol software provided.

1.2 Transmission Control Protocol/Internetworking Protocol (TCP/IP)

The use of TCP/IP and related protocols continues to grow, raising some interesting points to the Open System Interconnection (OSI) model. Many people believe that

TCP/IP is a more viable approach for a number of reasons. First, TCP/IP is here and it works. Second, a wealth of products are available that use the TCP/IP protocol suites. Third, it has a well-founded, functioning administrative structure through IAB. Fourth, it provides easy access to documentation. Fifth, it is used in many UNIX products.

The Transmission Control Protocol/Internetworking Protocol (TCP/IP) actually is a set of protocols that defines how all transmissions are exchanged across the Internet. It is made of five layers: physical, data link, network, transport, and application. The application layer in TCP/IP can be equated with the combination of session, presentation, and application layers of the OSI model.

At the transport layer, TCP/IP defines two protocols: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). At the network layer, the main protocol defined by TCP/IP is Internetworking Protocol (IP), although there are some other protocols that support data movement at this layer. At the physical and data link layers, TCP/IP does not define any specific protocol. A network in a TCP/IP internetwork in a TCP/IP can be a local area network (LAN), a metropolitan area network (MAN), or a wide area network (WAN).

Internetwork Protocol (IP) is an unreliable and connectionless datagram protocol – a best-effort delivery service. The term best-effort means that IP provides no error checking or tracking. IP assumes the unreliability of the underlying layers and does its best to get a transmission through to its destination, but with no guarantees.

The Transmission Control Protocol (TCP) provides full transport layer services to applications. TCP is a reliable stream transport port-to-port protocol. The term stream, in this context means connection-oriented; a connection must be established between both ends of a transmission before either may transmit data. Reliability is ensures by provision for error detection and retransmission of damaged frames; all segments must be received and acknowledged before the transmission is considered complete.

The User Datagram Protocol (UDP) is the simpler transport layer protocol. It is an end-to-end transport level protocol that adds only port addresses, checksum error control, and the length information to the data from the upper layer. It does not provide any sequencing or reordering functions and cannot specify the damaged packet when reporting an error. UDP contains only a checksum; it does not contain an ID or sequencing number for a particular data segment.

1.3 Project objective

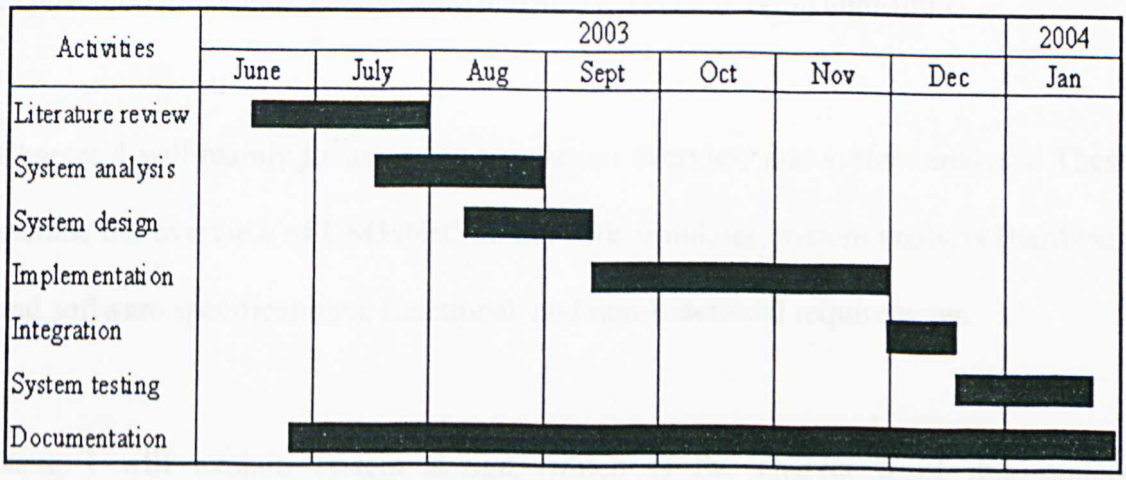
As this is my final year project, the primary objective is to enhance the current version of network simulator, Java Network Simulator (JaNetSim) that developed at the Network Research lab, Faculty of Computer Science and Information Technology, University of Malaya with other new or latest technology. This including add-in new functionalities to existing components, or add-in new components with their corresponding functionalities. This is needed to make the simulator up-to-date with the latest technology before it can perform its functionality to simulate various type of network.

Besides that, the second objective is to study the various of congestion avoidance mechanisms exist nowadays. The congestion avoidance mechanism is important to avoid high link utilization and thus prevents the network bottleneck problem. Congestion avoidance is also useful to stop misbehaving users and ensures fairness to all network users based on their requirements. Meanwhile congestion avoidance involves queue management technique as proposed for use in the differentiated-services initiative of the IETF. The differentiated-services model provides a very simple facility for negotiating and supporting service contracts between service providers and end-users.

1.4 Project scope

- Allow network simulation with RIO feature as a new congestion avoidance mechanism.
- Implement RIO mechanism in UMJaNetSim and allow user to choose the RIO parameters during simulation.
- Show the simulation result and parameters selected.
- Compare the simulation result with the RED.

1.5 Project schedule



1.6 Report organization

The following is the organization of the project documentation.

Chapter 1 will give the introduction of the Internet and also the most widely used internetworking protocol, TCP/IP. Besides that, this chapter also contains the objective, scope and schedule of this project.

Chapter 2 will give the literature review about the topic related with this project. First, I will introduce about the congestion and congestion avoidance. After that, I will discuss about some of the congestion avoidance mechanisms, which is Drop Tail, RED (Random Early Detection), FRED (Fair RED), and WRED (Weighted RED). I also give some introduction to the network simulator including their functionalities.

The next chapter is chapter 3 will discuss about the RIO algorithm in detail. The explanation includes RIO algorithm and also packets marking algorithm.

Chapter 4 will mainly focus on the simulation overview and system analysis. These include the overview of UMJaNetSim network simulator, system analysis (hardware and software specifications), functional, and non-functional requirements.

Next, I will explain system design, which is the chapter 5 of this project documentation. This chapter will further study about the RIO specifications and parameters setting. Besides that, the simulation flow will also being discuss in detail in this chapter.

Later on the chapter 6 will explore on the system implementation. The implementation will focus on the parameters and variables declared in the existing system. The methods defined in the system will also be explained in this chapter.

Chapter 7 is focus on the system testing. The testing is done by creating simulation topology and configures the added parameters before start the simulation. The result of the simulation shown there and explained.

The last chapter, chapter 8 will conclude the implementation of RIO in the simulator. The strengths, limitations and also future work of this implementation are the focus of this chapter.

Chapter 2

Literature Review

Chapter 2: Literature Review

2.1 Congestion Overview

In the Internet, connections share network resources like link-bandwidth, buffers at routers, and so forth. Packets from different connection contend for use of output link at the router, where each of these contending packets is placed in a queue waiting to be transmitted over the output link. When number of such packets contending for the same link increases beyond the buffer capacity, buffer become overflows and packets have to be dropped. When such packet drops become common event, the network is said to be congested.

Meanwhile, in the packet switching networks, there are two common causes of congestion. One is at interface points where a slow output link serves a fast input link. This situation can mostly be found at the edge of the network, when a slow WAN connection serves LAN traffic, or when a shared across network medium is used by a fast traffic source. The other common location of congestion is inside the switching nodes, such as routers. Packets arriving from multiple input ports are often routed to the same output port, creating congestion at the output port even all the links have the same line speed.

In response to congestion, a router is required to notify some connections to back off, i.e., reduce their incoming traffic at that node. Choosing the connections to notify is a difficult task. If all the connections are notified of the congestion then it leads to underutilization of the output link in next phase. Also fairness constraint dictates that such notification policy be uniform over a period of time.

Besides that, Internet traffic is a mix of connections that implement congestion control (like TCP) and ones that do not (like UDP), the router must ensure that at time of congestion, when the responsive flows back off, the non-responsive connections do not get unfair advantage over the responsive flows. The router needs to have mechanism to identify and police such sources.

The degrading effect of congestion on the application performance can be observed by monitoring the throughput and the response time in the network. Throughput indicates the throughput at the TCP layer, and response time is the delay between packet transmission and ACK reception. This well-known throughput/delay behavior of a node under increasing traffic load is sketched in Figure 2.1.

Traffic load is the average packet arrival rate. For light traffic loads the throughput increases linearly with increasing load while delay remains constant. This is a region of low link utilization with no queuing at all. The constant delay equals to the transmission delay between end-points. When the load increases further, a slight increase in the delay is observable due to the onset queuing. The throughput keeps increasing in the region benefiting from the buffering in the node. In this region the maximum link utilization is achieved, and the average delay is only slightly larger than the transmission delay. With further increase of traffic load, the average queue size increases rapidly, and buffer overflows start to take place. This has a degrading effect on both the throughput and the response time. Now the node has entered the state of congestion, and can approach rapidly to a point of zero throughput and infinite response time if the load keeps increasing – the so called "congestion collapse".

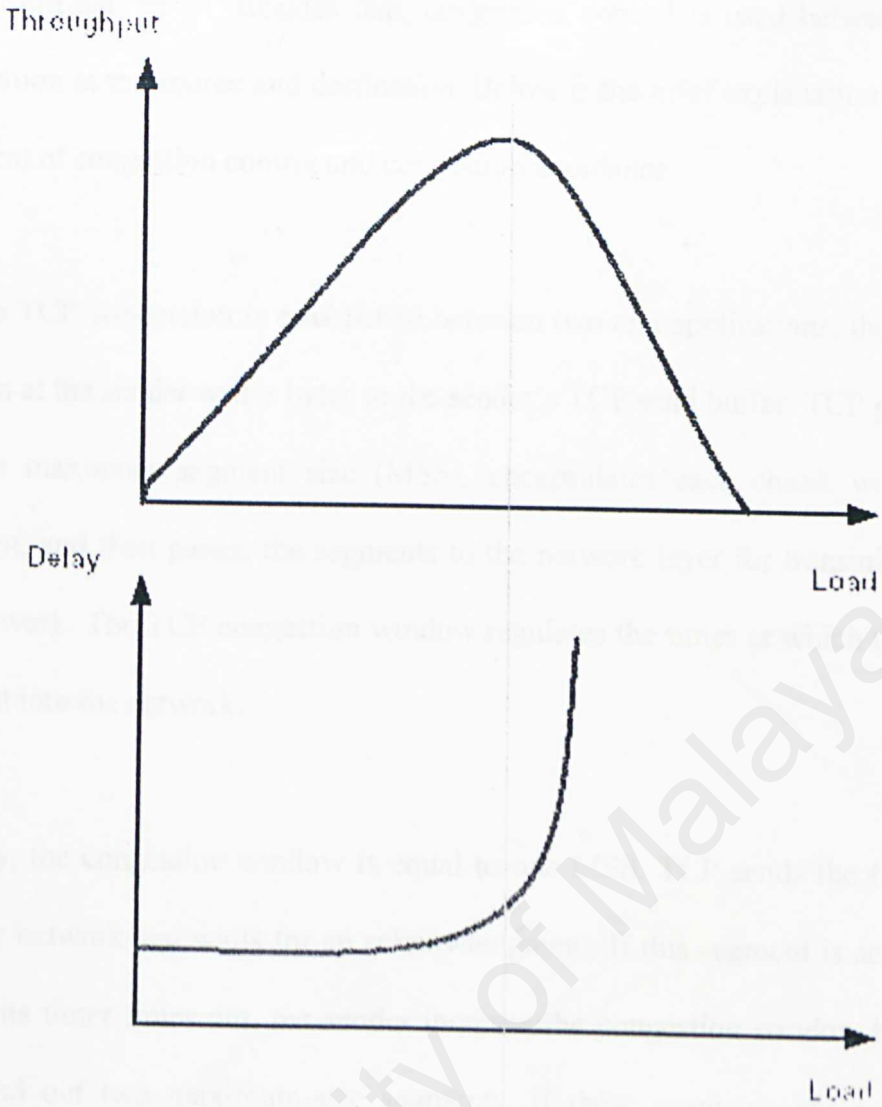


Figure 2.1: Typical behavior of a network under increasing traffic load. The right hand side of the maximum throughput point is considered "congestion region".

2.2 Introduction of Congestion Avoidance

Congestion avoidance techniques monitor network traffic loads in an effort to anticipate and avoid congestion at common network bottlenecks. Congestion avoidance is achieved through packet dropping. It is different from the congestion control mechanisms now exist in the Internet nowadays such as Slow Start, Fast

Retransmit and so on. Besides that, congestion control is used between two TCP application at the source and destination. Below is the brief explanation between the different of congestion control and congestion avoidance.

Once a TCP connection is established between two end applications, the application process at the sender writes bytes to the sender's TCP send buffer. TCP grabs chunks of size maximum segment size (MSS), encapsulates each chunk within a TCP segment, and then passes the segments to the network layer for transmission across the network. The TCP congestion window regulates the times at which the segments are sent into the network.

Initially, the congestion window is equal to one MSS. TCP sends the first segment into the network and waits for an acknowledgment. If this segment is acknowledged before its timer times out, the sender increase the congestion window by one MSS and send out two maximum-size segments. If these segments are acknowledged before their timeouts, the sender increases the congestion window by one MSS for each acknowledged segments, giving a congestion window of four MSS, and sends out four maximum-sized segments. This procedure continues as long as the congestion window is below the threshold and the acknowledgment arrive before their corresponding timeouts.

During this phase of the congestion control procedure, the congestion window increases exponentially fast, i.e., the congestion window is initialized to one MSS, after one round-trip times (RTT) the window is increased to two segments, after two RTT the window is increased to four segments, after three RTT the window size is

increased to eight segment and so on. This phase of the algorithm is called Slow Start because it begins with a small congestion window equal to one MSS and then accelerates rapidly.

The slow start phase ends when the window size exceeds the value of certain threshold. Once the congestion window is larger than the current value of the threshold, the congestion window grows linearly rather exponentially. This has the effect of increasing the congestion window by one in each RTT for which the entire window's worth of acknowledgments arrives. This phase of the algorithm is called congestion avoidance.

The congestion avoidance phase continues as long as the acknowledgments arrive before their corresponding timeouts. But the window size, and hence the rate at which the TCP sender can send, cannot increase forever. Eventually, the TCP rate will be such that one of the links along the path becomes saturated, and which point loss (and a resulting timeout at the sender) will occur. When a timeout occurs, the value of threshold is set to half the value of the current congestion window, and the congestion window is reset to one MSS. The sender then again grows the congestion window exponentially fast using the slow start procedure until the congestion window hits the threshold.

Congestion avoidance takes care of violation [2]. Lost packets are a good indication of congestion on the network. In the current Internet, the TCP transport protocol detects congestion only after a packet has been dropped from the gateway. However, it would clearly be undesirable to have large queues that were full much of the time;

this would significantly increase the average delay in the network. Therefore, with increasingly high-speed networks, it is increasingly important to have mechanisms that keep throughput high but average queue sizes low.

The throughput/delay behavior of a node under varying traffic load is extremely useful to define the difference between controlling the congestion and avoiding it. A congestion control system, like the one implemented in the TCP, prevents the network from falling into the congestion collapse state, by reducing the load after the congestion region is entered. This mechanism has been serving the Internet for two decades.

The IP network does not provide any state information to the user, e.g. black box assumption. Therefore, TCP needs to rely on packet loss or extreme delays to figure out that the load in the network reached a critical point, and that the transmission windows should be reduced to avoid congestion collapse. On the other hand, TCP tends to increase the load if there are no indications of congestion, in order to achieve the maximum utilization it can. The resulting behavior of the state of a node under TCP load is one that swings between the regions of less than maximum utilization and congestion (Figure 2.2).

Congestion avoidance, however, aims to maintain a stable state at the point with maximum throughput and minimum delay. It tries to prevent entering a congestion state in the first place, instead of entering and escape constantly. Congestion avoidance would minimize packet loss, and keep the average queue size low. In order to achieve this proactive operation, the congestion avoidance system requires

the cooperation of the network. Most specifically, three components are required to complement a congestion avoidance system:

i. A congestion detection mechanism in the network

ii. A congestion notification system

iii. A congestion control mechanism

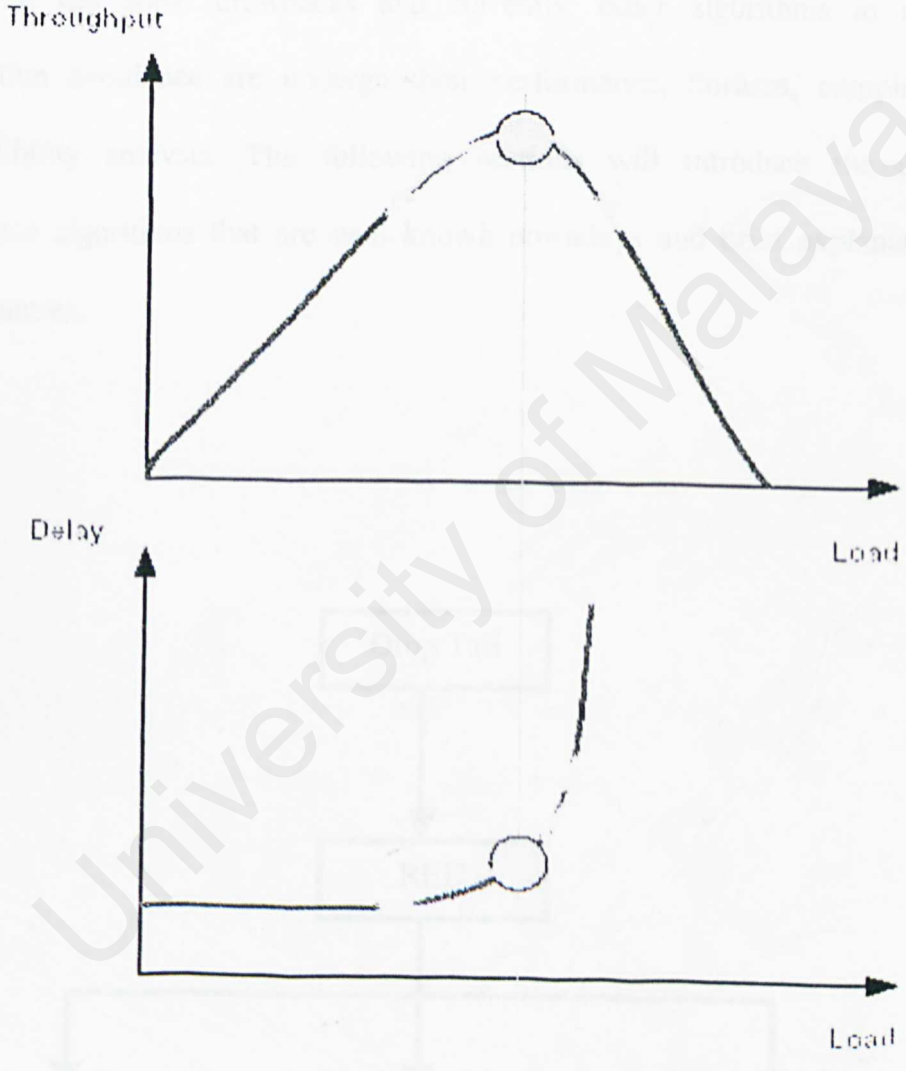


Figure 2.2: TCP congestion control prevents the network from entering the congestion collapse region (zero throughput and infinite delay). But the state of the network swings back and forth between congestion and low utilization. Congestion avoidance aims to maintain at stable operation point at high utilization and low delay.

In this few years, the computer scientists around the world have proposed several methods as congestion avoidance algorithms. The first and standard algorithm is based on the first-in-first-out (FIFO) principle. When the average queue size exceeds certain threshold, the packets are dropped based on the time of the packet being in the queue. The packet comes in last the queue will be dropped first. Meanwhile, this algorithm has some drawbacks and currently, other algorithms to implement congestion avoidance are undergo their performance, fairness, complexity, and compatibility analysis. The following sections will introduce the congestion avoidance algorithms that are well known nowadays and brief explanation about their features.

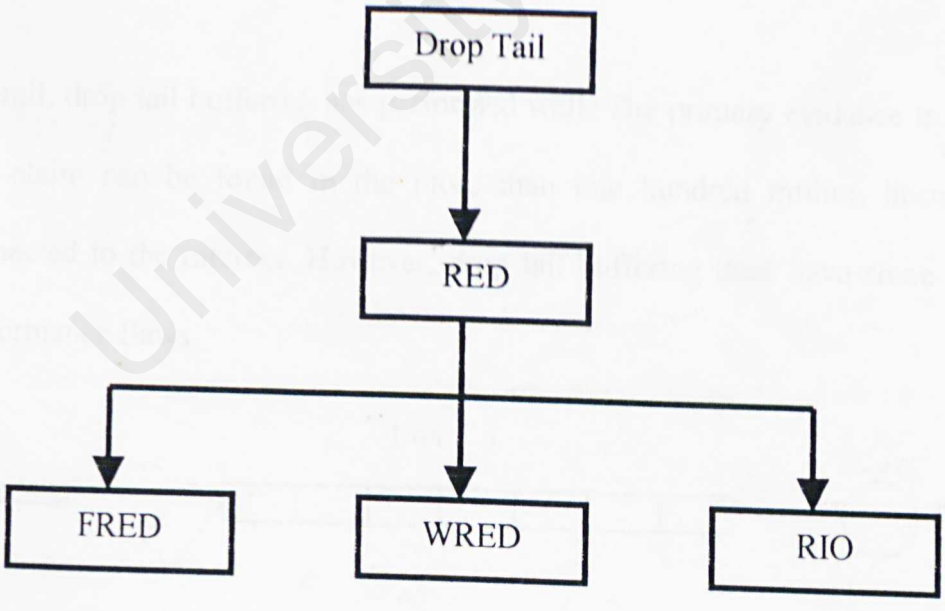


Figure 2.3: Evolution of congestion avoidance mechanisms

2.3 Drop Tail

The default queuing behavior in Internet routers has long been drop tail. This algorithm is using the First-In-First-Out (FIFO) policy, which will start drop from tail when the queue is full. With this scheme packets are enquired at the tail of a queue as they arrive and desuetude from the head of the queue when there is capacity on the link.

There was no special rationale in providing these queuing semantics. There is simply the default behavior of a finite capacity queue. The focus when these queues were added to routers was simply to have a queue of some form to provide buffering associated with the outbound link. The buffer is there simply to accommodate transient overloads that result from the bursty nature of the Internet. The general rule of thumb in determining the buffer's capacity was simply to allocate space equivalent to two or four times the delay bandwidth.

Overall, drop tail buffering has performed well. The primary evidence in support of this claim can be found in the more than one hundred million hosts currently connected to the Internet. However, drop tail buffering does have some significant performance flaws.

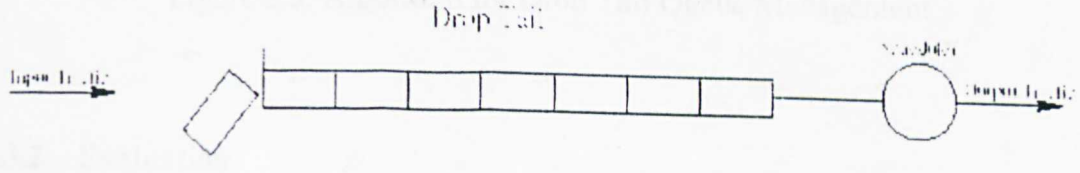


Figure 2.4: Drop Tail

2.3.1 Algorithm Description

The algorithm for implementing a drop tail FIFO queue is a simple modification of the standard queue as a linked list in data structure. When a packet arrives, if the queue is not full, the packet is enqueued. If the queue is full, the packet is dropped. When a packet departs, it is dequeued and sent. The enqueue and dequeue functions are expected to update the value of q_len (queue length). The code is shown in the figure below:

```
Constants:
int q_limit;                    //upper limit on queue size

Variables:
int q_len = 0;                  //current queue occupancy

General functions:
void enqueue(P);                //enqueue P and update q_len
Packet dequeue();               //dequeue p and update q_len
void send(P);                   //transmit P
void drop(P);                   //discard P

for each arriving packet P {
  if( q_len < q_limit )         //if queue is not full
    enqueue(P);
  else
    drop(P);
} //end for each arriving

for each departing packet {
  P = dequeue();                //dequeue and send the packet
  Send(P);
} //end for each departing
```

Figure 2.5: Algorithm for Drop Tail Queue Management

2.3.2 Evaluation

FIFO scheduling accomplishes its major design goal: accommodating bursty traffic. However, it suffers from problems of lock-out and full queues [3]. Lock-out refers to a phenomenon in which the shared resource, link bandwidth, is unfairly consumed

exclusively by a small number of flows. The remaining flows are locked-out of (i.e., denied access to) the queue and, consequently, locked-out of the outbound link. In this phenomenon, the queue is occupied only by the packets from a small number flows while the packets associated with most flows are consistently discarded.

As a result, most flows receive none of the link bandwidth, and starve. This phenomenon occurs because of timing effects which result in some flow's packet always arriving to find the queue full. For example, consider a situation where many sources are periodically sending bursts of packets that in aggregate exceed the queue's capacity. If these sources become synchronized, all sending nearly simultaneously, the first packets to arrive (e.g. from the source closer to the bottleneck link) will find a queue with some available capacity while the subsequent packets will be discarded. If the same relative order is maintained between the sources, those sources that send first will consistently make progress while the other flows will consistently have all packets discarded and thus, starve.

Full queues are queues that are usually occupied to capacity. If bursts of packets arrive to a full queue, many packets are dropped simultaneously. This can lead to large oscillations in the network utilization. If the dropped packets are from different flows there may be synchronized responses (back-off) among multiple flows. Synchronized back-off is a phenomenon in which many sources simultaneously receive congestion notification and reduce their generated load. As a result, the overall load on the network may drop below the capacity of the link and then rise back to exceed the link's capacity resulting in a full queue and once again leading to

simultaneous drops. This oscillating behavior is exactly counter to the buffer's intended function, acting as a smoothing filter.

Note that full queues and long queues are not necessarily equivalent. A long queue is one containing a large number of packets, regardless of capacity; while a full queue is one containing the maximum number of packets it can hold. Long queues cause problems because of queue-induced latency. However, the fact that a queue is long says nothing about its available capacity. A high capacity queue may have many packets enqueued but still have a great deal of unused capacity, leaving it with capacity to accept newly arriving packets and allowing it to perform its intended function of accommodating packet bursts. While full may also be long queues, the length of the queue is not the primary problem with full queues. A queue that is usually full is not able to perform its primary function of accommodating bursts.

2.4 RED

For solving the problems of conventional Drop Tail routers, researches on AQM (Active Queue Management) mechanisms have been performed actively in the last few years [4]. AQM mechanism controls the queue length (i.e., the number of packets in the router's buffer) by actively discarding arriving packets before the router's buffer becomes full. For instance, one type of AQM mechanism called Random Early Detection (RED) was proposed by Floyd and Jacobson [5] as an effective mechanism to control the congestion in the network routers or gateways. It also helps to prevent the global synchronization in the TCP connections sharing a

congested router and to reduce the bias against bursty connections. Currently it has been implemented in various routers and has been recommended for queue management by IETF.

The RED approach does not possess the same undesirable overhead characteristics as some non-FIFO queuing techniques (e.g. simple priority queuing, class based queuing, weighted fair queuing fair queuing). With RED, it is simply a matter who gets into the queue in the first place – no packet reordering or queue management takes place. When packets are placed into the outbound queue, they are transmitted in the order, which they are queued.

While the principles behind RED gateways are fairly general and RED gateways can be useful in controlling the average queue size even in the network where the transport protocol cannot be trusted to be cooperative, RED gateways are intended for a network where the transport protocol responds to congestion indications from the network. The gateway congestion control mechanism in RED gateways simplifies the congestion control job required of the transport protocol, and should be applicable to transport layer congestion control mechanisms other than the current version of TCP, such as protocols with rate-based rather than window-based flow control.

However, some aspects of RED gateways are specifically targeted to TCP/IP networks. The RED gateway is designed for a network where a single marked or dropped packet is sufficient to signal the presence of congestion to the transport layer protocol.

The RED congestion control mechanisms monitor the average queue size for each output queue and using randomization, choose connections to notify of that congestion. Transient congestion is accommodated by a temporary increase in the queue. Longer-lived congestion is reflected by an increase in the computed average queue size, and results in randomized feedback to some of the connections to decrease their windows. The probability that a connection is notified of congestion is proportional so that connection's share of the throughput through the gateway.

Gateways that detect congestion before the queue overflows are not limited to packet drops as the method for notifying connections of congestion. RED gateways can mark a packet by dropping it at the gateway by setting a bit in the packet header, depending on the transport protocol. When the average queue size exceeds a maximum threshold, the RED gateway marks every packet that arrives at the gateway. If RED gateways mark packets by dropping them rather than by setting a bit in the header when the average queue size exceeds the maximum threshold, the RED gateway controls the average queue size even in the absence of a cooperating transport protocol.

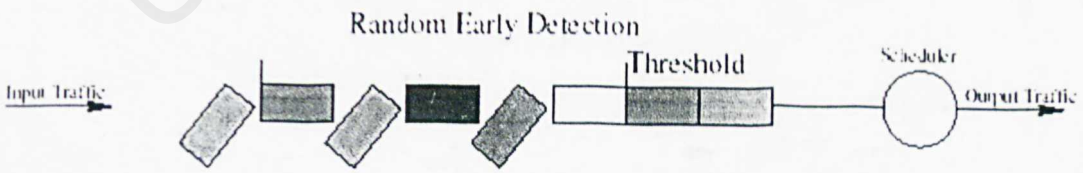


Figure 2.6: Random Early Detection

2.4.1 Algorithm Description

The RED gateway calculates the average queue size using a low pass filter with an exponential weighted moving average. The average queue size is compared to two thresholds: a minimum and a maximum threshold. When the average queue size is less than the minimum threshold, no packets are marked. When the average queue size is greater than the maximum threshold, every arriving packet is marked. If marked packets are, in fact, dropped or if the all source nodes are cooperative, this ensures that the average queue size does not significantly exceed the maximum threshold.

A RED router is configures with the following parameters: min_{th}, max_{th}, and P_{max}. It works as illustrated in the figure 2.7 – the x-axis is avg, the average queue size, and the y-axis is the probability of dropping an arriving packet. There are three phases in RED, defined by the average queue size in the range of [0, min_{th}), [min_{th}, max_{th}), [max_{th}, ∞), respectively. The three phase are normal operation. Congestion avoidance and congestion control.

Figure 2.7: RED drop probability

During the normal operation phase, when the average queue size is below min_{th}, the router does not drop any packets. When the average queue size is between the two thresholds, the router is operating in the congestion avoidance phase, and each packet

drop serves the purpose of notifying the end-host transport layer to reduce its sending rate. Therefore, the dropping probability is a fraction of P_{max}, and is usually small. When the average queue size is above min_{th}, the router drops every arriving packet, hoping to maintain a short queue size.

Initialization:

avg = 0
count = -1
for each packet arrival
calculate the new average queue size avg:
if the queue is nonempty
*avg = (1 - wq)*avg + wq q*
else
m = f(time – q_time)
*avg = (1 - wq)^m * avg*
if $min_{th} < avg < max_{th}$
increment count
calculate probability pa:
$pb = max_p (avg - max_{th})/(max_{th} - min_{th})$
$pa = pb/(1 - count \cdot pb)$
with probability pa:
mark the arriving packet
count = 0
else if $max_{th} < avg$
mark the arriving packet
count = 0
else count = -1
when queue becomes empty
q_time = time

Variables:

avg: average queue size
q_time: start of the queue idle time
count: packets since last marked packet

Fixed parameters:

wq: queue weight
min_{th}: minimum threshold for queue
max_{th}: maximum threshold for queue
max_p: maximum value for pb

Figure 2.8: Algorithm for RED gateway (Part 1 of 2)

Other:

pa: current packet-marking probability
q: current queue size
time: current time
f(t): a linear function of the time t

Figure 2.8: Algorithm for RED gateway (Part 2 of 2)

As average queue size avg varies from min_{th} to max_{th}, the packet-marking probability *pb* varies linearly from 0 to max_p:

$$pb = max_p*(avg - min_{th})/(max_{th} - min_{th}).$$

If we directly use this formula we find the inter-marking interval is not a uniform random variable, to do this. We apply following formula to Pb

$$pa = pb/(1 - count * pb)$$

The final packet-marking probability pa increases slowly as the count increases since the last marked packet. This ensures that marking of packets is fairly uniform

One option for the RED gateway is to measure the queue in bytes rather than in packets. With this option, the average queue size accurately reflects the average delay at the gateway. When this option is used, the algorithm would be modified to ensure that the probability that a packet is marked is proportional to the packet size in bytes:

$$pb = pb\ PacketSize/MaximumPacketSize$$

$$pa = pb/(1 - count*pb)$$

In this case a large FTP packet is more likely to be marked than is a small TELNET packet.

If the queue remains empty for a large period of time, the old value of avg should have a lesser share in new value of queue average size, RED ensures that with the following equation.

$$avg = (1\text{-}wq)^m * avg$$

where m denotes the time for which queue has remained idle.

2.4.2 Evaluation

The RED algorithm meets its design goals, avoiding the problems of lock-out and full queues. It also provides effective feedback to responsive flows. It is also worthwhile to note that RED can be deployed without changes to existing protocols or infrastructure beyond the single router that is being added to. As a result, it can be gradually deployed into the Internet. The following describe the goals that have been met by RED gateway:

- **Congestion avoidance** – if the RED gateway in fact drops packets that arriving at the gateway when the average queue size reaches the maximum threshold, then the RED gateway guarantees that the calculated average queue size does not exceed the maximum threshold. If the RED gateway sets a bit in packet headers when the average queue size exceeds the maximum threshold rather than dropping packets, the RED gateway relies on the cooperation of the sources to control the average queue size.
- **Appropriate time scales** – after notifying a connection of congestion by marking a packet, it takes at least a round-trip time for the gateway to see a reduction in the arrival rate. In RED gateway, the time scale for the detection of congestion roughly matches the time scale required for connections to

respond to congestion. RED gateways do not notify connections to reduce their windows as a result of transient at the gateway.

- **No global synchronization** – the rate at which RED gateways mark packets depends on the level of congestion. During low congestion, the gateway has a low probability of marking each arriving packet and, as congestion increases, the probability of marking each packet increases. RED gateways avoid global synchronization by marking packets as low a rate as possible.
- **Simplicity** – the RED gateway algorithm could be implemented with moderate overhead in current network.
- **Fairness** – the RED gateway does not discriminate against particular connections or classes of connections (this is contrast to Drop Tail). For the RED gateway, the fraction of marked packets for each connection is roughly proportional to that connection's share of the bandwidth. RED gateways provide a mechanism to identify the level of congestion, and could also be used to identify connections using large share of the total bandwidth.
- **Appropriate for wide range of environments** – the randomized mechanism for marking packets is appropriate for networks with connections with a range of round-trip times and throughput, and for a large range in the number of active connections at one time. Changes in the load are detected through changes in the average queue size, and the rate at which packets are marked is adjusted correspondingly.

However, RED is still vulnerable to misbehaving flows. Although a RED router that uses drops as its notification method can constrain unresponsive flows somewhat, the

unresponsive flows will still force the responsive flows to reduce their load to near zero.

RED is unfair to low speed connections. When the high speed is reached, RED drops packets randomly, and probably the dropped packet belongs to a connection that is using less resources than its fair share. The same happens when the average queue length is at fixed point within the two thresholds since all incoming packets are dropped with the same probability. In addition, RED cannot deal with misbehaving users. Even though the maximum threshold is enforced, if the user does not decrease its rate, RED does not have any mechanisms to protect other users. Instead, the misbehaving users could end up getting the whole bandwidth. Finally, TCP congestions with larger window sizes and smaller round-trip times usually consume more bandwidth.

Besides that, the RED mechanism is extremely sensitive to changes to values of the RED parameters and the performance of RED largely depends on the fine-tuning of these parameters. Although these parameters allow flexibility to the network managers to adapt to the network conditions, the optimal values of the parameters for which the network performance is maximum are hard to predict. No rules are known that can guide us in setting these parameters. Worse still a set of parameters that are known to perform well for a particular type of traffic may give bad performance if the traffic pattern changes. Setting these the RED parameters is still remains an inexact science.

2.5 FRED

One of the most important features of RED is that the fact RED provides fairness by dropping packets according to the connection's share of the bandwidth. However, RED still suffers from several other fairness problems. Experiments with a mixture of sources with short and long round-trip times (RTT) as well as adaptive TCP sources and non-adaptive constant bit rate sources (sources that do not react to congestion indications) demonstrate that low RTT connections get more bandwidth than long RTT connections, and that RED is ineffective in handling non-adaptive sources.

Flow RED or FRED [6] solves these fairness issues by maintaining thresholds and buffer occupancies for each active flow (per-connection information). Two thresholds are used per connection in order to guarantee a minimum buffer space as well as to control misbehaving users. This is how FRED protects long RTT connections against low RTT connections and adaptive sources against non-adaptive sources. In addition, two global thresholds are also used, as in FRED, in order to keep the global average queue length within the desired limits. Solving these fairness issues in not free. FRED needs to keep per-flow information to drop packets accordingly and achieve its goals, and this is a costly operation for the routers. FRED must identify every flow that has packets in the buffer and update the information of those flows on a per-packet basis. In order to do so, FRED must look at the packet source and destination addresses, port numbers, and protocol ID of every packet.

Further FRED identifies non-responsive aggressive connections, and penalizes such connections by allowing only the such connections to buffer just their fair share of

bandwidth, i.e. bursts from such connections are not be accommodated by the router and such burst packets will be dropped. On other hand if a connections has been responsive then, even if the connection has already used its fair share, the next incoming packet from that connection is not deterministically dropped but is given a random dropper where it is probabilistically dropped, depending upon the average queue length. Hence a burst of packet from such responsive connection would be accommodated in the queue. This behavior works as a sort of incentive for connections to be responsive to packet drop, and helps avoid congestion. FRED addresses the problems of very small connections like telnet that have very small data to send and hence use very less than their share of the available bandwidth. Also such connections do not have data always ready to send. FRED has provision for such connections. FRED always allows packets from such connections unless the queue as exceeded its max threshold.

FRED as it just builds on RED, continues to have its improvement over Drop-Tail and is also able to achieve isolation, protection albeit at cost of added complexity of per-active flow accounting. Like RED, FRED does not make any assumptions about queuing architecture, it will work with a FIFO gateway.

2.5.1 Algorithm Description

Apart from normal RED variables like average queue size avg, min_{th} and max_{th} thresh-hold, max_q, FRED also uses following variables:

i. max_q and min_q : These per flow variables refer to the maximum and minimum number of packets a connection would be allowed to have in the buffer.

ii. avg_{cq}: A number representing the fair buffer share of a connection at the router.

iii. $qlen_i$: A per-flow variable representing the number of packets a connection has currently enqueued in the buffer.

iv. $strike_i$: A variable is used to identify unresponsive connections. An unresponsive connection has non-zero value for this variable. An unresponsive connection is one that tries to exceed the limit of "max_q" number of packets.

```
for each arriving packet P
  if P is from a flow i that does not have packets in the queue
        //initialize the connections per-flow variables
        qleni = 0;
        strikei = 0;

//identify and manage non-adaptive flows
//The incoming packet would be deterministically dropped and its flow
marked as non-responsive under following cases

if ( qleni >= maxq ||    //the connection has exceeded its maximum limit
   (avg >= maxth && qleni > 2*avecq) || (qleni >= avecq && strikei > 1)
)

//(The connection is already a non-responsive connection and hence all its
above its fair share are dropped. (note this limit for responsive connection
is twice the fair share)

strikei ++;
drop packet P;
return;    //drop packet and return no further processing required
```

Figure 2.9: Algorithm for FRED (part 1 of 2)

```
//queue size is between max_th and min_th, operate in random drop mode:
if (min_th <= avg < max_th)
( //subject to random drop only if connection is not a very small connection
(like telnet) else packet is unconditionally accepted
        if (qlen_i >= MAX(min_q, avg_cq))
        {
          calculate drop probability pa ;
          with probability pa;
          drop packet P;
          return;
        }

//The average queue size is less than minimum threshold, hence accept the
incoming packet
        else if ( avg < min_th )
        {
          count = -1;
          accept the incoming packet
        }

//The average queue size has exceeded the max_th and hence drop incoming
packet
        else
        {
          drop packet P;
          return;
        }
```

Figure 2.9: Algorithm for FRED (part 2 of 2)

2.5.2 Evaluation

FRED seeks to provide fairness between flows. It does this by isolating flows from one another by maintaining per-flow statistics and constraining those flows that consume more than their fair share. The results of this approach can be compared to the results with FIFO and RED using the TCP throughput graph as shown in figure below.

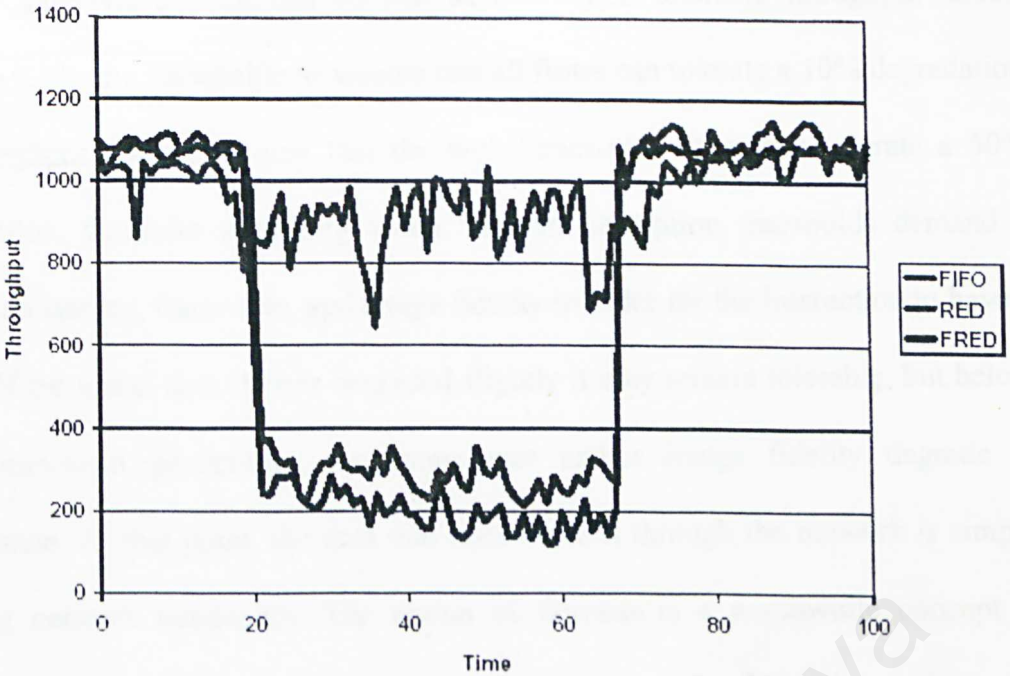


Figure 2.10: Aggregate TCP Throughput (kb/s) vs. Time (second) in the presence of an unresponsive, high-bandwidth UDP flow for FIFO, RED and FRED.

The misbehaving blast is active during time [15, 70] second. While there is some decrease in TCP throughput, the overall performance is much better than that seen when simply using RED or FIFO. FIFO has the lowest throughput during the blast while the throughput for RED is slightly better and FRED is much better. There is no congestion collapse.

One point to debate with FRED is the definition of fair. RED attempt to be fair by assuring that all flows received feedback and were constrained in proportion to their actual packet arrive rate. RED attempted to insure that the throughput of all flows was degraded by the same percentage by dropping the same percentage of packets for all flows. However, FRED attempts to insure that all flows are constrained to roughly equal shares of the link's capacity. This fails to consider the fact that flows

are associated with applications that may have minimum tolerable throughput values. Perhaps it is more reasonable to assume that all flows can tolerate a 10% degradation in throughput than to assume that the high bandwidth flows can tolerate a 50% degradation. Consider streaming video. Human perception thresholds demand a minimum latency, frame rate, and image fidelity in order for the interaction to have a value. If the initial data flow is degraded slightly it may remain tolerable, but below some minimum packet-rate, the frame rate and/or image fidelity degrade to uselessness. At that point, the data that does make it through the network is simply wasting network bandwidth. The notion of fairness is a worthwhile concept to explore. However, it may be necessary to introduce flexibility by making the thresholds configurable instead of simply trying to offer equal shares.

In summary, FRED offers good TCP performance and constrains misbehaving flows well. However its notion of fairness by allocating all flows an equal share may be too inflexible. FRED is noteworthy for its application of the queue management on a per-flow basis, for its identification of robust and misbehaving flows, and for its introduction of the concept of fairness to queue management.

2.6 WRED

Actually, nowadays, there are many version of RED have been proposed as an enhancement of the RED algorithm to overcome its drawbacks. WRED – Weighted RED, is a congestion avoidance algorithm that randomly drops packets when congestion is detected prior to the interface overflowing completely and causing uncontrollable dropped packets.

The random early detection (RED) algorithms are designed to avoid congestion in internetworks before it becomes a problem. RED works by monitoring traffic load at points in the network and stochastically discarding packets if the congestion begins to increase. The result of the drop is that the source detects the dropped traffic and slows its transmission. RED is primarily designed to work with TCP in IP internetwork environments.

WRED combines the capabilities of the RED algorithm with IP precedence [7]. This combination provides for preferential traffic handling for higher-priority packets. It can selectively discard lower-priority traffic when the interface starts to get congested and can provide differentiated performance characteristics for different classes of service. Therefore, WRED allows the reading of the Type of Service (ToS) in the IP packets and as such preference is given to the type of packets to be dropped during congestion.

WRED is useful on any output interface expected to have congestion. WRED is usually used in the core routers of a network, rather than the edge. Edge routers assign IP precedence to packets as they enter the network. The precedence is defined in the ToS field (8 bits) by changing it to Precedence (3 bits) and Type of Service (4 bits) fields.

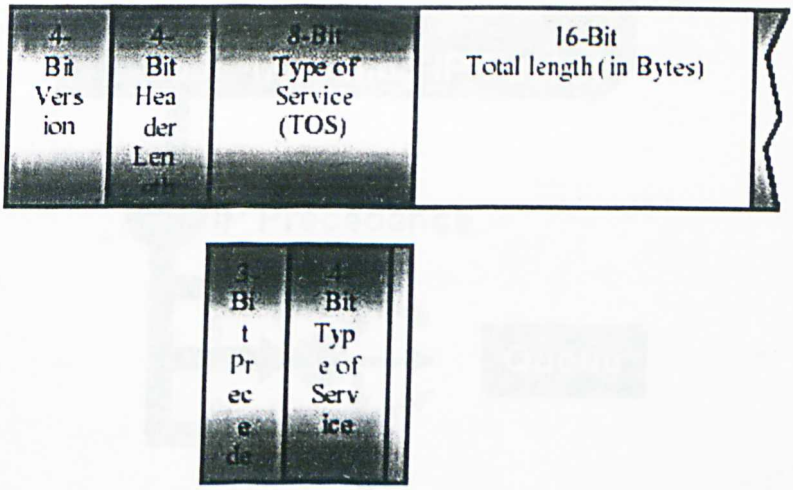


Figure 2.11: IP Header

Within each queue, a finite number of packets can be housed. A full queue causes tail drops. Tail drops are dropped packets that could not fit into the queue because the queue was full. This is undesirable because the packet discarded may have been a high-priority packet and the router did not have a chance to queue it. If the queue is not full, the router can look at the priority of all arriving packets and drop the lower-priority packets, allowing high-priority packets into the queue. Through managing the depth of the queue (the number of packets in the queue) by dropping various packets, the router can do its best to make sure that the queue does not fill and that tail drops are not experienced. This allows the router to make the decision on which packets get dropped when the queue depth increases. WRED also helps prevent overall congestion in an internetwork. WRED uses a minimum threshold for each IP precedence level to determine when a packet can be dropped. (The minimum threshold must be exceeded for WRED to consider a packet as a candidate for being dropped.)

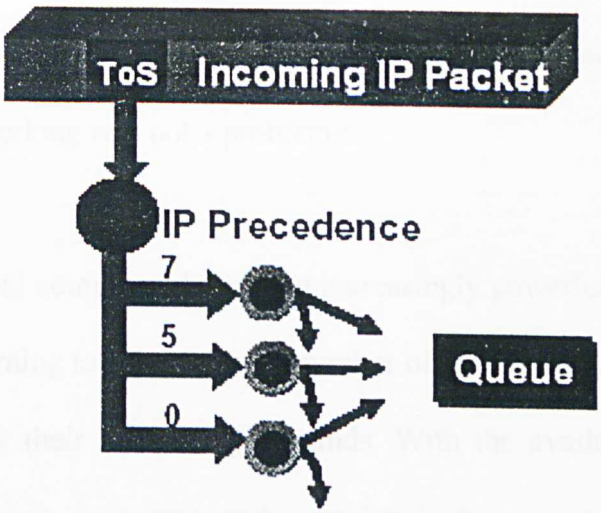


Figure 2.12: WRED

However, WRED has the following restrictions:

- WRED is only useful when the bulk of the traffic is TCP/IP traffic. With TCP, dropped packets indicate congestion, so the packet source will reduce its transmission rate. With other protocols, packet sources may not respond or may resend dropped packets at the same rate. Thus, dropping packets does not decrease congestion.
- WRED treats non-IP traffic as precedence 0, the lowest precedence. Non-IP traffic will be dropped more often than IP traffic.

2.7 Introduction of Network Simulator

Twenty years ago, when computers were more the exception than the norm in the world of office automation, most personal computers were used as stand-alone devices for simple tasks like word processing, and the media for exchange of

information was still paper. At that time, most serious computing was done on large mainframes, each with many terminals, working almost independently of each other, and computer networking was not a problem.

Today, with personal computers becoming increasingly powerful, and cost-effective, more people are turning to using a larger number of less powerful and less expensive machines to satisfy their computing demands. With the availability of such wide variety of computing resources and peripheral devices, a need to exchange information between various computers arose. Computers, with the ability to be used as flexible communication tools, have become an integral part of today's information hungry society.

In the networked environment, a computer is a powerful tool for information gathering, organization, storage and disbursement. A communication network represents the backbone of most major computing systems today, and is most often a critical factor in the performance of the entire computer system. Careful design and analysis of the communication network can often lead to coast-effective solutions that satisfy the communication needs of an organization. On the other hand, a poorly designed communication system will result in costly investments in maintenance, constant monitoring, troubleshooting and eventual replacement.

While many tools exist for the design and analysis of communication network systems, there exit none powerful enough to deal with the complex systems into which communication networks have evolved. Most simulation tools are capable only of simulating small, unrealistic, simple communication networks. Many

different network design principles exist today, and even these change constantly as further research into new network designs, and analysis of current network systems shows which networking paradigms are effective for satisfying the networking needs of today. What is needed is a flexible tool that is powerful enough to simulate the widely varies and complex communication networks commonly found in many organization today, and yet one, that is capable of evolving as networking needs and design paradigms change.

The simulator is used as a **network planning tool** by simulating various network configurations and traffic loads, and obtaining statistics such as utilization of network links and throughput rates of virtual circuits. It can be used to answer questions such as: where will the bottlenecks be in the circuits?; what is the effect of changing the speed of a link?; or will adding a new application cause congestion?

The simulator can also be used as a **protocol analysis tool** to study the total system effect of a particular network protocol. For example, one can investigate the effectiveness of various flow control mechanisms for ATM networks and address such issues as mechanisms for fair bandwidth allocation, protocol overhead, and bandwidth utilization. The simulator is designed in such a way that modules simulating components of an network can be easily changed, added, or deleted.

2.8 Summary

Each of these three approaches I have discussed approached the issue of fairness and isolation between different types of flows in a different manner than RED. RED simply applied the same drop-rate to all flows in an effort to give equal feedback and equally constrain all flows. FRED took the approach that fairness means allowing all flows to claim equal shares of the link capacity. Meanwhile, WRED provide fairness based on the precedence on the packets, which give higher priority to those packets will higher precedence.

Chapter 3
RIO

Chapter 3: RIO

3.1 Introduction of RIO

RIO – RED with IN and Out bit/profile, is an active queue management technique proposed for use in the differentiated-services initiative of the IETF. The differentiated-services model provides a very simple facility for negotiating and supporting service contracts between service providers and end-users. It allows end-users or network administrators to negotiate profiles specifying the type of traffic load they will place on the network. The service provider then promises to give traffic that conforms to its negotiated profile preferential treatment. RIO is an integrated approach that combines service profiles, policing at network ingress points, and a preferential drop policy based on the RED queue management mechanism [8]. As packets pass through network ingress points, they are policed to be sure they conform to the negotiated profile. Those packets that conform to the profile are marked as in-profile while those that exceed the negotiated profile are marked as out-of-profile.

By applying the RED algorithm differently to each category of packets, the router can preferentially drop out-of-profile packets before dropping those that are in-profile. In this way, RIO provides feedback to responsive flows and isolates the flows that conform to their profiles from the effects of those flows that misbehave.

Flows are associated with a service profile. The profile constitutes an agreement on the behavior of that flow (or group of flows) between the end-system on one side and the service provider on the other side. Upon ingress to the service provider's network, packets are tagged as In or Out of profile by a policing mechanism, which

will discuss in detail in the later section. If the flow is exceeding it profile, then those packets that represent its excess will be tagged as out-of-profile and those packets may be preferentially dropped.

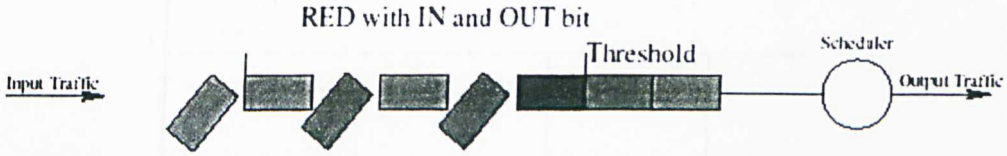


Figure 3.1: RIO

3.2 RIO Algorithm

RIO uses the same mechanism as in RED but is configured with two sets of parameters, one for in-profile packets and one for out-of-profile packets. Upon each packet arrival at the router, the router checks whether the packet is tagged as in-profile or out-of-profile. If it is an in-profile packet, the router calculates *avg_in*, the average queue for the in-profile packets; if it is an out-of-profile packet, the router calculates the *avg_total*, the average total average queue size for all (both in-profile and out-of-profile packets) arriving packets. The probability of dropping an in-profile packet depends on *avg_in*, and the probability of dropping an out-of-profile packet depends on *avg_total*.

As shown in the figure below, there are three parameters for each of the twin algorithms. The three parameters are *min_in*, *max_in*, and P_{max_in}, which define the normal operation [0, *min_in*), congestion avoidance [*min_in*, *max_in*), and

congestion control [*max_in*, ∞) phase for in-profile packets. Similarly, *min_out*, *max_out*, and P_{max_out} define the corresponding phases for out-of-profile packets.

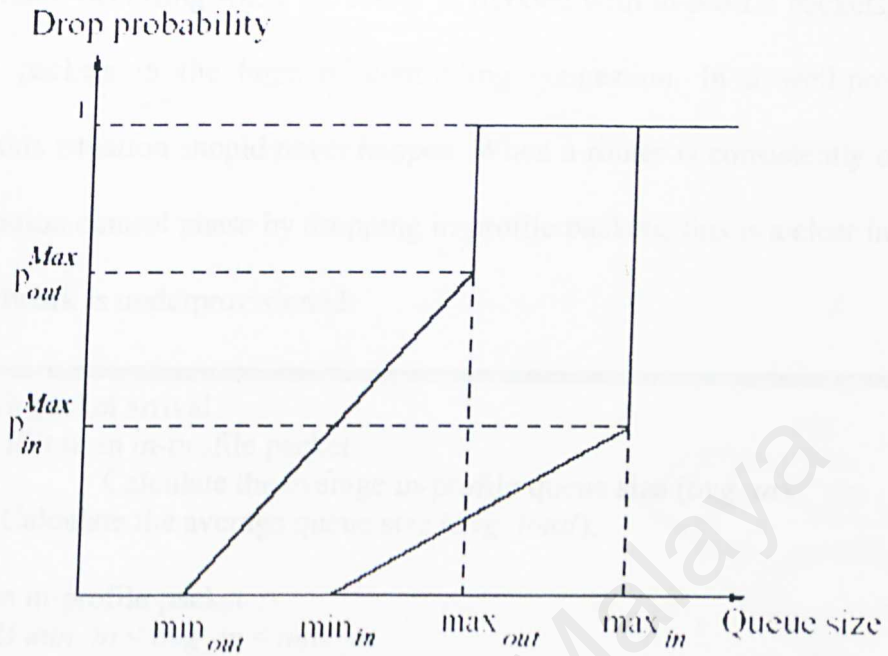


Figure 3.2: RIO drop probability

The discrimination against out-of-profile packets in RIO is created by carefully choosing the parameters for (*min_in*, *max_in*, P_{max_in}), and (*min_out*, *max_out*, P_{max_out}). As illustrated in the figure above, a RIO router is more aggressive in dropping out-of-profile packets on three counts:

- First, it drops out-of-profile packets much earlier than it drops in-profile packets; this is done by choosing *min_out* smaller than *min_in*.
- Secondly, in the congestion avoidance phase, it drops out-of-profile packets with a higher probability by setting P_{max_out} higher than P_{max_in}.
- Third, it goes into congestion control phase for the out-of-profile packets much earlier than in-profile packets by choosing *max_out* much smaller than *max_in*.

In essence, RIO router drops out-of-profile packets first when it detects incipient congestion, and drops all out-of-profile packets if the congestion persists. Only for the last resort, occurring when the router is flooded with in-profile packets, it drops in-profile packets in the hope of controlling congestion. In a well-provisioned network, this situation should never happen. When a router is consistently operating in a congestion control phase by dropping in-profile packets, this is a clear indication that the network is underprovisioned.

```
For each packet arrival
    If it is an in-profile packet
        Calculate the average in-profile queue size (avg_in);
    Calculate the average queue size (avg_total);

If it is an in-profile packet
    If min_in < avg_in < max_in
        Calculate probability P_in;
        With probability P_in, drop this packet;
    Else if max_in < avg_in
        Drop this packet;

If this is an out-of-profile packet
    If min_out < avg_total < max_out
        Calculate probability P_out;
        With probability P_out, drop this packet;
    Else if max_out < avg_total
        Drop this packet;
```

Figure 3.3: RIO algorithm

The choice of using *avg_total*, the total average queue size to determine the probability of dropping an out-of-profile packet, is subtle. Unlike in-profile packets, which the network can properly provision for, the out-of-profile packets represent opportunistic traffic, and there is no valid indication of what amount of out-of-profile packets is proper. If we had used the average out-of-profile packets queue to control

the dropping of out-of-profile packets, this would not cover the case where the total queue is growing due to arriving in-profile packets.

We could have used *avg_in,* the average queue for the in-profile packets, to see how much "free space" (buffer space) the routers have for out-of-profile packets, i.e., drop fewer out-of-profile packets when the *avg_in* is small and drop more out-of-profile packets when *avg_in* is large. But this only works when the number of in-profile packets in the queue is large, so the routers have good control on the number of out-of-profile packets and the total queue length.

By using the *avg_total*, the total average queue size, the routers can maintain short queue length and high throughput no matter what kind of traffic mix they have. It is conceivable that one could achieve a more responsive control of out-of-profile packets by changing the dropping parameters to depend on both the average in-profile queue size, *avg_in* and the average total queue size, *avg_total.*

3.3 Packet Marking

The RIO algorithm is meant to work along with "profile meters" located at the network access point, at the network egress point or at both location simultaneously. These meters tag or mark packets as in-profile or out-of-profile packets based on predefined contracts between the user and the services provider. The network provider allocates enough resources to comply with the user requirements. The

profile meters are "time-sliding window (TSW)" taggers with two components: a rate estimator and a tagging algorithm.

TSW refers to the rate estimator algorithm. TSW provides a smooth estimate of the TCP sending rate over a period of time. With the estimated rate, *avg_rate*, the tagging algorithm can tag packets as out packets once the traffic exceeds a certain threshold.

A rate estimator is used to smooth out the burstiness of TCP traffic as well as to be sensitive to instantaneous sending rates. This can be achieved through which TSW estimates the sending rate upon each packet arrival and decays, or forgets, the past history of time. The design of TSW is also very simply. TSW maintains three state variables – *Win_length*, which is measured in units of time, *Avg_rate*, the rate estimate upon each packet arrival, and *T_front*, which is the time of the last packet arrival. TSW is used to estimate the rate upon each packet arrival, so state variables *Avg_rate* and *T_front* are updated each time a packet arrives, but *Win_length* is preconfigured when the profile meter is installed. The TSW rate estimator works as shown in the figure below:

Initially:

Win_length = a constant;
Avg_rate = connection's target rate, R_T;
T_front = 0;

Upon each packet arrival, TSW updates its state variables as follows:

Bytes_in_TSW = Avg_rate * Win_length;
New_bytes = Bytes_in_TSW + pkt_size;
Avg_rate = New_bytes / (now – T_front + Win_length);
T_front = now;

Where ***now*** is the time of the current packet arrival time; and ***pkt_size*** is the packet size of the arrival packet.

Figure 3.4: TSW Algorithm

The window length, determines the worth of past history of the algorithm remembers, or in other words the weight of the past against the present. After the rate estimation the following tagging algorithm is executed.

If avg_rate <= R_T //target_rate == R_T
Mark packets as in-profile;
Else
With probability Pout = (avg_rate - R_T) / R_T;
Mark packets as out-of-profile;
Else
Mark packets as in-profile;

The probabilistic marking allows for the well-known oscillations of a TCP connection in pursuit of its maximum rate. As a matter of fact, for a TCP connection to achieve a given average rate, the connection should be allowed to oscillate around that value. The drawback is that this mechanism will permit other types of

connections, like a CBR-like (constant bit rate) connection, to get in average more than the target rate.

The policer is attached to an intermediate node representing a router and monitors the aggregate of traffic which enters/leaves the nodes. If the rate of the aggregate is below its target rate, packets are forwarded unchanged, but if the target rate is exceeded, the packet arriving with marked in-profile get remarked to out-of-profile before being forwarded.

3.4 Summary

RIO is noteworthy as it represents one of the first instances of combining classification with queue management. Instead of classifying packets to assign them to different priority queues to be dealt with by a scheduling mechanism. RIO uses a simple binary classification to determine which set of active queue management parameters to apply. All of the packets still go in the same queue so there is no need for the complexity of a scheduling mechanism. Moreover, order is maintained. This has positive effects for both multimedia and TCP. Because multimedia packets are usually played out when they are received, out-of-order packets are often discarded to maintain the illusion of continuity (since subsequent frames have already displayed). For TCP, out-of-order packets can trigger duplicate acknowledgments and unnecessarily trigger congestion control mechanisms. However, the separation statistics allow the two classes to be managed in different ways.

Chapter 4
Simulator Overview and System Analysis

Chapter 4: Simulator Overview and System Analysis

4.1 JaNetSim Network Simulator Overview

JaNetSim is Java Network Simulator developed at the Network Research Lab, Faculty of Computer Science and Information Technology (FCSIT), University of Malaya. The current version of the simulator is 0.66. the basic concepts used by JaNetSim are shown as follows:

- Discrete-event model
- Central simulation engine with a centralized event manager.
- Simulation scenario consists of a finite number of interconnected components (simulation objects), each with a set of parameters (component properties).
- Simulation execution involves components sending messages among each other. A message is sent by scheduling an event (to happen some later time) for the target component.

With the above architecture, the simulator can simulate virtually "anything" that can be modeled by a network of components that send messages to one another. These concepts are adopted from the NIST ATM/HFC Network Simulator.

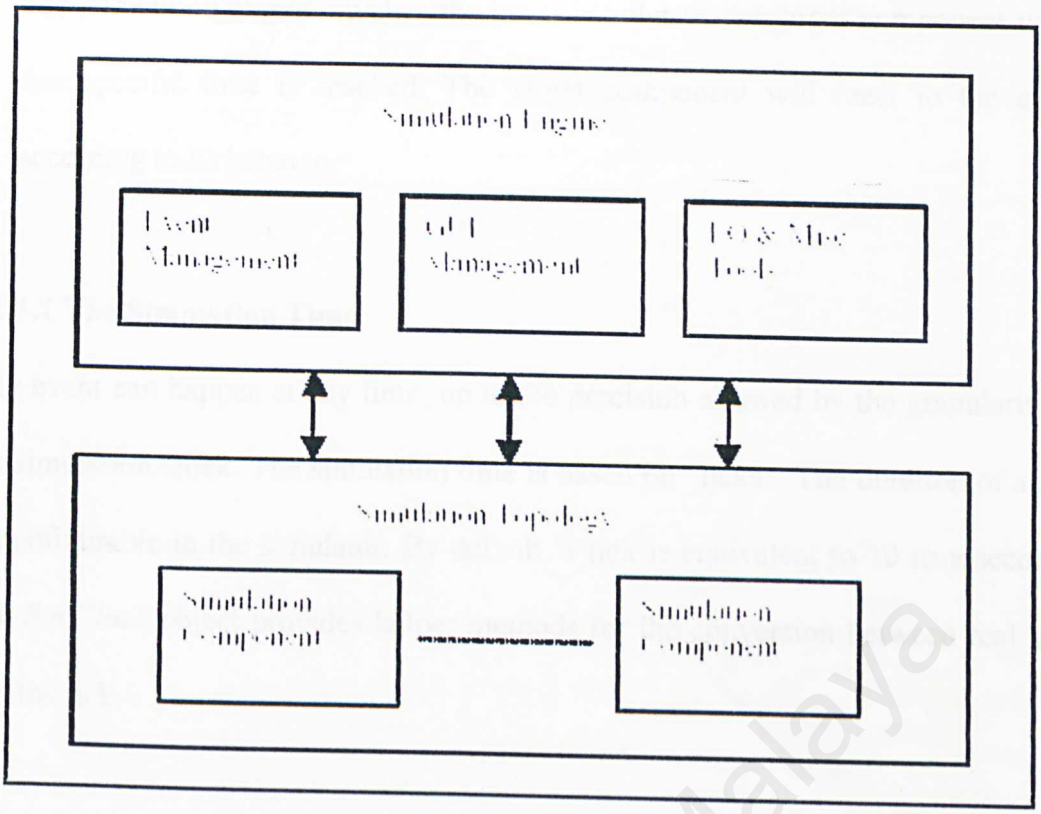


Figure 4.1: JaNetSim overall architecture

The simulation engine is the sole event manager and is responsible for the managing of all the user interface elements. The engine also provides convenient means for file saving and data logging, among other tools.

4.1.1 Event Management

The event manager manages an event queue and a global simulation clock. When simulation runs, the simulation engine interacts with the simulation topology (consists of all the simulation components) through two main operations:

- A simulation component schedules an event for a target component (can be the source component itself) to be happen at a specific time

- The simulation engine invokes the event handler of the target component when that specific time is reached. The target component will react to the event according to its behavior.

4.1.1.1 The Simulation Time

Any event can happen at any time, up to the precision allowed by the granularity of the simulation clock. The simulation time is based on "ticks". The duration of a tick is configurable in the simulator. By default, a tick is equivalent to 10 nanoseconds. The *SimClock* object provides helper methods for the conversion between real time and the tick.

4.1.2 GUI Management

It is possible to run the simulator in non-GUI mode once a scenario is built. The building of simulation scenarios is done using the GUI mode. The simulator has a main window, and within the main frame, there is a main simulation view area, which displays the simulation components and the connections among them. Basically the GUI manager handles all these automatically with only minimal effort from component developers.

A scenario is built by creating simulation components, connecting components and setting component parameters. The GUI manager handles all three tasks, and will signal the relevant components when these operations happen so that a component can react accordingly.

Basically, building a component that needs no custom component graphics and only uses pre-built parameter types (integer, double, Boolean etc.) does not require any GUI-related programming. For better GUI and more complex parameter types, JaNetSim provides a set of APIs to help simplifying this task.

4.1.3 RED Implementation

The version 0.66 of JaNetSim has been implemented RED algorithm as well. It is implemented at the IP router. The RED implementation is optional; which mean that users can choose whether want to use RED during the simulation or not. Besides that, the RED parameters, which are RED queue weight, RED minimum queue threshold, RED maximum queue threshold, RED maximum probability and RED packets transmission time are configurable by the users.

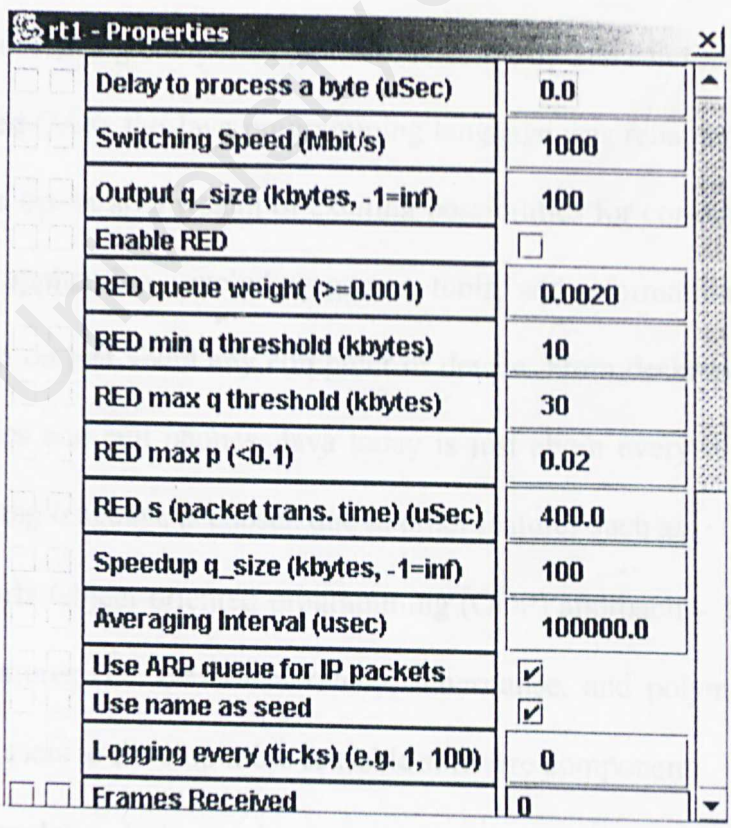


Figure 4.2: RED parameters

4.2 Software Specification

First of all, the platform/operating system required to run this simulator is any platform that support Java Virtual Machine. This includes Microsoft Windows operating systems, Sun Solaris, Macintosh, Unix and so forth.

Besides that, in order to implement RIO mechanism in JaNetSim, the programming language that will be used in Java programming language. The main reason Java programming language is chosen is due to the current JaNetSim network simulator is written in Java. If anyone wants to enhance the network simulator using other programming language than Java, then he/she has to rewrite every component, event and parameter with the programming language that have been chosen. This is very time consuming and not advisable.

Java is a groundbreaking computing platform released by Sun Microsystems in 1995. Originally called OAK, the Java programming language was renamed as Java in the same year. Java opens up a wealth of exciting possibilities for consumers. It enables just about any application - including games, tools, and information programs and services - to run on just about any computer or device. From desktop PCs to mobile handheld devices and cell phones, Java today is just about everywhere. Moreover, Java programming language is chosen due to other features such as:

- It supports Object-oriented programming (OOP) approach – Java has all the OOP features (i.e. classes hierarchy, inheritance, and polymorphism). This enables users to develop more reusable software component.
- It is simple – Java is simpler than other object-oriented programming language such as C++. Java has simplified C++ programming language by

both adding features beyond those found in C++ and by removing some of the features that make C++ more difficult to master.

- It is platform-independent – the programs written in Java are first compiled into Java byte code, which is machine independent. The Java virtual machine interprets the byte code into code that could be run in many different platform, which may have different instruction length.
- It supports multithreading – applications contain threads of execution, each thread designating a portion of a program that may execute concurrently with other threads. Multithreading allows computer to perform operation concurrently and synchronized. Synchronization is very useful in creating distributed systems. Besides that, Java also provides a low-priority garbage collector thread that reclaims dynamically allocated memory that is no longer needed. The garbage collector runs when processor time is available and there are no higher priority runnable threads. The garbage collector runs immediately when the system is out of memory to try to reclaim memory.
- It is robust – the Java objects can contain no references to data external to themselves or other known objects. This ensures that an instruction cannot contain the address of data storage in another application or in the operating system itself, either of which would cause the program and perhaps the operating system itself to terminate or "crash". The Java virtual machine makes a number of checks on each object to ensure integrity.
- It is secure – closely related to Java's robustness is its focus on security. Because Java does not use pointers to directly reference memory locations, as is prevalent in C and C++, Java has a great deal of control over the code that exists within the Java environment.

Overall, Java is a suitable programming language that will use to develop programs which reusability is an very important issue. Network simulator is a program that will always need to change and update as the network technologies may change frequently. Therefore, frequently enhancement in network simulator is needed and with software reusability, it could be done more easily.

On the other hand, since the Java 2 Software Development Kit (J2SDK) Standard Edition does not provide any graphical user interface to allow users make use of their toolkits in more convenient way, therefore by having Integrated Development Environment (IDE) is an added advantage for me. There are several IDE available in the market nowadays. Some of them are freeware while the other required purchase from the respective company. Examples of IDE are: Borland JBuilder, JCreator LE, NetBeans and so on.

I have decided to choose JCreator LE due to several reasons. First, it is a freeware that can be downloaded from the Internet and therefore allows free distribution. Secondly, it does provides a very simple GUI mode to make the interaction between users and toolkit more convenient – just at buttons click. Moreover, it also provides debugging tool to make the debugging more easier.

4.3 Hardware Specification

On the other hand, hardware specification to perform network simulation is very cheap. It can be run in any personal computers (platforms) that support Java such as Microsoft Windows operating systems, Linux and Solaris. Besides that, the JDK (Java Development Kit) 1.3.0 or later software must be installed in order to run this simulator.

There is no client-server environment is needed to run this simulator. The simulator is able to simulate the network based on the user's settings and no any other real hardware is required. This has make the simulator more cost effective.

4.4 Functional Requirement

The main purpose of implementing RIO mechanism in JaNetSim is to make it capable to perform simulation with the newer network technology. This is why the network simulator tools are very important and useful when dealing with network design, management and maintenance. Therefore, the functional requirement for this project is as follow:

- The simulator should be able to perform network simulation with or without RIO mechanism, which mean that RIO is optional during simulation.
- The simulator should allow the user to configure the parameter for the RIO mechanism before start a simulation.

- The system should allow only one type of congestion avoidance mechanism apply anytime. Therefore RED should be disable when RIO is applied and otherwise, the RIO should be disable when the RED is applied.

4.5 Non-functional requirement

Beside functional requirements, this project also has several non-functional requirements such as:

- The simulator should provide the graphics user interface (GUI) to enable the user to select the RIO before start the simulation and also set the parameters that are appropriated.
- As proposed, the RIO mechanism should performs the congestion avoidance more effective than the RED mechanism.

4.6 Summary

This chapter gives the overview of the Java Network Simulator of version 0.66 and also the implementation of RED algorithm. After that, the analysis on the software and hardware specifications is done. The functional and non-functional requirements are also defined at the end of this chapter.

Chapter 5
System Design

Chapter 5: System Design

5.1 Location of profile meter in the network

The general approach of the RIO mechanism is to define a service allocation profile for each user and to design a mechanism in the router that favors traffic that is within those service allocation profiles. The core of the idea is very simple – monitor the traffic for each user as it enters the network and tag the packets as either In-profile or Out-of-profile based on their service allocation profiles, then at each congested router, preferentially drop packets that are tagged as being Out-of-profile.

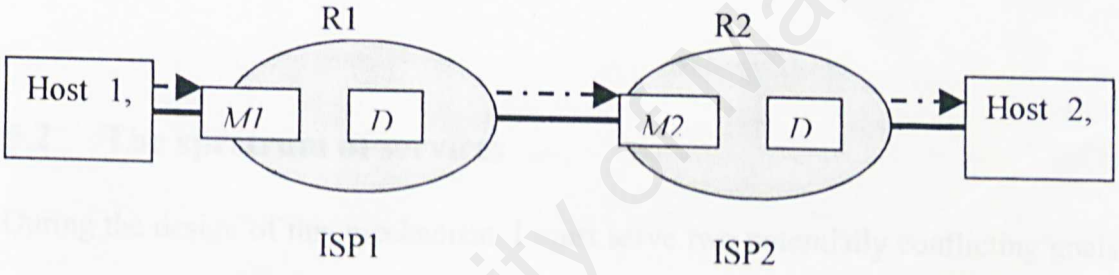


Figure 5.1: Location of profile meter in the network

The figure above illustrates the transmission of the packets for the sender to the destination. All the routers (R) in the network have adopted a preferential dropping algorithm (D). In the simple sender-based scheme the function that checks whether traffic fits within a profile is implemented by tagging packets at the edge of the network, e.g. the profile meter, M1 is on the access link from the H1 to the ISP1. A profile describes an expectation of service obtained by a customer from a provider. These relationships exist at many points in the network, ranging from the individual users and their campus local area networks (LAN's) to the peering relationships

between global ISP's. Any such boundary may be an appropriate place for the profile meter.

Furthermore, the packet tagging associated with this service allocation profile will in the general case, is performed by devices at both side of a boundary. One such device, located at ingress point of the network, the "checking meter" , sits on the arriving traffic side of a network boundary, checks the incoming traffic and marks packets as Out-of-profile if the arriving traffic exceeds the assigned profile. The example of this kind of profile meter is M1 and M2 as shown in the figure above.

5.2 The spectrum of services

During the design of this mechanism, I must serve two potentially conflicting goals. First, I would like to implement a set of simple services that are useful and easy to understand and adopt. Second, I do not want to embed the above services into the mechanisms so that the framework cannot adapt to new applications with new service requirements in the future. The decoupling of the service allocation profiles at the edge of the network allows this flexibility. To oversimplify, the preferential dropping scheme adopted in the routers in the center of the network will not change over time. Since the characteristics of a service is defined and captured by its corresponding profile meter, it is only necessary to create the profile meter at the edge of the network to adopt a new service.

Meanwhile, the services provided by this framework are diverse. As a simple example, it could be the equivalent of a dedicated link of some specified bandwidth from a source to a destination. Such a model is easy for users to understand. A more elaborate model can be aggregated commitment to a range of destinations, or anywhere within an ISP, sometimes called a private virtual network. A virtual network is by nature more difficult to offer with high assurance since offering commitments to "anywhere within a virtual network" implies that the ISP has provisioned its resources adequately to support all users sending in-profile traffic simultaneously to any destination.

Not all Internet traffic is continuous in its requirement for bandwidth. In fact, most Internet traffic is very bursty. It may thus be that a "virtual-link" service model is not what users really want. It is possible to support bursty traffic by changing the profile meter to implement this new sort of service. the key issue is to ensure, in the center of the network, that there is enough capacity to carry this bursty traffic and thus actually meet the commitments implied by the outstanding profiles.

In the center of the existing Internet, especially at the backbone routers of the major ISP's, there is a sufficiently high degree of aggregation that the bursty nature of individual users will not create a substantial provisioning issue in the center of the network, while possibly adding significant value to the service as perceived by the users.

In summary, three things must be considered when describing a service allocation profile.

- Traffic specifications: what exactly is provided to the customer (for example, 10 Mbps average throughput)?
- Geographic scope: to where is this service provided (examples might be a specific destination, a group of destinations, or all nodes on the local provider)
- Probability of assurance: with what level of assurance is the service provided, or alternately, what level of performance uncertainty can the user tolerate?

5.3 System flow

Before implement the RIO mechanism, I would like to make it more clear about the flow the system after the mechanism is implemented. Basically, it works in the same way as the RED mechanism. Before a user starts a simulation, he is required to build his network by using the available network devices such as router, switch and TCP application. This include build the link between every network device. The size and topology of the network are based on the user requirement.

After that, the user needs to configure the TCP applications with the appropriate setting. This may include the RIO mechanism, RED mechanism and others with the corresponding parameters. Meanwhile, the user can either choose to run the simulation with or without the RIO mechanism.

Once the simulation has been started, the TCP application will start sending out packet generated by other modules of the simulator. The packets then will forward to the destination through the link by using the appropriate routing algorithm. During

the transmission, the packet is definitely past through the switches and routers. There have two type of routers in the UMJaNetSim, which are IP router and RIP router. The RIP router is inherit from the IP router. Therefore, the RIP router will has all the IP router features plus other added features since both of them are using different protocols.

Later on, once the packet arrived at the router, the router will first checks whether the RIO mechanism option is chosen. If the option is not chosen, then the router will just forward the packet to the next destination or place in the queue if there is no enough bandwidth. Besides that, the packet may be dropped if the network is congested.

On the other hand, if the RIO mechanism is selected, then the packet marking algorithm and RIO algorithm will work as follow:

i. The packet will go through the profile meter that located at the front of the router. During this process, the rate estimator algorithm will perform some calculation and then the tagging algorithm will come in place to mark the packet as In-profile or Out-of-profile based on the user's parameter settings.

ii. The marking algorithm that I am going to use is to set the reserved (unused) bit in the DSCP (Differentiated Service Code Point) field in the IP header.

iii. The next phase is the router will exanimate the packet to check whether it is tagged as In-profile or Out-of-profile packet.

iv. If it is an In-profile packets, the router will then calculate the average queue size for In-profile packet and also total packets in the queue. The

average queue size is used to compare with the minimum and maximum thresholds for the In-profile packet which is set by the user to determine whether the packet should be dropped or not.

v. Meanwhile, if the packet is an Out-of-profile packet, then the router will calculate the total queue size in the buffer. Then this value will be used to compare with the minimum and maximum thresholds for the Out-of-profile packet which is set by the user to determine whether the packet should be dropped or not.

vi. The next phase is the packet will be forwarded to the next destination if the required bandwidth is available. Otherwise, it will queue in the router buffer.

The total time taken to perform the marking algorithm and RIO algorithm should be as short as possible. This is due to it may affect the round-trip-time (RTT) of the packets. Overall, the implementation should not cause too much delay, further changes and unnecessary overhead to the routers. The following flow chart will summarize the system flow.

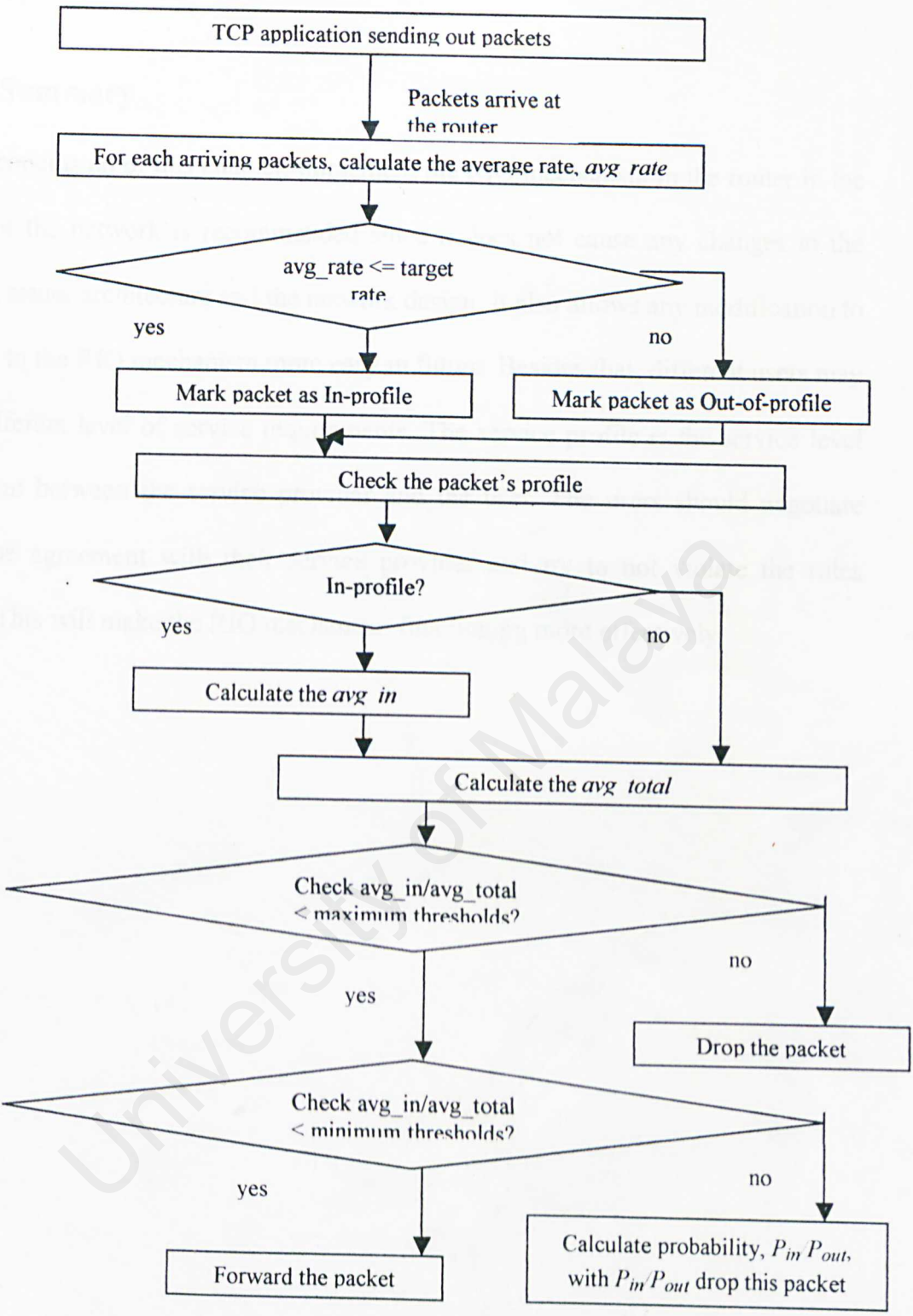


Figure 5.2: System flow diagram

5.4 Summary

As the conclusion of this chapter, implement the RIO mechanism in the router in the center of the network is recommended since it does not cause any changes to the existing router architecture and the network design. It also allows any modification to be done to the RIO mechanism more easy in future. Besides that, different users may have different level of service requirements. The service profile is the service level agreement between the service provider and the user. The users should negotiate about the agreement with their service provider and try to not violate the rules agreed. This will make the RIO mechanism functioning more effectively.

Chapter 6
System Implementation

Chapter 6: System Implementation

6.1 Implementation Overview and Simulator Version Being Used

Basically, the implementation of the RIO is done at the router of the simulator. This is mainly due to the router will be able to do the checking, marking and dropping whenever the source starts sending packets to the router it attached before the packets can be forwarded to the desired destination.

Meanwhile, the version of the network simulator chosen is UM Java Network Simulator version 0.49. This is an ATM (Asynchronous Transfer Mode) based network. The main feature of an ATM network is the network is a cell-based network compare to the traditional packet-based network. The cell is a fixed size frame that is 53 bytes (424 bits), while in the packet-based network, the packet size is variant. This has made the ATM network is faster than other traditional packet-based network.

In the simulator, the ATM-LSR will be used as the router for my implementation. ATM-LSR is a label switch with several LSC-ATM interfaces. The router forwards the cells among these interfaces using labels carried in the VPI/VCI field of the cells. Besides that, the ATM-LSR must be running the IP-type network. Therefore, the simulation topology will consists of ATM-LSR, IP BTE and also IP CBR applications. To differentiate the IP over ATM from other applications support by the ATM network, the protocol field in the cell header will contain the value "PRO_DATA".

6.2 Coding Added

Overall, no special modification is needed to be done in the existing coding of the ATM-LSR (ATMLSR.java) during implementation. Since the router will create a new port for each connection, therefore several new variables and parameters will be defined in the class Port inside the ATMLSR.java. These variables and parameters are defined follow the style as other variables and parameters that are defined by the original programmers.

The following is the list of the variables and parameters added in the class Port together with the brief explanation.

```
private class Port implements Serializable {
  /*----------------existing variables and parameters go here----------------*/

  //new queue and the queue size that will be used to hold the receiving cells and
    perform the RIO algorithm
  java.util.List out_Q=null;
  SimParamInt out_Q_size=null;

  //the average arrival rate of the cell
  double avg_rate=0;

  //the time of the current cell arrival (value same as theSim.now())
  long last_arrival=0;

  //the profile of the cell, default is false (Out-profile)
  boolean rio_marked=false;

  //start of the queue idle time ("borrowed" from RED parameter)
  long red_q_time=0;
```

```
//the In-profile and total (In-profile and Out-profile) average queues size
SimParamDouble rio_avg_in;
SimParamDouble rio_avg_total;

//total non-dropped cells that have arrived since the last cell dropped for both
queues
int rio_in_count=-1;
int rio_total_count=-1;

//the number of the cell being dropped
SimParamInt rio_drop_count;

} //end of class Port
```

On the other hand, these variables and parameters are also initialized with their corresponding default/initial values whenever necessary. As defined by the data type, these variables and parameters will be used to hold the value corresponding to the precision required (such as integer or double value).

Besides that, these variables and parameters will appear in some of the methods defined in the ATMLSR.java. For instance, in the method

void addNeighbor(SimComponent comp)

these variables and parameters are need to be added when a new port is created for each neighbor added/connected to the ATM-LSR component. In this method, some of these variables and parameters will be coded to display in the Properties option of the ATM-LSR component. For example, *rio_avg_in* and *rio_avg_total* will display the current In-profile and total average queues size and these value will be updated

whenever the new value is calculated. To update the value of the parameters being displayed, the method

update(theSim.now())

will be called with the *theSim.now()* as the parameter to be used as the time (tick) to update with the current value of the parameter. The same steps also done to the method

void removeNeighbor(SimComponent comp)

whenever a component is removed/disconnected from the ATM-LSR component. These parameters will also needed to remove from being display in the Properties option window straight away after the component is removed.

Moreover, in the method

void reset()

these variables and parameters will also reset to their default/initial values. This method is called when the user click the "Reset" button to reset the simulation. When this method is called, all the variables and also parameters added are need to re-initialize again. After that, some of these parameters that being displayed in the Properties option window will have to update with the current value, which is the default/initial value.

After that, the RIO parameters are then declare and added in the Properties option window of the router (ATM-LSR). These parameters are added as the global parameters for all the sources that connected to the router. This mean that the all the sources connected will have the same target rate, minimum and maximum threshold and maximum drop probability.

```
//RIO parameters
 private SimParamIntTag sw_red_or_rio;
 private SimParamDouble sw_rio_target_rate;
 private SimParamDouble sw_rio_in_minth;
 private SimParamDouble sw_rio_in_maxth;
 private SimParamDouble sw_rio_in_maxp;
 private SimParamDouble sw_rio_out_minth;
 private SimParamDouble sw_rio_out_maxth;
 private SimParamDouble sw_rio_out_maxp;
//end RIO parameters
```

As shown above, these parameters are declared at the beginning of the ATMLSR.java file. These parameters are then construct in the method

private void sw_create()

which is called by the class constructor. The constructions of these parameters are to add-in into the Properties option window with their corresponding default values. These value are not fixed and therefore the user will be able to change it before start a simulation. The declaration and construction of these parameters are also follow the style of the existing parameters.

The first parameter, *private SimParamIntTag sw_red_or_rio,* is used to create a pull-down menu that enable user to choose the congestion avoidance method they would like to apply during simulation. There is three option available: None, RED or RIO. By doing this way, the user will be able to choose only one type of congestion avoidance method during simulation as defined in the method requirements in chapter 4.

```
java.util.List red_or_rio=new java.util.ArrayList();
red_or_rio.add(new String("None"));
red_or_rio.add(new String("RED"));
red_or_rio.add(new String("RIO"));
sw_red_or_rio=newSimParamIntTag("Congestion Avoidance
                Type",getName(),ctick,false,true,red_or_rio,0);
params.add(sw_red_or_rio);
```

From the above coding, an ordered collection variable, *red_or_rio*, is declared to store the option of the congestion avoidance type and then assign it to the *sw_red_or_rio* tag parameter. The last line of the coding will add this parameter option in the Properties option window.

Meanwhile, *private SimParamDouble sw_rio_target_rate* is created to enable the user to set the target rate of the simulation. The target rate is used to monitor all the sources that connected to the router. Each source therefore will has the same target rate.

Both In-profile and total (In-profile and Out-profile) queues will have the same parameters, which are minimum and maximum thresholds for the respective queue, and also maximum drop probability. Therefore three parameters will be defined for each queue. For In-profile queue, the parameters include:

```
private SimParamDouble sw_rio_in_minth;
private SimParamDouble sw_rio_in_maxth;
private SimParamDouble sw_rio_in_maxp;
```

and for the total queue (In-profile and Out-profile), the corresponding parameters will be defined as:

private SimParamDouble sw_rio_out_minth;

private SimParamDouble sw_rio_out_maxth;

private SimParamDouble sw_rio_out_maxp;

As mentioned in the chapter 3, the RIO uses twin REDs. Therefore, some of the parameters defined for the RED may need to reuse in the RIO. The parameters are:

//RED parameter set

private SimParamDouble sw_red_wq;

private SimParamDouble sw_red_s;

//end RED parameter set

The *private SimParamDouble sw_red_wq* is the RED queue weight. The queue weight is used to calculate the average queue size when consider the portion of the queue and average queue size. On the other hand, *private SimParamDouble sw_red_s* is used as an value in a linear method of the time to calculate the average queue size when the queue is empty. At here, therefore, both parameters will be used to in the same way when perform the calculation for both In-profile and total (In-profile and Out-profile) queues.

The next step is to declare and define two methods, *sw_use_rio* and *sw_use_rio_dropping* inside the ATMLSR.java. Both of this methods will perform the RIO checking, marking and dropping algorithm. These methods is called after the method

private void sw_receive_ip_datagram(Cell cell,Port voport)

is called when received an IP datagram. The switch case at the beginning of this method definition will determine what type of congestion avoidance method is being

chose by the user and then the corresponding congestion avoidance method will be called.

6.2.1 Method *sw_use_rio*

The method

private void sw_use_rio(Cell cell, Port voport)

is defined and added in the ATMLSR.java. Two parameters are passed to this method, which are *cell* and *voport*. The *cell* parameters are required to refer to each cell arrived and therefore treat them differently. This is due to the RIO checking and marking algorithms are perform to each arrival cell at the router. On the other hand, the *voport* parameter is used to refer to the port, which is created for the source to send "in" the cells.

The following is the method definition:

```
private void sw_use_rio(Cell cell, Port voport) {
    voport.avg_rate=424/SimClock.Tick2Sec(theSim.now()-voport.last_arrival);
    voport.last_arrival=theSim.now();
    if( voport.avg_rate <= (sw_rio_target_rate.getValue()*1000000) )
        voport.rio_marked=true; //In-profile
    else
        voport.rio_marked=false; //Out-profile
}
```

Overall, the checking and marking algorithms are preformed per port-based. After that, the corresponding parameters of that particular port will be updated with the current value. The first line will calculate the average cell arrival rate, *avg_rate*, which is equal to the source's bit rate. I am using the IP CBR for my simulation. IP

CBR is a constant bit rate application which cell sending rate takes a constant, fixed value specified in the connection contract. Therefore, this value will also always be the same all the time.

The second line will then update the variable *last_arrival* to the current simulator time (equally to the *theSim.now()*). After the checking algorithm is complete, the marking algorithm will then take place. The *avg_rate* will be compared with the target rate. Target rate (Mbps) is a parameter available at the ATM LSR component and is configurable by the user. If the *avg_rate* is less than the target rate, the cell will be marked (true) as In-profile and otherwise, it will remain unmarked (false) as the default value.

6.2.2 Method *sw_use_rio_dropping*

After the method *sw_use_rio* is called to perform the checking and marking for each cell arrival, the method

private void sw_use_rio_dropping(Cell cell, Port voport)

then will be called to perform the dropping algorithm for RIO. Since each cell is treats independently, the method will also receives two parameters: *cell* and *voport*. Parameter cell is used to refer to individual cell that arrived at the router and if it has to be dropped, the only this particular cell will be affected. The parameter *voport* is used to refer to the connection of the source to the router. Therefore, by passing these two parameters, we will be able to treat each cell of every source and also each source independently.

Below is the method definition and also the explanation:

Basically, for each cell arrival, the new average total (In-profile and Out-profile) queue size will be calculate whether the cell is marked or not. The calculation method is same as the RED calculation method, which different calculation is performed by consider the queue is empty or not. After the new average total is calculated, the value display in the router's Properties option window will then be updated.

```
if(voport.out_Q.isEmpty()) { //q empty
 double m=SimClock.Tick2USec(theSim.now() - voport.red_q_time) /
 sw_red_s.getValue();
 voport.rio_avg_total.setValue(Math.pow(1.0-sw_red_wq.getValue(),m) *
 voport.rio_avg_total.getValue());
}
else { //q is not empty
 voport.rio_avg_total.setValue( voport.rio_avg_total.getValue() +
   sw_red_wq.getValue() * (voport.out_Q.size() -
   voport.rio_avg_total.getValue()) );
}
```

Then, the boolean value *dropped* is initial to false, which mean that the cell is not drop by default. Afterward, it will check cell profile. If the arrival cell is marked (In-profile), then the new average In-profile queue will be calculated. The calculation involved here is same as the calculation for average total (In-profile and Out-profile) queue, which also differentiate between empty and non-empty queue.

```
boolean dropped=false;
if(voport.rio_marked) { //In-profile
 if(voport.out_Q.isEmpty()) { //q empty
  double m = SimClock.Tick2USec(theSim.now() –
            voport.red_q_time)/sw_red_s.getValue();
  voport.rio_avg_in.setValue( Math.pow(1.0 - sw_red_wq.getValue(), m) *
            voport.rio_avg_in.getValue());
 }
 else { //q is not empty
  voport.rio_avg_in.setValue( voport.rio_avg_in.getValue() +
        sw_red_wq.getValue() * (voport.out_Q.size() –
        voport.rio_avg_in.getValue()) );
 }
```

As the same time, after the calculation is performed and the value is updated in the router Properties option window, the dropping algorithm will take place. The method will first get the minimum and maximum thresholds for In-profile from the router Properties option window. Later on, the new calculated average In-profile queue will compared with the maximum threshold.

If average In-profile queue exceed the maximum threshold, the cell will definitely be dropped and the total non-dropped cells count that have arrived since the last cell dropped for In-profile queue will set to 0. Otherwise, if the average In-profile queue size more than the minimum threshold and less than the maximum threshold, the total non-dropped cells count that have arrived since the last cell dropped for In-profile queues will increase by 1. Then, the probability, *p_in*, to drop this cell is calculate based on the maximum drop probability, minimum and maximum thresholds parameters and also the current average queue size for In-profile queue.

If the total non-dropped cells count for In-profile is larger than ($1/p\_in$), the this cell will be dropped and the total non-dropped cells count will set to 0. For the last condition where the average In-profile queue size is less than minimum threshold, the cell will not be dropped and the total non-dropped count will reset to –1.

```
double in_minth=sw_rio_in_minth.getValue();
double in_maxth=sw_rio_in_maxth.getValue();

if(voport.rio_avg_in.getValue() >= in_maxth) { //avg_in > in_maxth
 dropped=true;
 voport.rio_in_count=0;
}
else if(voport.rio_avg_in.getValue() >= in_minth) {
 //in_minth < avg_in < in_maxth
 voport.rio_in_count++;
 double p_in = sw_rio_in_maxp.getValue() * (voport.rio_avg_in.getValue() -
              in_minth) / (in_maxth-in_minth);
 if(voport.rio_in_count > (1 / p_in)) {
     dropped=true;
     voport.rio_in_count=0;
 }
}
else { //avg_in < in_minth
 voport.rio_in_count=-1;
 }
}
```

The same algorithm is performed if the cell is remain unmarked during cell's profile checking (out-profile). But at here, the maximum drop probability, minimum and maximum thresholds parameters for the Out-profile will be referred and used. The

average total (In-profile and Out-profile) queue size is used for the Out-profile as well.

```
else { //Out-profile
 double out_minth=sw_rio_out_minth.getValue();
 double out_maxth=sw_rio_out_maxth.getValue();

 if(voport.rio_avg_total.getValue() >= out_maxth) { //avg_total > out_maxth
  dropped=true;
  voport.rio_total_count=0;
 }
 else if(voport.rio_avg_total.getValue() >= out_minth) {
  //out_minth < avg_total < out_maxth
  voport.rio_total_count++;
  double p_out = sw_rio_out_maxp.getValue() * (voport.rio_avg_total.getValue()
                - out_minth) / (out_maxth-out_minth);
  if(voport.rio_total_count > (1 / p_out)) {
   dropped=true;
   voport.rio_total_count=0;
  }
 }
 else { //avg_total < out_minth
  voport.rio_in_count=-1;
 }
}

voport.rio_avg_in.setValue(voport.rio_avg_in.getValue());
voport.rio_avg_total.setValue(voport.rio_avg_total.getValue());
voport.rio_avg_in.update(theSim.now());
voport.rio_avg_total.update(theSim.now());
```

At this stage, the checking, marking and the decision of dropping for the RIO algorithm are done. If the cell is marked to be dropped, then the cell will be dropped

and the cell drop count of the respective connection to the router is increment by 1. the cell drop count value will also update in the option window immediately. Otherwise, the cell is then put into the queue, *out_Q*, to schedule it for the next event as usual.

```
if(dropped) {
  voport.rio_drop_count.setValue(voport.rio_drop_count.getValue()+1);
  voport.rio_drop_count.update(theSim.now());
  cell=null;
}
else {
  voport.out_Q.add(cell);
  voport.out_Q_size.setValue(voport.out_Q_size.getValue()+1);
}

voport.red_q_time=theSim.now();
//output cell to link if possible
if(!voport.out_Q.isEmpty() && !voport.link_busy) {
  voport.link_busy=true;
  sw_schedule_output(voport);
}
}
```

6.3 Summary

This chapter presented the method and coding involved for the implementation of RIO algorithm in the simulator. The parameters and variables declared are based on the standard style of coding in the simulator. During the implementation, explanation are inserted when necessary for future reference.

Chapter 7
System Testing

Chapter 7: System Testing

7.1 Simulation Setup

Before start the simulation to test the RIO algorithm, a simulation topology is needed to be setup. The simulation topology is a test case for the implementation that can be used to prove the RIO algorithm is properly implemented in the network simulator.

The most important simulation component is the router or called ATM-LSR. The ATM-LSR is capable to function under the IP network only. Besides that, three sources are created, which will generate the cells and pump into the network. These sources are the IP CBR (Constant Bit Rate) applications. These three IP CBR applications are sending cells with different rate, but the sending rate for each application is fixed all the time. Meanwhile, one IP CBR application will be used as the destination for the simulation setup. The figure below shows the simulation topology.

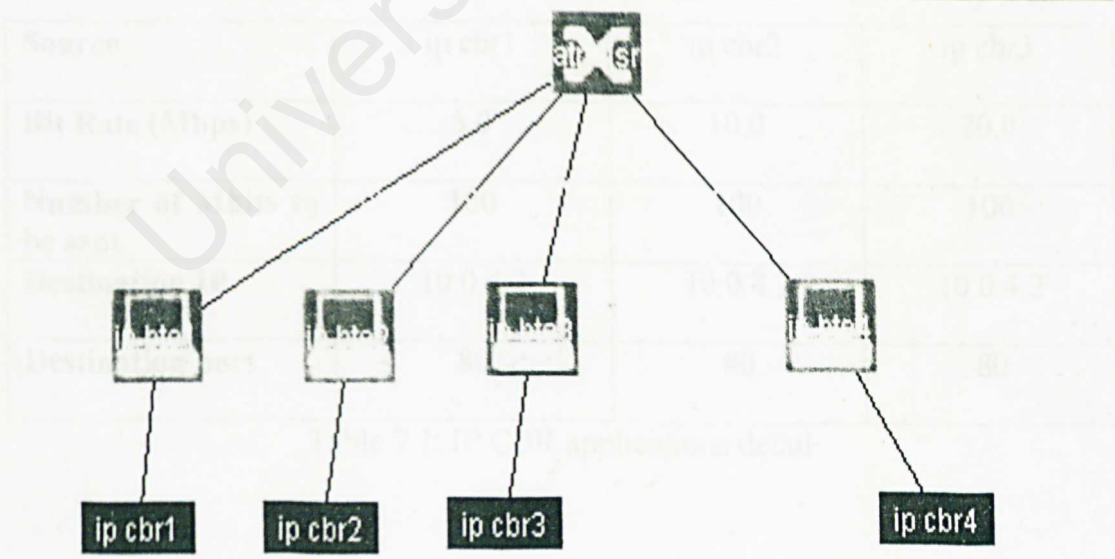


Figure 7.1: Simulation topology

From the figure, the IP BTE components are also needed to connect the IP CBR applications to the ATM-LSR through a link between the IP BTE and ATM-LSR. Originally, the IP BTE can be used to perform certain algorithm such as traffic profiling and other. Anyway, in order to simplify the simulation, the IP BTE components will act as an intermediate nodes or destination network (source "network") that will receive cells from IP CBR applications and the forward it to the ATM-LSR component.

7.2 Simulation Result

Firstly, the testing is done on the overall RIO algorithm. The simulation topology is as shown in the figure in previous page. The detail of the three IP CBR applications is as follow:

Source	ip cbr1	ip cbr2	ip cbr3
Bit Rate (Mbps)	5.0	10.0	20.0
Number of Mbits to be sent	100	100	100
Destination IP	10.0.4.2	10.0.4.2	10.0.4.2
Destination port	80	80	80

Table 7.1: IP CBR applications detail

Below is the list of parameters configured in the ATM-LSR component.

Congestion avoidance type	RIO
Switching slot time (Mbit/s)	1
Target Rate (Mbps)	10.0
RIO In-profile min threshold	0.50
RIO In-profile max threshold	0.75
RIO In-profile maxp	0.20
RIO Out-profile min threshold	0.35
RIO Out-profile max threshold	0.50
RIO Out-profile maxp	0.02
IP for interface to link1	10.0.1.1
IP for interface to link2	10.0.2.1
IP for interface to link3	10.0.3.1
IP for interface to link4	10.0.4.1

Table 7.2: ATM LSR parameters configuration

The ip bte4 source "network" field will has the value of 10.0.4.2, where this is the destination IP of the sources. The simulation is running for 40ms. Below is the simulation result.

	ip cbr1	ip cbr2	ip cbr3
Bit Rate (Mbps)	5.0	10.0	20.0
Total cell drop	2	4	13
First cell drop time (ms)	32.97	16.73	6.75

Table 7.3: Simulation result

As shown in the table above, the sources (ip cbr3) that violates the target rate will experience more cells drop that those sources that conform to theirs target rate. This is due to the cells sent by ip cbr3 application will always mark as "Out-profile" and the RIO algorithm will drop the Out-profile cells with higher probability that the cells that are marked as "In-profile". This is done by setting maximum drop probability for Out-profile much higher than the In-profile during the congestion avoidance phase. Therefore, it is obvious that the ip cbr3 application will also experience first cell drop much earlier than other applications that conform to their target rate.

Meanwhile, for ip cbr1 and ip cbr2 applications that are conforming to their target rate, both of them will also experience cells drop. When the average In-profile queue size for both applications exceed the minimum threshold, the total non-dropped cells that have arrived since last cell drop counter will increment by 1 for every non-dropped arrival cell. The counter is used here to protect against burst cell losses, since it will maintain a constant space between the dropped cells relative to the dropping probability. On the other hand, the traditional RED algorithm will randomly generate the drop probability for each cell based on the average queue length, minimum and maximum thresholds. In this case, if the generated drop probability is higher than maximum drop probability, then the particular cell will be dropped.

From the result shown in the table, we can conclude that for the In-profile applications, the application (ip cbr2) will the higher cell sending rate will experience cell drop much more earlier than the application (ip cbr1) with lower sending rate.

Besides that, the total cell dropped for ip cbr2 application is also greater than ipcbr1 application, as the ip cbr1 only experience 2 cells dropped compare to ip cbr2 is experience 4 cells dropped. The is due to the application with higher sending rate will has higher average queue length and therefore will exceed the minimum threshold earlier than the application with lower sending rate. The figures below show the In-profile and Out-profile average queue length for each application in the router.

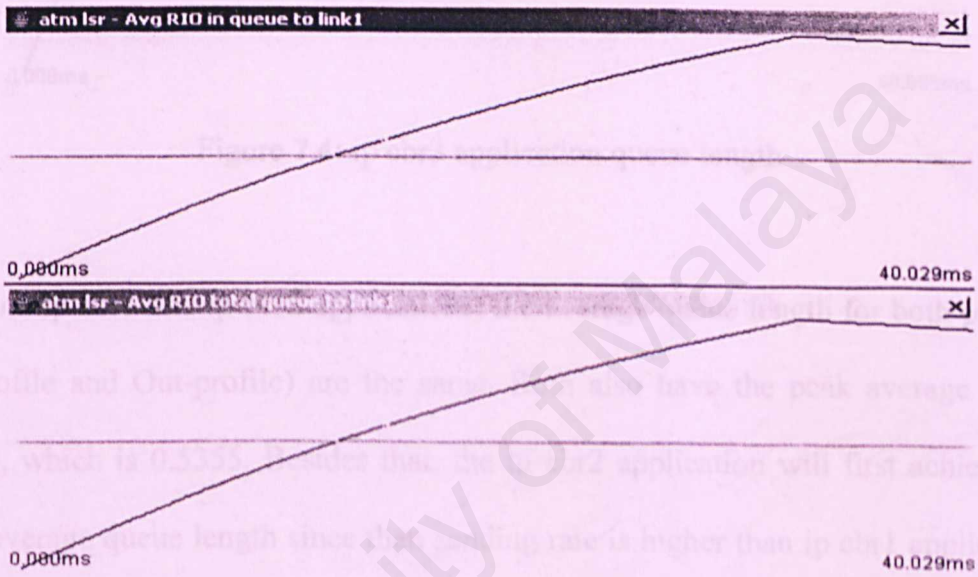


Figure 7.2: ip cbr1 application average queue length

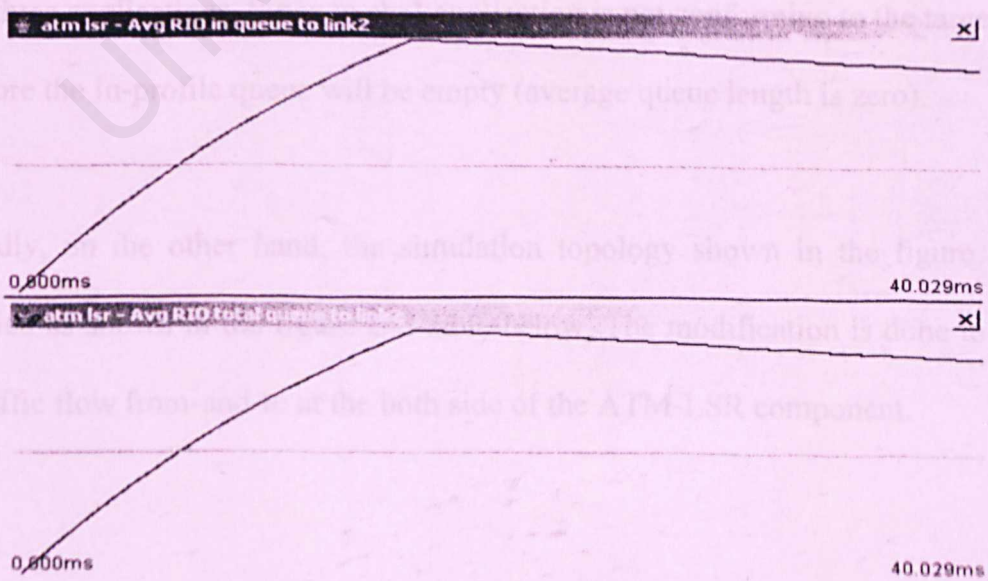


Figure 7.3: ip cbr2 application average queue length

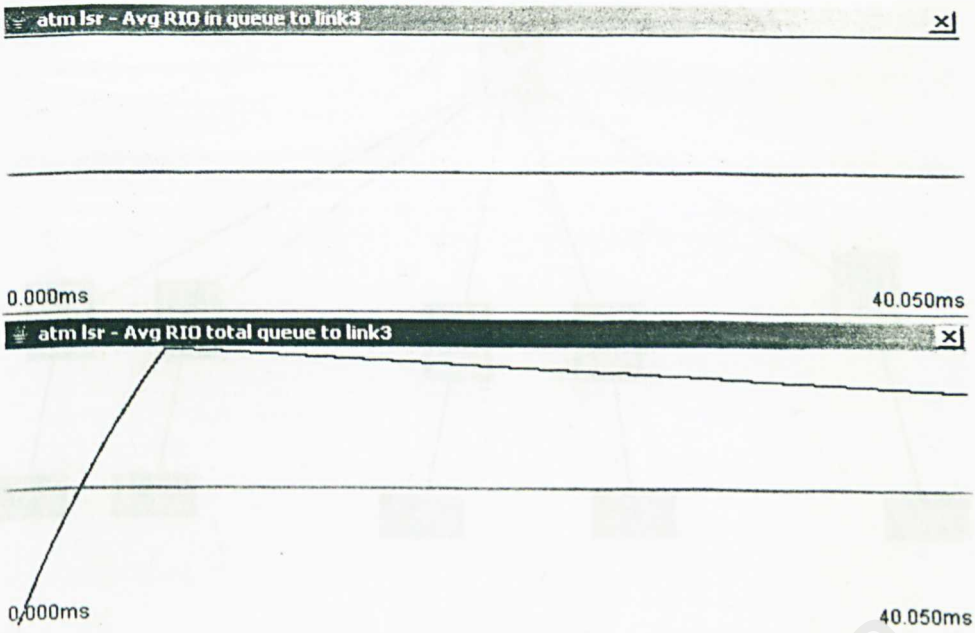


Figure 7.4: ip cbr3 application queue length

For both ip cbr1 and ip cbr2 applications, the average queue length for both profiles (In-profile and Out-profile) are the same. Both also have the peak average queue length, which is 0.5355. Besides that, the ip cbr2 application will first achieve the peak average queue length since then sending rate is higher than ip cbr1 application. Anyway, for the ip cbr3 application, the peak average queue length is 0.4460. This application is the first application achieves the peak average queue length among these three applications. Since ip cbr3 application is not conforming to the target rate, therefore the In-profile queue will be empty (average queue length is zero).

Secondly, on the other hand, the simulation topology shown in the figure 7.1 is modified as shown in the figure and table below. The modification is done to make the traffic flow from-and-to at the both side of the ATM-LSR component.

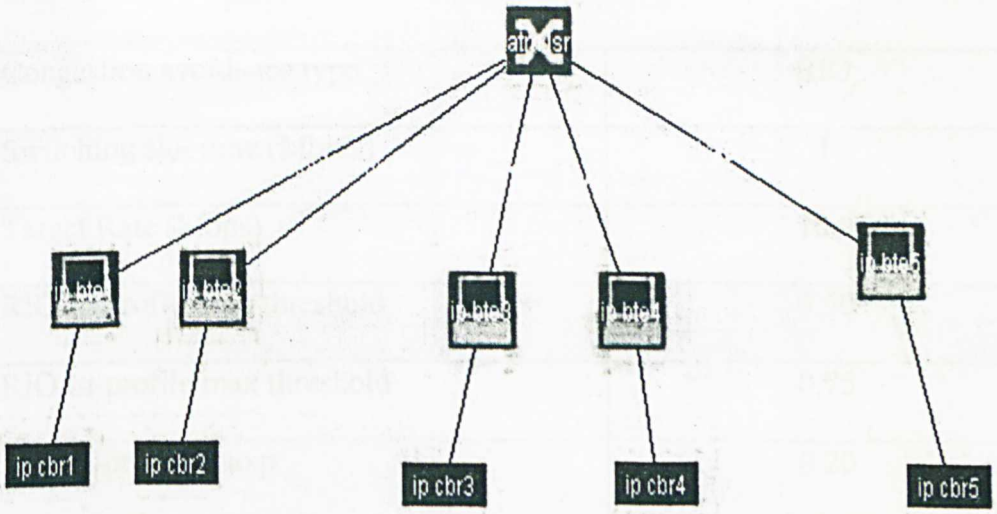


Figure 7.5: Modified simulation topology

	Source(s)	*Destination*
Flow 1	ip cbr1, ip cbr2	ip cbr4
Flow 2	Ip cbr5	ip cbr3

Table 7.4: Modified simulation topology - source(s) and destination

Source	ip cbr1	ip cbr2	ip cbr3	ip cbr4	ip cbr5
Bit Rate (Mbps)	5.0	20.0	-	-	10.0
Number of Mbits to be sent	100	100	-	-	100
Destination IP	10.0.4.2	10.0.4.2	-	-	10.0.5.2
Destination port	80	80	-	-	80

Table 7.5: Modified simulation topology - IP CBR applications detail

Congestion avoidance type	RIO
Switching slot time (Mbit/s)	1
Target Rate (Mbps)	10.0
RIO In-profile min threshold	0.50
RIO In-profile max threshold	0.75
RIO In-profile maxp	0.20
RIO Out-profile min threshold	0.35
RIO Out-profile max threshold	0.50
RIO Out-profile maxp	0.02
IP for interface to link1	10.0.1.1
IP for interface to link2	10.0.2.1
IP for interface to link3	10.0.3.1
IP for interface to link4	10.0.4.1
IP for interface to link5	10.0.5.1

Table 7.6: Modified simulation topology - ATM LSR parameters configuration

The simulation is also running for 40ms. The result is shown in the table below.

Source	ip cbr1	ip cbr2	ip cbr3	ip cbr4	ip cbr5
Bit Rate (Mbps)	5.0	20.0	-	-	10.0
Total cells drop	2	14	-	-	4
First cell drop time (ms)	33.122	6.724	-	-	16.690

Table 7.7: Modified simulation topology – simulation result

From the simulation result, the modified simulation topology also achieves the same result as the original result. The total cells dropped and the first cell drop time is almost the same. Therefore, the result are as expected. Anyway, the testing done here is to show that the applications on both sides of the router are able to send and receive cells at the same time. This mean that the RIO algorithm is able to function properly in duplex traffic as the algorithm is perform at the input port of the router only.

7.3 Summary

As the summary, the implementation of RIO in the network simulator is able to achieve the project objective. Besides that, the system functional and non-functional-requirements are also meet. This testing is important to make sure the RIO algorithm is functioning properly under different condition.

Chapter 8
Conclusion & Future Work

Chapter 8: Conclusion & Future Work

Overall, the implementation of RIO algorithm has provided valuable insight into the methodology of network simulation. Besides that, the concept of the RIO algorithm can also be validated by using the network simulation.

In the today's Internet, the high degree of sharing of the available bandwidth makes no commitment about the capacity that any user will actually receive. Therefore, the implementation of Differentiated Service (DiffServ) is seen to support Quality of Service (QoS) and ensures fairness among various type of users in the Internet. The RIO algorithm is proposed as an algorithm to discriminate between the users that conforms to their service level agreement (bandwidth in this case) and those users whom are not.

Thus, the users who do not violate their target rate will be marked and treat them with higher priority compare to the users who violate their target rate during normal operation (congestion avoidance phase). During congestion control phase, the "marked" sources (In-profile) are protect against the "unmarked" (Out-profile) sources by setting lower drop probability.

Therefore, it allows the service providers to design widely different service and pricing models, without having to build these models into all of the switches and routers of the network. In contrast, the service allocation profiles will change and adapt to needs of the future applications and business models of IPS's, and will only affect the edge of the network. This design thus pushes most of the complexity to the edge of the network, making it scalable and flexible.

8.1 System Strengths

There are several strengths of this implementation:

- The implementation of RIO algorithm is relatively simple and does not introduce significant changes in the router.
- The discrimination between In-profile and Out-profile sources can protect sources that are conforming to their target rate. This is because the Out-profile sources will experience more cells drop and much earlier than In-profile sources. Therefore, the Out-profile sources will have lower throughput.
- The dropping algorithm is able to prevent burst-cell losses. This is done by having a counter that will increment when non-dropped cells have arrived and reset the counter after a cell is dropped.

8.2 System Limitations

Besides that, there are several limitations within my implementation of RIO algorithm in the Java Network Simulator, which are:

- The sources application are limited to the CBR (Constant Bit Rate) application. In the real environment, the application may sending data at different rate all the time. Therefore, the IP VBR (Variable Bit Rate) application should also capable to use RIO algorithm in this case.
- There is no difference between the boundary and core routers for the simulation.

8.3 Future Work

The following are some of the promising avenues for the future work in this field:

- The RIO algorithm should be implemented in the packet-based network, which is the Java Network Simulator version 0.66. By doing this way, the behaviors of the TCP application will also influent the simulation result of RIO algorithm. This is due to the TCP application is having its own congestion control methods during congestion control phase.
- The implementation of different kinds of router - either boundary or core routers (based on the location of the router). By doing this way, the boundary router will do the checking and marking, meanwhile the core router will then do the dropping for the congestion avoidance and congestion control phases.
- Aggregate traffic – a profiler for aggregator traffic from a number of connections be suggested and implemented.

Appendix

Appendix

A. System User Manual ("README.txt")

```
+-------------------------------------------------------------------------------+
|        Implementation of RIO (Random Early Detection with In/Out)      |
+-------------------------------------------------------------------------------+
```

Below is the general guidelines to build a simple simulation topology and test the RIO algorithm:

1. Create a simulation topology. The simulation topology should consists of IP CBR application(s) as source(s) and destination, IP BTE(s) to interconnect the IP CBR application(s) to router through a link(s), and ATM-LSR as the router.

2. Set the "Bit Rate (Mbps)", "Number of MBits to be sent", "Repeat count", "Destination IP" and also "Destination port number" fields on the IP CBR application(s) properties. Other fields remain unchange (using the default value).

3. For the ATM-LSR, set the IP address for all the interfaces created. After that, choose the congestion avoidance method ("None"-default, "RED", or "RIO").

4. Dependent of which type of congestion avoidance method is chosen, then configure the relevant parameters. If the RIO is chosen, please don't forget to set the "Target Rate (Mbps)" field. Other fields of the RIO parameters may remain unchange.

5. Set the "Switching Slot time (Mbit/s)" field of the ATM-LSR to "1". By doing this way, the user will be able to see the result more clear.

6. For the destination's IP BTE, the "Source Network" need to be set with the IP address of the destination IP address. This IP address must be same as the IP address in the "Destination IP" field of the source(s).

7. Start the simulation and see the result.

8. Please take note that if you would like to get the result in a shorter time, you may change the "RED queue weight (>=0.001)" field of the ATM-LSR to a bigger value, such as "0.02". The bigger value you choose, the faster you will get the result.

Hope this guidelines can help you.

Reference

Reference

1. Uyless Black, "TCP/IP and Related Protocols", 3rd Edition, McGraw-Hill, 1998.
2. Sally Floyd, "TCP and Explicit Congestion Notification (ECN)", ACM Computer Communication Review, vol 24, pp 8-23, Oct. 1995.
3. M. A. Parris, "Class-Based Thresholds: Lightweight Active Router Queue Management for Multimedia Networking", University of North Carolina, 2001.
4. B. Braden et al., "Recommendations on queue management and congestion avoidance in the Internet,", Request for Comments (RFC) 2309, Apr. 1998.
5. Sally Floyd and V. Jacobson, "Random Early Detection gateways for congestion avoidance," IEEE/ACM Transactions on Networking, vol. 1, pp. 397-413, Aug. 1993.
6. D. Lim and R. Morris, "Dynamics of Random Early Detection," Proc. of SIGCOMM, pp. 127-137, 1997.
7. WRED: http:///www.networks.swin.edu.au/services/qos/#wred.
8. D. Clark and J. Wroclawski, "An Approach to Service Allocation in the Internet", Internet Draft draft-clark-diff-svc-alloc-00.txt. July 1997.
9. D.H. Nam, Y.S. Choi, B.C. Kim and Y.Z. Cho, "A Traffic Conditionng & Buffer Management Scheme for Fairness in Differentiated Services", Kyungpook National University, 2001.
10. K. Chang, V. Kapoor, E. Kusmierek and C. Wighe, "Comparison of Packet Marking Schemes in Differentiated Services", University of Minnesota, 1999.

11. A. Habib, S. Fahmy and B. Bhargava, "Design and Evaluation of an Adaptive Traffic Conditioner for Differentiated Services Networks", Purdue University, 2001.

12. W.H. Park, S. Bahk and H. Kim, "A modified RIO algorithm that alleviates the bandwidth skew problem in Internet Differentiated Service", Seoul National University, 2000.