[illegible] copy

Evaluation of the effect of Queuing, Scheduling and Dropping techniques on the QoS (quality of service) in DiffServ environment using simulation

Dissertation
Submitted in the partial fulfillment of the requirement for the award of Degree of

MASTER OF TECHNOLOGY
IN
COMPUTER SCIENCE & TECHNOLOGY

BY
MANISH BANSAL

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI-110067

January 2004

CERTIFICATE

This is to certify that the thesis entitled **"Evaluation of the effect of Queuing, Scheduling and Dropping techniques on the QoS (quality of service) in DiffServ environment using simulation"**, being submitted by **Mr. Manish Bansal** to the School of Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology in Computer Science & Technology, is a bonafide work carried out by him under the guidance and supervision of Prof. G. V. Singh, during the Monsoon Semester, 2003.

The matter embodied in this dissertation have not been submitted in part or full to any other University or Institution for the award of any degree etc.

[illegible]

MANISH BANSAL
(Student)

[illegible]

Prof. Karmeshu
Dean, SC&SS,
JNU, New Delhi-110067

[illegible]

Prof. G. V. Singh
(Supervisor)
05/01/2004

...dedicated to

my beloved brother

ACKNOWLEDGEMENT

I would like to pay obeisance at the feet of my parents for their blessings that are always with me in all my aspirations including my academics. I would like to sincerely thank my supervisor Prof. G.V. Singh, School of Computer and Systems Sciences, Jawaharlal Nehru University for his able guidance, encouragement and support provided in successful completion of this project.

I would like to forward my sincere thanks to the dean, Prof. Karmeshu for providing the necessary facilities. My special thanks go to Mr. Jugal Kishore for his kind and consistent help throughout the making of this dissertation. I will acknowledge the efforts and knowledge put by the development team of VINT project team at UC Berkeley, for developing the network simulator and making its source code available on the Internet as a freeware. I take this opportunity to thank all the faculty members and friends for their help and encouragement during the course of the project.

[illegible]

Manish Bansal

(Student)

ABSTRACT

In this dissertation work, the simulation study of effects of various queuing, scheduling and dropping techniques on the quality of service in DiffServ environment has been done by using network simulator NS2 devepoled at UC Berkeley. NS2 provides the simulation environment for the DiffServ networks. A Tcl script specifying the network configuration, data rate and packet size is given as the input to the simulator. Various policies and resource management techniques are used to control the end to end traffic. NS2 supports various algorithms for the simulation of the DiffServ traffic. These algorithms are used to analyze the different combinations of the traffic with the above mentioned parameters. The traffic is traced and viewed using the Nam network animator. After the simulation process the resulted data have been analyzed for the different output parameters such as number of lost/dropped packets, delay measurement and bandwidth utilization.

The simulation study shows that it is possible to implement the DiffServ architecture to provide the QoS in the Internet. These QoS can be grouped in to various categories depending on the user's SLA (Service Level Agreement) with the ISP (Internet Service Provider).

CONTENTS

CHAPTER 1
INTRODUCTION

1.1Background

Today's Internet provides only the Best Effort Service. Traffic is processed as quickly as possible, but there are no guarantees as to timeliness or actual delivery. With the rapid transformation of the Internet into a commercial infrastructure, demands for service quality have rapidly developed. It is becoming apparent that several service classes will likely be demanded. With the phenomenal growth of the Internet, as more hosts are connected, network service demands, eventually exceed capacity, but service is not denied. Instead it degrades gracefully. Although the resulting variability in delivery delays (jitter) and packet loss do not adversely affect typical Internet applications- email, file transfer and web applications—other applications can't adapt to inconsistent service levels. Delivery delays cause problems for applications with real time requirements, such as those that deliver multimedia, the most demanding of which are two-way applications like telephony.

Increasing bandwidth is a necessary first step for accommodating these real-time applications, but it is still not enough to avoid jitter during traffic bursts. Even on a relatively unloaded IP network, delivery delays can vary enough to continue to adversely affect real-time applications. To provide adequate service -- some level of quantitative or qualitative determinism -- IP services must be supplemented. This requires adding some "smarts" to the net to distinguish traffic with strict timing requirements from those that can tolerate delay, jitter and loss. That is what Quality of Service (QoS) protocols are designed to do. QoS does not create bandwidth, but manages it so it is used more effectively to meet the wide range or application requirements. The goal of QoS is to provide some level of predictability and control beyond the current IP "best-effort" service. One service class will provide predictable Internet services for companies that do business on the web. Such companies will be willing to pay a certain price to make their services reliable and to give their users a fast feel of their web sites. Another service class will provide low delay and low jitter services to applications such as Internet telephony and video conferencing. Companies will be willing to pay a premium price to run a high quality videoconference to save travel time and cost. Finally, the Best Effort Service will remain for those users who need connectivity only.

1.2Various mechanisms to provide the QoS over Internet

The Internet Engineering Task Force (IETF) has proposed many service models and mechanisms to meet the demand for QoS. Notably among them are the Integrated Services/RSVP model, the Differentiated Services (DS) model, MPLS, Traffic Engineering and Constraint Based Routing. The Integrated Services model is characterized by resource reservation. For real-time applications, before data are transmitted, the applications must first set up paths and reserve resources. RSVP is a signaling protocol for setting up paths and reserving resources. In Differentiated Services, packets are marked differently to create several packet classes. Packets in different classes receive different services. MPLS is a forwarding scheme. Packets are assigned labels at the ingress of an MPLS-capable domain. Subsequent classification, forwarding, and services for the packets are based on the labels. Traffic Engineering is the process of arranging how traffic flows through the network. Constraint Based Routing is to find routes that are subject to some constraints such as bandwidth or delay requirement.

Notably among them are Integrated Services/RSVP and Differentiated Services models. Some basic terms and definitions, which will be used in the description of these architectures, are as follows-

Flow: A stream of packets with the same source IP address, source port number, destination IP address, destination port number and protocol ID.

Service Level Agreement (SLA): A service contract between a customer and a service provider that specifies the forwarding service a customer should receive. A customer may be a user organization or another provider domain (upstream domain).

Traffic Profile: A description of the properties of a traffic stream such as rate and burst size.

Differentiated Services (DS) field: The field in which the Differentiated Services class is encoded. It is the Type of Service (TOS) octet in the IPv4 header or the Traffic Class octet in the IPv6 header.

Per-Hop-Behavior (PHB): The externally observable behavior of a packet at a DS-compliant router.

Mechanism: A specific algorithm or operation (e.g., queuing discipline) that is implemented in a router to realize a set of one or more per-hop behaviors.

Admission Control: The decision process of whether to accept a request for resources (link bandwidth plus buffer space).

Classification: The process of sorting packets based on the content of packet headers according to defined rules.

Behavior Aggregate (BA) Classification: The process of sorting packets based only on the contents of the DS field.

Multi-Field (MF) Classification: The process of classifying packets based on the content of multiple fields such as source address, destination address, TOS byte, protocol ID, source port number, and destination port number.

Marking: The process of setting the DS field in a packet.

Policing: The process of handling out of profile traffic, e.g., discarding excess packets.

Shaping: The process of delaying packets within traffic stream to cause it to conform to some defined traffic profile.

Scheduling: The process of deciding which packet to send first in a system of multiple queues.

Queue Management: Controlling the length of packet queues by dropping packets when necessary or appropriate.

Traffic Trunk: An aggregation of flows with the same service class that can be put into a MPLS Label Switched Path.

TCB: Traffic Conditioning Blocks are collections of data-path elements performing some policy.

1.2.1 Integrated Services and RSVP

The Integrated Services model proposes two service classes in addition to Best Effort Service. They are:

- Guaranteed Service for applications requiring fixed delay bound.
- Controlled Load Service.

For applications requiring reliable and enhanced best effort service. The philosophy of this model is that "there is an inescapable requirement for routers to be able to reserve resources in order to provide special QoS for specific user packet streams, or flows. This in turn requires flow-specific state in the routers". RSVP was invented as a signaling protocol for applications to reserve resources. The signaling process is illustrated in Fig. 1. The sender sends a PATH Message to the receiver specifying the characteristics of the traffic. Every intermediate router along the path forwards the PATH Message to the next hop determined by the routing protocol. Upon receiving a PATH Message, the receiver responds with a RESV Message to request resources for the flow. Every intermediate router along the path can reject or accept the request of the RESV Message. If the request is rejected, the router will send an error message to the receiver, and the signaling process will terminate. If the request is accepted, link bandwidth and buffer space are allocated for the flow and the related flow state information will be installed in the router.

Integrated Services is implemented by four components: the signaling protocol (e.g. RSVP), the admission control routine, the classifier and the packet scheduler. Applications requiring Guaranteed Service or Controlled-Load Service must set up the paths and reserve resources before transmitting their data. The admission control routines will decide whether a request for resources can be granted. When a router receives a packet, the classifier will perform a Multi-Field (MF) classification and put the packet in a specific queue based on the classification result. The packet scheduler will then schedule the packet accordingly to meet its QoS requirements.

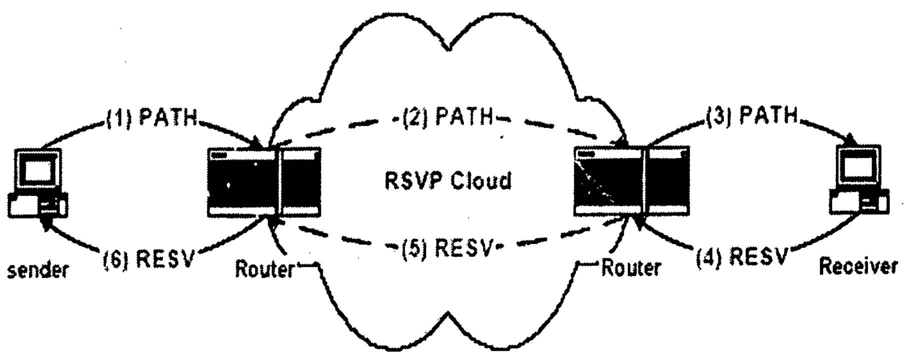


Figure 1. RSVP signaling

The problems with the Integrated Services architecture are:

1) The amount of state information increases proportionally with the number of flows. This places a huge storage and processing overhead on the routers. Therefore, this architecture does not scale well in the Internet core.
2) The requirement on routers is high. All routers must implement RSVP, admission control, MF classification and packet scheduling.
3) Ubiquitous deployment is required for Guaranteed Service. Incremental deployment of Controlled-Load Service is possible by deploying Controlled-Load Service and RSVP functionality at the bottleneck nodes of a domain and tunneling the RSVP messages over other part of the domain.

1.2.2 Differentiated Services

Because of the difficulty in implementing and deploying Integrated Services and RSVP, Differentiated Services (DS) is introduced. IPv4 header contains a TOS byte. Differentiated Services defines the layout of the TOS byte (termed DS field) and a base set of packet forwarding treatments (termed Per-Hop Behaviors, or PHBs). By marking the DS fields of packets differently and handling packets based on their DS fields, several differentiated service classes can be created. Therefore, Differentiated Services is essentially a relative-priority scheme. In order for a customer to receive

Differentiated Services from its Internet Service Provider (ISP), it must have a Service Level Agreement (SLA) with its ISP. A SLA basically specifies the service classes supported and the amount of traffic allowed in each class. A SLA can be static or dynamic. Static SLAs are negotiated on a regular, e.g. monthly and yearly, basis. Customers with Dynamic SLAs must use a signaling protocol, e.g. RSVP, to request for services on demand. Customers can mark DS fields of individual packets to indicate the desired service or have them marked by the leaf router based on MF classification. At the ingress of the ISP networks, packets are classified, policed and possibly shaped. The classification, policing and shaping rules used at the ingress routers are derived from the SLAs. The amount of buffering space needed for these operations is also derived from the SLAs. When a packet enters one domain from another domain, its DS field may be re-marked, as determined by the SLA between the two domains. Using the classification, policing, shaping and scheduling mechanisms, many services can be provided. For example:

1 **Premium Service:** It is for applications requiring reliability, low delay and low jitter service. Real-time traffic (e.g. VoIP and video conferencing) and mission-critical traffic (e.g. financial traffic) can benefit from such a service.

2 **Assured Service:** It is for applications requiring better reliability than Best Effort Service. Non-real time VPN service traffic can benefit from it.

3 **Best effort service:** It is the traditional Internet service.

A customer can choose the appropriate service based on the nature of its traffic (e.g., real-time or interactive or non-interactive). But what service an application will receive is eventually determined by the customer's willingness to pay. So it is possible that one customer's email traffic receives Assured service while another customer's video-conferencing traffic receives Best Effort service.

Note that the Differentiated Services only defines DS fields and PHBs. It is the ISPs' responsibility to decide what services to provide. Differentiated Services is significantly different from Integrated Services. First, there are only a limited number of service classes indicated by the DS field. Since service is allocated in the granularity of a class, the amount of state information is proportional to the number of classes rather than the number of flows. Differentiated Services is therefore more scalable. Second, sophisticated classification, marking, policing and shaping operations are only needed at boundary of the networks. ISP core routers need only to implement Behavior Aggregate (BA) classification. Therefore, it is easier to implement and deploy Differentiated Services. There is another reason why the second feature is desirable for ISPs. ISP networks usually consist of boundary routers connected to customers and core routers/switches interconnecting the boundary routers. Core routers must forward packets very fast and therefore must be simple. Boundary routers need not forward packets very fast because customer links are relatively slow. Therefore, they can spend more time on sophisticated classification, policing and shaping. Boundary routers at the Network Access Points (NAPs) are exceptions. They must forward packets very fast and do sophisticated classification, policing and shaping. Therefore, they must be well equipped.

1.3 Basic building blocks of DS architecture

1.3.1DS code point

When mapping a traffic packet to a PHB, the most significant input data is the code point in the DS field of the IP packet header. Below is the description of the DS field octet.

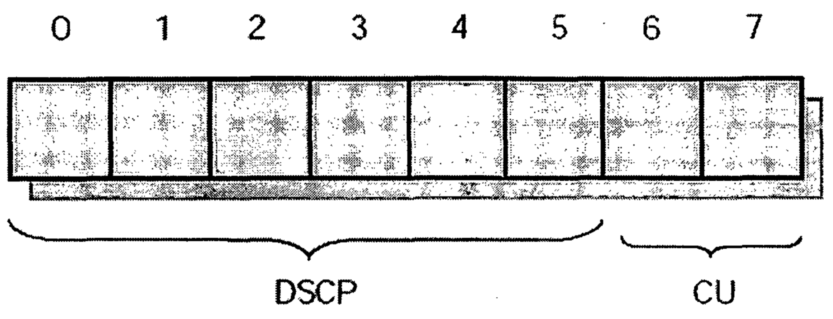


Figure 2. DS Field

As seen, six bits of the DS field are called as the code point (DSCP) and the remaining, which are currently unused, are reserved for the future.

1.3.2 DS domain

When the packets inside the network are forwarded to their destination, each of the packets gets treated only per hop basis during the forwarding path within the network region. It is noteworthy that the per-hop treatment is identical within the DS domain, however the mapping from the DS code-point to PHB may be different in different DS domains. A Ds domain i.e. a set of DS nodes, which operate according to a common service provisioning policy and has a common set of PHBs, is the entity that provides a coherent set of PHBs in the network domain. The DS domains may form a DS region that has the ability to support PHBs that are covering a wider area of domains within the DS region.

1.3.3 Per Hop Behavior (PHB)

A PHB is a description of the externally observable forwarding treatment applied at DS-compliant node to a Behavior Aggregate (BA). The concept of forwarding behavior can be interpreted to mean those aggregate actions that interior nodes perform with packets having a similar code point in the DS field. PHBs are usually needed in cases when several BAs are competing on resources in a node, which is then able to make service discrimination based on defined PHBs in that DS node. For a node, the PHB is foremost the means that can be used allocating resources for different behavior aggregates. PHBs can be identified according to how they prioritize the resources, such as buffers and bandwidth or according to how different PHBs are prioritized or how the traffic can be observed in terms of delay and loss. Also PHBs can be used as resource allocation building blocks and thus be grouped together. PHBs are implemented in boundary and interior nodes, usually by means of existing buffer management and scheduling algorithms. Also, in a node, here are possibly more than one PHB implementations that can be grouped together into aggregates or may be separate to each other. This basically implies that the PHB implementations need to be such that they do not prevent others to be operable.

1.3.4 PHB standardization

With respect to PHB standardization, the characteristics of PHBs are the subject to be standardized, not the actual algorithms that implement the PHBs. As the DSCP (DS Code Point) is an octet, there is a total no. of 64 code-points available to be used as PHB standardization. The code point space consists of three pools as follows—

Pool 1: A pool of 32 recommended PHBs to be assigned to standard actions.
Bit pattern: 'xxxxx0'.

Pool 2: A pool of 16 code points to be reserved for experimental or local use.
Bit pattern: 'xxxx11'.

Pool 3: A pool of 16 code points that can be used for experimental or local use, but which should be used for standardization use in case pool 1 gets overloaded.
Bit pattern: 'xxxx01'.

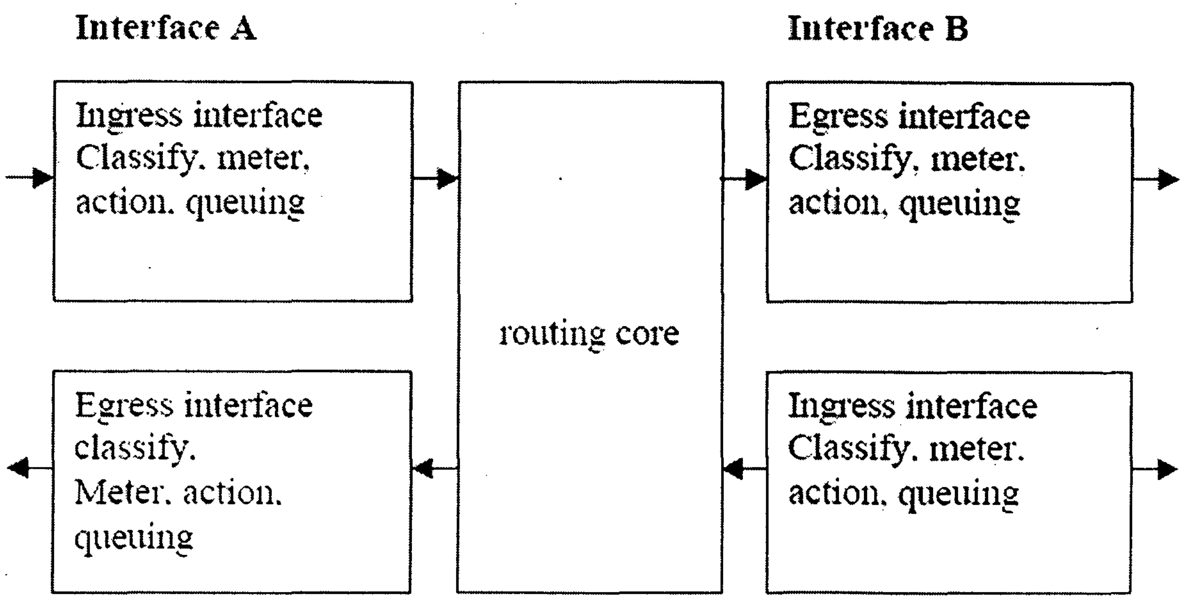
Examples of standardized PHBs:

1. **Assured Forwarding (AF) PHB group:** AF PHB group provides forwarding of IP packets in N independent AF classes, in which there are M different levels of drop precedence.

2. **Expedited Forwarding (EF) PHB group:** The EF PHB group can be used for providing services like 'virtual leased line' in which characteristics such as low loss, low latency, low jitter and assured bandwidth play significant role.
   These kinds of services have two parts:
   1. Possibility to configure the DS nodes in a manner that enables minimum departure rate.
   2. Policing and shaping the PHB aggregate so that its arrival rate at any DS node is always less than that DS node's configured minimum departure rate.

1.4 Conceptual functioning of DiffServ architecture

We have four QoS control functions in the data-path on traffic in each direction (figure 3). First we have the classification of messages according to some set of rules. The second action determines whether the data stream of the message is within or outside its rate by performing measurements on the stream. In addition, based on these measurements the router can perform a set of resulting actions according to drop policy, and mark the traffic with a Differentiated Services field. The traffic must be enqueued for output in the appropriate queue. The queue may shape the traffic or forward it with some minimum rate or maximum latency. This section presents the functional data-path elements in the conceptual Diffserv model and how they can be interconnected and configured.



Interface A
Interface B
Ingress interface Classify. meter. action. queuing
Egress interface Classify. meter. action, queuing
routing core
Egress interface classify. Meter. action. queuing
Ingress interface Classify. meter. action, queuing

Figure 3. Data-path in both directions between ingress and egress interface.

1.5 The security aspect of DS architecture

The security aspects of DSCP can be divided into two categories: theft and denial of services, and IPSec and tunneling interactions. The theft and denial of service defense takes place in the boundary nodes, and therefore DS domain boundary nodes must ensure that all traffic entering the domain is marked with code point values appropriate to the traffic and the domain. In IPSec tunneling, packet contains two IP headers: an outer header supplied by the tunnel ingress node and an encapsulated inner header supplied by the original source of the packet. In this case, the intermediate tunneling nodes operate on outer headers. When the tunneling packet enters the egress node, the IPSec processing removes the outer header and continues to forward the inner header to ingress nodes.

In the Differentiated Services model, incremental deployment is possible for Assured Service. DS incapable routers simply ignore the DS fields of the packets and give the Assured Service packets Best Effort Service. Since Assured Service packets are less likely to be dropped by DS-capable routers, the overall performance of Assured Service traffic will be better than the Best Effort traffic.

1.6 Problem definition

While the traffic traverses from the source to the destination, the main issues which are to be addressed and measured to evaluate the QoS are the following:

1. Proper bandwidth utilization
2. Delay measurement
3. Packet loss
4. Sequential delivery
5. Smooth flow of the traffic

There must be proper bandwidth utilization by the traffic and if some bandwidth can be given to another requiring queue of the traffic then the provisions should be made to share the bandwidth. There are services, which require low delay and low jitter. Traffic rate is controlled in such a way to give the best service to the user. There is a counter at each DS node to count the number of lost packets. For guaranteeing the sequential delivery, the appropriate routing algorithms are used to avoid any reordering. Buffer storage at the DS nodes is utilized properly in order to maintain the smooth flow of the traffic.

CHAPTER 2
PACKET FORWARDING MECHANISM FOR THE SIMULATION: ASSURED FORWARDING

In this simulation model, the packets are forwarded from end to end to provide QoS over and above the current best effort service. The AF is intended to assure a minimum level of throughput. To assure this minimum throughput, which is referred to as the target rate or committed information rate (CIR), AF introduces two components; a packet marking mechanism administrated by profile meters or traffic conditioners at edge routers and a queue management mechanism at core routers. The packet marking mechanism, monitors and marks packets according to the service profile at the edge of a network. If the measured flow conforms to the service profile, the packets belonging to this flow are marked with a high priority and receive better service. Otherwise, the packets belonging to the non-conformant part of a flow are marked with a low priority and receive best effort service. The queue management mechanism, deployed at core routers, provides preferential treatment for packets that have high priority. During times of congestion, high priority packets are forwarded preferentially and low priority packets are dropped with higher probability. This service guarantees a timely forwarding of packets within an agreed time frame.

In this service environment the IP packets are forwarded in *N* independent AF classes, in which there are *M* different levels of drop precedence. Therefore, the code-points with the classes and drop precedence form a matrix AFij, where $1 \leq i \leq N$ and $1 \leq j \leq M$. The values of N and M can be suitably chosen. In this simulation study the values are chosen to be four classes (N=4) with three drop-precedence (M=3). DS nodes must not aggregate two or more classes together i.e. two packets with the same class code-point but different drop precedence will be treated differently. There are following considerations which must be supported:

1 . AF class must be configurable to receive more forwarding resources than minimum.

2 . DS node must not break the precedence order within an AF class.

3 . All three drop-precedence code-points must be accepted in each of the AF classes.

4 . Code-points must not supersede the ordering in drop-precedence order.

5 . The DS node must not reorder the packets in the AF micro-flow (the end to end traffic).

6 . The DS node must minimize the long-term congestion while short-term bursts should be allowed.

7 . The dropping algorithm applied must be insensitive to short-term traffic characteristics, and treat packets in similar classes and code-points identically.

8 The dropping must be gradual, instead of abrupt one.

9 . The dropping algorithm parameters must be configurable.

The recommended code-points are as follows:

AF11 = '001010' AF12 = '001100' AF13 = '001110'
AF21 = '010010' AF22 = '010100' AF23 = '010110'
AF31 = '011010' AF32 = '011100' AF33 = '011110'
AF41 = '100010' AF42 = '100100' AF43 = '100110'

In the Differentiated Services model, incremental deployment is possible for Assured Service. DS incapable routers simply ignore the DS fields of the packets and give the Assured Service packets Best Effort Service. Since Assured Service packets are less likely to be dropped by DS-capable routers, the overall performance of Assured Service traffic will be better than the Best Effort traffic.

CHAPTER 3

ALGORITHMS/METHODOLOGY USED IN THE SIMULATION

Queuing systems store packets, they modulate the departure of packets of different streams and they selectively discard packets. The model decomposes the queuing block into the component elements that perform each of these functions: Queues, Schedulers and Algorithmic Droppers. These elements can be combined to a TCB containing one or more Queue and Scheduler elements and zero or more Algorithmic Droppers. Queues are simply places to store traffic until it can be transmitted. A Scheduler has one or more inputs and exactly one output and it schedules the departing order of the packets according to some algorithm. The Algorithmic Dropper selectively discards packets that go through it based on its discarding algorithm. The queuing Block is constructed from these three functional data-path elements and they may appear more than once in parallel or in series. In this simulation model, different queuing, scheduling, and dropping algorithms are used to evaluate their effect on QoS. Few of them frequently used are described below:

3.1 WRR (Weighted Round Robin) Algorithm

WRR algorithm is the extension of basic Round-Robin algorithm. In the Round Robin algorithm queues are served or allocated resources in a round robin fashion depending upon a fixed time slot. A particular queue will be serviced during its slot period after which it will have to give in the way to the next queue in the line. Next time it will be serviced only after all the queues are serviced in the round in the similar fashion. In WRR algorithm weights are assigned to each queue, which means that different queues can have different time slots and they are serviced during these time slots. In this way a priority can be assigned to each queue.

3.2 Class Based Queuing (CBQ)

It provides a unique solution for the traffic management by providing bandwidth-controlled QoS. class-based queuing offers a flexible approach to sharing link bandwidth across a hierarchy of traffic types. Each class can represent an aggregation of traffic or an individual connection. There are two basic attributes supporting the working of CBQ:

3.2.1 Priority levels

It allows higher priority queues to be serviced first within the limits of their bandwidth allocation so that the delay for more sensitive real-time traffic classes is reduced.

3.2.2 Borrowing privileges

It is an explicit class-based queuing mechanism for distribution of excess bandwidth. A class with borrowing privileges may initiate a borrowing request when it needs more bandwidth. If another class is not using its full bandwidth, synchronization features indicate that bandwidth is available. Borrowed bandwidth is granted to higher-priority classes before lower-priority classes.

3.3 Drop Tail Queuing

The basic queue algorithm for routers is the Drop Tail queuing algorithm. Drop Tail queues simply accept any packet that arrives when there is sufficient buffer space and drop any packet that arrives when there is insufficient buffer space. In this way, Drop Tail queuing poses a strict boundary limit on the no. of allowed packets in a link.

3.4 RED Queuing

RED (Random Early Detection) is a congestion avoidance algorithm that can be implemented in routers. RED gateways attempt to detect congestion by computing a weighted average queue size, since a sustained long queue is a sign of network congestion. For a RED gateway the following three phases sum up its algorithm:

Phasel: Normal Operation

If the average queue size is less than the minimum threshold, no packets are dropped.

Phase2: Congestion Avoidance

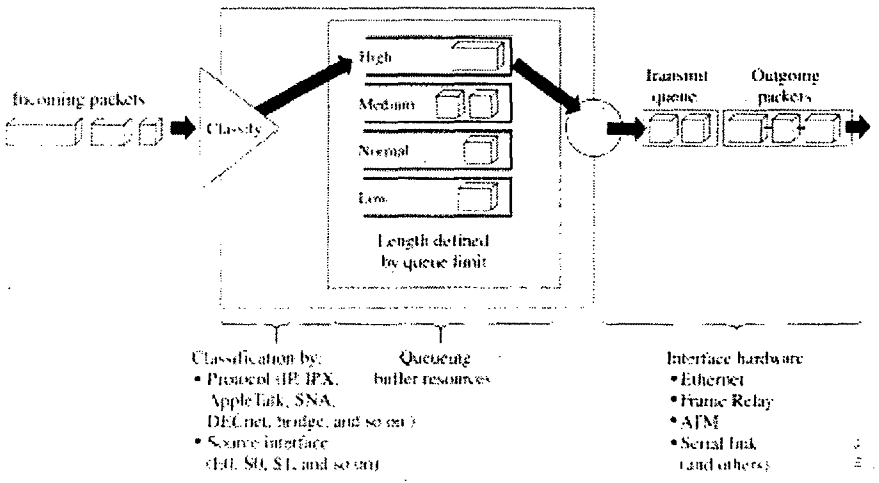
If the average queue-size is between the minimum and maximum thresholds, packets are dropped with a certain probability. This probability is a function of the average queue size, so that larger queues lead to higher drop probabilities.

Phase3: Congestion Control

If the average queue size is greater than the maximum threshold, all incoming packets are dropped. Upon packet arrival, a RED gateway checks the weighted average queue size against specified minimum and maximum thresholds. If there is congestion, it notifies, either by dropping a packet or by setting a bit in a header field of the packet, probabilistically.

3.5 Priority Queuing (PQ)

It ensures that important traffic is processed first. It was designed to give strict priority to a critical application, and is particularly useful for time-sensitive protocols such as SNA. Packets can be prioritized based on many factors, including protocol, incoming interface. packet size, and source or destination address. Priority queuing is especially appropriate in cases where WAN links are congested from time to time. If the WAN links are constantly congested, the customer should investigate protocol and application inefficiencies, consider using compression, or possibly upgrade to more bandwidth. If the WAN links are never congested, priority queuing is unnecessary. Because priority queuing requires extra processing and can cause performance problems for low-priority traffic, it should not be recommended unless necessary.



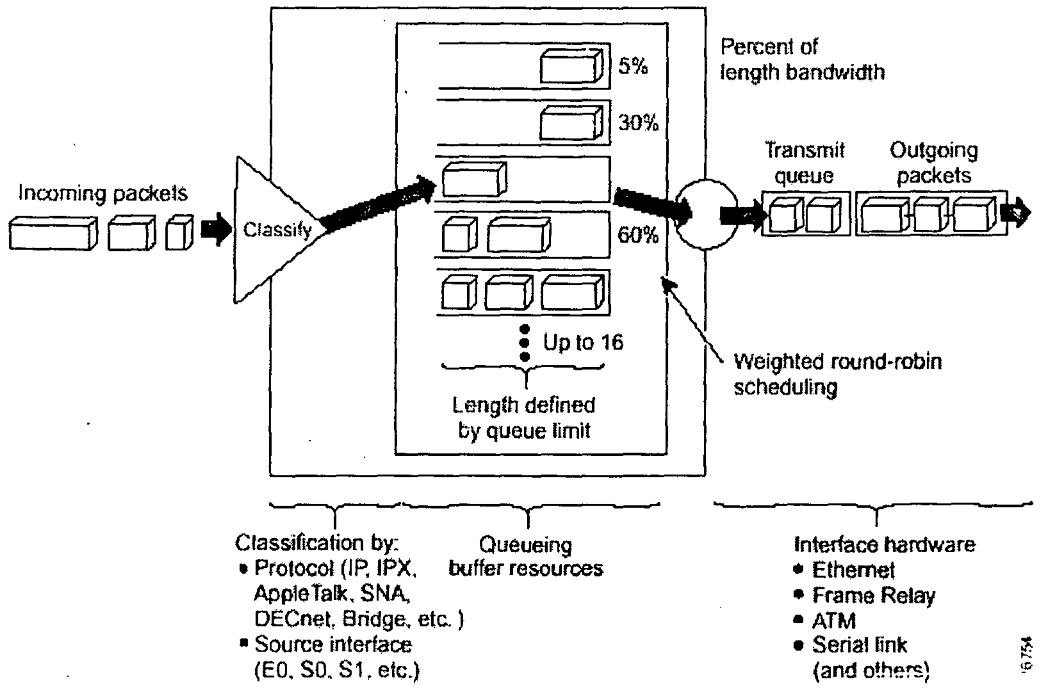
Incoming packets
Classify
High
Medium
Normal
Low
Length defined by queue limit
Transmit queue
Outgoing packets
Classification by:
• Protocol (IP, IPX, AppleTalk, SNA, DECnet, bridge, and so on)
• Source interface (E0, S0, S1, and so on)
Queueing buffer resources
Interface hardware
• Ethernet
• Frame Relay
• ATM
• Serial link (and others)

Figure 4. The behavior of Priority Queue

3.6 Custom Queuing

It was designed to allow a network to be shared among applications with different minimum bandwidth or latency requirements. Custom queuing assigns different amounts of queue space to different protocols and handles the queues in round-robin fashion. A particular protocol can be prioritized by assigning it more queue space. Custom queuing is more "fair" than priority queuing, although priority queuing is more powerful for prioritizing a single critical application. You can use custom queuing to provide guaranteed bandwidth at a potential congestion point. Custom queuing lets you assure each specified traffic type a fixed portion of available bandwidth, while at the same time avoid any application achieving more than a predetermined proportion of capacity when the link is under stress.

Custom Queueing

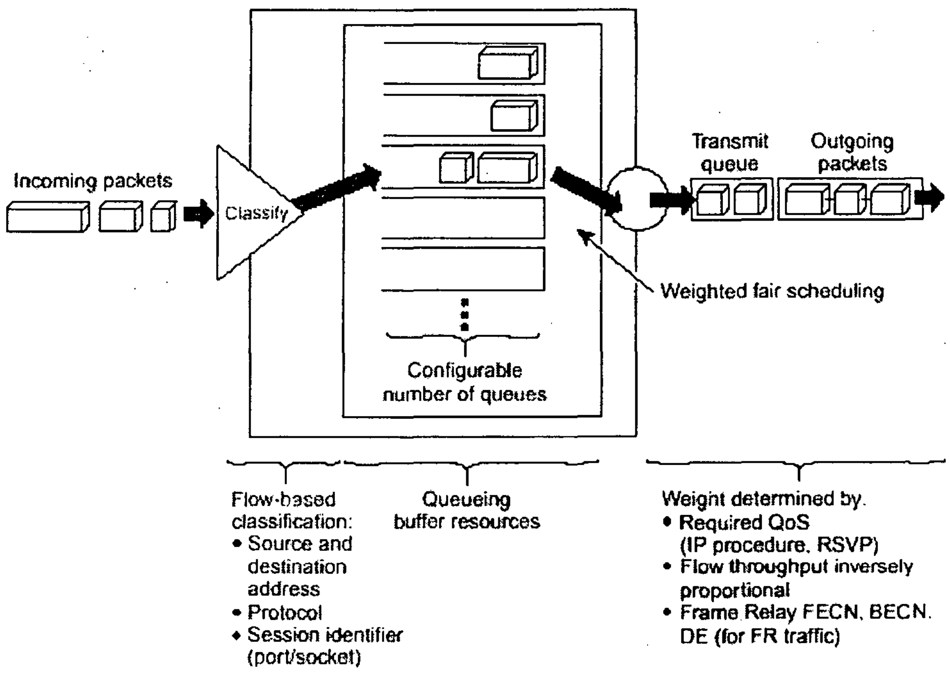


Percent of length bandwidth
5%
30%
60%
Transmit queue
Outgoing packets
Incoming packets
Classify
Up to 16
Weighted round-robin scheduling
Length defined by queue limit
Classification by:
• Protocol (IP, IPX, AppleTalk, SNA, DECnet, Bridge, etc.)
• Source interface (E0, S0, S1, etc.)
Queueing buffer resources
Interface hardware
• Ethernet
• Frame Relay
• ATM
• Serial link (and others)

3.7 Weighted Fair Queuing (WFQ)

It is an algorithm designed to reduce delay variability and provide predictable throughput and response time for traffic flows. A goal of WFQ is to offer uniform service to light and heavy network users alike. WFQ ensures that the response time for low-volume applications is consistent with the response time for high-volume applications. Applications that send small packets are not unfairly starved of bandwidth by applications that send large packets. Unlike custom and priority queuing, WFQ adapts automatically to changing network traffic conditions and requires little to no configuration.

Weighted Fair Queueing


Incoming packets
Classify
Configurable number of queues
Transmit queue
Outgoing packets
Weighted fair scheduling
Flow-based classification:
• Source and destination address
• Protocol
• Session identifier (port/socket)
Queueing buffer resources
Weight determined by.
• Required QoS (IP procedure, RSVP)
• Flow throughput inversely proportional
• Frame Relay FECN, BECN, DE (for FR traffic)

For applications that use the IP precedence field in the IP header, WFQ can allot bandwidth based on precedence. The algorithm allocates more bandwidth to conversations with higher precedence, and makes sure those conversations get served more quickly when congestion occurs. WFQ assigns a weight to each flow, which determines the transmit order for queued packets. IP precedence helps determine the weighting factor.

3.8 Packet-based Generalized Processor Sharing (PGPS)

Weighted fair queuing (WFQ) is considered to be one of the critical components in realizing the DiffServ model. It defines the scheduling policy for the out going link bandwidth being shared among all the classes. The goal of the WFQ is to provide a packet-based approximation of the Generalized Processor Sharing (GPS). A GPS scheduler can be regarded as the limiting form of a weighted round robin policy, where traffic from each session is treated as infinitely divisible fluid. It is characterized by positive weights w1, w2, ..., wN, for given N different classes. If the bandwidth of the out-going link is W, backlogged class i is guaranteed a rate $r_i = v_i W$ where v_i is equal to w_i divided by the sum of weights of the all backlogged queues.

3.9 Weighted Random Early Detection (WRED)

WRED is Cisco's implementation of RED. WRED combines the capabilities of the standard RED algorithm with IP precedence. This combination provides for preferential traffic handling for higher-priority packets. It selectively discards lower-priority traffic when an interface starts to get congested, rather than using simply a random method.

3.10 RIO algorithm

Another popular uses for RED is Random Early Detection with In/Out bit (RIO), which uses some form of packet tagging to indicate the drop priority of the packet to the core network routers. Each user (or traffic flow, depending upon the granularity) is assigned a service profile by the ISP based on the expected bandwidth utilization by the user. At the edge of the network domain, managed by the ISP (i.e., at the ingress points), user traffic is monitored by a profile meter to ensure that it stays within the profile. Any packets that are out of profile are marked as "out" while those that conform to the user profile, are marked as "in". In the network core, the "in" and "out" packets are treated with different drop priorities using RED. In short, "in" packets start being dropped only when the queue size crosses a higher threshold than in the case of "out" packets and get dropped with a lower probability than "out" packets. This ensures that in-profile traffic has less chance of getting dropped than out-of-profile traffic, and therefore gets predictable levels of service so long as it stays within profile (even when congestion occurs). The RIO scheme also makes it possible to use statistical multiplexing to utilize any excess bandwidth that may be available since it does not prevent out-of-profile packets from entering the network.

For the smooth flow of the traffic token bucket algorithm is used which is described below-

3.11 Token Bucket Algorithm

A token bucket algorithm is used to control the flow rate of the traffic. A token bucket is a formal definition of a rate of transfer. It has three components: a burst size, a mean rate, and a time interval. Although the mean rate is generally represented as bits per second, any two values may be derived from the third by the relation shown as follows: mean rate = burst size / time interval

Here are some definitions of these terms:

• Mean rate—Also called the committed information rate (CIR), it specifies how much data can be sent or forwarded per unit time on average.

• Burst size—Also called the Committed Burst size (CBS), it specifies in bytes per burst how much traffic can be sent within a given unit of time to not create scheduling concerns. For a shaper, it specifies bits per burst; for a policer, it specifies bytes per burst.

• Time interval—Also called the measurement interval, it specifies the time quantum in seconds per burst.

By definition, over any integral multiple of the interval, the bit rate of the interface will not exceed the mean rate. The bit rate, however, may be arbitrarily fast within the interval. A token bucket is used to manage a device that regulates the data in a flow. For example, the regulator might be a traffic policer or a traffic shaper. A token bucket itself has no discard or priority policy. Rather, a token bucket discards tokens and leaves to the flow the problem of managing its transmission queue if the flow overdrives the regulator.

In the token bucket algorithm, tokens are put into the bucket at a certain rate. The bucket itself has a specified capacity. If the bucket fills to capacity, newly arriving tokens are discarded. Each token is permission for the source to send a certain number of bits into the network. To send a packet, the regulator must remove from the bucket a number of tokens equal in representation to the packet size. If not enough tokens are in the bucket to send a packet, the packet either waits until the bucket has enough tokens or the packet is discarded or marked down. If the bucket is already full of tokens, incoming tokens overflow and are not available to future packets. Thus, at any time, the largest burst a source can send into the network is roughly proportional to the size of the bucket. Note that the token bucket mechanism used for traffic shaping has both a token bucket and a data buffer, or queue; if it did not have a data buffer, it would be a policer. For traffic shaping, packets that arrive that cannot be sent immediately are delayed in the data buffer. For traffic shaping, a token bucket permits burstiness but bounds it. It guarantees that the burstiness is bounded so that the flow will never send faster than the capacity of the token bucket plus the time interval multiplied by the established rate at which tokens are placed in the bucket. It also guarantees that the long-term transmission rate will not exceed the established rate at which tokens are placed in the bucket.

CHAPTER 4
SIMULATION ENVIRONMENT

4.1 Introduction to NS

The simulation environment for the implementation of the DS architecture is chosen to be the NS network simulator. NS is a free network simulation program that can be downloaded from the Internet and is compatible with a number of operating systems. The tool has substantial functionality for simulating different network topologies and traffic models. NS also has an open architecture that allows users to add new functionality. NS has been developed at the Lawrence Berkeley National Laboratory (LBNL) of the University of California, Berkeley (UCB). The extensibility of ns makes the tool very dynamic. ns is an event-driven network simulator. It has an extensible background engine implemented in C++ that uses OTcl (an object oriented version of Tcl) as the command and configuration interface. Thus, the entire software hierarchy is written in C++, with OTcl used as a front end. Nam is the network simulator, which provides visual views of the network simulation. Xgraph is used to obtain statistics and produce graphical results under the Unix/Linux platform. Alternatively, Tracegraph is used with Windows platform for the same purpose.

4.2 Tcl Scripts

A simulation is defined by an OTcl script. Running a simulation involves creating and executing a file with a ".tcl" extension, such as "simfile.tcl."

A Tcl ns script:

a. Defines a network topology (including the nodes, links, and scheduling and routing algorithms of a network).

b. Defines a traffic pattern (for example, the start and stop time of an FTP session).

c. Collects statistics and outputs the results of the simulation. Results are usually written to files, including files for Nam, the Network Animator program that comes with the full ns download.

NS is an event-driven simulator that derives its functionality through an OTcl interpreter, which runs in the background. This interpreter translates each statement in the Tcl file into an event. For example, the statement:

$ns at 0.5 "$tcp start"

is translated into event, which at 0.5 seconds into the simulation, starts up a TCP source.

4.3 Software Architecture

NS is an object-oriented simulator written in C++. This code serves as a backbone for the whole simulation process. The entire class hierarchy is implemented through this code and the classes provide a wide array of network features. New classes or modules can be added by extending the current class hierarchy. Each class consists of the following components:

4.3.1 Configuration parameters

Configuration parameters are class attributes that can be set and queried dynamically through the Tcl scripts. These simulation parameters are usually constant during the entire simulation, but they can be changed dynamically as desired. For example, the bandwidth of a link is usually set only at the start of a simulation. On the other hand, a traffic source can be configured to transmit packets of different sizes at different times.

4.3.2 State variables

Each class has a set of variables, many of which may be queried in a Tcl script to determine the state of that object. Usually, they are modified explicitly only when the object needs to be reset for another simulation run. For example, the length of a packet queue changes over time; and the instantaneous size of that queue can be queried through a Tcl command or used by a C++ method.

4.3.3 Methods

The methods associated with each class specify the actions that can be performed by that object. For example the *enque()* method for the RED gateway class specifies the enqueuing method for that object.

4.4 Adding a New Module

This section outlines the process of creating and adding new classes to the ns software hierarchy. Adding a new module to ns consists of three steps:

4.4.1 Determining the need

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

a. The ability to implement multiple physical RED queues along a single link.

b. The ability to implement multiple virtual queues on each physical queue, with independent sets of parameters for each virtual queue.

c. The ability to determine in which physical and virtual queue a packet is enqueued, depending upon user specifications.

4.4.2 Determining the class hierarchy positioning

Since this new class implements the generic Queue functionality and extends it with new methods and attributes, it is placed beneath the class Queue in the object hierarchy.

4.4.3 Writing code

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

Step 1: Creating the header file ("dsred.h")

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

Step 2: Creating the main C++ file ("dsred.cc")

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

```
static class dsREDClass : public TclClass {
public:
dsREDClass() : TclClass("Queue/dsRED") {}
TclObject* create(int, const char*const*) {
return (new dsREDQueue);
}
} class_dsred;
```

Step 3: Modifying "Makefile"

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

Step 4: Specifying default parameters for bound variables

A fourth step is necessary if configuration parameters are bound in the class constructor method. In that case, default values for all bound parameters should be added to the file "/ns-2/tcl/lib/ns-default.tcl." For example, the dsREDQueue constructor contains the binding:

```
bind("numQueues_", &numQueues_);
```

and the parameter is assigned default values in "/ns-2/tcl/lib/nsdefault.tcl":

```
Queue/dsRED set numQueues_ 4
```

After completing those steps and recompiling ns with the make command, Tcl scripts can use the new class.

The Assured Forwarding scheme has been proposed as a potential user of Diffserv. Assured Forwarding provides differential treatment of traffic by discarding more low priority packets during times of congestion than high priority packets. Although the Assured Forwarding mechanism does not explicitly require a particular queue type, it is suited for RED.

4.5 Diffserv Architecture NS Modules

4.5.1 Introduction

In order to design and implement the Diffserv architecture in ns, five modules had to be added to the class hierarchy: one for the base Diffserv router functionality (dsRED), one each for the edge and core routers, one for RED-based queuing and one for policing. Each module defines a single class.

4.5.2 dsRED Module

The dsRED module is the base module for the Diffserv implementation. It defines the dsREDQueue class, which is the parent class for the edgeQueue and coreQueue classes. The dsRED module is contained in the files "dsred.h" and "dsred.cc."

4.5.2.1 Purpose

The dsREDQueue class is the parent class for the edgeQueue and coreQueue classes. It implements all functionality and declares all parameters that are common to edge and core Diffserv routers.

4.5.2.2 Class hierarchy positioning

The dsREDQueue class extends the Queue class.

4.5.2.3 Graphical representation

A dsREDQueue uses the redQueue class, to form the following queue structure:
The queue structure consists of four physical queues, each containing three virtual RED queues, referred to as precedence levels. Each physical queue corresponds to a class of traffic; and each combination of a queue and precedence number is associated with a code point (or a drop preference).

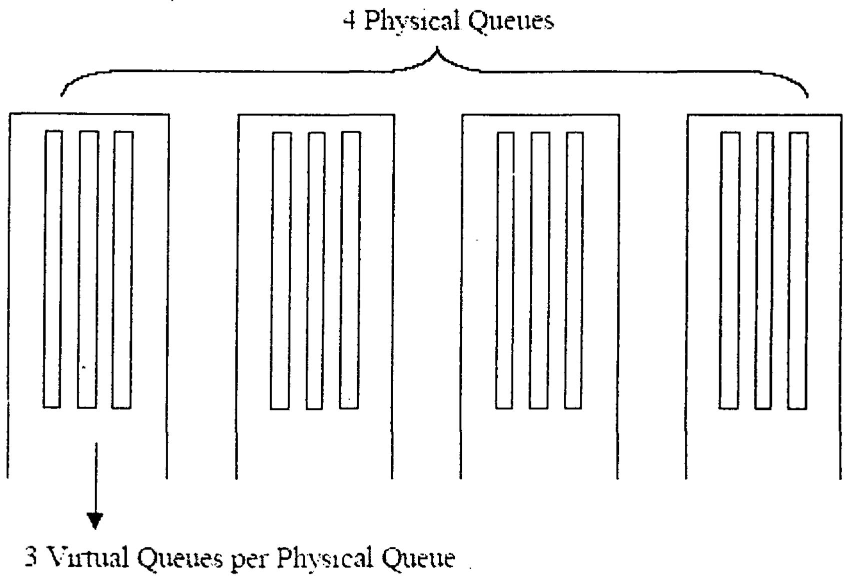


Figure 5. A dsREDQueue Instance

Packets are enqueued in a certain queue and precedence number according to their code point marking. They are treated according to the corresponding parameters for that queue and precedence number; thus, a code point specifies a certain level of service. Not all physical and virtual queues need be used, the user may configure a dsRED instance to have fewer physical or virtual queues through the Tcl script. These values may not be exceeded, however, without first altering the constants defined in "dsred.h" and recompiling ns.

4.5.3 PHB Table

The dsREDQueue class contains a data structure known as the PHB Table (per hop behaviour table). Edge devices handle marking packets with code points and core devices simply respond to existing code points. However, both devices need to determine how to map a code point to a particular queue and precedence level. The PHB Table handles this mapping by defining an array with three fields:

- Code point

- . Class (Physical Queue)
- . Precedence (Virtual Queue)

4.5.3.1 Tcl Configuration

The configuration commands listed in this section apply to edgeQueue and coreQueue instances, since both classes are children of dsREDQueue. Router configuration must be handled in the simulation before the arrival of any traffic. The dsREDQueue class has one bound variable: *numQueues_*. The default value for numQueues_ is set in the file "/ns-2/tcl/lib/ns-default.tcl" as 4. This value can be changed inside Tcl scripts as follows, assuming that *dsredq* is a Tcl variable referring to a dsREDQueue (or child) object:

```
$dsredq set numQueues_ 1
```

numQueues_ refers to the number of physical queues.

The number of virtual queues is not a bound variable, but can be configured with the command:

```
$dsredq setNumPrec 2
```

The numbers of physical and virtual queues are limited by constants inside the file "dsred.h," which should not be exceeded. No error checking is performed on the numQueues_ variable; it is assumed that the user will not exceed 4 physical queues. In general, only limited error-checking is performed on the Tcl configuration commands.

```
$dsredq configQ 0 1 10 20 0.10
```

This command configures the RED parameters for one virtual queue. The above example specifies that physical queue 0/virtual queue 1 has a min_{th} value of 10 packets, a max_{th} value of 20 packets, and a max_p value of 0.10. For DropTail queues, only the first three parameters are required and the second is disregarded because there is no notion of precedence level for a DropTail queue.

```
$dsredq addPHBEntry 11 0 1
```

The addPHBEntry command adds an entry to the PHB Table; in this example, code point 11 is mapped to physical queue 0 and virtual queue 1. In ns, the packets are defaulted to a code point of zero. Therefore, to handle best effort traffic one must add a PHB entry for the zero code point.

```
$dsredq meanPktSize 1500
```

This command specifies the mean packet size (in bytes), which is used for RED calculations. In addition, commands are available which allow the user to choose the scheduling mode between queues.

```
$dsredq setSchedularMode WRR
$dsredq addQueueWeights 1  5
```

The above pair of commands sets the scheduling mode to Weighted Round Robin and then sets the queue weight for queue 1 to 5. Other scheduling modes supported are Weighted Interleaved Round Robin (WIRR), Round Robin (RR), and Priority (PRI). The default scheduling mode is Round Robin.

For Priority scheduling, priority is arranged in sequential order with queue 0 having the highest priority. Also, one can set a limit on the maximum bandwidth a particular queue can get using the addQueueRate command.

```
$dsredq setSchedularMode PRI
$dsredq addQueueRate 0 5000000
```

For example, the above set of commands set the scheduling mode to Priority and puts a limit on the queue 0 bandwidth to 5 Mbps.

4.5.3.2 Tcl Querying

The values of the bound variables may be checked from a script; and the dsREDQueue Tcl interface also interprets three additional queries:

```
$dsredq printPHBTable
```

This command prints the entire PHB Table, one line at a time.

```
$dsredq printStats
```

This command is meant to be a debugging tool that can be altered as needed. Currently, it prints the defined number of physical and virtual queues.

```
$dsredq getAverage 0
```

This query returns the RED weighted average size of the specified physical queue.

4.5.4 redQueue Class

redQueue class defines a physical RED queue composed of multiple virtual queues. The dsRED module is contained in the files "dsredq.h" and "dsredq.cc." The redQueue class implements a single physical RED queue that contains multiple virtual queues. One underlying physical queue incorporates multiple virtual RED queues, each of which has a distinct set of RED parameters.

4.5.4.1 Purpose

The purpose of the Diffserv architecture is to provide different treatment to different traffic. The redQueue class enables that differentiation by defining virtual RED queues, each of which has independent configuration and state parameters. For example, the length of each virtual queue is calculated only on packets mapped to that queue. Thus, packet dropping decisions can be applied based on the state and configuration parameters of the virtual queues. The redQueue class is not equivalent to the REDQueue class, which was already present in ns. Instead, it is a modified copy of that class that includes the notion of virtual queues.

4.5.4.2 Tcl Configuration and Querying

Instances of the redQueue class only exist inside instances of the dsREDQueue class. All user interaction with the redQueue class is handled through the command interface of the dsREDQueue class.

4.5.5 Edge Module

The edge module implements a Diffserv edge router. It defines the *edgeQueue* class, which models an edge router. The edge module is contained in the files "edge.h" and "edge.cc."

4.5.5.1 Purpose

The edgeQueue class, as a child of the dsREDQueue class is responsible for maintaining multiple physical and virtual queues and processing those queues according to their parameters. Its additional responsibilities are:

- . Marking packets with code points.
- . Policing traffic aggregates.

4.5.5.2 Class Hierarchy positioning

The edgeQueue class is an extension of the dsREDQueue class.

4.5.6 Policy

The edgeQueue class contains an instance of the Policy class that handles all policy creation and enforcement. The Policy class is examined in Section 3.3.2. The *Policy* class is used by the edgeQueue class to handle all policy functionality. The policy module is contained in the files "dsPolicy.h" and "dsPolicy.cc."

4.5.6.1 Purpose

The Policy class handles the creation, manipulation, and enforcement of edge router policies. A policy determines the treatment that a traffic aggregate will receive at the edge device. Edge devices use policy information to determine with what code point to mark packets.

4.5.6.2 Policy Overview

A policy is established between a source and destination node. All flows matching that source-destination pair are treated as a single traffic aggregate. Each policy defines a policer type, a target rate, and other policer-specific parameters. As a minimum, each policy defines two code points, and the choice of code point depends on a comparison between the aggregate's target rate and current sending rate.

4.5.6.3 Policy Table

The Policy class uses a Policy Table to store the policies of each traffic aggregate. This table is an array that includes fields for the source and destination nodes, a policer type, a meter type, an initial code point, and various state information. For each policer type, only some of the state variables are used. The wasted memory space taken up by the unused fields is not considered significant. The fields of the Policy Table are:

- Source node ID
- Destination node ID
- Policer type
- Meter type
- Initial code point
- CIR (committed information rate)
- CBS (committed burst size)
- C bucket (current size of the committed bucket)
- EBS (excess burst size)
- E bucket (current size of the excess bucket)
- PIR (peak information rate)
- PBS (peak burst size)

- . P bucket (current size of the peak bucket)
- . Arrival time of last packet
- . Average sending rate
- . TSW window length

4.5.6.4 Policer Types

The Policy class currently supports six policer types:

TSW2CM (TSW2CMPolicer): uses a CIR and two drop precedences. The lower precedence is used probabilistically when the CIR is exceeded.

TSW3CM (TSW3CMPolicer): uses a CIR, a PIR, and three drop precedences. The medium drop precedence is used probabilistically when the CIR is exceeded and the lowest drop precedence is used probabilistically when the PIR is exceeded.

Token Bucket (tokenBucketPolicer): uses a CIR and a CBS and two drop precedences. An arriving packet is marked with the lower precedence if and only if it is larger than the token bucket.

Single Rate Three Color Marker (srTCMPolicer): uses a CIR, CBS, and an EBS to choose from three drop precedences.

Two Rate Three Color Marker (trTCMPolicer): uses a CIR, CBS, PIR, and a PBS to choose from three drop precedences.

4.5.6.4 Meter Types

The metering and policing methods are decoupled inside “edge.cc”, but currently each policer type maps to a specific policer. Each of the other policer types has its own associated meter.

4.5.6.5 Policer Table

Each policer takes an initial code point and chooses whether to retain it or downgrade it. The downgrading is consistent within a policer type. If two aggregates use the same policer and initial code point, each is downgraded to the same code point(s). The Policy class uses a Policer Table to store the mappings from a policy type and initial code point pair to its associated downgraded code point(s). This table is an array with four fields:

- Policer type
- Initial code point
- Downgraded code point 1
- Downgraded code point 2

The last field is not used for policer types with only two drop precedences, and it should be set to the worst code point for policer types with three drop precedences.

4.5.6.6 Configuration

The addPolicyEntry command is used to add an entry to the Policy Table. It takes different parameters depending on what policer type is used. The first two parameters after the command name are always the source and destination node IDs, and the next parameter is the policer type. Following the policer type are the parameters needed by that policer as summarized below:

- TSW2CM Initial code point CIR
- TSW3CM Initial code point CIR PIR
- TokenBucket Initial code point CIR CBS
- srTCM Initial code point CIR CBS EBS
- trTCM Initial code point CIR CBS PIR PBS

The rates CIR and PIR are specified in bits per second, the buckets CBS, EBS, and PBS are specified in bytes. Consider a Tcl script for which $q is a variable for an edge queue, and $s and $d are source and destination nodes. The following command adds a TSW2CM policer for traffic going from the source to the destination:

```
$q addPolicyEntry [$s id] [$d id] TSW2CM 10 2000000
```

The following parameters could be used in place of "TSW2CM . . ." to use a different policer:

```
TSW3CM 10 2000000 3000000
TokenBucket 10 2000000 10000
srTCM 10 2000000 10000 20000
trTCM 10 2000000 10000 3000000 10000
```

Note, however, that only one policy can be applied to any source-destination pair. The following command adds an entry to the Policer Table, specifying that the trTCM initial (green) code point 10 should be downgraded to yellow code point 11 and red code point 12:

```
$dsredq addPolicerEntry trTCM 10 11 12
```

There must be a Policer Table entry in place for every policer type/initial code point pair.

4.5.6.7 Querying

Four queries are interpreted by an edgeQueue class instance:

```
$dsredq printPolicyTable
```

This command prints the entire Policy Table, one line at a time.

```
$dsredq printPolicerTable
```

This command prints the entire Policer Table, one line at a time.

```
$dsredq getCBucket
```

This query returns the current size of the C Bucket, in bytes.

4.5.7 Core Module

4.5.7.1 Purpose

This class emulates the core router in the Diffserv architecture; thus, is intended to work downstream from an edge router. It forwards packets according to the marking done on them by the edge router. Packets having code points signifying low priority are dropped at a considerably higher rate than packets marked with code points of high priority. The core module is contained in the files "core.h" and "core.cc."

4.5.7.2 Class hierarchy positioning

This class inherits its behaviour from dsREDQueue class, therefore it is positioned below dsREDQueue class.

After having a grasp of the basic nitty-gritty of the NS simulator, we can move on to work implemented here in this dissertation.

CHAPTER 5
SIMULATION STUDY

This simulation simulates the DiffServ traffic to measure and analyze the quality of service. In this simulation, traffic is created using the Tcl scripts. These Tcl scripts are run under the NS simulator, which is an event driven program, being supported by various operating systems.

The set –up of the simulation can be described as below-
First the network topology is created to include the source nodes, destination nodes, DiffServ edge routers, and core routers with their ids. Next queues are created with specified traffic rate, delay time, packet size, bandwidth etc. among other inputs. In this simulation there is a CBR (Constant Bit Rate) connection between every source and destination node. Token Bucket and TSW3CM policers are used for policing the traffic. Now parameters are set for every queue including the number of physical and virtual queues, policer, scheduler, etc. Virtual queues are configured independently also with PHB entry added for each virtual queue. After configuring the queues, traffic agents are attached to the source nodes and connected with the destination nodes to carry the traffic. Finally the NS interpreter is called to run the entire set-up. Nam animator is used for the visual analysis of the traffic. The resulted data are printed using the appropriate constants.

Applying the various queuing techniques, it is observed that RED queuing is as much good as DropTail queuing. For scheduling the queues, WRR (Weighted Round Robin) is more efficient than PQ (Priority Queuing). WIRR (Weighted Interleaved Round Robin) algorithm is equally good with WRR. Policer's behavior is variable, sometimes Token Bucket is better and sometimes TSW3CM (Time Sliding Window with 3 Color Mode). The behavior of a policer is also dependent on a scheduling algorithm. The network topology after the simulation is shown below-

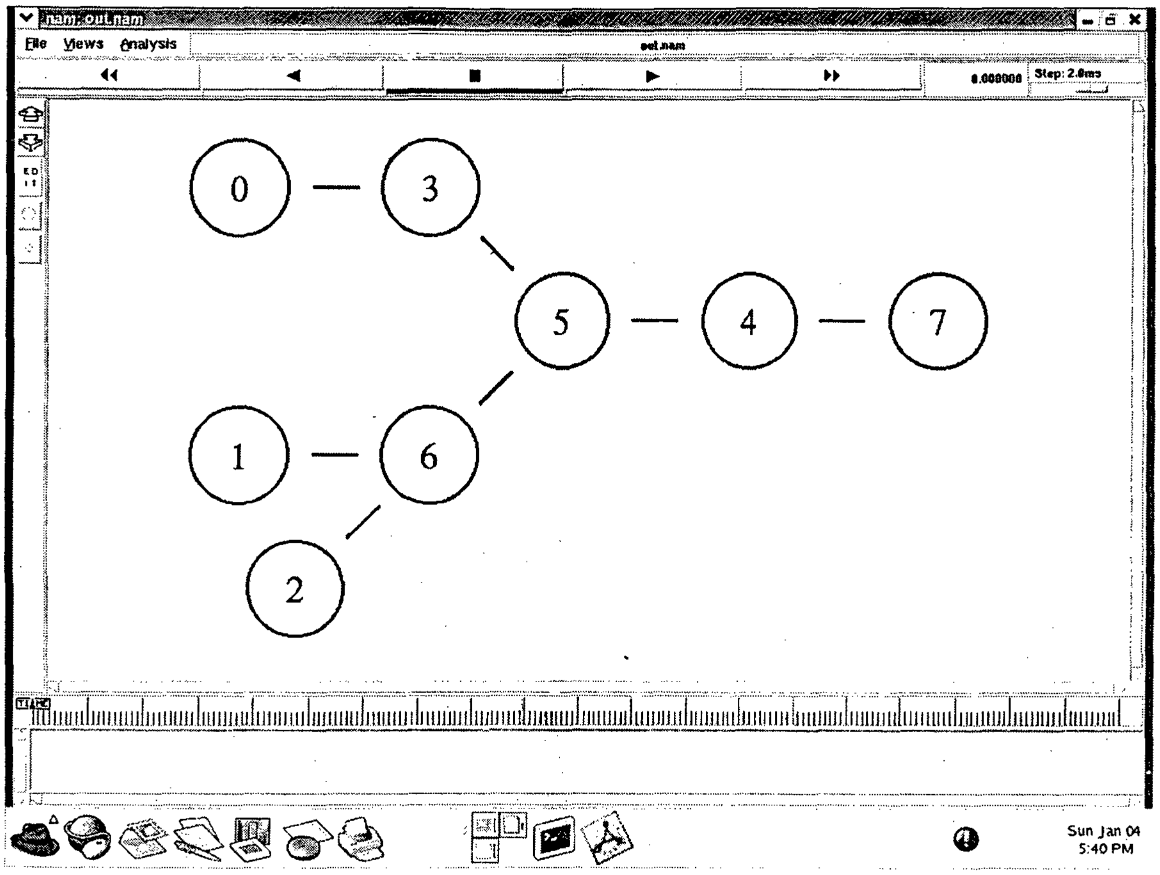


Figure 7. Network topology used in the simulation

Here nodes 0, 1, 2 are the source nodes, nodes 3, 4 and 6 are the edge router nodes, node 5 is the core node and node 7 is the destination node. Testing time for the simulation is chosen to be 20.0-second units. The traffic from the source node 1 is stopped during the period 5.0 to 10.0 and restarted at 10.0 time units. The behavior of the various queuing and scheduling algorithms can be seen in the Index of result tables. Though results here show a few algorithms being used, others can also be used.

CONCLUSION AND FUTURE WORK

This dissertation work shows the effect of various queuing, scheduling and dropping methods on the QoS of the Internet. Different algorithms for different queue sizes, traffic patterns and traffic rates, bandwidth were used to measure and analyze the end-to-end delay, number of packets successfully transmitted and also the packets dropped during the delivery. For simulation analysis, Nam network animator was used. It showed the visual description of the network topology, flow of the traffic, dropping of the packets, etc. The environment used was Linux operating system.

Simulation results showed that delay increases if the traffic is overloaded i.e. there are more flows between source and destination nodes. Less packets are dropped when traffic is within the service parameters limit, more packets are dropped if the traffic rate is increased. Bandwidth allocation to a link is also important. A link can not handle packets with the more traffic rate than allowed. More packets are dropped midway if the bandwidth of the link is less than the traffic rate.

Finally the simulation shows that it is possible to implement DiffServ architecture to provide QoS over Internet.

This simulation study can be extended to the real network to study the other hidden factors such as noise, power etc. which can influence the QoS of the network before the implementation in the public use.

BIBLIOGRAPHY

1. www.qosforum.com, "White Paper- QoS protocols & architectures".
2. QoS forum,"Quality of Service: Glossary of Terms", May 1999.
3. IETF "Differentiated Services" working group. See http://www.ietf.org/html.charters/diffserv-charter.html and http://www.ietf.org/ids.by.wg/diffserv.html.
4. J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, "Assured Forwarding PHB Group", RFC 2597, June 1999.
5. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, "An Architecture for Differentiated Services", RFC 2475, December 1998.
6. K. Nichols, S. Blake, F. Baker, D. Black, "Definition of the Differentiated Services Field (DS Field) in the Ipv4 and Ipv6 Headers", RFC 2474, December 1998.
7. Information on Class Based Queuing (CBQ) at http://www.aciri.org/floyd/cbq.html
8. Sasu Tarkoma, Informal Management Model for DiffServ Routers, October 2000.
9. Marko Luoma and Mika Ilvesmaki, Measurement based raffic classification in Differentiated Services.
10. J. Wroclawski, "The use of RSVP with IETF Integrated Services", RFC 2210, September 1997.
11. Mikko Kolehmainen, "DiffServ Per Hop Behaviors (PHBs)", October 2000.
12. X. Xiao & M. Ni. Lionel, Internet QoS: A Big Picture", IEEE Network Magazine, March 1999.
13. William Stallings, "High Speed Networks and Internets: Performance and quality of service ". Second Edition 2002.
14. Douglas E. Komar, "Computer Networks and Internets".
15. Andrew S. tanenbaum, "Computer Networks", 2000.
16. NS manual and tutorial.

APPENDIX

Tables of Results

Output 1: DropTail (Queuing) and WRR (Scheduling) with Token Bucket policer.

Policy Table1:

Flow (0 to 7): Token Bucket policer, initial code point 10, CIR 1000000.0 bps, CBS 10000.0 bytes.

Policer Table:

Token Bucket policer code point 10 is policed to code point 11.

Policy Table2:

Flow (1 to 7): Token Bucket policer, initial code point 10, CIR 1000000.0 bps, CBS 10000.0 bytes.

Flow (2 to 7): Token Bucket policer, initial code point 20, CIR 1000000.0 bps, CBS 10000.0 bytes.

Policer Table:

Token Bucket policer code point 10 is policed to code point 11.
Token Bucket policer code point 20 is policed to code point 21.

Packets Statistics at time 4.0 units

```
===========================
 CP  TotPkts  TxPkts  ldrops  edrops
 --  ---------  --------  --------  --------
All   5984    5003     563     418
 10   1016    1016      0       0
 11   2974    1993     563     418
 20    508     508      0       0
 21   1486    1486      0       0
```

Packets Statistics at time 6.0 units

```
===========================
 CP  TotPkts  TxPkts  ldrops  edrops
 --  --------  --------  -------  --------
All   8490    7247     696     547
 10   1392    1392      0       0
 11   4104    2861     696     547
 20    758     758      0       0
 21   2236    2236      0       0
```

```
Packets Statistics at time 10.0 units

==========================

 CP  TotPkts  TxPkts  ldrops  edrops
 --  -------  ------  ------  ------
All  12490    11247    696     547
 10   1892     1892     0       0
 11   5604     4361    696     547
 20   1258     1258     0       0
 21   3736     3736     0       0

Packets Statistics at time 15.0 units

==========================

 CP  TotPkts  TxPkts  ldrops  edrops
 --  -------  ------  ------  ------
All  19985    17512   1431    1042
 10   3150     3127    23      0
 11   9341     6891   1408    1042
 20   1883     1883     0       0
 21   5611     5611     0       0
```

Output 2: RED (Queuing)and WRR (Scheduling)with Token Bucket policer.

Policy Table1:

Flow (0 to 7): Token Bucket policer, initial code point 10, CIR 1000000.0 bps, CBS 10000.0 bytes.

Policer Table:

Token Bucket policer code point 10 is policed to code point 11.

Policy Table2:

Flow (1 to 7): Token Bucket policer, initial code point 10, CIR 1000000.0 bps, CBS 10000.0 bytes.

Flow 2 to 7: Token Bucket policer, initial code point 20, CIR 1000000.0 bps, CBS 10000.0 bytes.

Policer Table:

Token Bucket policer code point 10 is policed to code point 11.
Token Bucket policer code point 20 is policed to code point 21.

Packets Statistics at time 4.0 units

```
==========================
 CP  TotPkts  TxPkts  ldrops  edrops
 --  ---------  --------  --------  --------
All    5984    5003     563     418
 10    1016    1016       0       0
 11    2974    1993     563     418
 20     508     508       0       0
 21    1486    1486       0       0
```

```
Packets Statistics at time 6.0 units

=========================

 CP  TotPkts  TxPkts  ldrops  edrops
 --  -------  ------  ------  ------
All    8490    7247    696     547
 10    1392    1392      0       0
 11    4104    2861    696     547
 20     758     758      0       0
 21    2236    2236      0       0

Packets Statistics at time 10.0 units

=========================

 CP  TotPkts  TxPkts  ldrops  edrops
 --  -------  ------  ------  ------
All   12490   11247    696     547
 10    1892    1892      0       0
 11    5604    4361    696     547
 20    1258    1258      0       0
 21    3736    3736      0       0

Packets Statistics at time 15.0 units

=========================

 CP  TotPkts  TxPkts  ldrops  edrops
 --  -------  ------  ------  ------
All   19985   17512   1431    1042
 10    3150    3127     23       0
 11    9341    6891   1408    1042
```

20	1883	1883	0	0
21	5611	5611	0	0

Output 3: RED (Queuing)and PRI (Priority Queuing for scheduling) with Token Bucket policer.

Policy Table1:

Flow (0 to 7): Token Bucket policer, initial code point 10, CIR 1000000.0 bps, CBS 10000.0 bytes.

Policer Table:

Token Bucket policer code point 10 is policed to code point 11.

Policy Table 2:

Flow (1 to 7): Token Bucket policer, initial code point 10, CIR 1000000.0 bps, CBS 10000.0 bytes.

Flow (2 to 7): Token Bucket policer, initial code point 20, CIR 1000000.0 bps, CBS 10000.0 bytes.

Policer Table:

Token Bucket policer code point 10 is policed to code point 11.
Token Bucket policer code point 20 is policed to code point 21.

```
Packets Statistics at time 4.0 units

===========================

 CP  TotPkts   TxPkts  ldrops  edrops
 --  --------- -------- -------- --------
All   5984     5001     891     92
 10   1016     1016       0      0
 11   2974     2974       0      0
 20    508      428      80      0
 21   1486      583     811     92

Packets Statistics at time 6.0 units

===========================

 CP  TotPkts   TxPkts  ldrops  edrops
 --  --------- -------- ------- --------
All   8490     7243    1119     128
 10   1392     1392       0      0
 11   4104     4104       0      0
 20    758      678      80      0
 21   2236     1069    1039     128

Packets Statistics at time 10.0 units

===========================

 CP  TotPkts   TxPkts  ldrops  edrops
 --  --------- -------- -------- -------
All  12490    11243    1119     128
 10   1892     1892       0      0
 11   5604     5604       0      0
```

```
20    1258    1178     80      0
21    3736    2569    1039    128
```

Packets Statistics at time 15.0 units

```
============================
CP  TotPkts  TxPkts  ldrops  edrops
--  ---------  --------  --------  --------
All   19985   17511    2242    232
10    3150    3150      0     0
11    9341    9341      0     0
20    1883    1803     80     0
21    5611    3217    2162    232
```

Output 4: RED (Queuing)and WIRR (Scheduling) with Token Bucket policer

Policy Table1:

Flow (0 to 7): Token Bucket policer, initial code point 10, CIR 1000000.0 bps, CBS 10000.0 bytes.

Policer Table:

Token Bucket policer code point 10 is policed to code point 11.

Policy Table2:

Flow (1 to 7): Token Bucket policer, initial code point 10, CIR 1000000.0 bps, CBS 10000.0 bytes.

Flow (2 to 7): Token Bucket policer, initial code point 20, CIR 1000000.0 bps, CBS 10000.0 bytes.

Policer Table:

Token Bucket policer code point 10 is policed to code point 11.
Token Bucket policer code point 20 is policed to code point 21.

Packets Statistics at time 4.0 units

```
==========================
 CP  TotPkts  TxPkts  ldrops  edrops
 --  ---------  --------  -------  --------
All   5984   5003   554   427
 10   1016    997    19     0
 11   2974   2012   535   427
 20    508    508     0    0
 21   1486   1486     0     0
```

Packets Statistics at time 6.0 units

```
==========================
 CP  TotPkts  TxPkts  ldrops  edrops
 --  ---------  --------  --------  --------
All   8490   7244   694   552
 10   1392   1373    19     0
 11   4104   2877   675   552
 20    758    758     0    0
 21   2236   2236     0     0
```

```
Packets Statistics at time 10.0 units

==========================
 CP  TotPkts  TxPkts  ldrops  edrops
 --  ---------  --------  --------  --------
All   12490   11244    694    552
10    1892    1873    19     0
11    5604    4377   675    552
20    1258    1258     0     0
21    3736    3736     0     0

Packets Statistics at time 15.0 units

==========================
 CP  TotPkts  TxPkts  ldrops  edrops
 --  ---------  --------  --------  --------
All   19985   17508   1418   1059
10    3150    3131    19     0
11    9341    6883   1399   1059
20    1883    1883     0     0
21    5611    5611     0     0
```

Output 5: RED (Queuing) and WRR (Scheduling) with TSW3CM policer

Policy Table1:

Flow (0 to 7): TSW3CM policer, initial code point 10, CIR 1000000.0 bps, PIR 500000.0 bytes.

Policer Table:

TSW3CM policer code point 10 is policed to yellow code point 11 and red code point 12.

Policy Table2:

Flow (1 to 7): TSW3CM policer, initial code point 10, CIR 1000000.0 bps, PIR 1000000.0 bytes.

Flow (2 to 7): TSW3CM policer, initial code point 20, CIR 1000000.0 bps, PIR 1000000.0 bytes.

Policer Table:

TSW3CM policer code point 10 is policed to yellow code point 11 and red code point 12.
TSW3CM policer code point 20 is policed to yellow code point 21 and red code point 22.

Packets Statistics at time 4.0 units

```
==========================
CP  TotPkts  TxPkts  ldrops  edrops
--  -------- -------- -------- --------
All   5984    5027    847     110
10    1019     941     78       0
12    2971    2383    513      75
20     671     671      0    0
```

```
22   1323   1032   256   35

Packets Statistics at time 6.0 units

==========================

CP  TotPkts  TxPkts  ldrops  edrops
--  ---------  --------  --------  --------
All   8490   7274   1074   142
10   1285   1207    78    0
12   4211   3444   672   95
20    937   937    0    0
22   2057   1686   324   47

Packets Statistics at time 10.0 units

==========================

CP  TotPkts  TxPkts  ldrops  edrops
--  ---------  --------  --------  --------
All  12490  11274   1074   142
10   1506   1428    78    0
12   5990   5223   672   95
20   1435   1435    0    0
22   3559   3188   324   47
```

Packets Statistics at time 15.0 units

```
=============================
 CP  TotPkts  TxPkts  ldrops  edrops
 --  ---------  --------  --------  --------
All   19985   17562   2145    278
 10    2617    2520     97      0
 12    9874    8280   1404    190
 20    2073    2073      0      0
 22    5421    4689    644     88
```