ABSTRACT

Senapati, Mukul Madan. A Simulation Study of Cross Traffic on Expedited Forwarding in Differentiated Services Networks. (Under the direction of Dr. Mladen A. Vouk)

The purpose of this research was to simulate Differentiated Services enabled network topologies in order to study cross traffic impact on the Expedited Forwarding (EF) Per Hop Behaviors (PHBs). The results help us understand the extent to which packets tagged for Expedited Forwarding can be protected, and the guarantees that can be given to EF flows. Cross traffic and EF can exist and interact in many ways. This study analyzed two types of cross-traffic. In the first case a single EF micro-flow was considered, while all other EF micro-flows as well as any best effort traffic, was considered to be cross traffic that could affect this EF micro-flow. This case was analyzed under two conditions: a) a single EF micro-flow consisting of packets of small size, and b) a single EF micro-flow consisting of large packets. In the second case, the EF aggregate as a whole was considered composed of EF micro-flows of small as well as large packets, and any best effort traffic was considered cross-traffic that could affect the EF aggregate as a whole. Two schedulers, namely the Priority Queue Scheduler and the Weighted Round Robin Scheduler, were used to simulate the EF PHB and a comparison was made to determine which could provide better Quality of Service guarantees. Possible metrics for quantifying Quality of Service provided to the EF PHB were also investigated. Further, the bounds on the recent redefinition of the EF PHB by IETF were also studied. A variant of the EF PHB, called the Delay Bound (DB) PHB was defined by IETF. The DB PHB was studied, and recommended values for parameters that characterize the DB PHB were presented. Finally end-to-end Quality of Service obtained from the EF Per Domain Behavior was studied across multiple autonomous Differentiated Services domains governed by common Service Level Agreements.

A Simulation Study of Cross Traffic on Expedited Forwarding in Differentiated Services Networks

By
Senapati Mukul Madan

A thesis submitted to the Graduate Faculty of
North Carolina State University
In partial fulfillment of the
requirements for the degree of
Master of Science
Computer Network Engineering

Raleigh, North Carolina, USA
2002
Approved by:

Dr. Mladen A Vouk
Chair of Advisory Committee

Dr. Mihail Sichitiu
Co-Chair of Advisory Committee

Dr. Peng Ning
Member of Advisory Committee

Dedication

I will like to dedicate this work to my parents, sisters and grand parents who have all been a great source of inspiration and constant encouragement to me. They have all played a significant role in shaping me as a person, and I truly believe that it is because of them that I am what I am in life today.

Biography

Senapati Mukul Madan was born in Kolhapur, India in the state of Maharashtra. He was brought up in the city of Pune in India. He did his schooling from Loyola High School and Junior College in Pune. He then pursued his Bachelors in Engineering Degree with a specialization in Electronics Engineering from Vishwakarma Institute of Technology, a college affiliated to the Pune University and renowned for its program in Electronics Engineering. He worked in Pune with a multinational Information Technology company by the name Tata Technologies Private Limited for a year in the capacity of a Software Consultant. He then joined the North Carolina State University to pursue his Masters degree in Computer Networking, and has been working under the guidance of Dr. Mladen Vouk of the Computer Science Department in the capacity of a Research Assistant for the past 20months in the field of simulation of Differentiated Services networks.

Acknowledgements

I will like to thank Dr. Mladen A. Vouk, for his guidance and support. It was an honor and a privilege to work with him. I also want to thank him for giving me a chance to work on the simulation projects from Alcatel and BellSouth.

I also want to thank Dr. Mihail Sichitiu, for helping me with reviews and feedback while preparing the thesis draft. Dr. Sichitiu has been a very friendly and approachable person, and has been a great source of help in the final stages of my thesis.

I also want to thank Dr. Peng Ning for his suggestions and feedback.

I will like to give special thanks to Mr. Marhn Fullmer and Mr. Mark Doliner, who have always been there to help out whenever I needed guidance regarding operating systems, packages and networks. They have provided a lot of valuable advice in times of need.

Finally I want to thank my colleagues Ms. Uma Jayaraman, and Mr. Puneet Bhatia, who worked with me on numerous experiments, and who were always supportive to my ideas. We made a great team together, and it was a pleasure working with both of them.

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1. Motivation and Goals

The Internet was primarily designed to support best effort (BE) traffic. Unfortunately, it is not anymore constrained to just data traffic. The earlier specifications on which the Internet was built no longer apply. The growth of the Internet has fuelled the growth of data intensive media transmission, such as real time voice and video transmission. A number of new applications being developed for the Internet require stringent bounds on parameters such as the end-to-end delays and inter-packet jitter. In order to provide these bounds, there is a need for special Quality of Service (QoS) mechanisms. The Internet Protocol (IP), version 4, (Ipv4) by itself does not provide for a lot of flexibility as far as Quality of Service is concerned. This led to the development of alternate mechanisms based on policies that could be deployed on the existing network infrastructure with minimal changes, and still provide a suitable Quality of Service.

Differentiated Services (DS) is one such technology [1][2]. Aside from plain over-provisioning, DS is probably the most promising technology that can be implemented with relatively small disturbance to the existing infrastructure. Service Providers charge extra for services developed using service differentiation. One such service, Expedited Forwarding (EF) Per Hop Behavior (PHB) is of special interest since it is intended to have the most stringent QoS guarantees [4][7][8], and hence is most suitable for support of QoS sensitive applications. This thesis provides an analysis of some of the issues that arise in the EF PHB implementation, as well as in the application of that solution across a networking domain, i.e., it's Per Domain Behavior (PDB). This is deemed important as the premium EF service can be provided to customers only if the EF service has been optimally configured within the DS.

1.2. Open Issues

Some of the general open issues specific to Expedited Forwarding in Differentiated Services enabled networks are:

a) Detailed characterization of the EF PHB based on different scheduling mechanisms and policers [2][4][7][8][12].
b) Benchmarking of various DS mechanisms and provision of a guideline for implementers [1][2].
c) Definition of appropriate values of QoS metrics that could be used by service providers while defining Service Level Agreements[1] for commercial use [2].
d) Study of alternative QoS metrics for quantification of the QoS provided to the EF aggregate [12]. Study of the merit factors that characterize EF compliant DS nodes [7][8][9][12].
e) Study of the impact of buffer sizes as applied to the EF PDB where the amount of BE cross-traffic[2] as well as other EF traffic in the core network, may violate the EF PHB [2][7][8][13].

1.3. Specific Issues Addressed in this Thesis

Characterization of the EF PHB based on the QoS metrics, such as minimum, maximum and average per hop packet delays, delay jitter, and inter-packet jitter experienced by packets within the EF aggregate are of special interest when it comes to jitter sensitive applications such as video and voice-over IP. Interactive video has the most stringent QoS requirement, with a data loss ratio requirement of $1x10^{-9}$ packets and a delay requirement of 500μs per switching node. Voice applications are more tolerant to loss, but less tolerant to delay with a loss ratio requirement of $1x10^{-6}$ and a delay requirement of 500μs per switching node [28]. Studies have also shown voice to be tolerant to end-to-end delays of as large as 150ms without causing any significant degradation in conversational dynamics [29]. Since these applications are of special interest to the end user community, the study of parameters governing the QoS provided by the EF service has been the main motivation behind this thesis. Ideally the EF PHB was defined such that EF traffic would spend very less time in queues, awaiting service within DS nodes. This in effect would provide a virtual leased line

[1] , Service Level Agreements are explained in Section 2.1. of this thesis.
[2], Cross-traffic is any traffic attempting to compete on a networking device with the stream or traffic aggregate under consideration.

service to the traffic flow [4]. This thesis aims at analyzing how various factors influence the queuing delay encountered by EF traffic within DS nodes.

Significance of measured parameters in the analysis: The minimum and maximum per hop delays provide the fixed delay and the total delay[3] that a packet would encounter in a DS node. The delay jitter, which is the difference between the maximum and minimum delay, provides the variable delay a packet encounters in the DS node. This variable delay can be equated to the queuing delay experienced by packets within a DS node, which we are trying to minimize in DS. The extent to which we can reduce the queuing delay would determine the efficiency of a DS implementation. The inter-packet jitter provides an estimate to design the de-jitter buffers at the end destinations for streaming media applications.

Simulation techniques were used for running the experiments in this thesis, as several configuration parameters within the DS domains could be varied with considerable ease in order to run multiple scenarios of the same experiment. The simulation environment also permitted the study of topologies, especially for the EF PDB, which would not have been possible to set up due to limited resources. Details of the simulation environment are explained in Chapter 4. of this thesis.

Assumptions made in the Experiments: Several assumptions were made while designing the experiments in this thesis. All the DS nodes used in the experiments used the default implementation of a DS node within the simulator. Specific details regarding the internal architecture, the number of stages etc, were not accounted for in the simulations, as these would be vendor specific. All the DS nodes were assumed to be identical. This was a limitation in the analysis, as practically, DS nodes from various vendors with differing capabilities, will be employed in the networks. For the sake of simplicity, all the links were assumed to be Fast Ethernet links, while practically, multiple data link technologies are employed in the Internet. This was another limitation of this analysis. In practical networks, there is a possibility that packet loss occurs due to packet corruption during packet

[3] , Total delay is the sum of fixed and variable delays.

forwarding and transmission, but this was not accounted for in the experiments as the probability of packet loss due to data corruption in today's high-speed networks was considered to be very small, with a bit error rate of the order of 10^{-11} for Fast Ethernet [30]. The simulation time was restricted to 10seconds, as a single simulation run for this time duration could generate trace files of sizes up to 2Gb depending upon the number of packets transmitted in simulation time. Processing these trace files for relevant details was a time consuming task. Running the simulations any longer would result in even larger trace files. Running the simulations or observing similar practical implementations over a longer duration can result in an increase in the monitored parameters. Further the packets within the EF queue was assumed to be served in first in first out (FIFO) order. Practical implementations may use complex non-FIFO scheduling mechanisms within the EF queue depending upon the specific services and applications running on the DS networks.

The effects on individual micro-flows along with the EF aggregate comprising of these micro-flows were to be determined. As each of them would receive different QoS due to the presence of other EF micro-flows having different traffic characteristics and competing for the same resources, the QoS cannot be generalized. Hence analysis was made for:

a) A single EF micro-flow consisting of small packets within the EF aggregate.
b) A single EF micro-flow consisting of large packets within the EF aggregate.
c) The EF aggregate as a whole.

Determination of the small and large packet sizes is explained in Section 5.3. of this thesis.

A Comparative study of a Priority Queue scheduler and a Weighted Round Robin scheduler was done, when used with a Token Bucket policer to implement the EF PHB in a DS node, in order to provide norms for benchmarking other schedulers. This study was made for all the 3 cases mentioned above and consists of the following:

a) Effect of variation in the number of EF micro-flows and hence the net EF arrival rate for an output interface on the QoS metrics, to study effects of EF micro-flow aggregation and packet clustering.
b) Study of variation in the 95 and 99 percentile per hop packet delays and the effectiveness of using the standard deviation in per hop packet delay and a computed

threshold as a metric to quantify these delays, in order to provide a measure of confidence in providing the EF PHB.

c) Study of the effect of varying the net EF departure rate on a specific outgoing interface, when the net EF arrival rate was maintained constant, in order to determine the extent of over provisioning of resources, necessary to support EF traffic.
d) Study of the effect of variation in best effort cross-traffic in the network on the EF PHB to determine how traffic mapping to other PHBs, affect the EF PHB.
e) Study of the merit factors E_a and E_p when the EF configured rate for an output interface was varied [7], to study the range of possible values as compared to the delay jitter.
f) Study of the possible values of score S for the Delay Bound PHB [9] when the net arrival rate of DB traffic for an output interface was varied, to study the range of possible values.

A Case study of the EF PDB for studying the impact of cross-traffic on the EF traffic traversing consecutive DS domains, governed by common Service Level Agreements, thus creating a DS region was also done.

1.4. Thesis Layout

Chapter 2 defines terms used in this thesis and discusses the existing models for service differentiation. It provides a brief overview of the Differentiated Services Architecture. Chapter 3 discusses the EF PHB in detail. It also describes terms used in the case study of the EF PDB. Chapter 4 explains the tools and methods used to perform this study and it provides a brief overview of the Differentiated Services module in the network simulator, ns-2. Chapter 5 presents the results of the various simulation experiments. Chapter 6 presents the summary and conclusions. Chapter 7 contains the references. Appendix A contains the histograms used in the analysis of Experiment 2 on the EF PHB as a percentile analysis has been done for this experiment. Appendix B contains sample TCL scripts used to run the experiments on the network simulator ns-2, and the AWK scripts used to parse the trace files output by each simulation run to collect relevant data for the experiments.

2. Differentiated Services Architecture

2.1. Definitions used in this Thesis [1][2]

Some of the terms used in the Differentiated Services Architecture are defined below:

Behavior Aggregate (BA)	Collection of packets having the same DS codepoint and crossing a link in one direction.
Differentiated Services Domain (DS Domain)	A DS domain is defined as a contiguous portion of the Internet consisting of edge and core DS routers that are governed by a consistent set of policies and administered in a coordinated manner [1]. Multiple autonomous networks can be aggregated to form a single DS domain provided they are administered by a single entity.
Differentiated Services Region (DS Region)	Set of contiguous DS domains offering service differentiation over paths through those domains [2].
Micro-Flow	A single instance of an application-to-application flow of packets identified by their source and destination address and port numbers [1].
Per Hop Behavior (PHB)	A description of the externally observable forwarding treatment applied at the Differentiated Services-compliant node to a Behavior Aggregate.
Service Level Agreement (SLA)	A service contract between either, a customer and a service provider or between two service providers specifying the forwarding service a customer should receive.
Services	The overall forwarding treatment given to a particular traffic flow in one direction across a DS domain in accordance with a service level agreement.
Traffic Conditioning	Defined as the control functions such as marking, monitoring, policing and shaping applied to a Behavior Aggregate in order to make it compliant with the Service Level Agreement.

2.2. Models for Service Differentiation

Existing models of service differentiation can be classified into the following categories: Relative Priority Marking, Service Marking, Label Switching, Integrated Services / Resource Reservation Protocols, and Static Per-hop Classification [2]. The following sub-sections briefly describe these models.

2.2.1. Relative Priority Marking Model

The Ipv4 "precedence field"[4] is an example of a relative priority marking model implementation where the application, host, or proxy node selects a relative priority or precedence for a packet [2][31][38]. The network nodes along the path between the source and the destination examine the priority value within the packet's IP header and apply the appropriate priority forwarding behavior.

The IP precedence field consists of bits 0-2, and it is a subfield of the IPv4 Type of Service (TOS) octet. The values that the three-bit IP precedence field might take, are assigned to various uses, including network control traffic, routing traffic, and various levels of privilege. The least level of privilege is considered to be routine best effort traffic. Some routers exist that use the IP precedence field to select different per-hop forwarding treatments [1][33][34]. Although early BBN IMP packet switches dating back to the late 1960's implemented the precedence feature, early commercial routers and UNIX IP forwarding code generally did not [1]. As networks became more complex and customer requirements grew, commercial router vendors developed ways to implement various kinds of queuing services, including priority queuing, to handle the policies encoded in the precedence bits.

Routers using the priority model examine the IP addresses, the IP protocol numbers, the Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) port numbers, and possibly some other header fields.

However, the specifications of the packet forwarding treatments selected by the IP precedence field are not specific enough for predictable differentiated services. A scalable architecture for service differentiation must specify the packet forwarding treatments in much more depth.

[4] , Details of the Ipv4 Precedence field and the TOS field can be found in RFC 791 on the IETF website at http://www.ietf.org/rfc/rfc0791.txt

2.2.2. Service Marking Model

The IPv4 Type of Service (TOS) bits (bits 0-3) within the TOS octet are an example of a somewhat more detailed service marking model. Each packet is marked for a type of service, which could be minimize delay, maximize throughput, maximize reliability and minimize monetary cost [31][38]. Network nodes may select routing paths or forwarding behaviors, which are suitably engineered to satisfy these service requests [2].

There are two problems with this model:

- Since only 4 of the 8 bits in the TOS octet of the Ipv4 header are used as the significant TOS bits, only 16 different services can be provided.
- The model is applied on a per packet basis, and hence services applying to a sequence of packets cannot be applied here.

2.2.3. Label Switching Model

Multi Protocol Label Switching (MPLS) is an example of the label-switching model [25]. This model is state based and maintains state of traffic streams on all the nodes in the traffic path. Packets are classified into flows. At the ingress of such a network, each flow is associated with a Label Switched Path (LSP), and packets using this LSP are tagged with a forwarding label that is used for determining the next hop and the QoS to be delivered to this packet [26]. Technically, extremely fine granularity of resource allocation can be given to the flows as labels have only local significance. Traffic engineering and management can also be achieved by routing traffic via specific paths where multiple paths exist between two nodes within the network [24].

The main overhead of this technology is that it is stateful, and requires additional management and configuration to maintain the label switched paths. Also, as the network grows, the number of states maintained in a node approaches a value that is the square of the edge nodes when edge-to-edge label switched paths with provisioned resources are employed [2].

2.2.4. Integrated Services / Resource Reservation Protocol Model

The Integrated Services (Intserv)/Resource Reservation Protocol (RSVP) model uses path establishment, maintenance and teardown signaling mechanisms to control data flows directly at the path nodes. Packet classification and forwarding state is retained on each node along the data path. The Intserv / RSVP model relies upon traditional datagram forwarding in the default case, but allows sources and receivers to exchange signaling messages which establish additional packet classification and forwarding states on each node along the path between them [17]. This technique works with classical IP solutions, but does require extra code on the routing nodes.

A drawback is the absence of state aggregation. The number of states on each node scales in proportion to the number of concurrent reservations. The latter can grow to a very large number in large networks. Another drawback is the need for change in the application programming interfaces in order to support the RSVP signaling protocol.

The differentiated services mechanisms could be utilized to aggregate Intserv / RSVP state in the core of the network [35]. However, it is claimed that although the support of large number of classifier rules and forwarding policies may be computationally feasible, the management burden associated with installing and maintaining these rules on each node within a backbone network that will be traversed by a traffic stream is substantial [2].

2.2.5. Static Per Hop Classification Model

In this model, static classification and forwarding policies are implemented at each hop (router) along the network. These policies are not updated based on the active state of the network but on a periodic basis.

The problem with this model is that it doesn’t scale well because the policies, in contrast to Intserv/RSVP model, are statically maintained. Management of static policies in large networks is a big problem. Another problem is the change of routes in case of link or node failures within the network. Such a change is not automatic. It depends on scheduled periodic

updates, and as these periods can be long, static networks may result in substantial down time.

2.3. The Differentiated Services Architecture Model

Differentiated Services Architecture (DSA) model tries to address most of the problems brought up in the previous section. Traffic entering a Differentiated Services network is classified and conditioned at the DS ingress node according to pre-defined Service Level Agreements and accordingly assigned to a behavior aggregate. A codepoint uniquely identifies each behavior aggregate. This codepoint determines per hop forwarding treatment meted out to the packet in the core of the network. In other words, the DSA provides QoS by dividing traffic into different categories, marking each packet with a codepoint that indicates its category, and scheduling packets according to their codepoints.

This model has a number of advantages in comparison with the previously mentioned models. They include:

- The state of independent traffic flows need not be maintained within the network.
- No direct signaling mechanisms are required for setting up, maintaining and tearing down paths within the network, thus core nodes are not burdened, and no extra network bandwidth is spent on account of these mechanisms. However, policies still have to be defined on each node, and a mechanism needs to exist to do that.
- Applications can be unaware of this model and still use its services.
- A finer granularity of forwarding treatment can be given to packets than that offered by the Ipv4 precedence field or the Ipv4 type of service field, specifically one can have as many as 64 distinct forwarding treatments as 6 bits are used for the codepoint.
- QoS can be provided without changing the network infrastructure, except for the need to install DS capabilities on the network nodes.
- The implementation can be incremental and can coexist with non-differentiated services networks.
- This model scales well even in large networks [36][37].

2.3.1. The Differentiated Services Field Definition [1]

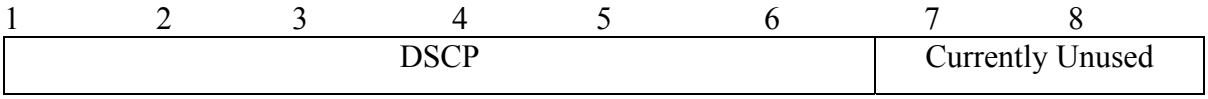

Figure 1: Definition of the Differentiated Services Codepoint in the Ipv4 TOS Field

Six bits of the Ipv4 TOS octet are used as a Differentiated Services Codepoint[5] (DSCP) to select the PHB a packet experiences at each DS node, resulting in 64 distinct codepoints. DS compliant nodes when determining the PHB to apply to a received packet ignore the value of the Currently Unused (CU) bits. The entire 6-bit DSCP field must be used by DS compliant nodes while mapping a packet belonging to a behavioral aggregate to a particular PHB.

2.3.2. Backward Compatibility with IP Precedence Field [1]

In order to maintain backward compatibility with the IP precedence field, a set of codepoints called Class Selector Codepoints mapping to PHBs have been defined that are compatible with the forwarding treatments selected by the IP Precedence Field. These Class Selector Codepoints may map to PHBs. The minimum requirements for these PHBs are called the Class Selector PHB requirements [1]. A specification of the packet forwarding treatments selected by the DSCP with a value of 'xxx000' and Currently Unused (CU) subfield unspecified, are reserved as a set of eight Class Selector Codepoints [1]. PHBs mapping to these codepoints must satisfy the Class Selector PHB requirements in addition to preserving the Default PHB requirement on codepoint '000000' [1]. As the Ipv4 Precedence marking is a relative priority model, a Class Selector Codepoint with a higher numerical value has a higher relative order than one with a lower numerical value. At least two independently forwarded classes of traffic must be provided by the set of PHBs mapped to by the eight Class Selector Codepoints. The Ipv4 Precedence field assigns values '110' and '111' for routing traffic. Hence a preferential forwarding treatment must be given by PHBs selected by codepoints '11x000' in comparison to the PHB selected by codepoint '000000'.

[5], Reference for codepoints can be found on the Internet Assigned Numbers Authority website, e.g., http://www.iana.org/assignments/dscp-registry

2.3.3. Some details

The Differentiated Services Architecture (DSA) is based on the simple concept of classifying packets at the network boundaries and marking these packets with appropriate codepoints based on the rules of the administrative policies, conditioning these packets in conformance with the policies, and giving these packets a forwarding treatment corresponding to their classification in one direction of propagation. Flow aggregation in the core is an important concept in DSA that aids in scaling this technology to large networks. Scalability is achieved by considering individual traffic micro-flows arriving at the edge DS nodes, and by aggregating them into well-defined traffic aggregates based on the codepoint to PHB mapping. These traffic aggregates are dealt with in the core of the DS Domain.

At the core, DS nodes are required to support mechanisms that classify, schedule and provide buffer management to packets for the appropriate forwarding treatment [1]. In addition, edge routers need to implement mechanisms to mark packets and condition them.

The DSA has two elements. The first concerns the mechanism of forwarding the packets. This includes the differential treatment given to a packet via scheduling and queuing mechanisms [1]. The forwarding path will also need to monitor packets for billing purposes and to ensure compliance of packets to the service agreements, and to police and shape the packets in the case they are not conforming to their service agreement. These functions are important to ensure that the service provider can actually provide the service promised to all customers. The second element is the setting of the node parameters that govern the packet forwarding process. This is an area where a lot of active research is being done.

Services are realized by the combination of traffic conditioning and marking at the ingress nodes of a DS network, along with the scheduling and queuing in the core DS nodes.

2.4. Operation of a Differentiated Services Network

The following is a very simplified picture of the entire architecture in operation. It is illustrated in Figure 2.

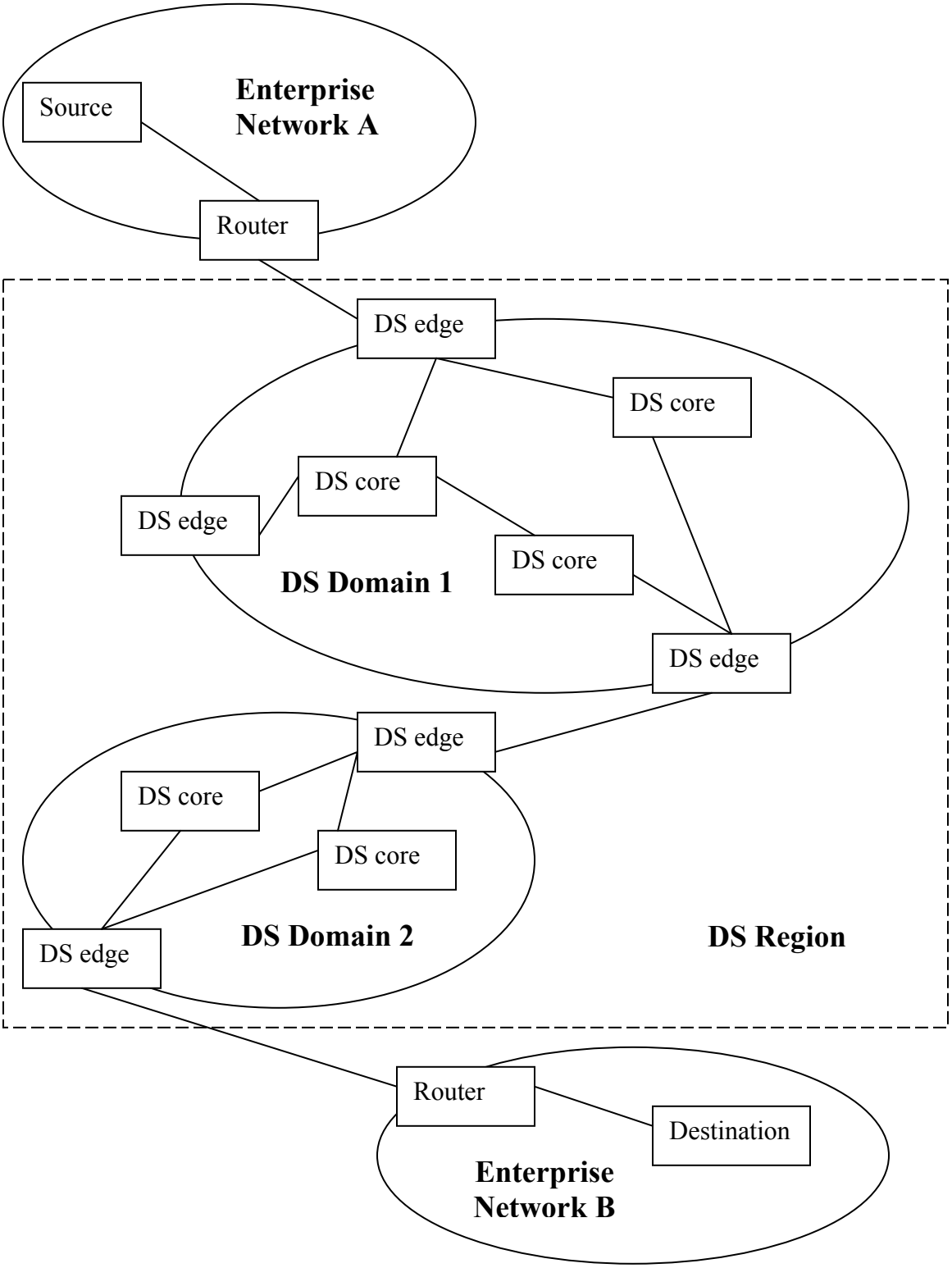

Figure 2: Sample Topology used to illustrate the working of the DSA.

Traffic originates as independent micro-flows at the source node within enterprise network A. This traffic is destined for a node within the enterprise network B. The boundary router within network A forwards the traffic to the ingress node, also called the edge node of DS domain 1. The edge node, depending upon the service level agreement existent between the service provider and the enterprise network A, classifies and marks the packet with a codepoint, conditions the traffic and assigns it to a per hop forwarding behavior. In the absence of a service level agreement all the traffic originating from Enterprise network A is considered best effort traffic and marked to the codepoint "000000", which maps to the default PHB. The default PHB is not meant to provide any form of QoS guarantees to the traffic. It just forwards packets when resources are available.

Once this is done, core DS nodes only forward the packets based on the DS codepoint assigned to packets. The core nodes forward the traffic to the egress DS node, which ultimately either delivers the packets directly to the router within the enterprise B network if it is directly connected to it, or it forwards the packets to the ingress router of another DS domain. In the latter case it is the ingress router of DS domain 2. This process continues till a DS domain can directly deliver the packets to the enterprise network B. In this case the traffic is sent from one DS domain to another DS domain. The downstream DS domain, which is DS domain 2 in our case, is free to assign a different codepoint to this traffic as long as it maps it to an equivalent PHB.

Any packet received at the downstream DS domain with a codepoint that doesn't map to any PHB in its PHB table is re-marked with the default codepoint of "000000" and treated as if it belonged to the default PHB.

In this manner the ingress nodes can receive traffic from multiple independent sources, independent enterprise networks, and upstream DS domains. They then classify the traffic and shape independent micro-flows into manageable aggregate flows. Once this is done, only aggregate flows are dealt with in the core of the DS domain. No state regarding the micro-

flows needs to be maintained within the core of the DS domain. Once the traffic reaches the edge of the DS domain, it can be de-aggregated and delivered to its respective locations.

In the Figure 2, the two consecutive DS domains form a DS region represented by the square dotted box surrounding them. This is because common traffic conditioning agreements are defined between these two DS domains to handle traffic through them.

2.5. Major Logical Components of a DS Node

A DS node can be considered to comprise of several DS elements. Depending upon whether the DS node is an edge node or a core node, the particular elements implemented can vary. The elements are:

- Packet Classifier
- Traffic Conditioner
- Buffer Manager
- Scheduler

2.5.1. Packet Classifier

The primary functions of the packet classifier are to differentiate between packets based on the contents of the packet header and send them to the traffic conditioner for further processing. Packet classifiers are an integral part of any DS node and hence must be implemented in the edge and core DS nodes. There are two types of packet classifiers [2]. One type of classifier called the BA classifier differentiates between packets solely based on the value of the DSCP. These types of classifiers are primarily implemented in the DS core nodes as they receive packets that are already marked with the DSCP by the edge nodes. The other type of classifier is the Multi-field Classifier. These classify packets based on multiple fields in the Ipv4 header. These fields mainly consist of the source and destination IP addresses and TCP or UDP port numbers and some other fields. These types of classifiers are primarily implemented in the edge DS nodes that receive unmarked traffic from non-DS domains or from neighboring DS domains.

2.5.2. Traffic Conditioner

A traffic conditioner on receiving traffic flows from the packet classifier further processes the traffic by passing it through several sub modules listed below.

- Meter
- Packet Marker
- Shaper
- Packet Dropper

The logical view of the Packet Classifier and Traffic Conditioner is shown in the Figure 3 below [2]:

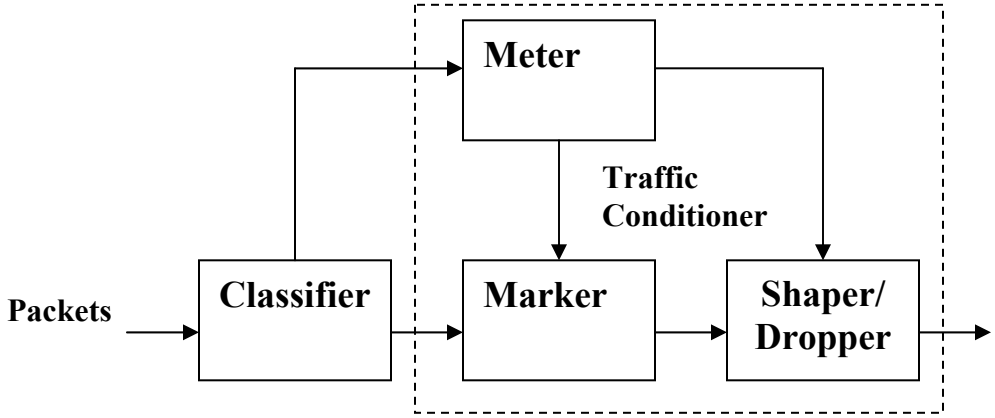

Figure 3: Block Diagram of the Classifier and the Traffic Conditioner within a DS Node.

2.5.2.1. Meter

The meter uses the traffic conditioning agreement as a reference and measures the temporal properties of a traffic flow to determine whether a packet is within the specified profile or misbehaving [1]. Based on this measurement, signals are passed to the other sub-modules to trigger specific actions on the packet.

2.5.2.2. Packet Marker

Based on the traffic streams received from the classifier, the packet marker module maps the packet to a particular PHB and sets the DS field within the Ipv4 header to a specific DSCP. On the basis of this codepoint, DS core nodes will give it the appropriate forwarding treatment by using the codepoint as an index to the codepoint to PHB mapping table. Packet marker is a required module in DS edge nodes. For any traffic originating within the DS domain and requiring service differentiation, some internal DS core nodes will need to have the functionality of a packet marker, as they will need to mark the DSCP within the Ipv4 packet header.

2.5.2.3. Shaper

The shaper modifies the inter-packet times within a traffic stream in order to make it compliant with the traffic profile. This is necessary in case the traffic stream has been backlogged in the immediate upstream DS node behind other traffic, and as a result appears as a burst of packets to the input interface of the DS edge node.

2.5.2.4. Dropper

The dropper is responsible for dropping packets belonging to a traffic stream in order to make the traffic stream compliant with the service profile. This process is defined as "policing" a stream [2]. The dropper and shaper modules are closely related and implemented many a times as a single module.

2.5.3. Buffer Manager

Packet switches require buffers in order to accommodate bursts of traffic. In order to reduce packet loss, it is necessary to buffer these packets until they can be served. The main functions of the buffer manager are to manage the packet queues for an output interface within the DS node [22]. Based on the SLAs in effect for this DS domain, the buffer manager selects the queue to send packets from. It also performs active queue management in order to

prevent congestion by means of mechanisms like Random Early Detection (RED) by keeping the average queue size small but still permitting small bursts of packets to be serviced [23].

2.5.4. Packet Scheduler

A scheduler is used in order to provide sharing of network resources and providing certain traffic flows with performance guarantees. A packet scheduler determines which packet queue within a DS node will be served next based on the PHBs and the services built on those PHBs within the DS domain. The scheduler implemented within a DS node principally determines the performance that a PHB receives. Each scheduler is designed to satisfy one or more of the following requirements: ease of implementation, ease and efficiency of admission control, fairness and protection, and performance bounds which aid in bounding the QoS metrics.

2.6. Per Hop Behavior [2]

The Per Hop Behavior (PHB) is the means by which a DS compliant node allocates resources to behavior aggregates. Based on this basic resource allocation on each DS node within a DS domain, useful services can be constructed. Traffic characteristics associated with a behavior aggregate are responsible for the behavior of a PHB. Other PHBs may also influence the treatment given to packets by the PHB under consideration. A PHB can be specified by means of the resources allocated to it compared to other PHBs, or by means of measurable QoS metrics like packet loss, packet delay etc [2].

Packet schedulers, policers and buffer management mechanisms form the basic building blocks for implementing PHBs. PHBs are always defined in terms of their behavior characteristics and not by means of their specific implementation, although possible implementations can be recommended. PHBs also need to be specified in a manner such that their inter relation with other PHBs within a DS node are clearly defined. A codepoint to PHB mapping table with an entry for the PHB under consideration is necessary in every DS node in order for the DS domain to support that PHB.

2.7. Per Domain Behavior [6]

Technically, the Per Domain Behavior[6] (PDB) is defined as a building block that outlines the relationship between classifiers, traffic conditioners, specific PHBs and particular configurations with a resulting set of specific observable attributes that may be characterized in several ways [6]. A PDB is used to describe the forwarding behavior experienced by a traffic aggregate, as it crosses a DS domain. PDBs are characterized by specific metrics quantifying the treatment a set of packets with the same DS codepoint will receive as they cross the DS domain. PDBs are meant to help Internet Service Providers construct services based on the DSA. A PDB can be thought to comprise of two aspects. The first part is the traffic conditioning at the boundary nodes to form the traffic aggregate, and the second is the treatment this traffic aggregate experiences within the DS domain. PDBs are always specified over a known network topology, and cannot be generalized to any topology.

[6] , Guidelines for PDB specifications can be found in RFC 3086 on the IETF website at http://www.ietf.org/rfc/rfc3086.txt

3. Expedited Forwarding Per Hop Behavior

This thesis focuses on the evaluation of some of the properties of the Expedited Forwarding (EF) PHB. Therefore, this chapter concentrates on describing this in more detail. EF PHB is defined for providing a low loss, low delay, low jitter and assured bandwidth end-to-end service through autonomous Differentiated Services Domains without per flow queuing [4].

The EF PHB is defined as a forwarding treatment for a particular DS aggregate where the departure rate[7] of the aggregate's packets from any interface of a DS node must exceed the arrival rate of the aggregate's packets for that interface [4].

The Experiments 1 to 6 on the EF PHB deal with the characteristics delay, jitter and bandwidth. All these characteristics in addition to packet loss are considered for the EF PDB case study. Loss is not noted for the EF PHB experiments as all the analysis is done within a DS core node and no loss is observed at this node as the incoming EF traffic is already conditioned at the ingress DS node. Unless explicitly stated as inter-packet jitter, jitter for the EF PHB, is defined as the variation between the minimum and maximum delays and termed delay jitter [7].

3.1. EF PHB Codepoint

The EF PHB is assigned the codepoint '101110' from the Standards Action pool [4]. This codepoint is marked at the ingress nodes of the DS domain. By definition, packets marked with the EF PHB will never be promoted or demoted to another PHB unlike the packets belonging to the AF PHB. EF traffic flows should be allowed to traverse multiple DS domains provided appropriate Service Level Agreements (SLAs) exist between every adjacent DS domain along the path of the traffic. However the absence of an SLA between two DS domains causes the ingress router of the downstream DS domain to drop the packets it receives from the upstream DS domain. This, of course, can be the source of problems and this factor has to be accounted for in any case study of the EF PDB.

[7] , The EF departure rate, EF service rate and EF configured rate at a DS node have been used interchangeably in this thesis.

3.2. Packet Delays Experienced by the EF PHB

Delay experienced by end-to-end transfer of packets can be attributed to at least five reasons: packetization delay, forwarding delay, queuing delay, serialization delay, and propagation delay.

Packetization delay is the duration of time at the sending host, that digital samples are held for placement into the packet payload until enough samples are collected to fill the packet payload.

Forwarding delay takes place in the sending host and the routers, and is the time taken to receive a packet, make a forwarding decision and then begin sending the packet on the outgoing interface.

Queuing delay is the delay experienced by a packet within routers and switches while waiting on packets that arrived before it to be serviced and sent on the outgoing interface. At any point in time a packet may be waiting in queue a variable amount of time while awaiting access to the output link. The instantaneous state of the router or switch decides the length of this delay.

Serialization delay occurs at the sending host and the intermediate routers, and is the time taken to physically put the bits of a packet on the wire when a node transmits a packet. Serialization depends upon the size of the packet and the speed of the output interface. It is not affected by the maximum EF configured rate for that interface. Serialization delay is always computed using the speed of the output interface.

Propagation delay is based on the speed of the transmission medium and can be considered a fixed delay of the topology under consideration, as it is governed by the physical properties of the transmission medium along the path.

We usually cannot control the forwarding delay, serialization delay, or propagation delay since they are properties of the hardware infrastructure, but we can usually try to control the queuing delay. The EF paradigm aims at reducing this queuing delay for EF traffic. Queuing delay in a well designed and properly functioning DS network can be several orders of magnitude less than the propagation delay. Further, policing and conditioning the traffic at the ingress DS nodes can result in bounding delay jitter, packet loss and the provision of assured bandwidth to EF traffic.

3.3. Characteristics

In order to provide the qualities namely low loss, delay and jitter, that are characteristic of the EF PHB, to a flow of traffic, this traffic should either experience no queues or should spend minimal amount of time in the queues inside the DS nodes. This is the only way delay and delay jitter can be bounded. We can ensure that packets marked for EF forwarding are routed through the network with the highest priority. Provided the DS network has been provisioned appropriately, none of the EF traffic queues will ever overflow, reducing packet loss to zero.

To provision a DS network to handle the EF traffic appropriately, and provide QoS guarantees, every node in the DS network supporting the EF PHB should be configured such that

a) The departure rate for this traffic flow in that node is higher than the maximum arrival rate, and
b) This traffic flow is appropriately isolated from other traffic flows belonging to other PHBs. (The latter could be AF and BE PHBs.)

It is recommended that the EF traffic forwarded by any node should average at least the EF configured rate when measured over any time interval greater than or equal to the time required to transmit a Maximum Transmission Unit (MTU) sized packet on that interface [4]. Ideally the EF departure rates and arrival rates can be the same, but practical considerations

limit us from doing this. The reason being, at 100 percent utilization[8], as the node cannot receive a packet on its input interface and finish forwarding the packet on its output interface in zero time, the node will always be backlogged with at least one packet. The Experiment 3 on the EF PHB in this thesis clearly demonstrates this point.

The EF traffic flow should also be conditioned at the DS edge node that is the ingress into the DS domain, so that it will never exceed the configured EF departure rate in any node in the path of that traffic-flow within that DS domain. EF traffic in excess of the maximum EF configured rate cannot be policed to other codepoints, and should be dropped at the DS ingress edge router.

3.4. Selection of Schedulers to Implement EF PHB

The EF PHB could be implemented with any scheduler that provides priority to select the traffic. The simplest option is the Priority Queue (PQ) Scheduler with the EF aggregate assigned the highest priority up to an EF configured rate. In this case as long as there are EF packets to be forwarded in any DS node, other traffic will be queued up until the time when there is no more EF traffic to forward, and provided the EF traffic rate is less than the EF configured rate for that output interface in that DS node.

A Round Robin (RR) Scheduler is the simplest emulation of the Generalized Processor Sharing scheduling discipline, where instead of serving each packet queue in an infinitesimal time, all the packet queues are served in a finite amount of time [10]. However no priority exists within the RR scheduler and hence it cannot be directly used to implement the EF PHB. A modified Round Robin scheduler called the Weighted Round Robin (WRR) Scheduler can be used. It is the one in which integral weights are assigned to each of the packet queues and the queues are served in proportion to the weight assigned to them.

Most practical implementations of the EF PHB use a single high priority queue or a single high-weight queue in class based queuing schedulers [12]. One of the limitations of the WRR

[8], Utilization is defined as the ratio of net EF arrival rate to the configured EF departure rate at the DS node.

schedulers is that the amount of data taken from a queue is based on the number and size of packets served from that queue in the measurement time. This limitation isn't present in the PQ scheduler as a queue is given priority up to a certain bandwidth of the output link. These implementations also offer a high degree of scalability, with minimum complexity and hence are attractive options for implementing the EF PHB [12]. Hence, for the experiments on EF PHB discussed in this thesis, the PQ scheduler is compared with the WRR scheduler. Experiments 1 to 6 on the EF PHB in this thesis evaluate the PQ scheduler and the WRR scheduler, to determine the conditions and the extent to which one is better than the other for implementing the EF PHB.

3.5. Policing EF PHB

Just as the EF traffic flows need to be protected from other traffic, other traffic also needs to be protected from unconditioned EF traffic. A policer is used to condition the traffic at the DS ingress node in order to ensure that EF traffic is less than the maximum EF arrival rate contracted for in the SLA. If this is not done, the EF traffic flows could starve the traffic belonging to other PHBs by consuming all the resources. Conditioning is achieved with the help of a token bucket policer associated with the EF queue. The token bucket policer is the recommended way of implementing the EF PHB [4].

The token bucket policing mechanism works as follows [27]: Tokens are added to a bucket of depth d tokens, at a rate r tokens per second. If the bucket is full, new tokens are discarded. Each token is worth a pre-defined number of bytes. Whenever a packet arrives at the queue, and, if there is a sufficient number of tokens to cover the packet size, the packet is preserved (queued) until it is sent on to the output interface, else it is discarded. Rate r is the maximum steady-state rate at which the data should be received for a queue. Higher steady-state input rates result in packet loss. The bucket depth d, determines the amount of back-to-back data that can be received for the queue [27]. Thus the two parameters, the token bucket depth d and the rate r, characterize the token bucket policer.

3.6. Redefinition of the EF PHB

The EF PHB was recently redefined by the IETF. It is now characterized with mathematical relationships [7]. This was done to formally define the timescale over which the configured EF rate should be measured. The earlier definition of the EF PHB was much more relaxed with respect to the timescales over which the measurements should be made [7][8]. Hence even measurements on a compliant EF stream (based on that definition) and carried over short intervals, could yield noncompliant results as proved in [12]. With short measurement periods, sampling errors may arise giving incorrect statistics. With longer measurement periods, an averaging effect will take place, and noncompliance in subsections of the measurement period may go undetected.

The new definition specifies that every DS node supporting the EF PHB must be characterized by

a) A maximum EF departure rate, and
b) Bounds on the deviation of the actual departure time from the ideal departure time of each packet [7][8].

This is done as follows. The EF behavior is now defined by two sets of equations. One set applies to the EF aggregate as a whole, and the other set applies to each individual EF packet. The following equations need to be satisfied on a DS node supporting EF PHB on an interface configured at a maximum rate R [7].

For an EF aggregate behavior as a whole:

$D_j <= F_j + E_a$ for all $j > 0$ _____(1)

Where, D_j = actual departure time of packet j, F_j = ideal departure time of packet j, E_a = error term for the treatment of the EF aggregate. E_a represents the worst-case deviation between the actual and ideal departure time of an EF packet. Hence E_a provides an upper bound on $(D_j – F_j)$ for all j.

Furthermore, F_j is defined by basic relationships f_0 = 0, d_0 = 0, and the recursion

F_j = max (A_j, min (d_j-1, f_j-1)) + L_j/R for all j >0 _____(2)

Where, D_j = actual departure time of packet j, F_j = ideal departure time of packet j, A_j = arrival time of packet j, L_j = size of the j^{th} packet in bits, R = maximum configured EF rate at output I in bits/second.

The second set of equations applies to each individual packet marked for EF forwarding:

D_j <= F_j + E_p for all j >0 _____(3)

Where, D_j = actual departure time of packet j, F_j = ideal departure time of packet j, E_p = error term for the treatment of individual EF packets. E_p represents the worst-case deviation between the actual and ideal departure time of an EF packet. Hence E_p provides an upper bound on (D_j – F_j) for all j.

Furthermore, F_j is defined by basis relationships f_0 = 0, d_0 = 0, and the recursion

F_j = max (A_j, min (d_j-1, f_j-1)) + L_j/R for all j >0 _____(4)

Where, D_j = actual departure time of packet j, F_j = ideal departure time of packet j, A_j = arrival time of packet j, L_j = size of the j^{th} packet in bits, R = maximum configured EF rate at output I in bits/second.

E_a and E_p are both expressed in units of time. The maximum EF configured rate R can be the line rate or less, and E_a and E_p can either be specified as a function of R or as a worst-case value for all possible values for R. E.g. for a 100Mbps output interface, R can be either 100Mbps or less and the E_a and E_p can either be specified at 10Mbps, 20Mbps etc or as a worst case value for all possible values of R.

When EF packets are stored in a single first in first out (FIFO) queue awaiting service in a DS node, the E_a value is equal to the E_p value. No packet re-ordering takes place within an EF micro-flow or between EF micro-flows. This is the case in most practical implementations of the EF PHB [7][8] and also in the simulations in this thesis. Let us call

this common value as E. Thus E_a = E_p = E. The relationship between this E and the delay jitter is explained in Section 3.6.5. of this thesis.

3.6.1 Measuring E_a

A method for measuring E_a is proposed here. The equations 1 and 2 explained in Section 3.6. of this thesis, can be implemented by maintaining, for each outgoing interface, a linked list of elements whose members are "the packet arrival time". A unique linked list is maintained for each output interface. Whenever a packet is received, on any one of its input interfaces, destined for a specific output interface, the element is filled with the packet arrival time and added to the tail of the linked list belonging to this output interface. Whenever a packet is sent out of that particular interface, the departure time is noted and the arrival time from the element at the head of the linked list is retrieved and the E_a is calculated. The first time this computation is done, the E_a value is stored. For successive E_a computations, if the calculated E_a value is larger than the previously stored E_a value, the new value is stored. This is followed by the removal of the element at the head of the linked list. As no state of packets is maintained, this processing can be performed in real time.

A setup similar to the one described above, but using arrays in AWK scripts was used while measuring the E_a values from the data collected in trace files after a simulation run in this thesis. Sample E_a calculations are shown in Section 3.6.3. of this thesis.

3.6.2. Measuring E_p

A method of measuring E_p is proposed here. The equations 1 and 2 explained in Section 3.6. of this thesis governing E_p can be measured by maintaining a linked list of structures whose members are packet arrival time and a unique ID that uniquely identifies the packet. A unique linked list is maintained for each output interface. Whenever a packet is received, on any one of its interfaces, destined for a specific output interface, the structure is filled with the packet arrival time and the unique ID and then added to the tail of the linked list. Whenever a packet is sent out of a particular interface, the departure time is noted and the corresponding structure that identifies this packet in the linked list is referenced by means of

the unique ID, to obtain the packet arrival time. The E_p can now be calculated. The first time this computation is done, the E_p value is stored. For successive E_p computations, if the calculated E_p value is larger than the previously stored E_p value, the new value is stored. This is followed by the removal of this structure from the linked list.
A setup similar to the one described above, but using arrays in AWK scripts was used while measuring the E_p values from the data collected in trace files after a simulation run in this thesis. Sample E_p calculations are shown in Section 3.6.3. of this thesis.

One issue is what happens when packet arrival is recorded, but the packet is, for some reason, dropped in the DS node. Its entry could remain in the linked list and provide incorrect results. Hence, in the case a packet is dropped, its member in the linked list must be removed, and this event must be logged. These events can be used for throughput and packet loss calculations. Thus the dropped EF packet does not contribute to the measurement of E_a and E_p of the EF traffic, as recommended by IETF [7]. E_a and E_p calculations are made only on packets that were successfully forwarded by the node. This was also the method adopted in the simulations in this thesis.

A drawback of measuring E_p in a real implementation using the technique explained above is that the processing time per packet in the node will increase. This will be because, on receiving a packet destined for a specific interface, its IP header will have to be read to extract the fields and to generate a unique ID for this packet. Again, while sending this packet, the fields from its IP header will have to be extracted to determine its unique ID. Searching the linked list to retrieve the structure belonging to this specific packet in the linked list will also add to the overhead. One way in which this overhead can be reduced is for the measurement and monitoring to happen by means of a periodic offline (asynchronous) test using previously collected data, rather than in real time.

The IETF has recommended the use of the equations 1, 2, 3 and 4 explained in Section 3.6. of this thesis to characterize the EF service provided by DS nodes, but possible values of the factors E_a and E_p have not been provided as they are implementation dependant. The

intent of this thesis has been to implement the two sets of equations for E_a and E_p and to determine the general range of possible values of E_a and E_p that are possible in a DS node under certain conditions. The relation of the E_a and E_p terms to the delay jitter was also determined. A study of the possible values for R, which is the EF configured rate and the corresponding E_a and E_p values was determined in Experiment 5 on the EF PHB, where the two sets of equations governing E_a and E_p were applied to a network with a Priority Queue scheduler implementation and to a network with Weighted Round Robin scheduler implementation.

3.6.3. Sample Calculations

Considering the EF aggregate is served in FIFO order (i.e. E_a = E_p = E) at a DS node, and considering we begin measurement at time zero. Let the speed of the output link be 100Mbps. Let the EF configured rate at the DS node be 33Mbps. Let the arrival time, A_1, of the first IP packet since the start of measurement at the DS node be 0.250ms from the start of measurement and its size, L_1, be 200bytes (including IP header) which is equal to 1600bits. Applying the equations 1 and 2, and considering f_0 = 0 and d_0 = 0, we get, the ideal departure time F_1 = max (A_1, min (d_0, f_0)) + L_1 / R = 0.298ms. The actual departure time D_1 is 0.365ms. The error term E = (D_1 – F_1) = 0.067ms. The largest value of E observed over the measurement period will provide us with the worst-case value of E.

3.6.4. Figures of Merit

E_a and E_p can be considered as figures of merit of a device [7]. A smaller value of E_a signifies a smoother service rate (scheduling) for the EF traffic flows over short timescales, while a larger value signifies a burstier scheduler whose steady-state properties will hold only over longer timescales. It is preferable to have a smaller E_a. A device with a smaller E_a will conform to the EF PHB over both shorter and longer timescales.

A smaller value of E_p signifies a tighter bound on the delay experienced by an individual packet, whereas a larger value of E_p is an indicator of queuing delays due to flows arriving on multiple interfaces destined for the same output interface. These delays may arise

normally if EF packets arrive simultaneously on more than one input interface around the same time, and are destined for the same output interface. In that case the node can use a scheme to break up the tie between these packets and queue them accordingly. Depending on the order in which the packets are queued, some of them will experience greater delays than others. E_a will always be lesser than E_p in case of a non FIFO EF queue or equal to E_p in case of FIFO EF queue.

3.6.5. Delay and Jitter [7][8]

Assuming that the EF aggregate is served in FIFO order at a DS node (i.e. E_a = E_p = E), the per hop delay can be calculated from the formula

$$D_t = Q_t/R + E \quad (5)$$

Where, D_t is the per hop delay of an EF packet arriving at time t,
R = maximum configured EF rate at output interface in bits/second
And Q_t is size of the EF queue in bits at time t

As an EF packet may be sent out as soon as it is received, the minimum delay through the node, Dmin, can be found by determining the fixed delay through the node. The maximum D_t observed for EF packets over the period of measurement at a DS node is the maximum value of per hop delay called Dmax. The difference between Dmax and Dmin will provide an upper bound on the delay jitter.

3.7. The Delay Bound PHB

The earlier definition of the EF PHB led to the definition of a new PHB called Delay Bound (DB) PHB, which has properties similar to EF PHB, but provides a strict bound on the delay variation (delay jitter) of conformant packets through a node [4][9]. The reasoning behind this definition is that the difference in time between when a packet might have been delivered, and when it is delivered, will never exceed a specifiable bound [9].

For a conformant DB traffic flow, the following equation can be applied to ensure that the flow is treated with the DB PHB.

$D_i - E_i <= S * MTU/A$ _____ (6)

Where, D_i = actual time a DB packet is delivered. This can also be considered as the time a packet will be delivered in the presence of competing traffic.

E_i = ideal time a DB packet should be delivered. This can also be considered as the time a packet will be delivered in the absence of competing traffic.

A = net arrival rate of DB traffic at the node for the specific output interface.

MTU = maximum transmission unit in bits for the output interface.

S = a constant called score that is a characteristic of the device at rate A.

The score is dependant on the scheduling mechanism and configuration of the device.

Experiment 6 on the EF PHB in this thesis used the equation 6 to determine the range of values of A and their corresponding S values.

3.7.1. Codepoint for the DB PHB

The codepoint used for the DB PHB is '101111' and comes from the Experimental pool of codepoints [9]. The EF PHB, in comparison uses a codepoint of '101110' from the Standards Action pool of codepoints. One thing to note is that any codepoint of the form 'xxxxx0' comes from the Standards action pool of DSCP (e.g. the EF PHB codepoint) while any codepoint of the form 'xxxxx1' comes from the Experimental or Local Use pool of DSCP.

3.7.2. Jitter for DB PHB

Inter-packet jitter is not a monitored parameter while considering the DB PHB as the DB PHB aims at bounding a related but different parameter called the delay jitter, which is the variation in delay between the time packets would depart in the absence of competing traffic and when they would depart in the presence of competing traffic i.e. The difference in the maximum and minimum delays [9].

4. Tools and Methods

4.1. Simulation Environment

All the simulations for this thesis were run on the Network Simulator (ns-2) version2, beta release 6 and beta release 8. The simulator was implemented on two SUN ULTRA10 SPARC workstations running the Solaris 7 operating system.

4.2. Network Simulator ns-2

Ns-2 is a freely available discrete event simulator targeted at networking research. A number of organizations are performing research on improving and contributing modules to the simulator. The University of Southern California primarily maintains the source code for the simulator. Ns-2 provides substantial support for simulation of TCP, UDP, multicast and routing protocols over wired and wireless (local and satellite) networks [19].

4.2.1. Limitations of ns-2

Ns-2 is not a polished and finished product, but the result of an on-going effort of research and development. In particular, bugs in the software are still being discovered and corrected. It is entirely the responsibility of the user to validate the results generated by the use of the simulator.

4.2.2. Characteristics of ns-2

Ns-2 is as an object-oriented simulator, written in C++, with an OTcl interpreter as a front-end [19]. The simulator supports a class hierarchy in C++ (also called the compiled hierarchy), and a similar class hierarchy within the OTcl interpreter (also called the interpreted hierarchy). The two hierarchies are closely linked to each other and there is a one-to-one correspondence between a class in the interpreted hierarchy and one in the compiled hierarchy. The root of this hierarchy is the class TclObject. Users create new simulator objects through the interpreter; these objects are instantiated within the interpreter, and are closely mirrored by a corresponding object in the compiled hierarchy. The interpreted class

hierarchy is automatically established through methods defined in the class TclClass. User instantiated objects are mirrored through methods defined in the class TclObject. There are other hierarchies in the C++ code and OTcl scripts that are not mirrored in the manner of TclObject.

4.2.3. Need for Split Language Programming

Systems programming is required for the simulation of communication protocols where access to, and manipulation of packet headers and bytes is possible. These protocols should be able to run over large data sets to appropriately model the algorithms and generate sufficient information from which results can be derived with an acceptable degree of confidence. All these above mentioned tasks make run-time speed an important criterion.

The C++ programming language is chosen for the task of implementing algorithms and protocols, as it can be designed in a modular manner allowing individual contributors to provide small sections of the simulator. Running the executable code of a compiled C++ package is very time efficient.

The other task of any simulation involves fine-tuning and varying the parameters used for successive simulation runs. Sometimes it is important to vary the configurations marginally and study the performance. For all these tasks, if we had to make changes to the C++ program, it will be a very time consuming job. There is also the chance of unknowingly modifying a section of the source code of the algorithms, which can ultimately lead to simulation results that can't be accounted for, invalidating the entire simulation. For all these reasons, the time required to change the parameters and start the next run becomes important. OTcl is chosen for this task. As it is an interpreted language, it does not need to be compiled after making changes to the simulation scripts.

Tclcl is used for linking C++ objects to OTcl variables.

4.2.4. Usage of Languages

Development of newer modules for the simulator in order to implement newer protocols and algorithms or to improve or extend the behavior of existing modules requires the use of the C++ language. The entire package will have to be recompiled after the changes are made, to actually make the changes take effect for a simulation run. Writing simulations based on the features and behaviors of algorithms and protocols already implemented in ns-2 requires the use of the OTcl language.

An example is: Links are OTcl objects that assemble delay, queuing, and possibly loss modules [19]. If we can design our simulations with these objects, then only OTcl needs to be used. On the other hand the implementation of a special queuing, or scheduling mechanism will warrant the use of the C++ language.

4.3. Overview of the Differentiated Services Module in ns-2

The Advanced IP Networks group in Nortel Networks originally developed the Differentiated Services module for ns-2. This very same module was integrated into the network simulator, version 2, Beta release 8.

The Differentiated Services module in ns-2 currently defines four classes of traffic, each of which has three drop precedences. These drop precedences enable differential treatment of traffic within a single class. Each of the 4 classes is mapped to a separate physical RED queue, thus queuing traffic belonging to this class in this queue. Each physical RED queue is further split into 3 virtual queues - one for every drop precedence. The RED parameters like minimum threshold, maximum threshold can be varied for each of the virtual queues, enabling differential treatment to be meted out to the virtual queues within a physical queue. A class is assigned resources in accordance to its relative priority among other classes. Hence during congestion, packets belonging to a higher priority class will suffer less loss compared to packets belonging to a lower priority class. Also within a class, a packet with lower drop precedence is given preferential treatment compared to a packet with higher drop precedence.

4.3.1. Major Components of the Differentiated Services Module [20]

"*Three major components comprise the Differentiated services module:*

- *Policy: Policy is specified by network administrator about the level of service a class of traffic should receive in the network.*
- *Edge routers: Edge routers marks packets with a codepoint according to the policy specified.*
- *Core routers: Core routers examine the packets codepoint marking and forward them accordingly.*

Diffserv attempts to restrict complexity only to the edge routers."

4.3.2. RED Queue for Diffserv [20]

"In ns, the Diffserv functionality is captured in a Queue object, which is implemented as an alternative to other queue types such as DropTail, CBQ, and RED. A Diffserv queue contains the abilities:

1. *to implement multiple physical RED queues along a single link;*
2. *to implement multiple virtual queues within a physical queue, with individual set of parameters for each virtual queue;*
3. *to determine in which physical and virtual queue a packet is enqueued, according to policy specified.*

The class dsREDQueue consists of four (defined as numQueues_) physical RED queues, each containing three (defined as numPrec) virtual 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 codepoint (or a drop preference), which specifies a certain level of service. The physical RED queue is defined in class redQueue. The redQueue class enables traffic 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. 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. The dsREDQueue class contains a data structure known as the Per Hop Behavior Table (PHB Table). Edge devices handle marking packets with codepoints and core devices simply respond to existing codepoints. However, both devices need to determine how to map a codepoint to a particular queue and precedence level. The PHB Table handles this mapping by defining an array with three fields:

```
struct phbParam {
  int codePt_;  // corresponding codepoint
  int queue_;   // physical queue
  int prec_;    // virtual queue (drop precedence)
};"
```

4.3.3. Policy [20]

"The class Policy is used by the class edgeQueue to handle all policy functionality. A policy is established between a source and destination node. All flows matching the 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 codepoints; and the choice of codepoint depends on a comparison between the aggregate's target rate and current sending rate. Each traffic aggregate has an associated policer type, meter type, and initial codepoint. The meter type specifies the method for measuring the state variables needed by the policer. For example, the TSW Tagger is a meter that measures the average traffic rate, using a specified time window. When a packet arrives at an edge router, it is examined to determine to which aggregate it belongs. The meter specified for that aggregate is invoked to update all state variables. Then the policer is invoked to determine how to mark the packet. Depending on the aggregate's state variables, either the specified initial codepoint is used or a downgraded codepoint is used and the packet is enqueued accordingly. 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 codepoint, and other state information. The Policy class uses a Policer Table to store the mappings from a policy type and initial codepoint pair to its associated downgraded codepoint(s).

Currently, six different policy models are defined, which are:

1. *TSW2CM (TSW2CMPolicer): uses CIR and two drop precedences. The lower precedence is used probabilistically when the CIR is exceeded.*
2. *TSW3CM (TSW3CMPolicer): uses CIR, 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.*
3. *Token Bucket (tokenBucketPolicer): uses CIR and 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.*
4. *Single Rate Three Color Marker (srTCMPolicer): uses CIR, CBS, and an EBS to choose from three drop precedences.*
5. *Two Rate Three Color Marker (trTCMPolicer): uses a CIR, CBS, PIR, and a PBS to choose from three drop precdences."*

5. Experiments

5.1. Network Topology used for the EF PHB Experiments

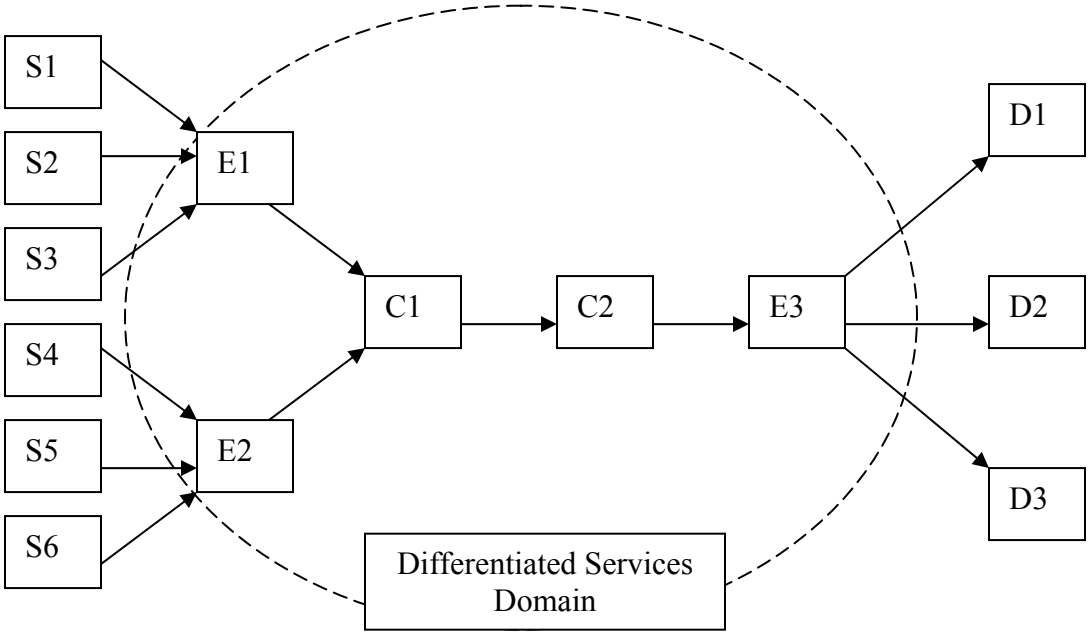

Figure 4: Topology used for Experiments 1 – 6, consisting of 6 Sources, S1 – S6, 3 Sinks D1 –D3, 3 Edge Routers E1 - E3 and 2 Core Routers C1 - C2.

5.2. Characteristics Common to all EF PHB Experiments

Unidirectional constant bit rate (CBR) UDP flows were used for the Best Effort (BE) and Expedited Forwarding (EF) flows in the simulations as Voice over IP is a CBR service and the DS Architecture provides service differentiation only in one direction of traffic flow and hence is asymmetric [2]. Also unlike TCP, UDP has no mechanisms for acknowledging data or providing any form of flow control hence allowing us to study the worst-case situations.

The Quality of Service metrics being monitored for the first 4 Experiments on the EF PHB were the minimum, maximum and average[9] delays and delay-jitter experienced by packets in one core router C2 in a DS domain, as the Per Hop Behavior (PHB) is defined as the externally observable forwarding treatment applied at a DS node to a behavior aggregate [1].

[9] , The terms Average delay and mean delay have been used interchangeably in this thesis.

Two schedulers namely the Priority Queue (PQ) scheduler and the Weighted Round Robin (WRR) scheduler were compared. Other schedulers can be studied as well, using this study as a guideline. This is left for future study.

5.3. Packet Size Determination

The main motivation for the development of the EF PHB was to provide services like Voice over IP and streaming media.

5.3.1. Consideration of the Smallest Packet Size

The table below gives the codec and their corresponding payload size, bandwidth, and bandwidth with Real-Time Transport Protocol (RTP) header compression, without using Voice Activity Detection (VAD) as CBR traffic sources were considered [11].

Table 1: Tabulation of Codec Specifications[10].

Compression Technique (Codec bit rate) in Kbps	Payload Size in Bytes	Bandwidth in Kbps	Bandwidth with RTP header compression in Kbps
G.711 (64)	240	76	66
G.711 (64)	160 (default)	83	68
G.726 (32)	120	44	34
G.726 (32)	80 (default)	50	35
G.726 (24)	80	38	27
G.726 (24)	60 (default)	42	27
G.728 (16)	80	25	18
G.728 (16)	40 (default)	35	19
G.729 (8)	40	17.2	9.6
G.729 (8)	20 (default)	26.4	11.2
G.723.1 (6.3)	48	12.3	7.4
G.723.1 (6.3)	24 (default)	18.4	8.4
G.723.1 (5.3)	40	11.4	6.4
G.723.1 (5.3)	20 (default)	17.5	7.4

[10] , The calculations in Table 1 were done considering a Multilink Point-to-point (MLPPP) layer 2 protocol which adds 6bytes to the Layer 2 header. Fast Ethernet Technology was considered in this Thesis which adds 14bytes to the layer 2 header. This will result in all the values quoted in the Table 1 to be slightly more than those stated for MLPPP.

One thing to note in the table above is that for any given codec, as the voice payload per packet increased, the net bandwidth used reduced, as fewer packets were needed to transmit the same amount of data hence reducing the net overhead caused by IP, UDP and RTP headers which accompany every packet. The disadvantage of increasing the voice payload per packet is an increase in the packetization delay.

The above calculations were computed as follows:
Voice packet size = layer 2 header + (IP + UDP + RTP) Header + voice payload.
Voice packets per second = codec bit rate / voice payload size
Bandwidth = voice packet size * voice packets per second

In the experiments, the default specifications for a G.711 (64) codec were considered. Compression of IP, UDP and RTP headers were not taken into consideration.
Thus the UDP packet size in bytes = UDP header + RTP header + voice payload
= 8 + 12 + 160
= 180bytes
The Ethernet Frame = Ethernet Frame header + IP Datagram header + UDP packet
= (14 + 20 + 180) bytes
= 214 bytes

5.3.2. Consideration of the Largest Packet Size

To consider the other extreme, Ethernet MTU sized packets were considered.
Ethernet MTU = 1500bytes. Of these 1500 bytes, considering the IP header used 20bytes, leaving 1480 bytes for the UDP packet (including header and payload).
Thus UDP packets of 1480bytes were considered.
The Ethernet Frame = Ethernet Frame header + IP Datagram header + UDP packet
= (14 + 20 + 1480) bytes
= 1514 bytes

5.4. Calculation of Delay and Jitter

For the first 4 experiments on the EF PHB, the per hop packet delay experienced by packets in an EF compliant DS node was computed as follows:

Per hop delay = (packet transmission time at DS node) – (packet arrival time at DS node)

For the first 4 Experiments, the delay-jitter was considered to be the absolute difference between the minimum and the maximum packet delays observed over simulation time. This can be equated to the queuing delay experienced by packets within the DS node.

The calculation of worst-case inter-packet jitter was calculated for the Experiment 3, 4 on the EF PHB and the case study of the EF PDB. Inter-packet jitter is defined as the absolute value of the difference between the arrival time difference of two adjacent packets in a micro-flow and their departure time difference [21]. Hence inter-packet jitter = $|(a_k - a_j) – (d_k - d_j)|$

Where j and k are consecutive EF packets arriving at the DS node, a_k and a_j are the arrival times[11] of the j and k packets at the DS node respectively and d_k and d_j are the departure times[12] of the j and k packets at the DS node respectively.

5.5. Simulation Setup

All the links were 100Mbps links.

A mixture of UDP packets of sizes 180 bytes and 1480bytes for BE and EF flows were used.

Simulation time was 10seconds. This time was chosen assuming that the time duration would be sufficient to study the effects of CBR traffic within the network, and also considering that extremely large trace files (of the order of 500 to 700Mb) were generated by each simulation run for this duration. Each of these trace files had to be further processed by AWK programs to retrieve relevant information. A single run of an AWK program can take 3 to 4 hours at a minimum to complete, as at times data in excess of 66,00,000 lines needed to be processed.

Token bucket Policer as recommended for the EF PHB [4] was used.

Number of physical queues in each node: 2 – one for EF PHB and the other for BE PHB.

Number of virtual queues within a physical queue in each node: 2

[11] , Arrival time is the time the last bit of the packet is received on the input interface of the DS node.

[12] , Departure time is the time the last bit of the packet is sent on the output interface of the DS node.

For each cell in the tables for the Experiments on the EF PHB, three measured figures were noted. The first value was obtained by monitoring a single EF flow consisting of 180bytes UDP packets. The second value was obtained by monitoring a single EF flow of 1480byte UDP packets, and the third value was obtained by monitoring the EF aggregate as a whole.

5.6. Experiment 1 on EF PHB

Motivation: The purpose behind this experiment was to study the effects of EF micro-flow aggregation. Values of QoS parameters observed for a single EF micro-flow using the entire resources reserved for EF traffic would be different than those observed for multiple EF micro-flows using the EF PHB, as now the packets belonging to each of the EF micro-flows will be competing with packets belonging to other micro-flows for the same EF resources.

Aim: Performance evaluation of PQ and WRR schedulers as applied to the EF PHB. In this experiment, the number of EF flows through the network were varied and hence the amount of EF traffic arriving to an output interface of a DS node, but the total bandwidth used by the EF aggregate was kept less than, and in one case equal to the maximum EF configured rate. The QoS parameters minimum, maximum and average delays and delay jitter experienced by the EF PHB at one core DS node inside the DS domain were used for the evaluation. The EF queue was assigned 30% of the output link. The network was congested with BE traffic to study scheduler characteristics under worst-case conditions. All the EF and BE sources were started at the same time to model the worse-case situation where a packet from each of the flows arrived at the edge routers at the same time, which would create a burst of packets to flow inside the core of the DS domain and the EF packets towards the end of the burst would face the worst possible delays within DS nodes. Two sets of experiments were conducted, one for PQ scheduler and the other for WRR scheduler, each consisting of 6 scenarios.

Specifications common to both sets of Experiments

Each EF flow was transmitted at a rate of 1.5Mbps, and the burst size was appropriately set to a large value to allow large bursts of packets to accommodate the worst-case scenario, that would lead to packet clustering within the DS domain. There were 4 BE flows carrying traffic at a rate of 31Mbps, thus creating congestion in the network.

Specifications particular to Experiments on the PQ scheduler

Scheduler: Priority Queue Scheduler with queue rate for EF queue set to 30Mbps.

5.6.1. Results for Priority Queue Scheduler

Table 2: Experiment 1 Results for Priority Queue Scheduler.

No. Of EF flows	Minimum delay in ms	Maximum delay in ms	Average delay in ms	Worst-case delay jitter in ms
1	0.114, 0.218,	0.233, 0.336,	0.147, 0.250,	0.119, 0.118,
2	0.114, 0.218, 0.114	0.297, 0.337, 0.337	0.147, 0.252, 0.159	0.183, 0.119, 0.223
5	0.114, 0.218, 0.114	0.293, 0.334, 0.351	0.148, 0.249, 0.151	0.179, 0.116, 0.237
10	0.114, 0.218, 0.114	0.338, 0.340, 0.408	0.150, 0.254, 0.164	0.224, 0.122, 0.294
19	0.114, 0.218, 0.114	0.420, 0.340, 0.557	0.154, 0.253, 0.184	0.306, 0.122, 0.443
20	0.114, 0.218, 0.114	7.859, 0.480, 8.101	0.184, 0.252, 0.214	7.745, 0.262, 7.987

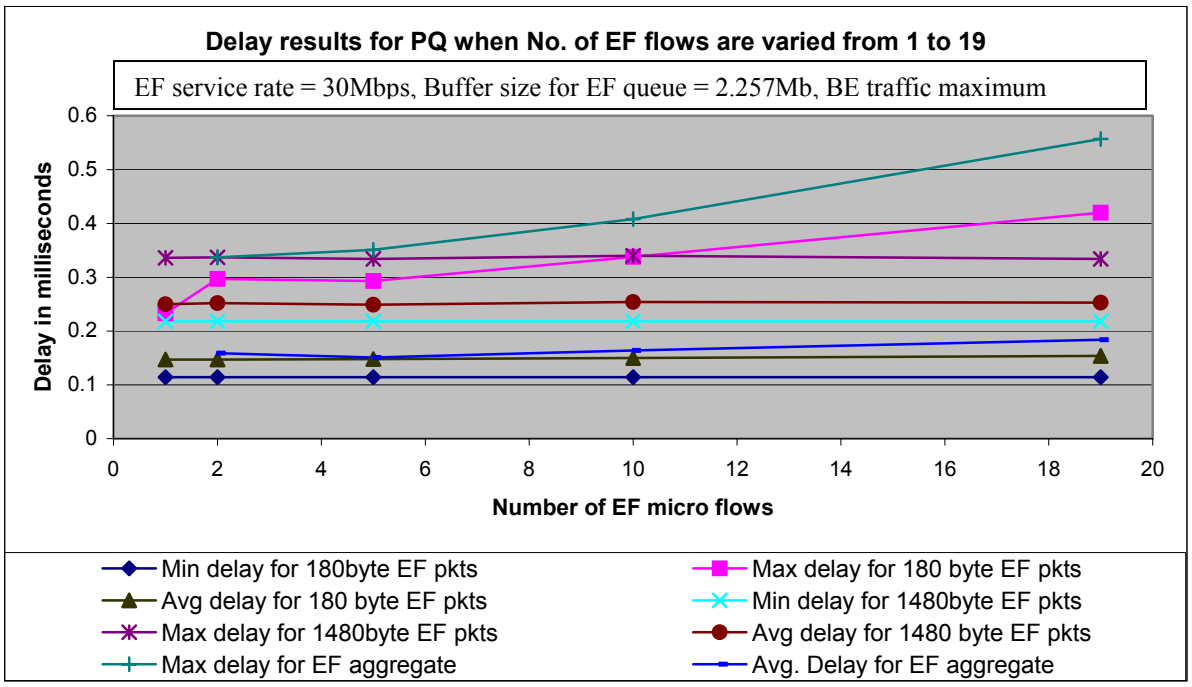

Figure 5: Minimum, maximum and average per hop delays plotted for PQ for micro-flow of small packets, large packets & EF aggregate when EF flows are varied from 1 to 19

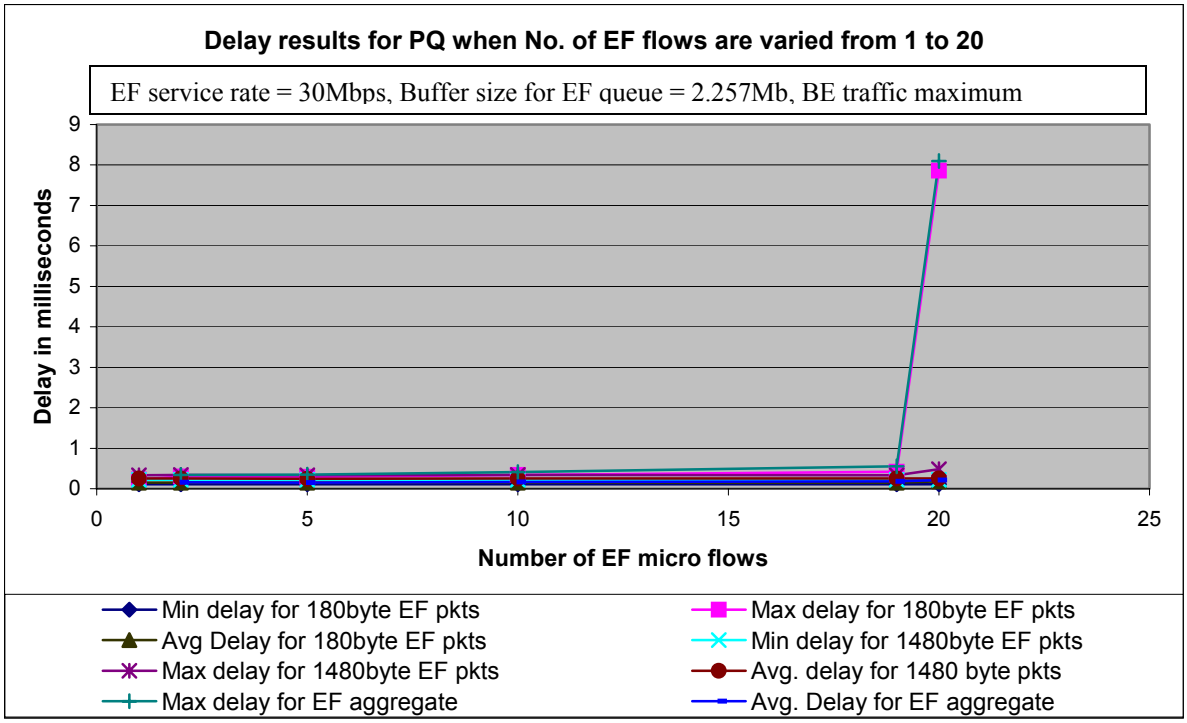

Figure 6: Minimum, maximum and average per hop delays plotted for PQ for micro-flow of small packets, large packets & EF aggregate when EF flows are varied from 1 to 20

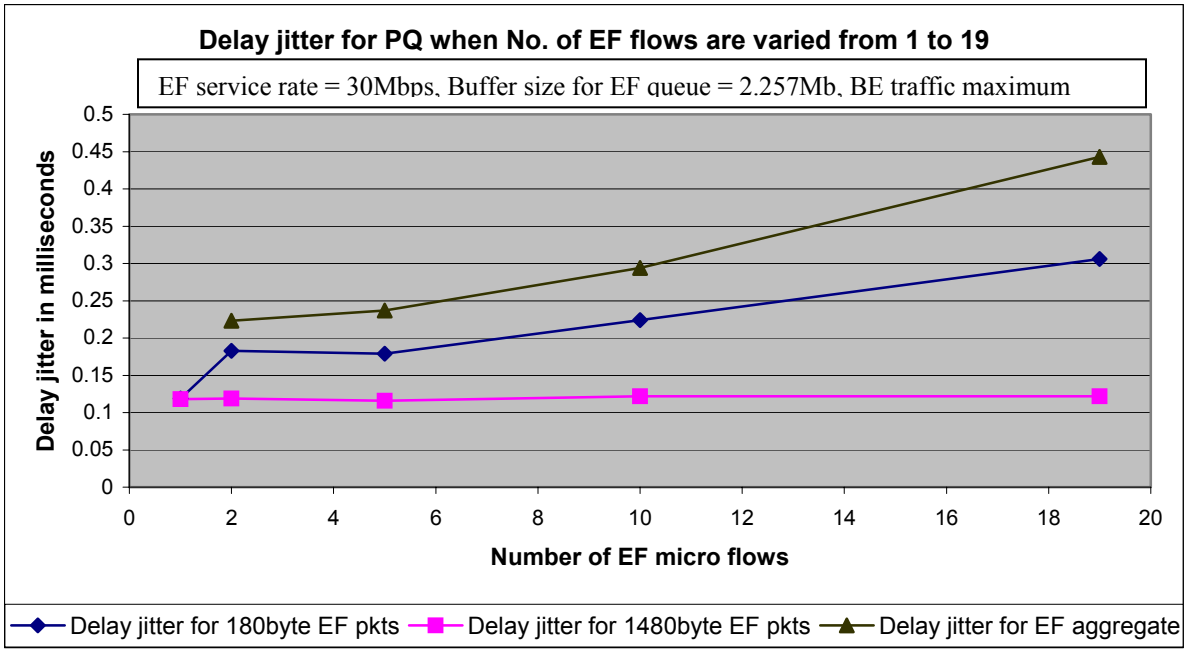

Figure 7: Delay jitter plotted for PQ for micro-flow of small packets, large packets and the EF aggregate when EF flows are varied from 1 to 19.

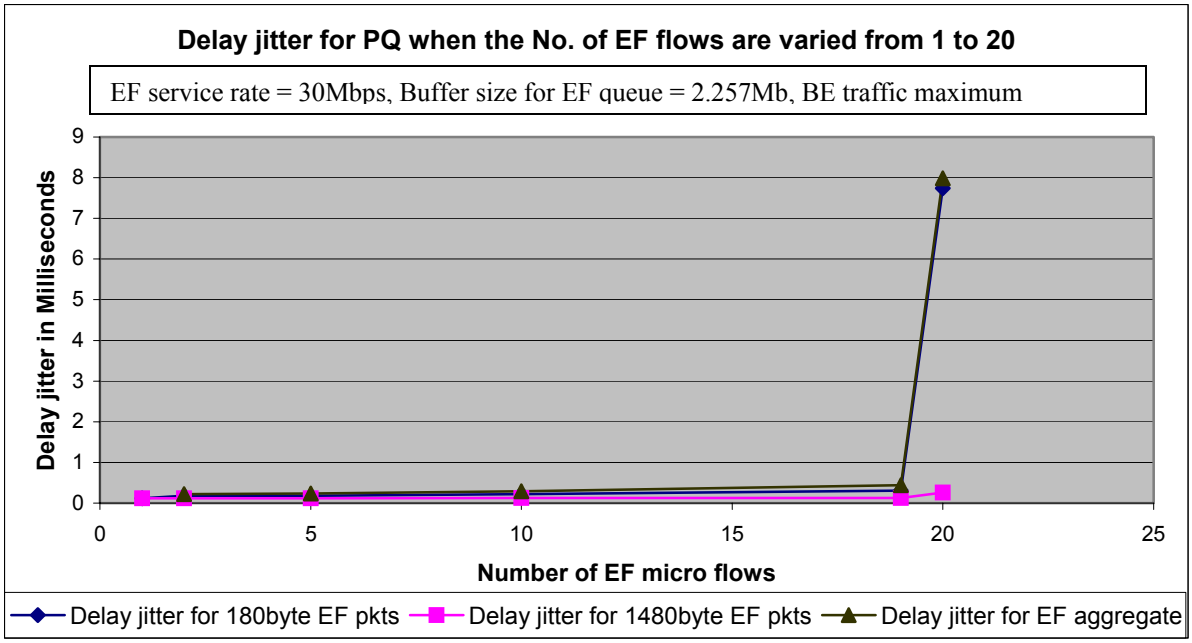

Figure 8: Delay jitter plotted for PQ for micro-flow of small packets, large packets and the EF aggregate when EF flows are varied from 1 to 20.

Specifications particular to Experiments on WRR scheduler

Scheduler: WRR with EF and BE queues assigned a weight of 3 and 7 respectively.

5.6.2. Results for Weighted Round Robin Scheduler

Table 3: Experiment 1 Results for Weighted Round Robin Scheduler.

No. Of EF flows	Minimum delay in ms	Maximum delay in ms	Average delay in ms	Worst-case delay Jitter in ms
1	0.114, 0.218,	0.423, 0.525,	0.211, 0.320,	0.309, 0.307,
2	0.114, 0.218, 0.114	0.424, 0.524, 0.524	0.212, 0.320, 0.223	0.310, 0.306, 0.410
5	0.114, 0.218, 0.114	0.502, 0.711, 0.711	0.210, 0.357, 0.243	0.388, 0.493, 0.597
10	0.114, 0.218, 0.114	0.640, 0.810, 0.952	0.217, 0.360, 0.280	0.526, 0.592, 0.838
19	0.114, 0.218, 0.114	1.513, 0.893, 1.932	0.419, 0.459, 0.574	1.399, 0.675, 1.818
20	0.114, 0.218, 0.114	2.216, 1.513, 2.798	0.560, 0.497, 0.756	2.102, 1.295, 2.684

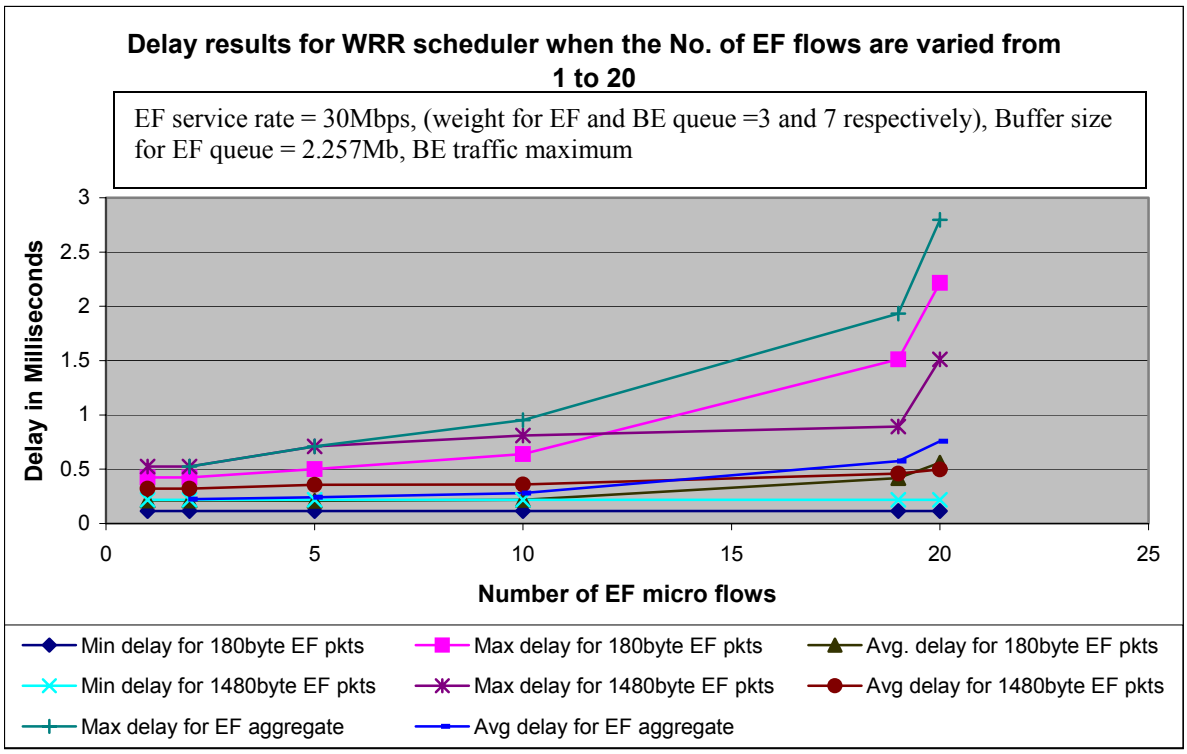

Figure 9: Minimum, maximum and average delays plotted for WRR for micro-flow of small packets, large packets and EF aggregate when EF flows are varied from 1 to 20.

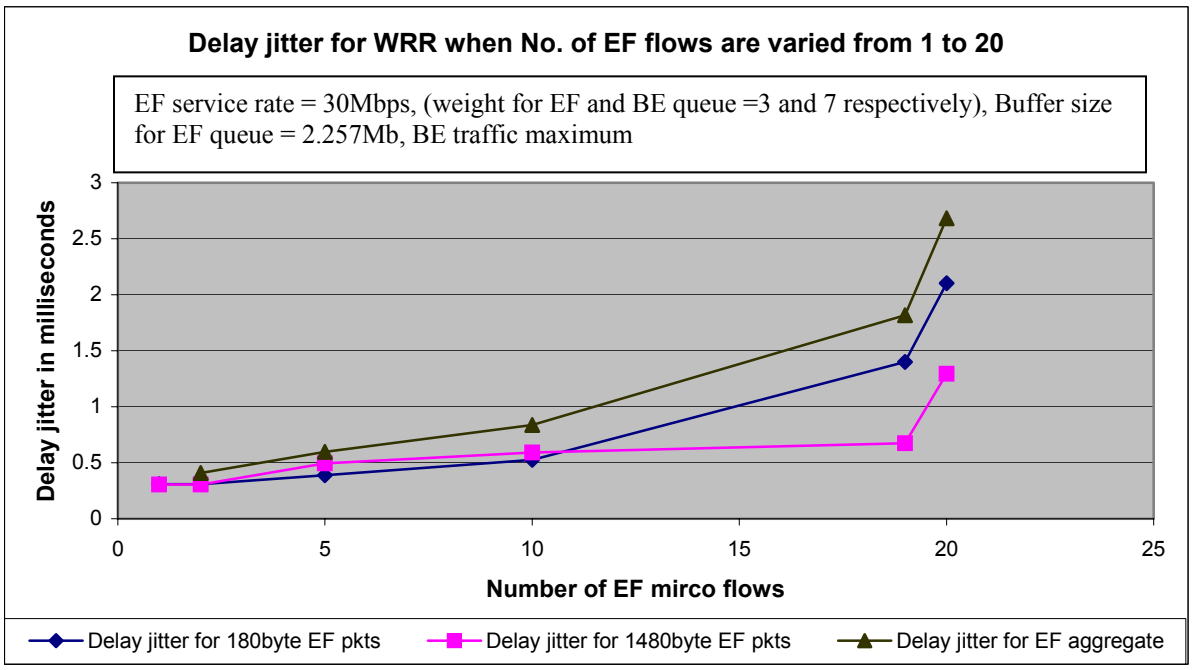

Figure 10: Delay jitter plotted for WRR for micro-flow of small packets, large packets and the EF aggregate when EF flows are varied from 1 to 20.

5.6.3. Conclusions for Experiment 1 on EF PHB

Explanation of the minimum per hop packet delay: The minimum packet delay obtained from the simulations was:

Minimum per hop delay for 180byte UDP packets = 0.114ms

Minimum per hop delay for 1480byte UDP packets = 0.218ms

For 180byte UDP packets after adding the 20byte IP header and 14byte Ethernet header, the Ethernet frame consisted of 214 bytes.

For 1480byte UDP packets after adding the 20byte IP header and 14byte Ethernet header, the Ethernet frame consisted of 1514 bytes.

Serialization Delay for 180byte UDP packets = 214 * 8 / 100Mbps = 0.017ms

Serialization Delay for 1480byte UDP packets = 1514 * 8 / 100Mbps = 0.121ms

On subtracting the serialization delays from the respective total minimum delays, it was observed that in both the cases a constant delay of 0.097ms was obtained.

This was assumed to be the fixed delay within the DS node. This assumption was made based on the documentation in the ns manual where it is stated that: "*A source application generates a packet and forwards it to its target which must be a replicator. The replicator copies the packet and forwards to targets in the active slots, which are either delay modules or loss modules. If they are loss modules, a decision is made whether to drop the packet. If yes, the packet is forwarded to the loss modules drop target. If not, the loss module forwards it to its target, which must be a delay module. The delay module will forward the packet with a delay to its target which must be a receiver application.*"

Hence the minimum packet delay obtained from the simulations was composed of a fixed delay and the serialization delay. As delay jitter, which is the queuing delay, was of interest, and as the fixed delay gets negated while considering the delay jitter, the fixed delay was not set explicitly. Rather the default fixed delay generated by the simulator was used.

While considering maximum per hop packet delays, the queuing delays were considered to be non-zero and variable, and added to per hop packet delays. The worst-case delay jitter in the Tables 2 and 3 of this thesis was equated to the worst-case queuing delay experienced by packets.

5.6.3.1. Priority Queue Scheduler

For an EF flow consisting of 180byte UDP packets: Variation in the number of EF micro-flows did not have much impact on the minimal and average delays. A gradual linear increase was noted in the maximum delay and the delay jitter from 5 EF flows to 19 EF flows at the rate of 0.00846ms per extra EF flow using the EF PHB. However with 20 EF flows, the entire EF bandwidth was used up, and a sharp increase was noted in the maximum delay experienced by packets and correspondingly the delay jitter. The reason for this behavior could be attributed to the fact that an EF packet belonging to this flow came at the end of a burst of EF packets from multiple sources. This coupled with the policing of the EF aggregate to 30Mbps within the DS node resulted in it being delayed till the packets in front of it could be sent out.

For an EF flow consisting of 1480byte UDP packets: Variation in the number of EF flows did not have much impact on the minimum, maximum and the average delays indicating that as the packet sizes approached the path MTU, the variance in the QoS parameters reduced.

For the EF aggregate as a whole: Variation of the number of EF flows did not have much impact on minimum and average delays. A gradual linear increase in the maximum delay and the delay jitter from 5 EF flows to 19 EF flows at the rate of 0.01471ms per extra EF flow using the EF PHB was noted. With 20 EF micro-flows, a sharp increase in all the QoS parameter values was observed. This could again be attributed to the same reason given for the values seen for EF micro-flow of 180byte UDP packets, which was that these packets with very large delays might have been the result of packet clustering and policing of the EF aggregate to 30Mbps. When the EF arrival rate equaled the EF departure rate, on account of the internal delay in a DS node, a backlog would always exist for the EF aggregate as the number of packets arriving at the router would always be at least one greater than the number of EF packets that left the router as the node can't process a packet in zero time [8]. To conclude, by restricting the net EF arrival rate to a value slightly smaller than the EF configured rate, the service provider will be able to provide reliable delay bounds to the EF traffic.

5.6.3.2. Weighted Round Robin Scheduler

For EF flow consisting of 180byte UDP packets, 1480byte UDP packets and EF aggregate:

The WRR scheduler's characteristics were as expected. The minimum delay experienced by packets remained constant. The maximum delay, average delay and the delay jitter increased in a non-linear manner when the number of flows and hence the net EF traffic increased.

Aggregation of EF micro-flows lead to packet clustering within the EF aggregate. This clustering was due to multiple EF queues mapping to the same output interface. In the worst case, 20 EF micro-flows could lead to 20 EF packets arriving back-to-back causing an increase in the instantaneous length of the EF queue and a corresponding increase in the queuing delay. The general observations made in this experiment were validated by similar observations that were noted for EF micro-flow aggregation in [13].

Comparing the above two schedulers, and considering that the EF traffic is less than the EF configured rate it was observed that the PQ scheduler provided far better QoS. For all the three cases considered, namely the single EF micro-flow with 180byte UDP packets, the single EF micro-flow with 1480byte UDP packets and for the EF aggregate as a whole, the maximum delay, average delay and delay jitter experienced with the PQ scheduler, was less than half of that experienced with the WRR scheduler under similar conditions. This conclusion was valid, provided the net EF arrival rate was less than the EF configured rate.

5.7. Experiment 2 on EF PHB

Motivation: For both the schedulers in Experiment 1, very large maximum delays and delay-jitter were noted for the EF traffic when 20 EF flows, each transmitting at a rate of 1.5Mbps, were using the entire 30Mbps reserved for the EF aggregate. It was necessary to determine the fraction of packets that experienced these delays. The determination of the 95 and 99 percentile delays, which could be significantly less than the maximum delays, could aid service providers while defining appropriate SLAs that met the EF traffic requirements.

Aim: After noting the results obtained in Experiment 1, another analysis was made on the results to study the total percentage of packets that were beyond a computed threshold. A normal distribution of packet delays was assumed. A threshold was calculated as shown in Section 5.7.1. of this thesis and all packets above this threshold were recorded. The program test.awk was used.

5.7.1. Evaluation of the use of a Threshold in Providing QoS Guarantees

Understanding of 95 and 99 percentile packet delays: 95 percent of the traffic would see delays less than a particular value. This particular value signified the 95-percentile delay. The 99-percentile packet delay indicated the value such that 99 percent of the traffic would see delays less than that value.

If it is possible to compute a threshold value such that 95% of the traffic would suffer packet delays smaller than the threshold value, then definite guarantees could be provided by service providers to their customers, without mentioning the worst case delays experienced by a small fraction of packets which could be very large.

This threshold was calculated as follows:
Min Deviation from the Avg. = Min (|Avg. Delay – Min Delay|, |Avg. Delay – Max Delay|)
Threshold = (Avg. Delay + Min Deviation from Avg.)

Any packet that experienced a delay larger than this threshold was considered deviant and recorded.

5.7.1.1. Results for the Priority Queue Scheduler

Table 4: Experiment 2 Results for Priority Queue Scheduler.

No. Of EF flows	|Avg– min| delay	|Avg– max| delay	Min(|Avg– min|, |Avg – max|)	Threshold in ms	No. Of deviant packets	Percentage of deviant packets
1	0.033, 0.032,	0.086, 0.086,	0.033, 0.032,	0.180, 0.283,	2257, 283,	22.66, 22.35,
2	0.033, 0.034, 0.045	0.150, 0.085, 0.178	0.033, 0.034, 0.045	0.18, 0.286, 0.204	2291, 277, 2528	21.99, 21.87, 21.63
5	0.034, 0.031, 0.037	0.145, 0.085, 0.200	0.034, 0.031, 0.037	0.182, 0.280, 0.188	2316, 265, 9417	22.23, 20.93, 21.93
10	0.036, 0.036. 0.50	0.188, 0.086, 0.244	0.036, 0.036, 0.50	0.186, 0.290, 0.214	2284, 267, 11817	21.92, 21.09, 20.22
19	0.040, 0.035, 0.70	0.266, 0.087, 0.373	0.040, 0.035, 0.70	0.194, 0.288, 0.254	2318, 263, 18881	22.25, 20.77, 17.74
20	0.070, 0.034, 0.100	7.675, 0.228, 7.887	0.070, 0.034, 0.100	0.254, 0.286, 0.314	786, 261, 16769	7.54, 20.61, 14.36

5.7.1.2. Results for Weighted Round Robin Scheduler

Table 5: Experiment 2 Results for Weighted Round Robin Scheduler.

No. Of EF flows	|Avg – min| delay	|Avg – max| delay	Min(|Avg– min|, |Avg – max|)	Threshold in ms	No. Of deviant packets	Percentage of deviant packets
1	0.097, 0.102,	0.212, 0.205,	0.097, 0.102,	0.309, 0.422,	846, 45,	8.12, 3.55,
2	0.097, 0.102, 0.109	0.212, 0.204, 0.301	0.097, 0.102, 0.109	0.308, 0.422, 0.332	850, 66, 900	8.16, 5.2, 7.7
5	0.096, 0.139, 0.129	0.292, 0.354, 0.468	0.096, 0.139, 0.129	0.306, 0.496, 0.372	949, 140, 4497	9.11, 11.04, 10.47
10	0.103, 0.142, 0.166	0.423, 0.267, 0.672	0.103, 0.142, 0.166	0.320, 0.502, 0.446	1038, 141, 6207	9.96, 11.12, 10.62
19	0.305, 0.241, 0.460	1.094, 0.434, 1.358	0.305, 0.241, 0.460	0.724, 0.700, 1.034	1868, 68, 12793	17.9, 5.36, 12.02
20	0.446, 0.279, 0.642	1.656, 1.016, 2.042	0.446, 0.279, 0.642	1.006, 0.776, 1.398	1576, 67, 9437	15.13, 5.29, 8.076

5.7.1.3. Observations

On evaluating the number of deviant packets from the Tables 4 and 5 and analyzing the Histogram Analysis in Appendix A, it was clear that the distribution of packet delays was definitely not a perfectly normal distribution, but rather sloped gently to the right. The threshold metric cannot be used in its primitive form for determining the 95-percentile values for the WRR scheduler as the percentile figures instead of being a constant value, when plotted against the number of flows, approximated to a bell shaped distribution. A slight modification of this metric may however be used for the PQ scheduler as it is possible to obtain 78 percentile guarantees instead of 95 percentile guarantees for both the EF micro-flows under consideration and the EF aggregate. The possibility of using this modified metric is discussed later in this experiment.

5.7.2. Calculation of Standard Deviation in Per Hop Packet Delay

To study the distribution of these packets around the mean, the variance and standard deviations in packet delays were calculated using the formulae listed below Table 6:

Table 6: Variance and Standard Deviation in Packet Delay when number of flows is varied.

No. of EF flows	Priority Queue scheduler		WRR scheduler	
	Variance in packet delay	Std. Deviation of packet delay in ms	Variance in packet delay	Std deviation of packet delay in ms
1	0.0013 $x10^{-6}$, 0.0013 $x10^{-6}$,	0.03560, 0.03559,	0.0046 $x10^{-6}$, 0.0044 $x10^{-6}$,	0.0677, 0.06619,
2	0.0013 $x10^{-6}$, 0.0013 $x10^{-6}$, 0.0024 $x10^{-6}$	0.03625, 0.03555, 0.048753	0.0046 $x10^{-6}$, 0.0048 $x10^{-6}$, 0.0058 $x10^{-6}$	0.0678, 0.0691, 0.07603
5	0.0013 $x10^{-6}$, 0.0012 $x10^{-6}$, 0.0017 $x10^{-6}$	0.03632, 0.03454, 0.04072	0.0047 $x10^{-6}$, 0.0097 $x10^{-6}$, 0.0086 $x10^{-6}$	0.0688, 0.0985, 0.0924
10	0.0016 $x10^{-6}$, 0.0013 $x10^{-6}$, 0.0032 $x10^{-6}$	0.04061, 0.03622, 0.05668	0.0079 $x10^{-6}$, 0.0098 $x10^{-6}$, 0.0174 $x10^{-6}$	0.0891, 0.0989, 0.1319
19	0.0026 $x10^{-6}$, 0.0012 $x10^{-6}$, 0.0076 $x10^{-6}$	0.05143, 0.03531, 0.08724	0.0807 $x10^{-6}$, 0.0197 $x10^{-6}$, 0.1189 $x10^{-6}$	0.2941, 0.140, 0.344
20	0.0844 $x10^{-6}$, 0.0013 $x10^{-6}$, 0.0877 $x10^{-6}$	0.2905, 0.03582, 0.296	0.1439 $x10^{-6}$, 0.0317 $x10^{-6}$, 0.1875 $x10^{-6}$	0.379, 0.1780, 0.432

Variance = (Σ (packet delay – mean packet delay)^2) / number of EF packets

Standard deviation = sqrt (variance)

The file variance.awk was used to calculate the variance and standard deviation as applicable to packet delay for both the schedulers.

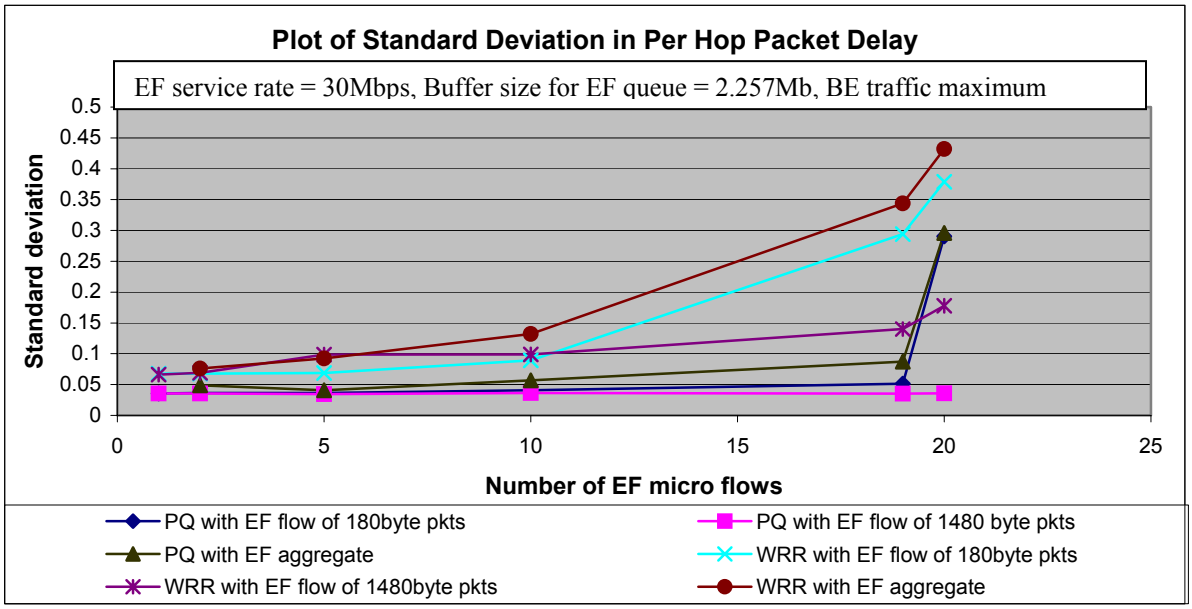

Figure 11: Standard deviation in per hop packet delay when number of EF flows is varied.

5.7.2.1. Observations

Using the standard deviation, ideally it should be possible to estimate the region around the mean that 68, 95 and 99 percent of the packet delays are located using terms like one standard deviation, two standard deviation and three standard deviations from the mean, if the distribution is symmetric about the mean. It was observed that one standard deviation from the mean in per hop packet delay for both the EF micro-flows and the EF aggregate in the case of the PQ scheduler was less than half of that observed in the WRR scheduler for similar flows under similar conditions. In the case of the PQ scheduler, the smaller standard deviation indicated that the variation in delay jitter was restricted to a very small value. In the case of the WRR scheduler, the distribution was more spread out indicating a large variation in delay jitter. Ideally, though not practically possible, the distribution of packet delays for the EF flows should be a sharp spike indicating that all the packets see the same delay which

is the fixed delay. One more observation was that as the number of flows and hence the amount of EF traffic increased in all the cases, the distribution spread out. This was because as the number of flows increased, larger bursts of packets developed within the network, even though every source was CBR. These packet bursts tend to increase the queuing delay experienced by packets in the burst awaiting service. Latter packets in the burst experienced larger delays as packet burst sizes increased.

5.7.3. Computation of 95 and 99 Percentile Delays

In order to study how effective the use of a threshold and standard deviation were, the 95 and 99 percentile per hop packet delays were noted from the data gathered in Experiment 1.

5.7.3.1. Results for Priority Queue Scheduler

Table 7: 95 and 99 Percentile Per Hop Packet Delay for Priority Queue Scheduler.

Percentile analysis for Priority Queue scheduler		
No. Of EF flows	**95-percentile packet delay in ms**	**99-percentile packet delay in ms**
1	0.221, 0.324,	0.230, 0.333,
2	0.221, 0.325, 0.237	0.230, 0.334, 0.315
5	0.221, 0.325, 0.226	0.231, 0.334, 0.267
10	0.225, 0.328, 0.288	0.271, 0.335, 0.336
19	0.256, 0.323, 0.366	0.322, 0.335, 0.444
20	0.299, 0.328, 0.411	0.966, 0.336, 1.065

5.7.3.1.1. Observations

On examining the percentile figures in the Table 7, and referring to the threshold values computed in Table 4, it can be said for both the EF micro-flows under consideration, that by adding a constant of value ranging between 0.039×10^{-3} and 0.041×10^{-3} to the threshold or the second standard deviation[13] from the mean, the 95-percentile delay can be accurately predicted as long as the EF traffic is less than the EF configured rate. The 99-percentile delay for the EF flow consisting of UDP packets of 1480 bytes can also be predicted by adding a constant value ranging from 0.048×10^{-3} to 0.050×10^{-3} to the threshold or the second standard deviation. The 99-percentile delay for EF flow consisting of 180 bytes however

[13] , Second standard deviation is equal to (mean delay + one standard deviation calculated in Table 6).

cannot be predicted accurately by the addition of this simple constant. However to a constant of 0.050 x10^{-3} if an incremental factor of 0.00541 x 10^{-3} is added for an increase in the net EF arrival rate by every 1Mbps, 99-percentile delay guarantees can be provided if the net EF arrival rate is less than the EF configured rate. For the EF aggregate as a whole, adding a fixed constant to the threshold or the second standard deviation did not provide 95-percentile delay guarantees, but depending upon the rate of the EF traffic arriving to the output interface, to the constant of 0.033 x 10^{-3}, an incremental factor could be computed that added 0.00309 x 10^{-3} for an increase in the EF arrival rate by every 1Mbps between 3Mbps and 28.5Mbps. On adding this new constant to the threshold or the second standard deviation, the 95-percentile delay for the EF aggregate could be predicted accurately. To accurately predict the 99-percentile delay for EF aggregate, to the constant of 0.156 x 10^{-3}, an incremental factor could be computed that added 0.00407 x 10^{-3} for an increase in the EF arrival rate by every 1Mbps between 3Mbps and 28.5 Mbps.

The 95 and 99-percentile delay values for the entire EF aggregate are governed by complex factors that need to be researched further. The main reason why metrics like the second standard deviation and the threshold explained above can be used in the determination of 95 and 99 percentile delays depend primarily upon the variation of the average delays. If the average delays remained constant or increased linearly over variation of number of flows and hence the net EF arrival rate, it was easy to compute a constant that would be a characteristic of an implementation with the PQ scheduler and can be used for providing guarantees.

5.7.3.2. Results for Weighted Round Robin Scheduler

Table 8: 95 and 99 Percentile Per Hop Packet Delay for Weighted Round Robin Scheduler.

Percentile analysis for Weighted Round Robin scheduler		
No. Of EF Flows	**95-percentile packet delay in ms**	**99-percentile packet delay in ms**
1	0.318, 0.421,	0.400, 0.490,
2	0.317, 0.424, 0.370	0.400, 0.506, 0.418
5	0.319, 0.547, 0.418	0.401, 0.608, 0.494
10	0.386, 0.547, 0.532	0.504, 0.607, 0.695
19	0.971, 0.704, 1.218	1.140, 0.783, 1.433
20	1.252, 0.782, 1.505	1.549, 1.047, 1.813

5.7.3.2.1. Observations

On analyzing the relationship between the 95 and 99 percentile packet delays and the threshold and standard deviation values from Tables 5 and 6 in this thesis, it was clear that the relation couldn't be established with a simple fixed constant along with the threshold and standard deviation for neither the individual EF flows, nor the EF aggregate for the WRR scheduler as was possible for the PQ scheduler. This was because the 95 and 99 percentile delays followed a non-linear trend. However interpolating between the two extremes of the curve, conservative 95 and 99 percentile delay estimates could be provided. The study of the factors governing the determination of such a constant can be done in a future study.

5.7.4. Conclusions for Experiment 2 on EF PHB

A histogram analysis of all the EF flows that were monitored was made for both the schedulers under varying conditions. From this analysis, the values of the standard deviation, and the values of |min delay – avg. delay| and |max delay – avg. delay| from the Table 4 and 5 in this thesis, it was clear that the distribution of per hop packet delay was asymmetric around the mean delay. A large number of packets had experienced delays lesser than, but close to the average value and a fraction of packets had experienced very large delays. The histograms further proved the point that per hop packet delay distribution was more tightly bound around the mean delay with the PQ scheduler, than the WRR scheduler. The second standard deviation, or threshold with a constant value added to either of them as explained above for the PQ scheduler implementations could be used as figures of merit while characterizing the performances of DS nodes supporting the EF PHB as it was possible to predict these values with sufficient accuracy. Detailed studies would be required to determine constants for providing 95 and 99 percentile delay statistics for the WRR scheduler.

5.8. Experiment 3 on EF PHB

Motivation: Experiment 1 and the percentile analysis in Experiment 2 indicated that if the EF arrival rate equaled the EF departure rate, a fraction of the packets experienced very large delays. It would be useful to determine the amount of over provisioning of resources required for the EF aggregate to obtain low values of the QoS parameters being monitored.

Aim: In this experiment, the number of EF flows were kept constant to 20, each flow was 1.5Mbps, thus bounding the net EF arrival rate to 30Mbps. The EF configured rate at every DS node, which was the share of the output link assigned to the EF aggregate was varied to determine the optimal value that would provide with the lowest minimum, maximum and average delays and delay jitter and inter-packet jitter for the EF traffic whose net arrival rate was bounded. This experiment was also conducted in two sets. One set for the PQ scheduler and another for the WRR scheduler, each consisting of 9 scenarios.

Specifications particular to Experiments on PQ scheduler

Scheduler: Priority Queue Scheduler with EF queue rate varied from 30Mbps to 60Mbps.

5.8.1. Results for the Priority Queue Scheduler

Table 9: Experiment 3 Results for Priority Queue Scheduler.

Configured EF rate in Mbps	Arrival to departure ratio	Minimum delay in ms	Maximum delay in ms	Average delay in ms	Delay jitter in ms	Inter-packet jitter in ms
30	1:1	0.114, 0.218, 0.114	7.859, 0.480, 8.010	0.184, 0.252, 0.214	7.745, 0.262, 7.896	7.660, 0.261, 7.811
30.3	1:1.01	0.114, 0.218, 0.114	0.435, 0.343. 0.587	0.156, 0.253, 0.185	0.321, 0.125, 0.473	0.309, 0.221, 0.425
31	1:1.03	0.114, 0.218, 0.114	0.435, 0.343. 0.587	0.156, 0.253, 0.185	0.321, 0.125, 0.473	0.309, 0.221, 0.425
31.5	1:1.05	0.114, 0.218, 0.114	0.435, 0.343. 0.587	0.156, 0.253, 0.185	0.321, 0.125, 0.473	0.309, 0.221, 0.425
33	1:1.1	0.114, 0.218, 0.114	0.435, 0.343. 0.587	0.156, 0.253, 0.185	0.321, 0.125, 0.473	0.309, 0.221, 0.425
36	1:1.2	0.114, 0.218, 0.114	0.435, 0.343. 0.587	0.156, 0.253, 0.185	0.321, 0.125, 0.473	0.309, 0.221, 0.425
45	1:1.5	0.114, 0.218, 0.114	0.435, 0.343. 0.587	0.156, 0.253, 0.185	0.321, 0.125, 0.473	0.309, 0.221, 0.425
60	1:2	0.114, 0.218, 0.114	0.435, 0.343. 0.587	0.156, 0.253, 0.185	0.321, 0.125, 0.473	0.309, 0.221, 0.425

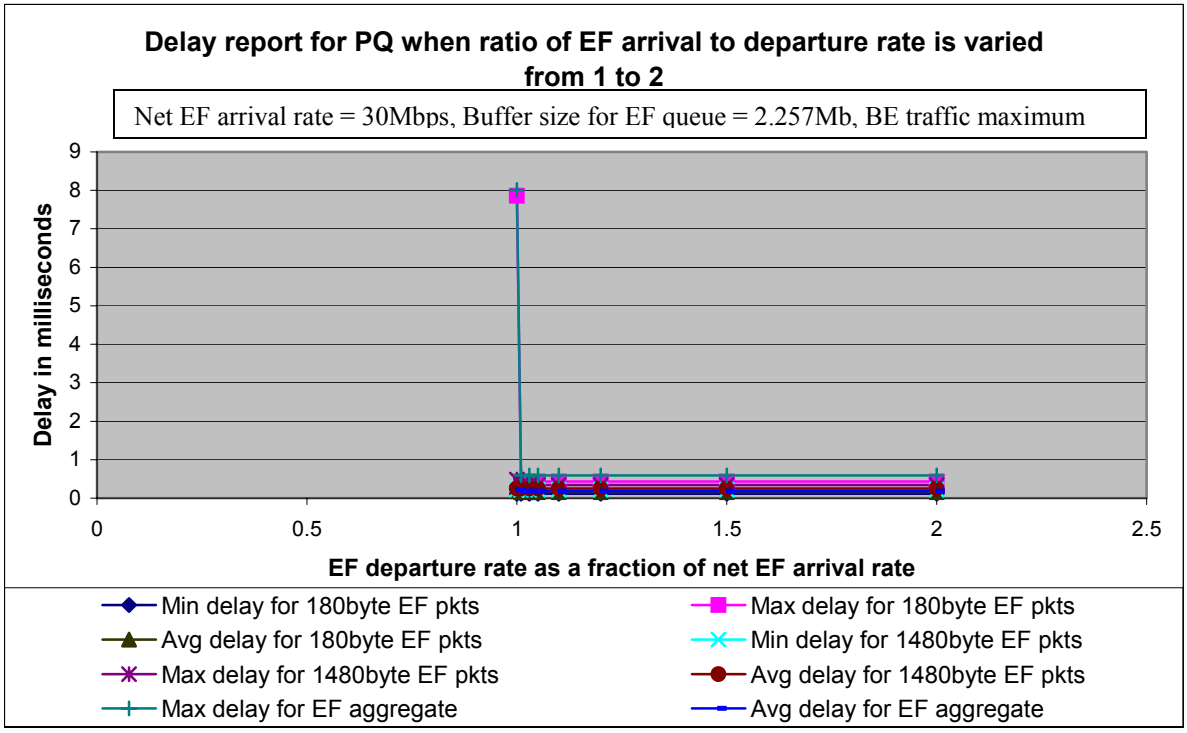

Figure 12: Delay report for PQ when ratio of EF arrival to departure rate is varied from 1 to 2

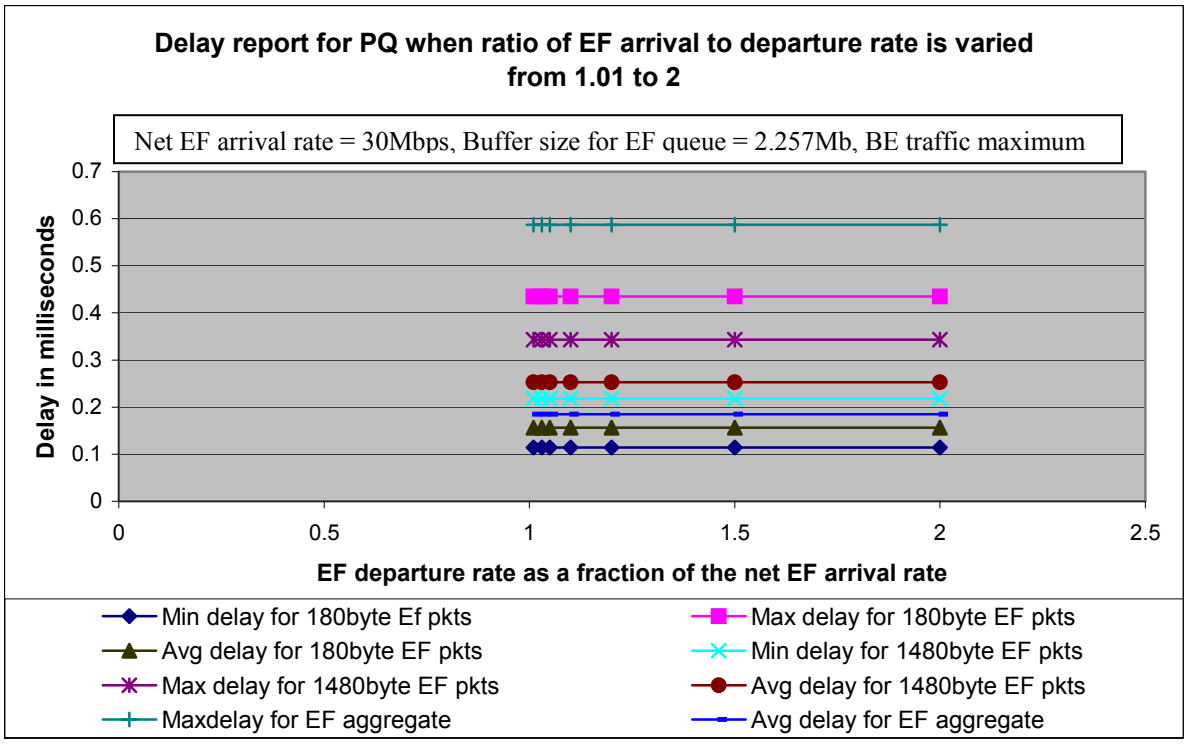

Figure 13: Delay report for PQ when ratio of EF arrival to departure rate is varied from 1.01 to 2

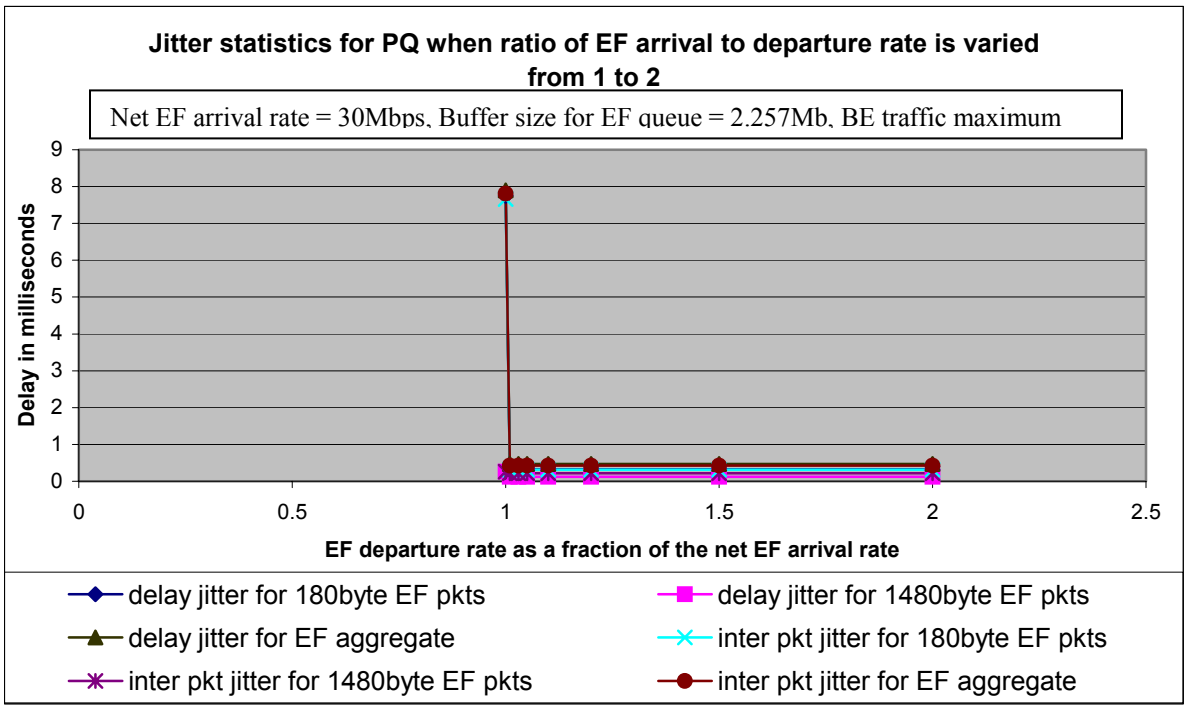

Figure 14: Jitter statistics for PQ when the ratio of EF arrival to departure rate is varied from 1 to 2

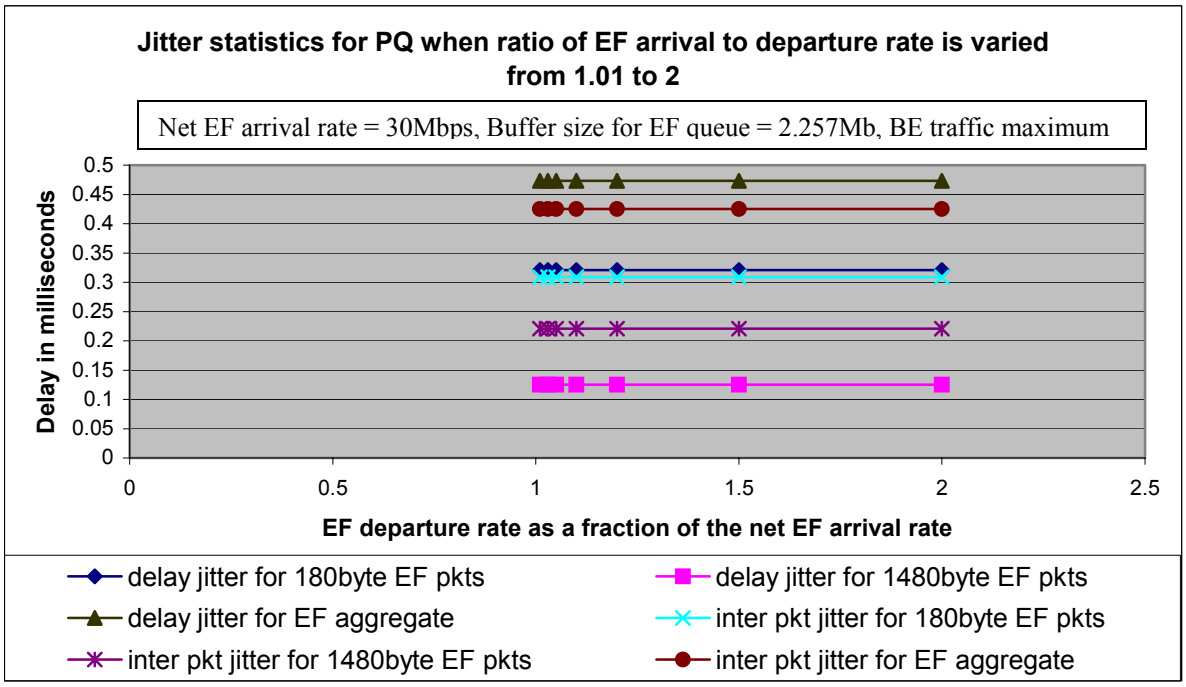

Figure 15: Jitter statistics for PQ when the ratio of EF arrival to departure rate is varied from 1.01 to 2.

Specifications particular to Experiments on WRR scheduler

Scheduler: WRR scheduler with EF queue rate varied from 30Mbps to 60Mbps.

5.8.2. Results for the Weighted Round Robin scheduler

Table 10: Experiment 3 Results for Weighted Round Robin Scheduler.

Configured EF rate in Mbps	Arrival to departure ratio	Minimum delay in ms	Maximum delay in ms	Average delay in ms	Delay jitter in ms	Inter-packet jitter in ms
30	1:1	0.114, 0.218, 0.114	2.216, 1.513, 2.798,	0.560, 0.497, 0.756	2.102, 1.295, 2.684	0.890, 0.843, 1.585
30.3	1:1.01	0.114, 0.218, 0.114	9.947, 8.667, 9.948	0.931, 0.900, 0.946	9.833, 8.448, 9.834	9.525, 8.286, 9.525
31	1:1.03	0.114, 0.218, 0.114	2.499, 1.875, 3.216	0.747, 0.767, 0.888	2.385, 1.657, 3.102	1.939, 1.484, 2.349
31.5	1:1.05	0.114, 0.218, 0.114	4.238, 4.035, 4.239	0.909, 0.883, 0.934	4.124, 3.817, 4.125	3.712, 3.817, 3.593
33	1:1.1	0.114, 0.218, 0.114	2.305, 1.771, 3.044	0.719, 0.770, 0.831	2.191, 1.553, 2.930	1.953, 1.420, 2.392
36	1:1.2	0.114, 0.218, 0.114	1.585, 0.974, 2.022	0.387, 0.487, 0.493	1.471, 0.756, 1.908	1.021, 0.641, 1.374
45	1:1.5	0.114, 0.218, 0.114	1.128, 0.789, 1.478	0.295, 0.392, 0.366	1.014, 0.570, 1.364	0.788, 0.507, 1.072
60	1:2	0.114, 0.218, 0.114	0.553, 0.452, 0.762	0.165, 0.269, 0.217	0.439, 0.234, 0.648	0.321, 0.293, 0.595

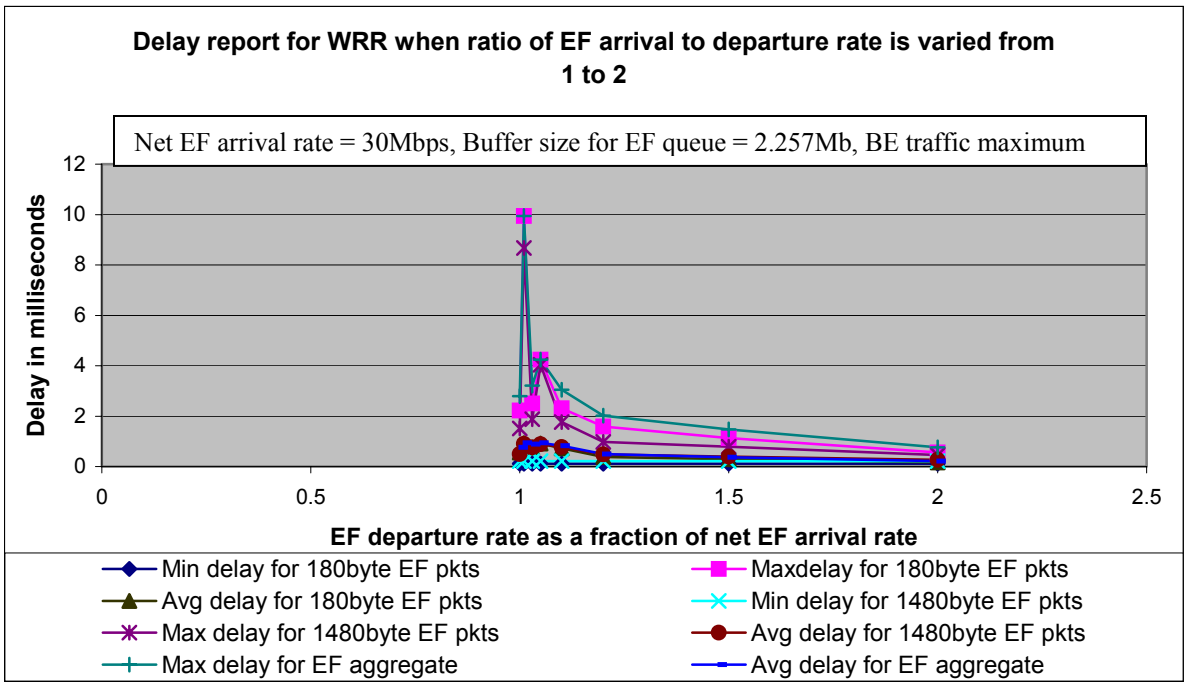

Figure 16: Delay report for WRR when the ratio of EF arrival to departure rate is varied from 1 to 2.

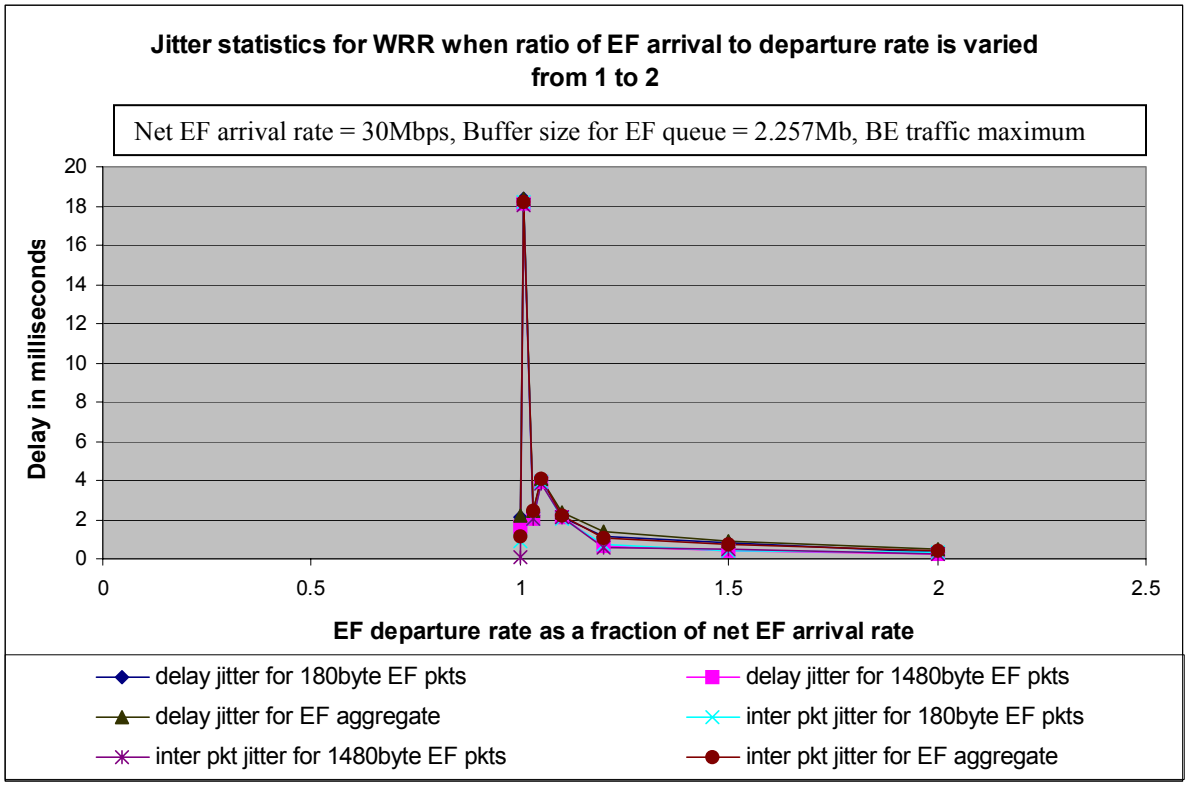

Figure 17: Jitter statistics for WRR when the ratio of EF arrival to departure rate is varied from 1 to 2.

5.8.3. Conclusions for Experiment 3 on EF PHB

The monitored QoS metrics for both the schedulers showed very high values, when net EF arrival rate equaled the EF configured rate at the DS node as also observed in Experiment 1.

Priority Scheduler: It was observed for the PQ scheduler that for an EF traffic arrival to departure rate ratio of 1:1.01, we could ensure that the QoS metrics stabilized in value for both the EF micro-flows under consideration and the EF aggregate as a whole. The EF traffic received the QoS mentioned in Experiments 1 and 2. Increasing the EF configured rate further had no effect on the QoS parameters, which remained constant. Service providers could use this ratio while configuring their DS nodes with PQ implementations for the EF aggregate, and while providing guarantees to customers.

WRR Scheduler: The above conclusion did not hold for the WRR scheduler, which showed oscillatory behavior for all the QoS metrics when the ratio of EF arrival to departure rate was in the range 1:1 to 1:1.1. The main explanation for this oscillatory behavior and the high QoS metric values for the EF arrival to departure rate ratios of 1:1.01 and 1:1.05 for the EF queue could be attributed to the basic property of the WRR scheduler and the weights assigned to the EF and BE queues in order to give the EF queue the mentioned departure rates, as the weights had to be integer multiples. Hence, in order to service the EF queue at a rate 1.01 times the arrival rate, it was assigned a weight of 303 and the BE queue, a weight of 697. This meant that after serving every 303 EF packets, the 304^{th} EF packet would be queued in the worst case till 697 BE packets were serviced resulting in an increase in the QoS metric values for packets queued in the EF queue at that time. The same explanation applied when the EF queue was serviced at a rate 1.05 times the net EF arrival rate, as it was assigned a weight of 63 and the BE queue a weight of 137. This was a limitation of the WRR scheduler as the weights had to be selected appropriately to very small integer values, further restricting the freedom in configuring EF departure rates. The general trend observed was an exponential decrease in the QoS parameters namely, minimum, maximum and average packet delays, delay jitter and inter-packet jitter as the arrival to departure rate ratio of EF traffic was increased. Even when the arrival to departure rate ratio for the EF queue with the WRR scheduler was increased to 1:2, the performance still did not match that of the PQ

scheduler whose arrival to service rate ratio for the EF queue was set to 1:1.01. When a ratio of 1:1.5 or greater was maintained for the WRR scheduler, a service equivalent to the one noted in Experiment 1 and 2 on the EF PHB in this thesis could be delivered to the EF traffic.

5.9. Experiment 4 on EF PHB

Motivation: If the arrival rate of EF traffic was maintained constant, and so were the resources reserved for EF traffic, it was necessary to determine how variation in the best effort cross-traffic in the network affected the values of the monitored QoS parameters.

Aim: In this experiment the number of EF micro-flows was kept constant to 20, each flow was carrying traffic at the rate of 1.5Mbps, utilizing the entire 30Mbps reserved for EF traffic. The level of best effort congestion in the network was varied from 0 to 100 percent, by varying the amount of BE traffic flowing through the network. The effect of this variation on the EF traffic was monitored for the PQ scheduler and the WRR scheduler.

5.9.1. Results for the Priority Queue Scheduler

Table 11: Experiment 4 Results for Priority Queue Scheduler.

BE Traffic as a % of total traffic	Minimum delay in ms	Maximum delay in ms	Average delay in ms	Delay jitter in ms	Inter Packet Jitter in ms
0.00342	0.114, 0.218, 0.114	0.381, 0.227, 0.457	0.129, 0.219, 0.157	0.204, 0.009, 0.343	0.204, 0.221, 0.343
7.06	0.114, 0.218, 0.114	0.555, 0.227, 0.599	0.140, 0.219, 0.169	0.441, 0.009, 0.485	0.441, 0.221, 0.485
39.99	0.114, 0.218, 0.114	0.728, 0.227, 0.881	0.147, 0.220, 0.178	0.614, 0.009, 0.767	0.614, 0.221, 0.767
50	0.114, 0.218, 0.114	1.080, 0.328, 1.276	0.175, 0.228, 0.210	0.966, 0.110, 1.162	0.893, 0.221, 1.162
57.14	0.114, 0.218, 0.114	1.252, 0.234, 1.576	0.189, 0.221, 0.229	1.138, 0.016, 1.462	1.079, 0.221, 1.277
62.5	0.114, 0.218, 0.114	1.679, 0.259, 2.044	0.233, 0.227, 0.277	1.565, 0.041, 1.930	1.504, 0.221, 1.632
66.66	0.114, 0.218, 0.114	2.568, 0.328, 2.735	0.351, 0.225, 0.397	2.454, 0.110, 2.621	2.380, 0.221, 2.621
69.168	0.114, 0.218, 0.114	3.687, 0.592, 3.969	0.569, 0.243, 0.627	3.573, 0.374, 3.855	3.441, 0.355, 3.758
69.5	0.114, 0.218, 0.114	5.617, 1.271, 5.617	0.627, 0.257, 0.687	5.503, 1.053, 5.503	5.399, 1.037, 4.257

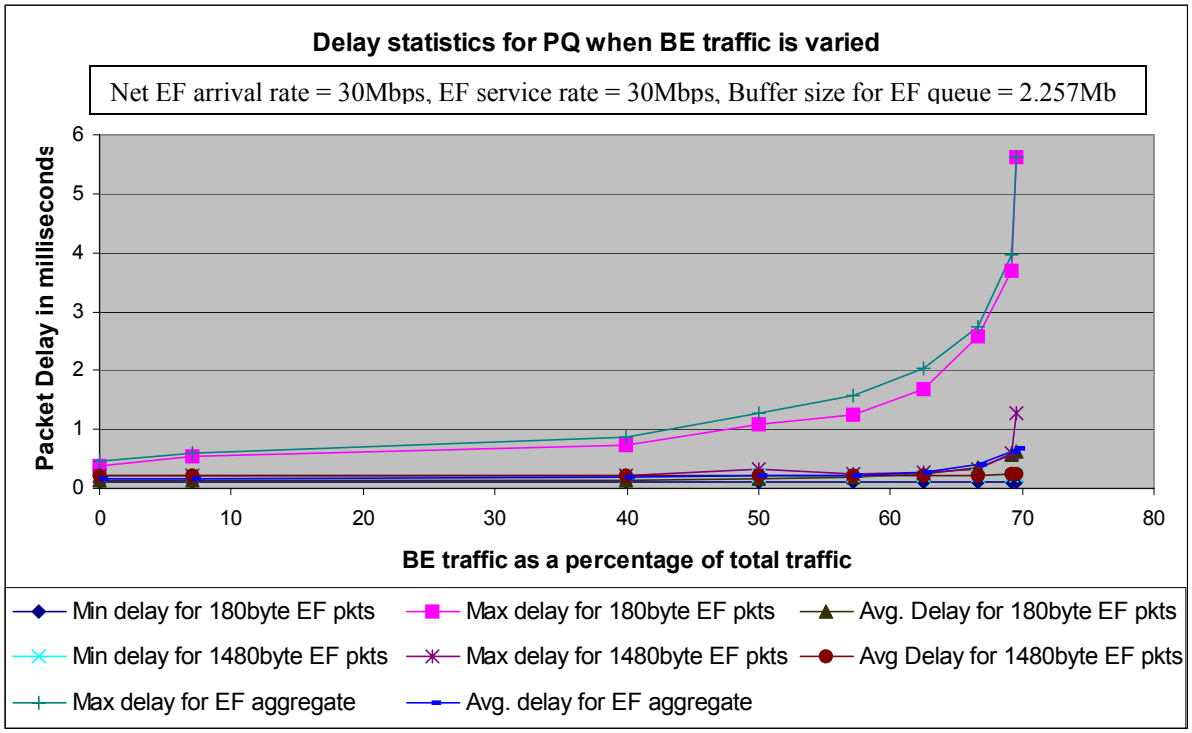

Figure 18: Delay report for PQ when the net BE traffic is varied.

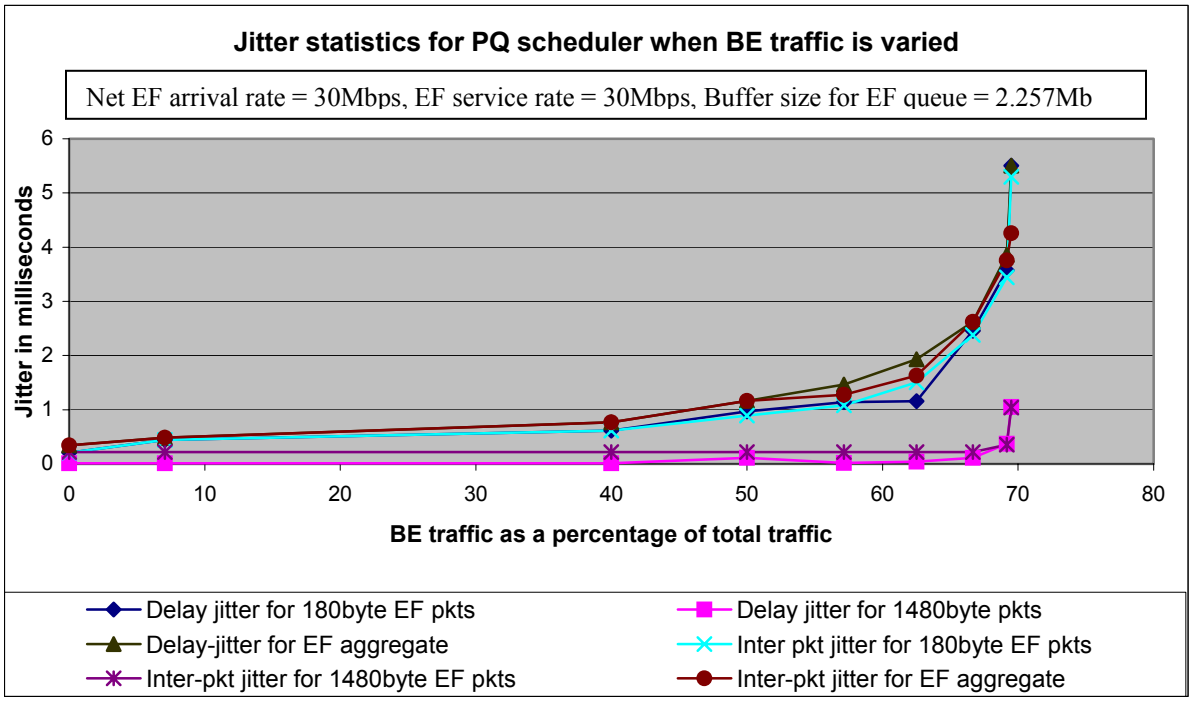

Figure 19: Jitter statistics for PQ when the net BE traffic is varied.

5.9.2. Results for the Weighted Round Robin Scheduler

Table 12: Experiment 4 Results for Weighted Round Robin Scheduler.

BE Traffic as a % of total traffic	Minimum delay in ms	Maximum delay in ms	Average delay in ms	Delay jitter in ms	Inter-Packet Jitter in ms
0.00342	0.114, 0.218, 0.114	0.318, 0.236, 0.457	0.129, 0.219, 0.157	0.204, 0.018, 0.343	0.204, 0.236, 0.343
7.06	0.114, 0.218, 0.114	0.436, 0.236, 0.556	0.141, 0.219, 0.175	0.322, 0.018, 0.442	0.322, 0.236, 0.441
39.99	0.114, 0.218, 0.114	0.595, 0.264, 0.743	0.161, 0.222, 0.205	0.481, 0.046, 0.629	0.481, 0.264, 0.614
50	0.114, 0.218, 0.114	0.712, 0.420, 1.061	0.204, 0.255, 0.278	0.598, 0.202, 0.947	0.598, 0.293, 0.913
57.14	0.114, 0.218, 0.114	0.858, 0.392, 1.213	0.254, 0.241, 0.334	0.744, 0.174, 1.099	0.698, 0.308, 1.012
62.5	0.114, 0.218, 0.114	1.212, 0.420, 1.651	0.342, 0.267, 0.458	1.098, 0.202, 1.537	0.815, 0.336, 1.247
66.66	0.114, 0.218, 0.114	1.405, 0.492, 1.858	0.519, 0.247, 0.680	1.291, 0.274, 1.744	0.933, 0.336, 1.598
69.168	0.114, 0.218, 0.114	1.489, 0.526, 1.942	0.603, 0.287, 0.817	1.375, 0.308, 1.828	1.138, 0.336, 1.574
69.5	0.114, 0.218, 0.114	1.916, 0.985, 2.420	0.642, 0.323, 0.867	1.802, 0.767, 2.306	1.145, 0.666, 1.892

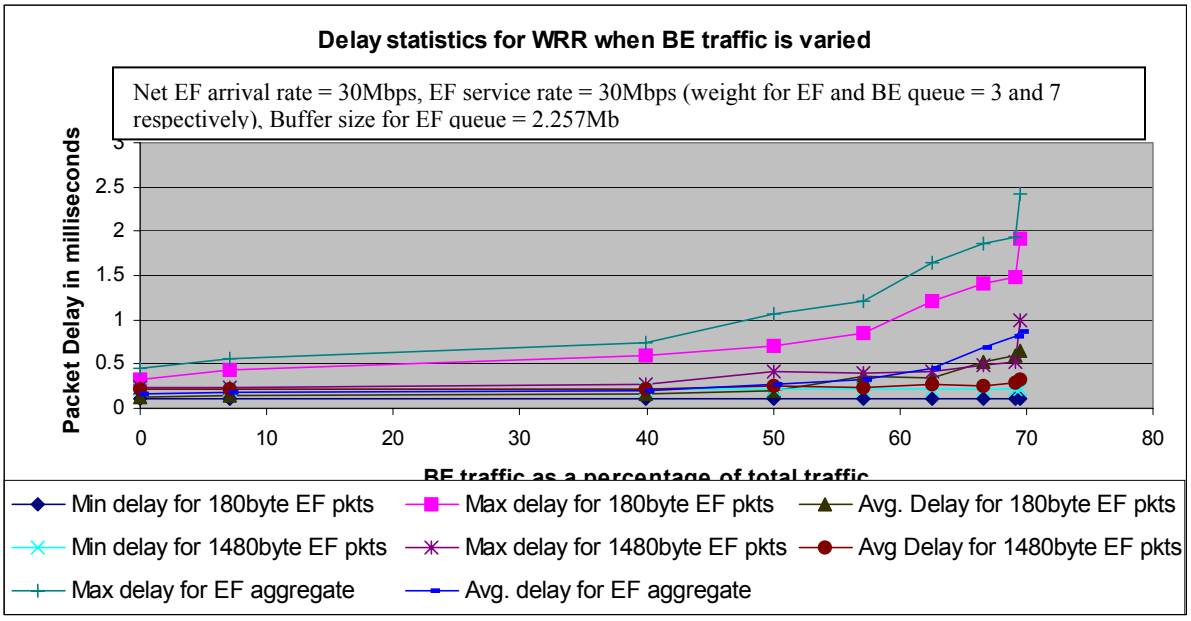

Figure 20: Delay report for WRR when the net BE traffic is varied.

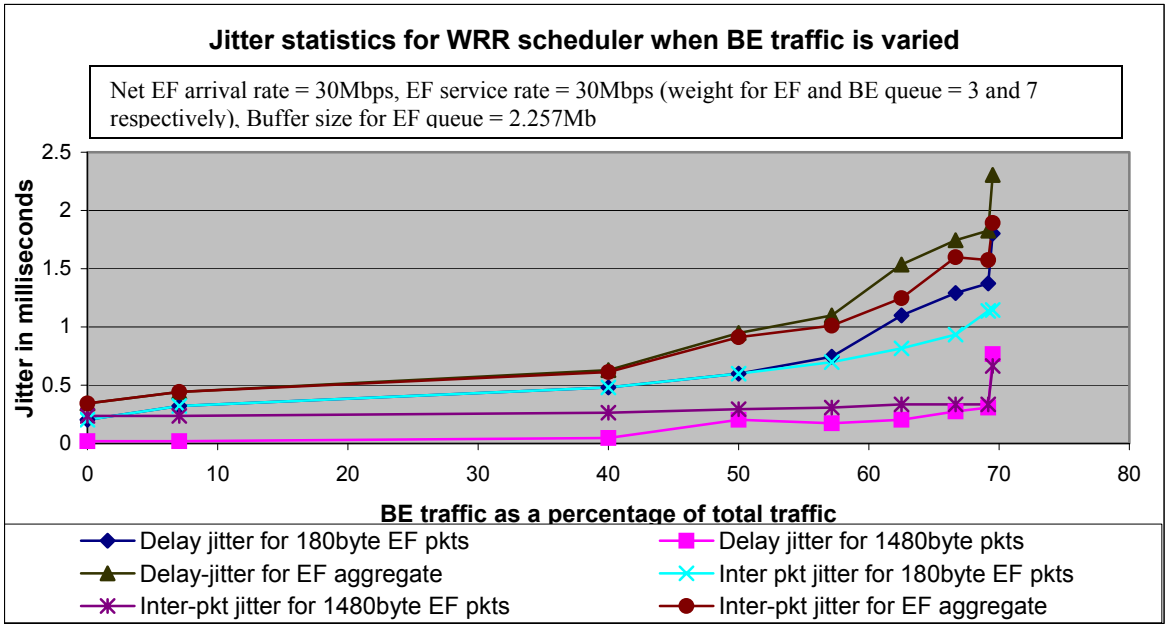

Figure 21: Jitter statistics for WRR when the net BE traffic is varied.

5.9.3. Conclusions for Experiment 4 on EF PHB

The results from Experiment 1 and Experiment 4 indicated that, generally the contribution to the queuing delays due to aggregation of EF micro-flows was less than that due to the increase in the amount of background best effort traffic. This observation was validated by a similar observation made in [13].

PQ Scheduler: It was observed for the PQ scheduler that with an increase in the BE traffic, the maximum per hop delay seen by the EF micro-flow with 180byte UDP packets and the EF aggregate as a whole followed an exponentially increasing pattern, with a sharp rise after the 50percent mark. However all the other QoS Metrics and mainly the average delay values for both the micro-flows considered and for the net EF aggregate were almost constant and very low, indicating that although very large values for the maximum per hop delay were noted, it was safe to conclude that the bulk of the traffic suffered very low per hop delays, and that it was possible to provide the 95 and possibly the 99 percentile delay guarantees in spite of variation in the other traffic in the network. For the jitter statistics it was noted that the delay jitter and inter-packet jitter values for the EF micro-flow with 1480byte UDP

packets showed a constant value, while for the EF micro-flow with 180byte UDP packets and the EF aggregate, the rise was exponential with a sharp rise after the 50 percent mark.

WRR Scheduler: It was observed for the WRR scheduler that apart from the maximum per hop packet delays which were lower than the PQ values, all the other delays and especially the average delays were higher than those observed for the PQ scheduler under similar conditions. This validated the observations made in Experiment 2 on the EF PHB where the 95 and 99 percentile delay values for the WRR were higher than those for the PQ scheduler. The jitter statistics did not follow the same trend however as the WRR scheduler showed lower values for delay jitter and inter-packet jitter than the PQ scheduler. Again the EF micro-flow with 1480byte UDP packets showed a constant value, while for the EF micro-flow with 180byte UDP packets and the EF aggregate, the rise was exponential with a sharp rise after the 50 percent mark.

To conclude, although the absolute maximum delay values for the PQ were higher than the WRR, the bulk of the traffic would see delays around the average value, which was smaller in the PQ than the WRR. As far as the worst-case delay jitter and inter-packet jitter were concerned, the WRR fared better during congestion than the PQ, but again these values were the worst-case values that depended upon the maximum delay values. The average delay jitter and inter-packet jitter values may not follow this same trend.

One of the things to note in Experiment 3 and 4 was that only 10.84percent of the packets in the EF aggregate belonged to EF micro-flows consisting of 1480byte UDP packets. The remaining 89.155 percent of the EF traffic comprised of packets belonging to EF micro-flows with 180byte UDP packets. Thus although the net traffic in bits per second was the same for the group of EF micro-flows with 180byte UDP packets and the group of micro-flows with 1480byte UDP packets, the EF aggregate followed the trend set by the micro-flows of 180byte UDP EF packets.

5.10. Experiment 5 on EF PHB

Motivation: On account of the vagueness and difficulty in implementing the original definition of the EF PHB, a more formal definition was necessary to characterize the EF PHB [7][8]. This experiment studied the values of E_a and E_p obtained from those equations that could further be used to characterize the EF service provided by service providers as figures of merit applicable to the schedulers.

Aim: As the merit factors E_a and E_p were referred to with reference to the EF configured rate at the DS node under consideration, the data sets collected for Experiment 3 on the EF PHB, where the EF configured rate of a DS node was varied, were used in this analysis. Results were noted only for the net EF aggregate as merit factors E_a and E_p apply to the EF aggregate as a whole.

5.10.1. Suggested Values of E_a and E_p

Table 13: Experiment 5 Results for E_a and E_p.

EF Configured Rate as a % of the output link	Priority Scheduler		WRR Scheduler	
	Figure of merit E_a in ms	Figure of merit E_p in ms	Figure of merit E_a in ms	Figure of merit E_p in ms
30%	4.403	4.403	1.492	1.492
30.3%	0.18	0.18	4.803	9.894
30.9%	0.181	0.181	2.311	2.311
31.5%	0.182	0.182	4.187	4.187
33%	0.185	0.185	2.359	2.359
36%	0.189	0.189	1.091	1.091
45%	0.197	0.197	0.755	0.755
60%	0.206	0.206	0.396	0.396

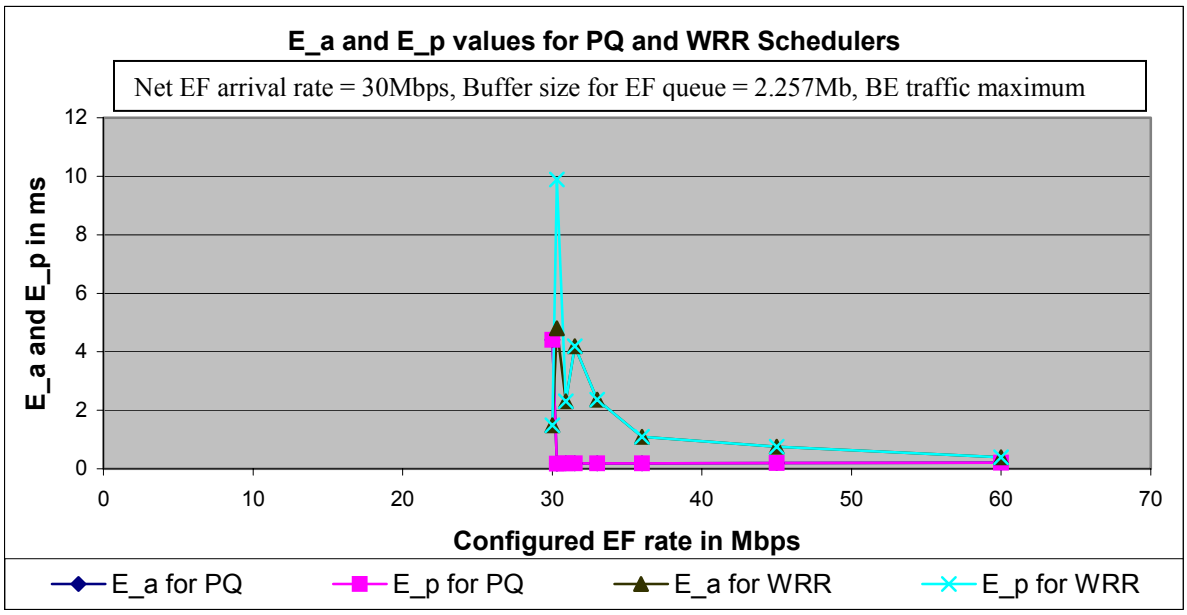

Figure 22: Calculated values of E_a and E_p for PQ and WRR.

5.10.2. Conclusions for Experiment 5 on EF PHB

At lower EF arrival to departure rate ratios, for both the schedulers, both the E_a and E_p values were high indicating packets being queued within the DS node. This indicated that there was a strong chance that at any given time within the EF queue, the same packet was used for the E_a and E_p calculations indicating FIFO service within the EF aggregate as mentioned in Section 3.6.3. in this thesis.

For the PQ scheduler, the E_a and E_p values showed very little variance and were equal to each other at an arrival to departure rate ratio of 1:1.01. E_a is influenced by the inability to preempt a MTU sized non EF packet that has just begun transmission when an EF packet arrives at the DS node and E_p is influenced by the number of interfaces [7].

As the arrival to departure rate ratio was increased for the WRR scheduler from 1:1.1, the E_a and E_p values were equal and did not stabilize to a value like the PQ scheduler but reduced exponentially as the EF service rate was increased. There was also the oscillatory behavior observed in the E_a and E_p values for the WRR as observed in Experiment 3 on

the EF PHB. This behavior could be attributed to the basic property of the WRR scheduler where integer multiple queue weights were assigned to the EF and BE queues, and to the fact that at certain EF arrival to service rate ratios, the weights could be very large. The E_a values for WRR scheduler will be higher than that observed for the PQ scheduler as now packets from other queues will also have to be serviced in round robin fashion even when EF packets are awaiting service in the DS node. Thus the E_a and E_p could be successfully used as metrics to determine appropriate EF arrival to service rate ratios for which the EF compliant DS nodes could be configured.

A service provider could provide a single worst case E_a and E_p value like 0.206ms noted in the Table 13 for the PQ scheduler measured at an EF arrival to departure rate ratio of 1:1.01 and greater, and it would be applicable to any EF departure rate that was at least 1.01 times the net EF arrival rate. This is possible because of the slight variation in the E_a and E_p values. The same does not hold for the WRR scheduler as the E_a and E_p values varied over a wide range in accordance with the configured EF departure rate.

It was also observed for the WRR scheduler, that even with an EF arrival to departure rate ratio of 1:3[14], it couldn't match the E_a and E_p values of the PQ scheduler configured for an EF arrival to departure rate ratio of 1:1.01. Only when the EF queue was assigned 99% of the total output link[15], was the WRR scheduler able to match the E_a and E_p values of the PQ scheduler, but this configuration setting of the WRR scheduler cannot be used commercially as we would in effect be over provisioning the entire link to support only EF traffic.

5.11. Experiment 6 on DB PHB

Motivation: For the delay bound PHB, the definition of the DB behavior is characterized by the net input rate of DB traffic and the output delay variation [9]. The DB PHB is governed by the equation 6 mentioned in Section 3.7. of this thesis. This experiment aimed at determining the possible values of the score S which is a constant and a characteristic of the DB compliant DS node at the input rate R, when the net DB arrival rate is varied.

[14] , This reading is not noted in Table 13 of this thesis.
[15] , This reading is not noted in Table 13 of this thesis.

Aim: As the Score S is a characteristic of the net DB arrival rate, this analysis was made from the data collected in Experiment 1 on the EF PHB, and results were noted only for the net DB aggregate as the score S applied to the DB aggregate as a whole. The delay jitter was used for the calculation of score S. The MTU of the outgoing interface was considered to be 1500bytes, the MTU of an Ethernet interface.

5.11.1. Suggested Values of Score S

Table 14: Experiment 6 Results for possible values for Score, S.

DB Input rate in Mbps	PQ Scheduler		WRR scheduler	
	Delay jitter in ms	Score S	Delay jitter in ms	Score S
1.5	0.118	0.118	0.307	0.307
3	0.223	0.446	0.410	0.82
7.5	0.237	1.185	0.597	2.985
15	0.294	2.94	0.838	8.38
28.5	0.443	8.417	1.818	34.542
30	7.987	159.74	2.684	53.68

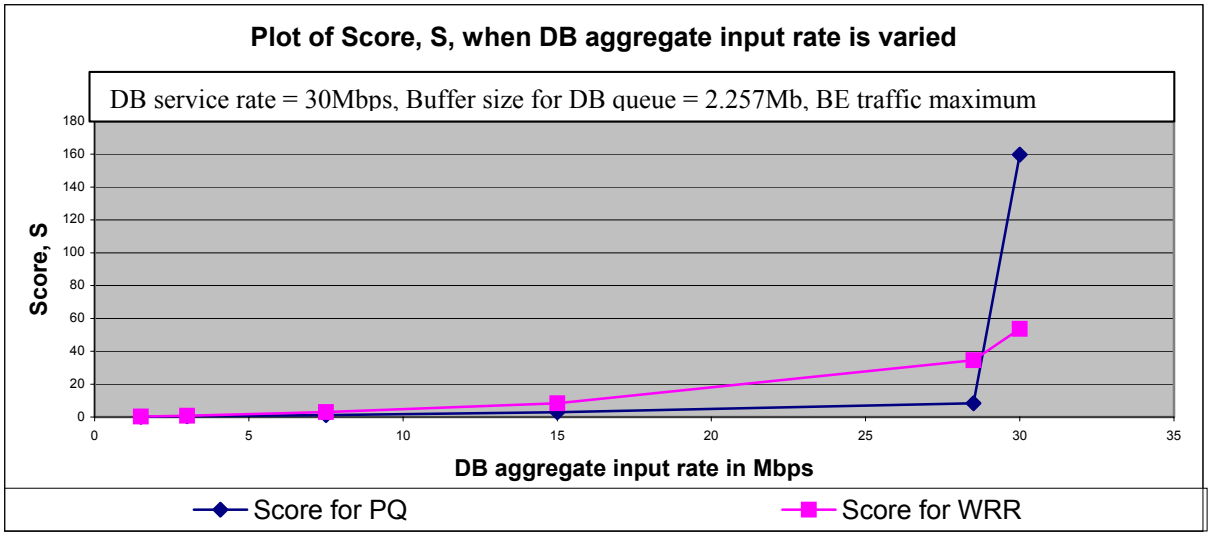

Figure 23: Score S for DB PHB, when DB aggregate input rate is varied.

5.11.2. Conclusions for Experiment 6 on DB PHB

According to recommendations and from observing the equation governing the DB PHB, it was clear that a smaller value of score S for a given DB input rate was desired as there was a direct relationship between the score S and the delay jitter which could be equated to the

queuing delay that we were trying to minimize in the EF PHB as also noted in the earlier experiments [9]. For the EF PHB and specifically the DB PHB, bounding the variation in the maximum and minimum delay that a packet could face was important, as a number of applications that can tolerate a limited delay could be built on this PHB. If a bound was given to the maximum delay, the receiving application could arrange for an appropriately sized buffer to accommodate the worst case delay, so that the receiving application delayed access to the buffered data till a time equal to this worst case delay.

From observing the results above it was clear the a PQ scheduler provided a lower score than a WRR scheduler, provided the net arrival rate of the input DB traffic was less than the DB configured rate at the DS node. This fact was also noted in Experiment 1 on the EF PHB proving that the score S is a characteristic of the scheduler used to implement the DB PHB.

5.12. Case Study of the EF PDB

As a priority queue scheduler from the experiments on the EF PHB had assured better performance in almost all cases at the PHB level, it was used in combination with a token bucket policer to study the EF PDB.

5.12.1. Object of the EF PDB Case Study

Two main experiments were conducted for this study, each consisting of 5 scenarios. The primary purpose of these experiments was to prove that EF traffic can indeed be isolated from BE traffic, and also to study the minimum, maximum and average end-to-end delays experienced by EF packets and the computation of the worst-case inter-packet jitter [21]. A common service provisioning policy was defined between each of the three DS domains thus forming a DS region. Hence there was no need for remarking codepoints at intermediate DS domain boundaries. Traffic conditioning between neighboring DS domains was also not necessary due to the same codepoint to PHB mapping being used in all the DS domains.

5.12.2. Network Topology for the EF PDB Case Study

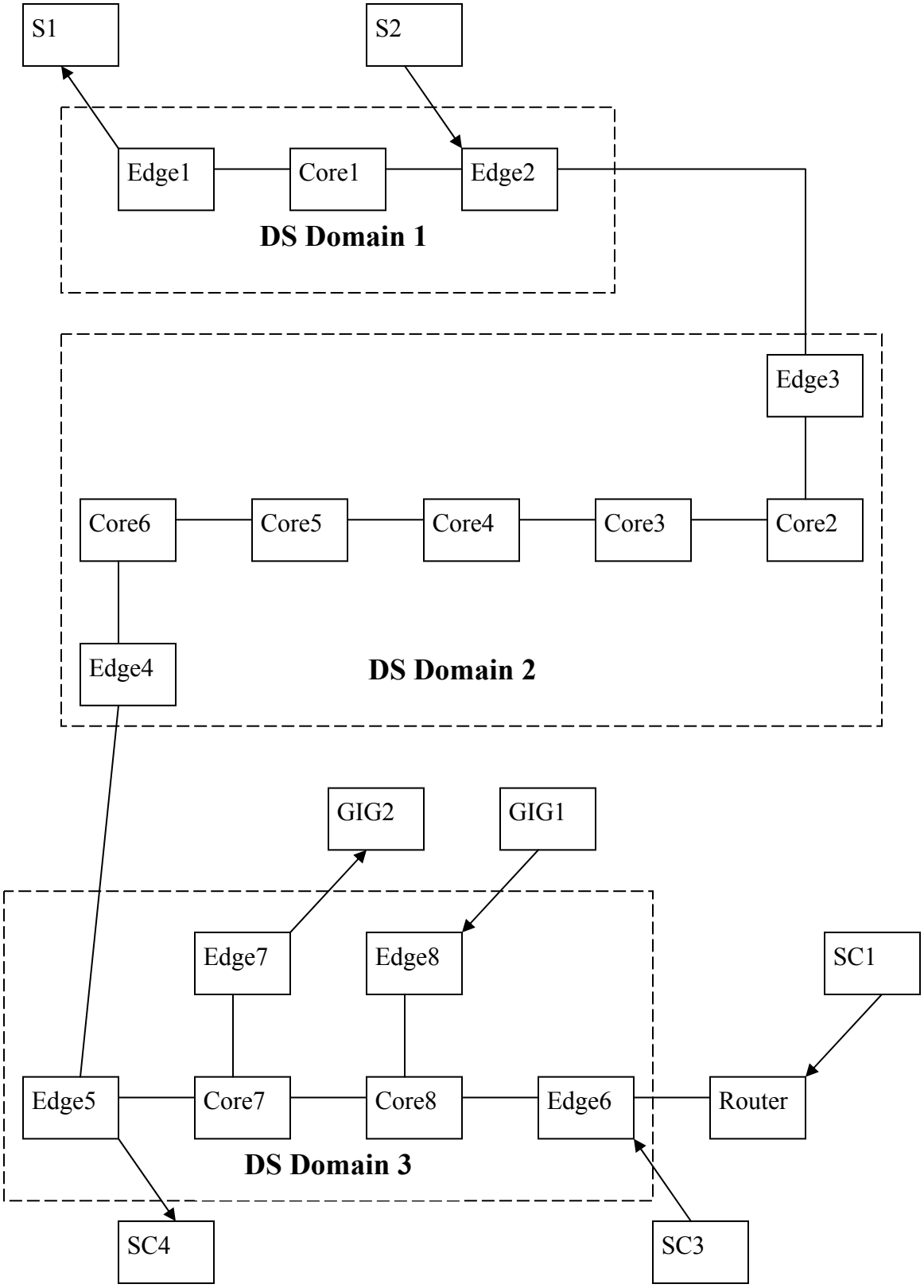

Figure 24: Network topology used for studying EF PDB Case Study

5.12.3. Experiment 1 on EF PDB

Traffic marked for Expedited Forwarding was sent from a host SC1 connected to remote DS domain 3 to a host S1 connected to DS domain 1 at the rate of 24Mbps. With each successive scenario, the networks were slowly saturated with best effort cross-traffic, from absolutely no congestion, to very heavy congestion.

5.12.4. Experiment 2 on EF PDB

The above experiment was repeated with traffic marked for Expedited Forwarding being replaced by best effort traffic to provide a comparative study of the extent of protection prioritized traffic might obtain in a congested network.

5.12.5. EF PDB Case Study Specifics

Simulation time: 10seconds for each scenario.
UDP Packet Size: 1480bytes (Explained in Section 5.3.2. of this thesis)
Queues & Schedulers used in DS Networks: Two queues were implemented in every DS node; one for EF traffic and one for best effort traffic and a strict priority queue scheduler were used.
Configured EF departure rate: 25Mbps in every node (The ratio of EF arrival to departure rate was maintained at 1:1.041 which was in excess of the minimal ratio of 1:1.01 as observed from Experiment 3 and 5 on the EF PHB for the priority queue scheduler)
Link speeds within DS domain 1 and 3: 100Mbps
Link speeds from DS domain 3 to DS domain 2: 45Mbps
Link speed within DS domain 2: 2.4Gbps
Link speed from DS domain 2 to DS domain1: 100Mbps

5.12.6. Details of Experiment 1 and 2 on the EF PDB

For Experiment 1, traffic was sent from SC1 to S1: 24Mbps for 10seconds marked EF with the following congestion points:

Table 15: Congestion Points for EF PDB Case Study.

Scenario	SC3 to SC4 in Mbps	Gig1 to Gig2 in Mbps	Edge router 5 in DS domain 3 to Edge router 2 in DS domain 1 in Mbps	S2 to S1 in Mbps
1	0	0	0	0
2	25	25	0	25
3	70	70	17.75	70
4	80	80	22.25	80
5	95	95	29	95

Experiment 2 was a repeat of Experiment 1 with EF traffic from SC1 to S1 replaced by BE traffic.

5.12.7. Tools Used for Analysis

Each simulation script invoked the Xgraph tool to graphically display the bandwidth used by traffic flows over the simulation time and the Network Animator tool (NAM), which provided a view of the entire topology over the simulation time. Packet flows, queues, packet drops could be observed in the network, thereby helping to gain a better understanding of the problem zones within the network.

AWK scripts were run on the trace files generated by each simulation, to determine the minimum, maximum and average end-to-end packet delays of the concerned flow.

5.12.8. Record of Results from EF PDB Case Study

5.12.8.1. Results for Packet Loss

Results obtained from the field trial[16] as compared to those obtained from the simulation for packet loss for Experiment 1 where traffic marked for Expedited Forwarding and Experiment 2 where equivalent best effort traffic was transmitted between DS domain 1 and DS domain

[16] , The Differentiated Services test bed team at North Carolina State University performed the field trial, and only the observations by that team on packet loss and end-to-end delays were noted below for validating the observations made through the simulations.

2 was noted in the Table 16. In each successive scenario cross traffic was more than that in the previous scenario.

Table 16: Packet Loss for EF PDB Case Study.

Scenarios	Experiment 1		Experiment 2	
	Packet loss from Field trial as a percentage of total monitored flow	Packet loss from Simulations as a percentage of total monitored flow	Packet loss from Field trial as a percentage of total monitored flow	Packet loss from Simulations as a percentage of total monitored flow
1	0	0%	Less than 1%	0%
2	0	0%	Less than 1 %	0%
3	Less than 1%	0%	40%	39.202%
4	Less than 1 %	0%	49%	48.889%
5	Less than 1 %	0%	63%	64.039%

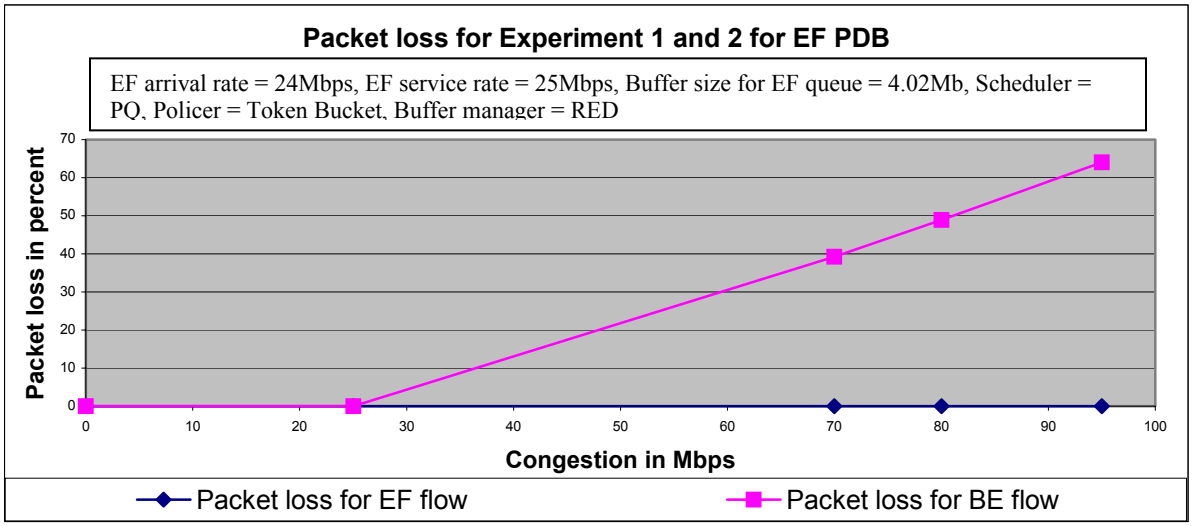

Figure 25: Packet loss for the EF PDB Case Study.

5.12.8.1.1. Observations on Packet Loss

The results obtained from the field trial matched those obtained from the simulations. It was observed that traffic marked for Expedited Forwarding was indeed protected from the rest of the traffic in terms of packet loss. It was seen that equivalent best effort traffic on the same path experienced significant packet loss as the non DS network could no more protect the

particular BE traffic flow being monitored. A linear relationship was observed between the packet loss observed in the equivalent best effort traffic and the congestion points. This linear relationship is a characteristic of this network topology under the conditions specified. Depending upon the network topology, SLAs and cross traffic, this relationship may vary.

5.12.8.2. Results for End-to-end Delay and Jitter Statistics

Table 17: End-to-end Delay and Jitter Statistics for Experiment 1 for EF PDB Case Study.

Scenario	Minimum delay in ms	Maximum delay in ms	Average delay in ms	End-to-end Delay Jitter in ms	End-to-end Inter-packet jitter in ms
1	26.207	26.289	26.287	0.082	26.207
2	26.207	26.406	26.321	0.199	26.207
3	26.207	26.758	26.442	0.551	26.207
4	26.288	26.774	26.491	0.486	26.326
5	26.288	26.777	26.501	0.488	26.485

Table 18: End-to-end Delay and Jitter Statistics for Experiment 2 for EF PDB Case Study

Scenario	Minimum delay in ms	Maximum delay in ms	Average Delay in ms	End to End Delay Jitter in ms	End-to-End Inter-packet jitter in ms
1	26.207	26.289	26.287	0.082	26.207
2	26.207	26.524	26.351	0.317	26.207
3	26.207	28.793	27.594	2.586	26.207
4	26.288	29.333	27.775	3.045	26.444
5	26.320	29.554	27.071	3.234	26.513

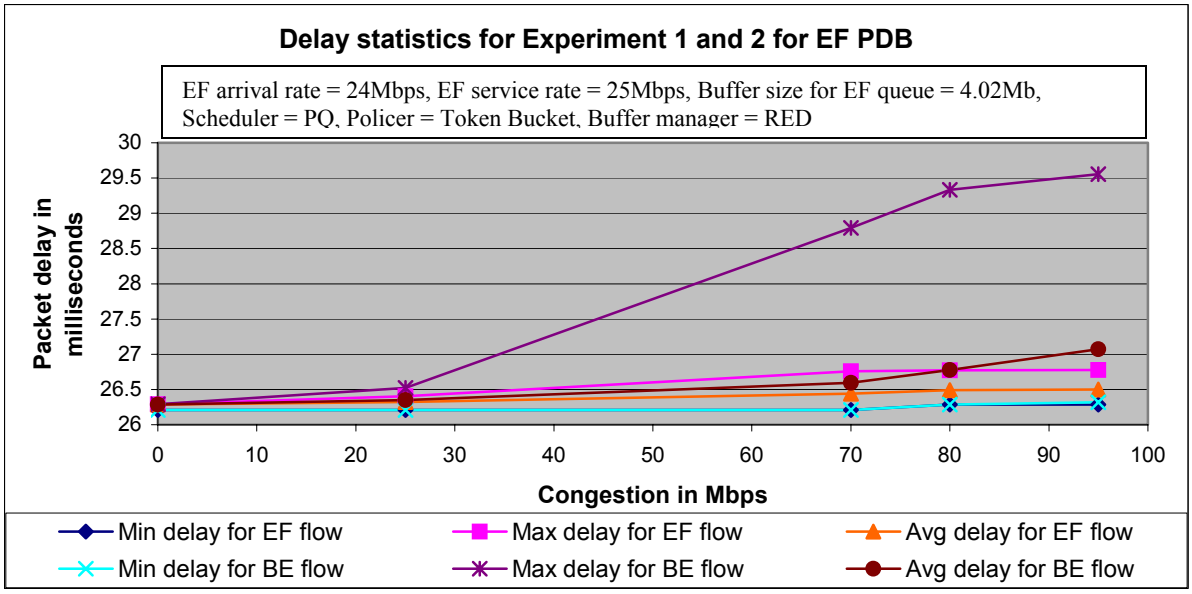

Figure 26: Delay statistics for EF PDB Case Study.

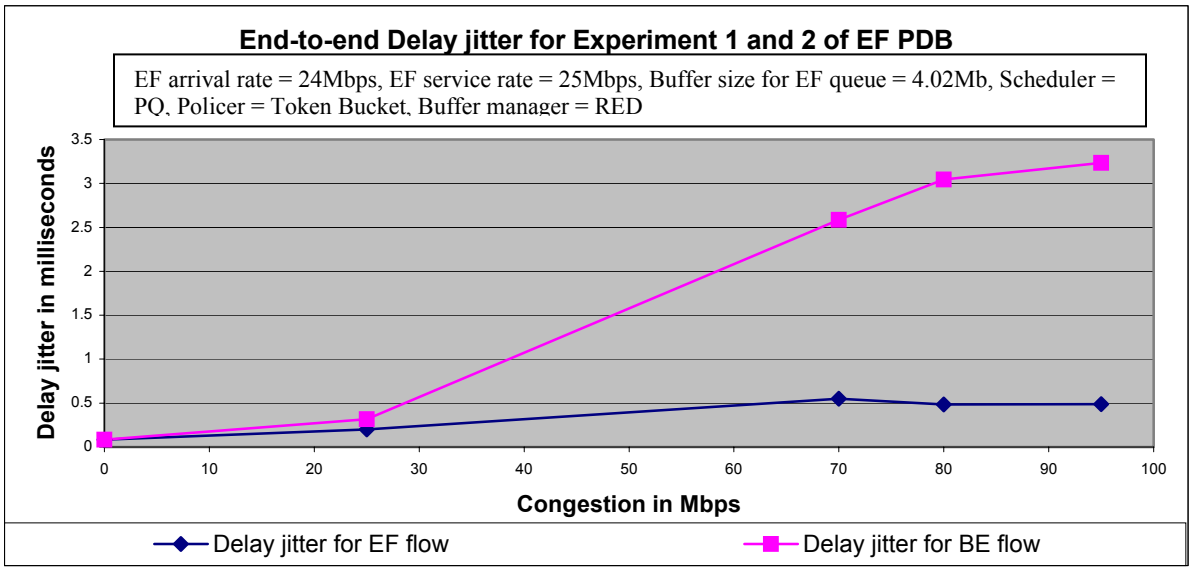

Figure 27: Delay jitter statistics for EF PDB Case Study

5.12.8.2.1. Observations on End-to-end Delay and Jitter Statistics

When the best effort traffic congestion was varied from minimum to maximum, the variation in the maximum end-to-end delay for the EF flow, was only 0.488ms indicating that less than 0.5ms of time was consumed in queuing delays in the DS networks. But for the equivalent

BE flow it was 3.255ms which was about 6 to 7 times more than that seen for the EF traffic indicating that we were successful in reducing the queuing delay within this particular topology by a factor of 6 to 7. Similarly the average delay varied only 0.214ms for the EF flow while it varied 0.784 for the equivalent BE flow, which was more than three times that of the EF flow. It was observed that the end-to end delay jitter was bounded below 0.5ms for the EF flow, but for the equivalent BE flow, the end-to end delay jitter increased proportional to the congestion in the network. The values of the end-to-end delays obtained from the simulations were validated with the help of observations supplied by the Differentiated Services test bed team that conducted the field tests.

5.12.9. Conclusions of the EF PDB Case Study

We were successfully able to protect the EF traffic across multiple DS domains and limit the queuing delays within the topology. Evaluating this network topology specifically, we were also able to limit the average end-to-end delay well below the 150ms limit above which delays can be perceived by the human ear. However coder and packetization delay at the source and the delay added by de-jitter buffer at the destination were not accounted for in the Tables 17 and 18 in this thesis. If we assumed the algorithmic delay and coder delay for the G.711 protocol to be 0.125ms and 20ms respectively, and de-jitter buffer delay to be 1.5 times the worst-case inter-packet jitter (26.485ms) noted in the simulations above, which amounted to 39.72ms, the delay budget would still be within the 150ms limit imposed by the sensitivity of the human ear, making this implementation suitable for video and voice over IP applications. However as the one-way delay was greater than 25ms, echoing would create a problem in case of VoIP and echo cancellation would be required. In case of streaming media applications, the inter-packet jitter measurements can aid in designing the size of the de-jitter buffer at end destinations depending upon the amount of data that needs to be buffered before starting playback.

6. Conclusions

The EF PHB was simulated with the help of a token bucket policer and a) PQ scheduler and b) WRR scheduler. Both implementations were studied for per hop delays, and jitter. On a whole, the PQ scheduler fared better than the WRR scheduler in providing the EF service.

Experiment 1 on the EF PHB was used to evaluate the QoS provided to EF traffic based on the amount of EF traffic it was competing with. This experiment also realized the impact of aggregating numerous EF micro-flows, on individual EF micro-flows. Metrics like standard deviation and a calculated threshold were evaluated to determine their suitability in predicting the 95 and 99-percentile delay guarantees that service providers could provide to customers in Experiment 2 on the EF PHB. It was seen that 95 percentile delay guarantees could be provided with a simple constant while a more complex constant was required to provide 99 percentile delay guarantees. It was also easy to determine these guarantees for the PQ than the WRR scheduler. Experiment 3 on the EF PHB was used to evaluate the amount of over provisioning necessary for the EF PHB in order to give the EF traffic appropriate forwarding treatment. Very low QoS metric values were realized for the PQ scheduler with an arrival to departure rate ratio of 1:1.01 for EF traffic. The WRR scheduler matched these values only when the EF configured rate was set to 99 percent of the total output interface rate. Experiment 4 on the EF PHB proved the impact of BE cross traffic on the EF PHB for both the schedulers was more severe than that due to micro-flow aggregation. Experiment 5 on the EF PHB was used to determine the values of the figures of merit E_a and E_p used to quantify the EF PHB. It was seen that the E_a values were always equal to the corresponding E_p values indicating FIFO treatment. The E_a and E_p values stabilized for the PQ scheduler with a very small variation, but this was not applicable to the WRR scheduler, which showed an exponentially decreasing trend in the values as the EF configured rate at the DS node was increased. Experiment 6 on the EF PHB was used to evaluate the possible values of score S, which is used to characterize DS nodes supporting the Delay Bound PHB. The EF PDB case study aided in providing a comparative evaluation of the QoS provided to EF and BE traffic across multiple DS domains.

One thing observed in all these experiments for both the schedulers was that the EF micro-flow consisting of 180byte UDP packet micro-flows experienced a larger per hop packet delay as compared to the 1480byte UDP packet micro-flows. The reason being that even if a single 180byte UDP packet was queued behind a single 1480byte UDP packet, the larger packet was almost 9 times the size of the smaller packet, and this was equivalent to the smaller packet being queued behind 9 small EF packets. If the smaller packet was queued behind more than one large sized packet then the delay experienced by it was worse by a factor 9 times the number of large packets before it in the queue. It can be concluded that packets the size of the path MTU can be assured a stable QoS with more confidence than packets that are a fraction of the path MTU. The QoS metrics stabilize to a constant value as packet sizes approach the path MTU.

One solution to decrease the queuing delay experienced by smaller packets is by creating multiple EF sub-queues within the EF queue based on EF packet sizes and implementing 2 levels of scheduling within each DS node. The first level of scheduling can operate on the EF, BE and AF aggregates. The second level of scheduling can operate on two or more separate EF sub-queues for small size packets and large size packets. The EF sub-queue of smaller packet sizes can be given higher priority than the EF sub-queue of larger packet sizes. This would lead to non FIFO treatment to EF packets within a DS node. Appropriate SLAs would need to be established to limit the number of small sized EF packets permitted within the DS domain. Additional research would need to be done to ensure that the EF sub-queue consisting of large size packets isn't starved by the higher priority EF sub-queue else we will no more be able to provide EF guarantees to large size EF packets. Having two levels of scheduling will also add to internal delays within each DS node, and hence its effectiveness will have to be determined only after thorough study.

From this study, the PQ scheduler can be used as a benchmark to compare other schedulers for implementing the EF PHB, knowing the limitations of the PQ scheduler, namely a sharp deterioration in the QoS metrics when the net EF arrival rate equaled the EF configured rate.

6.1. Areas of Future Research

Some of the areas of future research are listed below. Work is in progress in a number of these areas.

a) The use of multiple scheduling levels as mentioned in the Section 6. of this thesis, can be evaluated to determine whether it would be practically feasible to implement multiple EF sub-queues within the EF queue for the EF PHB based on packet sizes.

b) Study of the effect of variation in buffer sizes on the QoS metrics as it affects EF micro-flows and the EF aggregate as a whole and the determination of appropriate buffer sizes to implement the EF PDB reliably without over provisioning or under provisioning.

c) Study of other scheduling mechanisms to provide the EF PHB in DS nodes according to the new definition of the EF PHB and the merit factors, to determine if any specific advantages can be gained from them that would benefit the implementation of the EF PHB or the DB PHB [7][8].

d) Standardization of implementations for the EF PDB.

e) The simultaneous implementation of the Expedited Forwarding, Assured Forwarding (AF) and Best Effort (BE) PHBs in a DS node, the detailed study of inter-relation between them and the extent to which QoS metrics of a particular PHB are affected by other PHBs under various worst case situations. This study can be extended to a PDB level.

f) Benchmarking of Service Level Agreements and the development of reliable services based on the EF PHB and these agreements.

g) Development and characterization of new PHBs based on the needs of next generation Internet applications.

h) Appropriate mapping of the EF codepoint to layer 2 QoS mechanisms like Multi Protocol Label Switching (MPLS) to provide a streamlined EF service [14][15][16].

7. References

[1] Nichols, K., Blake, S., Baker, F., Black, D., "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.

[2] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Weiss, W., "An Architecture for Differentiated Services", RFC 2475, December 1988.

[3] Heinanen, J., Baker, F., Weiss, W., Wroclawski, J., "Assured Forwarding PHB Group", RFC 2597, June 1999.

[4] Jacobson, V., Poduri, K., "An Expedited Forwarding PHB", RFC 2598, June 1999.

[5] Black, D., "Differentiated Services and Tunnels", RFC 2983, October 2000.

[6] Nihols, K., Carpenter, B., "Definition of Differentiated Services Per Domain Behaviors and Rules for their Specification", RFC 3086, April 201.

[7] Davie, B., Bennett, J., Benson, K., Le Boudec, J., Courtney, W., Davari, S., Firoiu, V., Stiliadis, D., "An Expedited Forwarding PHB (Per-Hop Behavior)", RFC 3246, March 2002.

[8] Charny, A., Bennett, J., Benson, K., Le Boudec, J., Courtney, W., Davari, S., Firoiu, V., Kalmanek, C., Ramakrishnan, K., "Supplemental Information for the New Definition of the EF PHB (Expedited Forwarding Per-Hop Behavior)", RFC 3247, March 2002.

[9] Armitage, G., Carpenter, B., Casati, A., Crowcroft, J., Halpern, J., Kumar, B., Schnizlein, J., "A Delay Bound Alternative Revision of RFC 2598", RFC 3248, March 2002.

[10] Keshav, S., "An Engineering Approach to Computer Networking", 1997

[11] "Voice Over IP - Per Call Bandwidth Consumption", < (online) http://www.cisco.com/warp/public/788/pkt-voice-general/bwidth_consume.html>

[12] Bennett, J., Benson, K., Charny, A., Courtney, W., LeBoudec, J., "Delay Jitter Bounds and Packet Scale Rate Guarantee for Expedited Forwarding", Proceedings of Infocom, April 2001, <(online) http://www.ieee-infocom.org/2001/paper/101.pdf >, 2001.

[13] Ferrari, T., Chimento, P., "A Measurement-based Analysis of the Expedited Forwarding PHB Mechanisms", Proceedings of IWQoS 2000, Pittsburgh June 2000, February 2002.

[14] Brim, S., Faucheur, F., "Per Hop Behavior Identification Codes", RFC 2836, May 2000.

[15] Black, D., Brim, S., Carpenter, B., Faucheur, F., "Per hop Behavior Identification Codes", RFC 3140, June 2001.

[16] Black. D., "Differentiated Services and Tunnels", RFC 2983, October 2000.

[17] Braden, R., Clark, D., Shenker, S., "Integrated Services In the Internet Architecture: an Overview", RFC 1633, June 1994.

[18] Yang, C., Fu, C., Tu, Y., "Enterprise Traffic with a Differentiated Service Mechanism", International Journal of Network Management, Volume 11, Issue 2, March – April 2001 pp 113 – 128, 2001.

[19] Fall, K., Varadhan, K., "The ns Manual (formerly ns Notes and Documentation)", The VINT Project, A collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC.

[20] Pieda, P., Ethridge, J., Baines, M., Shallwani, F., "A Network Simulator, Differentiated Services Implementation", Open IP, Nortel Networks, 2000.

[21] Jacobson, V., Nichols, K., Poduri, K., "The Virtual Wire' Per-Domain Behavior", Internet Draft, <draft-ietf-diffserv-pdb-vw-00.txt>, January 2001

[22] Dwekat, Z., "Construction and Evaluation of a Service Level Agreement Test-Bed.", A thesis submitted at North Carolina State University in partial fulfillment of the requirements for the Degree of Master of Science in Computer Networking, 2001.

[23] Floyd, S., "Random Early Detection Gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, Volume 1, Issue 4, August 1993.

[24] Awduche, D., Agogbua, J., O'Dell, M., McManus, J., "Requirements for Traffic Engineering over MPLS", RFC 2702, September 1999.

[25] Rosen, E., Vishwanathan, A., Callon, R., "Multi Protocol Label Switching Architecture", RFC 3031, January 2001.

[26] Rosen, E., Tappan, D., Fedorkow, G., Rekhter, Y., Farinacci, D., Li, T., Conta, A., "MPLS Label Stack Encoding", RFC 3032, January 2001.

[27] Chimento, P., "Standard Token Bucket Terminology", < (online) http://qbone.internet2.edu/bb/Traffic.pdf>, May 2000.

[28] Shah, S., "Bringing Comprehensive Quality of Service Capabilities in Next-Generation Networks", White Paper, < (online) http://e-www.motorola.com/collateral/QOSM-WP.pdf>, October 2001.

[29] Hersent, O., Gurle, D., Petit, J., "IP Telephony Packet-based multimedia communications systems", 2000.

[30] "Test 25.2.4 - Bit Error Rate Verification", (online) < http://www.iol.unh.edu/testsuites/fe/tests/pmd/test25.2.4.html>, January 1999.

[31] Stevens, R., "TCP/IP Illustrated, Volume 1. The Protocols", February 1994.

[33] Baker, F., "IP Quality of Service: Better Than Best Effort", Business Communications Review, pp. 28-31, March 1998.

[34] "Design for Nonstop E-Commerce", Packet Magazine Archives, July 1999.

[35] Bernet, Y., Yavatkar, R., Ford, P., Baker, F., Zhang, L., Nichols, K., Speer, M., "A Framework for Use of RSVP with Diff-serv Networks", < draft-ietf-diffserv-rsvp-01.txt>, 2001

[36] Sikora, J., Teitelbaum, B., "Differentiated Services for Internet2", < (online) www.internet2.edu/qos/may98Workshop/html/diffserv.html >, May 1998.

[37] Greene, B., Marshall, A., "Optimal Network Partitioning for QoS Protocols", < (online) http://www.ee.qub.ac.uk/dsp/research/telecomms/publications/papers/ben_teletraffic00.pdf >, 2000.

[38] "Internet Protocol", RFC 791, September 1981

8. Appendix

Appendix A

Histograms for PQ for single EF flow of 180byte packets for Experiment 2:

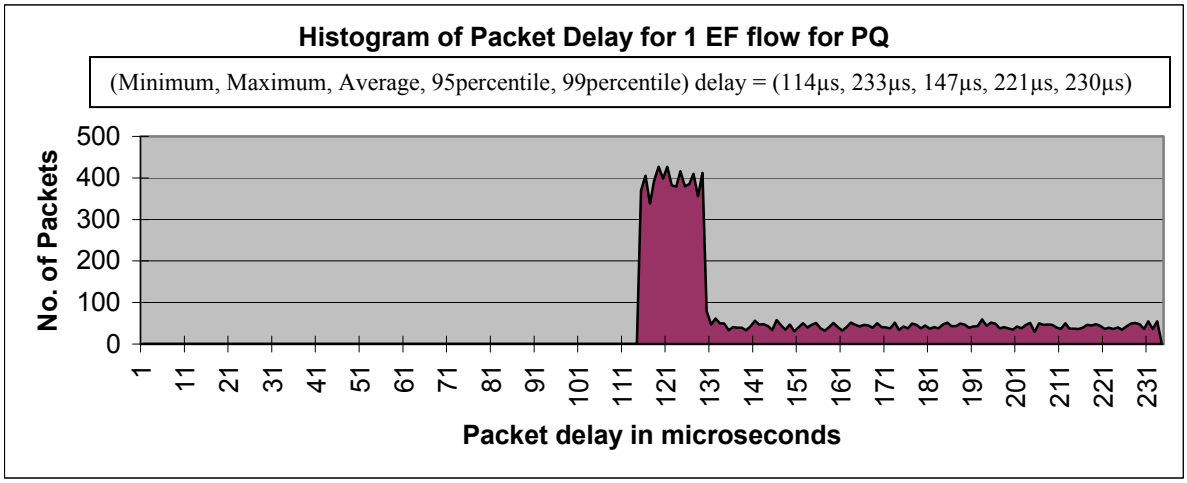

Figure 28: Histogram 1

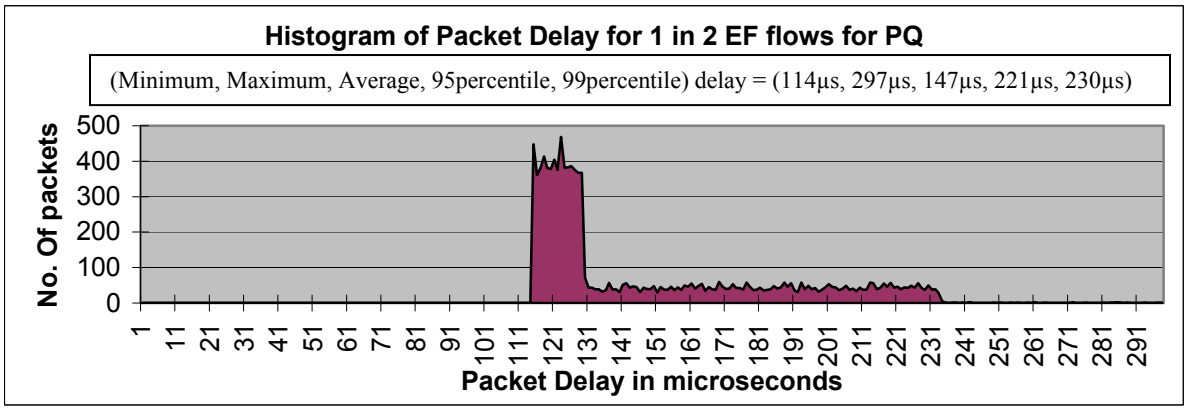

Figure 29: Histogram 2

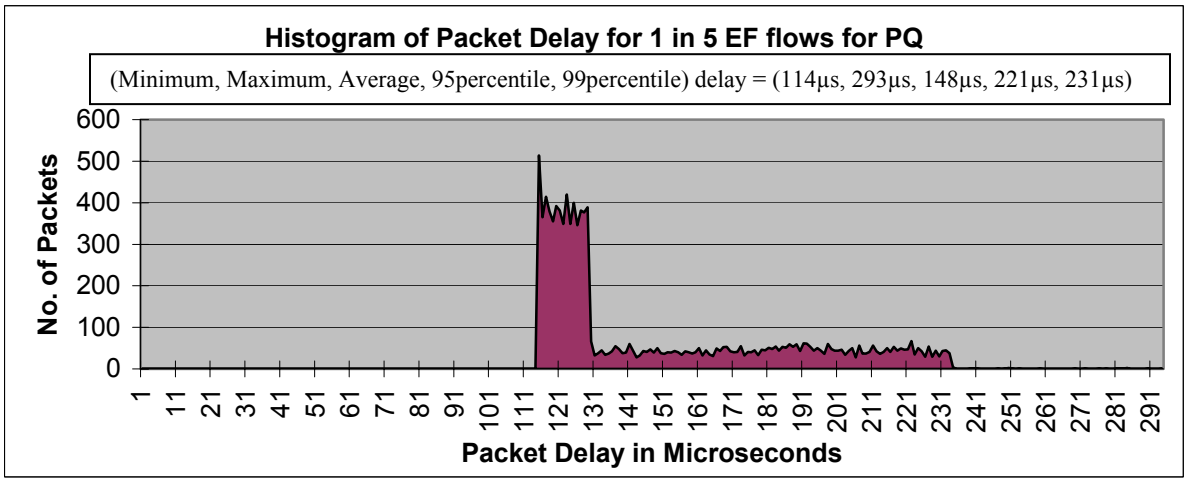

Figure 30: Histogram 3

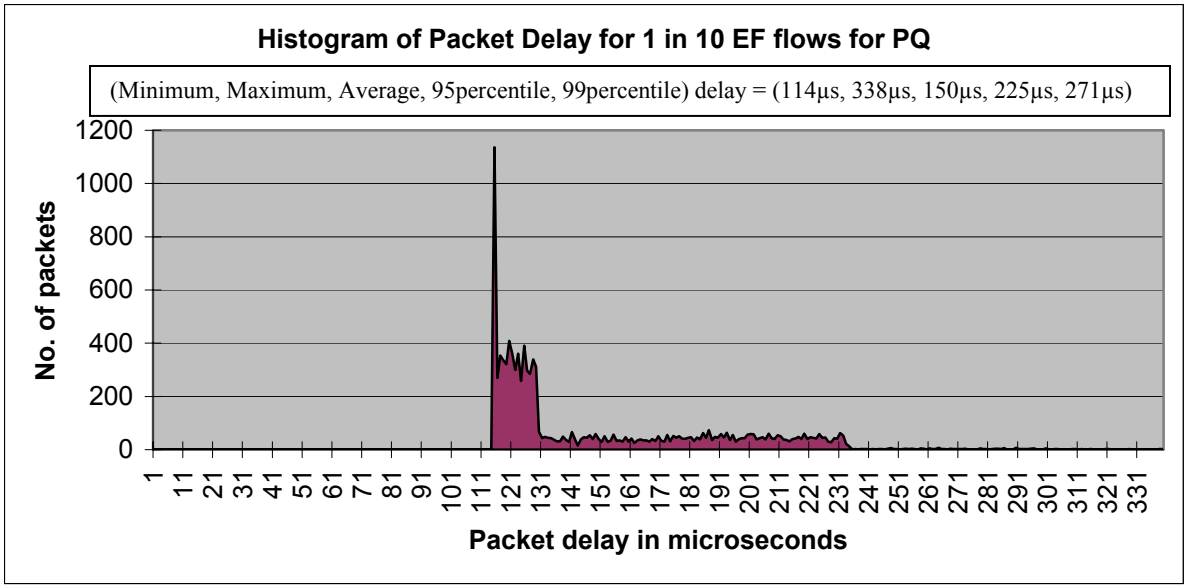

Figure 31: Histogram 4

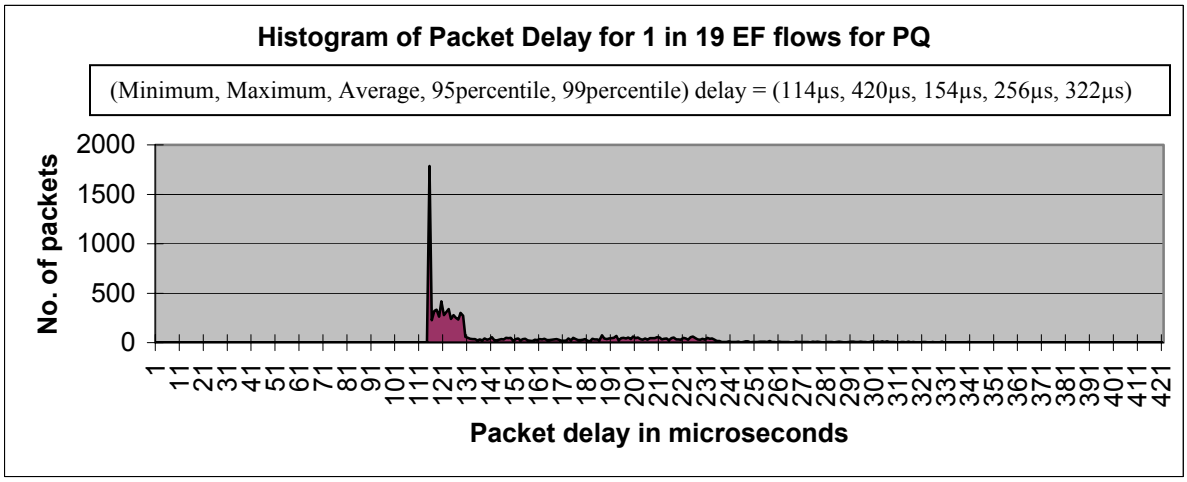

Figure 32: Histogram 5

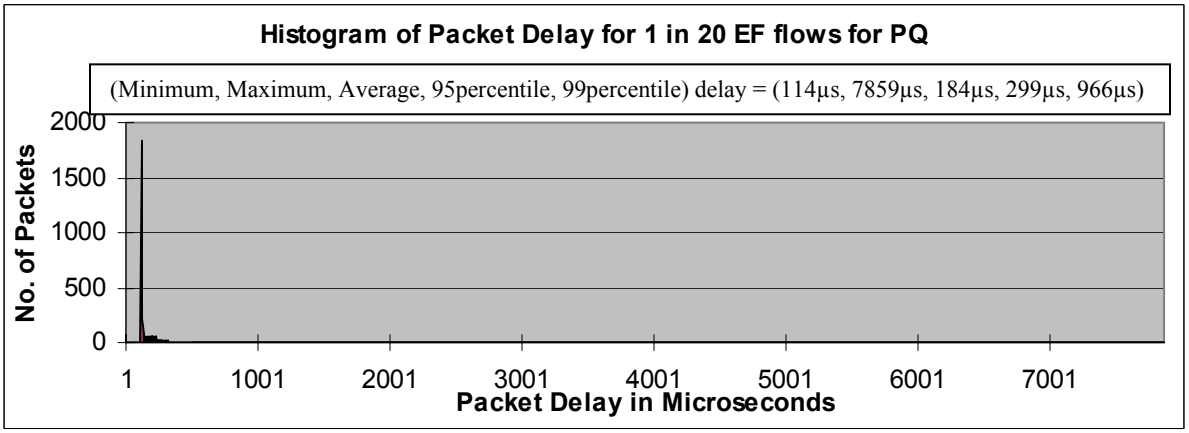

Figure 33: Histogram 6

Histograms for PQ for single EF flow of 1480byte packets for Experiment 2:

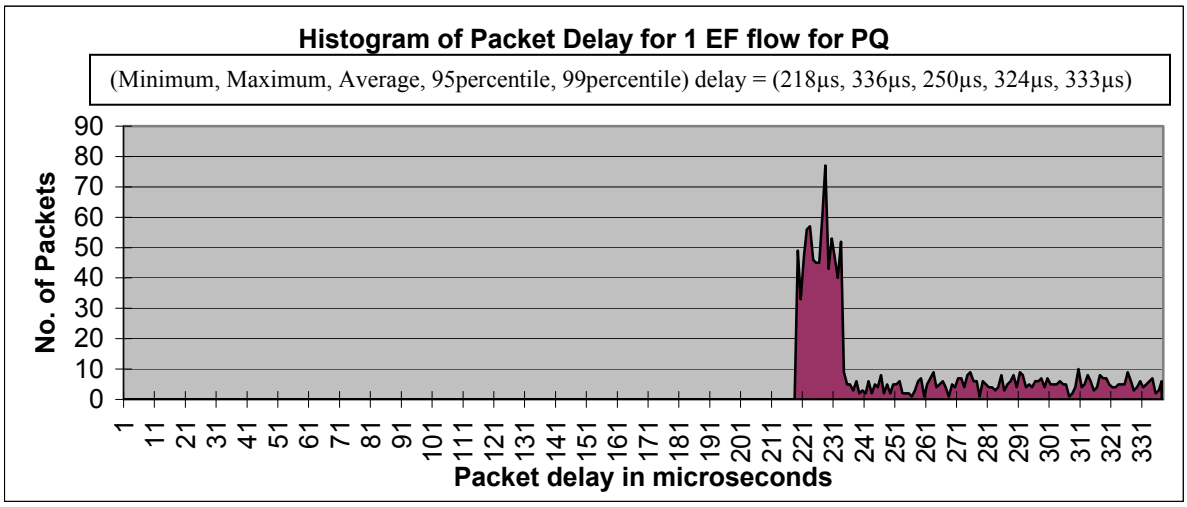

Figure 34: Histogram 7

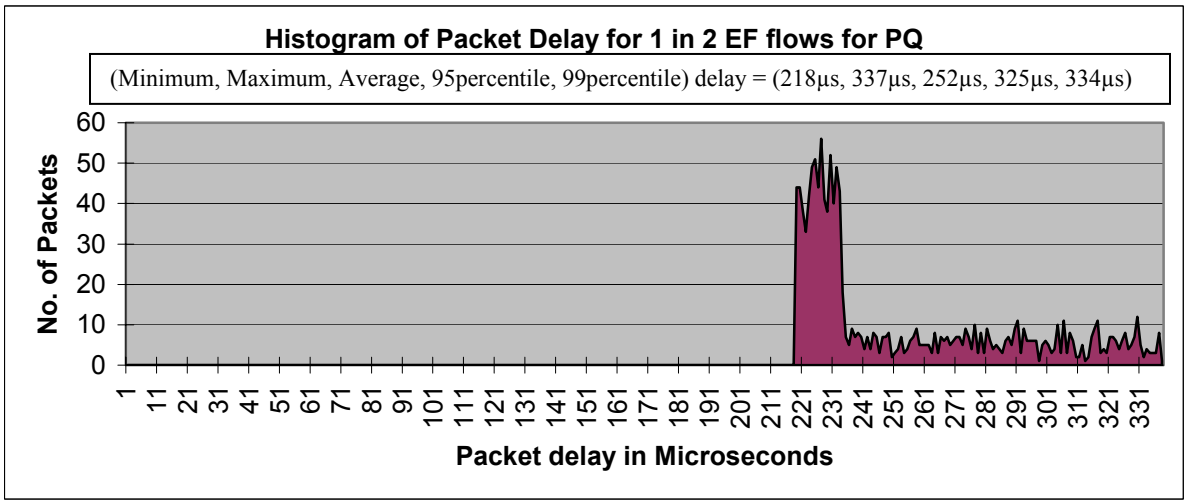

Figure 35: Histogram 8

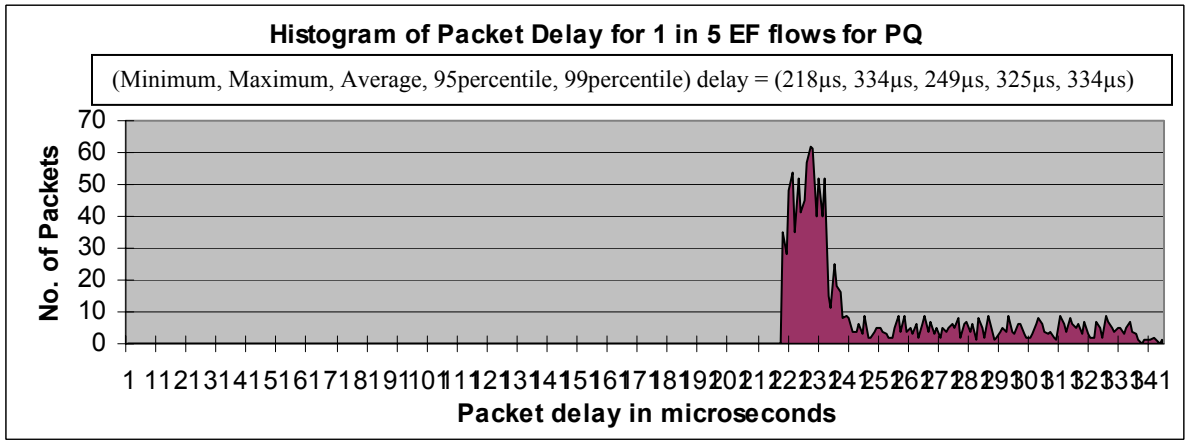

Figure 36: Histogram 9

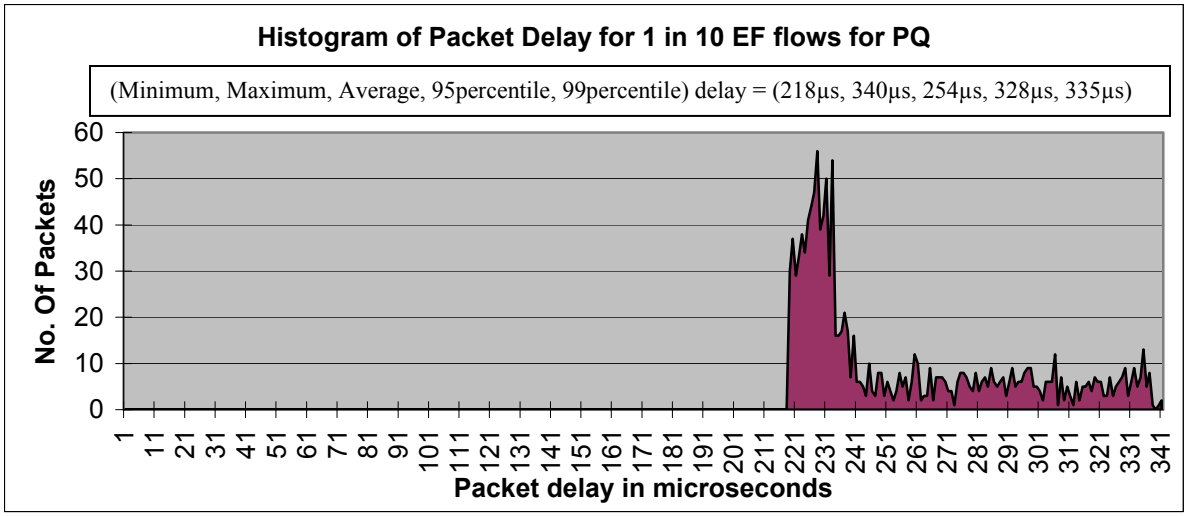

Figure 37: Histogram 10

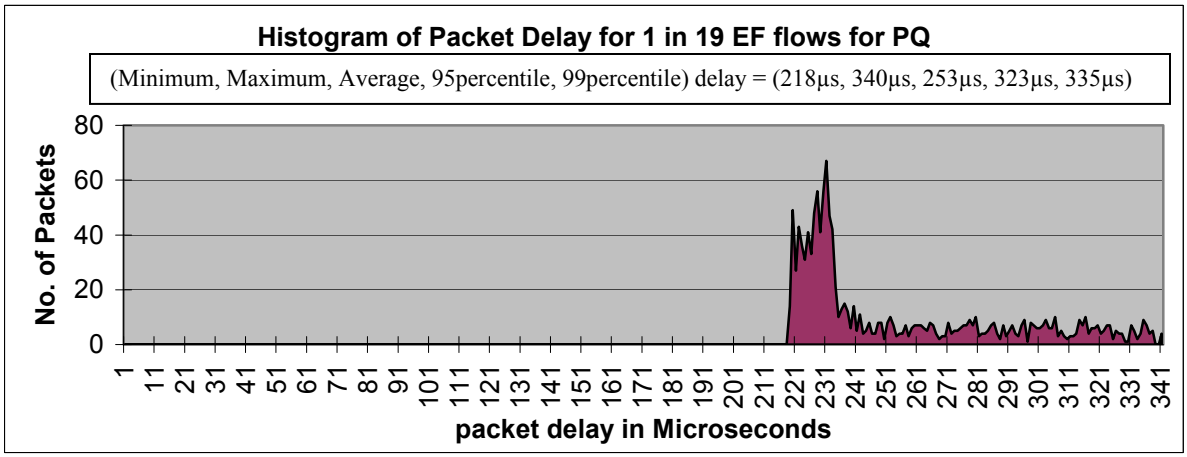

Figure 38: Histogram 11

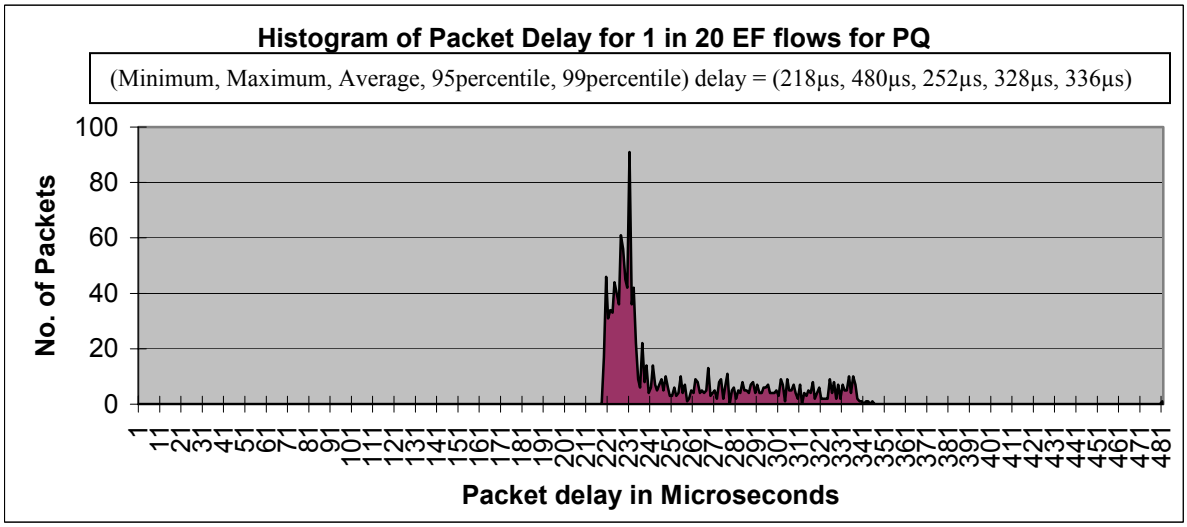

Figure 39: Histogram 12

Histograms for PQ for net EF aggregate for Experiment 2:

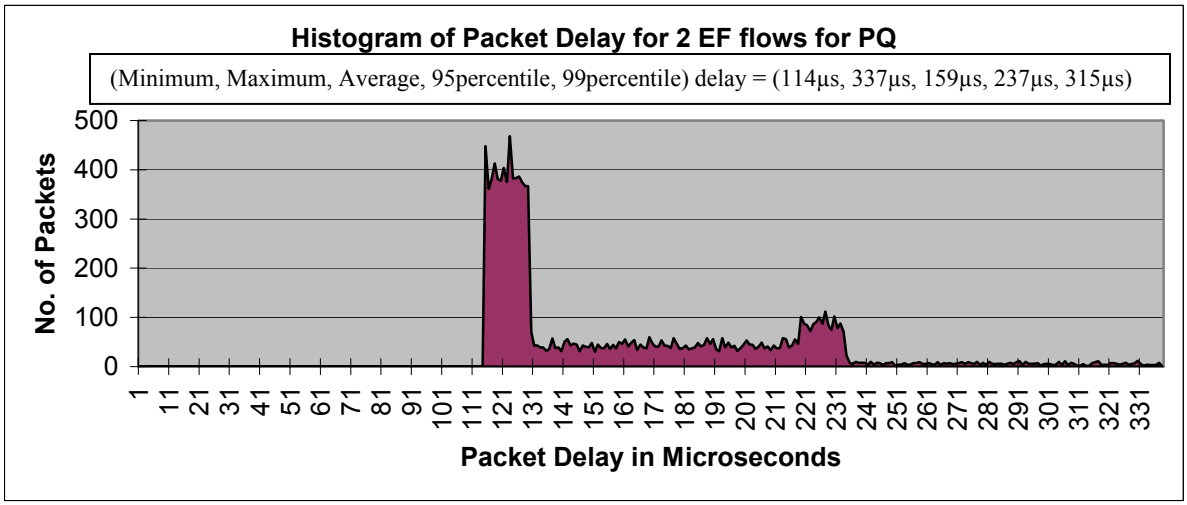

Figure 40: Histogram 13

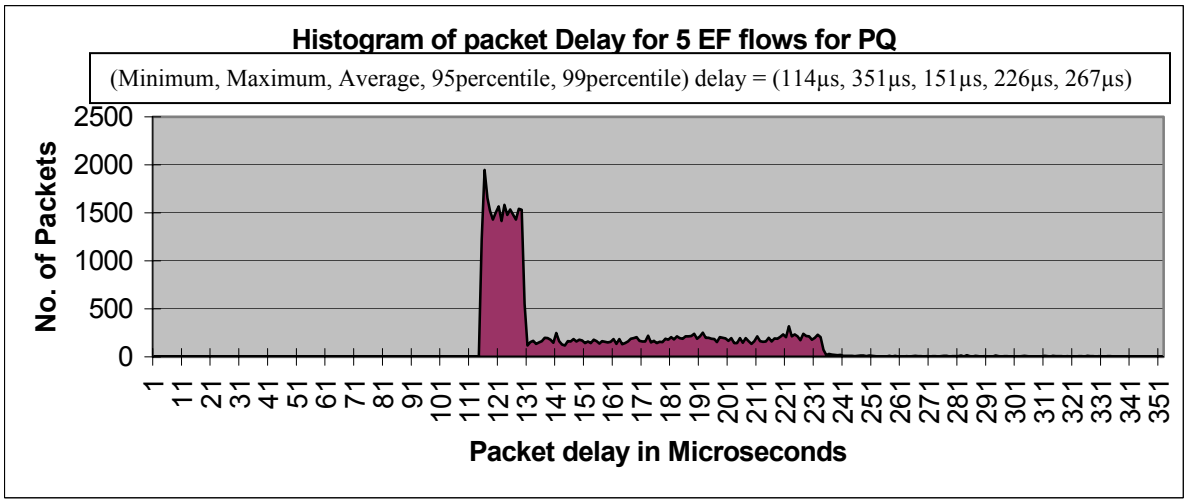

Figure 41: Histogram 14

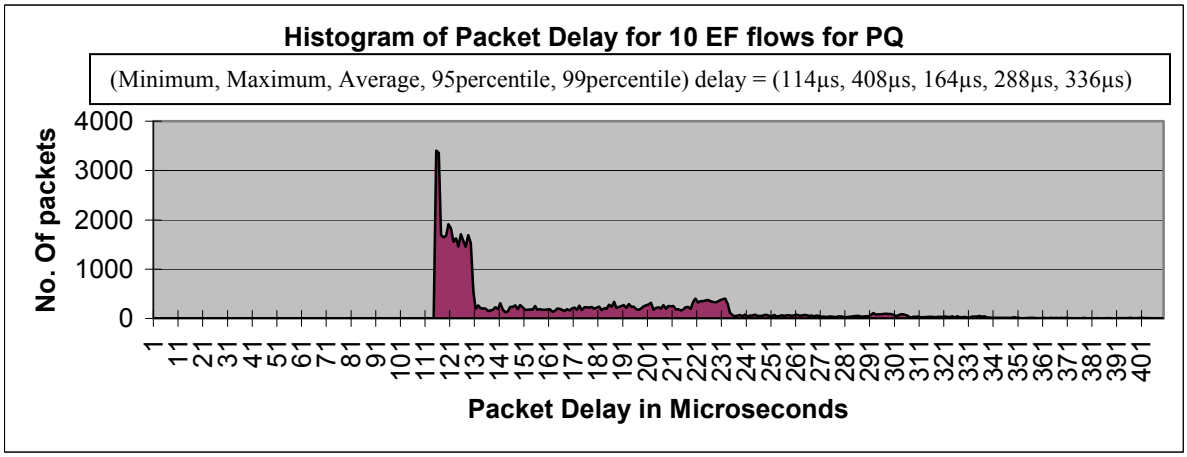

Figure 42: Histogram 15

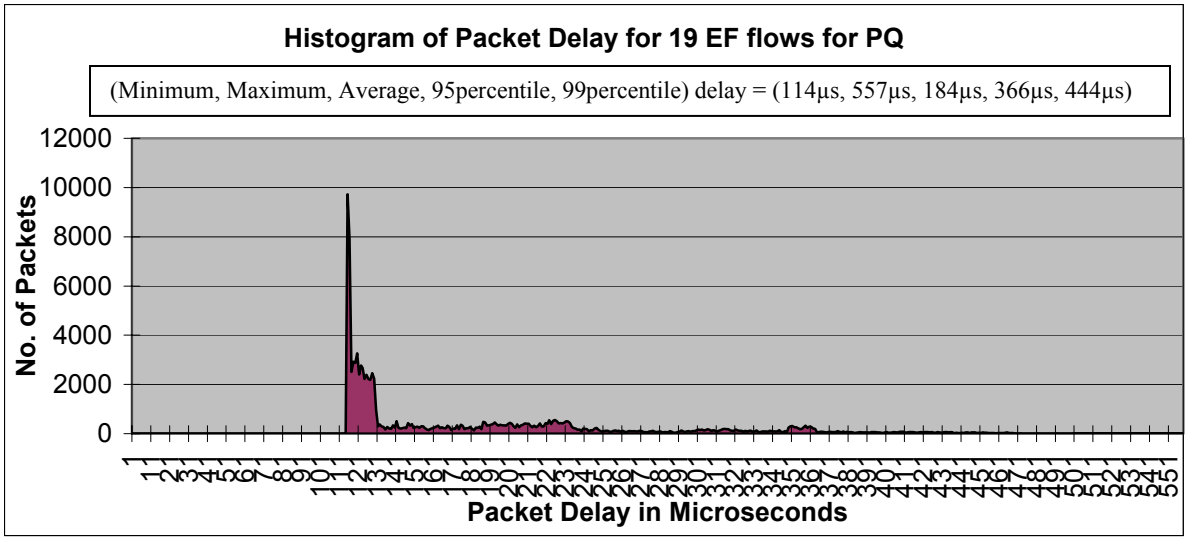

Figure 43: Histogram 16

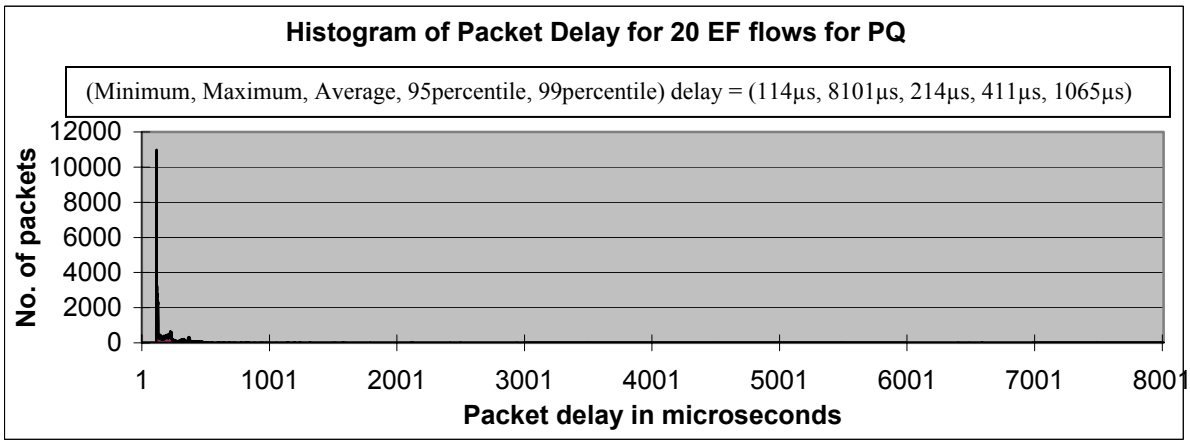

Figure 44: Histogram 17

Histograms for WRR for single EF flow of 180byte packets for Experiment 2:

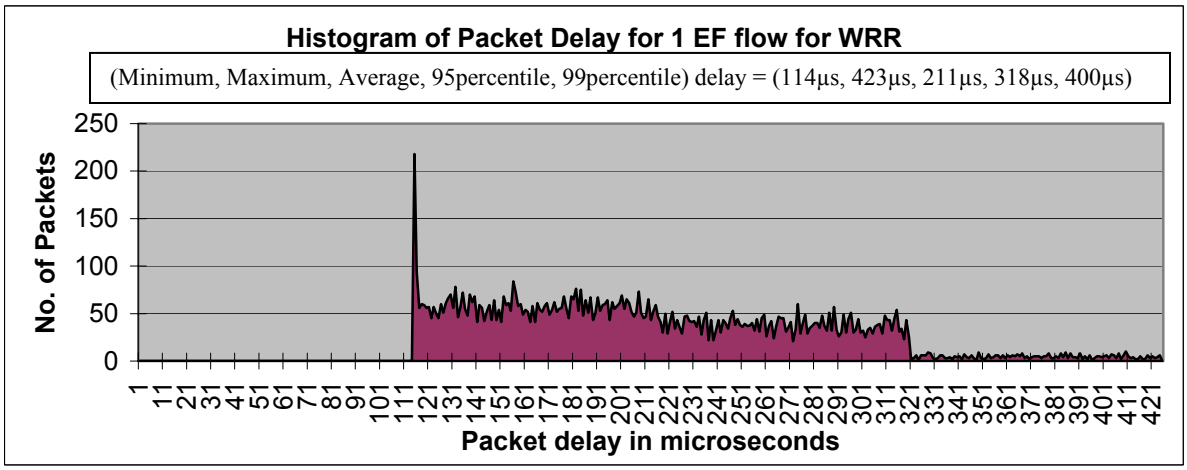

Figure 45: Histogram 18

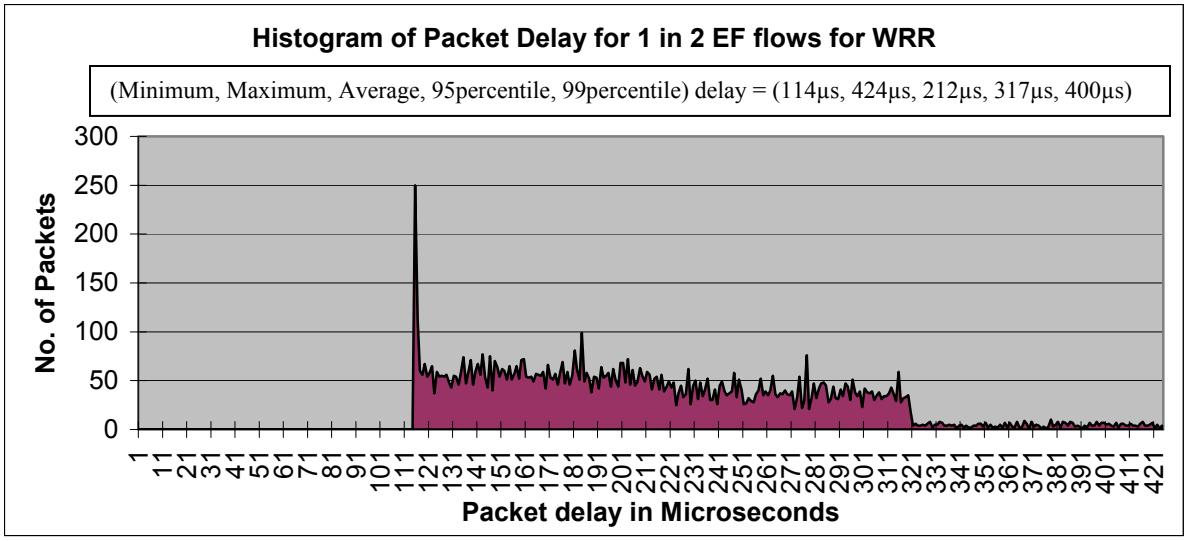

Figure 46: Histogram 19

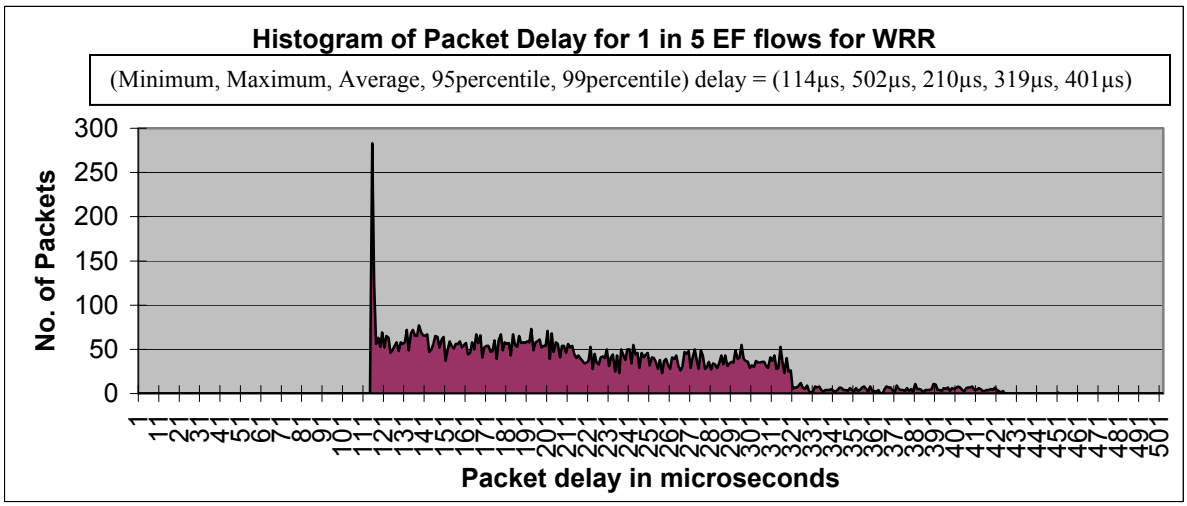

Figure 47: Histogram 20

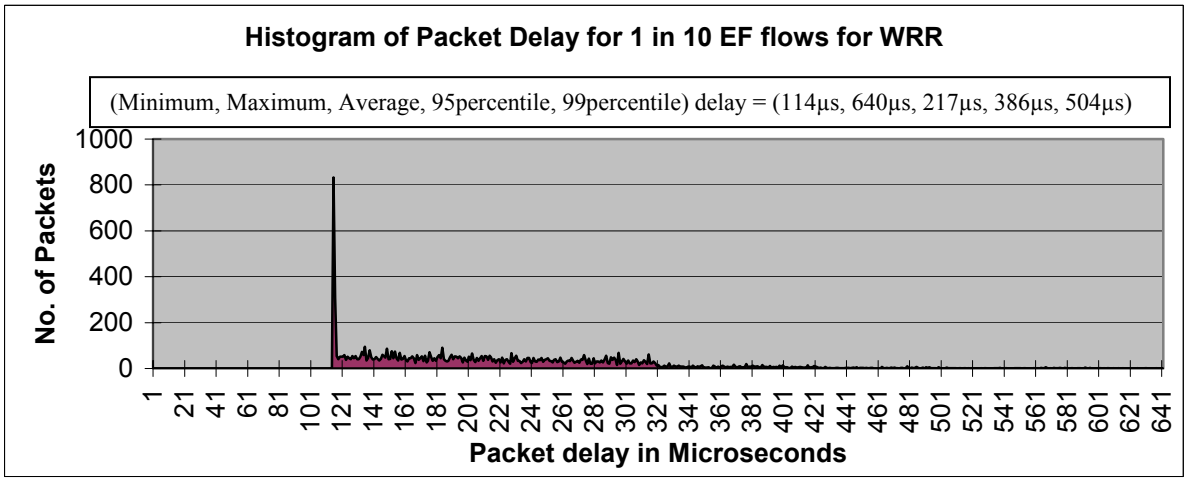

Figure 48: Histogram 21

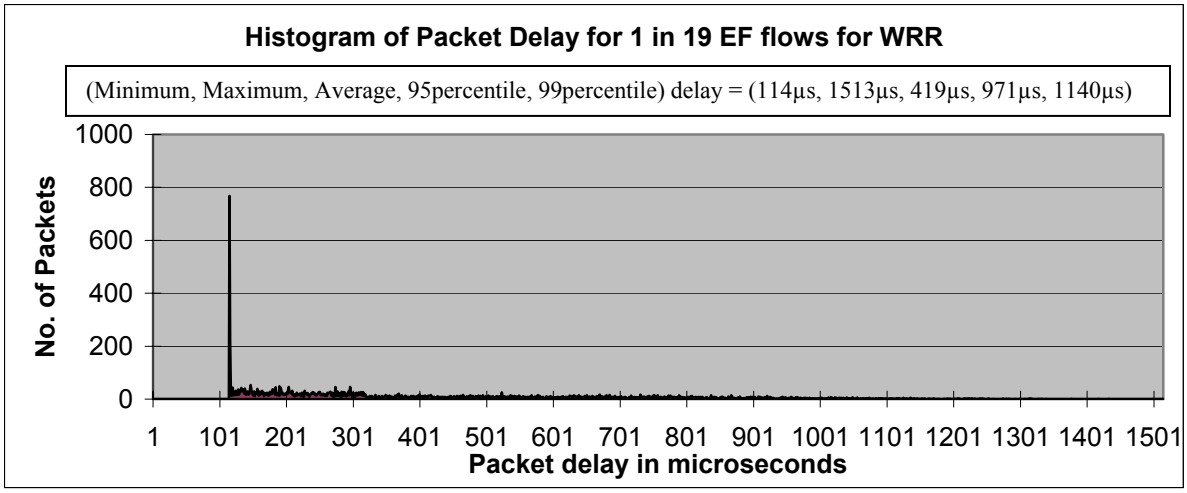

Figure 49: Histogram 22

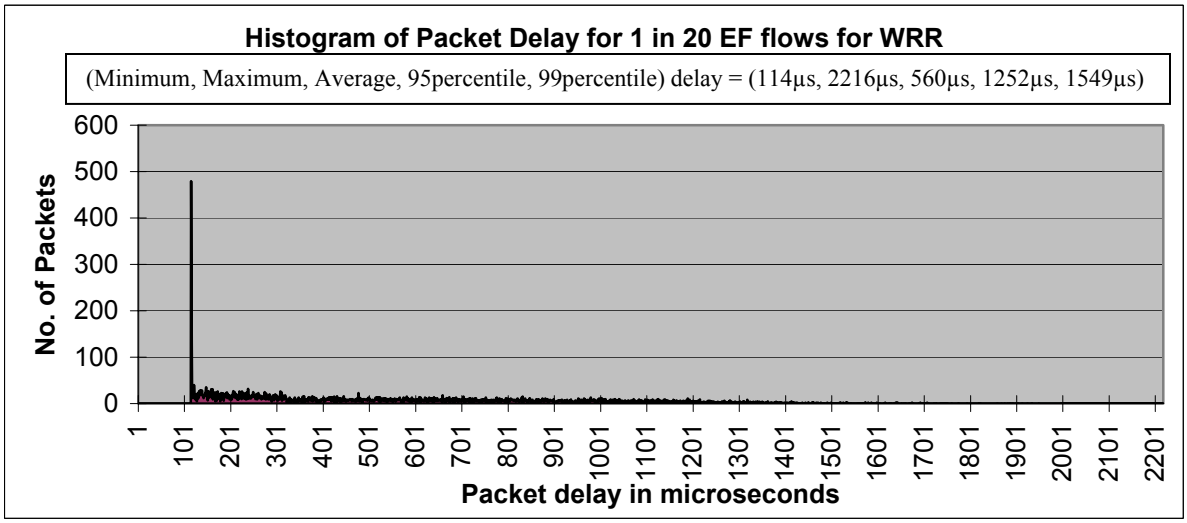

Figure 50: Histogram 23

Histograms for WRR for single EF flow of 1480byte packets for Experiment 2:

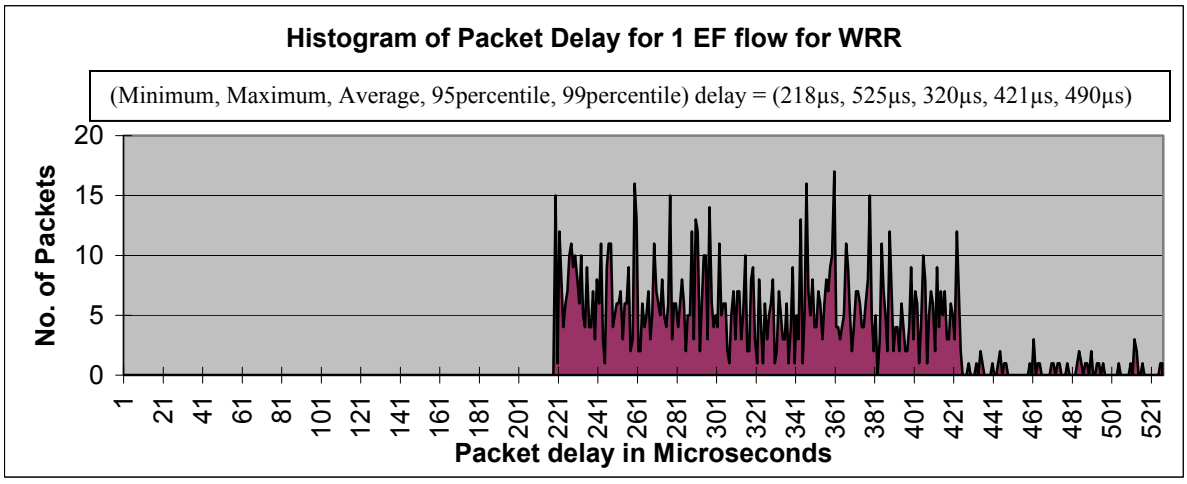

Figure 51: Histogram 24

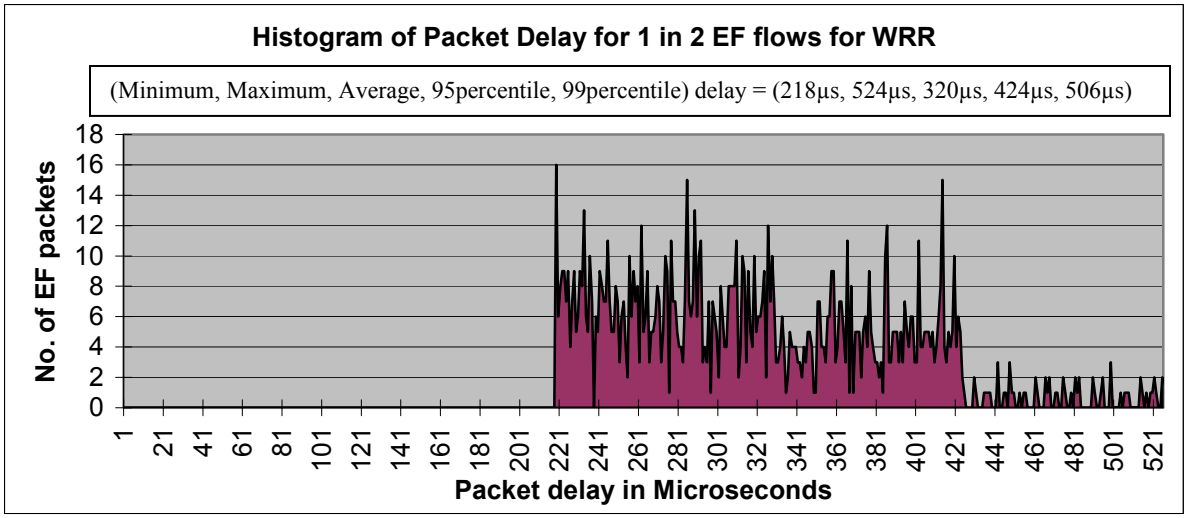

Figure 52: Histogram 25

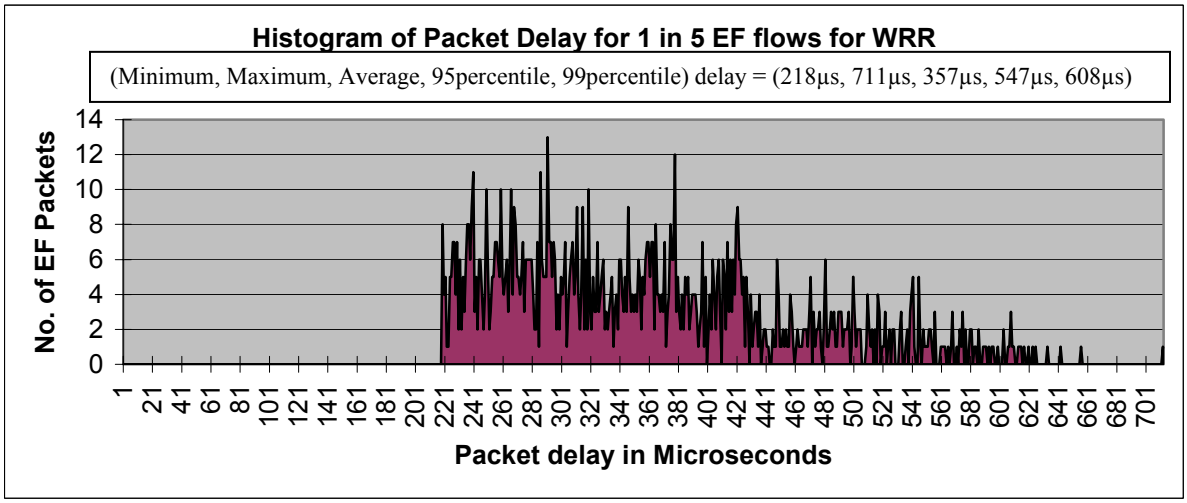

Figure 53: Histogram 26

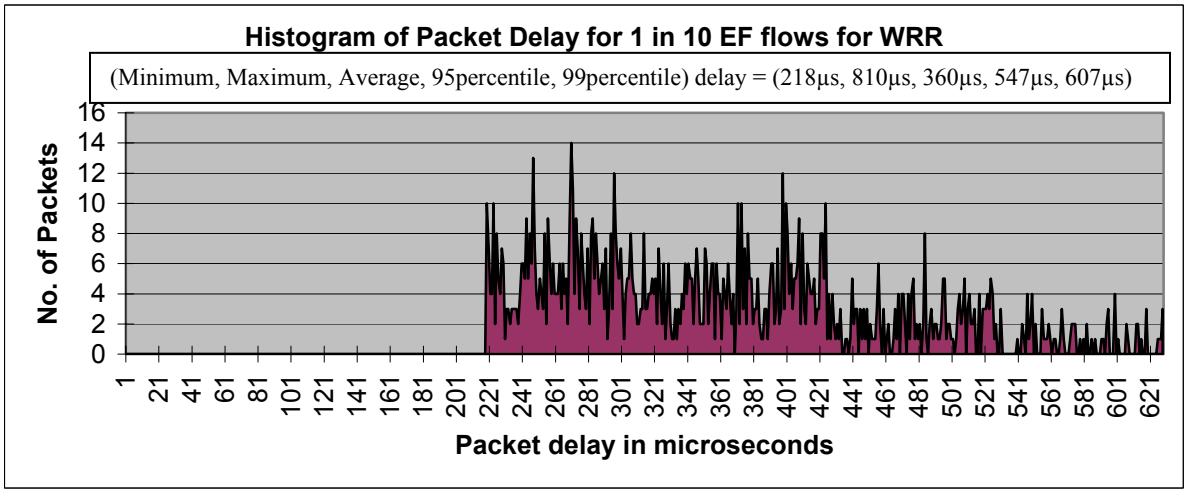

Figure 54: Histogram 27

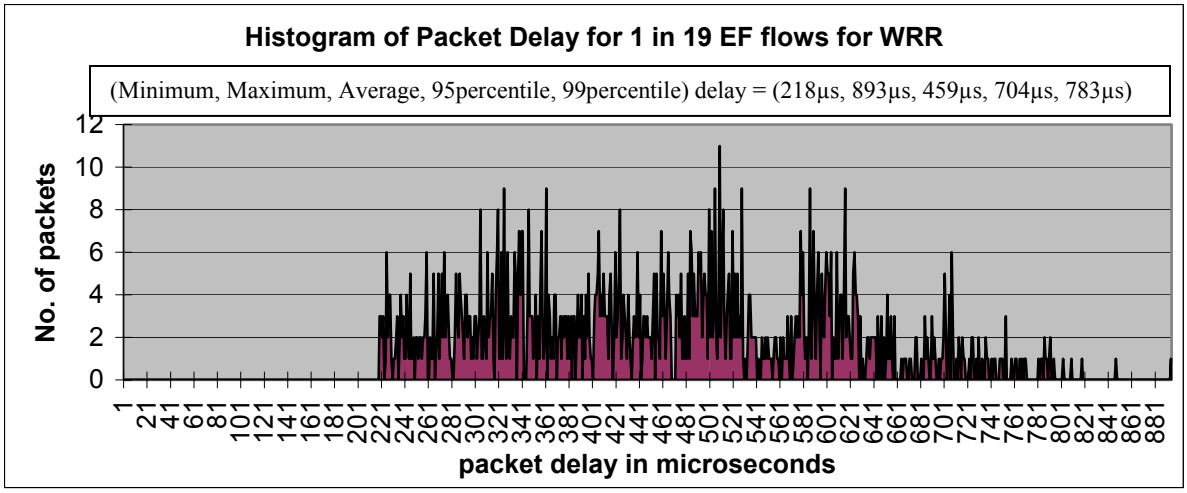

Figure 55: Histogram 28

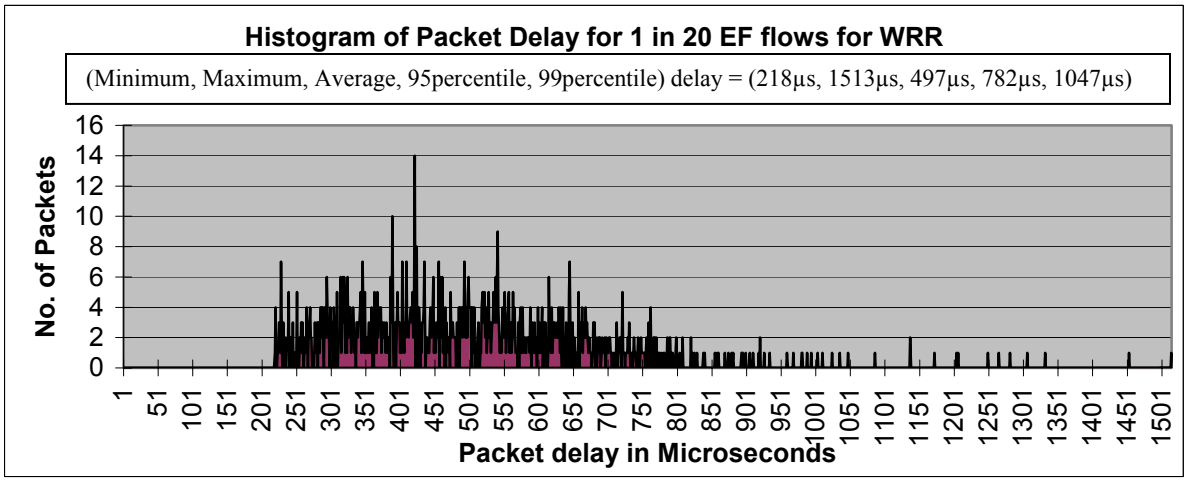

Figure 56: Histogram 29

Histograms for WRR for net EF aggregate for Experiment 2:

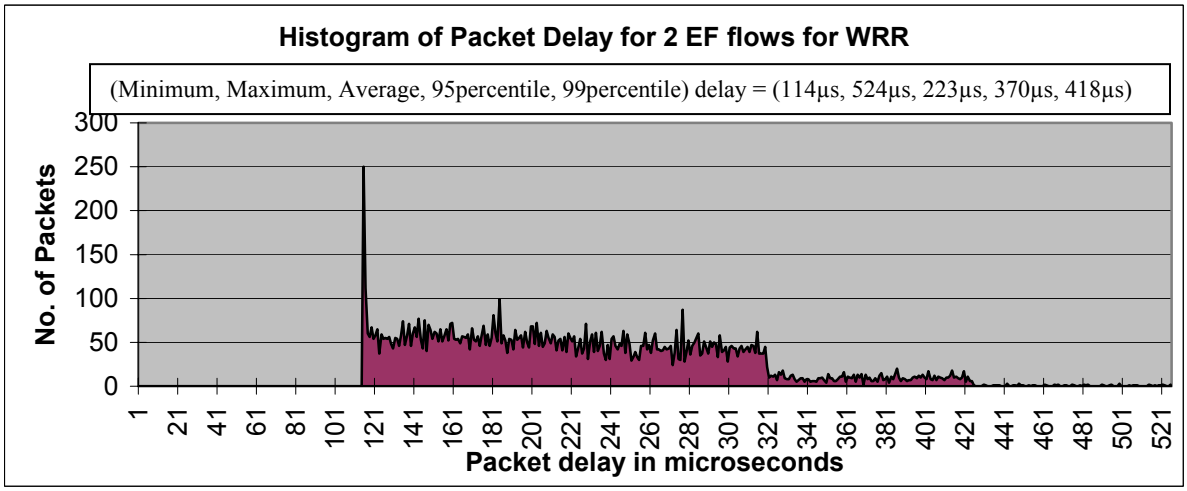

Figure 57: Histogram 30

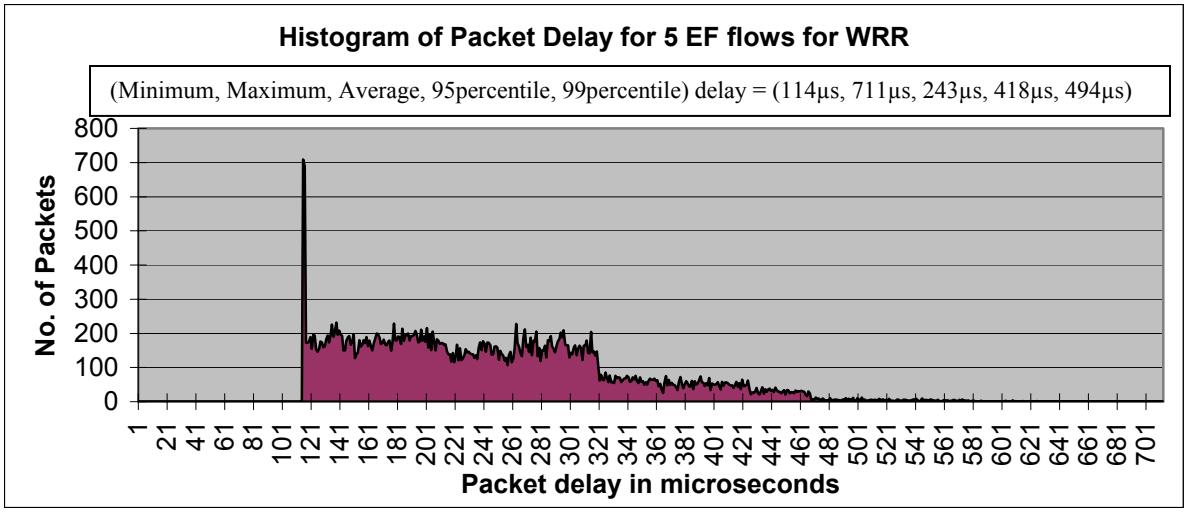

Figure 58: Histogram 31

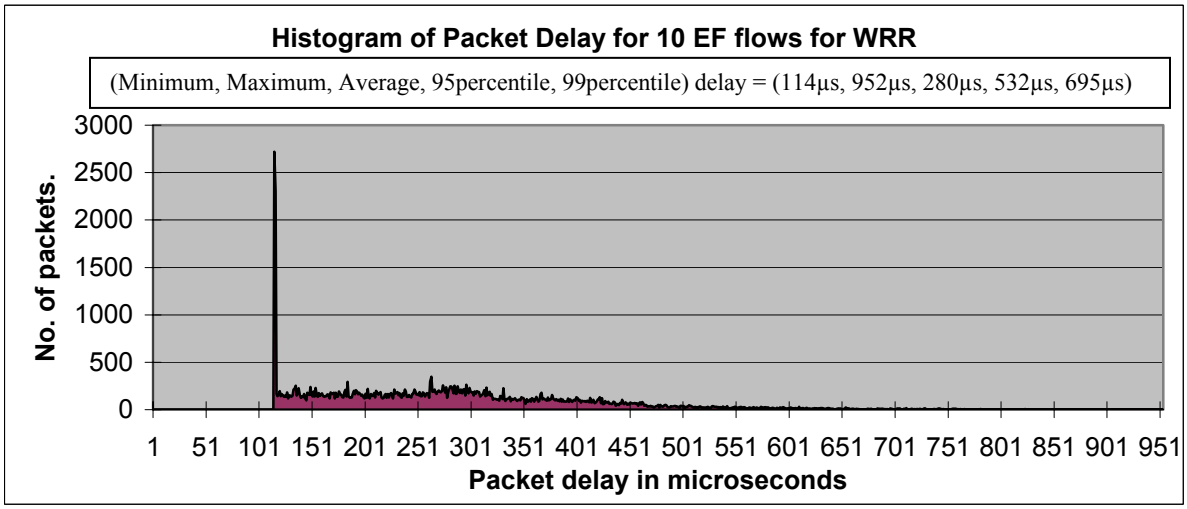

Figure 59: Histogram 32

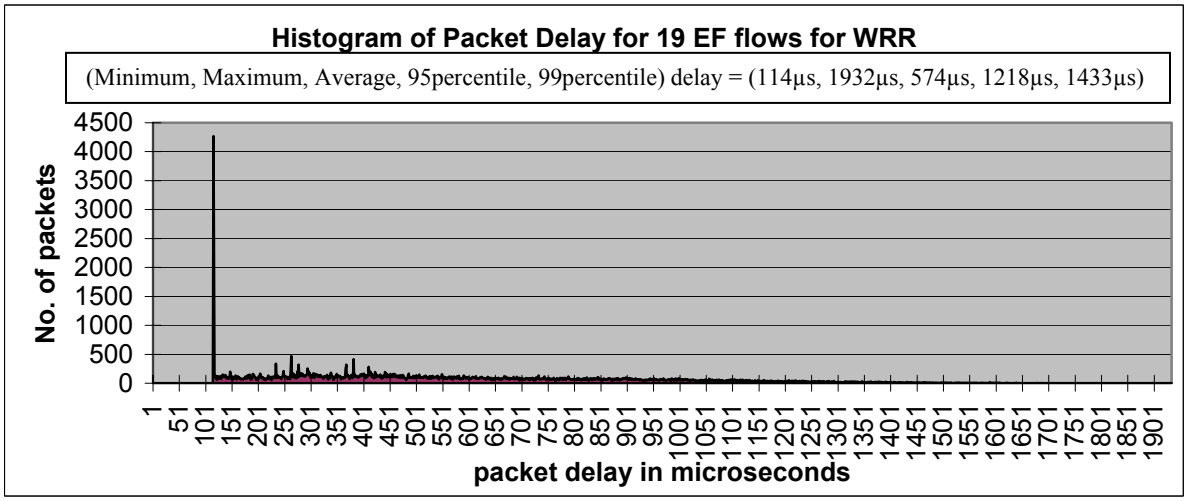

Figure 60: Histogram 33

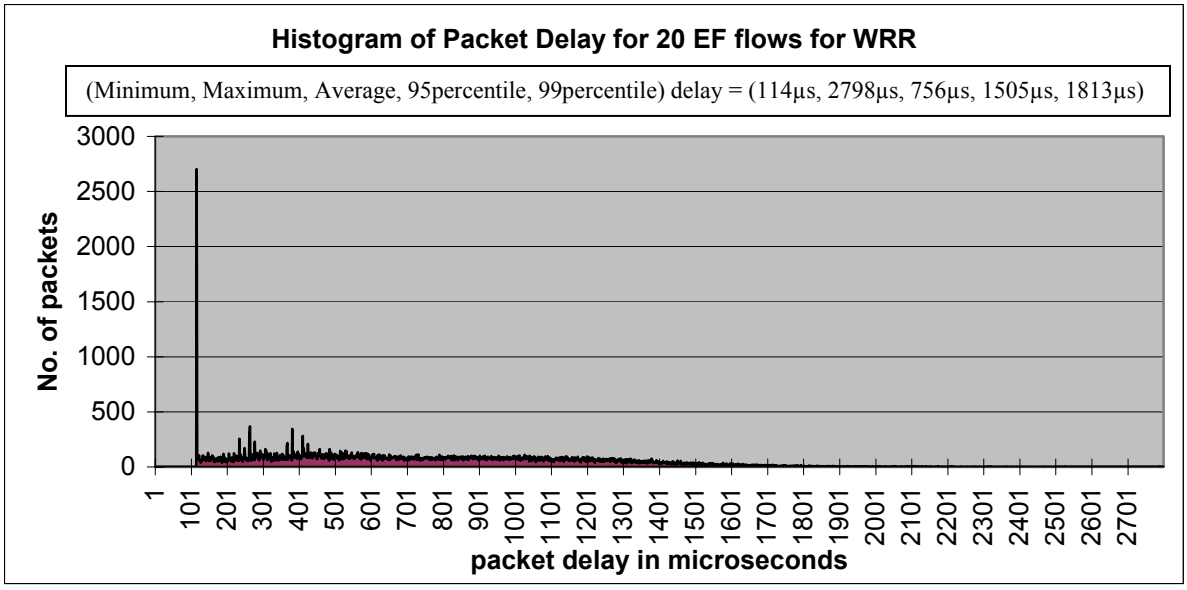

Figure 61: Histogram 34

Appendix B

Sample tcl file used in Experiment 1 for PQ scheduler for the scenario with 2EF flows:

```
#---------------------------------------------------------------------------
# FileName   : PRI_scenario4.tcl
# Author     : Mukul Senapati
# Date       : 30th June, 2002
# Notes      : Study of performance by PQ scheduler, by varying number of
#              EF flows.
#              This experiment contains 2 EF flows.
#              Sending 1 CBR EF traffic flow from S3 to D2 at 1.5Mbps.
#              Sending 1 CBR EF traffic flow from S6 to D3 at 1.5Mbps.
# Congestion : 1) Sending from S1 to D1 at 31Mbps, BE flow
#              2) Sending from S2 to D1 at 31Mbps, BE flow
#              3) Sending from S4 to D2 at 31Mbps, BE flow
#              4) Sending from S5 to D3 at 31Mbps, BE flow
# Command    : ns PRI_scenario4.tcl
#---------------------------------------------------------------------------

set ns [new Simulator]

$ns color 10 blue
$ns color 11 purple
$ns color 20 green
$ns color 21 yellow

set  f [open out.tr w]
set f1 [open BE1.tr w]
set f2 [open BE2.tr w]
set f3 [open EF1.tr w]
set f4 [open BE3.tr w]
set f5 [open BE4.tr w]
set f6 [open EF2.tr w]
$ns trace-all $f

#set nf [open out.nam w]
#$ns namtrace-all $nf

proc finish { } {
       global  f1 f2 f3 f4 f5 f6 ns #nf
       $ns flush-trace
            #close $nf
            close $f1
            close $f2
            close $f3
            close $f4
            close $f5
            close $f6
            #exec nam out.nam &
            #exec xgraph BE1.tr BE2.tr BE3.tr BE4.tr EF1.tr EF2.tr -
geometry 1200x800 &
            exit 0
            }
```

```
set cir1  0
set cbs1  0
set rate1 31000000

set cir2  0
set cbs2  0
set rate2 31000000

set cir3  15000000
set cbs3  15000000
set rate3 1500000

set cir4  0
set cbs4  0
set rate4 31000000

set cir5  0
set cbs5  0
set rate5 31000000

set cir6  15000000
set cbs6  15000000
set rate6 1500000

set testTime 10.0
set packetSize1 1480
set packetSize2 180
set packetSize3 830

# Set up the network topology shown at the top of this file:
set s1 [$ns node]
set s2 [$ns node]
set s3 [$ns node]
set s4 [$ns node]
set s5 [$ns node]
set s6 [$ns node]

set e1 [$ns node]
set e2 [$ns node]
set e3 [$ns node]

set core1 [$ns node]
set core2 [$ns node]

set d1 [$ns node]
set d2 [$ns node]
set d3 [$ns node]

$ns duplex-link $s1 $e1 200Mb 0.1ms DropTail
$ns duplex-link $s2 $e1 200Mb 0.1ms DropTail
$ns duplex-link $s3 $e1 200Mb 0.1ms DropTail
$ns duplex-link $s4 $e2 200Mb 0.1ms DropTail
$ns duplex-link $s5 $e2 200Mb 0.1ms DropTail
$ns duplex-link $s6 $e2 200Mb 0.1ms DropTail
$ns duplex-link $d1 $e3 200Mb 0.1ms DropTail
```

```
$ns duplex-link $d2 $e3 200Mb 0.1ms DropTail
$ns duplex-link $d3 $e3 200Mb 0.1ms DropTail

$ns simplex-link $e1 $core1 100Mb 0.1ms dsRED/edge
$ns simplex-link $core1 $e1 100Mb 0.1ms dsRED/core
$ns simplex-link $e2 $core1 100Mb 0.1ms dsRED/edge
$ns simplex-link $core1 $e2 100Mb 0.1ms dsRED/core
$ns simplex-link $e3 $core2 100Mb 0.1ms dsRED/edge
$ns simplex-link $core2 $e3 100Mb 0.1ms dsRED/core
$ns simplex-link $core1 $core2 100Mb 0.1ms dsRED/core
$ns simplex-link $core2 $core1 100Mb 0.1ms dsRED/core

$ns duplex-link-op $s1 $e1 orient down
$ns duplex-link-op $s2 $e1 orient down-right
$ns duplex-link-op $s3 $e1 orient right
$ns duplex-link-op $s4 $e2 orient right
$ns duplex-link-op $s5 $e2 orient up-right
$ns duplex-link-op $s6 $e2 orient up
$ns simplex-link-op $e1 $core1 orient down-right
$ns simplex-link-op $core1 $e1 orient up-left
$ns simplex-link-op $e2 $core1 orient up-right
$ns simplex-link-op $core1 $e2 orient down-left
$ns simplex-link-op $core1 $core2 orient right
$ns simplex-link-op $core2 $core1 orient left
$ns simplex-link-op $e3 $core2 orient left
$ns simplex-link-op $core2 $e3 orient right
$ns duplex-link-op $e3 $d1 orient up-right
$ns duplex-link-op $e3 $d2 orient right
$ns duplex-link-op $e3 $d3 orient down-right

$ns simplex-link-op $e1 $core1 queuePos 0.5
$ns simplex-link-op $core1 $e2 queuePos 0.5
$ns simplex-link-op $e2 $core1 queuePos 1.5
$ns simplex-link-op $core1 $e1 queuePos 1.5
$ns simplex-link-op $core1 $core2 queuePos 0.5
$ns simplex-link-op $core2 $core1 queuePos 1.5
$ns simplex-link-op $core2 $e3 queuePos 0.5
$ns simplex-link-op $e3 $core2 queuePos 1.5

#SET QUEUE HANDLES
set qE1C1 [[$ns link $e1 $core1] queue]
set qC1E1 [[$ns link $core1 $e1] queue]
set qC1E2 [[$ns link $core1 $e2] queue]
set qE2C1 [[$ns link $e2 $core1] queue]
set qE3C2 [[$ns link $e3 $core2] queue]
set qC2E3 [[$ns link $core2 $e3] queue]
set qC1C2 [[$ns link $core1 $core2] queue]
set qC2C1 [[$ns link $core2 $core1] queue]

# Set DS RED parameters from Edge1 to Core1:
$qE1C1 setSchedularMode PRI
$qE1C1 addQueueRate 0 30000000
$qE1C1 meanPktSize $packetSize3
$qE1C1 set numQueues_ 2
$qE1C1 setNumPrec 2
```

```
$qE1C1 addPolicyEntry [$s1 id] [$d1 id] TokenBucket 20 $cir1 $cbs1
$qE1C1 addPolicyEntry [$s2 id] [$d1 id] TokenBucket 20 $cir2 $cbs2
$qE1C1 addPolicyEntry [$s3 id] [$d2 id] TokenBucket 10 $cir3 $cbs3
$qE1C1 addPolicerEntry TokenBucket 10 11
$qE1C1 addPolicerEntry TokenBucket 20 21
$qE1C1 addPHBEntry 10 0 0
$qE1C1 addPHBEntry 11 0 1
$qE1C1 addPHBEntry 20 1 0
$qE1C1 addPHBEntry 21 1 1
$qE1C1 configQ 0 0 160 180 0.02
$qE1C1 configQ 0 1 140 160 0.10
$qE1C1 configQ 1 0 15 30 0.1
$qE1C1 configQ 1 1 8 15 0.30

# Set DS RED parameters from Core1 to Edge1:
$qC1E1 setSchedularMode PRI
$qC1E1 addQueueRate 0 30000000
$qC1E1 meanPktSize $packetSize3
$qC1E1 set numQueues_ 2
$qC1E1 setNumPrec 2
$qC1E1 addPHBEntry 10 0 0
$qC1E1 addPHBEntry 11 0 1
$qC1E1 addPHBEntry 20 1 0
$qC1E1 addPHBEntry 21 1 1
$qC1E1 configQ 0 0 160 180 0.02
$qC1E1 configQ 0 1 140 160 0.10
$qC1E1 configQ 1 0 15 30 0.1
$qC1E1 configQ 1 1 8 15 0.30

# Set DS RED parameters from Edge2 to Core1:
$qE2C1 setSchedularMode PRI
$qE2C1 addQueueRate 0 30000000
$qE2C1 meanPktSize $packetSize3
$qE2C1 set numQueues_ 2
$qE2C1 setNumPrec 2
$qE2C1 addPolicyEntry [$s4 id] [$d1 id] TokenBucket 20 $cir4 $cbs4
$qE2C1 addPolicyEntry [$s5 id] [$d1 id] TokenBucket 20 $cir5 $cbs5
$qE2C1 addPolicyEntry [$s6 id] [$d3 id] TokenBucket 10 $cir6 $cbs6
$qE2C1 addPolicerEntry TokenBucket 10 11
$qE2C1 addPolicerEntry TokenBucket 20 21
$qE2C1 addPHBEntry 10 0 0
$qE2C1 addPHBEntry 11 0 1
$qE2C1 addPHBEntry 20 1 0
$qE2C1 addPHBEntry 21 1 1
$qE2C1 configQ 0 0 160 180 0.02
$qE2C1 configQ 0 1 140 160 0.10
$qE2C1 configQ 1 0 15 30 0.1
$qE2C1 configQ 1 1 8 15 0.30

# Set DS RED parameters from Core1 to Edge2:
$qC1E2 setSchedularMode PRI
$qC1E2 addQueueRate 0 30000000
$qC1E2 meanPktSize $packetSize3
$qC1E2 set numQueues_ 2
$qC1E2 setNumPrec 2
```

```
$qC1E2 addPHBEntry 10 0 0
$qC1E2 addPHBEntry 11 0 1
$qC1E2 addPHBEntry 20 1 0
$qC1E2 addPHBEntry 21 1 1
$qC1E2 configQ 0 0 160 180 0.02
$qC1E2 configQ 0 1 140 160 0.10
$qC1E2 configQ 1 0 15 30 0.1
$qC1E2 configQ 1 1 8 15 0.30

# Set DS RED parameters from Core1 to core2:
$qC1C2 setSchedularMode PRI
$qC1C2 addQueueRate 0 30000000
$qC1C2 meanPktSize $packetSize3
$qC1C2 set numQueues_ 2
$qC1C2 setNumPrec 2
$qC1C2 addPHBEntry 10 0 0
$qC1C2 addPHBEntry 11 0 1
$qC1C2 addPHBEntry 20 1 0
$qC1C2 addPHBEntry 21 1 1
$qC1C2 configQ 0 0 160 180 0.02
$qC1C2 configQ 0 1 140 160 0.10
$qC1C2 configQ 1 0 15 30 0.10
$qC1C2 configQ 1 1 8 15 0.30

# Set DS RED parameters from Core2 to core1:
$qC2C1 setSchedularMode PRI
$qC2C1 addQueueRate 0 30000000
$qC2C1 meanPktSize $packetSize3
$qC2C1 set numQueues_ 2
$qC2C1 setNumPrec 2
$qC2C1 addPHBEntry 10 0 0
$qC2C1 addPHBEntry 11 0 1
$qC2C1 addPHBEntry 20 1 0
$qC2C1 addPHBEntry 21 1 1
$qC2C1 configQ 0 0 160 180 0.02
$qC2C1 configQ 0 1 140 160 0.10
$qC2C1 configQ 1 0 15 30 0.10
$qC2C1 configQ 1 1 8 15 0.30

# Set DS RED parameters from Edge3 to Core2:
$qE3C2 setSchedularMode PRI
$qE3C2 addQueueRate 0 30000000
$qE3C2 meanPktSize $packetSize3
$qE3C2 set numQueues_ 2
$qE3C2 setNumPrec 2
$qE3C2 addPolicyEntry [$s1 id] [$d1 id] TokenBucket 20 $cir1 $cbs1
$qE3C2 addPolicyEntry [$s2 id] [$d1 id] TokenBucket 20 $cir2 $cbs2
$qE3C2 addPolicyEntry [$s3 id] [$d2 id] TokenBucket 10 $cir3 $cbs3
$qE3C2 addPolicyEntry [$s4 id] [$d1 id] TokenBucket 20 $cir4 $cbs4
$qE3C2 addPolicyEntry [$s5 id] [$d1 id] TokenBucket 20 $cir5 $cbs5
$qE3C2 addPolicyEntry [$s6 id] [$d3 id] TokenBucket 10 $cir6 $cbs6
$qE3C2 addPolicerEntry TokenBucket 10 11
$qE3C2 addPolicerEntry TokenBucket 20 21
$qE3C2 addPHBEntry 10 0 0
$qE3C2 addPHBEntry 11 0 1
```

```
$qE3C2 addPHBEntry 20 1 0
$qE3C2 addPHBEntry 21 1 1
$qE3C2 configQ 0 0 160 180 0.02
$qE3C2 configQ 0 1 140 160 0.10
$qE3C2 configQ 1 0 15 30 0.10
$qE3C2 configQ 1 1 8 15 0.30

# Set DS RED parameters from Core2 to Edge3:
$qC2E3 setSchedularMode PRI
$qC2E3 addQueueRate 0 30000000
$qC2E3 meanPktSize $packetSize3
$qC2E3 set numQueues_ 2
$qC2E3 setNumPrec 2
$qC2E3 addPHBEntry 10 0 0
$qC2E3 addPHBEntry 11 0 1
$qC2E3 addPHBEntry 20 1 0
$qC2E3 addPHBEntry 21 1 1
$qC2E3 configQ 0 0 160 180 0.02
$qC2E3 configQ 0 1 140 160 0.10
$qC2E3 configQ 1 0 15 30 0.10
$qC2E3 configQ 1 1 8 15 0.30

# SET UP A CBR CONNECTION BETWEEN EACH SOURCE AND THE DESTINATION: SOURCE
IS A UDP/ CBR traffic FOR ALL 6 FLOWS

set udp1 [new Agent/UDP]
$ns attach-agent $s1 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$cbr1 set packet_size_ $packetSize1
$udp1 set packetSize_ $packetSize1
$cbr1 set rate_ $rate1
set null1 [new Agent/LossMonitor]
$ns attach-agent $d1 $null1
$ns connect $udp1 $null1

set udp2 [new Agent/UDP]
$ns attach-agent $s2 $udp2
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2
$cbr2 set packet_size_ $packetSize1
$udp2 set packetSize_ $packetSize1
$cbr2 set rate_ $rate2
set null2 [new Agent/LossMonitor]
$ns attach-agent $d1 $null2
$ns connect $udp2 $null2

set udp3 [new Agent/UDP]
$ns attach-agent $s3 $udp3
set cbr3 [new Application/Traffic/CBR]
$cbr3 attach-agent $udp3
$cbr3 set packet_size_ $packetSize1
$udp3 set packetSize_ $packetSize1
$cbr3 set rate_ $rate3
set null3 [new Agent/LossMonitor]
```

```
$ns attach-agent $d2 $null3
$ns connect $udp3 $null3

set udp4 [new Agent/UDP]
$ns attach-agent $s4 $udp4
set cbr4 [new Application/Traffic/CBR]
$cbr4 attach-agent $udp4
$cbr4 set packet_size_ $packetSize2
$udp4 set packetSize_ $packetSize2
$cbr4 set rate_ $rate4
set null4 [new Agent/LossMonitor]
$ns attach-agent $d1 $null4
$ns connect $udp4 $null4

set udp5 [new Agent/UDP]
$ns attach-agent $s5 $udp5
set cbr5 [new Application/Traffic/CBR]
$cbr5 attach-agent $udp5
$cbr5 set packet_size_ $packetSize2
$udp5 set packetSize_ $packetSize2
$cbr5 set rate_ $rate5
set null5 [new Agent/LossMonitor]
$ns attach-agent $d1 $null5
$ns connect $udp5 $null5

set udp6 [new Agent/UDP]
$ns attach-agent $s6 $udp6
set cbr6 [new Application/Traffic/CBR]
$cbr6 attach-agent $udp6
$cbr6 set packet_size_ $packetSize2
$udp6 set packetSize_ $packetSize2
$cbr6 set rate_ $rate6
set null6 [new Agent/LossMonitor]
$ns attach-agent $d3 $null6
$ns connect $udp6 $null6

proc record {} {
global null1 null2 null3 null4 null5 null6 f1 f2 f3 f4 f5 f6
set ns [Simulator instance]
set time 0.5
set bw1 [$null1 set bytes_]
set bw2 [$null2 set bytes_]
set bw3 [$null3 set bytes_]
set bw4 [$null4 set bytes_]
set bw5 [$null5 set bytes_]
set bw6 [$null6 set bytes_]
set now [$ns now]
puts $f1 "$now [expr $bw1/$time*8/1000000]"
puts $f2 "$now [expr $bw2/$time*8/1000000]"
puts $f3 "$now [expr $bw3/$time*8/1000000]"
puts $f4 "$now [expr $bw4/$time*8/1000000]"
puts $f5 "$now [expr $bw5/$time*8/1000000]"
puts $f6 "$now [expr $bw6/$time*8/1000000]"
$null1 set bytes_ 0
$null2 set bytes_ 0
```

```
$null3 set bytes_ 0
$null4 set bytes_ 0
$null5 set bytes_ 0
$null6 set bytes_ 0
$ns at [expr $now+$time] "record"
}

$qE1C1 printPolicyTable
$qE1C1 printPolicerTable
$qE1C1 printPHBTable PRI
$qE2C1 printPolicyTable
$qE2C1 printPolicerTable
$qE2C1 printPHBTable PRI
$qE3C2 printPolicyTable
$qE3C2 printPolicerTable
$qE3C2 printPHBTable PRI

$ns at 0.0 "record"
$ns at 0.0 "$cbr1 start"
$ns at 0.0 "$cbr2 start"
$ns at 0.0 "$cbr3 start"
$ns at 0.0 "$cbr4 start"
$ns at 0.0 "$cbr5 start"
$ns at 0.0 "$cbr6 start"

#$ns at 10.0 "$qE1C1 printStats"
$ns at 10.0 "$qC2E3 printStats"

$ns at $testTime "$cbr1 stop"
$ns at $testTime "$cbr2 stop"
$ns at $testTime "$cbr3 stop"
$ns at $testTime "$cbr4 stop"
$ns at $testTime "$cbr5 stop"
$ns at $testTime "$cbr6 stop"

$ns at [expr $testTime + 1.0] "finish"

$ns run
```

Sample awk script used to capture per hop delay statistics after a simulation run.

```
BEGIN {
   # Author     : Mukul Senapati
   # Date       : 30th June, 2002
   # Notes      : simple awk script to generate per hop packet
   #              statistics namely maximum, minimum and average packet
   #              delay for a specific EF flow being monitored

   highest_packet_id = 0;
}
{
   action = $1;
   time = $2;
   node_1 = $3;
   node_2 = $4;
   src = $5;
   flow_id = $8;
   node_1_address = $9;
   node_2_address = $10;
   seq_no = $11;
   packet_id = $12;
   hold = $13;
if ($3 == "9" && $4 == "10")
{
if (($10 >= "12.0" && $10 < "13.0" ) || ($10 >= "13.0" && $10 < "14.0" ))
 {
   if ( packet_id > highest_packet_id )
   {
      highest_packet_id = packet_id;
   }

   # note the arrival time of the packet here
   if ( start_time[packet_id] == 0 )  start_time[packet_id] = time;

   # checking receive means avoiding recording drops
   if ( action != "d" )
   {
      if ( action == "r" )
      {
            end_time[packet_id] = time;
      }
   }
   else
   {
      end_time[packet_id] = -1;
   }
  }
 }
}
END {
      flag = 0;
      counter =1;
      arr = 0;
      dep = 0;
```

```
      jitter1 = 0;
      jitter = 0;
      for ( packet_id = 0; packet_id <= highest_packet_id; packet_id++ )
      {
            start = start_time[packet_id];
            end = end_time[packet_id];
            packet_duration[packet_id] = end - start;
            if(flag ==0 && packet_duration[packet_id] != 0)
            {
                  flag = 1;
                  lowest = packet_duration[packet_id];
                  highest = packet_duration[packet_id];
            }

      if ( start < end )
      {
            counter++;
            arr_diff = start - arr;
            dep_diff = end - dep;
            if (arr_diff >= dep_diff)
                  jitter1 = arr_diff - dep_diff;
            else
                  jitter1 = dep_diff - arr_diff;

            arr = start;
            dep = end;

            if (jitter < jitter1)
                  jitter = jitter1;

            avg_delay = (avg_delay * (counter - 1)+
      packet_duration[packet_id])/counter;
            if(lowest > packet_duration[packet_id])
            {
                  lowest = packet_duration[packet_id];
            }
            if(highest < packet_duration[packet_id])
            {
                  highest = packet_duration[packet_id];
            }
            printf("%f\n", packet_duration[packet_id]);
       }
   }

   printf("FINAL END TO END DELAY STATISTICS\n");
   printf("_____________________________________\n\n");
   printf("LOWEST END TO END DELAY = %f\n\n", lowest);
   printf("HIGHEST END TO END DELAY = %f\n\n", highest);
   printf("AVERAGE END TO END DELAY = %f\n\n", avg_delay);
   printf("max - min delay = %f\n\n", (highest - lowest));
   printf("worst case jitter = %f\n\n", jitter);
   printf("NO OF PKTS = %f\n\n", counter);
}
```

---

histogram.awk: Sample awk script used to generate data for the histograms:

```
BEGIN {
   # Author     : Mukul Senapati
   # Date       : 30th June, 2002
   # Notes      : simple awk script to generate the histograms for
   #              packet delays per hop packet
   # Command    : awk -f histogram.awk delay1.txt > result1.txt


   value = 0;
   maxvalue=0;
}
{
value = $1*1000000;
histogram[value]++;
if (maxvalue < value)
      maxvalue = value;
}
END { for(i = 0; i <= maxvalue; i++)
      printf("%d\n", histogram[i]);}
```

---

variance.awk: Sample awk script used to determine the variance and the standard deviation in the per hop packet delays

```
BEGIN {
   # Author     : Mukul Senapati
   # Date       : 30th June, 2002
   # Notes      : simple awk script to generate the variance and
   #              standard deviation for per hop packet delays
   # Command    : awk -f variance.awk delay1.txt > variance1.txt

counter = 0;
sum = 0;
variance = 0;
stddeviation = 0;
}
{
      if ($1 > "0.0")
      {
      # Note fill in the avg packet delay in quotes
      # in the command below.
      sum = sum + (($1 - "0.000313")*($1 - "0.000313"))
      counter++;
      }
}
END {
printf("sum = %.8f\n\n", sum);
printf("counter = %f\n\n", counter);
printf("variance = %.10f\n\n", (sum/counter));
printf("standard deviation = %.10f\n\n", sqrt(sum/counter));
}
```

---

threshold.awk: Sample awk script used to determine the number of packets having packet delays above the computed threshold value.

```
BEGIN {
   # Author     : Mukul Senapati
   # Date       : 30th June, 2002
   # Notes      : simple awk script to determine number of packets
   #              outside the normal distribution if considered to be
   #              uniform around the mean
   # Command    : awk -f threshold.awk delay1.txt > threshold1.txt


counter = 0;
}
{
# Fill up the computed threshold value in quotes below
if ($1 > "0.000298")
      counter++;
}
END { printf("no of packets above std deviation = %f\n\n", counter);}
```

---

calculate_Ea.awk: Sample awk script used to determine the figure of merit E_a, which is the error term for the treatment of the EF aggregate [7][8].

```
BEGIN {
   # Author     : Mukul Senapati
   # Date       : 4th July, 2002
   # Notes      : simple awk script to determine the figure of merit
   #              E_a, which is the error term used for the treatment
   #              of the EF aggregate.
   # Command    : awk -f calculate_Ea.awk out.tr > Ea_result1.txt

highest_packet_id = 0;
   check_departure = 0;
   inpktcount = 0;
   outpktcount = 0;
   min = 0;
   packetid1 = 0;
   packetid2 = 0;
   track = 0;
}

{
      action = $1;
      time = $2;
      node_1 = $3;
      node_2 = $4;
      src = $5;
      size = $6;
      flow_id = $8;
      node_1_address = $9;
      node_2_address = $10;
      seq_no = $11;
      packet_id = $12;
```

```
   hold = $13;
      if ($3 == "9" && $4 == "10")
      {
      if(($10 >= "12.0" && $10 < "13.0")||($10 >= "13.0" && $10 < "14.0"))
      {
            if (($10 >= "12.0" && $10 < "13.0" ))
            {
                  if (packet_id > packetid1)
                  {
                        packetid1 = packet_id;
                        inpktcount++;
                        A[inpktcount] = time;
                  }
            }
            if (($10 >= "13.0" && $10 < "14.0" ))
            {
                  if (packet_id > packetid2)
                  {
                        packetid2 = packet_id;
                        inpktcount++;
                        A[inpktcount] = time;
                  }
            }

            if (( action == "r" ) || (action == "d"))
            {
                  end_time[packet_id] = time;
                  if(outpktcount == "0.0")
                  {
                        outpktcount++;
                        F[outpktcount] = A[outpktcount] + (size/30000000);
                        D[outpktcount] = time;
                  }
                  else
                  {
                        outpktcount++;
                        if(D[(outpktcount -1)] <= F[(outpktcount -1)])
                              {min = D[(outpktcount -1)];}
                        else
                              {min = F[(outpktcount -1)];}
                  if ( (A[outpktcount]) > (min))
                        F[outpktcount] = (A[outpktcount]+(size/30000000));
                  else
                        F[outpktcount] = (min + (size/30000000));
                  D[outpktcount] = time;
                  M[outpktcount] = min;
            }
            E[outpktcount] = D[outpktcount] - F[outpktcount];
      }
      }
  }
}
END {
      E_a = 0;
      if (inpktcount > outpktcount)
```

```
            limit = inpktcount;
      else limit = outpktcount;
            for ( packet_id = 0; packet_id <= limit; packet_id++) {
      temp = E[packet_id];
      if (E_a < E[packet_id])
            E_a = E[packet_id];
      printf("%f %f %f %f %f\n", A[packet_id], D[packet_id], F[packet_id],
M[packet_id], E[packet_id]);
}

   printf("FINAL E_a DELAY STATISTICS\n");
   printf("_____________________________\n\n");
   printf("the E_a factor = %f\n\n", E_a);
   printf("the output pkts count is = %f\n\n", outpktcount);
   printf("the input pkts count is = %f\n\n", inpktcount);
}
```

---

calculate_Ep.awk: Sample awk script used to determine the figure of merit E_p, which is the error term for the treatment of individual EF packets as recommended in [7] and [8].

```
BEGIN {
   # Author     : Mukul Senapati
   # Date       : 4th July, 2002
   # Notes      : simple awk script to determine the figure of merit
   #              E_p, which is the error term used for the treatment
   #              of individual EF packets.
   # Command    : awk -f calculate_Ep.awk out.tr > Ep_result1.txt

   highest_packet_id = 0;
   check_departure = 0;
   inpktcount = 0;
   outpktcount1 = 0;
   outpktcount2 = 0;
   previouspkt = 0;
   packetid1 = 0;
   packetid2 = 0;
}
{
   action = $1;
   time = $2;
   node_1 = $3;
   node_2 = $4;
   src = $5;
   pktsize = $6;
   flow_id = $8;
   node_1_address = $9;
   node_2_address = $10;
   seq_no = $11;
   packet_id = $12;
   hold = $13;
if ($3 == "9" && $4 == "10")
{
if (($10 >= "12.0" && $10 < "13.0" )||($10 >= "13.0" && $10 < "14.0" ))
```

```
{
      if (($10 >= "12.0" && $10 < "13.0" ))
      {
            if (packet_id > packetid1)
            {
                  packetid1 = packet_id;
                  inpktcount++;
                  A1[packetid1] = time;
            }
      }
      if (($10 >= "13.0" && $10 < "14.0" ))
      {
            if (packet_id > packetid2)
            {
                  packetid2 = packet_id;
                  inpktcount++;
                  A2[packetid2] = time;
            }
      }
      if ( action != "d")
      {
        if ( action == "r" )
        {
          if (($10 >= "12.0" && $10 < "13.0" ))
          {
            if(outpktcount1 == "0.0"){
                  outpktcount1++;
                  F1[packet_id] = A1[packet_id] + (pktsize/30000000);
                  D1[packet_id] = time;
                  F_prev = F1[packet_id];
                  D_prev = D1[packet_id];
            }
            else{
                  if(D_prev <= F_prev)
                        min = D_prev;
                  else
                        min = F_prev;
                  outpktcount1++;
                  if ( A1[packet_id] >= min)
                        F1[packet_id] = A1[packet_id]+(pktsize/30000000);
                  else
                        F1[packet_id] = min+(pktsize/30000000);
                  D1[packet_id] = time;
                  F_prev = F1[packet_id];
                  D_prev = D1[packet_id];
            }
            E1[packet_id] = D1[packet_id] - F1[packet_id];
          }
          if (($10 >= "13.0" && $10 < "14.0" )) {
            if(outpktcount2 == "0.0"){
                  outpktcount2++;
                  F2[packet_id] = A2[packet_id] + (pktsize/30000000);
                  D2[packet_id] = time;
                  F_prev = F2[packet_id];
                  D_prev = D2[packet_id];
```

```
            }
            else{
                  if(D_prev <= F_prev)
                        min = D_prev;
                  else
                        min = F_prev;
                  outpktcount2++;
                  if ( A2[packet_id] >= min)
                        F2[packet_id] = A2[packet_id]+(pktsize/30000000);
                  else
                        F2[packet_id] = min+(pktsize/30000000);
                  D2[packet_id] = time;
                  F_prev = F2[packet_id];
                  D_prev = D2[packet_id];
            }
            E2[packet_id] = D2[packet_id] - F2[packet_id];
          }
        }
      }
      else
            end_time[packet_id] = -1;
    }
  }
}
END {
      E_p1 = 0;
      for ( packet_id = 0; packet_id <= packetid1; packet_id++) {
            if ( E1[packet_id] > "0.0"){
                  temp = E1[packet_id];
                  if (E_p1 < E1[packet_id])
                        E_p1 = E1[packet_id];
                        printf("%f %f %f %f\n", A1[packet_id],
D1[packet_id], F1[packet_id], E1[packet_id]);
                  }
            }
      for ( packet_id = 0; packet_id <= packetid2; packet_id++) {
            if ( E2[packet_id] > "0.0"){
                  temp = E2[packet_id];
                  if (E_p2 < E2[packet_id])
                        E_p2 = E2[packet_id];
                        printf("%f %f %f %f\n", A2[packet_id],
D2[packet_id], F2[packet_id], E2[packet_id]);
            }
      }
   printf("FINAL E_p DELAY STATISTICS\n");
   printf("_____________________________________\n\n");
   printf("the E_p factors = %f %f\n\n", E_p1, E_p2);
   printf("the output pkts count is = %f %f %f\n\n", outpktcount1,
outpktcount2, (outpktcount1 + outpktcount2));
   printf("the input pkts count is = %f %f %f \n\n", inpktcount,
packetid1, packetid2);
}
```

Output generated by the ns simulator for the scenario1 of Experiment 1 for the PQ Scheduler. Output of all the scenarios is presented.

The flows 2 and 5 comprise the EF flows, within each EF flow there exist 10 EF microflows.
The flows 0, 1, 3 and 4 comprise the Best Effort traffic flows, used to congest the network.
One thing to note is that once the traffic has been conditioned at the boundary ingress nodes, there is no packet loss noted for the EF flows. Had loss occurred, it would have been shown to be policed to codepoint 11, but the absence of this codepoint in the final packet statistics indicates that no loss takes place in the core of the network.

```
PACKET STATISTICS: FOR EXPT1, PRI
---------------------

For Scenario1: For 2 EF flows
------------------------------

bash-2.05$ ns scenario1.tcl

Flow (0 to 11): Token Bucket policer, initial code  point 20, CIR 0.0 bps,
CBS 0.0 bytes.
Flow (1 to 11): Token Bucket policer, initial code  point 20, CIR 0.0 bps,
CBS 0.0 bytes.
Flow (2 to 12): Token Bucket policer, initial code  point 10, CIR
15000000.0 bps, CBS 15000000.0 bytes.

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

Code Point 10 is associated with Queue 0, Precedence 0
Code Point 11 is associated with Queue 0, Precedence 1
Code Point 20 is associated with Queue 1, Precedence 0
Code Point 21 is associated with Queue 1, Precedence 1

Scheduler Type : PRIORITY SCHEDULER

Flow (3 to 12): Token Bucket policer, initial code  point 20, CIR 0.0 bps,
CBS 0.0 bytes.
Flow (4 to 13): Token Bucket policer, initial code  point 20, CIR 0.0 bps,
CBS 0.0 bytes.
Flow (5 to 13): Token Bucket policer, initial code  point 10, CIR
15000000.0 bps, CBS 15000000.0 bytes.

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

Code Point 10 is associated with Queue 0, Precedence 0
Code Point 11 is associated with Queue 0, Precedence 1
Code Point 20 is associated with Queue 1, Precedence 0
Code Point 21 is associated with Queue 1, Precedence 1

Scheduler Type : PRIORITY SCHEDULER
```

```
Flow (0 to 11): Token Bucket policer, initial code  point 20, CIR 0.0 bps,
CBS 0.0 bytes.
Flow (1 to 11): Token Bucket policer, initial code  point 20, CIR 0.0 bps,
CBS 0.0 bytes.
Flow (2 to 12): Token Bucket policer, initial code  point 10, CIR
15000000.0 bps, CBS 15000000.0 bytes.
Flow (3 to 12): Token Bucket policer, initial code  point 20, CIR 0.0 bps,
CBS 0.0 bytes.
Flow (4 to 13): Token Bucket policer, initial code  point 20, CIR 0.0 bps,
CBS 0.0 bytes.
Flow (5 to 13): Token Bucket policer, initial code  point 10, CIR
15000000.0 bps, CBS 15000000.0 bytes.

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

Code Point 10 is associated with Queue 0, Precedence 0
Code Point 11 is associated with Queue 0, Precedence 1
Code Point 20 is associated with Queue 1, Precedence 0
Code Point 21 is associated with Queue 1, Precedence 1

Scheduler Type : PRIORITY SCHEDULER

Packets Statistics
================================================================================
=========================================
 Code Point       Total-bps       Total-Pkts    Tx-bps       Tx- Pkts
drops(q>max thresh)    drops(q<max thresh)
================================================================================
=========================================
All             32905881          163008       32905881         163008
0                       0
 10               991569             4912        991569           4912
0                       0
 21             31914312           158096      31914312         158096
0                       0
```

Sample tcl file used in Case Study Experiment 1 of the EF PDB to determine the effect of cross traffic on the EF Traffic flow.

```
#-----------------------------------------------------------------------
# FileName   : EXP1_scenario5.tcl
# Author     : Mukul Senapati
# Date       : 8th September, 2001  to 21st September, 2001
# Notes      : The NCSU Diffserv cloud and Remote Diffserv cloud
configured over Internet2.
# Description: Experiment 1, Scenario5.
#            Sending from SC1 to NCSU DS box, Machine 4: 24MBPS for 10 sec
marked EF.
# Congestion : 1) Sending from SC3 to SC4 = 95Mbps
#              2) Sending from GIG1 to GIG2 = 95Mbps
#              3) Sending from Remote 7507 to NCSU 7507 = 29Mbps
#              4) Sending from Smartbits 192.168.2.194 to 192.168.2.226 =
25Mbps
#-----------------------------------------------------------------------

set ns [new Simulator]

$ns color 10 blue
$ns color 11 purple
$ns color 20 green
$ns color 21 yellow

set  f [open out.tr w]
set f1 [open BE1.tr w]
set f2 [open BE2.tr w]
set f3 [open BE3.tr w]
set f4 [open EF.tr w]
$ns trace-all $f

#set nf [open out.nam w]
#$ns namtrace-all $nf

proc finish { } {
       global  f1 f2 f3 f4 ns #nf
       $ns flush-trace
            #close $nf
            close $f1
            close $f2
            close $f3
            close $f4
            #exec nam out.nam &
            exec xgraph BE1.tr BE2.tr BE3.tr EF.tr -geometry 1200x800 &
            exit 0
            }

set cir1  0
set cbs1  0
set rate1 95000000
```

```
set cir2  0
set cbs2  0
set rate2 95000000

set cir3  0
set cbs3  0
set rate3 29000000

set cir4  25000000
set cbs4  25000000
set rate4 25000000

set testTime 10.0
set packetSize 1480

# Set up the network topology shown in the case study of the EF PDB:
set s1 [$ns node]
set s2 [$ns node]

set e1 [$ns node]
set e2 [$ns node]
set e3 [$ns node]
set e4 [$ns node]
set e5 [$ns node]
set e6 [$ns node]
set e7 [$ns node]
set e8 [$ns node]

set core1 [$ns node]
set core2 [$ns node]
set core3 [$ns node]
set core4 [$ns node]
set core5 [$ns node]
set core6 [$ns node]
set core7 [$ns node]
set core8 [$ns node]

set dest1 [$ns node]
set SC1 [$ns node]
set SC3 [$ns node]
set SC4 [$ns node]
set GIG1 [$ns node]
set GIG2 [$ns node]

#CONFIGURING THE LINKS FOR NCSU, INTERNET2 AND REMOTE DIFFSERV CLOUDS

$ns duplex-link $s1 $e1 200Mb 0.1ms DropTail
$ns duplex-link $s2 $e2 200Mb 0.1ms DropTail
$ns duplex-link $e2 $e3 200Mb 0.1ms DropTail
$ns duplex-link $e4 $e5 200Mb 0.1ms DropTail
$ns duplex-link $e6 $dest1 200Mb 0.1ms DropTail
$ns duplex-link $dest1 $SC1 200Mb 0.1ms DropTail
$ns duplex-link $e6 $SC3 200Mb 0.1ms DropTail
$ns duplex-link $e5 $SC4 200Mb 0.1ms DropTail
```

```
$ns duplex-link $e7 $GIG2 200Mb 0.1ms DropTail
$ns duplex-link $e8 $GIG1 200Mb 0.1ms DropTail

$ns simplex-link $e1 $core1 100Mb 0.1ms dsRED/edge
$ns simplex-link $core1 $e1 100Mb 0.1ms dsRED/core
$ns simplex-link $e2 $core1 100Mb 0.1ms dsRED/edge
$ns simplex-link $core1 $e2 100Mb 0.1ms dsRED/core
$ns simplex-link $e3 $core2 100Mb 4ms dsRED/edge
$ns simplex-link $core2 $e3 100Mb 4ms dsRED/core
$ns simplex-link $e4 $core6 45Mb 4ms dsRED/edge
$ns simplex-link $core6 $e4 45Mb 4ms dsRED/core
$ns simplex-link $e5 $core7 100Mb 0.1ms dsRED/edge
$ns simplex-link $core7 $e5 100Mb 0.1ms dsRED/core
$ns simplex-link $e6 $core8 100Mb 0.1ms dsRED/edge
$ns simplex-link $core8 $e6 100Mb 0.1ms dsRED/core
$ns simplex-link $e7 $core7 100Mb 0.1ms dsRED/edge
$ns simplex-link $core7 $e7 100Mb 0.1ms dsRED/core
$ns simplex-link $e8 $core8 100Mb 0.1ms dsRED/edge
$ns simplex-link $core8 $e8 100Mb 0.1ms dsRED/core
$ns simplex-link $core2 $core3 2488Mb 4ms dsRED/core
$ns simplex-link $core3 $core2 2488Mb 4ms dsRED/core
$ns simplex-link $core3 $core4 2488Mb 4ms dsRED/core
$ns simplex-link $core4 $core3 2488Mb 4ms dsRED/core
$ns simplex-link $core4 $core5 2488Mb 4ms dsRED/core
$ns simplex-link $core5 $core4 2488Mb 4ms dsRED/core
$ns simplex-link $core5 $core6 2488Mb 4ms dsRED/core
$ns simplex-link $core6 $core5 2488Mb 4ms dsRED/core
$ns simplex-link $core7 $core8 100Mb 0.1ms dsRED/core
$ns simplex-link $core8 $core7 100Mb 0.1ms dsRED/core

#ORIENTATION OF THE LINKS FOR NCSU DIFFSERV CLOUD

$ns duplex-link-op $s1 $e1 orient down-right
$ns simplex-link-op $e1 $core1 orient right
$ns simplex-link-op $core1 $e1 orient left
$ns simplex-link-op $e2 $core1 orient left
$ns simplex-link-op $core1 $e2 orient right
$ns duplex-link-op $e2 $e3 orient right
$ns duplex-link-op $s2 $e2 orient down-right

#ORIENTATION OF THE LINKS FOR INTERNET2 DIFFSERV CLOUD
$ns simplex-link-op $e3 $core2 orient right
$ns simplex-link-op $core2 $e3 orient left
$ns simplex-link-op $core2 $core3 orient right
$ns simplex-link-op $core3 $core2 orient left
$ns simplex-link-op $core3 $core4 orient right
$ns simplex-link-op $core4 $core3 orient left
$ns simplex-link-op $core4 $core5 orient right
$ns simplex-link-op $core5 $core4 orient left
$ns simplex-link-op $core5 $core6 orient right
$ns simplex-link-op $core6 $core5 orient left
$ns simplex-link-op $e4 $core6 orient left
$ns simplex-link-op $core6 $e4 orient right
$ns duplex-link-op $e4 $e5 orient right
```

```
#ORIENTATION OF THE LINKS FOR REMOTE DIFFSERV CLOUD
$ns simplex-link-op $e5 $core7 orient right
$ns simplex-link-op $core7 $e5 orient left
$ns simplex-link-op $core7 $core8 orient right
$ns simplex-link-op $core8 $core7 orient left
$ns simplex-link-op $e6 $core8 orient left
$ns simplex-link-op $core8 $e6 orient right
$ns simplex-link-op $e7 $core7 orient down-left
$ns simplex-link-op $core7 $e7 orient up-right
$ns simplex-link-op $e8 $core8 orient down-left
$ns simplex-link-op $core8 $e8 orient up-right
$ns duplex-link-op $e6 $dest1 orient right
$ns duplex-link-op $dest1 $SC1 orient right
$ns duplex-link-op $e6 $SC3 orient down-right
$ns duplex-link-op $e5 $SC4 orient down-right
$ns duplex-link-op $e8 $GIG1 orient up-right
$ns duplex-link-op $e7 $GIG2 orient up-right

#POSITIONING THE QUEUES FOR THE NCSU DIFFSERV CLOUD
$ns simplex-link-op $e1 $core1 queuePos 0.5
$ns simplex-link-op $core1 $e2 queuePos 0.5
$ns simplex-link-op $e2 $core1 queuePos 1.5
$ns simplex-link-op $core1 $e1 queuePos 1.5

#POSITIONING THE QUEUES FOR THE INTERNET2 DIFFSERV CLOUD
$ns simplex-link-op $e3 $core2 queuePos 0.5
$ns simplex-link-op $core2 $core3 queuePos 0.5
$ns simplex-link-op $core3 $core4 queuePos 0.5
$ns simplex-link-op $core4 $core5 queuePos 0.5
$ns simplex-link-op $core5 $core6 queuePos 0.5
$ns simplex-link-op $core6 $e4 queuePos 0.5
$ns simplex-link-op $e4 $core6 queuePos 1.5
$ns simplex-link-op $core6 $core5 queuePos 1.5
$ns simplex-link-op $core5 $core4 queuePos 1.5
$ns simplex-link-op $core4 $core3 queuePos 1.5
$ns simplex-link-op $core3 $core2 queuePos 1.5

#POSITIONING THE QUEUES FOR THE REMOTE DIFFSERV CLOUD
$ns simplex-link-op $e5 $core7 queuePos 0.5
$ns simplex-link-op $core7 $core8 queuePos 0.5
$ns simplex-link-op $core8 $e8 queuePos 0.5
$ns simplex-link-op $core7 $e7 queuePos 0.5
$ns simplex-link-op $e7 $core7 queuePos 1.5
$ns simplex-link-op $e8 $core8 queuePos 1.5
$ns simplex-link-op $e6 $core8 queuePos 1.5
$ns simplex-link-op $core8 $core7 queuePos 1.5
$ns simplex-link-op $core7 $e5 queuePos 1.5

#QUEUE HANDLES FOR NCSU DIFFSERV CLOUD
set qE1C1 [[$ns link $e1 $core1] queue]
set qC1E1 [[$ns link $core1 $e1] queue]
set qC1E2 [[$ns link $core1 $e2] queue]
set qE2C1 [[$ns link $e2 $core1] queue]

#QUEUE HANDLES FOR NCSU INTERNET2 CLOUD
```

```
set qE3C2 [[$ns link $e3 $core2] queue]
set qC2E3 [[$ns link $core2 $e3] queue]
set qC2C3 [[$ns link $core2 $core3] queue]
set qC3C2 [[$ns link $core3 $core2] queue]
set qC3C4 [[$ns link $core3 $core4] queue]
set qC4C3 [[$ns link $core4 $core3] queue]
set qC4C5 [[$ns link $core4 $core5] queue]
set qC5C4 [[$ns link $core5 $core4] queue]
set qC5C6 [[$ns link $core5 $core6] queue]
set qC6C5 [[$ns link $core6 $core5] queue]
set qC6E4 [[$ns link $core6 $e4] queue]
set qE4C6 [[$ns link $e4 $core6] queue]

#QUEUE HANDLES FOR ABILENE DIFFSERV CLOUD
set qE5C7 [[$ns link $e5 $core7] queue]
set qC7E5 [[$ns link $core7 $e5] queue]
set qC7C8 [[$ns link $core7 $core8] queue]
set qC8C7 [[$ns link $core8 $core7] queue]
set qC8E6 [[$ns link $core8 $e6] queue]
set qE6C8 [[$ns link $e6 $core8] queue]
set qE7C7 [[$ns link $e7 $core7] queue]
set qE8C8 [[$ns link $e8 $core8] queue]
set qC7E7 [[$ns link $core7 $e7] queue]
set qC8E8 [[$ns link $core8 $e8] queue]

#####################################################################
# CONFIGURING THE DIFFSERV PARAMETERS FOR NCSU CLOUD

# For the NCSU Diffserv Cloud
# Set DS RED parameters from Edge1 to Core1:
$qE1C1 setSchedularMode PRI
$qE1C1 addQueueRate 0 25000000
$qE1C1 meanPktSize $packetSize
$qE1C1 set numQueues_ 1
$qE1C1 setNumPrec 2
$qE1C1 addPolicyEntry [$s1 id] [$SC1 id] TokenBucket 10 $cir4 $cbs4
$qE1C1 addPolicerEntry TokenBucket 10 11
$qE1C1 addPHBEntry 10 0 0
$qE1C1 addPHBEntry 11 0 1
$qE1C1 configQ 0 0 160 180 0.02
$qE1C1 configQ 0 1 140 160 0.10

# Set DS RED parameters from Core1 to Edge1:
$qC1E1 setSchedularMode PRI
$qC1E1 addQueueRate 0 25000000
$qC1E1 meanPktSize $packetSize
$qC1E1 set numQueues_ 1
$qC1E1 setNumPrec 2
$qC1E1 addPHBEntry 10 0 0
$qC1E1 addPHBEntry 11 0 1
$qC1E1 configQ 0 0 160 180 0.02
$qC1E1 configQ 0 1 140 160 0.10

# Set DS RED parameters from Edge2 to Core1:
$qE2C1 setSchedularMode PRI
```

```
$qE2C1 addQueueRate 0 25000000
$qE2C1 meanPktSize $packetSize
$qE2C1 set numQueues_ 1
$qE2C1 setNumPrec 2
$qE2C1 addPolicyEntry [$SC1 id] [$s1 id] TokenBucket 10 $cir4 $cbs4
$qE2C1 addPolicerEntry TokenBucket 10 11
$qE2C1 addPHBEntry 10 0 0
$qE2C1 addPHBEntry 11 0 1
$qE2C1 configQ 0 0 160 180 0.02
$qE2C1 configQ 0 1 140 160 0.10

# Set DS RED parameters from Core1 to Edge2:
$qC1E2 setSchedularMode PRI
$qC1E2 addQueueRate 0 25000000
$qC1E2 meanPktSize $packetSize
$qC1E2 set numQueues_ 1
$qC1E2 setNumPrec 2
$qC1E2 addPHBEntry 10 0 0
$qC1E2 addPHBEntry 11 0 1
$qC1E2 configQ 0 0 160 180 0.02
$qC1E2 configQ 0 1 140 160 0.10

##########################################################################
# CONFIGURING THE DIFFSERV PARAMETERS FOR INTERNET2 CLOUD

# Set DS RED parameters from Edge3 to Core2:
$qE3C2 setSchedularMode PRI
$qE3C2 addQueueRate 0 25000000
$qE3C2 meanPktSize $packetSize
$qE3C2 set numQueues_ 2
$qE3C2 setNumPrec 2
$qE3C2 addPolicyEntry [$s1 id] [$SC1 id] TokenBucket 10 $cir4 $cbs4
$qE3C2 addPolicyEntry [$e3 id] [$e4 id] TokenBucket 20 $cir3 $cbs3
$qE3C2 addPolicerEntry TokenBucket 10 11
$qE3C2 addPolicerEntry TokenBucket 20 21
$qE3C2 addPHBEntry 10 0 0
$qE3C2 addPHBEntry 11 0 1
$qE3C2 addPHBEntry 20 1 0
$qE3C2 addPHBEntry 21 1 1
$qE3C2 configQ 0 0 160 180 0.02
$qE3C2 configQ 0 1 140 160 0.10
$qE3C2 configQ 1 0 15 30 0.10
$qE3C2 configQ 1 1 1 2 0.99

# Set DS RED parameters from Core2 to Edge3:
$qC2E3 setSchedularMode PRI
$qC2E3 addQueueRate 0 25000000
$qC2E3 meanPktSize $packetSize
$qC2E3 set numQueues_ 2
$qC2E3 setNumPrec 2
$qC2E3 addPHBEntry 10 0 0
$qC2E3 addPHBEntry 11 0 1
$qC2E3 addPHBEntry 20 1 0
$qC2E3 addPHBEntry 21 1 1
$qC2E3 configQ 0 0 160 180 0.02
```

```
$qC2E3 configQ 0 1 140 160 0.10
$qC2E3 configQ 1 0 15 30 0.10
$qC2E3 configQ 1 1 1 2 0.99

# Set DS RED parameters from Core2 to core3:
$qC2C3 setSchedularMode PRI
$qC2C3 addQueueRate 0 25000000
$qC2C3 meanPktSize $packetSize
$qC2C3 set numQueues_ 2
$qC2C3 setNumPrec 2
$qC2C3 addPHBEntry 10 0 0
$qC2C3 addPHBEntry 11 0 1
$qC2C3 addPHBEntry 20 1 0
$qC2C3 addPHBEntry 21 1 1
$qC2C3 configQ 0 0 160 180 0.02
$qC2C3 configQ 0 1 140 160 0.10
$qC2C3 configQ 1 0 15 30 0.10
$qC2C3 configQ 1 1 1 2 0.99

# Set DS RED parameters from Core3 to core2:
$qC3C2 setSchedularMode PRI
$qC3C2 addQueueRate 0 25000000
$qC3C2 meanPktSize $packetSize
$qC3C2 set numQueues_ 2
$qC3C2 setNumPrec 2
$qC3C2 addPHBEntry 10 0 0
$qC3C2 addPHBEntry 11 0 1
$qC3C2 addPHBEntry 20 1 0
$qC3C2 addPHBEntry 21 1 1
$qC3C2 configQ 0 0 160 180 0.02
$qC3C2 configQ 0 1 140 160 0.10
$qC3C2 configQ 1 0 15 30 0.10
$qC3C2 configQ 1 1 1 2 0.99

# Set DS RED parameters from Core3 to core4:
$qC3C4 setSchedularMode PRI
$qC3C4 addQueueRate 0 25000000
$qC3C4 meanPktSize $packetSize
$qC3C4 set numQueues_ 2
$qC3C4 setNumPrec 2
$qC3C4 addPHBEntry 10 0 0
$qC3C4 addPHBEntry 11 0 1
$qC3C4 addPHBEntry 20 1 0
$qC3C4 addPHBEntry 21 1 1
$qC3C4 configQ 0 0 160 180 0.02
$qC3C4 configQ 0 1 140 160 0.10
$qC3C4 configQ 1 0 15 30 0.10
$qC3C4 configQ 1 1 1 2 0.99

# Set DS RED parameters from Core4 to core3:
$qC4C3 setSchedularMode PRI
$qC4C3 addQueueRate 0 25000000
$qC4C3 meanPktSize $packetSize
$qC4C3 set numQueues_ 2
$qC4C3 setNumPrec 2
```

```
$qC4C3 addPHBEntry 10 0 0
$qC4C3 addPHBEntry 11 0 1
$qC4C3 addPHBEntry 20 1 0
$qC4C3 addPHBEntry 21 1 1
$qC4C3 configQ 0 0 160 180 0.02
$qC4C3 configQ 0 1 140 160 0.10
$qC4C3 configQ 1 0 15 30 0.10
$qC4C3 configQ 1 1 1 2 0.99

# Set DS RED parameters from Core4 to core5:
$qC4C5 setSchedularMode PRI
$qC4C5 addQueueRate 0 250000000
$qC4C5 meanPktSize $packetSize
$qC4C5 set numQueues_ 2
$qC4C5 setNumPrec 2
$qC4C5 addPHBEntry 10 0 0
$qC4C5 addPHBEntry 11 0 1
$qC4C5 addPHBEntry 20 1 0
$qC4C5 addPHBEntry 21 1 1
$qC4C5 configQ 0 0 160 180 0.02
$qC4C5 configQ 0 1 140 160 0.10
$qC4C5 configQ 1 0 15 30 0.10
$qC4C5 configQ 1 1 1 2 0.99

# Set DS RED parameters from Core5 to core4:
$qC5C4 setSchedularMode PRI
$qC5C4 addQueueRate 0 25000000
$qC5C4 meanPktSize $packetSize
$qC5C4 set numQueues_ 2
$qC5C4 setNumPrec 2
$qC5C4 addPHBEntry 10 0 0
$qC5C4 addPHBEntry 11 0 1
$qC5C4 addPHBEntry 20 1 0
$qC5C4 addPHBEntry 21 1 1
$qC5C4 configQ 0 0 160 180 0.02
$qC5C4 configQ 0 1 140 160 0.10
$qC5C4 configQ 1 0 15 30 0.10
$qC5C4 configQ 1 1 1 2 0.99

# Set DS RED parameters from Core5 to core6:
$qC5C6 setSchedularMode PRI
$qC5C6 addQueueRate 0 25000000
$qC5C6 meanPktSize $packetSize
$qC5C6 set numQueues_ 2
$qC5C6 setNumPrec 2
$qC5C6 addPHBEntry 10 0 0
$qC5C6 addPHBEntry 11 0 1
$qC5C6 addPHBEntry 20 1 0
$qC5C6 addPHBEntry 21 1 1
$qC5C6 configQ 0 0 160 180 0.02
$qC5C6 configQ 0 1 140 160 0.10
$qC5C6 configQ 1 0 15 30 0.10
$qC5C6 configQ 1 1 1 2 0.99

# Set DS RED parameters from Core6 to core5:
```

```
$qC6C5 setSchedularMode PRI
$qC6C5 addQueueRate 0 25000000
$qC6C5 meanPktSize $packetSize
$qC6C5 set numQueues_ 2
$qC6C5 setNumPrec 2
$qC6C5 addPHBEntry 10 0 0
$qC6C5 addPHBEntry 11 0 1
$qC6C5 addPHBEntry 20 1 0
$qC6C5 addPHBEntry 21 1 1
$qC6C5 configQ 0 0 160 180 0.02
$qC6C5 configQ 0 1 140 160 0.10
$qC6C5 configQ 1 0 15 30 0.10
$qC6C5 configQ 1 1 1 2 0.99

# Set DS RED parameters from Edge4 to Core6:
$qE4C6 setSchedularMode PRI
$qE4C6 addQueueRate 0 25000000
$qE4C6 meanPktSize $packetSize
$qE4C6 set numQueues_ 2
$qE4C6 setNumPrec 2
$qE4C6 addPolicyEntry [$SC1 id] [$s1 id] TokenBucket 10 $cir4 $cbs4
$qE4C6 addPolicyEntry [$e4 id] [$e3 id] TokenBucket 20 $cir3 $cbs3
$qE4C6 addPolicerEntry TokenBucket 10 11
$qE4C6 addPolicerEntry TokenBucket 20 21
$qE4C6 addPHBEntry 10 0 0
$qE4C6 addPHBEntry 11 0 1
$qE4C6 addPHBEntry 20 1 0
$qE4C6 addPHBEntry 21 1 1
$qE4C6 configQ 0 0 160 180 0.02
$qE4C6 configQ 0 1 140 160 0.10
$qE4C6 configQ 1 0 15 30 0.10
$qE4C6 configQ 1 1 1 2 0.99

# Set DS RED parameters from Core6 to Edge4:
$qC6E4 setSchedularMode PRI
$qC6E4 addQueueRate 0 25000000
$qC6E4 meanPktSize $packetSize
$qC6E4 set numQueues_ 2
$qC6E4 setNumPrec 2
$qC6E4 addPHBEntry 10 0 0
$qC6E4 addPHBEntry 11 0 1
$qC6E4 addPHBEntry 20 1 0
$qC6E4 addPHBEntry 21 1 1
$qC6E4 configQ 0 0 160 180 0.02
$qC6E4 configQ 0 1 140 160 0.10
$qC6E4 configQ 1 0 15 30 0.10
$qC6E4 configQ 1 1 1 2 0.99

#####################################################################
# CONFIGURING THE DIFFSERV PARAMETERS FOR REMOTE CLOUD

# Set DS RED parameters from Edge5 to Core7:
$qE5C7 setSchedularMode PRI
$qE5C7 addQueueRate 0 25000000
$qE5C7 meanPktSize $packetSize
```

```
$qE5C7 set numQueues_ 2
$qE5C7 setNumPrec 2
$qE5C7 addPolicyEntry [$s1 id] [$SC1 id] TokenBucket 10 $cir4 $cbs4
$qE5C7 addPolicyEntry [$SC4 id] [$SC3 id] TokenBucket 20 $cir1 $cbs1
$qE5C7 addPolicerEntry TokenBucket 10 11
$qE5C7 addPolicerEntry TokenBucket 20 21
$qE5C7 addPHBEntry 10 0 0
$qE5C7 addPHBEntry 11 0 1
$qE5C7 addPHBEntry 20 1 0
$qE5C7 addPHBEntry 21 1 1
$qE5C7 configQ 0 0 160 180 0.02
$qE5C7 configQ 0 1 140 160 0.10
$qE5C7 configQ 1 0 15 30 0.10
$qE5C7 configQ 1 1 1 2 0.99

# Set DS RED parameters from Core7 to Edge5:
$qC7E5 setSchedularMode PRI
$qC7E5 addQueueRate 0 25000000
$qC7E5 meanPktSize $packetSize
$qC7E5 set numQueues_ 2
$qC7E5 setNumPrec 2
$qC7E5 addPHBEntry 10 0 0
$qC7E5 addPHBEntry 11 0 1
$qC7E5 addPHBEntry 20 1 0
$qC7E5 addPHBEntry 21 1 1
$qC7E5 configQ 0 0 160 180 0.02
$qC7E5 configQ 0 1 140 160 0.10
$qC7E5 configQ 1 0 15 30 0.10
$qC7E5 configQ 1 1 1 2 0.99

# Set DS RED parameters from Edge7 to Core7:
$qE7C7 setSchedularMode PRI
$qE7C7 addQueueRate 0 250000000
$qE7C7 meanPktSize $packetSize
$qE7C7 set numQueues_ 2
$qE7C7 setNumPrec 2
$qE7C7 addPolicyEntry [$GIG2 id] [$GIG1 id] TokenBucket 20 $cir1 $cbs1
$qE7C7 addPolicerEntry TokenBucket 10 11
$qE7C7 addPolicerEntry TokenBucket 20 21
$qE7C7 addPHBEntry 10 0 0
$qE7C7 addPHBEntry 11 0 1
$qE7C7 addPHBEntry 20 1 0
$qE7C7 addPHBEntry 21 1 1
$qE7C7 configQ 0 0 160 180 0.02
$qE7C7 configQ 0 1 140 160 0.10
$qE7C7 configQ 1 0 15 30 0.10
$qE7C7 configQ 1 1 1 2 0.99

# Set DS RED parameters from Core7 to Edge7:
$qC7E7 setSchedularMode PRI
$qC7E7 addQueueRate 0 25000000
$qC7E7 meanPktSize $packetSize
$qC7E7 set numQueues_ 2
$qC7E7 setNumPrec 2
$qC7E7 addPHBEntry 10 0 0
```

```
$qC7E7 addPHBEntry 11 0 1
$qC7E7 addPHBEntry 20 1 0
$qC7E7 addPHBEntry 21 1 1
$qC7E7 configQ 0 0 160 180 0.02
$qC7E7 configQ 0 1 140 160 0.10
$qC7E7 configQ 1 0 15 30 0.10
$qC7E7 configQ 1 1 1 2 0.99

# Set DS RED parameters from Core7 to core8:
$qC7C8 setSchedularMode PRI
$qC7C8 addQueueRate 0 25000000
$qC7C8 meanPktSize $packetSize
$qC7C8 set numQueues_ 2
$qC7C8 setNumPrec 2
$qC7C8 addPHBEntry 10 0 0
$qC7C8 addPHBEntry 11 0 1
$qC7C8 addPHBEntry 20 0 0
$qC7C8 addPHBEntry 21 0 1
$qC7C8 configQ 0 0 160 180 0.02
$qC7C8 configQ 0 1 140 160 0.10
$qC7C8 configQ 1 0 15 30 0.10
$qC7C8 configQ 1 1 1 2 0.99

# Set DS RED parameters from Core8 to core7:
$qC8C7 setSchedularMode PRI
$qC8C7 addQueueRate 0 25000000
$qC8C7 meanPktSize $packetSize
$qC8C7 set numQueues_ 2
$qC8C7 setNumPrec 2
$qC8C7 addPHBEntry 10 0 0
$qC8C7 addPHBEntry 11 0 1
$qC8C7 addPHBEntry 20 1 0
$qC8C7 addPHBEntry 21 1 1
$qC8C7 configQ 0 0 160 180 0.02
$qC8C7 configQ 0 1 140 160 0.10
$qC8C7 configQ 1 0 15 30 0.10
$qC8C7 configQ 1 1 1 2 0.99

# Set DS RED parameters from Edge6 to Core8:
$qE6C8 setSchedularMode PRI
$qE6C8 addQueueRate 0 25000000
$qE6C8 meanPktSize $packetSize
$qE6C8 set numQueues_ 2
$qE6C8 setNumPrec 2
$qE6C8 addPolicyEntry [$SC1 id] [$s1 id] TokenBucket 10 $cir4 $cbs4
$qE6C8 addPolicyEntry [$SC3 id] [$SC4 id] TokenBucket 20 $cir1 $cbs1
$qE6C8 addPolicerEntry TokenBucket 10 11
$qE6C8 addPolicerEntry TokenBucket 20 21
$qE6C8 addPHBEntry 10 0 0
$qE6C8 addPHBEntry 11 0 1
$qE6C8 addPHBEntry 20 1 0
$qE6C8 addPHBEntry 21 1 1
$qE6C8 configQ 0 0 160 180 0.02
$qE6C8 configQ 0 1 140 160 0.10
$qE6C8 configQ 1 0 15 30 0.10
```

```
$qE6C8 configQ 1 1 1 2 0.99

# Set DS RED parameters from Core8 to Edge6:
$qC8E6 setSchedularMode PRI
$qC8E6 addQueueRate 0 25000000
$qC8E6 meanPktSize $packetSize
$qC8E6 set numQueues_ 2
$qC8E6 setNumPrec 2
$qC8E6 addPHBEntry 10 0 0
$qC8E6 addPHBEntry 11 0 1
$qC8E6 addPHBEntry 20 1 0
$qC8E6 addPHBEntry 21 1 1
$qC8E6 configQ 0 0 160 180 0.02
$qC8E6 configQ 0 1 140 160 0.10
$qC8E6 configQ 1 0 15 30 0.10
$qC8E6 configQ 1 1 1 2 0.99

# Set DS RED parameters from Edge8 to Core8:
$qE8C8 setSchedularMode PRI
$qE8C8 addQueueRate 0 25000000
$qE8C8 meanPktSize $packetSize
$qE8C8 set numQueues_ 2
$qE8C8 setNumPrec 2
$qE8C8 addPolicyEntry [$GIG1 id] [$GIG2 id] TokenBucket 20 $cir1 $cbs1
$qE8C8 addPolicerEntry TokenBucket 10 11
$qE8C8 addPolicerEntry TokenBucket 20 21
$qE8C8 addPHBEntry 10 0 0
$qE8C8 addPHBEntry 11 0 1
$qE8C8 addPHBEntry 20 1 0
$qE8C8 addPHBEntry 21 1 1
$qE8C8 configQ 0 0 160 180 0.02
$qE8C8 configQ 0 1 140 160 0.10
$qE8C8 configQ 1 0 15 30 0.10
$qE8C8 configQ 1 1 1 2 0.99

# Set DS RED parameters from Core8 to Edge8:
$qC8E8 setSchedularMode PRI
$qC8E8 addQueueRate 0 250000000
$qC8E8 meanPktSize $packetSize
$qC8E8 set numQueues_ 2
$qC8E8 setNumPrec 2
$qC8E8 addPHBEntry 10 0 0
$qC8E8 addPHBEntry 11 0 1
$qC8E8 addPHBEntry 20 1 0
$qC8E8 addPHBEntry 21 1 1
$qC8E8 configQ 0 0 160 180 0.02
$qC8E8 configQ 0 1 140 160 0.10
$qC8E8 configQ 1 0 15 30 0.10
$qC8E8 configQ 1 1 1 2 0.99

#####################################################################

# SET UP A CBR CONNECTION BETWEEN EACH SOURCE AND THE DESTINATION: SOURCE
IS A UDP/ CBR traffic FOR ALL 4 FLOWS
```

```
set udp1 [new Agent/UDP]
$ns attach-agent $SC3 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$cbr1 set packet_size_ $packetSize
$udp1 set packetSize_ $packetSize
$cbr1 set rate_ $rate1
set null1 [new Agent/LossMonitor]
$ns attach-agent $SC4 $null1
$ns connect $udp1 $null1


set udp2 [new Agent/UDP]
$ns attach-agent $GIG1 $udp2
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2
$cbr2 set packet_size_ $packetSize
$udp2 set packetSize_ $packetSize
$cbr2 set rate_ $rate2
set null2 [new Agent/LossMonitor]
$ns attach-agent $GIG2 $null2
$ns connect $udp2 $null2


set udp3 [new Agent/UDP]
$ns attach-agent $e4 $udp3
set cbr3 [new Application/Traffic/CBR]
$cbr3 attach-agent $udp3
$cbr3 set packet_size_ $packetSize
$udp3 set packetSize_ $packetSize
$cbr3 set rate_ $rate3
set null3 [new Agent/LossMonitor]
$ns attach-agent $e3 $null3
$ns connect $udp3 $null3


set udp4 [new Agent/UDP]
$ns attach-agent $SC1 $udp4
set cbr4 [new Application/Traffic/CBR]
$cbr4 attach-agent $udp4
$cbr4 set packet_size_ $packetSize
$udp4 set packetSize_ $packetSize
$cbr4 set rate_ $rate4
set null4 [new Agent/LossMonitor]
$ns attach-agent $s1 $null4
$ns connect $udp4 $null4


proc record {} {
global null1 null2 null3 null4 f1 f2 f3 f4
set ns [Simulator instance]
set time 0.5
set bw1 [$null1 set bytes_]
set bw2 [$null2 set bytes_]
set bw3 [$null3 set bytes_]
```

```
set bw4 [$null4 set bytes_]
set now [$ns now]
puts $f1 "$now [expr $bw1/$time*8/1000000]"
puts $f2 "$now [expr $bw2/$time*8/1000000]"
puts $f3 "$now [expr $bw3/$time*8/1000000]"
puts $f4 "$now [expr $bw4/$time*8/1000000]"
$null1 set bytes_ 0
$null2 set bytes_ 0
$null3 set bytes_ 0
$null4 set bytes_ 0
$ns at [expr $now+$time] "record"
}

$qE1C1 printPolicyTable
$qE1C1 printPolicerTable
$qE1C1 printPHBTable PRI
$qE2C1 printPolicyTable
$qE2C1 printPolicerTable
$qE2C1 printPHBTable PRI
$qE3C2 printPolicyTable
$qE3C2 printPolicerTable
$qE3C2 printPHBTable PRI
$qE4C6 printPolicyTable
$qE4C6 printPolicerTable
$qE4C6 printPHBTable PRI
$qE5C7 printPolicyTable
$qE5C7 printPolicerTable
$qE5C7 printPHBTable PRI
$qE6C8 printPolicyTable
$qE6C8 printPolicerTable
$qE6C8 printPHBTable PRI

$ns at 0.0 "record"
$ns at 0.0 "$cbr1 start"
$ns at 0.0 "$cbr2 start"
$ns at 0.0 "$cbr3 start"
$ns at 0.0 "$cbr4 start"

$ns at 10.0 "$qE6C8 printStats"
$ns at 10.0 "$qC8C7 printStats"
$ns at 10.0 "$qC7E7 printStats"
$ns at 10.0 "$qE4C6 printStats"
$ns at 10.0 "$qC1E1 printStats"

$ns at $testTime "$cbr1 stop"
$ns at $testTime "$cbr2 stop"
$ns at $testTime "$cbr3 stop"
$ns at $testTime "$cbr4 stop"

$ns at [expr $testTime + 1.0] "finish"

$ns run
```