ABSTRACT

DWEKAT, ZYAD AHMAD. Construction and Evaluation of a Service Level Agreement Test-Bed. (Under the direction of Drs. Mladen A. Vouk and George N. Rouskas)

Differentiated Services (DiffServ) enable quality-of-service provisioning by coupling traffic aggregates with a specific forwarding path treatment. Per-Hop forwarding Behavior (or PHB) describes the forwarding path behavior required in network nodes, while the Per-Domain Behavior (or PDB) describes the behavior experienced by a particular set of packets as they cross a DiffServ domain. PHB and PDB implementations with traffic conditioners, provisioning strategies and billing models are building blocks used in Services Level Agreements (SLA).

The goal of the project was to construct an SLA test-bed using soft DiffServ routing nodes. This test-bed, and the associated tools, is intended for in-depth studies of SLA translation and mapping algorithms. The specific objectives of the work reported in this thesis were to construct the test-bed, and then verify, validate and evaluate different elements of the SLA test-bed. Specifically, proof-of-concept experiments were conducted to show that one can implement a general Services Level Agreement (SLA) based on DiffServ, to show how to translate SLA requirements into real test-bed parameters, and then run experiments, collect data and make performance measurements.. The study is empirical and involves implementation of some of the quality of service (QoS) related IETF drafts and RFCs in a five-node soft DiffServ-based SLA testbed. This includes implementation and empirical evaluation of Expedite Forwarding (EF) PHB, Assured Forwarding (AF) PHB, Virtual Wire (VW) PDB, and of Assured Forwarding PDB.

Virtual Wire PDB was implemented using both the Priority Scheduler and a Weighted Fair Queuing Scheduler. The effect of Best Effort packet size and rate on the protected EF traffic was investigated empirically. This provided proof of concept data on how successful can a VW implementation be in protecting Voice over IP (VoIP) streams. We

used both emulated VoIP streams, and real VoIP phones in highly congested traffic environments.

Assured Rate PDB was implemented using appropriate traffic conditioning, buffer management and scheduling techniques. The issue of what to do with non-conforming AF traffic was investigated empirically - whether to drop the non-conforming packets or to downgrade them to Best Effort. It was noticed that strict dropping of the non-conforming packets will sometimes underutilize the channel, while downgrading them to Best Effort will create a reordering problem.

Construction and Evaluation of a Service Level Agreement Test-Bed

By

Zyad Ahmad Dwekat

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 Networking

Raleigh

2001

APPROVED BY

Dr. George Rouskas (Co-Chair)

Dr. Arne A. Nilsson

Dr. Mladen Vouk (Chairman)

BIOGRAPHY

Zyad Dwekat was born in Saudi Arabia , in Dareen , a city by the sea. He obtained his Bachelor of Engineering degree in Electrical Engineering in Mu'tah University , Jordan. He then joined North Carolina State University for his Master degree in Computer Networking. He worked as a Research Assistant for Dr. Vouk in the field of Differentiated Services for more than a year .

ACKNOWLEDGMENTS

I wish to thank my advisor Dr. Mladen Vouk for his support and encouragement. It was a privilege and honor to have been able to work with him. I wish to thank Dr. Rouskas and Dr. Nilsson for their support and feedback. I wish to thank BellSouth and Alcatel, since this work was supported in part by grants from both companies.

I would like to express my thanks to Kesava Parasad Narasimhan for guiding me during the initial part of this research. His knowledge and dedication was a great inspiration for me. I would like to thank Amit Kulkarni and Mrujendra Singhai for helping me during my research. I would like to thank Marhn Fullmer for supporting our research needs in the Networking Lab.

No words are sufficient to express my gratitude to my parents , my two brothers and my wife , for their love and incredible support, without this would not have been possible.

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1. Motivation and Goals

Motivation for the work presented in this thesis stems from two current trends. One is an increasing inclination towards voice over IP (VoIP) solutions, both within organizations, and on a broader scale. For example, NC State University has been conducting a campus-wide building-scale (50+ VoIP telephones) pilot tests with H323-based VoIP solutions. H323 and VoIP offer a number of opportunities, both in service expansion (such as video-phone services), more rapid deployment of services and service change requests, and cost savings. Although individual VoIP services do not generally require a lot of bandwidth, what they require needs to be guaranteed. So far the results of the NC State tests have been very encouraging. Some of the issues remain open. This includes the ability of the typical data networks to actually provide some guarantees for VoIP and video services, and the availability of a typical campus, or metropolitan or wide-area network.

Another trend is towards introduction of some form of quality of service into the next generation network on the NC State Campus, in the Research Triangle Park metropolitan area, and in the North Carolina as a whole. Some of the service level agreements (SLA) that arise in that context involve the desire of the University of North Carolina campuses to a) have a certain level of guaranteed bandwidth, and b) an ability to share unused excess bandwidth in a reliable fashion. The problem becomes quite complex when one starts considering the geographical dispersion of some of the university outreach units, and the necessity to provide both reliable and fair service. A workable solution may involve supporting an array of service levels, from Best Effort (BE), to several levels of Assured Forwarding or equivalent, to Expedited Forwarding or equivalent. A natural candidate for implementation of such services is the Differentiated Services (DiffServ) solution [12] [15] [17].

The current work is concerned with implementation and evaluation of a testbed, and its elements, for evaluation of general Service Level Agreement implementation using DiffServ. The goal was to assess whether a full implementation of DiffServ a) can provide sufficient flexibility for studying general SLA implementations, b) can provide guarantees that may be needed for QoS-sensitive applications such as voice over IP (VoIP), and c) whether implementation of assured forwarding services is viable and sustainable in the context of the test-bed. This is of special interest when studying SLAs that may involve dynamic downgrade of certain flow streams, and may thus involve the packet re-ordering problem [10]. Specific objectives involve implementation and assessment of guaranteed and assured forwarding services (both per-hop and per-domain), and of a proof-of-concept evaluation of the performance of these implementations, in the context of a very detailed and finely tunable implementation of DiffServ on Linux-based networking nodes.

1.2. Why Differentiated Services?

Traditionally, network service providers offer customers two types of service: either a form of dedicated circuit-based service or a shared best-effort service. The former usually caries some guarantees. The latter offers almost no guarantees. Most service differentiation has been in the pricing structure (individual vs. business rates) or the connectivity type (dial-up access vs. leased line, etc.). However, in recent years, increased usage of the Internet has reduced network over-capacity, and in proliferation of applications that require some form of bandwidth, delay or jitter guarantees. As a result, service providers are finding it necessary to offer their customers alternative levels of service. As well as meeting new customer expectations, this allows service providers to improve their revenues through premium pricing and competitive differentiation of service offerings, which in turn can fund the necessary expansion of the network [10].

1.3. What is DiffServ?

The Differentiated Services architecture offers a framework within which service providers can offer each customer a range of network services which are differentiated on

the basis of performance in addition to pricing tiers used in the past [12] [15] [17] [21]. Customers request a specific performance level on a packet by packet basis by marking the DiffServ field of each packet with a specific value . This value specifies the Per-Hop Behavior (PHB) to be allotted to the packet within the provider's network. Typically, the customer and provider negotiate a profile (policing profile) describing the rate at which traffic can be submitted at each service level. Packets submitted in excess of this profile may not be allotted the service level requested. An especially interesting feature of differentiated services is its potential for scalability. This allows DiffServ to be deployed in very large networks that share a per-domain behavior (PDB) and/or policies,. The scalability is achieved by forcing as much of the complexity as possible out of the core of the network and into the edge devices. Individual edge devices process lower volumes of traffic, and lesser numbers of flows, than the core, but offer richer services on per-micro-flow basis.

1.4. Some Issues

Although many IETF RFCs and Drafts describe the DiffServ basic architecture, none of these publication describe, or recommend, a certain implementation for DiffServ services. As a result, the implementation of DiffServ components, such as Per Hop Behavior (PHB) and Per Domain behavior, is an open issue left to the implementers. The work reported here tries to explore this area. It tries to find an acceptable way to implement some of the DiffServ PHB's and PDB's, and use them as building blocks to construct a generic sort of Serive Level Agreements (SLA).

The approach is to build a DiffServ test bed and then conduct a set of SLA experiments. The SLA testbed is based on the implementation of soft DiffServ router in Linux done, in part, by Narasimhan [18]. Present work extends some of the functionalities of this implementation in the Buffer management and packet scheduling areas. It also carried out a thorough verification test of the soft router [41] components, such as classifier, buffer manager, packet scheduler, etc separately, and as part of a system of routers. This soft router was used as the building block for a DiffServ-based SLA test discussed in this thesis. Some of the reasons for construction of this test-bed are a) it was more flexible

than a commercially available routers, b) it has more features than commercially available equipment in the area of algorithms of interest, c) it is easy to change both the existing algorithms and add new ones, d) it is easy to instrument so that data specific to an experiment can be collected, e) it is considerably less expensive than a similar suite of commercial routers.

1.5. Thesis Layout

In chapter 2 this thesis introduces the general architecture of the differentiated services and the specifics of the soft, Linux-based, implementation on which the experiments were run on. Chapter 3 talks about provisioning of differentiated services based on the IETF RFCs and the drafts currently proposed. It also discusses some of the efforts made in implementation of DiffSev services. Chapter 4 discusses the implementation of per-hop Expedited Forwarding (EF) and per-domain Virtual Wire support using a Linux-based five node soft DiffServ router testbed developed at NC State. In evaluating the implementation, both generated and real VoIP streams were used. Chapter 5 presents an implementation and experimental evaluation of Assured Rate PHB and PDB. Chapter 6 offers conclusions and suggestions for future work,. References are in Section 7, and the Appendices contain some specific information related to the experiments and the solutions discussed in the thesis.

2. DiffServ Architecture

2.1. Model

The differentiated services architecture is based on a simple idea [10][12][15][19][21]. Traffic entering a network is classified, and possibly conditioned, at the boundaries of the network, and then it is assigned to different core behavior aggregates. Thus, in theory, DiffServ flows are policed and marked at the first DiffServ sensitive ingress node, according to a negotiated Service Level Agreement (SLA) or contract which also specifies the traffic profile. Then, subsequent nodes (e.g., core routers) deal only with aggregated traffic. This eliminates the need to recognize and store information about each individual flow in the network core. On exit from the network different traffic flows are de-aggregated and distributed to end-user nodes. The main purpose of DiffServ is to provide a scalable framework for offering a range of services in the Internet with Quality of Service (QoS) support and without the need to maintain per-flow state in every network node in the core (e.g., router).

The actual marking of the traffic flows is achieved through the type-of-service (TOS) field of the IP packet. In the DiffServ incarnation, this field contains a 6 bit codepoint called the DiffServ codepoint (DSCP). The DSCP field determines the per-hop behavior (PHB) that the flow should receive. The PHBs define the forwarding behavior for the flow at that particular network node [15][21]. The out-of-profile packets are either dropped or marked with a different PHB by the ingress router. The ingress router classifies traffic into aggregates based on DSCP. This is then policed according to the aggregate profiles. In an attempt to satisfy some QoS requirements for the flows. The QoS, and more generally SLA, requirements may be specified in quantitative or statistical terms of, for example, throughput, delay, jitter, and loss, or may otherwise be specified in terms of some relative priority or cost of access to network resources.

As mentioned before, DiffServ enabled network achieves scalability by aggregating traffic flows into more manageable flows, and processes these aggregates instead of

individual flows. In its most basic form, DiffServ provides service differentiation in one direction of traffic flow, and therefore it is asymmetric. Symmetric DiffServ is achievable, but requires additional resources.

2.2. DiffServ Domain

A DiffServ Domain [10][12] is a contiguous set of DiffServ nodes which operate with a common service provisioning policy, and set of PHB groups implemented on each node. A DiffServ domain has a well-defined boundary consisting of DiffServ boundary nodes which classify and possibly condition ingress traffic to ensure that packets which transit the domain are appropriately marked to select a PHB from one of the PHB groups supported within the domain. Nodes within the DiffServ domain select the forwarding behavior for the packets based on their DiffServ codepoint. That value is then mapped to one of the supported PHBs using either the recommended codepoint to PHB mapping, or a locally customized mapping.

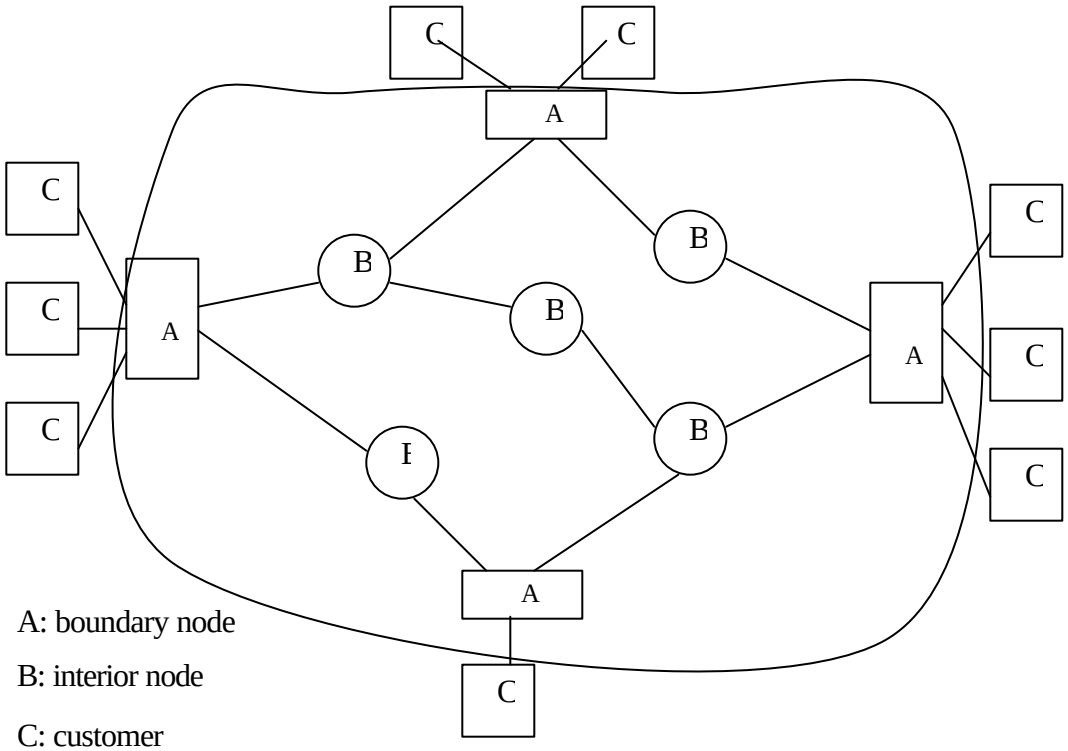


Figure 1. Illustration of a DiffServ Domain

Figure 1 illustrates a DiffServ domain. It shows that a domain consists of a set of DiffServ capable nodes governed by a common policy for provisioning services. Each DiffServ domain has an ingress DiffServ node that provides fine-grained classification. This node selects a PHB to be associated with the flow and conditions the flow if necessary. DiffServ domain consists basically of boundary nodes and interior nodes. DiffServ boundary nodes interconnect the DiffServ domain to other DiffServ or non-DiffServ -capable domains. DiffServ interior nodes only connect to other DiffServ interior or boundary nodes within the same DiffServ domain. Both DiffServ boundary nodes and interior nodes must be able to apply the appropriate PHB to packets based on the DiffServ codepoint. In addition, DiffServ boundary nodes may be required to perform traffic conditioning functions as defined by a traffic conditioning agreement (TCA) between their DiffServ domain and the peering domain that they connect to. DiffServ boundary nodes act both as a DiffServ ingress node and a DiffServ egress node for different directions of traffic .

Traffic enters a DiffServ domain at a DiffServ ingress node and leaves a DiffServ domain at a DiffServ egress node. A DiffServ ingress node is responsible for ensuring that the traffic entering the DiffServ domain conforms to any TCA between it and the other domain to which the ingress node is connected. A DS egress node may perform traffic conditioning functions on traffic forwarded to a directly connected peering domain, depending on the details of the TCA between the two domains [18].

2.3. Per-Hop Behaviors (PHB)

The PHB is the means by which a node allocates resources to behavior aggregates. The observable behavior of a PHB may depend on certain constraints on the traffic characteristics of the associated behavior aggregate (BA) or the characteristics of other BAs. PHBs may be specified in terms of their resource (e.g., buffer, bandwidth) priority relative to other PHBs, or in terms of their relative observable traffic characteristics (e.g., delay, loss).

The PHBs are implemented in nodes by means of some buffer management and packet scheduling mechanisms. In general, a variety of implementation mechanisms may be suitable for implementing a particular PHB group. A PHB is selected at a node by a mapping of the DiffServ codepoint in a received packet. Standardized PHBs have recommended codepoint. These PHBs may be used as building blocks to allocate resources and should be specified as a group (PHB group) for consistency. PHB groups will usually share a common constraint applying to each PHB within the group. PHB groups should be defined such that the proper resource allocation between groups can be inferred, and integrated mechanisms can be implemented which can simultaneously support two or more groups. All codepoints must be mapped to some PHB and codepoints that are not mapped to a standardized PHB should be mapped to a Default PHB.

2.4. Basic Components

The following DiffServ elements are based on the architecture of the soft, Linux-based, DiffServ router implementation [18] that was used in the test-bed constructed to conduct experiments reported in this work. A "logical" DiffServ router is illustrated in Figure 2. It consists of the following basic logical components:

- Classifiers
- Traffic Conditioners,
- Meters
- Buffer Manager
- Shapers
- Link Schedulers

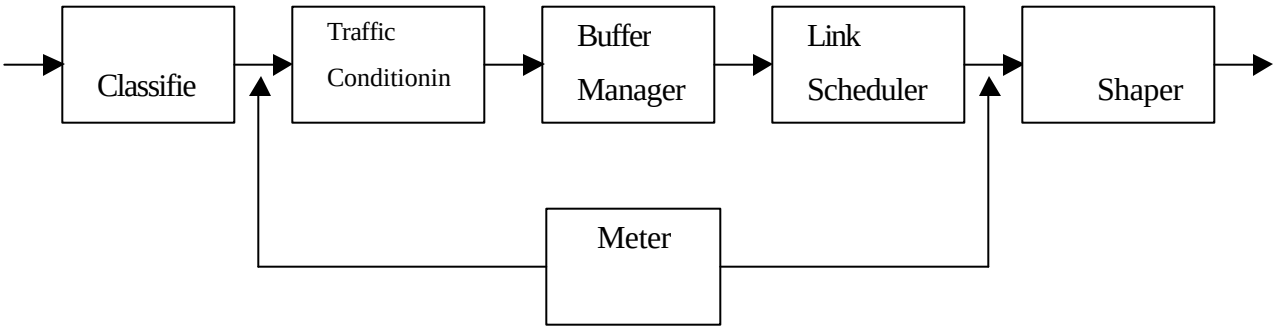


Figure 2. Basic components of a DiffServ Router

2.5. Classifiers

A classifier selects packets based on certain fields of the packet header and according to certain rules. The packets are identified to belong to some flow defined by a flow ID. They are then passed on to a traffic conditioner defined for that particular flow. There are two types of packet classifiers defined in the DiffServ architecture –behavioral aggregate classifier and multi-field classifier.

2.5.1. Behavioral Aggregate Classifier

A BA classifier classifies packets based on the DSCP field only. Since it separates packets based on just the DSCP field, it becomes difficult to classify packets by customer, particularly if they come through the same interface. Hence, it is impossible to enforce the TCAs on a per-customer basis using a BA classifier. Multi Field classifiers are preferred for such cases.

2.5.2. Multi Field Classifier

A Multi-Field (MF) classifier classifies packets based on combinations of various fields of the IP packet header. These fields may be the destination address, source address, DSCP, protocol ID, source port number and the destination port number. At minimum, a boundary router must support behavioral-aggregate classification. In addition, routers may support varying degrees of multi-field classification. This permits fine-grained classification.

2.6. Meters

A meter is a monitoring interface that enables collection of statistics regarding traffic carried at various DiffServ levels. These statistics are important for accounting purposes and for tracking compliance to service level agreements negotiated with customers. Specifically, counter information on how many packets/bytes were transferred in-profile vs. out-of-profile would be useful on a customer-by-customer basis. This same information should be provided for the coarser granularity DSCPs all the way down to the finer granularity flow-by-flow profiling where applicable. Boundary routers use classifiers to identify classes of traffic submitted for transmission through the DiffServ network. Once traffic is classified at the input to the router, traffic from each class is typically passed to a meter. The meter is used to measure the rate at which traffic is being submitted for each class. This rate is then compared against a traffic profile, which is part of the TCA. Based on the results of the comparison, the meter deems particular packets to be conforming to the profile or non-conforming. Appropriate policing actions are then applied to out-of-profile packets.

2.7. Traffic Conditioner

A traffic conditioner (TC) is used to make sure the traffic conforms to the negotiated profile. The traffic conditioner may be a marker, dropper or shaper. A traffic profile represents the temporal properties of the traffic stream. The traffic conditioner uses a meter and determines whether the packet is in profile or out of profile. It may decide to drop or re-queue an out of profile packet and normally queues an in-profile packet. The condition-action parameters are decided by the service provider and may be different at each DiffServ router.

2.7.1. Marker

The marker is used to set the DS Code Point based on certain rules [18]. These rules depend on the classification of packet and/or if it conforming or non-conforming to its traffic profile.

2.7.2. Shaper

Shapers delay packets passing through the router such that they are brought into compliance with a traffic profile. Boundary routers are not required to provide shaping functionality, but may do so for the following reasons:

1. Ingress routers may use shaping as a form of policing - when submitted traffic is deemed non-conforming, it must be policed to protect the DiffServ network. One form of policing is to delay submitted traffic in a shaper until it conforms to the profile specified in the TCA. This is usually referred to as policer shaping.
2. Egress routers may shape behavior aggregate traffic before it is submitted to a subsequent provider's network. This preventative measure avoids policing action in the subsequent network. This is usually referred to as egress shaping.

2.7.3. Droppers

Droppers are used to discard non-conforming packets. This is the simplest form of policing that may be supported by ingress routers.

2.8. Buffer Manager

The main function of a buffer manager is to manage the queues. The two main aspects to buffer management are :

- Queue Selection
- Congestion Control

2.8.1. Queue Selection

The issue here is regarding selection of a queue for the packet. The strategies adopted for this include Class Based Queuing (CBQ) and Microflow Based Queueing. The main goal of CBQ is to aggregate traffic into classes. Use of this class-hierarchy enables link sharing too. When the classification is more fine-grained and thus we have more classes, it becomes a microflow-based queuing.

2.8.2. Congestion Control

The other aspect of buffer management deals with the issue of controlling congestion. An active queue management strategy helps in controlling congestion. The buffer manager controls the length of the queues by dropping packets at appropriate times. Queues are an integral part of any network as they are required to handle traffic bursts. But long queues can increase the delay in a network. Hence the need for effective queue management strategies.

2.9. Link Scheduler

A scheduler is an element which gates the departure of each packet that arrives at one of its inputs, based on a scheduling algorithm. It has one or more input and exactly one output. Every input has an upstream element to which it is connected, and a set of parameters that affects the scheduling of packets received at that input A scheduler should meet the following requirements :

- Isolation of Flows : The scheduler should be able to isolate misbehaving flows.
- Low Delays : The scheduler should not introduce additional delays in the packet.
- Utilization : The scheduler should not waste its resources.
- Fairness : The scheduler should not unduly favor one connection over the other when there are additional resources available.
- Simplicity : The scheduling algorithm must be fast, simple and easy to implement.
- Scalability : The scheduler must scale well for a large number of connections.

2.10. Soft Linux-Based DiffServ Test-Bed

Based on implementation of soft DiffServ router in linux work done by Narasimhan [18], we extend some of the functionalities of his implementation, such as Buffer management and packet scheduling, and carried out a thorough verification test for the router [41] by testing and verifying the functionality of each component, such classifier, buffer manager, packet scheduler, etc., separately. Then we used this router as the building block for a DiffServ test bed where we conduct our SLA experiments .

The DiffServ Test Bed was built using five (5) soft DiffServ Routers connected to simulate an ingress edge router , a set of core routers and egress edge router as shown in Figure 3.

Note that two traffic generators were part of the test-bed, NetCom SmartBits[1] A and B, each with four 10/100 Mbps Ethernet interfaces. The five routers were configured into eleven sub nets. Each Linux router has three or more networking cards. Routers were either

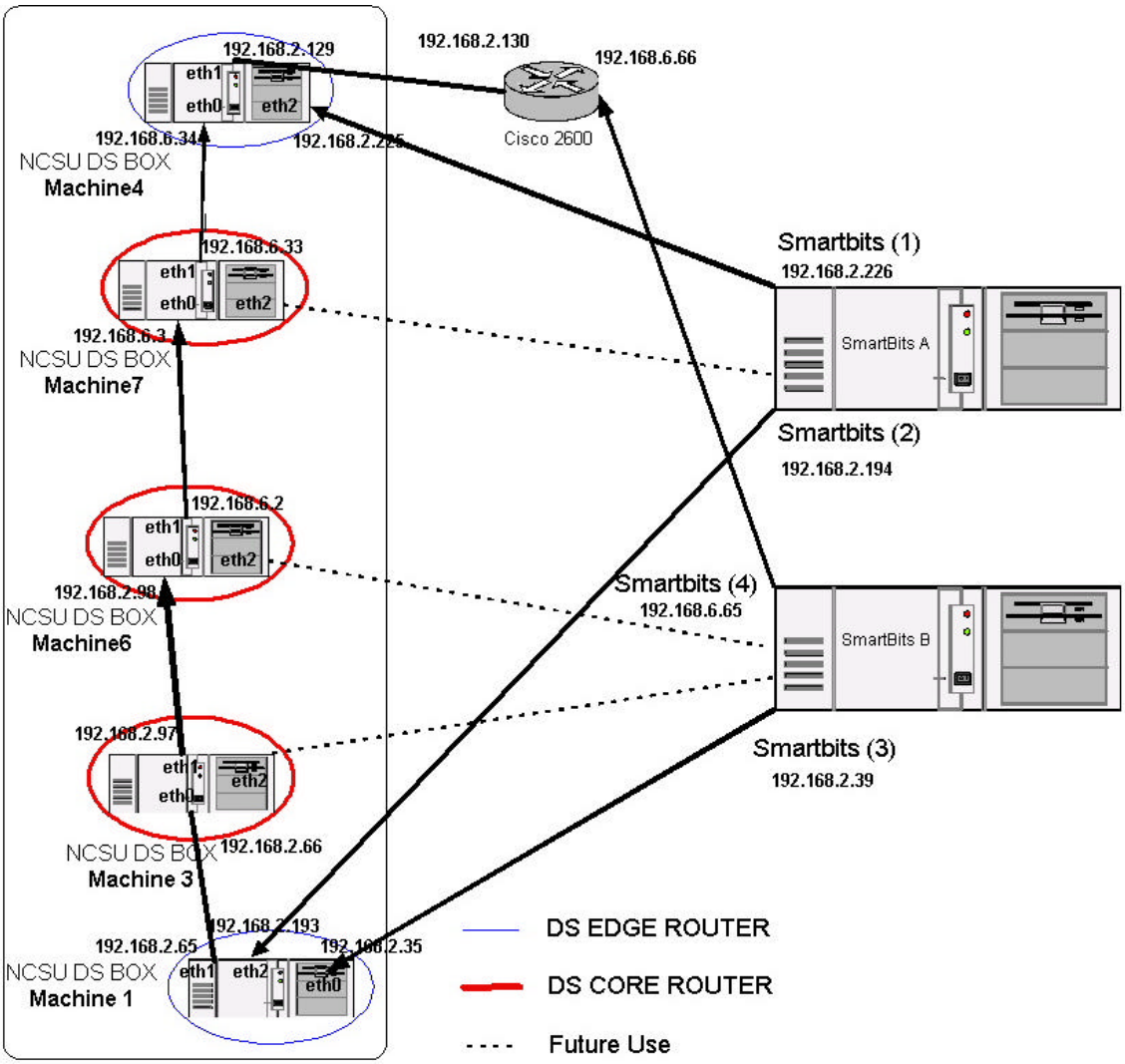


Figure 3. Basic DiffServ Test Bed Configuration

[1] SmartBits is a registered trademark of Spirent Communications.

directly connected to each other, or through switches. The interconnection of routers with the SmartBits was designed to simulate as many traffic sources and sinks as possible. The basic DiffServ test-bed was extended with SLA supporting scripts and processes (e.g., EF and AF PHBs and PDBs) the suitability of which is the subject of the present work.

2.11. Ns Simulator

2.11.1. Introduction

Ns-based [45] scripts were written to simulate our DiffServ test bed and repeat and validate some of the EF experiments using the simulation. Sample ns script is shown in appendix D. Ns is a very popular simulator in networking research. Ns is open-source freeware, and it is constantly maintained and updated by its large user base, and a small group of developers. According to [45]:

"Ns is a discrete event simulator targeted at networking research. Ns provides substantial support for simulation of IPv4, UDP, TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. More recently MPLS and Ipv6 modules have become available. Ns began as a variant of the REAL network simulator in 1989. It has evolved substantially since then. In 1995 ns development was supported by DARPA through the VINT project at LBL, Xerox PARC, UCB, and USC/ISI. Currently ns development is support through DARPA with SAMAN and through NSF with CONSER, both in collaboration with other researchers including ACIRI. Ns has always included substantial contributions from other researchers, including wireless code from the UCB Daedelus and CMU Monarch projects and Sun Microsystems.

Ns is an object oriented simulator, written in C++, with an OTcl interpreter as a front end. The simulator supports a class hierarchy in C++, and a similar class hierarchy within the OTcl interpreter. The two hierarchies are closely related to each other; from the user's perspective, 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; these other hierarchies are not mirrored in the manner of Tcl Object.

Ns uses two languages because simulator has two different kinds of things it needs to do. On one hand, detailed simulations of protocols requires a systems programming language which can efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets. For these tasks run-time speed is important and turn-around time (run simulation, find bug, fix bug, recompile, re-run) is less important.

On the other hand, a large part of network research involves slightly varying parameters or configurations, or quickly exploring a number of scenarios. In these cases, iteration time (change the model and re-run) is more important. Since configuration runs once (at the beginning of the simulation), run-time of this part of the task is less important.

Ns meets both of these needs with two languages, C++ and OTcl. C++ is fast to run but slower to change, making it suitable for detailed protocol implementation. Otcl runs much slower but can be changed very quickly (and interactively), making it ideal for simulation configuration. (via tclcl) provides glue to make objects and variables appear on both languages."

2.11.2. Ns Capabilities

Ns can Create :

- Traffic sources like web, ftp, telnet, cbr, stochastic traffic.
- Various queuing disciplines (drop-tail, RED, FQ, SFQ,DRR, etc.) and QoS (e.g., IntServ and DiffServ).

- Terrestrial, satellite and wireless networks with various routing algorithms (DV, LS, PIM-DM, PIM-SM, AODV,DSR).

Ns can Visualize:

- Packet flow, queue build up and packet drops.
- Protocol behavior: TCP slow start, self-clocking, congestion control, fast retransmit and recovery.
- Node movement in wireless networks.
- Annotations to highlight important events.
- Protocol state (e.g., TCP cwnd).

2.11.3. Ns DiffServ implementation

The ns DiffServ architecture [45] provides QoS by dividing traffic into different categories, marking each packet with a code point that indicates its category, and scheduling packets according to their code points. The DiffServ module in ns currently defines four classes of traffic, each of which has three drop precedences. Those drop precedences enable differential treatment of traffic within a single class. A single class of traffic is enqueued into one corresponding physical RED queue, which contains three virtual queues (one for each drop precedence). Different RED parameters are used for the virtual queues, causing packets from one virtual queue to be dropped more frequently than packets from another. A packet with a lower drop precedence is given better treatment in times of congestion because it is assigned a code point that corresponds to a virtual queue with relatively lenient RED parameters.

According to [45] *"The DiffServ module in ns has three major components:*

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 code point according to the policy specified.*

core routers*: Core routers examine packets' code point marking and forward them accordingly.*

The DiffServ functionality in ns is captured in a Queue object. A DiffServ queue (dsREDQueue) derived from the base class Queue, contains the abilities:

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

Six different policy models are defined in ns , which are:

- *TSW2CM (TSW2CMPolicer): uses a CIR and two drop precedences. The lower precedence is used probabilistically when the CIR is exceeded.*
- *TSW3CM (TSW3CMPolicer): uses a CIR, a PIR, and three drop precedences. The medium drop precedence is used probabilistically when the CIR is exceeded and the lowest drop precedence is used probabilistically when the PIR is exceeded.*
- *Token Bucket (tokenBucketPolicer): uses a CIR and a CBS and two drop precedences. An arriving packet is marked with the lower precedence if and only if it is larger than the token bucket.*
- *Single Rate Three Color Marker (srTCMPolicer): uses a CIR, CBS, and an EBS to choose from three drop precedences.*
- *Two Rate Three Color Marker (trTCMPolicer): uses a CIR, CBS, PIR, and a PBS to choose from three drop precdences.*

Scheduling modes supported are Weighted Interleaved Round Robin (WIRR), Round Robin (RR), and Priority (PRI). For Priority scheduling, priority is arranged in sequential order with queue 0 having the highest priority."

3. Service Provisioning

3.1. SLAs, SLSs and TCSs

Service, as defined by IETF [10], is the overall treatment of a defined subset of a customer's traffic within a DS-domain, or end-to-end".

Although PHBs are at the heart of the differentiated services architecture, it is the service obtained as a result of marking traffic for a specific PHB, which is of value to the customer. PHBs are merely building blocks for services. Service providers combine PHB implementations with traffic conditioners, provisioning strategies and billing models which enable them to offer services to their customers. Providers and customers negotiate agreements with respect to the services to be provided at each customer / provider boundary. These take the form of Service Level Agreements (**SLAs**).

At each differentiated service customer/provider boundary, the technical aspects of the service provided is defined in the form of an Service Level Specification **(SLS)** which specifies the overall features and performance which can be expected by the customer. Because DS services are unidirectional the two directions of flow across the boundary will need to be considered separately. An important subset of the SLS is the Traffic Conditioning Specification, or (**TCS**). The TCS specifies detailed service parameters for each service level. Such parameters include:

1. Detailed service performance parameters such as expected throughput, drop probability, latency.
2. Constraints on the ingress and egress points at which the service is provided, indicating the `scope' of the service. Service scopes are discussed later.
3. Traffic profiles which must be adhered to for the requested service to be provided, such as token bucket parameters.
4. Disposition of traffic submitted in excess of the specified profile.
5. Marking services provided.
6. Shaping services provided.

In addition to the details in the TCS, the SLS may specify more general service characteristics such as:

1. Availability/Reliability, which may include behavior in the event of failures resulting in rerouting of traffic
2. Encryption services
3. Routing constraints
4. Authentication mechanisms
5. Mechanisms for monitoring and auditing the service
6. Responsibilities such as location of the equipment and functionality, action if the contract is broken, support capabilities
7. Pricing and billing mechanisms

3.1.1. Qualitative and Quantitative services [10]

Some services can be clearly categorized as qualitative or quantitative depending on the type of performance parameters offered. Examples of qualitative services are as follows:

1. Traffic offered at service level A will be delivered with low latency.
2. Traffic offered at service level B will be delivered with low loss.

The assurances offered in examples 1 and 2 are relative and can only be verified by comparison.

Examples of quantitative services are as follows:

3. 90% of in profile traffic delivered at service level C will experience no more than 50 msec latency.
4. 95% of in profile traffic delivered at service level D will be delivered.

3.1.2. The Scope of a Service [10]

The scope of a service refers to the topological extent over which the service is offered. For example, assume that a provider offers a service to a customer which connects to their network at ingress point A. The service may apply to:

1. all traffic from ingress point A to any egress point

2. all traffic between ingress point A and egress point B

3. all traffic from ingress point A to a set of egress points

3.2. An Expedited Forwarding PHB [21]

The PHB (per-hop behavior) is a basic building block in the Differentiated Services architecture. IETF defines a PHB called Expedited Forwarding (EF). EF is intended to provide a building block for low delay and low loss services by ensuring that the EF aggregate is served at a certain configured rate.

The dominant causes of delay in packet networks are speed-of-light propagation delays on wide area links and queuing delays in switches and routers. Since propagation delays are a fixed property of the topology, delay and jitter are minimized when queuing delays are minimized. In this context, jitter is defined as the variation between maximum and minimum delay. The intent of the EF PHB is to provide a PHB in which suitably marked packets usually encounter short or empty queues. Furthermore, if queues remain short relative to the buffer space available, packet loss is also kept to a minimum.

To ensure that queues encountered by EF packets are usually short, it is necessary to ensure that the service rate of EF packets on a given output interface exceeds their arrival rate at that interface over long and short time intervals, independent of the load of other (non-EF) traffic. This specification defines a PHB in which EF packets are guaranteed to receive service at or above a configured rate and provides a means to quantify the accuracy with which this service rate is delivered over any time interval. It also provides a means to quantify the maximum delay and jitter that a packet may experience under bounded operating conditions.

Note that the EF PHB only defines the behavior of a single node. The specification of behavior of a collection of nodes is defined by Per-Domain Behavior (PDB) which we will be describe later.

Intuitively, the definition of EF is simple: the rate at which EF traffic is served at a given output interface should be at least the configured rate R, over a suitably defined interval, independent of the offered load of non-EF traffic to that interface.

Two difficulties arise when we try to formalize this intuition:

- it is difficult to define the appropriate time scale at which to measure R. By measuring at short time scales we may introduce sampling errors; at long time scales we may allow excessive jitter.

- EF traffic clearly cannot be served at rate R if there are no EF packets waiting to be served, but it may be impossible to determine externally whether EF packets are actually waiting to be served by the output scheduler. For example, if an EF packet has entered the router and not exited, it may be awaiting service, or it may simply have encountered some processing or transmission delay within the router.

3.2.1. EF Support Requirements

A node that supports EF on an interface I at some configured rate R MUST satisfy the following equations:

$$d_j <= f_j + E_a \tag{3.1}$$

where f_j is defined iteratively by

$f_0 = 0$, $d_0 = 0$

$$f_j = \max(a_j, \min(d_{j-1}, f_{j-1})) + \frac{l_j}{R} \qquad \text{for all } j > 0 \tag{3.2}$$

where:

- d_j is the time that the last bit of the j-th EF packet to depart actually leaves the node from the interface I.
- f_j is the target departure time for the j-th EF packet to depart from I, the "ideal" time at or before which the last bit of that packet should leave the node.
- a_j is the time that the last bit of the j-th EF packet destined to the output I to arrive actually arrives at the node.
- l_j is the size (bits) of the j-th EF packet to depart from I.

- l_j is measured on the IP datagram (IP header plus payload) and does not include any lower layer (e.g. MAC layer) overhead.
- R is the EF configured rate at output I (in bits/second).
- E_a is the error term for the treatment of the EF aggregate. Note that E_a represents the worst case deviation between actual departure time of an EF packet and ideal departure time of the same packet, i.e., E_a provides an upper bound on $(d_j - f_j)$ for all j.
- d_0 and f_0 do not refer to a real packet departure but are used purely for the purposes of the recursion. The time origin should be chosen such that no EF packets are in the system at time 0.

E_a may be thought of as "figure of merit" for a device. A smaller value of E_a means that the device serves the EF aggregate more smoothly, at rate R over relatively short timescales, whereas a larger value of E_a implies a more bursty scheduler which serves the EF aggregate at rate R only when measured over longer intervals.

3.2.2. Implementation Examples [21][46]

A priority queue is widely considered as the canonical example of an implementation of EF. A "perfect" output buffered device (i.e. one which delivers packets immediately to the appropriate output queue) with a priority queue for EF traffic will provide both a low delay and low E_a . We note that the main factor influencing E_a will be the inability to pre-empt an MTU-sized non-EF packet that has just begun transmission at the time when an EF packet arrives at the output interface, plus any additional delay that might be caused by non-preemptive queues between the priority queue and the physical interface.

Another example of an implementation of EF is a weighted round robin scheduler. Such an implementation will typically not be able to support values of R as high as the link speeds because the maximum rate at which EF traffic can be served in the presence of competing traffic will be affected by the number of other queues and the weights given to them. Furthermore, such an implementation is likely to have a value of E_a that is

higher than a priority queue implementation, all else being equal, as a result of the time spent serving non-EF queues by the round robin scheduler.

3.2.3. Figure of Merit (E) for different implementations [21][46]

Let us consider an ideal output buffered node where packets entering the device from an input interface are immediately delivered to the output scheduler. In this model the properties of the output scheduler fully define the values of the parameters E_a .

Strict Non-preemptive Priority Queue

A Strict Priority scheduler in which all EF packets share a single FIFO queue which is served at strict non-preemptive priority over other queues satisfies the EF definition with the latency term E = MTU/C where MTU is the maximum packet size and C is the speed of output link.

WF2Q

Another scheduler that satisfies the EF definition with a small latency term is WF2Q.

A class-based WF2Q scheduler, in which all EF traffic shares a single queue with the weight corresponding to the configured rate of the EF aggregate satisfies the EF definition with the latency term E = MTU/C+MTU/R.

Deficit Round Robin (DRR)

For DRR [7], both E can be shown to grow linearly with N*(r_max/r_min)*MTU, where r_min and r_max denote the smallest and the largest rate among the rate assignments of all queues in the scheduler, and N is the number of queues in the scheduler .

Start-Time Fair Queuing (SFQ) and Self-Clocked Fair Queuing (SCFQ)

For SCFQ [14] E can be shown to grow linearly with the number of queues in the scheduler.

3.3. Assured Forwarding PHB Group [15]

3.3.1. Introduction

There is a demand to provide assured forwarding of IP packets over the Internet. In a typical application, a company uses the Internet to interconnect its geographically distributed sites and wants an assurance that IP packets within this intranet are forwarded with high probability as long as the aggregate traffic from each site does not exceed the subscribed information rate (profile). It is desirable that a site may exceed the subscribed profile with the understanding that the excess traffic is not delivered with as high probability as the traffic that is within the profile. It is also important that the network does not reorder packets that belong to the same microflow, as defined in [27], no matter if they are in or out of the profile.

3.3.2. Assured Forwarding PHB

Assured Forwarding (AF) PHB [15] group is a means for a provider DS domain to offer different levels of forwarding assurances for IP packets received from a customer DS domain. Four AF classes are defined, where each AF class is in each DS node allocated a certain amount of forwarding resources (buffer space and bandwidth). IP packets that wish to use the services provided by the AF PHB group are assigned by the customer or the provider DS domain into one or more of these AF classes according to the services that the customer has subscribed to.

Within each AF class IP packets are marked (again by the customer or the provider DS domain) with one of three possible drop precedence values. In case of congestion, the drop precedence of a packet determines the relative importance of the packet within the AF class.

A congested DS node tries to protect packets with a lower drop precedence value from being lost by preferably discarding packets with a higher drop precedence value.

A DS node MUST NOT reorder AF packets of the same microflow when they belong to the same AF class regardless of their drop precedence. There are no quantifiable timing requirements (delay or delay variation) associated with the forwarding of AF packets.

3.3.3. AF PHB implementation

An AF implementation MUST attempt to minimize long-term congestion within each class, while allowing short-term congestion resulting from bursts. This requires an active queue management algorithm. An example of such an algorithm is Random Early Drop (RED) [13].

An AF implementation MUST detect and respond to long-term congestion within each class by dropping packets, while handling short-term congestion (packet bursts) by queuing packets.

The dropping algorithm MUST be insensitive to the short-term traffic characteristics of the microflows using an AF class. The dropping algorithm MUST treat all packets within a single class and precedence level identically.

3.3.4. AF PHB Services Examples [15]

The AF PHB group could be used to implement, for example, the so called Olympic service, which consists of three service classes:

bronze, silver, and gold. Packets are assigned to these three classes so that packets in the gold class experience lighter load (and thus have greater probability for timely forwarding) than packets assigned to the silver class. Same kind of relationship exists between the silver class and the bronze class. If desired, packets within each class may be further separated by giving them either low, medium, or high drop precedence.

The bronze, silver, and gold service classes could in the network be mapped to the AF classes 1, 2, and 3. Similarly, low, medium, and high drop precedence may be mapped to AF drop precedence levels 1, 2, or 3.

The drop precedence level of a packet could be assigned, for example, by using a leaky bucket traffic policer, which has as its parameters a rate and a size, which is the sum of two burst values: a committed burst size and an excess burst size. A packet is assigned low drop precedence if the number of tokens in the bucket is greater than the excess burst size, medium drop precedence if the number of tokens in the bucket is greater than zero, but at most the excess burst size, and high drop precedence if the bucket is empty. It may also be necessary to set an upper limit to the amount of high drop precedence traffic from

a customer DS domain in order to avoid the situation where an avalanche of undeliverable high drop precedence packets from one customer DS domain can deny service to possibly deliverable high drop precedence packets from other domains.

Another way to assign the drop precedence level of a packet could be to limit the user traffic of an Olympic service class to a given peak rate and distribute it evenly across each level of drop precedence.

This would yield a proportional bandwidth service, which equally apportions available capacity during times of congestion under the assumption that customers with high bandwidth microflows have subscribed to higher peak rates than customers with low bandwidth microflows.

The AF PHB group could also be used to implement a loss and low latency service using an over provisioned AF class, if the maximum arrival rate to that class is known a priori in each DS node.

3.4. Per Domain Behaviors (PDBs)

The differentiated services framework enables quality-of-service provisioning within a network domain by applying rules at the edges to create traffic aggregates and coupling each of these with a specific forwarding path treatment in the domain through use of a codepoint in the IP header .

The DiffServ WG has defined the general architecture for differentiated services and has focused on the forwarding path behavior required in routers, known as "per-hop forwarding behaviors" (or PHBs).

The next step is to formulate examples of how forwarding path components (PHBs, classifiers, and traffic conditioners) can be used to compose traffic aggregates whose packets experience specific forwarding characteristics as they transit a differentiated services domain.

The WG has decided to use the term per-domain behavior, or PDB, to describe the behavior experienced by a particular set of packets as they cross a DS domain. A PDB is characterized by specific metrics that quantify the treatment a set of packets with a particular DSCP (or set of DSCPs) will receive as it crosses a DS domain.

A PDB specifies a forwarding path treatment for a traffic aggregate and, due to the role that particular choices of edge and PHB configuration play in its resulting attributes, it is where the forwarding path and the control plane interact. The measurable parameters of a PDB should be suitable for use in Service Level Specifications at the network edge.

3.5. The `Virtual Wire' (VW) Per-Domain Behavior [19]

3.5.1. Introduction

A Virtual Wire (VW) PDB is intended to send "circuit replacement" traffic across a DiffServ network. That is, this PDB is intended to mimic, from the point of view of the originating and terminating nodes, the behavior of a hard-wired circuit of some fixed capacity.

This PDB should be suitable for any packetizable traffic that currently uses fixed circuits (e.g., telephony, telephone trunking, broadcast video distribution, leased data lines) and packet traffic that has similar delivery requirements (e.g., IP telephony or video conferencing).

`Virtual Wire' (VW) can be constructed in any domain supporting the DiffServ EF PHB plus appropriate domain ingress policers. Although one attribute of VW is the delivery of a peak rate, in VW this is explicitly coupled with a bounded jitter attribute. Network hardware has become sufficiently reliable that the overwhelming majority of network loss, latency and jitter are due to the queues traffic experiences while transiting the network. Therefore providing low loss, latency and jitter to a traffic aggregate means ensuring that the packets of the aggregate see no (or very small) queues.

Creating the VW PDB involves:

- Configuring individual nodes so that the aggregate has a well-defined minimum departure rate.
- Conditioning the entire DS domain's aggregate (via policing and shaping) so that its arrival rate at any node is always less than that node's configured minimum departure rate.

3.5.2. VW Jitter Bound [19]

The VW PDB has two major attributes: an assured peak rate and a bounded jitter, and it uses the EF PHB to implement a transit behavior with the required attributes.

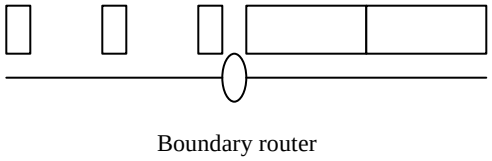


Figure 4. Time of a CBR stream at a high to low bandwidth transition

Figure 4 shows a Constant Bit rate CBR stream of size *S* packets being sourced at rate R. At the domain egress border router, the packets arrive on a link of bandwidth *B (= n*R) and depart to their destination on a link of bandwidth R.

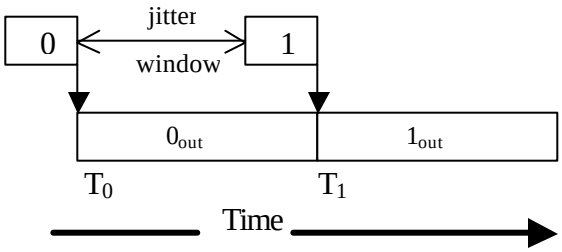


Figure 5. Details of arrival / departure relationships

Figure 5 shows the detailed timing of events at the router. At time T_0 the last bit of packet 0 arrives so output is started on the egress link. It will take until time for packet 0 to be completely output. As long as the last bit of packet 1 arrives at the border router before , the destination node will find the traffic indistinguishable from a stream carried

the entire way on a dedicated wire of bandwidth R. This means that packets can be *jittered* or displaced in time (due to queue waits) as they cross the domain and that there is a *jitter window* at the border router of duration

$$\Delta = S/R - S/B = (S/R) * (n-1)/n \qquad (3.3)$$

that must bound the sum of all the queue waits seen be a packet as it transits the domain. As long as this sum is less than , the destination will see service identical to a dedicated wire. Note that the jitter window is (implicitly) computed relative to the first packet of the flight of packets departing the boundary router and, thus, can only include *variable* delays. Any transit delay experienced by all the packets, be it propagation time, router forwarding latency, or even average queue waits, is removed by the relative measure so the sum described in this paragraph is not sensitive to *delay* but only to *delay variatio*n. Also note that when packets enter the domain they are already separated by so, effectively, everything is pushed to the left edge of the jitter window and there's no slack time to absorb delay variation in the domain. However by simply delaying the output of the first packet to arrive at E by one packet time *(S/*R), the phase reference for all the traffic is reset so that all subsequent packets enter at the right of their jitter window and have maximum slack time during transit.

The preceding derives the jitter window in terms of the bandwidths of the ingress circuit and intra-domain links. In practice, the 'givens' are more likely to be the intra-domain link bandwidths and the jitter window (which can be no less than the EF bound associated with each output link that might be traversed by some VW flow(s)). Rearranging equation based on this gives *R*, the maximum amount of bandwidth that can be allocated to the aggregate of all VW circuits, as a function of the EF bound (i.e., jitter window):

$$R = \frac{S}{\Delta} * \frac{n-1}{n} \qquad (3.4)$$

Note that the upper bound on VW traffic that can be handled by any output link is simply the MTU divided by the link's EF bound.

3.5.3. VW Jitter Sources [19]

There are three potential sources of queue wait for a VW packet:

1. it can queue behind non-EF packets (if any)
2. it can queue behind another VW packet from the same customer
3. it can queue behind VW packet(s) from other customers

For case (1), the EF 'priority queuing' model says that the VW traffic will never wait while a non-EF queue is serviced so the only delay it can experience from non-EF traffic is if it has to wait for the finish of a packet that was being sent at the time it arrived. For an output link of bandwidth B, this can impose a worst-case delay of *S*/B.

Case (2) can only happen if the previous packet hasn't completely departed at the time the next packet arrives. Since each ingress VW stream is strictly shaped to a rate R, any two packets will be separated by at least time *S/R* so having leftovers is equivalent to saying the departure rate on some link is $<R$ over this time scale. But the EF property is precisely that the departure rate MUST be $>R$ over any time scale of *S/R* or longer so this can't happen for any legal VW/EF configuration. Or, to put it another way, if case (2) happens, either the VW policer is set too loosely or some link's EF bound is set too tight.

Case (3) is a simple generalization of (2). If there are a total of *n* customers, the worst possible queue occurs if all *n* arrive simultaneously at some output link. Since each customer is individually shaped to rate R, when this happens then *no* new packets from any stream can arrive for at least time *S*/R. At the end of this time, there can only be leftover packets in the queue if the departure rate $< nR$ over this time scale. Conforming to the EF property means that any link capable of handling the aggregate traffic must have a departure rate $> nR$ over any time scale longer than $S/(nR)$ so, again, this can't happen in any legal VW/EF configuration. For case (1), a packet could be displaced by non-EF traffic once per hop so the edge-to-edge jitter is a function of the path length. But

this isn't true for case (3): The strict ingress policing implies that a packet from any given VW stream can meet any other VW stream in a queue at most once.

This means the worst case jitter caused by aggregating VW customers is a linear function of the number of customers in the aggregate but completely independent of topology.

3.5.4. VW Example uses [19]

An enterprise could use VW to provision a large scale, internal VoIP telephony system. Say for example that the internal links are all Fast Ethernet (100Mb/s) or faster and arranged in a 3 level hierarchy (switching/aggregation/routing) so the network diameter is 5 hops. Typical telephone audio codecs deliver a packet every 20ms. At this codec rate, RTP encapsulated G.711 voice is 200 byte packets & G.729 voice is 60 byte packets. 20ms at 100 Mb/s is 250 Kbytes (~150 MTUs, ~1200 G.711 calls or ~4,000 G.729 calls) which would be the capacity if the net were carrying *only* VW telephony traffic.

Worse case jitter from other traffic through a diameter 5 enterprise is 5 MTU times or 0.6 ms leaving between 19 ms (optimistic) to 10 ms (ultra conservative)for VW. 10ms at 100Mb/s is 125Kbytes so using the most conservative assumptions we can admit ~600 G.711 or ~2000 G.729 calls *if the ingress can simultaneously police both packet & bit rat*e. If the ingress can police only one of these, we can only admit ~75 calls because each packet might be as long as an MTU.

3.6. Assured Rate Per-Domain Behavior [20]

3.6.1. Introduction

The Assured Rate PDB is intended to carry traffic aggregates that require assurance for a specific bandwidth level.

The AR PDB is suitable for carrying traffic aggregates that require rate assurance but do not require delay and jitter bounds. The traffic aggregate will also have the opportunity to obtain excess bandwidth beyond the assured rate. The PDB can be created using the DiffServ AF PHB along with suitable policers at the domain ingress nodes.

This PDB ensures that traffic conforming to a committed information rate (CIR) will incur low drop probability. The aggregate will have the opportunity of obtaining excess bandwidth beyond the CIR but there is no assurance. In addition to the CIR, the edge rules may also include other traffic parameters such as the peak information rate (PIR) to place additional constraints for packets to which the assurance applies or to further differentiate packets which exceed the CIR.

3.6.2. Assured Rate PDB Specification [20]

The specification for this PDB consist of two parts:

1. A set of Edge rules that classifies packets arriving at the domain ingress into a traffic aggregate, performs metering/policing on the aggregate and associates a packet marking with the aggregate. Traffic shaping does not need to be performed on the aggregate as it enters the domain.
2. Per-node PHB treatment for the traffic aggregate as it weaves its way from the domain ingress to the domain egress.

Edge Rules

As packets enter the domain they will be classified into a traffic aggregate based on the specified filter at the domain ingress interface of the border router. The filter MUST be associated with a traffic profile that specifies committed information rate (CIR) AND a description on how it is to be measured. For example, the measurement may be based on a committed burst size (CBS) or an averaging time interval (T1).

The policer causes each packet arriving into the domain to be marked with one of up to three levels of drop precedence, which we call (in the increasing order) green, yellow, red. The packets to which the assurance applies, MUST be marked green. The excess packets MUST be marked as either yellow or red. The details of packet colouring are dependent on the specific policer utilized at the ingress router.

Red color packets SHOULD be delivered with equal or lower probability than yellow color packets. A special case of this is that all red color packets are discarded by the ingress policer.

Yellow packets SHOULD not be dropped by the ingress policer. They MAY be dropped by the buffer management mechanisms of the ingress router but that will be due to PHB treatment.

The green, yellow and red packets MUST be marked with the DSCP for AFx1, AFx2 and AFx3 PHBs respectively, where x MUST be any one value from 1 to N. N is the number of AF classes supported by the routers in the domain.

PHB Configuration

As described above, the AR traffic aggregate is to be treated using PHBs AFx1, AFx2 and AFx3 from a single AF class x. The resultant combination of the edge rules and PHB treatment within a single AF class, will ensure that:

"Within each AF class IP packets are marked (again by the customer or the provider DS domain) with one of three possible drop precedence values. In case of congestion, the drop precedence of a packet determines the relative importance of the packet within the AF class. A congested DS node tries to protect packets with a lower drop precedence value from being lost by preferably discarding packets with a higher drop precedence value."

3.6.3. Assured Rate PDB implementation Requirements [20]

The requirement to achieve the PDB is as follows:

Nodes internal to the domain SHOULD not drop packets marked to receive treatment with AFx1. Under exceptional circumstances, network nodes MAY have to drop AFx1 packets for a short period. In such cases, they should only start dropping AFx1 packets after they have started dropping all AFx2 and AFx3 packets. In the case where the AF class is lightly loaded, AFx2 and/or AFx3 packets MAY also be transmitted successfully through the node. This will allow the aggregate to obtain excess bandwidth beyond its assured rate.

The AR PDB MUST have the following parameters:

- A committed information rate (CIR) that is assured with high probability. The AR PDB specification does not define "high" quantitatively, but an SLS MAY do so.
- Traffic parameters that are needed to measure CIR. The AR PDB specification does not define these parameters, since they depend on the policer used. Examples include a Committed Burst Size (CBS) and an averaging interval (T1).
- A maximum packet size for the aggregate - MAX_PACKET_SIZE.

3.7. Some previous Efforts in Implementing DiffServ services

Researchers in [1] studied the priority queuing when applied for EF traffic with BE traffic. They studied the effect of the background traffic packet size distribution and EF profile on the one-way delay probability function. Authors of [2] implement EF and AF in the minimum processing using two queues one for EF and one for AF and using simple variation of SCFQ . The rate guarantee is provided within that queue using proposed buffer management, where a stream is allocated an amount of buffer proportional to the fraction of link band-width it is entitled to. The main benefits of this method is that rate guarantees can be provided to individual streams without incurring the complexity of a scheduler. Paper [3] talks about available schedulers and buffer management and how to use them to set a guarantee service.

In [4] the loss and delay behaviors that can be provided is analytically compared using the services based on combinations of two router mechanisms, *threshold dropping* and *priority scheduling* and two packet marking mechanisms, *edge-discarding* and *edge-marking*. The authors of that paper showed that priority scheduling provides lower expected delays to preferred packets. In addition, They found that a considerable additional link bandwidth is needed with threshold dropping to provide the same delay behavior as priority scheduling. They also found that both router mechanisms provide similar loss rates to preferred packets with an exception for extremely bursty sources, in which case threshold dropping had better performance.

They further examined the effect, on the throughput of a TCP connection, of forwarding out-profile packets into the network. They derived a simple Markov model for determining throughput of a TCP connection when in-profile packets observe different loss rates than out-profile packets. They found that it is possible to improve the throughput significantly even when a small portion of traffic is sent as in-profile packets. However, They observed that in order to fully utilize the benefit of out-profile packets, a TCP source must carefully determine the amount of out-profile packets it will send in addition to the in-profile packets. This result suggests that it is beneficial to combine edge-marking with threshold dropping for throughput sensitive applications.

Reference [5] describes and solves simple analytic models of two proposed schemes, the Assured Service scheme and the Premium Service scheme.

4. Expedited Forwarding and Virtual Wire Implementation

This section presents a discussion on the EF services implemented in the our DiffServ based SLA testbed.. Virtual wire PDB was used to implement a virtual leased line service. It was evaluated using real and emulated VoIP streams. The EF PHB used in the implementation was effected in two ways: a) using Priority Scheduler and b) using Self Clocked Fair Queuing (SCFQ) scheduler. Both implementations were tested and measurements were made for loss, delay and jitter. Empirical evaluation of the EF service requirement was verified first using test-bed element testing, then using application testing. Latter was effected using VoIP streams to show that they can be protected against full traffic load on a shared interface.

4.1. Experimental Environment

As mentioned earlier, the basic test-bed was developed in a previous research effort by Narasimhan [18]. That thesis describes in detail the original implementation of a differentiated services router in software in a Linux environment. This work has extended that test-bed, and is focusing on addition of tools (implementations, scripts, etc) that support general SLA studies, and on evaluation of the elements that support this SLA test-bed.

The basic logical components of the soft DiffServ router implementation (i.e., an SLA test-bed node) used here are:

1. **Classifier**:
   b) Behavioral Aggregate Classifier where classification is according to DSCP
   c) Multi Field Classifier It can classify according to destination address, source address, or protocol ID.
2. **Traffic Conditioner**:
   a) Single Rate Three Color Marker (SRTCM)
   b) Two Rate Three Color Marker (TRTCM)
3. **Buffer Manger**:
   a) Normal(Tail Drop)
   b) Threshold
   c) RED

d) Multi_RED
e) Multi_threshold

4. **Link Scheduler**:
   a) FIFO
   b) Static Priority
   c) Weighted Round Robin:
      - Decrement Mode – packet Number/Packet Size
      - Service Mode – Burst/ Non Burst Mode
   d) Self_clocked Fair Queuing SCFQ
   e) Virtual Clock VC
   f) Hierarchical scheduling
   g) Integrated Scheduler
5. **Shaper (rate-based shaper)**

This PHB engine is used as a **basic node** of the DiffServ-based SLA test-bed. Before the actual SLA elements and solutions were evaluated, there were two testing and evaluation phases aimed at the employed experimental environment itself: component testing of soft DiffServ router, and testing of the test-bed itself. The system was tested first by the original developer. This was followed by an additional extensive evaluation by the author of this thesis and several other NC State researchers. The latter series of tests includes not only the scenarios and test-suites (component testing and test-bed level tests), but also comparisons against a DiffServ simulator. The tests are described in detail in a separate report [41], some elements of which are reproduced in Appendix A.

4.1.1. Component testing

This testing was part of the second-round of verification and validation of the functionality of the logical elements and the basic functions of the soft DiffServ router. Each component, such as a classifier, buffer manager, scheduler, and so on, was tested separately. These tests were performed by our DiffServ team [41]. Appendix A shows some of these tests results.

Tests showed that DiffServ router implementation is functioning as expected, and can be used as part of a real network configuration.

4.1.2. Test Bed

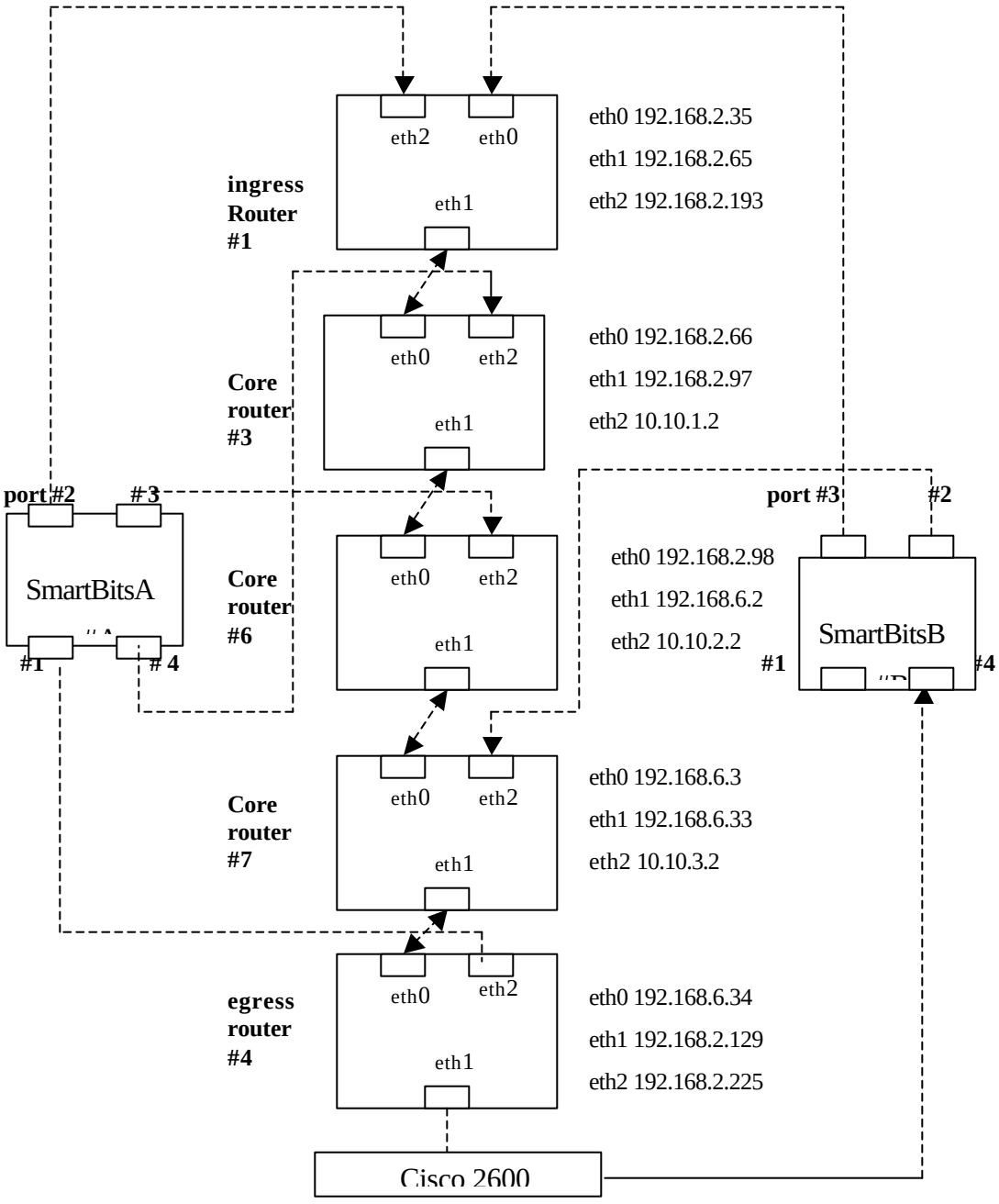


Figure 6. Basic DiffServ Test Bed Configuration

The DiffServ Test Bed was built using five (5) soft DiffServ Routers connected to simulate an ingress edge router, a set of core routers and an egress edge router, as shown

in Figure 6. Note that two traffic generators were part of the test-bed, SmartBits A and B, each with four 10/100 Mbps Ethernet interfaces. The five routers were configured into eleven sub nets. Each Linux router has three or more networking cards. They were either directly connected to each other, or through switches. The interconnection of routers with the SmartBits was designed to simulate as many traffic sources and sinks as possible. The basic DiffServ test-bed was extended with SLA supporting scripts and processes (e.g., EF and AF PHBs and PDBs) the suitability of which is the subject of the present work.

4.1.3. Emulation of Internet Background Traffic (or Cross-Traffic)

In order to evaluate DiffServ in more realistic scenarios, an option was added to the test-bed to emulated the Internet background traffic as "cross-traffic," or traffic that provided a load on the test-bed routers to simulate an operational environment. We used the analysis carried out in [35] to determine the parameters that best match this kind of traffic.

Figure 7 illustrates the the distribution of packet sizes from a 24-hour time period, in both directions, of traffic exchanged on an intercontinental trunk connection [35]. Sampled population was more than 100 billion of packets. This figure illustrates the predominance of small packets, with peaks at the common sizes of 44, 552, 576, and 1500 bytes. The small packets, 40-44 bytes in length, include TCP acknowledgement segments, TCP control segments such as SYN, FIN, and RST packets, and telnet packets carrying single characters (keystrokes of a telnet session). Many TCP implementations that do not implement path MTU discovery use either 512 or 536 bytes as the default Maximum Segment Size (MSS) for nonlocal IP destinations, yielding a 552-byte or 576-byte packet size. A Maximum Transmission Unit (MTU) size of 1500 bytes is characteristic of Ethernet-attached hosts. A similar profile was used to produce "cross-traffic" when such was used in the experiments discussed here.

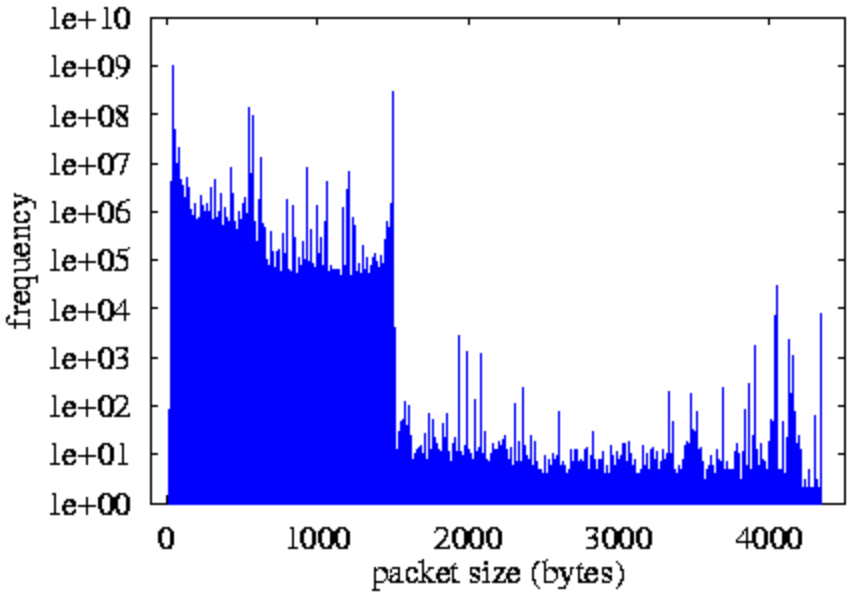


Figure 7. Packet size distribution on "Internet." [35]

4.1.4. Traffic Generators

Specialized equipment for traffic generation - the SmartBits 200 by Netcom Systems - was deployed as single-point measurement units that support packet time stamping in hardware and providing a precision of 100 nsec. The SB200s were equipped with ML-7710 10/100 Ethernet interfaces. This allowed very precise traffic control and measurements. For example, interfaces were used to originate Expedited Forwarding, Assured Forwarding or Best Effort traffic flows to receive them once they have passed through the network. In this way precise one-way measurements can be made. For example, very precise delay measurements could be made that are not affected by clock synchronization errors that normally occur when two-point measurements are made.

4.2. Virtual Leased Line

The special service constructed and evaluated in the current work is that of Virtual Wire PDB as proposed in [19]. It is evaluated in the context of Voice over IP (VoIP) applications. Specifically, considered is the implementation of the Virtual Leased Line (VLL) service which is then evaluated by its ability to protect VoIP traffic

The VLL service is targeted for applications and customers that require predictable point-to-point performance [19]. A virtual leased line is a point-to-point pipe with a guaranteed peak transmission rate. No packets are lost due to network congestion, and delay and jitter are very low. Applications that need such service include Voice over IP, transaction processing, and multimedia applications that require low queuing delay and jitter. VLL offers a simple, well understood performance model (approximating leased line performance) [19], and so should be appealing for any application, including web applications, where predictable performance is highly valued. The reserved rate may be renegotiated, and thus it may vary over time. Packets arriving at a rate exceeding the negotiated rate, R, are either dropped or buffered at the point of origination of the flow. Furthermore, no packet of a flow is dropped due to congestion. However, losses may occur due to other factors.

The VLL traffic is assumed classified, and marked with DSCP 100110, on the customer premises. The provider polices incoming traffic and drops out of profile packets

4.3. Voice Over IP (VoIP)

One of the most interesting applications that can be protected using VLL service is VoIP. This section highlights some of the related features and requirements.

A typical method of transporting voice through an IP-based network requires addition of at least three headers to the voice sample, one for each layer of communication. These headers are for IP, UDP and RTP. A typical IPv4 header has 20 octets; a UDP header has 8 octets, and an RTP header has 12 octets. The total length of this header overhead is typically 40 octets (bytes), or 320 bits. These headers (or overhead, if one wants to think about it that way) are sent each time a packet containing voice samples is transmitted. The additional bandwidth occupied by this header information is determined by the number of packets sent per second. A typical voice payload represents about 20ms of audio.

Hence, if we assume that one voice packet represents 20 milliseconds of voice, then 50 such packets are required to cover one second of speech. Since each sample carries a IP/UDP/RTP header overhead of 320 bits, in each second 16,000 header bits are sent. Therefore, as a general rule of thumb, it may be assumed that header information will add 16kbps to the bandwidth requirement of a voice over IP channel. For example, if an 8kbps algorithm such as G.729 is used [8], the total bandwidth required to transmit each voice channel is about 24kbps.

The designer of any solution that includes VoIP will need to decide upon which coding algorithm, or CODEC, to use. *CODEC*s perform the conversion from an analogue voice waveform to a digital stream of information. They *sample* the analogue signal at regular intervals (125 microseconds is a typical value), and convert the measured analogue value into a numeric representation (this is known as *quantisation*). The resultant output are discreet blocks of information sent at regular intervals. Table 1 illustrates some of the relevant characteristics of the most common coding algorithms. The reader can obtain more detailed information about VoIP algorithms and issues from the following reference [8].

Table 1. Common coding algorithms characteristics

Coding algorithm		Bandwidth	Sample	IP bandwidth
G.711	PCM	64kbps	0.125ms	80kbps
G.723.1	ACELP	5.6kbps	30ms	16.27kbps
		6.4kbps		17.07kbps
G.726	ADPCM	32kbps	0.125ms	48kbps
G.728	LD-CELP	16kbps	0.625ms	32kbps
G.729(A)	CS-ACELP	8kbps	10ms	24kbps

4.4. Metrics

In order to assess the quality of service (QoS), we used two relevant key QoS metrics: *one-way packet delay* and *instantaneous packet delay variation or variability.* These are the key parameters for applications that have stringent QoS demands. Along with packet loss, poor values of these metrics are strongly correlated to end-user view of what "poor" quality of voice is.

Key to the proper measurements is the notion of *wire time*. It is assumed that the measurement device has an observation post on the IP link: The packet arrives at a particular wire time when the first bit appears at the observation point, and the packet departs from the link at a particular wire time when the last bit of the packet has passed the observation point.

One-way Delay is defined formally in RFC 2679 [42]. This delay is measured as the difference between the wire time of the packet arriving on the link observed by the sender, and the wire time of the last bit of the packet observed by the receiver.

Instantaneous Packet Delay Variation (IPDV) is formally defined by the IPPM working group [43]. It is based on one-way delay measurements, and it is defined for (consecutive) pairs of packets. A *singleton* IPDV measurement requires two packets. If we let D_i be the one-way delay of the i^{th} packet, then

$$IPDV = D_i - D_{i-1} \tag{4.1}$$

According to common usage, IPDV-jitter is computed according to the following formula:

$$IPDV\text{-}jitter = |IPDV| \tag{4.2}$$

Packet-loss Percentage: the percentage of packets lost during the whole duration of a stream.

4.5. Experimental Set-Up

The experiments conducted as part of this work were not comprehensive. They were intended more as "proof-of-concept" evaluations that will be, in subsequent projects, backed up with detailed simulations and additional, more comprehensive, experimental work. In the current context, the general idea of an experiment is to configure routers in the test bed so that their DiffServ capabilities protect the flow representing the VoIP traffic against some Best Effort cross traffic .

In a typical experimental scenario, VoIP traffic was presented by a 1-3 Mbps flow on a 10 Mbps Ethernet channel (i.e., about 10-30 % of channel capacity). It was assumed that typically the voice packet size is 128 bytes. Cross traffic was presented as an almost full load stream (about 9.5 Mbps) of 1514 byte packets.

First, the VoIP traffic was classified at the ingress router, and marked using EF DSCP value of 0xb8. All routers were configured to have two queues, one for the Expedited Forwarding traffic and the one for the Best Effort traffic. The Static Priority scheduler was used in one set of experiments. This was followed by experiments using a version of WFQ called Self Clocked Fair Queuing (SCFQ). Ns simulations were used to repeat and check on some of the EF experiments. Sample ns script is shown in the appendix D. In what follows, the graphs obtained from ns-2 simulations are usually, for comparison, shown next to graphs abstained from real experiments.

4.6. Confidence Intervals

At each measurement point, two UDP flows, EF and BE, were applied for 60 seconds. This was repeated at least 3 times. Average EF delay and jitter were computed using SmartBits. We note that the average delay was consistent, and was almost the same value, in each of the 3 or more measurements taken per point. However, jitter, as apparently measured using SmartBits was much less consistent over the repetitions. One reason was the way PC-based SmartFlow GUI interface to SmartBits [46] [47] measures

jitter, or standard deviation of the delay. It uses an approximation based on a very limited number of intervals (or "buckets") to sample the delay distribution. SmartFlow version employed here (version 1.20.018) uses at most 8 "buckets" to capture the whole set of observed delay observations. Obviously, this quantization affects the jitter distribution. If interval per bucket is wide enough (e.g., there are some extreme outliers), most of the delay distribution will fall into the same bucket, and the variance will be low, and possibly even zero. Of course, one could choose the right set of bucket intervals around the average delay, but this requires a lot of manual tuning and is not very productive in experiments where a number of different situations need to be compared, situations where average delays may differ significantly from one scenario to another. One option was to use scripting and directly download the data from the SmartBits bypassing the GUI interface (and graphics) offered by SmartFlow toolset. Instead of doing that, it was decided to use Iperf [49] to estimate for the jitter. Iperf gave much more consistent results. To get an estimate of the confidence interval for the average delay and jitter measurements, we computed the bounds assuming binomial error distribution. Assessment was done using the following parameters. Consider the following estimates:

Aerage Delay Measurement point : EF rate = 1 Mbps , BE rate = 8 Mbps, EF packet size = 128 bytes, BE packet size = 1500 bytes

Table 2. Average Delay measurement Results

Average Delay (micro sec)
3742.4
3731
3726
3727.7
3726.6
3730.1
3726.3
3733.6
3727.2
3727.2

Jitter Measurement point : EF rate = 1 Mbps , BE rate = 4 Mbps, EF packet size = 1470 bytes, BE packet size = 1500 bytes

Table 3. Jitter measurement Results

Jitter (micro sec)
448.2
482.3
440
404
491.3
501
449
480
403.6
502

Using equations found in [44] we calculate the confidence interval as follows:

Suppose x_1, x_2, ….. x_n are the measurements (results) table 2 and 3 above.

The sample mean is

$$\overline{X}(n) = \frac{\sum_{i=1}^{n} x_i}{n} \tag{4.3}$$

The sample variance is

$$s^2 = \sum_{i=1}^{n} [x_- - \overline{X}(n)]^2 \Big/ n-1 \tag{4.4}$$

Because of the small number of observation (n=10) we used the t-distribution with (n-1) degree of freedom, where the 100*(1-α) percent confidence interval for the real mean (μ) is given by the estimated mean and the +/- bounds shown below

$$\overline{X}(n) \pm t_{n-1,1-a/2}\sqrt{s^2(n)/n} \tag{4.5}$$

where $t_{n-1,1-\alpha/2}$ is the upper (1-α/2) critical point for the t distribution with n-1 degree of freedom.

The 90 percent confidence interval for the mean of average delay measurements

$$\overline{X}delay(n) \pm t_{n-1,1-a/2}\sqrt{s^2(n)/n} = 3729.81 \pm 2.92 \text{ micro sec}$$

The 90 percent confidence interval for mean of Jitter measurements

$$\overline{X}jitter(n) \pm t_{n-1,1-a/2}\sqrt{s^2(n)/n} = 460.14 \pm 21.4 \text{ micro sec}$$

We see that the average delay appears to still have smaller relative bounds than corresponding average jitter. We also see that, in the current set-up, the jitter is about 10% of the delay. Of course, this example only illustrates the magnitude of the bounds given a particular measurement point, and the exact numbers will differ from point to point.

4.7. Results

4.7.1. EF Queue Size Requirements

In this case, Static Priority Scheduler (SPS) was used to give the EF priority over the Best Effort queue. As mentioned earlier, the EF traffic was set to 30% of the 10 Mbps channel capacity, i.e., 3 Mbps, and the EF packet size was 128 bytes (which is close to a typical voice packet size). To investigate the worse case situation, the Best Effort traffic

IP packet size was set to maximum so that the Ethernet frame size was always 1514 bytes. Both streams were applied at the ingress router, and they both followed the same path to the egress router, thus competing for the channel capacity through the five routers.

Table 4. EF Queue size requirements

(Priority Scheduler, BE =9 Mbps, EF= 3 Mbps)

EF Buffer size (byte)	Loss Rate %	Jitter (micro sec)	Avg Delay (micro sec)
128	67.1	935.5	19085
256	66.2	900	21138
384	63.8	837.5	21890
640	57.9	794.2	24045
1280	33.45	972.1	18704
1920	6.9	4099.4	17108
2560	0.198	4543	17898
2816	0.1	4043	17063
3200	0.008	4378	20817
4480	0		

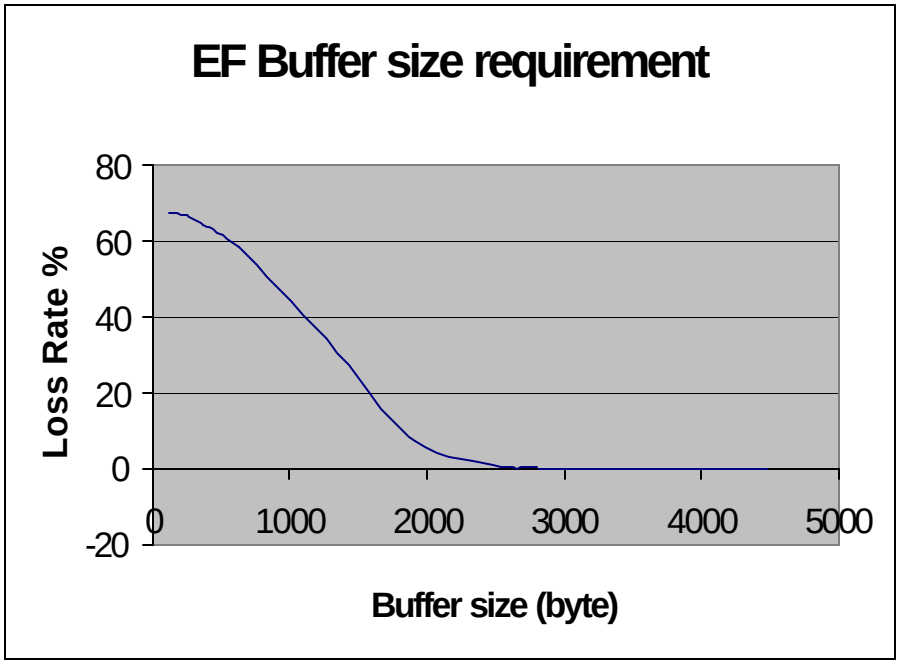


Figure 8. Average EF parameter values vs. EF Buffer size

When the EF traffic has absolute priority over BE traffic, the output priority scheduler typically does not send out any BE packet while there are EF packet waiting. However, there may be problems even under these circumstances. For example, EF packets themselves may start a queue build up if an EF packets arrive in an excessive bursts, or when there is a BE packet already being served. Different sizes of EF and BE packets may aggravate the situation. Accumulation may get worse with multiple hops, as in the case of our test bed configuration. Hence, EF buffers need to have an appropriate size to accommodate the variability and avoid packet loss and undue delays.

To get a rough idea of how much buffer size we need for per-hop EF queue (and to explore the usefulness of the test-bed for experiments of this type), let us do a simplified calculation and then compare it with observations in the test-bed. Consider a single router, and consider the situation where an EF packet arrives just at the moment where a BE packet is already in the process of being served. Given the previous assumptions about the parameters and rates (i.e., 10Mbps channel, 3Mbps EF rate of 128 byte packets, and BE frame size of 1514 bytes at 9.5 Mbps), then, at worst:

Service Time for one BE packet (Tb) = 1514 x 8 bit /10,000,000 bps = 1.2112 msec

Avg. No of EF packets arriving during Tb = Tb x (3,000,000 bps / 128 x 8 bits per packet)
= 1.2112 x 2.9 = 3.5 packets

This gives approximate worst case numbers for one router in the packet path. It assumes that one BE packet is being serviced at the time a EF packet arrives. Over five router hops this can accumulate to around (5 x 3.55) 18 packets of delay if we assume that the burst length is the same on all routers, i.e one packet. The problem becomes compounded when we also consider packet accumulation due to bursts which vary in length between one packet and a maximum burst of 3.5 (i.e 4) packets. With this assumption, the average burst length is (4/2) 2 packets. Hence, the packet backlog now becomes (18x2) 36 packets.

An experimental estimate of the parameters was done by gradually increase the EF buffer size and measuring the loss rate until no losses were noticed. Under the given conditions, this occurred at around queue size of 4480 bytes. This is approximately equal to 35 EF packets which is close to the 36 packets worse case analysis number computed earlier. Table 2 and Figure 8 illustrate the results.

4.7.2. Effect of BE packet size on Delay and Jitter of EF traffic

Table 5. Experimental EF Delay and Jitter vs. BE packet size (Priority)
(Priority scheduler, BE = 8 Mbps, EF(128 bytes) = 1 Mbps, Buffer size = 4480)

BE packet size (Bytes)	EF Avg. delay(micro sec)	EF Jitter(ms)
64	760.9	0.014
128	789	0.068
256	1130.2	0.12
375	1460	0.095
512	1827	0.095
720	2325	0.152
1024	3252.7	0.379
1200	3744	0.51
1300	4032	0.158
1450	4438	0.049
1500	4580	0.52
1514	4622	0.643

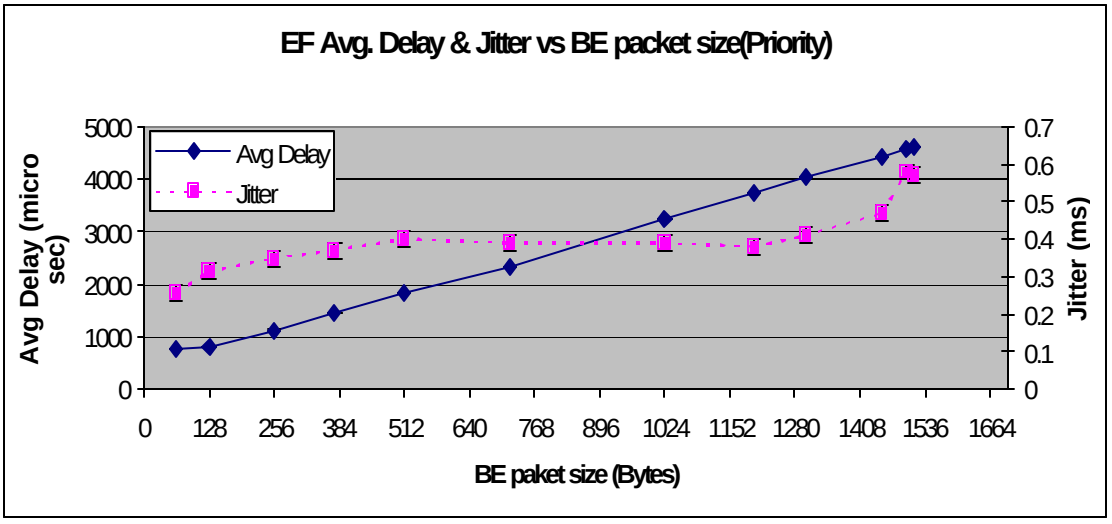


Figure 9. Experimental EF delay and jitter vs. background BE packet size(Priority)

In this experiment, the EF Buffer sizes were set according to the results from the previous section (i.e., 4480 Bytes) . With this buffer size, the EF traffic was protected against full load Best Effort traffic. The next step was to see if the delay and the jitter requirement for the emulated VoIP streams can be met for the worst case background- or cross-traffic. First, we varied the background BE packet sizes to see the effect the packet size has on the delay and the jitter of the EF traffic. The results are shown in Tables 5 (priority scheduling experiment) and Table 6 (SCFQ experiment) and Figures 9 (priority experiment),10 (priority simulation),11 (SCFQ experiment).

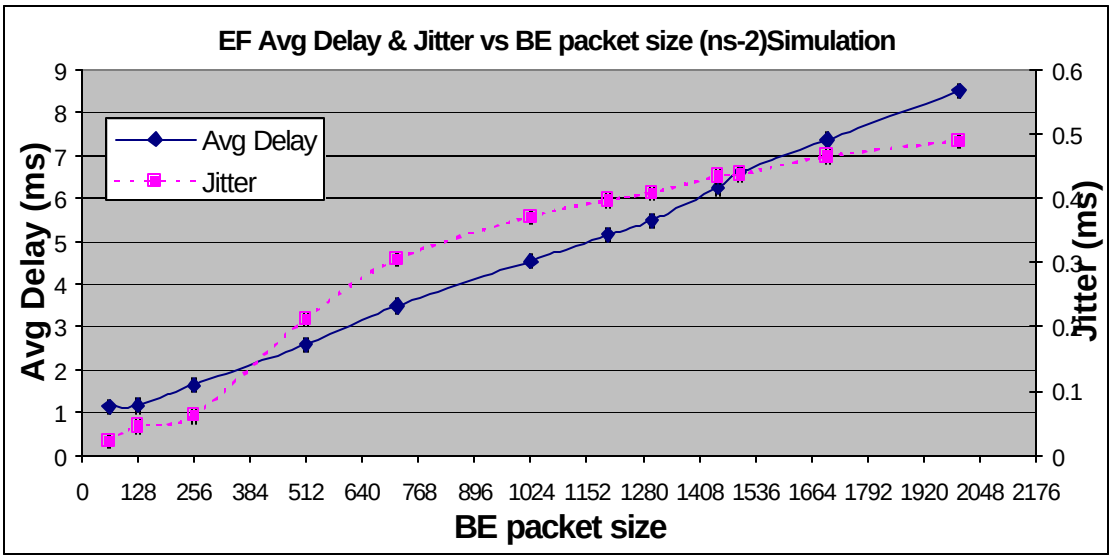


Figure 10. EF delay and jitter vs. background BE packet size using ns-2 simulation for priority scheduler

Table 6. Experimental EF Delay and Jitter vs. BE packet size (SCFQ)
(SCFQ scheduler, BE = 8 Mbps EF(128 bytes) = 1 Mbps Buffer size = 4480)

BE packet size (Bytes)	EF Avg. delay(micro sec)	EF Jitter(ms)
64	753	0.01
128	777	0.063
256	1125.3	0.116
512	1832.3	0.099
720	2401.7	0.156
1024	3248.9	0.386
1200	3729.81	0.502
1300	4025.7	0.163
1450	4438	0.012
1500	4578	0.56
1514	4621.4	0.612

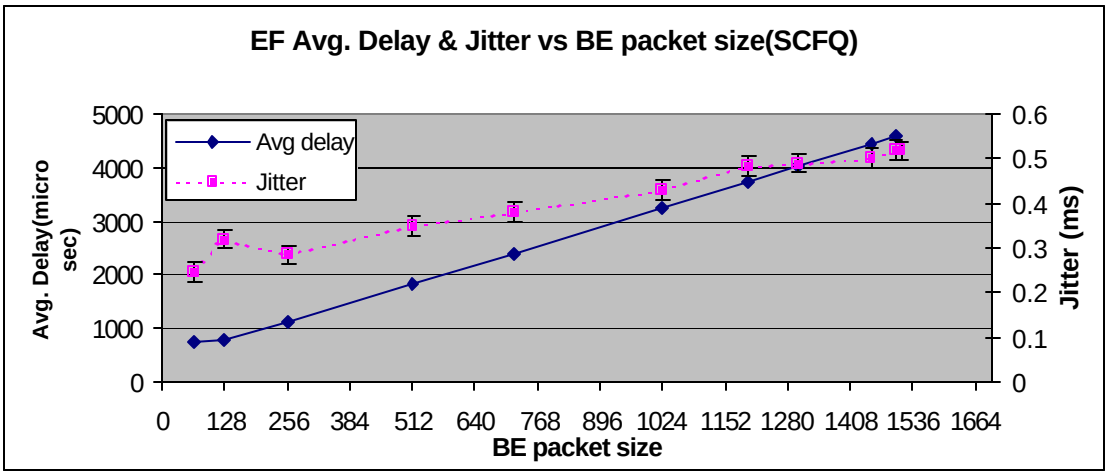


Figure 11. EF Delay and Jitter vs. BE packet size (SCFQ)

Priority scheduling simulation shows that the average EF delay increasing linearly with the cross-traffic BE packet size. This EF delay come from two sources. The first source is due to events where a BE packet is in the process of being served when the EF packer arrives (and, therefore EF packet needs to wait for BE service to finish). The second delay come from waiting behind other EF packets in a queue. Both delays can be affected by the BE packet size. The larger the BE packet size, the longer the EF packet has to wait, and during that time more EF packet may also arrive into the queue and this result in an increased average EF delay.

Simulations show that the EF jitter also increases with the BE packet size, but this increase may not be linear. EF jitter comes from the variation in the delay. This variation depends on many factors. For example, on whether the EF packet arrives into an empty or non-empty queue, it depends on the queue size (tail-drop), and on whether a BE packet, or another EF packet, is being served when the EF packet arrives, etc. Most of these factors also interact with the seize of BE packets.

Inspection of the experimental data (Figure 9, and Table 5) shows that the measurements agree with the simulations to a reasonable degree. Hence, the conclusion is that the collective performance of the test-bed is consistent with the expectations and that it can probably be used with a reasonable degree of confidence to further study similar effects.

Of course, specific studies do need to take into account their specific parameters, as well as possible additional parameters and environmental settings.

4.7.3. Effect of BE rate on Delay and Jitter of EF traffic:

To illustrate how the BE cross-traffic effects on the EF streams can be studied within the test-bed, the BE load was varied in steps show in Table 7. The average delay and jitter were measured for the EF traffic. Both Priority scheduler and the SCFQ scheduler were used. The results are shown in Table 7 (priority experiment), Table 8 (SCFQ experiment), Figures 12 (priority experiment), 13 (priority simulation), and Figure 14 (SCFQ experiment).

Table 7. Experimental EF Delay and Jitter vs. BE rate.
(Priority scheduler, BE(1500 bytes), EF(128 bytes) = 1 Mbps, Buffer size = 4480)

BE load %	EF Avg. delay(micro sec)	EF Jitter(ms)
0	767.1	0.005
10	1958	0.162
20	3182.9	0.181
30	3893	0.099
40	4248	0.449
50	4463	0.551
60	4605	0.137
70	117265.6	1.88
80	117185	1.78
90	117861	1.9
100	195852	2.1

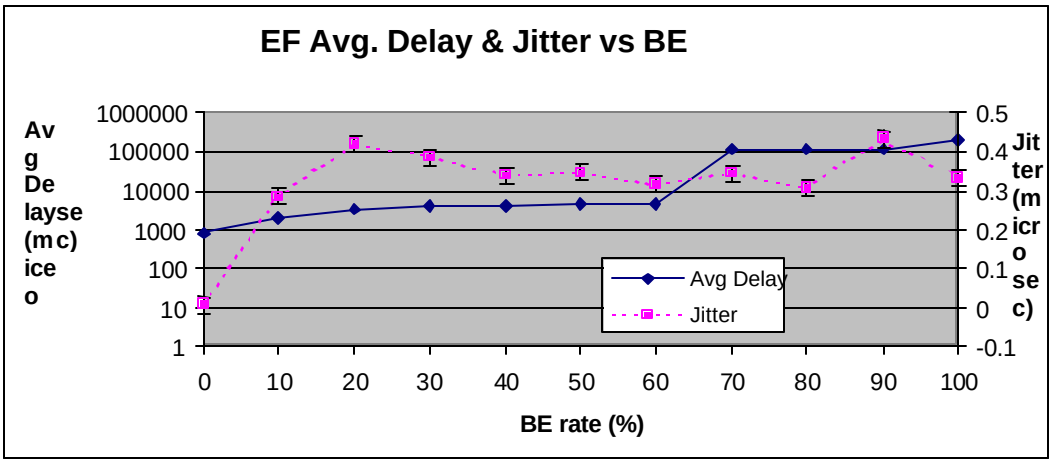


Figure 12. Experimental EF packets jitter vs. background rate for Priority scheduler

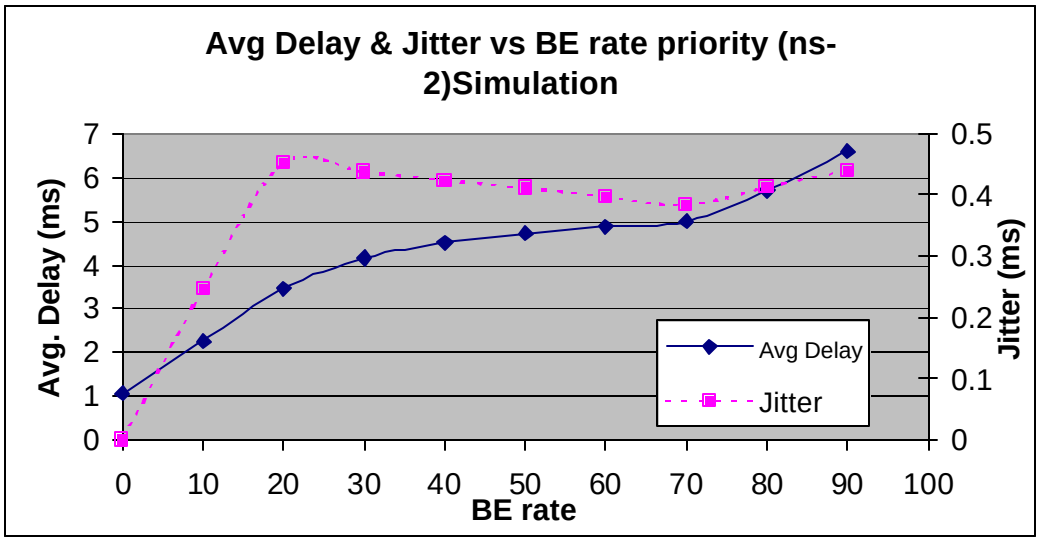


Figure 13. EF packets jitter vs. background rate for the ns-2 simulator

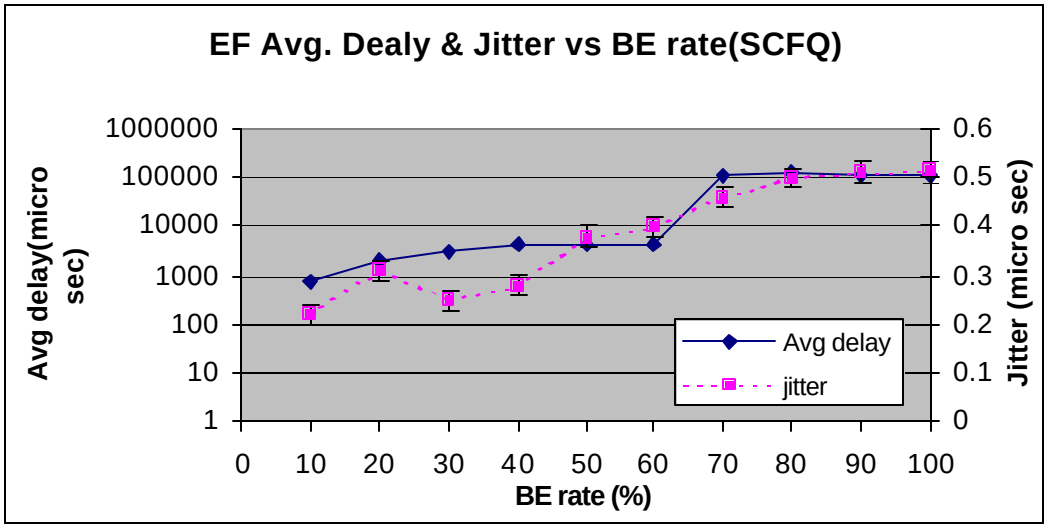


Figure 14. Experimental EF packets jitter vs. background rate for the SCFQ scheduler

Table 8. Experimental EF Delay and Jitter vs. BE rate (SCFQ)

BE load %	EF Avg. delay(micro sec)	EF Jitter(ms)
0	767.1	0.006
10	773.2	0.2
20	1948.1	0.255
30	3157	0.107
40	4213.4	0.44
50	4424.7	0.631
60	4566.2	0.139
70	119560.2	1.043
80	120155.7	1.226
90	118577.4	1.285
100	119172	1.416

All graphs (real experiments and simulation) show that EF average delay increases with BE rate. However, in the experiments when the BE load reaches about 7 Mbps (and EF flow= 3Mbps), the capacity of the link is exceeded (100% link utilization) and there is a sudden jump in the delay figures for the EF stream (although there were no losses in the EF stream). This is probably due to a combination of effects resulting from the BE over-pressure on the link. For example, full utilization of the link maximizes the probability that a BE packet is being served when an EF packet arrives, but it also must have some

additional influence, something that the simulation did not capture. In actual experimental evaluation this effect would merit further investigation.

4.7.4. Number of hops

To illustrate experiments concerned with the number of network hops, the following was done. Each router in the path was given identical settings, and then EF Delay and Jitter were evaluated at each hop. The results are shown in Table 9 (priority experiment), Table 10 (SCFQ experiment) and Figures 15 (priority experiment) ,16 (priority simulation) and 17 (SCFQ experiment). Note that in these experiments were run at an early stage of this project in which a four-node network was used, and where the jitter measurements were made using the SmarfFlow interface. The latter probably explains some observed anomalies, such as the sudden decrease in jitter after the 4th hop, something that would not be expected, and which is probably an artifact of the SmartFlow tool.

Table 9. Experimental EF Avergae Delay and Jitter vs. number of hops for priority scheduler

No of Hops	Avg Delay (micro sec)	Jitter (micro sec)	lossrate %
1	6065	2166	0
2	12358	7400	0
3	12991	9347	0
4	17354	6937	0

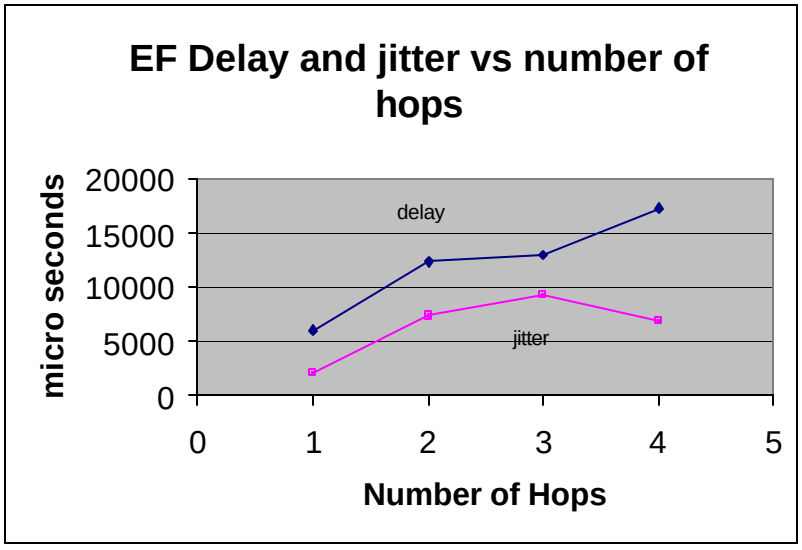


Figure 15. Experimental EF delay and jitter vs. Number of routers (Priority Scheduler)

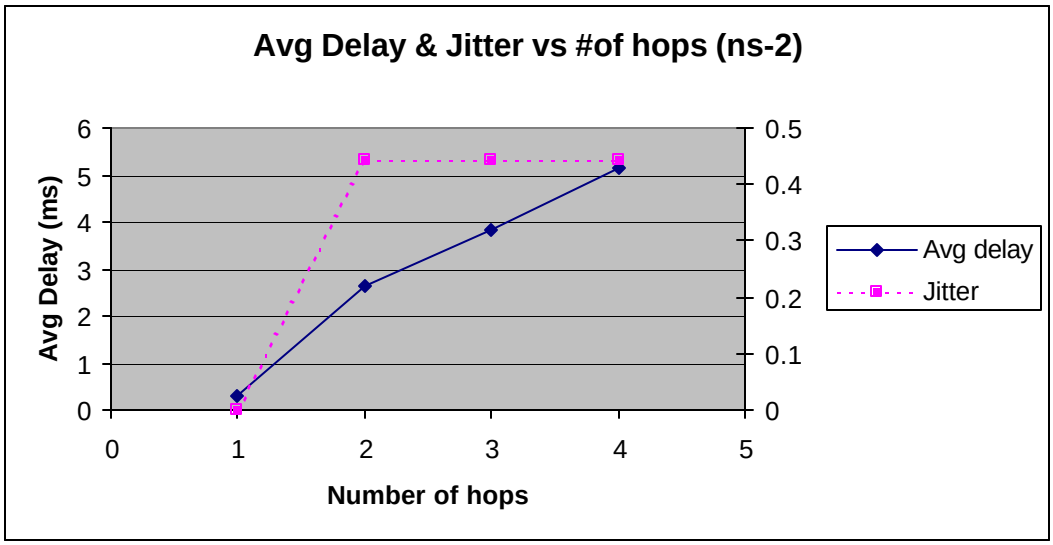


Figure 16. EF delay and jitter vs. No. of routers using Priority (ns-2 simulation)

Table 10. Experimental EF Average Delay and Jitter vs. number of hops for SCFQ scheduler

No of Hops	Avg Delay (micro sec)	Jitter (micro sec)	lossrate %
1	7072	463.3	0
2	13109	6228	0
3	16722	5686	0
4	19438	3592	0

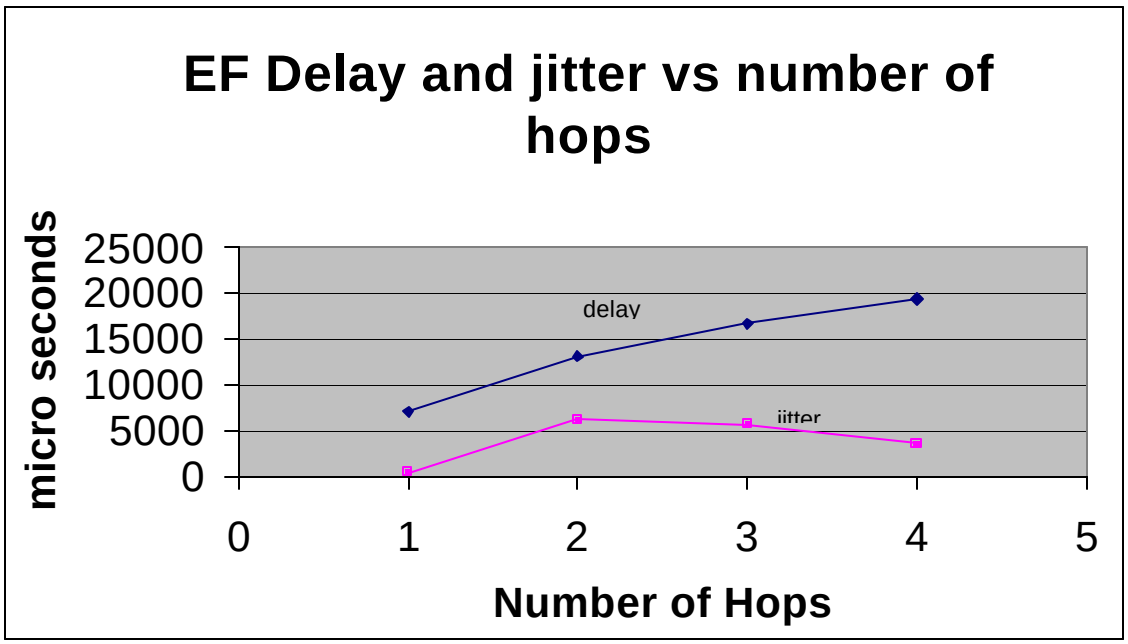


Figure 17. Experimental EF delay and jitter vs. No. of routers using SCFQ

As the graphs show, the average delay increases with the number of hops. This means that each hop adds its share of delay to the total EF delay. Apparently, the number of hops do not significantly affect the jitter.

4.8. Summary

This section was concerned with proof-of-concept experiments intended to show that a) the test-bed can be used to implement EF channels (as VLL) for critical streams (such as VoIP), and b) that the measurements of some typical quality of ser vice parameters (such as delay and jitter) are consistent with simulations of the environment and offer insights that can be helpful in real experiments.

To conduct these experiments each node of the test-bed was configured to quaranteee EF traffic up to 3 Mbps, and provide it priority over Best Effort. The 3 Mbps bandwidth was guaranteed using SRTCM Traffic conditioner, and two types of schedulers: a) Static Priority and b) SCFQ, a type of Weighted Fair Queuing. The Traffic conditioning was done only on the ingress router, and the same scheduling was used in all nodes in an experiment.

Proof-of-concept experiments involved illustration of the influence of BE packet size on protected emulated VoIP streams, influence of BE load on the EF streams, and the impact of the number of hops. These showed that one can construct basic SLAs that may involve EF and BE streams. While EF, in theory, should provide both delay and bandwidth guarantees, in practice, depending on the implementation, these guarantees may fall into a range and may not necessarily be upheld-able under very extreme conditions. The next section considers how the test-bed can accommodate SLAs that may involve assured forwarding traffic, traffic where only a reasonable guarantee needs to be give only for the bandwidth rather than both bandwidth and delays.

5. An Assured Rate PDB Implementation

There is a demand for assured forwarding (AF) of IP packets over the Internet. The Assured Rate PDB is intended to carry traffic aggregates that require assurance for a specific bandwidth level, but not necessarily of the delay and jitter. This PDB ensures that traffic conforming to a committed information rate (CIR) will incur low drop probability, and it will have the opportunity of obtaining excess bandwidth over a specified domain. However, there is no assurance of the latter.

In our proof-of-concept experiments with five serially connected DiffServ capable routers, Assured Rate PHB was effected using Single Rate Three Color Marker SRTCM for traffic conditioning, Multiple Threshold RED (Random Early Discard) for Buffer Management, and Static Priority for scheduling .

5.1. Assured Rate PDB Specification

5.1.1. Edge Rules

In the context of the SLA test-bed experiments discussed here. The following holds. As packets enter the domain from the ingress router they are classified into the four AF classes, or the default Best Effort class, according the filter rules set in the ingress router. Each filter is associated with a traffic profile specified by Single Rate Three color Marker (SRTCM) parameters for each class.

All the four classes are set to the same SRTCM parameter as follows

```
cir = 1,000,000 bps
cbs = 3,028 bytes
ebs = 6,056 bytes
```

where cir stands for committed information rate, cbs for committed burst size and ebs for excess burst size. The policer marks each packet arriving into the domain with one of three levels of drop precedence [16]. Those are green (Afx1), yellow (Afx2) , and

red(Afx3), where x is any value from 1 to 4, and it designates the number of AF classes Table 11 shows these codepoints.

Table 11. AF classes codepoints

	Class 1	Class 2	Class 3	Class 4
Low Drop Prec (green)	010000	011000	100000	101000
Medium Drop Prec(yellow)	010010	011010	100010	101010
High Drop Prec (red)	010100	011100	100100	101100

In this part of experiment the "red" packets will be discarded (dropped) by the ingress router.

5.1.2. Per Hop Behavior configuration

After marking the packets using Afx1, Afx2,and Afx3 PHBs, each packet must be treated based on its DSCP value as follows.

"Red" packets are dropped at ingress router. Within each AF class, a congested DiffServ node tries to protect packets with lower drop precedence, the ones with value "green," from being lost by preferentially discarding packet with higher drop precedence value, i.e., "yellow." This was accomplished by the following configuration described in the following paragraphs. Note here that we use lower case "red" to refer to the color of the packet according to its traffic profile, and upper case "RED" to refer to Random Early Discard as a buffer management technique.

Four queues were set for the four AF classes, and a fifth queue for the Best Effort traffic. Priority scheduler was used to give AF queues more priority over the Best Effort. Multiple RED threshold buffer manager was used when we set different RED threshold parameter for the "green" and the "yellow" as shown in Figure 18. Different RED parameters for yellow and green color mean that in each class yellow color packets may have more probability of being dropped than Green packet of the same class.

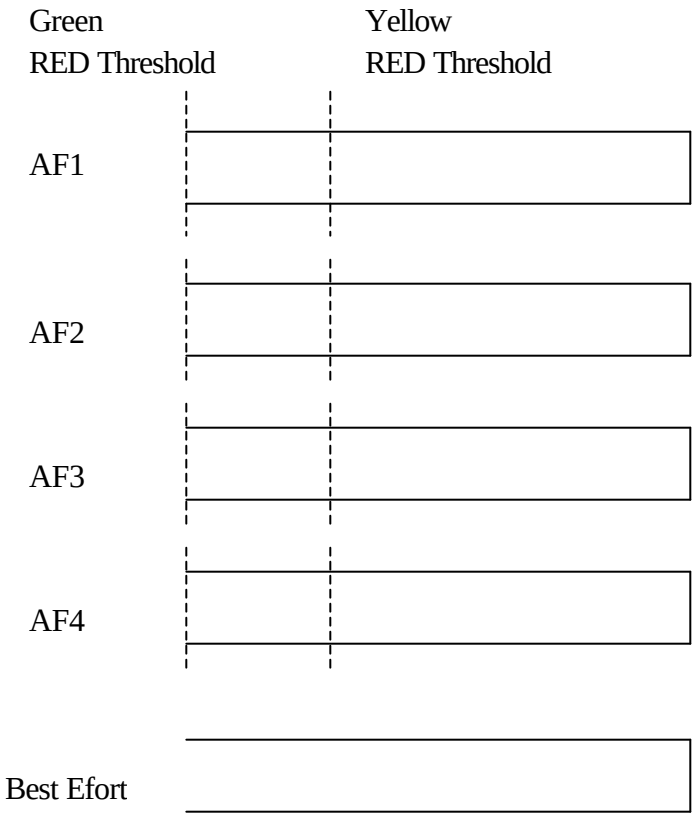


Figure 18. Buffer management for AF queues

5.2. An Empirical Evaluation

We applied four flows at the ingress router representing the four AF classes, and one flow represent the BE traffic. As mentioned earlier, the AF flows were conditioned to 1 Mbps. We started by fixing the BE rate at 9 mbps (90% load), and increasing the AF rate gradually and measuring the loss rate. The results are shown in Table 12, and Figure 19 below. We see that the AF flows see no losses until they reach their guaranteed rate of 1 Mbps. After that, the policer starts dropping the nonconforming packets.

Table 12. Loss rate vs. load for AF flows (BE = 9 Mbps)

AF load %	AF loss rate %	AF avg delay (micro sec)	AF jitter (micro sec)	BE loss%
5	0	24273	2567	10
10	0	24063	425	28.5
15	32.25	30586	225	29.17
20	49.19	21106	8867	23.55
25	59.36	22025	7445	25.49

* 10 % load equal to 1 Mbps

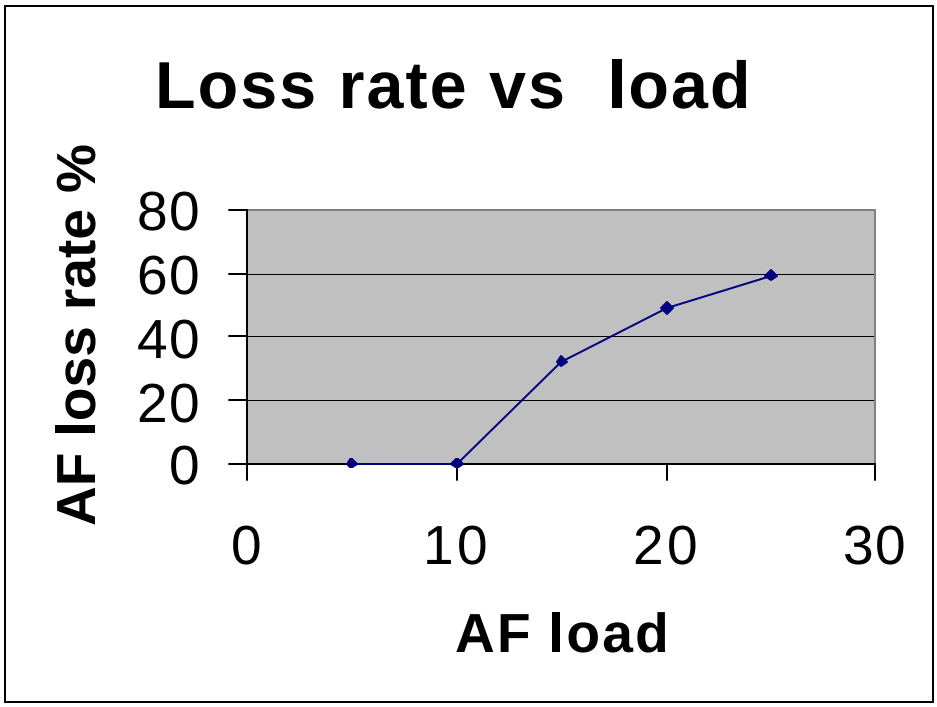


Figure 19. AF loss versus AF load. BE load = 9 Mbps

The same experiment was repeated with BE fixed at 1 Mbps (10 % load). Again the AF rate was gradually increased . As shown in Table 13 and Figure 20, a behavior similar to the previous case was observed. AF flows did not see losses until they reached their guaranteed rate of 1 Mbps. After that, the policer again started dropping nonconforming packets despite the fact that extra capacity was available. This is an interesting situation. It indicates that this particular solution may not be the best one to use if one wants to make use of excess capacity. Another solution may be a better one, one where the packets are downgraded to BE. However, that solutions carries with it the reordering problem, and requires a more sophisticated scheduling than is available in the system right now.

Table 13. Loss rate vs. load for AF flows (BE load = 1 Mbps)

AF load %	AF loss rate %	AF avg delay (micro sec)	AF jitter (micro sec)	BE loss%
5	0	6664	455	0
10	0	6372	425	0
15	32.25	6467	225	0
20	49.19	6375	319	0
25	59.36	6624	916	0

* 10 % equal = 1 Mbps

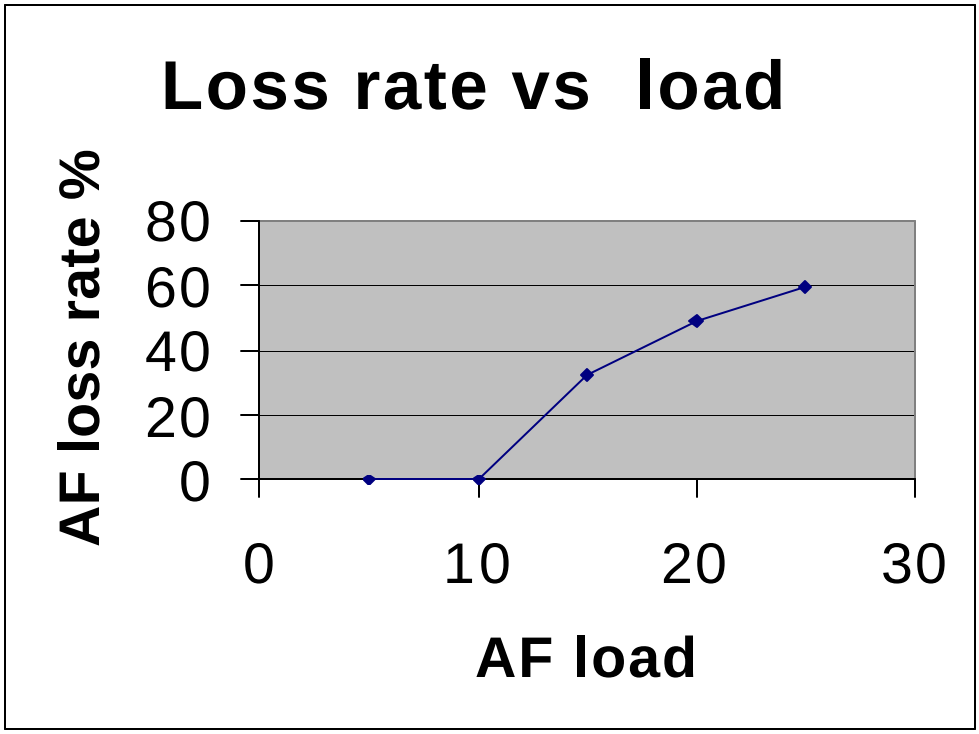


Figure 20. AF loss versus AF load for BE load of 1 Mbps

5.3. Summary

The set of experiments carried out using the AF traffic is very straight forward, and the configuration is one of many ways of implementing AF PHB. It has yielded results consistent with simulations, and it has demonstrated that the SLA-test-bed is ready to implement fairly complex SLAs. Those that involve EF, AF and BE traffic. Some issues remain. For example, ways in which AF can and is implemented and the impact on the BE traffic. Also, the matter of resource utilization. For example, one would like to use excess bandwidth when it is available to accommodate excess EF traffic, but whith some limited guarantees. The same with excess AF traffic.

In this context we plan to consider, in the future, other ways of implementing AF and of scheduling complex SLAs. For example, another way of implementing AF PHB is to use one queue and divide it between the four AF classes and the Best Effort class, and then to use different RED buffer management parameters to give AF traffic guarantees over Best Effort. In this way one uses buffer allocation to achieve the goal, instead of bandwidth allocation. Yet another way could be to use Weighted Fair Queuing between queues of each class of AF and the BE queue. This way allocates bandwidth to each class, but does not discriminate between BE and AF traffic. In our implementation, we are giving better treatment to AF traffic.

5.4. Reordering problem

When AF traffic exceeds its allocated rate, one option is to drop the AF packets. However, this is neither fair nor economical when resources (bandwidth) are available. So, one solution is to downgrade the nonconforming AF traffic to Best Effort (BE). Unfortunately, doing so in the direct (obvious) way, with currently available schedulers, creates a reordering problem – some of the packets form the same stream reach the destination out of order because they can take either the AF path, or the BE path. This is equivalent to packet loss, and it causes problems for AF guarantees. Work is in progress to on a scheduler that downgrades the traffic without suffering from the reordering problem. This beyond the scope of this thesis and will be discussed elsewhere in conjunction with issues related to construction and management of complex SLAs.

6. Conclusions and Future Work

The principal objective of the work presented in this thesis was to develop and assess a DiffServ-based SLA test-bed so that it can be used for evaluation of various SLA solutions and implementations. The test-bed will allow studies of complex SLAs, and of associated service, and validation of new SLA and DiffServ related algorithms in an environment that has very high flexibility and granularity.

6.1. Expedited Services

The DiffServ based SLA test-bed was built, and its elements were tested and empirically evaluated. The test-bed is now available for study of generic SLAs and associated services. Virtual wire PDB was used to implement a virtual leased line service and to demonstrate ability of the test-bed to support expedited forwarding. EF implementation was evaluated using real and emulated VoIP streams. The EF PHB was effected in two ways: a) using Priority Scheduler and b) using Self Clocked Fair Queuing SCFQ scheduler. Both implementations were tested and measurements were made for loss, delay and jitter. Proper implementation of the EF service requirements was verified using both element testing and application testing. Real and emulated VoIP streams were used in the latter case to show that they can be protected against full traffic load on a shared interface.

6.2. Assured Rate Services

Assured Rate PDB of the SLA test-bed was implemented using AF PHB. The latter was effected using Single Rate Three Color Marker SRTCM for traffic conditioning, Multiple Threshold RED for Buffer Management, and Static Priority for scheduling. The guaranteed rate for each AF flows was supported regardless of the Background Best Effort traffic, while the nonconforming AF packets were dropped . Experiments were run

to ascertain that the AF service was operational, and demonstrate its possible use. The issue of utilization of excess bandwidth and

6.3. Future work

The test-bed will be used to explore issues related to a variety of SLAs, as well as to test new and different DiffServ elements (such as schedulers), and domain-level solutions and quality of service algorithms. This work is just the beginning. One interesting area of work may be adaptive queue management (AQM) based on different variants of RED and other algorithms, addition of MPLS and QoS-enhanced MPLS functionalities to the test-bed, as well as exploration of new schedulers that would help maximize utilization of per-customer resources (such as bandwidth) on domain-wide basis and contribute to successful end-to-end quality of service implementations and guarantees.

7. References

[1] Priority Queuing Applied to Expedited Forwarding: a Measurement-Based Analysis; T. Ferrari, G. Pau, C. Raffaelli, Berlin Sep 26 2000, 1st Int. workshop on Quality of Future Intenet Services (QofIS'2000), Lecture Notes in Computer Science (Springer-Verlag) vol. 1922, pp. 167-181.

[2] R. Guerin, L. Li, S. Nadas, P. Pan, and V. Persi. The Cost of QoS Support in Edge Devices - An Experimental Study. In IEEE INFOCOM'99, New York, NY, 1999.

[3] R. Guerin and V. Peris, ***"Quality-of-Service in Packet Networks: Basic Mechanisms and Directions",*** Computer Networks 31 (1999) 169-189.

[4] S. Sahu, D. Towsley and J. Kurose, "A Quantitative Study of Differentiated Services for the Internet," CMPSCI Technical Report 99-09, University of Massachusetts, MA, 1999.

[5] M. May et al., **"Simple Performance Models of Differentiated Services Schemes for Internet",** INFOCOM'99, pp. 1385-1394

[6] R. Geurin, S. Kamat, V. Peris, and R. Rajan. *Scalable QoS Provision Through Buffer Management*. In Proceedings ACM Sigcomm'98, Sept 1998.

[7] M. Shreedhar, G. Varghese, ``Efficient Fair Queuing using Deficit Round Robin," August 1995.

[8] Voice over IP Bandwidth white paper **http://www.voipalculator.com/** support.html .

[9] D. D. Clark and W. Fang, **"Explicit Allocation of Best Effort Packet Delivery Service",** IEEE/ACM Trans. Net., Aug. 1998

[10] Bernet, Yoram., Binder, James., Blake, Steven., Carlson, Mark., Carpenter, Brian.,Keshav, Srinivasan., Davies, Elwyn., Ohlman, Borje., Varma, Dinesh., Wang, Zheng., Weiss, Walter., "A framework for Differentiated Services", Internet Draft, February 1999.

[11] Bernet, Y., Durham, D., and Reichmeyer, F., "Requirements of Diff-serv Boundary Routers", Internet Draft, November, 1998

[12] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and Weiss, W., "An Architecture for Differentiated Services", RFC 2475, December 1998.

[13] Floyd, S., and Jacobson, V., “Random Early Detection gateways for Congestion Avoidance”, IEEE/ACM Transactions on Networking, V.1 N.4, August 1993, pp. 397-413.

[14] Golestani, S.. “A self-clocked fair queueing scheme for broadband applications”. In Proceedings of IEEE INFOCOM'94, Toronto, CA, June 1994.

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

[16] Heinanen, J., and Guerin, R., "A Single Rate Three Color Marker", RFC 2697, September 1999.

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

[18] Narasimhan, Kesava Prasad.,” An Implementation of Differentiated Services In A Linux Environment.”, A thesis submitted at North Carolina State University of the requirements for the Degree of Master of Science.

[19] V. Jacobson, K. Nichols, and K. Poduri. The "virtual wire" per domain behavior. Internet draft, draft-ietf-DiffServ-pdb-vw-00.txt, work in progress, July 2000.

[20] N. Seddihg, B. Nandy, and J. Heinanen. An assured rate per-domain behavior for differentiated services. Internet draft,draft-seddigh-pdb-ar-00.txt, work in progress, October 2000.

[21] Jon Bennet..et al, “An Expedited Forwarding PHB”, Internet-Draf, April 2001.

[22] Reda N. Haddad , “ SLA to Controls Mapping in Differentiated Services”, Master of Science Thesis , North Carolina State University.

[23] Floyd, S. and V. Jacobson. "Link-sharing and Resource Management Models for Packet Networks", IEEE/ACM Transactions on Networking, Vol. 3 no. 4, pp. 365-386, August 1995.

[24] Postel, J., Ed. "Internet Protocol", STD 5, RFC 791, September 1981.

[25] Clark, D. and W. Fang. "Explicit Allocation of Best Effort Packet Delivery Service", IEEE/ACM Trans. on Networking, vol. 6, no. 4, August 1998, pp. 362-373.

[26] Braden, R., D. Clark and S. Shenker. "Integrated Services in the Internet Architecture: An Overview", RFC 1633, July 1994.

[27] Nichols, K., V. Jacobson, and L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet", INTERNET DRAFT <draft-nichols-diff-svc-arch-02>, November 1997.

[28] Braden, B., et al. "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.

[29] Fine, M., et al. " Differentiated Services Quality of Service Policy Information Base", Internet Draft <draft-ietf-DiffServ-pib-03.txt>, March 2001.

[30] Bernet, Y., et al. "An Informal Management Model for DiffServ Routers", Internet Draft <draft-ietf-DiffServ-model-06.txt>, February 2001.

[31] Stoica, Ion and Hui Zhang. "Providing Guaranteed Services without Per-Flow Management", Sigcomm 1999.

[32] Carpenter, Brian and Kathleen Nichols. "A Bulk Handling Per-Domain Behavior for Differentiated Services", Internet Draft <draft-ietf-DiffServ-pdb-bh-02.txt>, January 2001.

[33] Charny, A. ed. "EF PHB Redefined", INTERNET DRAFT <draft-charny-ef-definition-01.txt>, Nov 2000.

[34] Bless, R. and K. Wehrle. "A Lower Than Best-Effort Per-Hop Behavior", Internet Draft <draft-bless-DiffServ-phb-lbe-00.txt>, February 2001.

[35] Claffy, K., Greg Miller and Kevin Thompson. "The nature of the beast: recent traffic measurements from an Internet backbone". INET'98 paper.

[36] Goyal, M., et al. "Performance Analysis of Assured Forwarding," INTERNET DRAFT <draft-goyal-DiffServ-afstdy-00>, February 2000.

[37] Almesberger, Werner, et al. "Differentiated Services on Linux", INTERNET DRAFT <draft-almesberger-wajhak-DiffServ-linux-01.txt>, June 1999.

[38] Mercankosk, G. "The Virtual Wire Per Domain behavior Analysis and Extensions", INTERNET DRAFT <draft-mercankosk-DiffServ-pdb-vw-00.txt>, July 2000.

[39] Goderis, Danny, et al. "Service Level Specification Semantics, Parameters and negotiation requirements", INTERNET DRAFT <draft-tequila-DiffServ-sls-00.txt>, July 2000.

[40] Teitelbaum, Ben. "Qbone Architecture (v1.0)", 1999 .

[41] Zyad Dwekat, Amit Kulkarni and Mrugendra Singhai " Verification Test report for soft DiffServ router implementation ", June 2000.

[42] G. Almes, S. Kalidindi, M. Zekauskas "A One-way Delay Metric for IPPM", rfc 2679.

[43] S. Bradner, ``Benchmarking terminology for network interconnection devices," RFC 1242, Internet Engineering Task Force, July 1991.

[44] Averill M. Law and W. David Kelton, "Simulation Modeling and Analysis" , McGraw-Hill 1982.

[45] ns-2 ManualThe ns Manual , The VINT Project collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC Kevin Fall, Editor Kannan Varadhan.

[46] B. Davie, S. Davari, V. Firoiu, C. Kalmanek, K. Ramakrishnan, and D. Stiliadis, "Supplemental Information for the New Definition of EF PHB", Internet Draft, Feb. 2001.

[47] Spirent Communications, "SmartBits System Reference", February 2001 .

[48] Spirent Communications, "SmartFlow User Guide version 1.30 ", March 2001.

[49] Mark Gates, Alex Warshavsky, " Iperf version 1.1.1", NLANR , February 2000 .

Appendix A : Some Verification Testing Results

This is a sample of the verification of functionality of the DiffServ router. In this report we are testing Single rate Three color Marker and as results showed it is working fine.

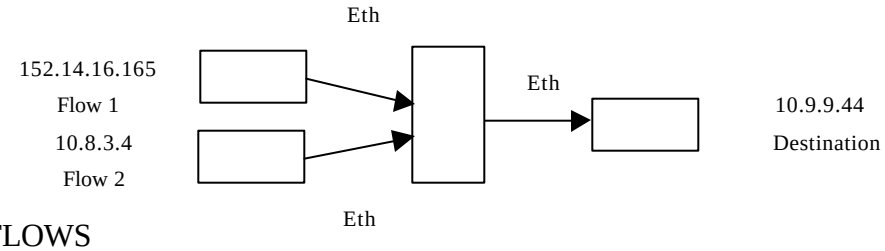
SRTCM type traffic conditioner

Test # 2 Date 11 /16 /2000

Test Description:

This test aims to verify the Single Rate Two Color Marker type of traffic conditioner.

Test Configuration:



SMART FLOWS

Number of Flows :2

Line Speed (MB)(10/100) = 10

Duration (sec): 30 Burst Size : 1

FlowID	SourceIP	DEST IP	Load %	Frame length	TOS byte
1	152.14.16.112	10.9.9.44	80	1514	0
2	10.8.3.4	10.9.9.44	80	1514	0

Classification:

Number of classified flows: 2

Flow id	Class Flag	Classification value
1	SRC_ADDR	152.14.16.112
2	SRC_ADDR	10.8.3.4

1. **Traffic Conditioner:**

Type (Dummytc,SRTCM,TRTCM): SRTCM

Flow id: 1

Cir = 3Mb cbs = 3028 ebs = 7570

Color	Action	Outflow Id
Green	Normal	1
Yellow	Drop	
Red	Drop	

Flow id: 2

Cir = 5Mb cbs = 3028 ebs = 7570

Color	Action	Outflow Id
Green	Normal	2
Yellow	Drop	
Red	Drop	

Buffer Manager:

Type (Normal,Threshold,RED,Multi_RED,Multi_threshold): Normal

Total queue size: 30280

Common parameters:

Number of queues: 2

Que_id	Que size	Que id	Que size
1	7570	2	7570

Mapping:

Outflow id	Que id	Class id
1	1	
2	2	

Link scheduler:

Type: FIFO

Results:

Flow id	Input throughput Mbps	Output throughput Mbps	Avg latency	Standard deviation
1	8 Mbps	3 Mbps		
2	8 Mbps	5 Mbps		

Conclusions:

SRTCM type traffic conditioning works. In the next stage of testing we will see the effect of varying burst size of effective bandwidth.

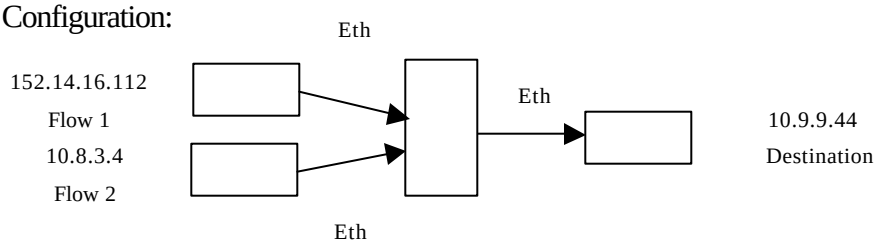
Static Priority Type Link Scheduler

Test # 7 Date 11 /23 /2000

Test Description:

This test aims to verify the Static Priority type of link scheduler.

Test Configuration:



SMART FLOWS

Number of Flows :2

Line Speed (MB)(10/100) = 10

Duration (sec): 30 Burst Size : 1

FlowID	SourceIP	DEST IP	Load %	Frame length	TOS byte
1	152.14.16.165	10.9.9.44	40 to 100 %	1514	0
2	10.8.3.4	10.9.9.44	40 to 100 %	1514	0

File Name(dstest2.flo): dstest_LS_StaticPriority.flo

DIFFSERV PARAMETERS:

File name (qdisc_test.rc): qdisc_LS_StaticPriority.rc

Classification:

Number of classified flows: 2

Flow id	Class Flag	Classification value
1	SRC_ADDR	152.14.16.165
2	SRC_ADDR	10.8.3.4

Traffic Conditioner:

Type (Dummytc,SRTCM,TRTCM): Dummytc

DUMMYTC:

Flow id: 1 & 2

Buffer Manager:

Type (Normal,Threshold,RED,Multi_RED,Multi_threshold): Normal

Total queue size: 30280

Number of queues: 2

Que_id	Que size	Que id	Que size
1	15140	2	15140

Mapping:

Outflow id	Que id	Class id

1	1	
2	2	

Link Scheduler:

Type: Static Priority

Que_id	Parameter 1= Priority	Parameter 2=	Parameter 3=	Parameter 4=
1	1			
2	1000			

Remarks:

Higher numerical value of priority indicates higher priority.

Results:

Flow id	Load %	Priority	Input throughput Mbps	Output throughput Mbps	Avg latency	Standard deviation
1	40 50 60 70 80 90 100	1	3.94 4.93 5.92 6.90 7.89 8.88 9.86	3.94 4.93 3.95 3.29 3.29 3.29 3.29		
2	40 50 60 70 80 90 100	1000	3.94 4.93 5.92 6.90 7.89 8.88 9.86	3.94 4.93 5.92 6.57 6.57 6.58 6.58		

Conclusions:

Static Priority type of link scheduler work. It forwards packet with high priority first as compared to packets from low priority.

Appedix B: Configuration Scripts file for Virtual Wire (SCFQ)

This a sample script file used to set all the DiffServ parameters onto the router. This file is for SCFQ used to implement EF PHB.

```
begin class_tc
     class_flag = DSCP
     dscp=20
     flow_id = 1
     tc_type = DUMMYTC
     begin tc_data
           outflowid = 1
     end tc_data
end class_tc
begin class_tc
     class_flag = DSCP
     dscp = b8
     flow_id = 2
     tc_type = DUMMYTC
     begin tc_data
           outflowid = 2
     end tc_data
end class_tc
begin buff_man
     type = NORMAL
     total_que_size = 15700
     shaping = false
end buff_man

begin que_data
     queue_id = 1
     queue_size = 7500
     shaping_rate = 10000000
     burst_len = 5000
end que_data

begin que_data
     queue_id = 2
     queue_size = 4480
     shaping_rate = 3000000
     burst_len = 640
end que_data
begin outflow_queue
     outflowid = 1
     queue_id = 1
end outflow_queue
begin outflow_queue
     outflowid = 2
     queue_id = 2
end outflow_queue
begin link_scheduler_header
     type = SCFQ
     burstmode = false
     decrbypktsize = false
```

```
end link_scheduler_header

begin link_scheduler_data
     queue_id = 1
     rate = 7000000
end link_scheduler_data

begin link_scheduler_data
     queue_id = 2
     rate = 3000000
end link_scheduler_data
```

Appendix C : Configuration file for Assured Rate PDB

This a sample script file used to set all the DiffServ parameters onto the router. This file is used for priority to implement AF PHB. For a full details of Scripts commands go to Kesava [18].

```
# example for ef,af,be
begin class_tc
      class_flag = DSCP
      dscp=20
      flow_id = 1
      tc_type = DUMMYTC
      begin tc_data
            outflowid = 1
      end tc_data
end class_tc
begin class_tc
      class_flag = DSCP
      dscp = b8
      flow_id = 2
      tc_type = DUMMYTC
      begin tc_data
            outflowid = 2
      end tc_data
end class_tc
begin class_tc
      class_flag = DSCP
      dscp = 38
      flow_id = 3
      tc_type = DUMMYTC
      begin tc_data
            outflowid = 3
      end tc_data
end class_tc


begin class_tc
      class_flag = DSCP
      dscp = 30
      flow_id = 4
      tc_type = DUMMYTC
      begin tc_data
            outflowid = 4
      end tc_data
end class_tc

begin class_tc
      class_flag = DSCP
      dscp = 28
      flow_id = 5
      tc_type = DUMMYTC
      begin tc_data
```

```
            outflowid = 5
      end tc_data
end class_tc


begin class_tc
      class_flag = DSCP
      dscp = 58
      flow_id = 6
      tc_type = DUMMYTC
      begin tc_data
            outflowid = 6
      end tc_data
end class_tc


begin class_tc
      class_flag = DSCP
      dscp = 50
      flow_id = 7
      tc_type = DUMMYTC
      begin tc_data
            outflowid = 7
      end tc_data
end class_tc

begin class_tc
      class_flag = DSCP
      dscp = 48
      flow_id = 8
      tc_type = DUMMYTC
      begin tc_data
            outflowid = 8
      end tc_data
end class_tc


begin class_tc
      class_flag = DSCP
      dscp = 78
      flow_id = 9
      tc_type = DUMMYTC
      begin tc_data
            outflowid = 9
      end tc_data
end class_tc


begin class_tc
      class_flag = DSCP
      dscp = 70
      flow_id = 10
      tc_type = DUMMYTC
      begin tc_data
            outflowid = 10
      end tc_data
end class_tc
```

```
begin class_tc
     class_flag = DSCP
     dscp = 68
     flow_id = 11
     tc_type = DUMMYTC
     begin tc_data
           outflowid = 11
     end tc_data
end class_tc



begin class_tc
     class_flag = DSCP
     dscp = 98
     flow_id = 12
     tc_type = DUMMYTC
     begin tc_data
           outflowid = 12
     end tc_data
end class_tc


begin class_tc
     class_flag = DSCP
     dscp = 90
     flow_id = 13
     tc_type = DUMMYTC
     begin tc_data
           outflowid = 13
     end tc_data
end class_tc

begin class_tc
     class_flag = DSCP
     dscp = 88
     flow_id = 14
     tc_type = DUMMYTC
     begin tc_data
           outflowid = 14
     end tc_data
end class_tc

begin buff_man
     type = MULTI_RED
     total_que_size = 150000
     shaping = false
end buff_man

begin que_data
     queue_id = 1
     queue_size = 15140
     shaping_rate = 10000000
     burst_len = 5000

    class1[MIN_THRESHOLD]= 15140
```

```
    class1[MAX_THRESHOLD]= 15140
    class2[MIN_THRESHOLD]= 15140
    class2[MAX_THRESHOLD]= 15140
    class3[MIN_THRESHOLD]= 15140
    class3[MAX_THRESHOLD]= 15140
    class1[WEIGHT]= 0.002
    class2[WEIGHT]= 0.002
    class3[WEIGHT]= 0.002
    class1[MAX_PROB]= 0.1
    class2[MAX_PROB]= 0.1
    class3[MAX_PROB]= 0.1
end que_data

begin que_data
     queue_id = 2
     queue_size = 3200
     shaping_rate = 3000000
     burst_len = 640

    class1[MIN_THRESHOLD]= 3200
    class1[MAX_THRESHOLD]= 3200
    class2[MIN_THRESHOLD]= 3200
    class2[MAX_THRESHOLD]= 3200
    class3[MIN_THRESHOLD]= 3200
    class3[MAX_THRESHOLD]= 3200
    class1[WEIGHT]= 0.002
    class2[WEIGHT]= 0.002
    class3[WEIGHT]= 0.002
    class1[MAX_PROB]= 0.1
    class2[MAX_PROB]= 0.1
    class3[MAX_PROB]= 0.1

end que_data

begin que_data
     queue_id = 3
     queue_size = 11000
     shaping_rate = 1000000
     burst_len = 4542

    class1[MIN_THRESHOLD]= 11000
    class1[MAX_THRESHOLD]= 11000
    class2[MIN_THRESHOLD]= 7570
    class2[MAX_THRESHOLD]= 11000
    class3[MIN_THRESHOLD]= 7570
    class3[MAX_THRESHOLD]= 11000
    class1[WEIGHT]= 0.002
    class2[WEIGHT]= 0.002
    class3[WEIGHT]= 0.002
    class1[MAX_PROB]= 0.1
    class2[MAX_PROB]= 0.1
    class3[MAX_PROB]= 0.1
end que_data

begin que_data
     queue_id = 4
     queue_size = 11000
```

```
     shaping_rate = 1000000
     burst_len = 4542

    class1[MIN_THRESHOLD]= 11000
    class1[MAX_THRESHOLD]= 11000
    class2[MIN_THRESHOLD]= 7570
    class2[MAX_THRESHOLD]= 11000
    class3[MIN_THRESHOLD]= 7570
    class3[MAX_THRESHOLD]= 11000
    class1[WEIGHT]= 0.002
    class2[WEIGHT]= 0.002
    class3[WEIGHT]= 0.002
    class1[MAX_PROB]= 0.1
    class2[MAX_PROB]= 0.1
    class3[MAX_PROB]= 0.1
end que_data


begin que_data
     queue_id = 5
     queue_size = 11000
     shaping_rate = 1000000
     burst_len = 4542

    class1[MIN_THRESHOLD]= 11000
    class1[MAX_THRESHOLD]= 11000
    class2[MIN_THRESHOLD]= 7570
    class2[MAX_THRESHOLD]= 11000
    class3[MIN_THRESHOLD]= 7570
    class3[MAX_THRESHOLD]= 11000
    class1[WEIGHT]= 0.002
    class2[WEIGHT]= 0.002
    class3[WEIGHT]= 0.002
    class1[MAX_PROB]= 0.1
    class2[MAX_PROB]= 0.1
    class3[MAX_PROB]= 0.1
end que_data


begin que_data
     queue_id = 6
     queue_size = 11000
     shaping_rate = 1000000
     burst_len = 4542

    class1[MIN_THRESHOLD]= 11000
    class1[MAX_THRESHOLD]= 11000
    class2[MIN_THRESHOLD]= 7570
    class2[MAX_THRESHOLD]= 11000
    class3[MIN_THRESHOLD]= 7570
    class3[MAX_THRESHOLD]= 11000
    class1[WEIGHT]= 0.002
    class2[WEIGHT]= 0.002
    class3[WEIGHT]= 0.002
    class1[MAX_PROB]= 0.1
    class2[MAX_PROB]= 0.1
    class3[MAX_PROB]= 0.1
```

```
end que_data

begin outflow_queue
     outflowid = 1
     queue_id = 1
        class_id = 1
end outflow_queue
begin outflow_queue
     outflowid = 2
     queue_id = 2
        class_id = 1
end outflow_queue
begin outflow_queue
     outflowid = 3
     queue_id = 3
        class_id = 3
end outflow_queue

begin outflow_queue
     outflowid = 4
     queue_id = 3
        class_id = 2
end outflow_queue

begin outflow_queue
     outflowid = 5
     queue_id = 3
        class_id = 1
end outflow_queue
begin outflow_queue
     outflowid = 6
     queue_id = 4
        class_id = 3
end outflow_queue
begin outflow_queue
     outflowid = 7
     queue_id = 4
        class_id = 2
end outflow_queue
begin outflow_queue
     outflowid = 8
     queue_id = 4
        class_id = 1
end outflow_queue

begin outflow_queue
     outflowid = 9
     queue_id = 5
        class_id = 3
end outflow_queue
begin outflow_queue
     outflowid = 10
     queue_id = 5
        class_id = 2
end outflow_queue
begin outflow_queue
     outflowid = 11
```

```
     queue_id = 5
        class_id = 1
end outflow_queue


begin outflow_queue
     outflowid = 12
     queue_id = 6
        class_id = 3
end outflow_queue
begin outflow_queue
     outflowid = 13
     queue_id = 6
        class_id = 2
end outflow_queue
begin outflow_queue
     outflowid = 14
     queue_id = 6
        class_id = 1
end outflow_queue


begin link_scheduler_header
     type = PRIORITY
     burstmode = false
     decrbypktsize = false
end link_scheduler_header

begin link_scheduler_data
     queue_id = 1
     priority=10
end link_scheduler_data

begin link_scheduler_data
     queue_id = 2
     priority=20
end link_scheduler_data

begin link_scheduler_data
     queue_id = 3
     priority=19
end link_scheduler_data

begin link_scheduler_data
     queue_id = 4
     priority=18
end link_scheduler_data

begin link_scheduler_data
     queue_id = 5
     priority=17
end link_scheduler_data

begin link_scheduler_data
     queue_id = 6
     priority=16
end link_scheduler_data
```

This file is for setting Integrated Scheduler configuration:

```
# example for ef,af,be
begin class_tc
     class_flag = DSCP
     dscp=20
     flow_id = 1
     tc_type = DUMMYTC
     begin tc_data
           outflowid = 1
     end tc_data
end class_tc
begin class_tc
     class_flag = DSCP
     dscp = b8
     flow_id = 2
     tc_type = DUMMYTC
     begin tc_data
           outflowid = 2
     end tc_data
end class_tc
begin class_tc
     class_flag = DSCP
     dscp = 80
     flow_id = 3
     tc_type = SRTCM
     begin tc_data
           cir = 1000000
           cbs = 3028
           ebs = 6056
           action[RED] = NORMAL
           outflowid[RED] = 3
           out_dscp[RED] = 38
           action[YELLOW] = NORMAL
           outflowid[YELLOW] = 4
           out_dscp[YELLOW] = 30
           action[GREEN] = NORMAL
           outflowid[GREEN] = 5
           out_dscp[GREEN] = 28
     end tc_data
end class_tc

begin class_tc
     class_flag = DSCP
     dscp = 84
     flow_id = 4
     tc_type = SRTCM
     begin tc_data
           cir = 1000000
           cbs = 3028
           ebs = 6056
           action[RED] = NORMAL
           outflowid[RED] = 6
           out_dscp[RED] = 58
           action[YELLOW] = NORMAL
           outflowid[YELLOW] = 7
```

```
            out_dscp[YELLOW] = 50
            action[GREEN] = NORMAL
            outflowid[GREEN] = 8
            out_dscp[GREEN] = 48
      end tc_data
end class_tc
begin class_tc
      class_flag = DSCP
      dscp = 88
      flow_id = 5
      tc_type = SRTCM
      begin tc_data
            cir = 1000000
            cbs = 3028
            ebs = 6056
            action[RED] = NORMAL
            outflowid[RED] = 9
            out_dscp[RED] = 78
            action[YELLOW] = NORMAL
            outflowid[YELLOW] = 10
            out_dscp[YELLOW] = 70
            action[GREEN] = NORMAL
            outflowid[GREEN] = 11
            out_dscp[GREEN] = 68
      end tc_data
end class_tc
begin class_tc
      class_flag = DSCP
      dscp = 8c
      flow_id = 6
      tc_type = SRTCM
      begin tc_data
            cir = 1000000
            cbs = 3028
            ebs = 6056
            action[RED] = NORMAL
            outflowid[RED] = 12
            out_dscp[RED] = 98
            action[YELLOW] = NORMAL
            outflowid[YELLOW] = 13
            out_dscp[YELLOW] = 90
            action[GREEN] = NORMAL
            outflowid[GREEN] = 14
            out_dscp[GREEN] = 88
      end tc_data
end class_tc
begin buff_man
      type = MULTI_RED
      total_que_size = 150000
      shaping = false
end buff_man

begin que_data
      queue_id = 1
      queue_size = 15140
      shaping_rate = 10000000
      burst_len = 5000
```

```
    class1[MIN_THRESHOLD]= 15140
    class1[MAX_THRESHOLD]= 15140
    class2[MIN_THRESHOLD]= 15140
    class2[MAX_THRESHOLD]= 15140
    class3[MIN_THRESHOLD]= 15140
    class3[MAX_THRESHOLD]= 15140
    class1[WEIGHT]= 0.002
    class2[WEIGHT]= 0.002
    class3[WEIGHT]= 0.002
    class1[MAX_PROB]= 0.1
    class2[MAX_PROB]= 0.1
    class3[MAX_PROB]= 0.1
end que_data

begin que_data
     queue_id = 2
     queue_size = 3200
     shaping_rate = 3000000
     burst_len = 640

    class1[MIN_THRESHOLD]= 3200
    class1[MAX_THRESHOLD]= 3200
    class2[MIN_THRESHOLD]= 3200
    class2[MAX_THRESHOLD]= 3200
    class3[MIN_THRESHOLD]= 3200
    class3[MAX_THRESHOLD]= 3200
    class1[WEIGHT]= 0.002
    class2[WEIGHT]= 0.002
    class3[WEIGHT]= 0.002
    class1[MAX_PROB]= 0.1
    class2[MAX_PROB]= 0.1
    class3[MAX_PROB]= 0.1

end que_data

begin que_data
     queue_id = 3
     queue_size = 11000
     shaping_rate = 1000000
     burst_len = 4542

    class1[MIN_THRESHOLD]= 11000
    class1[MAX_THRESHOLD]= 11000
    class2[MIN_THRESHOLD]= 7570
    class2[MAX_THRESHOLD]= 11000
    class3[MIN_THRESHOLD]= 7570
    class3[MAX_THRESHOLD]= 11000
    class1[WEIGHT]= 0.002
    class2[WEIGHT]= 0.002
    class3[WEIGHT]= 0.002
    class1[MAX_PROB]= 0.1
    class2[MAX_PROB]= 0.1
    class3[MAX_PROB]= 0.1
end que_data

begin que_data
```

```
     queue_id = 4
     queue_size = 11000
     shaping_rate = 1000000
     burst_len = 4542

    class1[MIN_THRESHOLD]= 11000
    class1[MAX_THRESHOLD]= 11000
    class2[MIN_THRESHOLD]= 7570
    class2[MAX_THRESHOLD]= 11000
    class3[MIN_THRESHOLD]= 7570
    class3[MAX_THRESHOLD]= 11000
    class1[WEIGHT]= 0.002
    class2[WEIGHT]= 0.002
    class3[WEIGHT]= 0.002
    class1[MAX_PROB]= 0.1
    class2[MAX_PROB]= 0.1
    class3[MAX_PROB]= 0.1
end que_data


begin que_data
     queue_id = 5
     queue_size = 11000
     shaping_rate = 1000000
     burst_len = 4542

    class1[MIN_THRESHOLD]= 11000
    class1[MAX_THRESHOLD]= 11000
    class2[MIN_THRESHOLD]= 7570
    class2[MAX_THRESHOLD]= 11000
    class3[MIN_THRESHOLD]= 7570
    class3[MAX_THRESHOLD]= 11000
    class1[WEIGHT]= 0.002
    class2[WEIGHT]= 0.002
    class3[WEIGHT]= 0.002
    class1[MAX_PROB]= 0.1
    class2[MAX_PROB]= 0.1
    class3[MAX_PROB]= 0.1
end que_data


begin que_data
     queue_id = 6
     queue_size = 11000
     shaping_rate = 1000000
     burst_len = 4542

    class1[MIN_THRESHOLD]= 11000
    class1[MAX_THRESHOLD]= 11000
    class2[MIN_THRESHOLD]= 7570
    class2[MAX_THRESHOLD]= 11000
    class3[MIN_THRESHOLD]= 7570
    class3[MAX_THRESHOLD]= 11000
    class1[WEIGHT]= 0.002
    class2[WEIGHT]= 0.002
    class3[WEIGHT]= 0.002
    class1[MAX_PROB]= 0.1
```

```
    class2[MAX_PROB]= 0.1
    class3[MAX_PROB]= 0.1
end que_data

begin outflow_queue
     outflowid = 1
     queue_id = 1
        class_id = 1
end outflow_queue
begin outflow_queue
     outflowid = 2
     queue_id = 2
        class_id = 1
end outflow_queue
begin outflow_queue
     outflowid = 3
     queue_id = 3
        class_id = 3
end outflow_queue

begin outflow_queue
     outflowid = 4
     queue_id = 3
        class_id = 2
end outflow_queue

begin outflow_queue
     outflowid = 5
     queue_id = 3
        class_id = 1
end outflow_queue
begin outflow_queue
     outflowid = 6
     queue_id = 4
        class_id = 3
end outflow_queue
begin outflow_queue
     outflowid = 7
     queue_id = 4
        class_id = 2
end outflow_queue
begin outflow_queue
     outflowid = 8
     queue_id = 4
        class_id = 1
end outflow_queue

begin outflow_queue
     outflowid = 9
     queue_id = 5
        class_id = 3
end outflow_queue
begin outflow_queue
     outflowid = 10
     queue_id = 5
        class_id = 2
end outflow_queue
```

```
begin outflow_queue
     outflowid = 11
     queue_id = 5
        class_id = 1
end outflow_queue
begin outflow_queue
     outflowid = 12
     queue_id = 6
        class_id = 3
end outflow_queue
begin outflow_queue
     outflowid = 13
     queue_id = 6
        class_id = 2
end outflow_queue
begin outflow_queue
     outflowid = 14
     queue_id = 6
        class_id = 1
end outflow_queue
begin link_scheduler_header
     type = PRIORITY
     burstmode = false
     decrbypktsize = false
end link_scheduler_header
begin link_scheduler_data
     queue_id = 1
     priority=10
end link_scheduler_data
begin link_scheduler_data
     queue_id = 2
     priority=20
end link_scheduler_data
begin link_scheduler_data
     queue_id = 3
     priority=19
     AF_flag = true
        red_threshold = 10

end link_scheduler_data

begin link_scheduler_data
     queue_id = 4
     priority=18
     AF_flag = true
        red_threshold = 10

end link_scheduler_data

begin link_scheduler_data
     queue_id = 5
     priority=17
     AF_flag = true
        red_threshold = 10

end link_scheduler_data
```

```
begin link_scheduler_data
     queue_id = 6
     priority=16
     AF_flag = true
        red_threshold = 10

end link_scheduler_data
```

Appendix D : ns simulator sample script

(Priority scheduler with two queues EF queue and BE queue).

```
#---------------------------------------------------------------------------
---------------------------------

# Author     : Zyad Dwekat
# Date       : 26th September, 2001
# Notes      : The NCSU DiffServ tesbed
# Description: Experiment 1.EF rate 3 Mbps BE rate 9 Mbps EF packet
#size 128 BE packet size 128
#
#---------------------------------------------------------------------------
---------------------------------


set ns [new Simulator]

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

set  f [open out128.tr w]
set f1 [open EF.tr w]
set f2 [open BE.tr w]
$ns trace-all $f

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


proc finish { } {
       global  f1 f2  ns nf
       $ns flush-trace
            close $nf
            close $f1
            close $f2
            exec nam out.nam &
            exec xgraph EF.tr BE.tr  -geometry 1200x800 &
            exit 0
            }


set cir1  0
set cbs1  0
set rate1 0

set cir2  0
set cbs2  0
```

```
set rate2 0

set cir3  0
set cbs3  0
set rate3 0

set cir4  10000000
set cbs4  150000
set ef_rate 3000000
set be_rate 9000000

set testTime 30.0
set ef_packetSize 128
set be_packetSize 128
set packetSize 1500

# Set up the network topology
set s1 [$ns node]
set s2 [$ns node]
set s3 [$ns node]
set s4 [$ns node]

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


set core1 [$ns node]
set core2 [$ns node]
set core3 [$ns node]


#CONFIGURING THE LINKS

$ns duplex-link $s1 $e1 20Mb 0.1ms DropTail
$ns duplex-link $s3 $e1 20Mb 0.1ms DropTail
$ns duplex-link $s2 $e2 20Mb 0.1ms DropTail
$ns duplex-link $s4 $e2 20Mb 0.1ms DropTail


$ns simplex-link $e1 $core1 10Mb 0.1ms dsRED/edge
$ns simplex-link $core1 $e1 10Mb 0.1ms dsRED/core
$ns simplex-link $e2 $core3 10Mb 0.1ms dsRED/edge
$ns simplex-link $core3 $e2 10Mb 0.1ms dsRED/core

$ns simplex-link $core1 $core2 10Mb 0.1ms dsRED/core
$ns simplex-link $core2 $core1 10Mb 0.1ms dsRED/core

$ns simplex-link $core2 $core3 10Mb 0.1ms dsRED/core
$ns simplex-link $core3 $core2 10Mb 0.1ms dsRED/core


#ORIENTATION OF THE LINKS

$ns duplex-link-op $s1 $e1 orient down-right
$ns duplex-link-op $s3 $e1 orient up-right

$ns simplex-link-op $e1 $core1 orient right
```

```
$ns simplex-link-op $core1 $e1 orient left

$ns simplex-link-op $core1 $core2 orient right
$ns simplex-link-op $core2 $core1 orient left

$ns simplex-link-op $core2 $core3 orient right
$ns simplex-link-op $core3 $core2 orient left


$ns simplex-link-op $core3 $e2 orient right
$ns simplex-link-op $e2 $core3 orient left

$ns duplex-link-op $s2 $e2 orient down-left

$ns duplex-link-op $s4 $e2 orient up-left



#POSITIONING THE QUEUES
$ns simplex-link-op $e1 $core1 queuePos 0.5
$ns simplex-link-op $core1 $core2 queuePos 0.5
$ns simplex-link-op $core2 $core3 queuePos 0.5
$ns simplex-link-op $core3 $e2 queuePos 0.5

$ns simplex-link-op $core1 $e1 queuePos 1.5
$ns simplex-link-op $core2 $core1 queuePos 1.5
$ns simplex-link-op $core3 $core2 queuePos 1.5
$ns simplex-link-op $e2 $core3 queuePos 1.5



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

set qC1C2 [[$ns link $core1 $core2] queue]
set qC2C1 [[$ns link $core2 $core1] queue]

set qC2C3 [[$ns link $core2 $core3] queue]
set qC3C2 [[$ns link $core3 $core2] queue]

set qE2C3 [[$ns link $e2 $core3] queue]
set qC3E2 [[$ns link $core3 $e2] queue]


#############################################################################
####
# CONFIGURING THE DIFFSERV PARAMETERS


# For the DiffServ Cloud
# Set DS RED parameters from Edge1 to Core1:
$qE1C1 setSchedularMode PRI
$qE1C1 addQueueRate 0 3000000
$qE1C1 meanPktSize $packetSize
$qE1C1 set numQueues_ 2
$qE1C1 setNumPrec 2
```

```
$qE1C1 addPolicyEntry [$s1 id] [$s2 id] TokenBucket 10 $cir4 $cbs4
$qE1C1 addPolicyEntry [$s3 id] [$s4 id] TokenBucket 20 $cir4 $cbs4
$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
$qC1E1 configQ 0 0 35 35 0.9
$qC1E1 configQ 0 1 35 35 0.9
$qC1E1 configQ 1 0 10 10 0.9
$qC1E1 configQ 1 1 10 10 0.9

# Set DS RED parameters from Core1 to Edge1:
$qC1E1 setSchedularMode PRI
$qC1E1 addQueueRate 0 3000000
$qC1E1 meanPktSize $packetSize
$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 35 35 0.9
$qC1E1 configQ 0 1 35 35 0.9
$qC1E1 configQ 1 0 10 10 0.9
$qC1E1 configQ 1 1 10 10 0.9

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

# Set DS RED parameters from Edge2 to Core3:
$qE2C3 setSchedularMode PRI
$qE2C3 addQueueRate 0 3000000
$qE2C3 meanPktSize $packetSize
$qE2C3 set numQueues_ 2
$qE2C3 setNumPrec 2
$qE2C3 addPolicyEntry [$s2 id] [$s1 id] TokenBucket 10 $cir4 $cbs4
$qE2C3 addPolicerEntry TokenBucket 10 11
$qE2C3 addPolicerEntry TokenBucket 20 21
$qE2C3 addPHBEntry 10 0 0
$qE2C3 addPHBEntry 11 0 1
$qE2C3 addPHBEntry 20 1 0
$qE2C3 addPHBEntry 21 1 1
$qC1E1 configQ 0 0 35 35 0.9
$qC1E1 configQ 0 1 35 35 0.9
$qC1E1 configQ 1 0 10 10 0.9
$qC1E1 configQ 1 1 10 10 0.9

# Set DS RED parameters from Core3 to Edge2:
$qC3E2 setSchedularMode PRI
$qC3E2 addQueueRate 0 3000000
$qC3E2 meanPktSize $packetSize
$qC3E2 set numQueues_ 2
$qC3E2 setNumPrec 2
$qC3E2 addPHBEntry 10 0 0
$qC3E2 addPHBEntry 11 0 1
$qC3E2 addPHBEntry 20 1 0
```

```
$qC3E2 addPHBEntry 21 1 1
$qC1E1 configQ 0 0 35 35 0.9
$qC1E1 configQ 0 1 35 35 0.9
$qC1E1 configQ 1 0 10 10 0.9
$qC1E1 configQ 1 1 10 10 0.9

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

# Set DS RED parameters from Core1 to core2:
$qC1C2 setSchedularMode PRI
$qC1C2 addQueueRate 0 3000000
$qC1C2 meanPktSize $packetSize
$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
$qC1E1 configQ 0 0 35 35 0.9
$qC1E1 configQ 0 1 35 35 0.9
$qC1E1 configQ 1 0 10 10 0.9
$qC1E1 configQ 1 1 10 10 0.9

# Set DS RED parameters from Core2 to core1:
$qC2C1 setSchedularMode PRI
$qC2C1 addQueueRate 0 3000000
$qC2C1 meanPktSize $packetSize
$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
$qC1E1 configQ 0 0 35 35 0.9
$qC1E1 configQ 0 1 35 35 0.9
$qC1E1 configQ 1 0 10 10 0.9
$qC1E1 configQ 1 1 10 10 0.9

##########


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

# Set DS RED parameters from Core2 to core3:
$qC2C3 setSchedularMode PRI
$qC2C3 addQueueRate 0 3000000
$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
$qC1E1 configQ 0 0 35 35 0.9
$qC1E1 configQ 0 1 35 35 0.9
$qC1E1 configQ 1 0 10 10 0.9
$qC1E1 configQ 1 1 10 10 0.9
```

```
# Set DS RED parameters from Core3 to core2:
$qC3C2 setSchedularMode PRI
$qC3C2 addQueueRate 0 3000000
$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
$qC1E1 configQ 0 0 35 35 0.9
$qC1E1 configQ 0 1 35 35 0.9
$qC1E1 configQ 1 0 10 10 0.9
$qC1E1 configQ 1 1 10 10 0.9

##########


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

# SET UP A CBR CONNECTIONS

set udp1 [new Agent/UDP]
$ns attach-agent $s1 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
$cbr1 set packet_size_ $ef_packetSize
$udp1 set packetSize_ $ef_packetSize
$cbr1 set rate_ $ef_rate
set null1 [new Agent/LossMonitor]
$ns attach-agent $s2 $null1
$ns connect $udp1 $null1

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


set udp2 [new Agent/UDP]
$ns attach-agent $s3 $udp2
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2
$cbr2 set packet_size_ $be_packetSize
$udp2 set packetSize_ $be_packetSize
$cbr2 set rate_ $be_rate
set null2 [new Agent/LossMonitor]
$ns attach-agent $s4 $null2
$ns connect $udp2 $null2

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


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

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

$qE1C1 printPolicyTable
$qE1C1 printPolicerTable
$qE1C1 printPHBTable PRI
$qE2C3 printPolicyTable
$qE2C3 printPolicerTable
$qE2C3 printPHBTable PRI

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

$ns at 30.0 "$qE1C1 printStats"
$ns at 30.0 "$qC1C2 printStats"

$ns at 30.0 "$qC2C3 printStats"
$ns at 30.0 "$qC3E2 printStats"

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

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

$ns run
```