IMPLEMENTAION OF A DIFFSERV FRMAEWORK FOR REAL TIME IP TRAFFIC

PROJECT WORK DONE AT

KUMARAGURU COLLEGE OF TECHNOLOGY

PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF THE DEGREE OF

MASTER OF ENGINEERING

OF BHARATHIAR UNIVERSITY , COIMBATORE.

P-654

SUBMITTED BY

V.VANITHA

Reg. No. 0037K0014

GUIDED BY

Mrs.L.S. JAYASHREE M.E.

Department of Computer Science & Engineering

KUMARAGURU COLLEGE OF TECHNOLOGY

Department of Computer Science & Engineering

Kumaraguru College Of Technology

(Affiliated to the Bharathiar University)

Coimbatore 641 006

CERTIFICATE

This is to certify that the project work entitled

IMPLEMENTATION OF A DIFFSERV FRAMEWORK FOR REAL TIME IP TRAFFIC

Done by

V.Vanitha

Reg. No. 0037K0014

Submitted in partial fulfillment of the requirements for the award of the degree of

Master of Engineering of Bharathiar University.

18/12/01

Professor and Head

18/12/01

Internal Guide

Submitted for University Examination held on 20.12.2001

20/12/01

Internal Examinar

External Examiner

CONTENTS

SYNOPSIS

Internet architecture offers a very simple point to point delivery service, which is based on the best effort delivery model. In this model, the highest guarantee the network provides is the reliable data delivery. This is adequate for traditional applications like ftp and telnet, which requires correct data delivery than prompt delivery. However several new real time applications like VoIP, Video conferencing, Video on demand are sensitive to the quality of service (QoS) they receive from the network. For example, E-mail is considered critical data but if it is delayed by 200 ms, it probably won't matter. But applications, which require two way communications like VOIP, experience quality degradation if the packets are delayed for that long.

In order to manage the multitude of applications, a network requires quality of service in addition to Best effort service. Thus before these applications can be widely used, the Internet infrastructure must be modified to support real time QoS and controlled end-to-end delays.

To facilitate true end-to-end QoS on an IP network, the Internet Engineering Task Force (IETF) has defined two models. (1) Integrated Service (IntServ) and (2) Differentiated Service (DiffServ). IntServ follows the signaled QoS model, where the end-host signal their QoS need to the network. On the other hand DiffServ works on the provisioned QoS model, Where network elements are set up to service multiple classes of traffic, with varying QoS requirements.

The main objective of this project is to build a DiffServ model for a network infrastructure incorporating real time applications in addition to traditional applications.

1. INTRODUCTION

1.1 THE CURRENT STATUS OF THE PROBLEM TAKEN UP

The Internet protocol (IP), and the architecture of Internet itself, is based on the simple concept that datagrams with source and destination address can traverse a network of (IP) routers independently, without the help of their sender or receiver. The reason for IP is simple is because it doesn't provide many services. IP provides addressing , and that enables the independence of each datagram. IP can fragment datagrams (in routers) and reassemble them (at the receiver), and that allows traversal of different network media.

But IP does not provide reliable data delivery. Routers are allowed to discard IP datagrams without notice to sender or receiver. Also Internet architecture does not provide any guarantee about when data will arrive or how much it can deliver. The highest guarantee the network provides is the reliable data delivery. This limitation has not been problem for traditional Internet applications. But the new breed of applications including audio and video demand high data throughput capacity (Bandwidth) and have low latency requirements when used in two-way communications. IP relies on upper level transports to keep track of datagrams and retransmit as necessary. This will assure only data delivery. Neither IP nor TCP can ensure timely delivery or provide any guarantees about the data throughput.

IP packet handling for real time applications have been already attempted using techniques and protocols like IntServ, TOS/IP Precedence, DiffServ, MPLS, etc and that results of such attempts have been published in research papers.

1.2 RELEVANCE AND IMPORTANCE OF THE TOPIC:

The Internet is changing every aspect of our lives – business, entertainments, and more. The increasing popularity of IP has shifted the paradigm from "IP over everything" to "Every thing over IP". Different applications have varying needs for delay, delay variation (jitter), bandwidth, packet loss, and availability. These parameters form the basis of QoS. IP network should be designed to provide the requisite QoS to applications.

1.2.1 DEFINITION FOR QoS

QoS means providing consistent, predictable data delivery service. In other words, satisfying customer application requirements.

QoS is the ability of a network element (e.g. an application, host or router) to have some level of assurance that its traffic and service requirements can be satisfied. There are number of characteristics that qualify QOS.

1.Minimizing delivery delay.

2.Minimizing delay variations.

3.Providing consistent data throughput capacity

1.2.2 QoS PARAMETERS

Latency

The time between a node sending a message and receipt of the message by another node.

Jitter

The distortion of a signal (video or voice) as it is propagated through the network, where the signal varies from its original reference timing, or order.

Bandwidth

The Maximum capacity of a connection. It is the measure of data transmission capacity. It can be visualized as a pipe that transfers data. The larger the pipe, the more data can be sent through it.

Packet loss

Example: 1% or less on network-wide monthly average packet loss.

Availability

Example: 99.9% premises to pop.

1.2.3 QoS Architecture Models for Traffic Engineering

Some applications are more stringent about their QoS requirements than others, and for this reason we have three basic types of QoS available

Over provisioning

The obvious solution to handle peak periods is to over-provision the network, to provide surplus bandwidth capacity in anticipation of these peak data rates during high-demand periods. But this is not economical and realistic alternative.

Resource reservation based – Integrated services (IntServ) Architecture

Network resources are apportioned according to an applications QoS request, and subject to bandwidth management policy. This model relies on the Resource Reservation Protocol (RSVP) to signal and reserve the desired QoS for each flow in the network. A flow is defined as an individual, unidirectional data stream between two applications.

Prioritization – differentiated service (DiffServ) Architecture

Network traffic is classified and apportioned network resources according to bandwidth management policy criteria. To enable QoS, network elements give preferential treatment to classifications identified as having more demanding requirements.

common set of service provisioning policies and PHB definitions.

DS egress node :A DS boundary node in its role in handling traffic as it leaves a DS domain.

DS ingress node :A DS boundary node in its role in handling traffic as it enters a DS domain.

DS interior node :A DS node that is not a DS boundary node.

DS field :The IPv4 TOS octet or the IPv6 Traffic Class octet when interpreted in conformance with definition given in RFC 2474.

DS region :A set of contiguous DS domain, which can offer differentiated services over paths across those DS domains.

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

2.LITERATURE SURVEY

Based on the literature survey conducted through publications and online materials, it was found that the following contributions have been made till date on this particular area of work.

2.1 IETF "Integrated Service" Working group

Integrated Service(IntServ)

IntServ follows the signaling QoS model, Where the end host signal their QoS need to the network. IntServ model relies on the Resource Reservation protocol (RSVP) to signal and reserve the desired QoS for each flow in the network.

RSVP

It is a signaling protocol that provides Reservation setup and control to enable the IntServ. RSVP is the most complex of all QoS technologies. As a result it provides the highest level of QoS in terms of service guarantees, granularity of resource allocation and detail of feedback to QoS enabled applications and users.

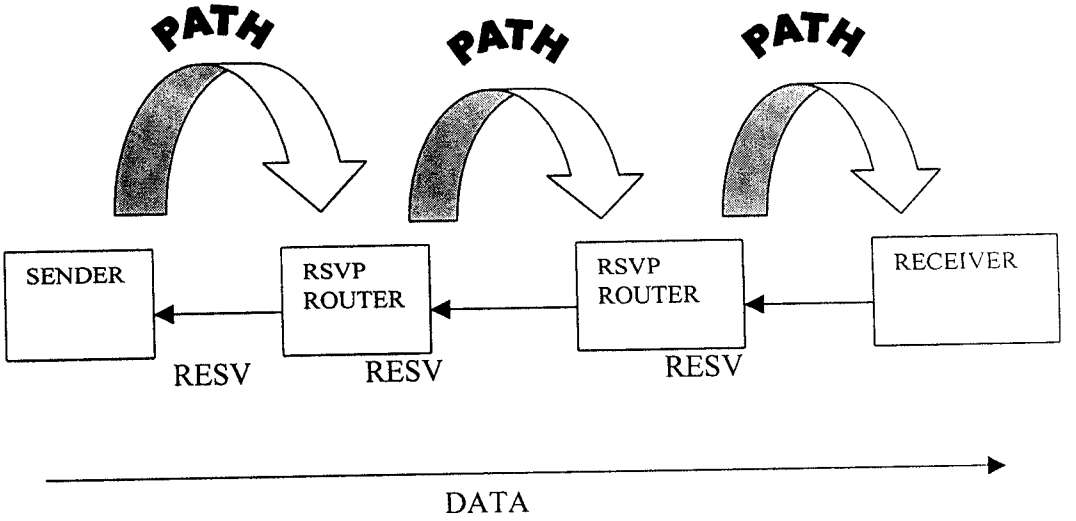

Figure 2.1: The Basic Exchange of RSVP messages to reserve resources in the network.The host transmits the Path message to the receiver, and the RESV message

1.Senders characterize outgoing traffic in terms of the upper and lower bounds of bandwidth, delay and jitter. RSVP sends a PATH message from sender that contains traffic specification (TSpec) information to the destination address. Each RSVP enabled router along the downstream route establishes a "path-state" that includes the previous source address of the PATH message (i.e. the next hop "upstream" towards the sender)

2.To make a resource reservation receivers send a RESV (Reservation request) message "upstream". In addition to the TSpec, the RESV message includes a request specification (Rspec) that indicates the type of Integrated service required-either controlled load or guaranteed-and a filter specification that characterizes the packets for which the reservation is being made (e.g. the transport protocol and port number). Together, the Rspec and filter spec represent a flow-descriptor that routers to identify each reservation.

3.When each RSVP router along the upstream path receives the RESV message, it uses the admission control process to authenticate the request and allocate the necessary resources. If the request cannot be satisfied (due to lack of resources or authorization failure). The router returns an error back to the receiver. If accepted, the router sends the RESV upstream to the next router.

4.When the last router receives the RESV and accepts the request, it sends a confirmation message back to the receiver

5.There is an explicit tear-down process for a reservation when sender or receiver ends an RSVP session.

Drawbacks

1.Reservation in each router is "soft" which means they need to be refreshed periodically by the receiver(s).

2.Reservations are receiver-based, in order to efficiently accommodate large heterogeneous receivers groups.

3.Although RSVP traffic can traverse non-RSVP routers, this creates a "weak-link" in the QoS chain where the service falls-back to "best effort"(i.e. there is no resource allocation across these links).

4.Since state information for each reservation needs to be maintained at every router along the path, scalability with hundreds of thousands of flows through a network core becomes an issue.

2.2 IETF "TOS/IP PRECEDENCE" WORKING GROUP

The TOS/IP PRECEDENCE SOLUTION

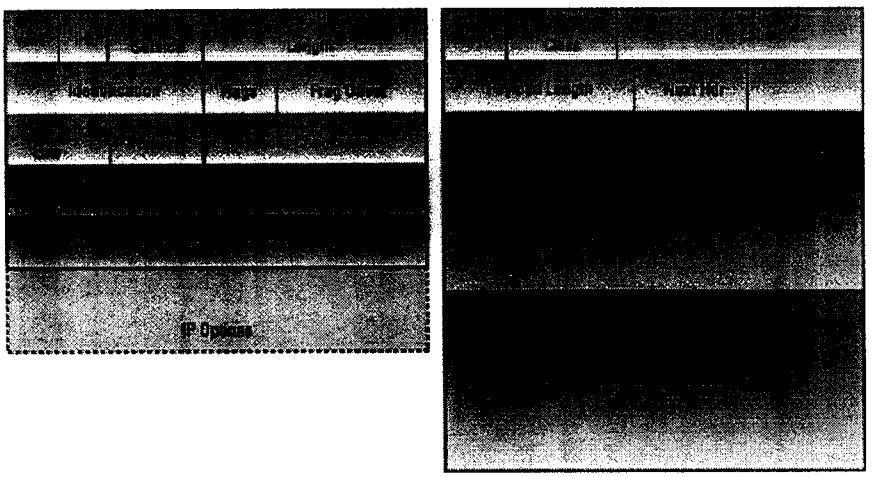

Figure 2.2 :Packet Format of IPv4 and IPV6

The three precedence bits are mainly used to classify packets at the edge of the network into one of the eight possible categories listed above. Packets of lower precedence (lower values) can be dropped in favor of higher precedence when there is congestion on a network. Further, each packet may be marked to receive one of two levels of delay, throughput, and reliability (the DTS bits) in its forwarding (RFC 791). However, this scheme has a few crucial limitations.

- The IP-Precedence scheme allows only specification of relative priority of a packet. It has no provisions to specify different drop precedence for packets of a certain priority. For example, a network administrator may want to specify both HTTP and Telnet traffic as high-priority packets. However, when there is congestion he/she wants the telnet packets to be dropped, before the HTTP (a valid reason may be because HTTP are carrying e-commerce traffic, while the

- Telnet packets are carrying user-sessions within the company for their enterprise resource planning [ERP] application). It is not possible to do this with the IP-Precedence scheme.

- The 3 bits restrict the number of possible priority classes to eight. Further, the Network Control and Internetwork Control classes are usually reserved for router-generated packets such as routing updates, ICMP messages, etc. This is done to protect the packets that are necessary for the health of the network. However, this cuts down the usable classes for production traffic to 6.

- Neither IP-Precedence, nor the DTS bits (bits 3,4,5—the original type of service subfield) are implemented consistently by network vendors today. In addition, RFC-1349 redefines the type of service subfield, by utilizing bits 3,4,5, and 6, and eliminating the DTS concept.

All of the above reduce the chances of successfully implementing end-to-end QoS using this scheme.

2.3 IETF "Multiprotocol Label Switching" working group

MULTIPROTOCOL LABEL SWITCHING (MPLS)

MPLS defines a protocol solution to improve and simplify the packet forwarding function and to provide sufficient network guarantees to support desired quality of service. To achieve this goal, MPLS adds connection-oriented mechanism to connectionless network protocols for traffic engineering and traffic management. The connection oriented mechanisms identify pre-determined paths through the network between any two end-points. Each path is then assigned a label known as Label Switched Path (LSP).

At each ingress point of the network, an edge router known as Label Edge Router (LER) examines the IP header to determine the LSP. The LER then encapsulates the packet with MPLS header containing the label information, and the packet is forwarded to the next hop. All subsequent routers known as Label Switch Routers (LSR) use the

label information from the header to determine the outgoing link and the new label for that outgoing link. The router then swaps the label in the MPLS header with the new label, and forwards the packet.

A more complex aspect of MPLS involves the distribution and management of labels among MPLS routers, to ensure they agree on the meaning of various labels. The Label Distribution Protocol (LDP) is specifically designed for this purpose.

3. PROPOSED LINE OF ATTACK

IETF has proposed many solutions to meet the demand for QoS. Significant ones among them are the IntServ and DiffServ. Intserv is characterized by resource reservation. For real time applications before data is transmitted the application must first setup paths and reserve resources. IntServ uses RSVP (Resource Reservation Protocol) for setting up paths and reserve resources. It requires a fundamental change to the current design of the Internet, by proposing the maintenance of per-flow state in the routers. This obviously was not scalable in the Internet where number of entries in the routing table is very large. Also intServ is very complex of all QoS technology and its implementation is very costly.

Since the Internet is a truly dynamic entity with its growth rate being exponential, and this growth is expected to continue for the foreseeable future, a scalable Architecture for service differentiation is required to accommodate this continued growth. Hence DiffServ framework becomes the more appropriate choice for IP applications demanding guaranteed QoS.

DiffServ supplies a very customizable QoS, depending on the needs of the traffic type. In the diffserv framework, packets carry their own state in a few bits of the IP header (the DS Codepoint), which also leads to scalability of this QoS mechanism, making it appropriate for end-to-end QoS. Diffserv provide service differentiation by creating service classes with different priorities using either type-of-service (TOS) field of Ipv4 or the priority bits of IPv6 headers. This priority scheme translates into higher throughput for higher priority classes. The goal for differentiated service is to define scalable service discrimination policy without maintaining the state of each flow and signaling at every hop.

DiffServ allows different classes of service to be provided for traffic streams on a common network infrastructure. DiffServ aggregates multitude of QoS enabled flows into a small number of aggregates. These aggregated flows are given differentiated treatment within the network. The diffserv approach attempts to push per-flow

complexity away from the network core and towards the edge of the network where both the forwarding speeds and fan-in of flows are smaller.

Recent Linux kernels offers a wide variety of traffic control functions that supports IP QoS. The current work aims at the implementation of a DiffServ framework in the Linux platform (Kernel Version 7.1) using IPv4 protocol.

4.DETAILS OF PROPOSED METHODOLOGY

Differentiated Service architecture is based on a simple model where traffic entering a network is classified and possibly conditioned at the boundaries of the network, and assigned to different behavior aggregates. Each behavior aggregate is identified by a single DS codepoint. Within the core of the network packets are forwarded according to the per-hop behavior associated with the DS codepoint.

Diffserv provides scalable service discrimination in the Internet without the need for per-flow state and signaling at every hop. Service can be constructed by the combination of

- Setting bits in an IP header field at network boundaries.
- Using those bits to determine how packets are forwarded by the nodes inside the network, and
- Conditioning the marked packets at network boundaries in accordance with the requirements of service.

4.1 DIFFSERV ARCHITECTURE MODEL

A node, which is enabled to support differentiated services functions and behaviors is called a DS node. Differentiated Service Domain (DS-domain) is a contiguous set of DS nodes which operate with a common service provisioning policy and set of PHB groups implemented on each node. A DS domain normally consists of one or more networks under the same administration e.g. organizations intranet. DS domain consists of DS boundary nodes and DS interior nodes. DS boundary nodes interconnect the DS domain to other DS or non-DS-capable domains, While DS interior nodes only connect to other DS interior or boundary nodes within the same DS domain. DS boundary nodes act both as a DS ingress node and as a DS egress node for different directions of traffic. Traffic enters a DS-Domain at a DS ingress node and leaves a DS

Diffserv assumes the existence of a service level agreement (SLA) between networks that share a border. The SLA establishes the policy criteria the traffic profile. It is expected that traffic will be policed and smoothed at egress points according to the SLA, and any traffic "out of profile" (i.e. Above the upper-bounds of bandwidth usage stated in the SLA) at an ingress point have no guarantees. The policy criteria used can include source and destination address, transport protocol, and/or port numbers. Basically, any context or traffic content (including headers or data) can be used to apply policy

4.2 DIFFERENTIATED SERVICE FIELD DEFINITION

TOS octet of IPV4 and traffic class octet of IPV6 are completely redefined by diffServ and is now called the Differentiated Service (DS) field.

Six bits of the DS field are used as a differentiated service codepoint (DSCP) to select the per-hop behavior (PHB) a packet experiences at each node. The last two bits are currently unused (CU) and it is reserved. The DS field structure is

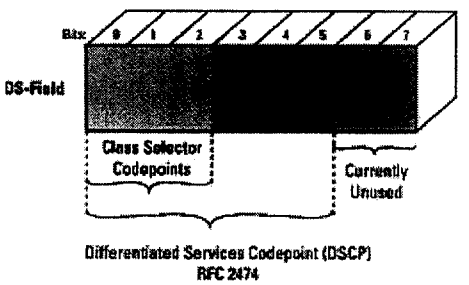

Figure 4.1 : DS field

DSCP - Differentiated service codepoint.

CU - Currently Unused.

The 6-bit DSCP field with in the DS field is capable of conveying 64 distinct codepoints. The codepoint space is divided into three pools.

POOL	CODEPOINT SPACE	NO.OF POSSIBLE CODEPOINT	ASIGNMENT POLICY
1	XXXXX0	32	Standards Action
2	XXXX11	16	Experimental/ Local Use
3	XXXX01	16	*Experimental /Local use

Table 4.1 : DiffServ codepoint

- - May be utilized for future standards Action allocation as necessary.
- Where X – refers to either '0' or '1'.

4.3 MAJOR COMPONENTS OF DIFFSERV MODEL

Differentiated Service architecture is based on a simple model where traffic entering a network is classified and possibly conditioned at the boundaries of the network, and assigned to different behavior aggregates. Each behavior aggregate is identified by a single DS codepoint. Within the core of the network packets are forwarded according to the per-hop behavior associated with the DS codepoint.

DiffServ architecture has two major components.

1. Traffic classification and conditioning.
2. Forwarding / Per-Hop Behaviors (PHB S)

4.3.1 TRAFFIC CLASSIFICATION AND CONDITIONING

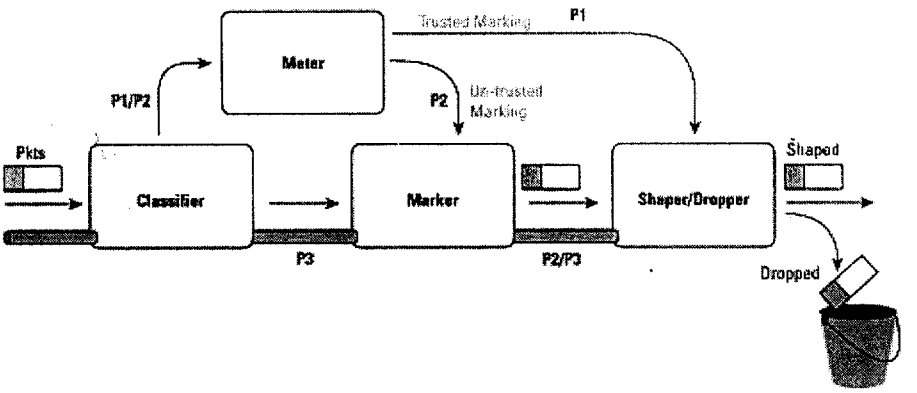

Figure 4.2 : Traffic classification and conditioning

PACKET CLASSIFIER

Classifiers are used to steer packets matching some specified rule to an element of traffic controller for further processing. Packets classifiers select packets in a traffic stream based on the content of some portion of the packet header.

There are two types of classifier.

1. The Behavior Aggregate (BA) classifier

   Classifies packet based on the DS codepoint only.

2. The Multi-Field (MF) Classifier

   Selects packet based on the value of a combination of one or more header fields such as source address, destination address, DS field, Protocol ID, Source port number and Destination port number.

TRAFFIC CONDITIONER

A Traffic profile specifies the temporal properties of a traffic stream selected by a classifier. It provides rules for determining whether a particular packet is in-profile or

out-of-profile. Different conditioning actions may be applied to the in-profile packets and out-of-profile packets.

Traffic conditioner may contain the following elements meter, marker, shaper and dropper.

METER

A meter is used (where appropriate) to measure the traffic stream against a traffic profile. The state of the meter with respect to a particular packet (E.g. Whether it is in-profile or out-of-profile) may be used to affect a marking dropping or shaping action.

PACKET MARKING

Packet markers set the DS field of a packet to a particular codepoint, adding the marked packet to a particular DS behavior aggregate.

SHAPER

Shaper delay some or all of the packets in a traffic stream in order to bring the stream into compliance with a traffic profile.

DROPPER

Droppers discard some or all of the packets in traffic stream in order to bring the stream into compliance with a traffic profile.

4.3.2 PER- HOP BEHAVIOR (PHB)

RFC 2475 defines PHB as the externally observable forwarding behavior applied at a DiffServ-compliant node to a DiffServ Behavior Aggregate (BA).

PHB is the means by which a node allocates resources to behavior aggregates. With the ability of the system to mark packets according to DSCP setting, collections of packets with the same DSCP setting and sent in a particular direction can be grouped into a BA. Packets from multiple sources or applications can belong to the same BA. In other words, a PHB refers to the packet scheduling, queuing, policing, or shaping behavior of a

The following sections describe the four available standard PHBs:

- Default PHB (as defined in RFC 2474)
- Class-Selector PHB (as defined in RFC 2474)
- Assured Forwarding (AFny) PHB (as defined in RFC 2597)
- Expedited Forwarding (EF) PHB (as defined in RFC 2598)

Default PHB

The default PHB essentially specifies that a packet marked with a DSCP value of 000000 (recommended) receives the traditional best-effort service from a DS-compliant node (that is, a network node that complies with all of the core DiffServ requirements). Also, if a packet arrives at a DS-compliant node, and the DSCP value is not mapped to any other PHB, the packet will get mapped to the default PHB.

Class-Selector PHB

To preserve backward-compatibility with any IP precedence scheme currently in use on the network, DiffServ has defined a DSCP value in the form xxx000, where x is either 0 or 1. These DSCP values are called Class-Selector Codepoints. (The DSCP value for a packet with default PHB 000000 is also called the Class-Selector Codepoint.)

The PHB associated with a Class-Selector Codepoint is a Class-Selector PHB. These Class-Selector PHBs retain most of the forwarding behavior as nodes that implement IP Precedence-based classification and forwarding.

For example, packets with a DSCP value of 11000 (the equivalent of the IP Precedence-based value of 110) have preferential forwarding treatment (for scheduling, queuing, and so on), as compared to packets with a DSCP value of 100000 (the equivalent of the IP Precedence-based value of 100). These Class-Selector PHBs ensure that DS-compliant nodes can coexist with IP Precedence-based nodes.

Assured Forwarding PHB

Assured Forwarding (AF) PHB is nearly equivalent to Controlled Load Service available in the integrated services model. AFny PHB defines a method by which BAs can be given different forwarding assurances.

For example, network traffic can be divided into the following classes:

- Gold: Traffic in this category is allocated 50 percent of the available bandwidth.
- Silver: Traffic in this category is allocated 30 percent of the available bandwidth.
- Bronze: Traffic in this category is allocated 20 percent of the available bandwidth.

Further, the AFny PHB defines four AF classes: AF1, AF2, AF3, and AF4. Each class is assigned a specific amount of buffer space and interface bandwidth, according to the SLA with the service provider or policy map.

Within each AF class, you can specify three drop precedence (dP) values: 1, 2, and 3.

Assured Forwarding PHB can be expressed as follows:

AFny.

In this example, *n* represents the AF class number (1, 2, or 3) and y represents the dP value (1, 2, or 3) within the AFn class.

In instances of network traffic congestion, if packets in a particular AF class (for example, AF1) need to be dropped, packets in the AF1 class will be dropped according to the following guideline:

dP(AFny) <= dP(AFnz) <= dP(AFnx)

where dP (AFny) is the probability that packets of the AFny class will be dropped. In other words, *y* denotes the dP within an AFn class.

In the following example, packets in the AF13 class will be dropped before packets in the

dP(AF13) <= dP (AF12) <= dP(AF11)

The dP method penalizes traffic flows within a particular BA that exceed the assigned bandwidth. Packets on these offending flows could be re-marked by a policer to a higher drop precedence.

An AFx class can be denoted by the DSCP value, xyzab0, where *xyz* can be 001, 010, 011, or 100, and *ab* represents the dP value.

DSCP Values and Corresponding Drop Precedence Values for Each AF PHB Class Drop Precedence	**Class 1**	**Class 2**	**Class 3**	**Class 4**
Low drop precedence	001010	010010	011010	100010
Medium drop precedence	011000	010100	011100	100100
High drop precedence	001110	0101110	011110	100110

Table 4.2 : Lists the DSCP value and corresponding dP value for each AF PHB class.

Expedited Forwarding PHB

Resource Reservation Protocol (RSVP), a component of the integrated services model, provides a Guaranteed Bandwidth Service. Applications such as Voice over IP (VoIP), video, and online trading programs require this kind of robust service. The EF PHB, a key ingredient of DiffServ, supplies this kind of robust service by providing low loss, low latency, low jitter, and assured bandwidth service.

EF can be implemented using PQ, along with rate-limiting on the class (or BA). When implemented in a DiffServ network, EF PHB provides a virtual leased line, or premium service. For optimal efficiency, however, EF PHB should be reserved for only the most critical applications because, in instances of traffic congestion, it is not feasible to treat all

EF PHB is ideally suited for applications such as VoIP that require low bandwidth, guaranteed bandwidth, low delay, and low jitter.

The recommended DSCP value for EF PHB is 101110.

4.4 TRAFFIC CONTROL IN LINUX

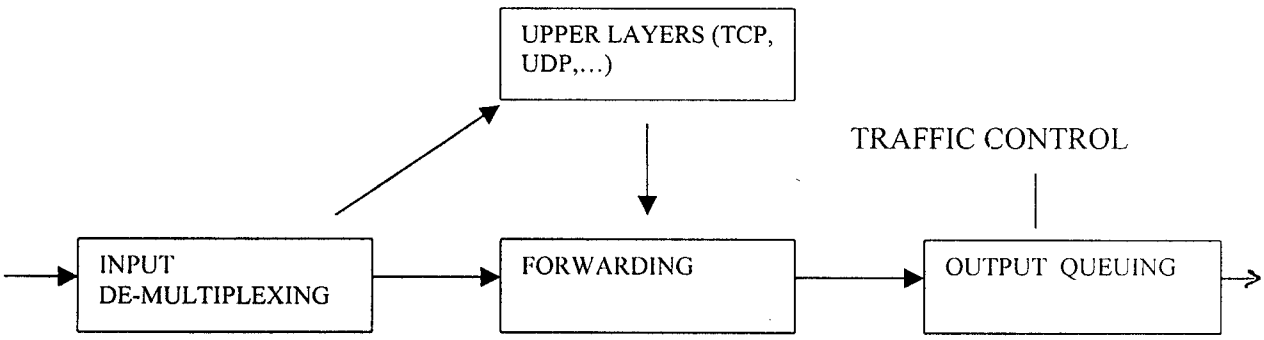

Figure 4.4 : Processing of network data

The input demultiplexer examines the incoming packets to determine if the packets are destined for the local node. If so, they are sent to the higher layers for further processing. If not it sends the packet to the forwarding block. The forwarding block which may also received locally generated packets from the higher layer, looks up the routing table and determines the next hop for the packet. After this it queues the packet to be transmitted on the output interface. It is at this point that the Linux traffic control comes into play. Linux traffic control can be used to build a complex combination of queuing disciplines, classes and filters hat control the packets that are sent on the output interface. Linux traffic control can

1.Decide if packets are queued or if they are dropped.

2.It can decide in which order packets are sent (to give priority to certain flows)

3.It can delay the sending of packets.

Traffic control is implemented just before the packet is sent to the device driver. Once the traffic control has released a packet for sending, the device driver picks it up and emits it on the network.

Traffic Control Components

QoS support in Linux kernel consists of the following 3 major basic building blocks.

1.Queueing disciplines.

2.Classes (within a queuing discipline)

3.Filters / Policers.

Each network device has a queuing discipline associated with it, which controls how packets enqueued on that device are treated.

Queuing discipline may be single queue without classes, where all packets are stored in the order in which they have been enqueued, and which is emptied as fast as the respective device can send.

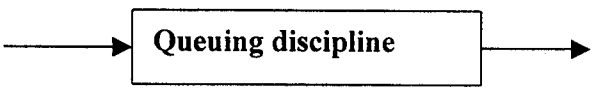

Figure 4.5 : A simple queuing discipline without classes

More elaborate queuing disciplines may use filters to distinguish among different classes of packets and process each class in a specific way, e.g.

By giving one class priority over other classes. Queuing disciplines and classes are intimately tied together. Classes normally don't take care of storing their packets themselves, but they use another queuing discipline to take care of that. Each class owns one queue, which is by default FIFO.

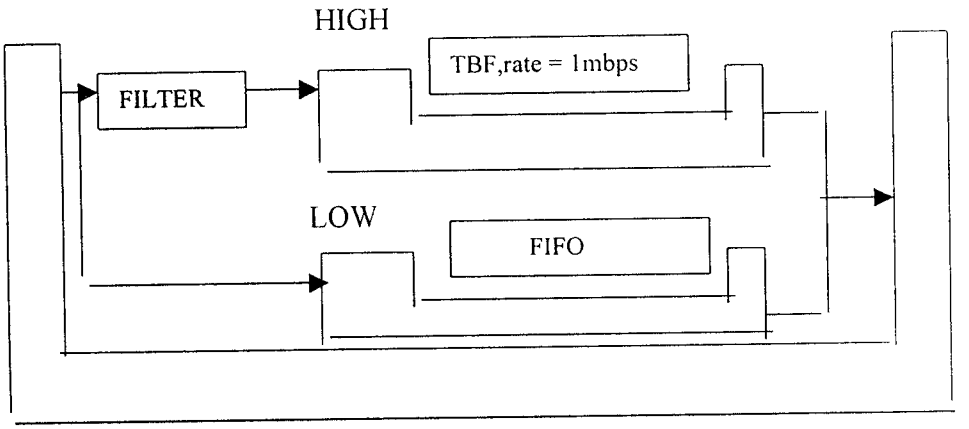

Figure 4.6: Queuing Discipline with two delay priorities

Figure shows a queuing discipline with two delay priorities. Packets, which are selected by the filter, go to the high priority class, while all other packets go to the low priority class. Whenever there are packets in the high priority queue, they are sent before packets in the low priority queue. In order to prevent high priority traffic from starving low priority traffic we use a token bucket filter (TBF). Finally, the queuing of low priority packets is done by a FIFO queuing.

Packets are enqueued as follows:

When the enqueue function of a queuing discipline is called, it runs one filter after the other until one of them indicates a match. It then queues a packet for the corresponding class, which usually means to invoke enqueue function of the queuing discipline owned by that class. Packets, which do not match any of the filters, are typically attributed to some default class.

Each class "owns" one queue, but it is in principle also possible that several classes share the same queue or even that a single queue is used by all classes of the respective queuing discipline. Usually when enqueuing packets, the corresponding flow(s) can be policed, e.g. by discarding packets, which exceed a certain rate.

4.5 IMPLEMENTATION

Implementing DiffServ architecture consists of two main modules. The first module is Edge architecture which takes care of classification and conditioning of incoming packets. Second module consists of core architecture, which takes care of forwarding the packets according to assigned DSCP value.

4.5.1 EDGE ARCHITECTURE

Flows from a customer end point enter a DS domain through the Edge Router. So, the architecture at the edge router has provision for setting the DSCP and also for policing the packets. The architecture is as shown in the figure.

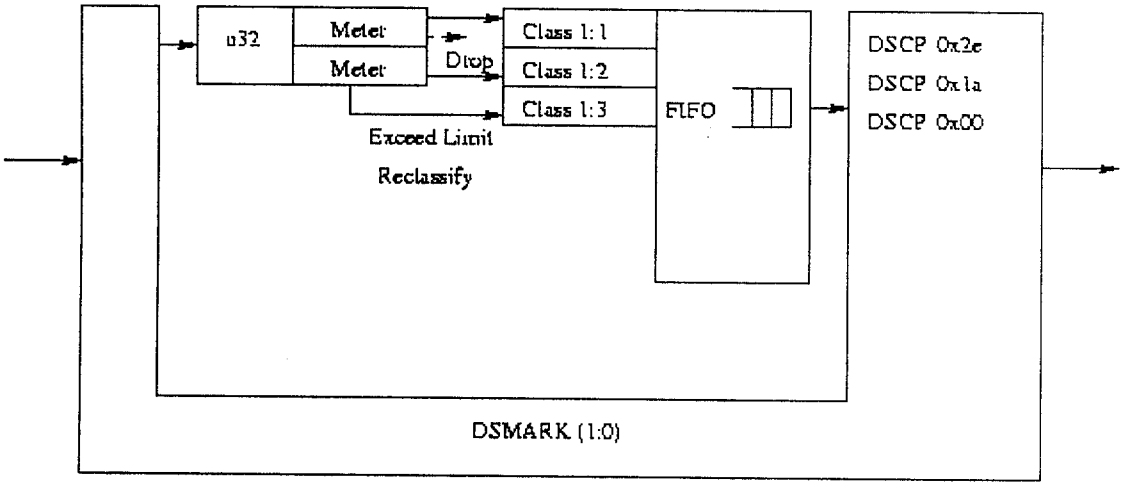

Figure 4.7: Edge Architecture

The first level queue in the architecture is the ds_mark queuing discipline, which will be

directly associated with the output device. The ds_mark queuing discipline sets the DSCP to the required codepoint.

From the ds_mark queuing discipline an u32 filter is created. The u32 filters are created for a particular flow. Each flow can be uniquely identified by the four tuple <src ip, dst ip, src port, dst port> or the three tuple <src ip, dst ip, dst port>. Each filter is

associated with a meter, which polices the flow for in profile and out of profile packets. An in profile EF flow's DSCP is marked with 0x2e and the out of profile packets are dropped. In AF the in profile packets are marked with the assigned codepoint and the out of profile packets are reclassified as best effort.

The packets emerging out of the Edge Router have their DSCPs marked and are also policed according to the bandwidth allocated to the customer by the Bandwidth Broker

4.5.2 CORE ARCHITECTURE

The Two Bit DiffServ Architecture at the core routers allows for the coexistence of AF and EF. The figure 4.8 shows the architecture.

The first level queue in the architecture is the ds_mark queuing discipline, which will be directly associated with the output device. This queuing discipline will extract the TOS byte from the incoming packet and pass it on to the tc_index classifier. The first level tc_index classifier extracts the DSCP from the TOS byte by masking with 0xfc and then right shifting by 2. For example, if the TOS is 0xb8,

TOS	0xb8	10111000
Mask	0xfc	10111000
Shift	2	00101110 (0x2e, the codepoint for EF)

It then stores the DSCP in skb->tc_index and feeds it onto the next level of tc_index filters. These filters are used to identify the AF class and drop precedence to which the packet belongs, if at all it belongs to the AF class. If it does, then it creates a new class based on the AF codepoint and stores the result of the classification in skb->tc_index. The value stored in the AF codepoint will be based on the AF class and the drop precedence within the AF class to which

the packet belongs. The TOS byte, the corresponding DSCP and the class ids for the different AF classes are summarized below.

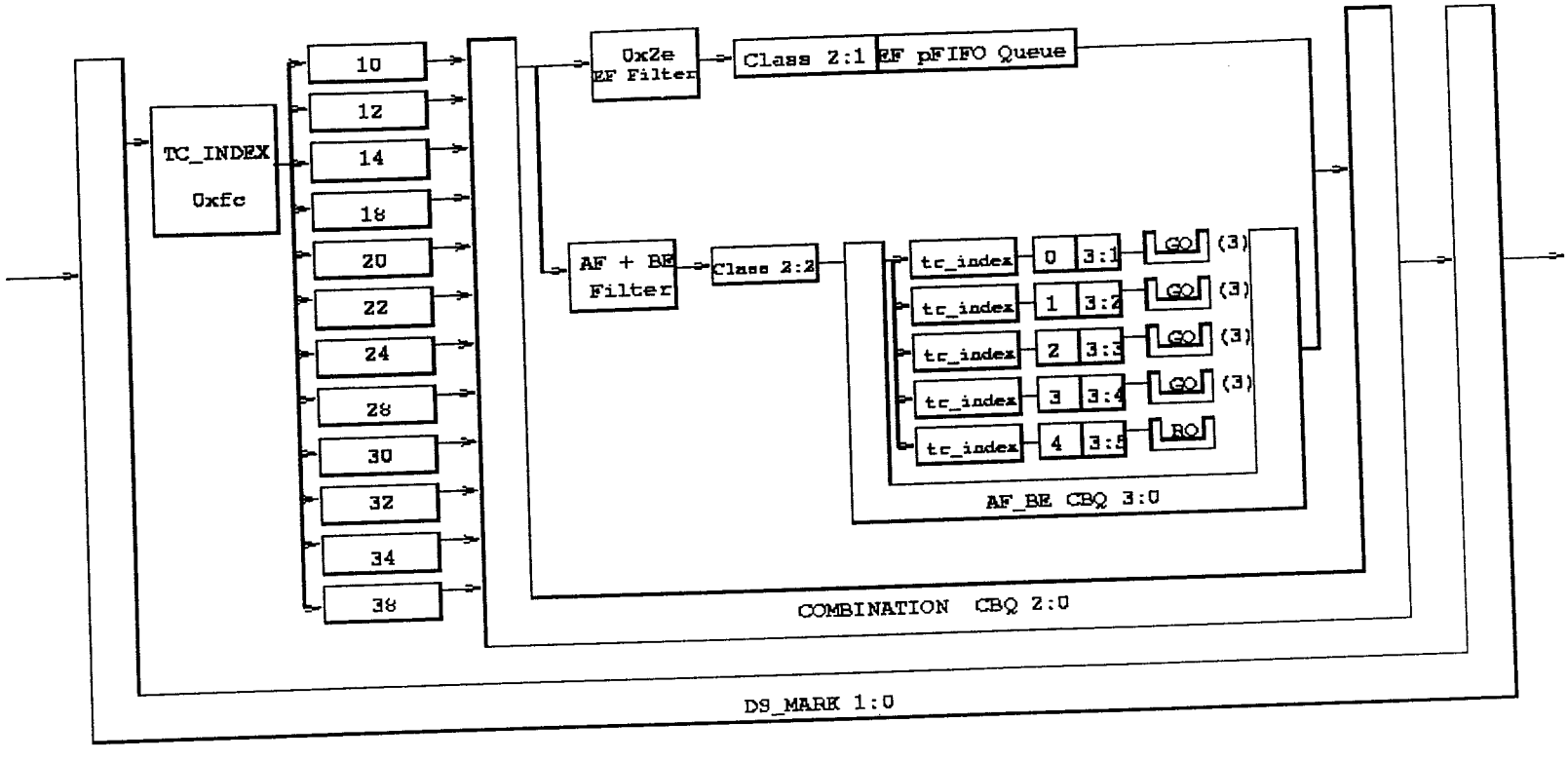

Figure 4.8 : Core Architecture

TOS (decimal)	Hex	Binary	DSCP (decimal)	Hex	Binary	Classed	AFClass
40	0x28	00101000	10	0x0a	00001010	1:111	AF 11
48	0x30	00110000	12	0x0c	00001100	1:112	AF 12
56	0x38	00111000	14	0x0e	00001110	1:113	AF 13
72	0x48	01001000	18	0x12	00010010	1:121	AF 21
80	0x50	01010000	20	0x14	00010100	1:122	AF 22
88	0x58	01011000	22	0x16	00010110	1:123	AF 23
104	0x68	01101000	26	0x1a	00011010	1:131	AF 31
112	0x70	01110000	28	0x1c	00011100	1:132	AF 32
120	0x78	01111000	30	0x1e	00011110	1:133	AF 33
136	0x88	10001000	34	0x22	00100010	1:141	AF 41
144	0x90	10010000	36	0x24	00100100	1:142	AF 42
152	0x98	10011000	38	0x26	00100110	1:143	AF 43

Table 4.3 : The TOS byte, the corresponding DSCP and the class ids for the different AF classes are summarized below

Once this classification is done, the packet is then handed over to a Class Based Queue (CBQ) *2:0*.This CBQ will be used to differentiate between AF, EF and best effort. When the packets enter this CBQ, the EF packets are directed to class *2:1* by the *0x2e* filter and all other packets belonging to the different AF classes and BE are sent to class *2:2* by the handle *0* filter. For EF, a pfifo is associated with the class *2:1*. The EF class is configured to be *bounded* and *isolated* to ensure that it receives the bandwidth allocated for it. Now another CBQ *3:0* is derived from class *2:2* for AF and BE. In turn for each of the AF

with a GRED to allow for multiple drop precedence (there is a virtual queue for every drop precedence within an AF class) and the BE class has a RED queuing discipline. In this *3:0* CBQ, there is the *0xf0* filter which extracts the class number from the skb->tc_index. Then there are tc_index filters for each of the classes with handles *1,2,3,4 and 5* respectively. The AF packets are then queued in the appropriate Generalized Random Early Detection (GRED) queuing discipline. The GRED is more generalized form of the RED. In times of congestion, when it has to drop packets, it will drop them with the drop probability that is associated with the queue. The GRED then queues up the packet in the CBQ, which in turn sends the packet to the ds_mark queuing discipline for the packet to be transmitted.

5.RESULTS OBTAINED

This system will help the network administrators to configure differentiated services on a linux router. It also allows the simultaneous configuration of AF and EF on a particular device on the router. This configuration is based on both EF_BE and two-bit diffserv architecture.

Core_router is a daemon program, which has to run at the core router listening for client requests to configure queues, classes and filters. The clients (used in place of bandwidth broker) can connect to it and configure either the two-bit architecture or the ef-be architecture.

The client can be started as follows.

./ core_client <configuration file>

The parameters for the configuration are listed in the config file, the file has a format which is given below.

For the two-bit architecture

<Router name> <device name> <device bandwidth> <Architecture>

<EF PHB> <EF bandwidth>

<AF PHB> <AF bandwidth> < Number of classes>

<AF class No.> <Priority> <bandwidth> < No.of drops> <dp1> <dp2> <dp3>

<BE> <Priority> <bandwidth> <drop probability>

Example

```
127.0.0.1 eth0  10Mbit two_bit
EF 2Mbit
AF 8Mbit 4
AF1 5 1Mbit 2 0.02 0.04
AF2 4 1Mbit 3 0.02 0.04 0.06
```

```
AF3 3 4Mbit 3 0.02 0.04 0.06
AF4 2 1Mbit 3 0.02 0.04 0.06
BE 7 500Kbit 0.4
```

For the ef-be architecture

<Router name> <device name> <device bandwidth> <Architecture>

<EF PHB> <EF bandwidth>

<BE> <Priority> <bandwidth> <drop probability>

Example

```
127.0.0.1 eth0  10Mbit two_bit
EF 2Mbit
BE 7 500Kbit 0.4
```

The edge router, which also waits for client request, is started up as follows.

./edge_router

This router mark the packets with the corresponding DSCP value and also police the packets according to the traffic profile.

This architecture is tested for configuration queues and classes using perl script and partial results are attached.

```
filter parent 1: protocol ip pref 4 u32
filter parent 1: protocol ip pref 5 u32

qdisc dsmark 1: indices 0x0040
class dsmark 1:1 parent 1: mask 0x03 value 0x00
class dsmark 1:2 parent 1: mask 0x03 value 0x28
```

CORE ARCHITECTURE

```
*********class**************
lass cbq 3: root rate 8Mbit (bounded,isolated) prio no-transmit

lass cbq 3:1 parent 3: leaf 8002: rate 1Mbit prio 5

lass cbq 2: root rate 10Mbit (bounded,isolated) prio no-transmit

lass cbq 2:1 parent 2: leaf 8001: rate 2Mbit (bounded,isolated) prio 1

lass cbq 2:2 parent 2: leaf 3: rate 8Mbit (bounded,isolated) prio 2

********filter**************

ilter parent 1: protocol ip pref 1 tcindex hash 0 mask 0x00fc shift 2
all_through
```

6.CONCLUSIONS AND FUTURE WORK

CONCLUSION

The Internet Protocol (IP) has enabled a global network between an endless variety of systems and transmission media. Around the world of life, email exchange and web browsing are part of everyday life for work, study and play. And all by indications other networks like phone radio, and television are also converging on IP to leverage its ubiquity and flexibility.

These different applications have different operational requirements that demand different network services. Increased network traffic requires increased network bandwidth capacity. So increasing the bandwidth is an essential first step towards solving the latency issue. But new two-way real time applications have other requirements. Increasing the network pipe-size is not the only answer.

There is a clear need for relatively simple and coarse method of providing different classes of service for Internet traffic, to support various types of applications, and specific business requirements.

This work transforms IP network from a best-effort framework to a rich diffserv framework within a domain to help the administrator to provision their network according to changing needs of the application.

DiffServ, as great as it is, in enabling scalable and coarse grained QoS throughout the network, has a drawback that it needs to be provisioned. Setting up the various classes throughout the network requires knowledge of the application and traffic statistics for aggregate of traffic on the network. This process of application discovery and profiling can be time consuming process.

FUTURE WORK

1.DiffServ architecture is implemented within a single DS domain. It could be extended across several DS-domains with Service Level Agreement (SLA) profile agreed upon among the administrators of these DS domains.

2.Also the current architecture can be extended to support a combined architecture incorporating diffServ's traffic aggregation and RSVP's reservation policy. This allows reservation through a DS-Domain and mapping of the reservation to a DSCP and PHB. This aggregated reservation overcomes the problem of maintaining thousands of RSVP soft states in each router's memory.

7.REFERENCES

1 S.Blake,D.Blake,M.Carlson,E.Davies,Z.wang,W.weiss,"An intoduction for Differentiated services",RFC 2474 December 1998.

2. K.Nichols,S.Blake,F.Backer,B.Blake, "Definition of the Differentiated service Field(DS Field) in the IPV4 and IPV6 Headers" RFC 2474 ,December 1998.

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

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

5. E.Rosen,A.Viswanathan,R.Callon, "Multiprotocol Label Switching Architecture", April 1999.

6.Almesberger,Werner,"Linux Traffic control Implementation overview".November 30,1998.

7.Saravanan Radhakrishnan,"Linux-Advanced Networking Overview Version 1",Aug 22,1999.

8.<draft-Almesberger-wajhak-diffserv-linux-00.txt> "Differentiated services on Linux" Feb 1999 Werner Almesberge.

Journals

1.Saleem N.Bhatti and Jon Crowcroft,University College London, IEEE Internet Computing, 48-57 , July-August 2000.

2.Chris Metz ,Cisco systems, IEEE Internet Computing, 95-99, May-June 1999.

Websites

1. www.ietf.org
2. www.qosforum.com
3. www.Linux.org
4. www.kernel.org.

Books

1.Begginning Linux programming, Wrox press Ltd, 1999 Richard Stones, Neil Matthew.

2.Red Hat Linux System administration - Unleashed , Techmedia 2000, Thomas schenk et al.

8.APPENDICES

DiffServ Glossary

AF	Assured Forwarding
ATM	Asynchronous Transfer mode
BA	Behavior Aggregate
BE	Best Effort
CBQ	Class Based Queue
DiffServ	Differentiated Service
DS	Differentiated Service Field
DSCP	Differentiated Service codepoint
DTR	Data Terminal Ready
EF	Expedited Forwarding
FIFO	First in First out
FTP	File Transfer Protocol
GRED	Generalized Random Early Detection
IETF	Internet Engineering Task Force
IntServ	Integrated Service
IP	Internet Protocol
LAN	Local Area Network
MPLS	Multi Protocol Label Switching
PHB	Per-Hop-Behavior
QoS	Quality of service

RED	Random Early Detection
RFC	Request for comments
RSVP	Resource Reservation Protocol
SLA	Service Level Agreement
TCP	Transmission Control Protocol
TOP	Type of Service
UDP	User Datagram Protocol
VoIP	Voice over Internet protocol

```
/* File Name          : core.h
 * Owner              : V.Vanitha
 * Creation date      : 29.9.2001
 * Description        : The Diffserv daemon - Header file
 *                      This file defines a few constants used in the configuration
 *                      the classes and queues,
 */

#ifndef _DIFFSPECD_H
#define _DIFFSPECD_H


#define DSPEC_TC "/usr/src/iproute2/tc/tc"
#define DSPEC_DSMARK_INDICES 64

#define DSPEC_AVG_PKT "1000"
#define DSPEC_MPU "64"
#define DSPEC_CELL 8
#define DSPEC_ALLOT "1514"
#define DSPEC_WEIGHT "1"

#define DSPEC_GRED_MIN_THRESHOLD "15KB"
#define DSPEC_GRED_MAX_THRESHOLD "45KB"
#define DSPEC_GRED_LIMIT "60KB"

#endif
```

```
/*  File Name      :core_client.c
 *  Owner          :V.Vanitha
 *  Creation Date  :24.09.2001
 *  Description    :Client which connects to DiffSpec Daemon
 *                  This code will read the configuration
 *                  parameters(Dev_name,Dev_bandwidth,
 *                  Architecture(two_bit,EF_BE)) ,from the input file and
 *                  send the appropriate parameters to core_router.
 *
 */

#define SERVER_PORT 4245
#define MAXLINE 50

#include <stdio.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include "user_interface.h"


int set_two_bit_ds(char* config_file)
{
    int sock;
    struct sockaddr_in serverAddr;
    struct hostent *serverName;
    struct in_addr *serverAddress;
    packet_info pkt;
    int hostAddr,count;

    char router_addr[16]="";
    char device_name[10]="";
    char device_bndw[10]="";
    char arch[10]="";

    char phb_cmd[3]="";
    char phb_bndw[10]="";
    int number_of_classes;

    char af_class[5]="";

    FILE *fp;

    char file_buffer[50];

    fp = fopen(config_file,"r");

    memset(file_buffer, 0, 50);
    fgets(file_buffer,MAXLINE,fp);
```

```
if (feof(fp))
{
  printf("Error in config file format\n");
  exit(0);
}

/*
 * Reading the device information from the file
 */
sscanf(file_buffer, "%s %s %s %s", router_addr, device_name,
      device_bndw, arch);

if((hostAddr = inet_addr(router_addr)) == -1)
{
  printf("No Host Entry found:\n");
  perror("Failed to locate server, reason");
  exit(0);
}


memset ((void *)&serverAddr, 0, sizeof(serverAddr));

/* creating the socket */

if((sock = socket(AF_INET, SOCK_STREAM, 0))<0)
{
  perror("Creating socket failed, reason:");
  exit(0);
}

serverAddr.sin_family = AF_INET;
serverAddr.sin_addr.s_addr = hostAddr;
serverAddr.sin_port = htons(SERVER_PORT);

if (connect(sock, (struct sockaddr *)&serverAddr, sizeof(serverAddr))< 0)
{
  perror("Connecting to server failed, reason");
  close(sock);
  exit(0);
}

/* Writing the device information to the server */
memset((void *)&pkt, 0, sizeof(pkt));
pkt.cmd = DEVICE;
strcpy(pkt.data.device.device_name, device_name);
strcpy(pkt.data.device.device_bndw, device_bndw);
if(strcmp(arch, "two_bit") == 0)
{

  pkt.data.device.setup_flag = TWO_BIT;
```

```
if(write(sock, &pkt, sizeof(pkt))<0)
{
    perror("Writing to the socket failed, reason:");
    close(sock);
    exit(0);
}

memset(file_buffer, 0, 50);
fgets(file_buffer,MAXLINE,fp);
if (feof(fp))
{
    printf("Error in config file format, EF specs not given\n");
    exit(0);
}
sscanf(file_buffer, "%s %s", phb_cmd, phb_bndw);
if(strcmp(phb_cmd, "EF") == 0)
{
    memset((void *)&pkt, 0, sizeof(pkt));
    pkt.cmd = PHB;
    pkt.data.phb.cmd = EF;
    strcpy(pkt.data.phb.phb_params.phb_bndw, phb_bndw);

    /*
     * Writing the EF information in the socket
     */
    if(write(sock, &pkt, sizeof(pkt))<0)
    {
      perror("Writing to the socket failed, reason:");
      close(sock);
      exit(0);
    }
    printf("EF set\n");
}

memset(file_buffer, 0, 50);
fgets(file_buffer,MAXLINE,fp);
if (feof(fp))
{
    printf("Error in config file format, AF specs not given\n");
    exit(0);
}
sscanf(file_buffer, "%s %s %d", phb_cmd, phb_bndw, &number_of_classes);

if(strcmp(phb_cmd, "AF") == 0)
{
    memset((void *)&pkt, 0, sizeof(pkt));
    pkt.cmd = PHB;
    pkt.data.phb.cmd = AF;
    strcpy(pkt.data.phb.phb_params.phb_bndw, phb_bndw);

    /*
     * Writing the AF information in the socket
     */
```

```
    if(write(sock, &pkt, sizeof(pkt))<0)
    {
      perror("Writing to the socket failed, reason:");
      close(sock);
      exit(0);
    }
    printf("AF set\n");

}

printf("number of classes: %d\n", number_of_classes);
for(count=0; count < number_of_classes; count++)
{
    memset((void *)&pkt, 0, sizeof(pkt));

    pkt.cmd = AF_CLASS;
    pkt.data.af_class.af_class_num = count+1;
    memset(file_buffer, 0, 50);
    fgets(file_buffer,MAXLINE,fp);
    if (feof(fp))
    {
      printf("Error in config file format, AF class %d specs
            not givenproperly \n", count+1);
      exit(0);
    }
    sscanf(file_buffer, "%s %d %s %d %lf %lf %lf",
          af_class, &pkt.data.af_class.priority,
          pkt.data.af_class.class_bndw,
          &pkt.data.af_class.num_of_dps,
          &pkt.data.af_class.drop_prob[0],
          &pkt.data.af_class.drop_prob[1],
          &pkt.data.af_class.drop_prob[2]);


    if(write(sock, &pkt, sizeof(pkt))<0)
    {
      perror("Writing to the socket failed, reason:");
      close(sock);
      exit(0);
    }
    printf("Class %d set \n", count+1);
}

memset(file_buffer, 0, 50);
fgets(file_buffer,MAXLINE,fp);
if (feof(fp))
{
    printf("Error in config file format, BE specs not given\n");
    exit(0);
}
sscanf(file_buffer, "%s",phb_cmd);
```

```
 if(strcmp(phb_cmd, "BE") == 0)
 {
     memset((void *)&pkt, 0, sizeof(pkt));
     pkt.cmd = PHB;
     pkt.data.phb.cmd = BE;
     sscanf(file_buffer, "%s %d %s %lf", phb_cmd,
           &pkt.data.phb.phb_params.be_class.priority,
           pkt.data.phb.phb_params.be_class.class_bndw,
           &pkt.data.phb.phb_params.be_class.drop_prob);

     /*
      * Writing the BE info into the socket
      */
     if(write(sock, &pkt, sizeof(pkt))<0)
     {
       perror("Writing to the socket failed, reason:");
       close(sock);
       exit(0);
     }
     printf("BE set\n");
  }

}
else
{
  if(strcmp(arch, "ef_be") == 0)
  {
      pkt.data.device.setup_flag = EF_BE;

      if(write(sock, &pkt, sizeof(pkt))<0)
      {
        perror("Writing to the socket failed, reason:");
        close(sock);
        exit(0);
      }

      memset(file_buffer, 0, 50);
      fgets(file_buffer,MAXLINE,fp);
      if (feof(fp))
      {
        printf("Error in config file format, EF specs not given\n");
        exit(0);
      }
      sscanf(file_buffer, "%s %s", phb_cmd, phb_bndw);
      if(strcmp(phb_cmd, "EF") == 0)
      {
        memset((void *)&pkt, 0, sizeof(pkt));
        pkt.cmd = PHB;
        pkt.data.phb.cmd = EF;
        strcpy(pkt.data.phb.phb_params.phb_bndw, phb_bndw);

        if(write(sock, &pkt, sizeof(pkt))<0)
        {
```

```
            perror("Writing to the socket failed, reason:");
            close(sock);
            exit(0);
          }
          printf("EF set\n");
        }

        memset(file_buffer, 0, 50);
        fgets(file_buffer,MAXLINE,fp);
        if (feof(fp))
        {
          printf("Error in config file format, BE specs not given\n");
          exit(0);
        }
        sscanf(file_buffer, "%s",phb_cmd);

        if(strcmp(phb_cmd, "BE") == 0)
        {
          memset((void *)&pkt, 0, sizeof(pkt));
          pkt.cmd = PHB;
          pkt.data.phb.cmd = BE;
          sscanf(file_buffer, "%s %d %s %lf", phb_cmd,
                &pkt.data.phb.phb_params.be_class.priority,
                pkt.data.phb.phb_params.be_class.class_bndw,
                &pkt.data.phb.phb_params.be_class.drop_prob);

          if(write(sock, &pkt, sizeof(pkt))<0)
          {
              perror("Writing to the socket failed, reason:");
              close(sock);
              exit(0);
          }
          printf("BE set\n");
        }
    }
    else
        printf("Invalid Architecture \n");

}

pkt.cmd = EOF_SETUP;

if(write(sock, &pkt, sizeof(pkt))<0)
{
  perror("Writing to the socket failed, reason:");
  close(sock);
  exit(0);
}
```

```
    if(write(sock, &pkt, sizeof(pkt))<0)
    {
      perror("Writing to the socket failed, reason:");
      close(sock);
      exit(0);
    }

    if(close(sock)<0)
    {
      perror("Close socket failed, reason");
      exit(0);
    }

    printf("Exiting... \n");

}

main(int argc, char *argv[])
{

    if (argc < 2)
    {
      printf(" Usage: core_client Router_Configuration_File\n");
      exit(0);
   }

    printf("Config file: %s\n", argv[1]);
    set_two_bit_ds(argv[1]);

}
```

```
/* File Name        :core_router.c
 * Owner            :V.Vanitha
 * Creation Date    :28.09.2001
 * Description      :The Diffspec daemon(To Implement PHB(AF,EF,BE) )
 *                   This daemon waits at a standard port and waits for clients
                     to connect and configure their interfaces.Configuration
                     means setting up of the queues and classes,for providing
                     differential services at the output interface.
 *
 */


#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <signal.h>


/* Own Includes */

#include "user_interface.h"
#include "tc_modifier.h"
#include "core.h"

#define DIFFSPECD_PORT 4245

global_info global_data;
qdisc qdisc_info;
filter_ip root_filter;
cbq_class class_info;
filter_ip filter_info;
char *cmd;

main()
{
    struct sockaddr_in sourceAddress, destAddress;
    int sock,newSocket;
    int size, bytes_read;
    packet_info packet;
    int end_of_setup_flag = 1;
    char buffer[1024];
    int fork_pid;

    memset(( void*)&sourceAddress, 0, sizeof(sourceAddress));
    memset(( void*)&destAddress, 0, sizeof(destAddress));

    size = sizeof(destAddress);
    sourceAddress.sin_family = AF_INET;
    sourceAddress.sin_addr.s_addr = INADDR_ANY;
    sourceAddress.sin_port = htons(DIFFSPECD_PORT);
```

```
if((sock = socket( AF_INET, SOCK_STREAM, 0))<0)
{
  perror("Error creating Socket:, Reason");
  exit(0);
}

if(bind(sock, (struct sockaddr *)&sourceAddress, sizeof sourceAddress)<0)
{
  perror("Error binding the socket to the port:, Reason");
  close(sock);
  exit(0);
}

if(listen(sock,5)<0)
{
  perror("Error listening on the socket:, Reason");
  close(sock);
  exit(0);
}

/*
 * The server is in an infinite while loop waiting for client requests,
 * once a request is received, a new process is forked off to handle the
 * request and the parent resumes accepting new connections
 */
while(1)
{
  /* Accepting the new connections */
  if((newSocket = accept(sock, (struct sockaddr *)&destAddress,&size))<0)
  {
      perror("Error accepting connections: Reason");
      close(sock);
      exit(0);
  }


  /*
   * Fork off a child to handle this client
   */
  fork_pid = fork();


  /*
   * In the child
   */
  if(fork_pid == 0)
  {

      /* Closing the old socket */
      close(sock);
```

```
while(end_of_setup_flag)
{

  if((bytes_read = read(newSocket, &packet,
                    sizeof(packet_info))) < 0)
  {
      perror("Reading from socket failed, Reason");
      close(newSocket);
      exit(0);
  }

  if(bytes_read > 0)
  {

      switch(packet.cmd)
      {
        case DEVICE:

            strcpy(global_data.device_name,
              packet.data.device.device_name);
            strcpy(global_data.device_bndw,
              packet.data.device.device_bndw);
            global_data.setup_flag =
              packet.data.device.setup_flag;

            tc_core_init();
            if (set_dsmark(packet))
              printf(" Dsmark Setup UnSuccessful \n");
            break;

        case PHB:
            if(packet.data.phb.cmd == AF)
              strcpy(global_data.af_phb_bndw,
                  packet.data.phb.phb_params.phb_bndw);

            if(set_phb(packet))
              printf(" PHB Setup UnSuccessful... \n");
            break;

        case AF_CLASS:

            if(set_af_class(packet))
              printf(" AF Class Setup UnSuccessful... \n");
            break;

        case EOF_SETUP:
            end_of_setup_flag = 0;
            close(newSocket);
            break;

      }
  }
}
```

```
    }

    /* In the parent, just close the new socket and continue to accept new
     * connections
     */
    else
    {
        signal(SIGCLD, SIG_IGN);
        close(newSocket);
          continue;
    }
  }
}

/* Setting up the dsmark queue & the initial filter to shift the TOS by
 * two bits and extract the code point, also in the two bit architecture
 * setting up the outermost CBQ for AF & EF PHBs
 */

int set_dsmark(packet_info packet)
{
/*
    qdisc qdisc_info;
    filter_ip root_filter;
*/
    qdisc_info.type = DSMARK;
    strcpy(qdisc_info.handle, "1:0");
    qdisc_info.q_info.ds_queue.indices = DSPEC_DSMARK_INDICES;
    qdisc_info.q_info.ds_queue.set_tc_index=1;
    strcpy(qdisc_info.q_info.ds_queue.parent, "NULL");
    if (add_queue(packet.data.device.device_name, 1, &qdisc_info) < 0)
      return -1;

    strcpy(root_filter.mask,"0xfc");
    strcpy(root_filter.shift,"2");
    strcpy(root_filter.parent,"1:0");
    strcpy(root_filter.handle,"NULL");
    strcpy(root_filter.classid,"NULL");
    root_filter.priority = 1;
    root_filter.pass_on = 0;
    if (add_filter(packet.data.device.device_name, &root_filter) < 0)
      return -1;
    memset(&qdisc_info, 0, sizeof(qdisc));
    qdisc_info.type = CBQ;
    strcpy(qdisc_info.handle, "2:0");
    strcpy(qdisc_info.q_info.cbq_queue.parent, "1:0");
    strcpy(qdisc_info.q_info.cbq_queue.bandwidth,
          packet.data.device.device_bndw);
    strcpy(qdisc_info.q_info.cbq_queue.avpkt,DSPEC_AVG_PKT);
    strcpy(qdisc_info.q_info.cbq_queue.mpu,DSPEC_MPU);
    qdisc_info.q_info.cbq_queue.cell = DSPEC_CELL;
```

```
   if (add_queue(packet.data.device.device_name, 0, &qdisc_info) < 0)
     return -1;
   return 0;

}

/* Setting up the appropriate classes and filters for AF, EF and BE */
int set_phb(packet_info packet)
{
/*
    cbq_class class_info;
    filter_ip filter_info;
    qdisc qdisc_info;
*/
    switch(packet.data.phb.cmd)
    {
      case AF:

          /* Creating the 2:2 class for AF and BE traffic and the filter to
           * direct the packets to this class. Also creating the 3:0 CBQ
           * (class based queuing discipline) for the 4 AF classes and BE
           * class. The filter to extract the AF class number is also done h
           * here
           */
          strcpy(class_info.parent, "2:0");
          strcpy(class_info.handle, "2:2");
          strcpy(class_info.bandwidth, global_data.device_bndw);
          strcpy(class_info.rate, packet.data.phb.phb_params.phb_bndw);
          strcpy(class_info.avpkt, DSPEC_AVG_PKT);
          strcpy(class_info.allot,DSPEC_ALLOT);
          strcpy(class_info.weight,DSPEC_WEIGHT);
          strcpy(class_info.maxburst,"10");
          class_info.bounded = 1;
          class_info.isolated = 1;
          class_info.priority = 2;
          if (add_cbq_class(global_data.device_name, 0, &class_info) < 0)
            return -1;

          filter_info.priority = 2;
          sprintf(filter_info.mask, "%d", 0);
          strcpy(filter_info.shift, "NULL");
          strcpy(filter_info.classid, "2:2");
          strcpy(filter_info.parent, "2:0");
          filter_info.pass_on = 0;
          sprintf(filter_info.handle, "%d", 0);
          if(add_filter(global_data.device_name, &filter_info) < 0)
            return -1;

          qdisc_info.type = CBQ;
          strcpy(qdisc_info.handle, "3:");
          strcpy(qdisc_info.q_info.cbq_queue.parent, "2:2");
```

```
    strcpy(qdisc_info.q_info.cbq_queue.bandwidth,
          packet.data.phb.phb_params.phb_bndw);
    strcpy(qdisc_info.q_info.cbq_queue.avpkt,DSPEC_AVG_PKT);
    strcpy(qdisc_info.q_info.cbq_queue.mpu,"64");
    qdisc_info.q_info.cbq_queue.cell = DSPEC_CELL;
    if (add_queue(global_data.device_name, 0, &qdisc_info) < 0)
      return -1;


    memset(&filter_info, 0, sizeof(filter_info));

    strcpy(filter_info.mask,"0xf0");
    strcpy(filter_info.shift,"4");
    strcpy(filter_info.parent,"3:0");
    strcpy(filter_info.handle,"NULL");
    strcpy(filter_info.classid,"NULL");
    filter_info.priority = 1;
    filter_info.pass_on = 1;
    if (add_filter(global_data.device_name, &filter_info) < 0)
      return -1;
    break;

case EF:

    /* For EF, the class 2:1 is created, a pfifo associated with it
     * and the filter to extract the code point is also created.
     */
    strcpy(class_info.parent, "2:0");
    strcpy(class_info.handle, "2:1");
    strcpy(class_info.bandwidth, global_data.device_bndw);
    strcpy(class_info.rate, packet.data.phb.phb_params.phb_bndw);
    strcpy(class_info.avpkt, DSPEC_AVG_PKT);
    strcpy(class_info.allot,DSPEC_ALLOT);
    strcpy(class_info.weight,DSPEC_WEIGHT);
    if (global_data.setup_flag == TWO_BIT)
      strcpy(class_info.maxburst,"10");
    else
      strcpy(class_info.maxburst,"300");
    class_info.bounded = 1;
    class_info.isolated = 1;
    class_info.priority = 1;
    if (add_cbq_class(global_data.device_name, 0, &class_info) < 0)
      return -1;

    qdisc_info.type = PFIFO;
    strcpy(qdisc_info.handle, "NULL");
    strcpy(qdisc_info.q_info.pfifo_queue.parent, "2:1");
    if (global_data.setup_flag == TWO_BIT)
      strcpy(qdisc_info.q_info.pfifo_queue.limit, "5");
    else
      strcpy(qdisc_info.q_info.pfifo_queue.limit, "30");

    if (add_queue(global_data.device_name, 0, &qdisc_info) < 0)
      return -1;
```

```
    memset(&filter_info, 0, sizeof(filter_info));
    filter_info.priority = 1;
    strcpy(filter_info.mask,"NULL");
    strcpy(filter_info.shift,"NULL");
    strcpy(filter_info.parent,"2:0");
    sprintf(filter_info.handle,"%d",46);
    strcpy(filter_info.classid, "2:1");
    filter_info.pass_on = 1;
    if (add_filter(global_data.device_name, &filter_info) < 0)
      return -1;
    break;

case BE:

    /* For BE, for the two bit architecture, the 3:5 class is created
     * a red queue is associated with it and a filter to direct the
     * packets to this class is created.
     */
    if (global_data.setup_flag == TWO_BIT)
    {
      strcpy(class_info.parent, "3:0");
      sprintf(class_info.handle, "3:5");
      strcpy(class_info.bandwidth, global_data.af_phb_bndw);
      strcpy(class_info.rate,
            packet.data.phb.phb_params.be_class.class_bndw);
      strcpy(class_info.avpkt, DSPEC_AVG_PKT);
      strcpy(class_info.allot,DSPEC_ALLOT);
      strcpy(class_info.weight,DSPEC_WEIGHT);
      strcpy(class_info.maxburst, "21");
      class_info.bounded = 1;
      class_info.isolated = 0;
      class_info.borrow = 0;
      class_info.priority =
          packet.data.phb.phb_params.be_class.priority;
      if (add_cbq_class(global_data.device_name, 0, &class_info) < 0)
          return -1;

      strcpy(filter_info.mask,"NULL");
      strcpy(filter_info.shift,"NULL");
      strcpy(filter_info.parent,"3:0");
      sprintf(filter_info.handle,"%d",0);
      sprintf(filter_info.classid,"3:5");
      filter_info.priority = 1;
      filter_info.pass_on = 0;
      if (add_filter(global_data.device_name, &filter_info) < 0)
          return -1;

      qdisc_info.type = RED;
      sprintf(qdisc_info.q_info.red_queue.parent,"3:5");
      qdisc_info.q_info.red_queue.red_parameters.burst = 20;
      qdisc_info.q_info.red_queue.red_parameters.drop_prob
          = packet.data.phb.phb_params.be_class.drop_prob;
```

```
  strcpy(qdisc_info.q_info.red_queue.red_parameters.max_threshold,
         DSPEC_GRED_MAX_THRESHOLD);
  strcpy(qdisc_info.q_info.red_queue.red_parameters.min_threshold,
         DSPEC_GRED_MIN_THRESHOLD);
  strcpy(qdisc_info.q_info.red_queue.red_parameters.bandwidth,
        global_data.device_bndw);
  strcpy(qdisc_info.q_info.red_queue.red_parameters.avpkt,
         DSPEC_AVG_PKT);
  strcpy(qdisc_info.q_info.red_queue.red_parameters.limit,
         DSPEC_GRED_MAX_THRESHOLD);
  if (add_queue(global_data.device_name, 0, &qdisc_info) < 0)
      return -1;
}
else
{
  /* If the architecture has only EF and BE, the 2:2 class is
   * created and a red queue associated with it and the
   * corresponding filter is also created
   */
  strcpy(class_info.parent, "2:0");
  sprintf(class_info.handle, "2:2");
  strcpy(class_info.bandwidth, global_data.device_bndw);
  strcpy(class_info.rate,
        packet.data.phb.phb_params.be_class.class_bndw);
  strcpy(class_info.avpkt, DSPEC_AVG_PKT);
  strcpy(class_info.allot,DSPEC_ALLOT);
  strcpy(class_info.weight,DSPEC_WEIGHT);
  strcpy(class_info.maxburst, "21");
  class_info.bounded = 1;
  class_info.isolated = 0;
  class_info.borrow = 0;
  class_info.priority =
      packet.data.phb.phb_params.be_class.priority;
  if (add_cbq_class(global_data.device_name, 0, &class_info) < 0)
      return -1;

  strcpy(filter_info.mask,"NULL");
  strcpy(filter_info.shift,"NULL");
  strcpy(filter_info.parent,"2:0");
  sprintf(filter_info.handle,"%d",0);
  sprintf(filter_info.classid,"2:2");
  filter_info.priority = 2;
  filter_info.pass_on = 1;
  if (add_filter(global_data.device_name, &filter_info) < 0)
      return -1;

  qdisc_info.type = RED;
  sprintf(qdisc_info.q_info.red_queue.parent,"2:2");
  qdisc_info.q_info.red_queue.red_parameters.burst = 20;
  qdisc_info.q_info.red_queue.red_parameters.drop_prob
      = packet.data.phb.phb_params.be_class.drop_prob;
```

```
            strcpy(qdisc_info.q_info.red_queue.red_parameters.max_threshold,
                   DSPEC_GRED_MAX_THRESHOLD);
            strcpy(qdisc_info.q_info.red_queue.red_parameters.min_threshold,
                   DSPEC_GRED_MIN_THRESHOLD);
            strcpy(qdisc_info.q_info.red_queue.red_parameters.bandwidth,
                   global_data.device_bndw);
            strcpy(qdisc_info.q_info.red_queue.red_parameters.avpkt,
                   DSPEC_AVG_PKT);
            strcpy(qdisc_info.q_info.red_queue.red_parameters.limit,
                   DSPEC_GRED_MAX_THRESHOLD);
            if (add_queue(global_data.device_name, 0, &qdisc_info) < 0)
                return -1;
          }
          break;
    }
    return 0;
}

/* This function creates the AF class, associates a GRED with it and also
 * creates a filter to look at the class number and direct the packets to the
 * appropriate class, this also configures the GRED for the different drop
 * priorities and creates the appropriate filters
 */
int set_af_class(packet_info packet)
{
    cbq_class class_info;
    filter_ip filter_info;
    qdisc setup_gred;
    qdisc config_gred;
    int handle_number;
    int count;

    strcpy(class_info.parent, "3:0");
    sprintf(class_info.handle, "3:%d", packet.data.af_class.af_class_num);
    strcpy(class_info.bandwidth, global_data.af_phb_bndw);
    strcpy(class_info.rate, packet.data.af_class.class_bndw);
    strcpy(class_info.avpkt, DSPEC_AVG_PKT);
    strcpy(class_info.allot,DSPEC_ALLOT);
    strcpy(class_info.weight,DSPEC_WEIGHT);
    strcpy(class_info.maxburst, "21");
    class_info.bounded = 1;
    class_info.isolated = 0;
    class_info.priority = packet.data.af_class.priority;
    if (add_cbq_class(global_data.device_name, 0, &class_info) < 0)
      return -1;

    strcpy(filter_info.mask,"NULL");
    strcpy(filter_info.shift,"NULL");
    strcpy(filter_info.parent,"3:0");
    sprintf(filter_info.handle,"%d",packet.data.af_class.af_class_num);
    sprintf(filter_info.classid,"3:%d", packet.data.af_class.af_class_num);
    filter_info.priority = 1;
    filter_info.pass_on = 0;
```

```
if (add_filter(global_data.device_name, &filter_info) < 0)
  return -1;

setup_gred.type = GRED;
setup_gred.q_info.gred_queue.format = 1;
sprintf(setup_gred.q_info.gred_queue.parent,"3:%d",
      packet.data.af_class.af_class_num);
strcpy(setup_gred.handle,"NULL");
setup_gred.q_info.gred_queue.gred_format.gred1.number_of_dps
  = packet.data.af_class.num_of_dps;
setup_gred.q_info.gred_queue.gred_format.gred1.default_dps
  = packet.data.af_class.num_of_dps - 1;
if (add_queue(global_data.device_name, 0, &setup_gred) < 0)
  return -1;

for (count = 0; count < packet.data.af_class.num_of_dps; count++)
{
  memset(&config_gred, 0, sizeof(config_gred));
  memset(&filter_info, 0, sizeof(filter_ip));

  config_gred.type = GRED;
  config_gred.q_info.gred_queue.format = 2;
  sprintf(config_gred.q_info.gred_queue.parent,"3:%d",
        packet.data.af_class.af_class_num);
  config_gred.q_info.gred_queue.gred_format.gred2.dp_number=count+1;
  config_gred.q_info.gred_queue.gred_format.gred2.burst = 20;
  config_gred.q_info.gred_queue.gred_format.gred2.drop_prob
      = packet.data.af_class.drop_prob[count];
  strcpy(config_gred.q_info.gred_queue.gred_format.gred2.max_threshold,
        DSPEC_GRED_MAX_THRESHOLD);
  strcpy(config_gred.q_info.gred_queue.gred_format.gred2.min_threshold,
        DSPEC_GRED_MIN_THRESHOLD);
  strcpy(config_gred.q_info.gred_queue.gred_format.gred2.bandwidth,
        global_data.af_phb_bndw);
  strcpy(config_gred.q_info.gred_queue.gred_format.gred2.avpkt,
        DSPEC_AVG_PKT);
  strcpy(config_gred.q_info.gred_queue.gred_format.gred2.limit,
       DSPEC_GRED_MAX_THRESHOLD);
  if (add_queue(global_data.device_name, 0, &config_gred) < 0)
      return -1;


  strcpy(filter_info.mask,"NULL");
  strcpy(filter_info.shift,"NULL");
  strcpy(filter_info.parent,"1:0");
  handle_number = 8*packet.data.af_class.af_class_num + (count+1)*2;

  sprintf(filter_info.handle,"%d",handle_number);
  sprintf(filter_info.classid,"1:1%d%d",
        packet.data.af_class.af_class_num, (count+1));
  filter_info.priority = 1;
```

```
 File Name       : user_interface.h
 Owner           : V.Vanitha
 Creation date   : 3.10.2001
 Description     : Header File.
                   This file defines the basic data structures to be provided
                    by the user for configuring the classes and queues.

*/


#ifndef _USER_INTERFACE_
#define _USER_INTERFACE_


#define MAX_ALLOC_SIZE 16

/* The PHBs provided
 * AF - Assured Forwarding
 * EF - Expedited Forwarding
 * BE - Best effort
 */
enum phb_type {

      AF,
      EF,
      BE
};

/* This defines the packet type.
 * DEVICE     - Device specific data - device name, device bandwidth etc
 * PHB            - The PHB data - PHB type and bandwidth
 * AF_CLASS   - AF Class specific details
 * EOF_SETUP - To signify end of setup
 * DELETE     - for deleting the queues and classes
 *
 */
enum pkt_type {

      DEVICE,
      PHB,
      AF_CLASS,
      EOF_SETUP,
      DELETE
};

/* This defines the architecture to be setup at the output interface.
 * EF_BE      - EF and BE
 * TWO_BIT    - Coexistence of AF, EF and BE - the two bit architecture
 */
typedef enum _setup {

      EF_BE,
      TWO_BIT
} setup;
```

```
ass specific info
F class number, bandwidth, priority, number of drop precedences and
rop precedence values

 struct _af_class_info {

nt af_class_num;
har class_bndw[MAX_ALLOC_SIZE];
nt priority;
nt num_of_dps;
ouble drop_prob[3];

ass_info;


acket to be sent from the client to the diffspec server has the
and to identify what info is being passed and the data portion contains
actual parameters required for configuration

 struct _packet {

num pkt_type cmd;
nion {

     device_info device;
     phb_info    phb;
     af_class_info af_class;
 data;

t_info;

 data structure is required for maintaining some information required
ally

 struct _global_info {

har device_name[16];
har device_bndw[MAX_ALLOC_SIZE];
har af_phb_bndw[MAX_ALLOC_SIZE];
etup setup_flag;

l_info;
```