Implementation and Configuration of a Linux Differentiated Services Router

Günther Stattenberger Torsten Braun

University of Berne
Institute of Computer Science and Applied Mathematics

[stattenb|braun]@iam.unibe.ch

November 23, 2000

CR Categories: C.2.1 Network Architecture and Design; C.2.2 Network Protocols
General Terms: Differentiated Services, Quality of Service
Additional Keywords: Linux, Implementation

Abstract

Despite of the increasing bandwith, that is available for users, there is a great demand for Quality of Service support in the Internet. Especially new applications like Voice over IP, Video on Demand and Digital Video Broadcast need a reliable bandwidth guarantee and certain delay / jitter values. The Differentiated Services approach is able to provide both, end-to-end bandwidth and delay guarantees to the users offering different services suitable for various applications. Our implementation of Differentiated Services on Linux Routers will be the basis for several research activities to investigate the behaviour of this promising approach to support Quality of Service.

*The work presented in this paper is supported by NEC Europe Ltd., Adenauerplatz 6, D-69115 Heidelberg, Tel: +49 6221 90511-0

Contents

Part I

A Short Overview of DiffServ

1 Introduction

Differentiated Service (DiffServ) is a well known way to support Quality of Service in the Internet. It is — unlike Integrated Services — based on an aggregation of flows and therefore without the need of multifield classification at each hop. Reservations are made for any aggregation of flows (e.g. for all flows between two subnets). These reservations are rather static since no dynamic reservations for a single connection are possible for scaling reasons.

IP packets are marked with different priorities, either in an end system or in a router. According to the different priority classes, the DiffServ routers reserve corresponding shares of resources (i.e. bandwidth and buffer space). Marking the packets is done by writing a DiffServ Codepoint (DSCP) into the Type of Service - byte (ToS byte) of the IP header (see figure 1).

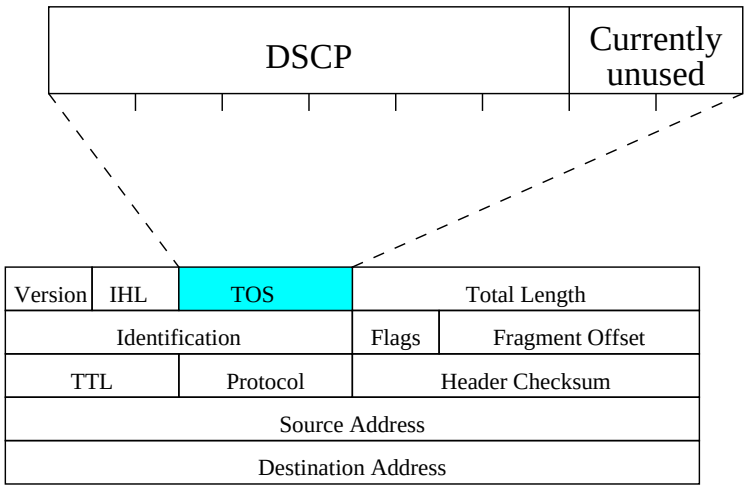

Figure 1: DiffServ Codepoint in the IP Header

Currently the first six bits of the ToS byte are used for the DSCP. All router implementations should support the recommended DSCP-to-PHB mapping. A PHB (per hop behaviour) is a forwarding behaviour which a router performs on a packet. In DiffServ such a forwarding behaviour is built of a combination of several components. These components are discussed in section 3. A chain of routers supporting the same PHB will provide an end-to-end Quality of Service.

2 Per Hop Behaviour (PHB)

Several services and their corresponding codepoints have been defined in the Internet community. Nowadays, DiffServ is mainly based on the two traffic classes defined in [2] and [1]. We will now give a short overwiew of those two classes.

2.1 Expedited Forwarding / Premium Service

Premium Service refers to the traffic handling commonly known as expedited forwarding which is defined in [2]. This service shall provide low delay, low loss and low jitter at a fixed rate. It will appear to the endpoints like a "virtual leased line". To fulfill those requirements, traffic marked for expedited forwarding has to meet very short queues. Therefore one has to ensure that there is not more EF-traffic arriving at one router than the router's settings allow to be transported. The departure rate at each hop should be independent of the intensity of any other traffic arriving at the router. DiffServ routers will for example give premium service packets priority over other traffic but in this case strictly police any traffic exceeding the negotiated limit to prevent premium service to starve any other traffic. Another very important topic is that reordering of EF packets must not appear. The recommended codepoint for the EF PHB is 101110.

2.2 Assured Service

Assured Forwarding defines a service which assures a high probability to forward the traffic through the network as long as the bandwidth does not exceed the negotiated limit. However, any traffic exceeding the profile will be forwarded too, but with higher probability to be dropped in case of congestion. It is also very important, that reordering of packets of the same microflow is strictly forbidden again.

There are four different assured service classes defined, each allocationg a specific share of resources (i.e. bandwidth and buffer space) and thus having a different level of forwarding assurance. Within these classes packets can be marked with three possible drop precedence values. In case of congestion the in-profile traffic will be protected by preferably dropping packets with higher drop precedence. An overview of the codepoint values for the assured forwarding classes is given in table 1.

3 DiffServ Components

Each implementation of a DiffServ router consists of a combination of different components (see figure 2), which interact in a certain way to ensure the proper forwarding of traffic according to the requirements of the individual PHB. Not all components are required in each DiffServ node, this depends on the router and on the service type. Those components provide different traffic conditioning functions that range from simple marking to complex shaping and policing actions.

Dropping Precedences	AF Classes			
	Class 1	Class 2	Class 3	Class 4
low	001010	010010	011010	100010
medium	001100	010100	011100	100100
high	001110	010110	011110	100110

Table 1: Assured Forwarding Codepoints

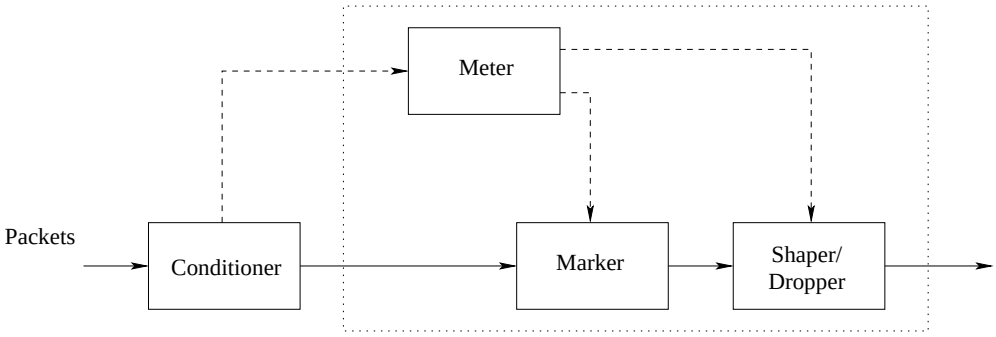

Figure 2: Combination of DiffServ Components in a Router

3.1 Classifier

A classifier matches packets according to its profile and forwards them to the corresponding component for further processing. There are two types of classifiers

- **Multi-Field Classifier**: A multi-field (MA) classifier matches on a combination of IP header fields (adresses, protocol ID, ToS byte) or even port numbers.

- **Behaviour Aggregate Classifier**: Contrary to the MA classifier the behaviour aggregate (BA) classifier only classifies on the DSCP of the packets.

3.2 Marker

Markers set the DSCP of IP packets. By setting the codepoint of packets, they are added to a traffic aggregate which provides important information for BA classifiers. Marking can be done statically (i.e. all packets are marked the same way) or depending on the state of some meter. This functionalty is normally used in ingress routers for tagging the traffic flow of a single host or for retagging whole traffic aggregates coming from a connected DiffServ domain.

3.3 Meter

Meters measure the amount of traffic that passes by. They are located in the forwarding chain of almost all traffic aggregates as they provide the basic information for many DiffServ components, like markers, shapers and policers.

3.4 Shaper

Shapers delay some packets on their transmisson path in order to bring the traffic flow into compliance with some Traffic Conditioning Specification (TCS). Those packets are stored in some queue and discarded if not enough buffer space is available. A properly configured shaper therefore provides some burst protection while not dropping the packets of the bursty traffic source. The usual location of a shaper is behind a classifier at the ingress router.

3.5 Policer

Unlike the shaper a policer does not store any packets but simply drop any packets that do not meet the traffic conditioning specification. A policer can be implemented as a shaper with little or no buffer space. Policers are normally used in interior- and egress routers as they rely on the traffic being correctly shaped.

4 DiffServ Router Types

A simple DiffServ Network with 2 ISPs is shown in figure 3. It shows the four DiffServ Router types, that form the two DiffServ domains and connect the two hosts and provide an end-to-end quality of service. This router specifications are valid only for traffic from domain A to domain B. For traffic flowing in the opposite direction the boundary routers (ingress and egress routers) change their types. Note, that a router may also act in two ways, e.g. as interior router for one traffic flow while being ingress router for another. This can be seen in figure 3 for the router adjacent to Host B. The next sections describe the requirements of these four router types which follow the DiffServ architecture [3].

4.1 First Hop Router

A first hop router normally is responsible for marking incoming packets according to its profile. This profile allows to define a DiffServ Code Point (DSCP) for each flow specified by the six-tuple (source address / netmask, source port, destination address / netmask, destination port, protocol, DSCP). For small networks this functionality can be easily included in an ingress router.

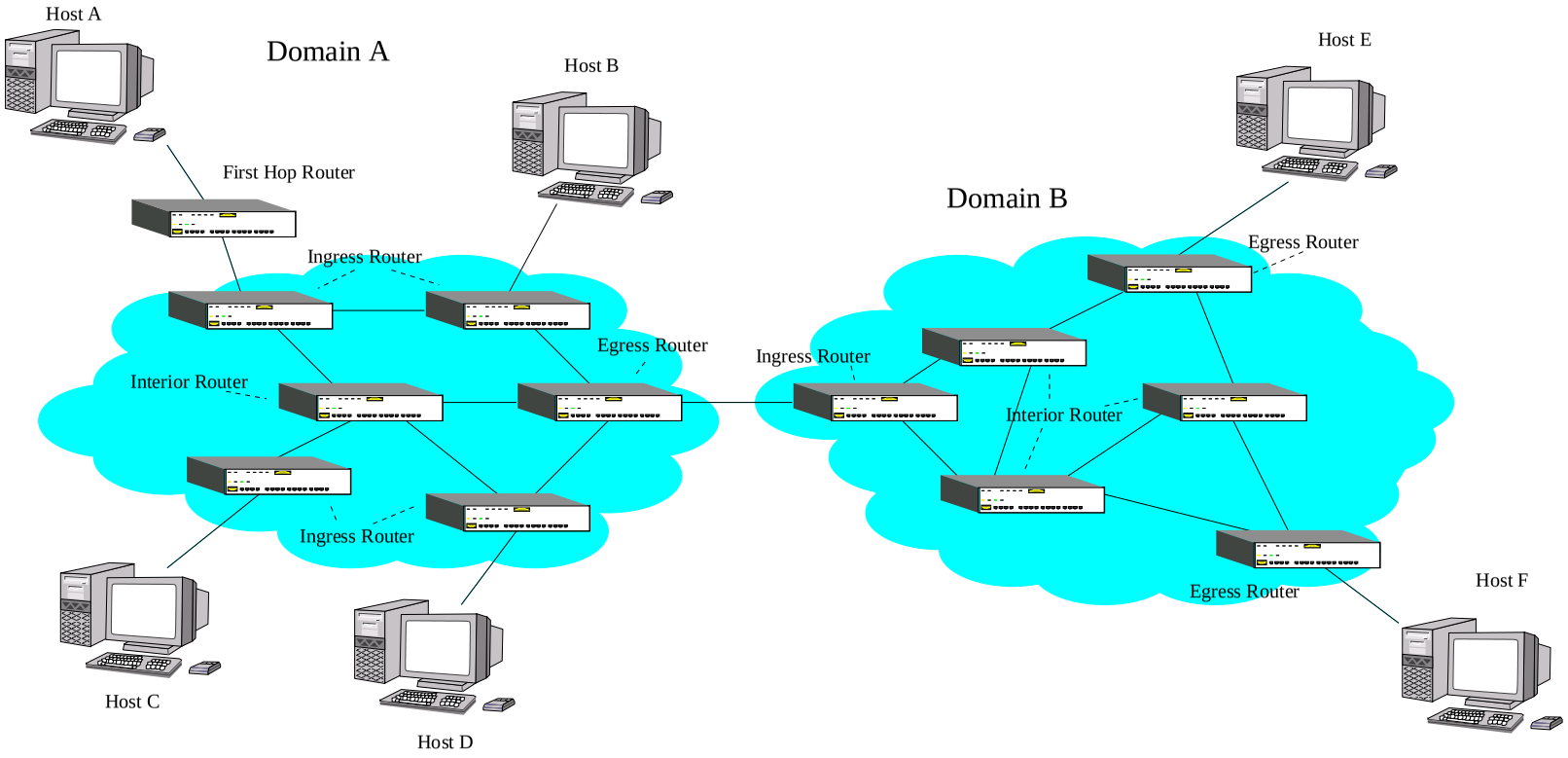

Figure 3: A simple DiffServ Network

4.2 Ingress Router

The configuration of the ingress router is the most complex one because at the ingress point each flow has to be handled seperately and therefore not all properties of flow aggregation are available. The ingress router has to ensure, that the traffic entering the DiffServ domain conforms to any traffic conditioning specification (TCS) between it and the connected domain. Therefore, it will have to perform some traffic conditioning functions like shaping or dropping.

4.3 Interior Router

Routers not located at the border of a DiffServ domain can be very simple, as the most complex functions of classification and traffic conditioning are performed at the ingress and egress points of the network. However an interior node may perform some limited traffic conditioning like codepoint-remarking or policing to ensure the proper forwarding behaviour of the traffic classes. Normally the interior router should not have to perform any policing actions, therefore it is useful to trace those actions to detect serious misconfigurations at the border routers.

4.4 Egress Router

An egress router can perform traffic conditioning functions on traffic leaving the DiffServ domain depending on the TCS between it and the connected domain. This functions normally will not depend on multifield classification but act on a behaviour aggregate. Therefore the configuration of the egress router is less complex than the configuration of the ingress router.

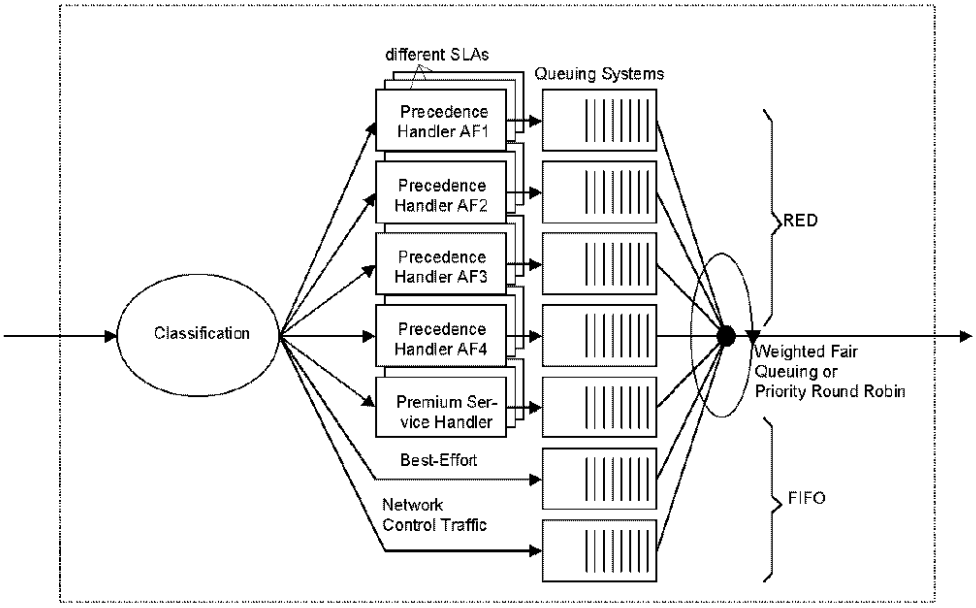

Figure 4: View of a First Hop, an Ingress or an Egress Router

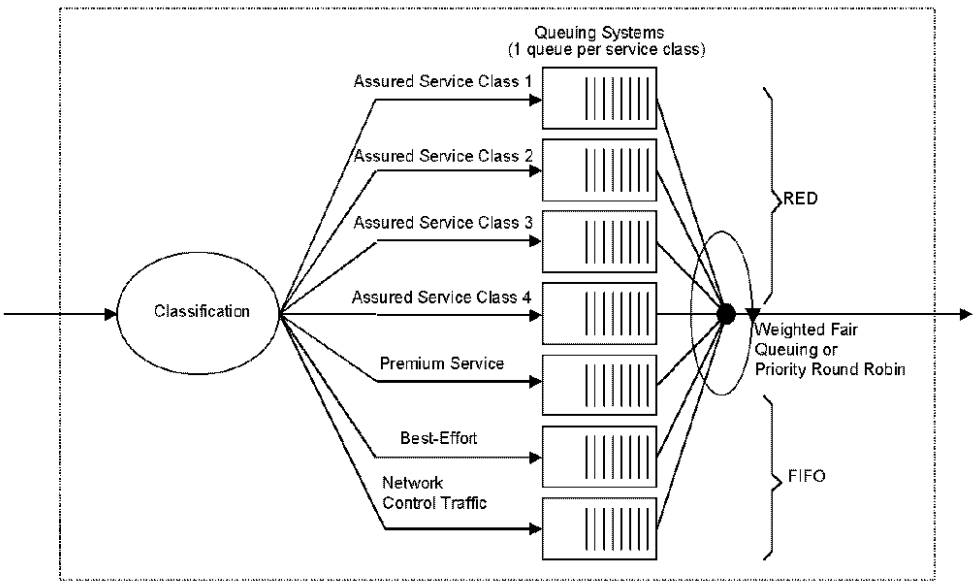

Figure 5: View of an Interior Router

Part II
Setting Up a DiffServ Router

5 Compiling the DiffServ Code

5.1 Source Files

Our actual implementation of DiffServ is built on top of a Linux 2.2.9 Kernel and some special software packets. Therefore we need a 2.2.9 Kernel Source tree [5] and a release of iproute2 version 2.2.4 (actually we use [6]) which are available at the given URLs. Additionally we need the dstable package and the kernel- and iproute2 patches that contain the DiffServ implementation of the University of Berne. This will soon be available in the public domain and can then be accessed by NEC Europe Ltd. (send an email request to brunner@ccrle.nec.de).

5.2 Building and loading the DiffServ Modules

We assume, that the kernel sources and all other source files are unpacked and placed at the usual location `/usr/src/` (see also the README file of the linuxds distribution). Then we can apply the patch containing the DiffServ code by using

```
patch --backup --strip=<depth>  --input=<file>
```

To select the DiffServ modules for compiling the option *Code maturity level options rightarrow Prompt for development and/or incomplete code/drivers* has to be enabled. Now one can select the DiffServ modules in the Kernel configuration skript, submenu *Networking Options → QoS and fair queueing.*

Once the kernel and the modules have been built and installed correctly (see [7] for details) the next step is to compile iproute2. First the Makefile at the `iproute2` directory has to be changed: the KERNEL_INCLUDE path should point to the correct linux kernel include directory. Some other changes depend on the libc - version and should be performed after consulting the README file. Now iproute2 can be compiled by `make`.

The last step is to compile `dstab` by running `make` in the dstab directory.

Now we can reboot the new kernel and load the DiffServ Modules into the kernel memory. This can be done manually by the superuser

```
modprobe <module>
```

where `<module>` is the name of the module to be loaded. During the installation the modules were copied to the `/lib/modules/`*`kernel-version`*`/misc` directory. All module names begin with `sch_`, appended the module names introduced in section 6.

It is much easier to load the modules at boot time by adding their names to the `/etc/modules` file. Now the modules will be loaded automatically every time the computer boots up.

5.3 Activating the Modules

As DiffServ heavily interferes with the traffic forwarding scheme it is now necessary to give a short introduction to Linux' traffic control mechanisms. A more detailled description can be found in [8] and [9].

The kernel part of the traffic control functions is separated in four components, namely queueing disciplines, classes within a queueing discipline, filters and policing. The queueing disciplines provide the important functions of enqueueing and dequeueing. The queueing discipline may have one or more classes. The classes do not store the packets by themselves but use other attached queueing disciplines for that purpose. Therefore by combining several queueing disciplines, a very flexible way of traffic control can be set up (see section 4).

The user level program to control the interaction between the interface and the queueing disciplines is called `tc` (*traffic control*), which is included in the iproute2 package. It is used to set up new queueing disciplines, to attach new queueing disciplines to classes, to change the settings and finally to remove them completely. The usage of `tc` is:

```
tc [OPTIONS] OBJECT {COMMAND | help}
```

where

```
OPTIONS = { -s[statistics] | -d[details] | -r[raw] }
OBJECT = { qdisc | class | filter}
```

As, apart from `dstable`, which is not invoked via `tc`, all DiffServ modules are queueing disciplines we now concentrate on the `tc qdisc` options.

```
tc qdisc [add| del | replace | change | get] \
dev <device> <node> handle <handle>: <module> <options>
```

where the options are listed below:

- <device>: The router's interface to which the module shall be attached. This is normally the outgoing interface of the traffic on this router.
- <node>: The location of the module in the tree of queueing disciplines. <node> is either `root` or `parent <parent_id>:<child_id>` using parent_id as the parent's handle and child_id as the child-class index of the parent (normally this is 1 as most queueing disciplines have only one child).
- <handle>: An identification number for the module used for building the tree of queuing disciplines.

The module names <module> and their options will be described in detail in sections 6.3 – 6.7.

6 Description of the DiffServ Modules

6.1 Queues

The different DiffServ components normally don't store any packets but rather rely on attached queues where they can forward the packets to. There are different queueing algorithms, which are included in various packages. For our DiffServ implementation we use a FIFO (first in first out) and a token bucket filter queue (TBF) included in `iproute2` and a TRIO queue (Threeway RED queue) implemented by ourselves.

6.1.1 FIFO Queue

The algorithm of the FIFO queue is the most simple one: packets are enqueued until the buffer space is filled and discarded afterwards, dequeueing is done in the order of appearance. There exist two types of FIFO queues, one based on packet counting, the other based on byte counting. The commands for the FIFO queues are

```
pfifo limit <size>
```

and

```
bfifo limit <size>
```

for packet-based and byte-based queues, taking the buffer size as the only argument.

6.1.2 Token Bucket Filter

A token bucket filter is a rate-delimiting mechanism, which can be attached to a queue. In case of the `iproute2`-TBF it is a FIFO queue and the TBF is at the dequeueing point. The token bucket filter algorithm uses a certain variable t representing an amount of bytes (the bucket) which is periodically filled up at the configured rate of the TBF. A dequeue event for a packet of size s can only be performed, when enough tokens are available ($t > s$). If the bucket is full, no more tokens can be stored.

A token bucket filter has the following usage:

```
tbf rate <bandwidth> buffer <burst1> limit <burst2>
```

where the rate specifies the bandwidth of the token bucket filter, the buffer size limits long term bursts whereas the limit size limits short term bursts. A more detailled explanation can be found in the source code [10].

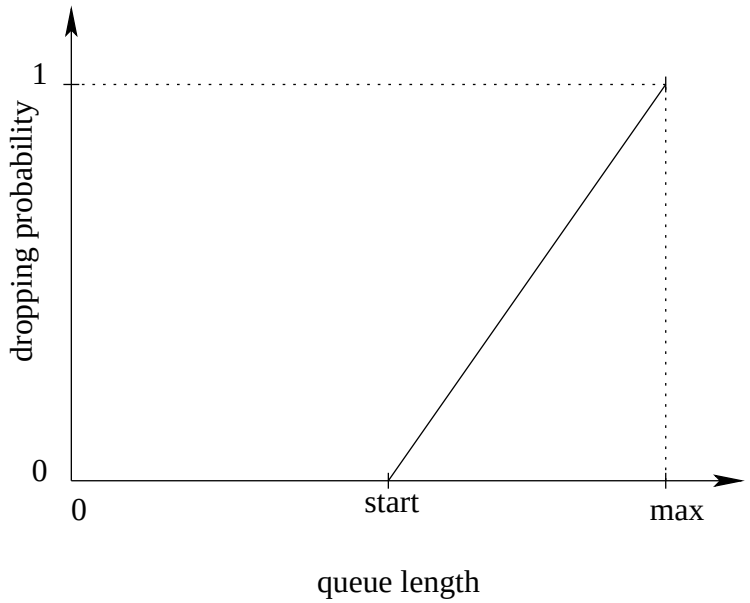

Figure 6: Dropping probability of a RED queue

6.1.3 TRIO Queue

The TRIO (Threeway RED for In and Out) queue is an extension of the RED (Random Early Drop) queue, which shall be introduced first. In contrast to the FIFO queue the RED queue starts dropping packets before the buffer space is filled. The probability of being dropped is a function of the queue length (see figure 6).

The TRIO queue now uses three different dropping probability functions for three different packet types on *one* queue (see figure 7). The three packet types are called low-, medium- and high dropping precedence packets according to the dropping probability funcion they are applied to. The effect of a TRIO queue is, that low dropping precedence packets are protected by the earlier dropping of high- and medium dropping precedence packets. A typical queue-state for a TRIO queue is shown in figure 7: the red and yellow marked high- and medium dropping precedence packets only occur in the first part of the queue, while the low dropping precedence packets can be stored up to the maximal queue size. This results in shorter queues for high- and medium dropping precedence traffic and a longer queue for low dropping precedence traffic and so the chance of low dropping precedence traffic to be dropped is reduced to a minimum.

The command for the TRIO queue looks like this:

```
trio limit <queue_length> \
low_begin <start1> low_end <end1> \
medium_begin <start2> medium_end <end2> \
high_begin <start3> high_end <end3>
```

The queue length is given in packets and the remaining six parameters are the settings for the three dropping probability functions for low-, medium- and high dropping probability

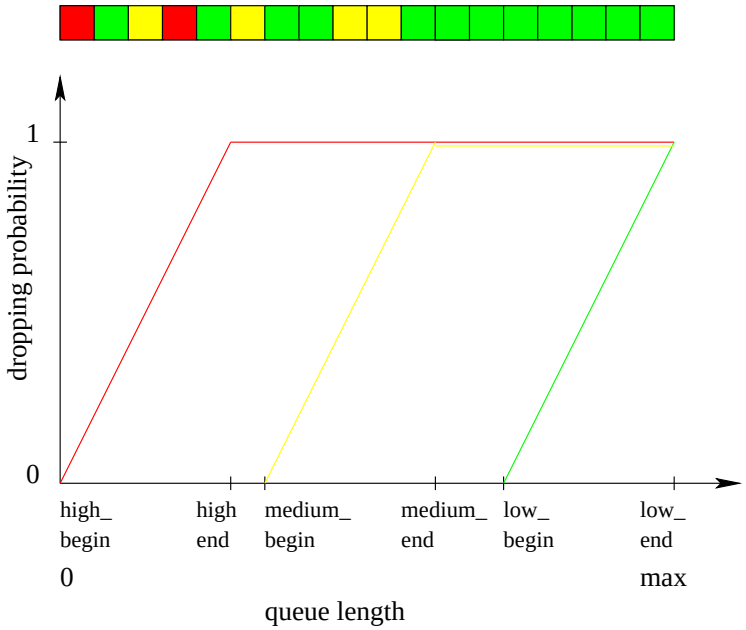

Figure 7: Dropping probabilities and state of a TRIO queue

traffic.

6.2 The DiffServ Table Module

6.2.1 The `dstab` command

Most DiffServ queueing discipline modules need a table of flows — aggregated or not — that describes the actions the queueing discipline has to perform on these flows. For this purpose we implemented a module — the `dstable` module — providing some functions for those tables to be loaded into the memory of the router (the `/proc`-filesystem) and to be accessed by the queueing disciplines. A table can be loaded by

```
dstab l <index> <filename>
```

where <index> is a number by which the table can be accessed by the queuing disciplines. Other options are available for removing a table <index> from the memory

```
dstab f <index>
```

(f is an abbreviation for "flush") and printing a specific table

```
dstab p <index>
```

or all active (that is non-empty) tables

```
dstab a
```

A description of the `dstab` command and the table format can also be found in the `README` file at the `dstab`-directory.

6.2.2 The Table Format

A table entry consists of a line representing a single or aggregated flow, which itself consists of eleven fields separated by whitespaces (i.e. blanks or tabs). Comment lines start with a #. The table entry elements are:

- `source address`: The source IP-address can be either a host or a network address. More advanced subnetting can be managed by the netmask (see the next entry).
- `netmask for source address`: The value of the netmask and the source address are combined via an AND operation and the result is used as reference to match the flow. Therefore the netmask 255.255.255.255 restricts the traffic source to a specific host whereas the netmask 0.0.0.0 can be used to ignore the source address.
- `source port`: The source port value can be used to restrict a flow to a specific UDP/TCP port number. The wildcard * can be used to ignore this field.
- `destination address`: see source address
- `destination address netmask`: see source address netmask
- `destination port`: see source port
- `protocol`: This value can be used to specify a certain IP protocol. Valid entries are strings like "UDP", "TCP" or "ICMP". To ignore the protocol, use *.
- `classpoint for incoming traffic`: This identifier serves as matching entry for already marked packets in a behaviour-aggregate classifier. Only the three most significant bits will be compared. Possible entries are the strings "ps", "rt", "as1", "as2", "as3" and "as4". These values represent the different traffic classes "premium service", "routing traffic" and "assured service" of class $1 - 4$ and are replaced by their codepoint values which are proposed in [1], [2] and [4]. To ignore the classpoint, * can be used again.

The values above are intended to identify a specific flow and serve as a filter mask to be used in a classifier. On the other hand the following values will be used as parameters by the queueing disciplines.

- `classpoint for outgoing traffic`: This is a parameter for the service handler that specifies the classpoint to which a packet matching the first eight entries will be marked. Valid entries are the same as for incoming traffic; the wildcard value * can be used to keep the service handler module from retagging the packets.

- `low dropping precedence rate` or `class index` or `premium service traffic rate`: This field has different meaning depending on the queueing discipline. For the assured service precedence handler it specifies the rate of low dropping precedence traffic. It is used as the rate of the token bucket filter limiting the amount of low dropping precedence packets. The premium service shaper interprets the value as a class index. Any traffic matching the profile (the first eight entries) will be forwarded to the class with the given index attached to the premium shaper (this will normally be a token bucket filter). The premium policer interprets the value as the maximum rate of premium service traffic that is allowed to cross the router. It will use it as the rate of its built-in token bucket filter.

- `medium dropping precedence rate`: this entry is only used by the assured service precedence handler to set the rate of its second token bucket filter, which limits the rate of medium dropping precedence traffic.

6.3 Service Handler

The service handler module marks incoming packets according to the profile found in the table at location <index>. It tags each packet matching the first eight table entries using the `classpoint for outgoing traffic`. It is normally used as the root queuing discipline in first hop - or ingress routers. To install a service handler use

```
serv_handler table_id <index>
```

with <index> describing the `table_id` number.

6.4 Classifier

The classifier forwards the traffic to one of its seven child classes, depending on the DSCP of the packets. The assured service packets will be enqueued to child classes 1 –4, premium service packets to child class 5, routing traffic to child class 6 and best effort traffig goes to child class 7. If there is no service handler the classifier will be the root queueing discipline. For dequeueing the classifier module also acts as a Priority - Weighted Round Robin. This means that the classifier will dequeue class 5 with the highest priority, class 6 with the second highest priority and all other classes are dequeued with least priority. These remaining classes (i.e. assured service and best effort traffic) are handled by a weighted round robin, that is each class has a different probability of being dequeued. This probability is compared with the share this traffic class got, which again is calculated from a log of the last packets sent. The log size and the probabilities of the first four classes are the parameters of the classifier command:

```
dsclsfr as1 <weight1> as2 <weight2> as3 <weight3> \
as4 <weight4> log_size <size>
```

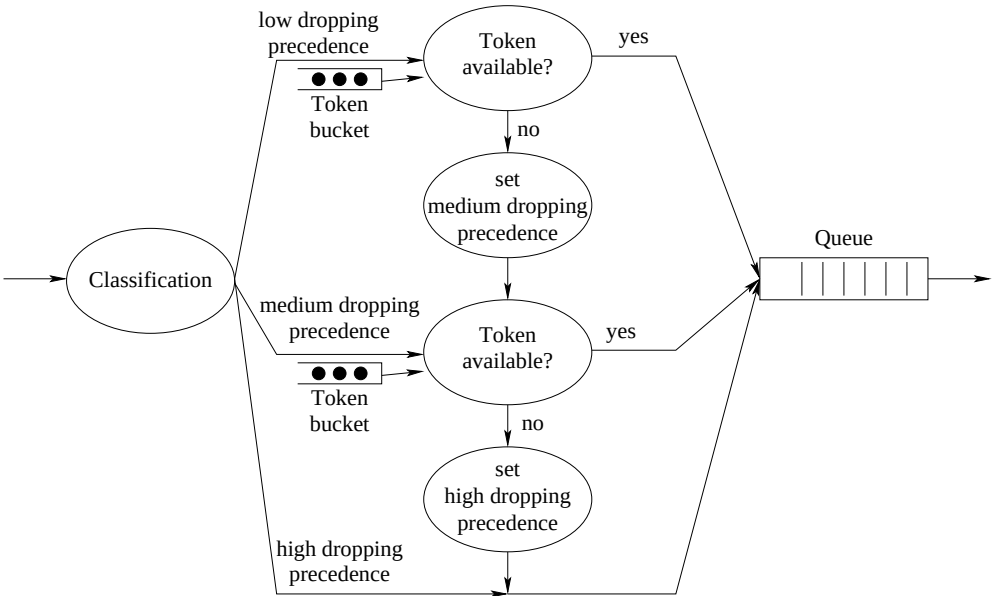

Figure 8: Functionality of the Precedence Handler

As the first four classes will normally be used for assured service, the according weights are labelled as1 – as4. The log size specifies the number of packets that are taken into account for calculating the shares of each traffic class. The weights are decimals between 0.0 and 1.0; their sum must be smaller than 1.0 as the rest is the share of bandwidth best effort traffic will get. This share must be > 0.0 (see [3]).

6.5 Precedence Handler

The precedence handler is responsible to remark assured service packets to medium or high dropping precedence if the total amount of assured service traffic exceeds the negotiated limit. For this purpose it uses the last two table entries (the low_priority and the medium_priority field) as the limits for bandwidth of low respectively medium dropping precedence traffic. The limitation is done by two internal token bucket filters (see figure 8).

As it is required to handle the four assured service classes independently, it is necessary to install four precedence handlers — one for each class. The installation is done by

```
prec_handler table_id <handle> dscp <class>
```

specifying <class> the assured service class (as1 – as4) and <handle>the table_id of the assured service flows' table.

6.6 Premium Service Shaper

The premium service shaper provides the functions to handle several premium service microflows separately by installing an individual classe for each flow. To each class a queue

matching the requirements of premium service (e.g. a token bucket filter, see [2]) can be attached. For this reason the premium service shaper is used only in first-hop - and ingress routers. It is called using

```
premium_shaper table_id <handle> classes <number>
```

given as parameters the number of classes to be installed and a handle to a table containing each premium service microflow and the corresponding class number in the `class index` field.

6.7 Premium Service Policer

The premium service policer does not store the incoming packets in a queue but only limits the total amount of bandwidth using a token bucket filter. The policer is therefore used in interior and egress routers to control and limit the amount of aggregated premium service traffic at the router. To install a premium service policer use

```
premium_policer table_id <handle>
```

with a handle to a table including a line with (normally) empty adress- and port fields but only the rate of the token bucket filter in the `premium service rate` field.

7 Router Configuration Example Scripts

Referring to sections 4 and 6 we will now present scripts to set up a small DiffServ Network that we built at University of Berne.

From now on we suppose that the DiffServ modules are already loaded into the kernel (see section 5.2). Then we can easily configure the router at the command line using the `tc` and `dstab` commands (see sections 5.3 and 6.2.1). To simplify the setup procedure, calling the configuration commands is mostly done by a small script. The router setup presented in the following sections can be interpreted as a "traffic forwarding tree", starting at the root queueing discipline with the packets going up to the queues at the leafs of the tree and being dequeued the same way they went up. Those trees are shown in figures 9 – 11.

First the configuration tables have to be loaded. These tables contain the flows which have to be handled by the queueing disciplines of the router, and some parameters. After that the tree of queueing disciplines can be built via `tc`. The names of the queueing discipline modules have been presented in section 6. We may omit calling the FIFO queues as this are the default queueing disciplines for routing and best effort traffic.

We suppose to create a separate table for each queueing discipline at each interface to keep tab on where each traffic flow goes to. As in our test network we use but one interface per router for DiffServ, we have to create four tables: one for the service handler, the premium service shaper, the precedence handler and the premium service policer.

All reservations are made for connections between Host A (10.1.1.1) and Host B (10.1.4.2) and are restricted to the two hosts (netmask 255.255.255.255). For a better understanding the port numbers indicate whether we use premium (port 50xx) or assured service (port 51xx) and also the bandwidth in MBit/s (ports xx05, xx10, xx30 for 5, 10 and 30 MBit/s). We do not discriminate between TCP and UDP (wildcard * in the `protocol` field).

The tables presented here combine several tables used for testing and were not used in the experiments, as we just tested each flow seperately. Yet they can be used as an example, especially for flow (or bandwidth) aggregation. We make reservations for 5 and 30 MBit/s premium service and 5 and 10 MBit/s assured service. The whole assured service bandwidth will be marked for low dropping precedence, any traffic exceeding this limit will be marked high dropping precedence (i.e. no medium dropping precedence traffic). Note, that within the tables all bandwith values are in bytes/s!

7.1 First Hop Router

The first hop router marks the incoming packets which match one (IP-address/netmask, port, protocol, dscp) tuple of its table to the specified `classpoint for outgoing traffic`. The appropriate MF classifier and marker are located in the `serv_handler`-module. This module gets a table as an option, which includes the MF-profile and the corresponding DSCP. Note, that this can also be used to remark packets, although not shown in the examples of table 2.

7.2 Ingress Router

The ingress router classifies the incoming packets and forwards them to the correct queueing disciplines. This is done by a MF-classifier, because at the ingress point each flow has to be handled separately as described in section 4. The classifier module `dsclsfr` has seven classes, four to handle the assured service classes and one each for premium service, routing traffic and best effort.

Assured service has to be handled by a meter, which measures the bandwidth of assured forwarding traffic and passes the results to a marker, which — if necessary — retags the packets to medium or high dropping precedence. Afterwards the traffic is stored in a TRIO queue. This is done by the `precedence_handler` and the `trio` modules. The settings for the TRIO queue are just an example and have to be adjusted to the amount of traffic and the router hardware.

In the ingress router premium service is handled by a shaper (the `premium_shaper` module, that needs a queue and a bandwidth regulation mechanism. This can be provided by the token bucket filter module `tbf`. Again the settings of the TBF , namely long- and short term burst size have to be adjusted individually.

As FIFO queues are the default traffic conditioning components attached to the classifier classes unless others are specified, we do not need to install any queueing disciplines at the sixth and seventh class (routing traffic and best effort traffic) as FIFO queues are sufficient for this kind of traffic.

For a complete overview of the configuration of a ingress router see also figure 9 and table 2.

7.3 Interior Router

Again the interior router has to classify incoming packets but it has the possibility of taking advantage of the traffic aggregation. Therefore the tables of the interior router only have set the `classpoint for incoming traffic` field and all other fields are wildcarded.

Within an interior router there is no need to perform any complex traffic conditioning functions but it may check, whether the bandwidth limitations are fulfilled. If not, it may drop some packets, which can be done by the `premium_policer` module for premium service or by the `trio` queues for assurred service. Again the FIFO queues for the last two classifier classes (see figure 10) can be omitted.

7.4 Egress Router

An egress router may perform traffic conditioning to meet the TCS between the downstream domain and itself. As this only affects traffic aggregations the classifier is again a BA-classifier (i.e. the profile in the table only contains a DSCP). The traffic conditioning is composed of a `premium_policer` for premium service and four `precedence_handlers` for the four assured service classes (see figure 11).

Table 2: Router Configuration Tables

First Hop Router (serv_handler.table)

#	srcaddr	srcmask	srcport	dstaddr	dstmask	dstport	proto	srcdscp	outdscp	lprio	mprio
	10.1.1.1	255.255.255.255	5005	10.1.4.2	255.255.255.255	5005	*	*	ps	0	0
	10.1.1.1	255.255.255.255	5030	10.1.4.2	255.255.255.255	5030	*	*	ps	0	0
	10.1.1.1	255.255.255.255	5105	10.1.4.2	255.255.255.255	5105	*	*	as1	0	0
	10.1.1.1	255.255.255.255	5110	10.1.4.2	255.255.255.255	5110	*	*	as1	0	0

Ingress Router (premium_shaper.table)

#	srcaddr	srcmask	srcport	dstaddr	dstmask	dstport	proto	srcdscp	outdscp	class	mprio
	10.1.1.1	255.255.255.255	5005	10.1.4.2	255.255.255.255	5005	*	ps	*	1	0
	10.1.1.1	255.255.255.255	5030	10.1.4.2	255.255.255.255	5030	*	ps	*	2	0

Ingress Router (precedence_handler.table)

#	srcaddr	srcmask	srcport	dstaddr	dstmask	dstport	proto	srcdscp	outdscp	lprio	mprio
	10.1.1.1	255.255.255.255	5105	10.1.4.2	255.255.255.255	5105	*	as1	*	625000	0
	10.1.1.1	255.255.255.255	5110	10.1.4.2	255.255.255.255	5110	*	as1	*	1250000	0

Interior and Egress Routers (premium_policer.table)

#	srcaddr	srcmask	srcport	dstaddr	dstmask	dstport	proto	srcdscp	outdscp	rate	mprio
	0.0.0.0	0.0.0.0	*	0.0.0.0	0.0.0.0	*	*	ps	*	4375000	0

Interior and Egress Routers (precedence_handler.table)

#	srcaddr	srcmask	srcport	dstaddr	dstmask	dstport	proto	srcdscp	outdscp	lprio	mprio
	0.0.0.0	0.0.0.0	*	0.0.0.0	0.0.0.0	*	*	as1	*	1875000	0

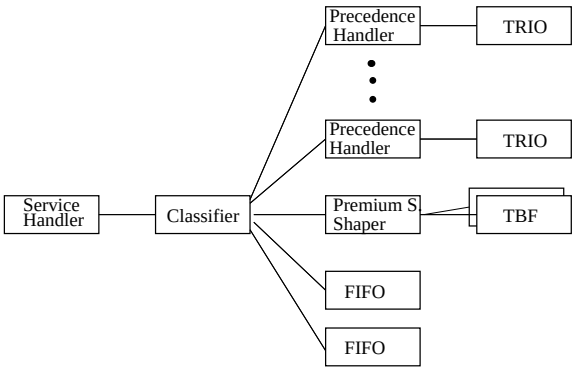

Figure 9: Configuration tree for a combined first hop / ingress router

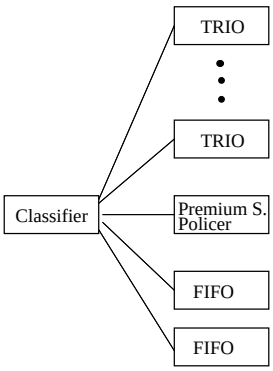

Figure 10: Configuration tree for interior routers

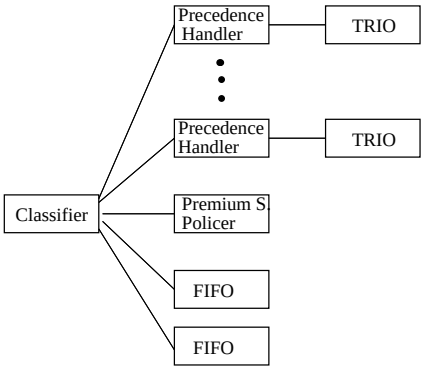

Figure 11: Configuration tree for egress routers

Figure 12: First Hop and Ingress Router Scripts

First Hop Router

```
#! /bin/bash
#
TC=/home/gast/tc/tc
TABLE0=/home/gast/diffserv/serv_handler.table
DEV0=eth1
#
dstab l 0 $TABLE0
#
$TC qdisc add dev $DEV0 root handle 1: serv_handler table_id 0
```

Ingress Router

```
#! /bin/bash
#
TC=/home/gast/tc/tc
TABLE0=/home/gast/diffserv/precedence_handler.table
TABLE1=/home/gast/diffserv/premium_shaper.table
DEV0=eth1
#
dstab l 0 $TABLE0
dstab l 1 $TABLE1
#
$TC qdisc add dev $DEV0 root handle 2: dsclsfr as1 0.4 as2 0.3 as3 0.1 as4 0.1 log_size 63
#
$TC qdisc add dev $DEV0 parent 2:1 handle 11: prec_handler table_id 0 dscp as1
$TC qdisc add dev $DEV0 parent 2:2 handle 12: prec_handler table_id 0 dscp as2
$TC qdisc add dev $DEV0 parent 2:3 handle 13: prec_handler table_id 0 dscp as3
$TC qdisc add dev $DEV0 parent 2:4 handle 14: prec_handler table_id 0 dscp as4
$TC qdisc add dev $DEV0 parent 2:5 handle 15: premium_shaper table_id 1 classes 2
#
$TC qdisc add dev $DEV0 parent 11:1 handle 101: trio limit 200 low_begin 0.8 low_end 1 medium_begin 0.4 medium_end 0.6 high_begin 0 high_end 0.2
$TC qdisc add dev $DEV0 parent 12:1 handle 102: trio limit 200 low_begin 0.8 low_end 1 medium_begin 0.4 medium_end 0.6 high_begin 0 high_end 0.2
$TC qdisc add dev $DEV0 parent 13:1 handle 103: trio limit 200 low_begin 0.8 low_end 1 medium_begin 0.4 medium_end 0.6 high_begin 0 high_end 0.2
$TC qdisc add dev $DEV0 parent 14:1 handle 104: trio limit 200 low_begin 0.8 low_end 1 medium_begin 0.4 medium_end 0.6 high_begin 0 high_end 0.2
$TC qdisc add dev $DEV0 parent 15:1 handle 105: tbf rate 5000kbit buffer 500kbit limit 500kbit
$TC qdisc add dev $DEV0 parent 15:2 handle 106: tbf rate 30000kbit buffer 3000kbit limit 3000kbit
```

Figure 13: Interior and Egress Router Scripts

Interior Router

```
#! /bin/bash
#
TC=/home/gast/tc/tc
TABLE1=/home/gast/diffserv/premium_policer.table
DEV0=eth1
#
dstab l 1 $TABLE1
#
$TC qdisc add dev $DEV0 root handle 2: dsclsfr as1 0.4 as2 0.3 as3 0.1 as4 0.1 log_size 63
#
$TC qdisc add dev $DEV0 parent 2:1 handle 11: trio limit 200 low_begin 0.8 low_end 1 medium_begin 0.4 medium_end 0.6 high_begin 0 high_end 0.2
$TC qdisc add dev $DEV0 parent 2:2 handle 12: trio limit 200 low_begin 0.8 low_end 1 medium_begin 0.4 medium_end 0.6 high_begin 0 high_end 0.2
$TC qdisc add dev $DEV0 parent 2:3 handle 13: trio limit 200 low_begin 0.8 low_end 1 medium_begin 0.4 medium_end 0.6 high_begin 0 high_end 0.2
$TC qdisc add dev $DEV0 parent 2:4 handle 14: trio limit 200 low_begin 0.8 low_end 1 medium_begin 0.4 medium_end 0.6 high_begin 0 high_end 0.2
$TC qdisc add dev $DEV0 parent 2:5 handle 15: premium_policer table_id 1
```

Egress Router

```
#! /bin/bash
#
TC=/home/gast/tc/tc
TABLE0=/home/gast/diffserv/precedence_handler.table
TABLE1=/home/gast/diffserv/premium_policer.table
DEV0=eth1
#
dstab l 0 $TABLE0
dstab l 1 $TABLE1
#
$TC qdisc add dev $DEV0 root handle 2: dsclsfr as1 0.4 as2 0.3 as3 0.1 as4 0.1 log_size 63
#
$TC qdisc add dev $DEV0 parent 2:1 handle 11: prec_handler table_id 0 dscp as1
$TC qdisc add dev $DEV0 parent 2:2 handle 12: prec_handler table_id 0 dscp as2
$TC qdisc add dev $DEV0 parent 2:3 handle 13: prec_handler table_id 0 dscp as3
$TC qdisc add dev $DEV0 parent 2:4 handle 14: prec_handler table_id 0 dscp as4
$TC qdisc add dev $DEV0 parent 2:5 handle 15: premium_policer table_id 1
#
$TC qdisc add dev $DEV0 parent 11:1 handle 101: trio limit 200 low_begin 0.8 low_end 1 medium_begin 0.4 medium_end 0.6 high_begin 0 high_end 0.2
$TC qdisc add dev $DEV0 parent 12:1 handle 102: trio limit 200 low_begin 0.8 low_end 1 medium_begin 0.4 medium_end 0.6 high_begin 0 high_end 0.2
$TC qdisc add dev $DEV0 parent 13:1 handle 103: trio limit 200 low_begin 0.8 low_end 1 medium_begin 0.4 medium_end 0.6 high_begin 0 high_end 0.2
$TC qdisc add dev $DEV0 parent 14:1 handle 104: trio limit 200 low_begin 0.8 low_end 1 medium_begin 0.4 medium_end 0.6 high_begin 0 high_end 0.2
```

References

[1] J. Heinanen et al. *Assured Forwarding PHB Group*, RFC 2597, June 1999

[2] V. Jacobson et al. *An Expedited Forwarding PHB*, RFC 2598, June 1999

[3] S. Blake at al. *An Architecture for Differentiated Services*, RFC 2475, December 1998

[4] K. Nichols et al. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*, RFC 2474, December 1998

[5] ftp://ftp.kernel.org/pub/linux/kernel/v2.2/linux-2.2.9.tar.gz

[6] ftp://ftp.sunet.se/pub/Linux/ip-routing/iproute2-2.2.4-now-ss990824.tar.gz

[7] ftp://sunsite.cnlab-switch.ch/ftp/mirror/Linux-HOWTO/Kernel-HOWTO.txt

[8] W. Almesberger *Linux Network Traffic Control — Implementation Overview*, ftp://lrcftp.epfl.ch/pub/people/almesber/pub/tcio-current.ps, April 1999

[9] S. Radhakrishnan *Linux - Advanced Networking Overwiew*, August 1999

[10] A.Kuznetsov, /usr/src/linux/net/sched/sch_tbf.c

[11] F. Baumgartner et al. *Differentiated Internet Services* in: G. Cooperman, E. Jessen, G. Michler (eds.): *Workshop on Wide Area Networks and High Performance Computing*, Lecture Notes in Control and Information Sciences 249, Springer, June 1999, pp. 37-60, ISBN 1-85233-642-0