Internet Engineering Task Force
Diffserv Working Group
INTERNET-DRAFT
Expires November 2000
draft-ietf-diffserv-model-03.txt

Y. Bernet
Microsoft
A. Smith
Extreme Networks
S. Blake
Ericsson
D. Grossman
Motorola

May 2000

A Conceptual Model for Diffserv Routers

Status of this Memo

Abstract

This draft proposes a conceptual model of Differentiated Services (Diffserv) routers for use in their management and configuration. This model defines the general functional datapath elements (classifiers, meters, markers, droppers, monitors, multiplexors, queues), their possible configuration parameters, and how they might be interconnected to realize the range of classification, traffic conditioning, and per-hop behavior (PHB) functionalities described in [DSARCH]. The model is

intended to be abstract and capable of representing the configuration parameters important to Diffserv functionality for a variety of specific router implementations. It is not intended as a guide to hardware implementation.

This model serves as the rationale for the design of an SNMP MIB [DSMIB] and for other configuration interfaces (e.g. [DSPIB]) and more detailed models (e.g. [QOSDEVMOD]): these should all be based upon and consistent with this model.

1. Introduction

Differentiated Services (Diffserv) [DSARCH] is a set of technologies which allow network service providers to offer different kinds of network quality-of-service (QoS) to different customers and their traffic streams. The premise of Diffserv networks is that routers within the core of the network handle packets in different traffic streams by forwarding them using different per-hop behaviors (PHBs). The PHB to be applied is indicated by a Diffserv codepoint (DSCP) in the IP header of each packet [DSFIELD]. Note that this document uses the terminology defined in [DSARCH, DSTERMS] and in Section 2.

The advantage of such a scheme is that many traffic streams can be aggregated to one of a small number of behavior aggregates (BA) which are each forwarded using the same PHB at the router, thereby simplifying the processing and associated storage. In addition, there is no signaling, other than what is carried in the DSCP of each packet, and no other related processing that is required in the core of the Diffserv network since QoS is invoked on a packet-by- packet basis.

The Diffserv architecture enables a variety of possible services which could be deployed in a network. These services are reflected to customers at the edges of the Diffserv network in the form of a Service Level Specification (SLS) [DSTERMS]. The ability to provide these services depends on the availability of cohesive management and configuration tools that can be used to provision and monitor a set of Diffserv routers in a coordinated manner. To facilitate the development of such configuration and management tools it is helpful to define a conceptual model of a Diffserv router that abstracts away implementation details of particular Diffserv routers from the parameters of interest for configuration and management. The purpose of this draft is to define such a model.

The basic forwarding functionality of a Diffserv router is defined in other specifications; e.g., [DSARCH, DSFIELD, AF-PHB, EF-PHB].

This document is not intended in any way to constrain or to dictate the implementation alternatives of Diffserv routers. It is expected that router implementers will demonstrate a great deal of variability in their implementations. To the extent that implementers are able to model their implementations using the abstractions described in this draft, configuration and management tools will more readily be able to configure and manage networks incorporating Diffserv routers of assorted origins.

- Section 3 starts by describing the basic high-level functional elements of a Diffserv router and then describe the various components, then focussing on the Diffserv-specific components of the router and a hierarchical management model for these components.

- Section 4 describes classification elements.

- Section 5 discusses meter elements.

- Section 6 discusses action elements.

- Section 7 discusses the basic queueing elements and their functional behaviors (e.g. shaping).

- Section 8 shows how the basic classification, meter, action and queueing elements can be combined to build modules called Traffic Conditioning Blocks (TCBs).

- Section 9 discusses open issues with this document

- Section 10 discusses security concerns.

2. Glossary

This memo uses terminology which is defined in [DSARCH] and in [DSTERMS]. Some of the terms defined there are defined again here in order to provide additional detail, along with some new terms specific to this document.

Classifier — A functional datapath element which consists of filters which select packets based on the content of packet headers or other packet data, and/or on implicit or derived attributes associated with the packet, and forwards the packet along a particular datapath within the router. A classifier splits a single incoming traffic stream into multiple outgoing ones.

Counter	A functional datapath element which updates a packet counter and also an octet counter for every packet that passes through it. Used for collecting statistics.
Filter	A set of wildcard, prefix, masked, range and/or exact match conditions on the components of a packet's classification key. A filter is said to match only if each condition is satisfied.
Multiplexer (Mux)	A functional datapath element that merges multiple traffic streams (datapaths) into a single traffic stream (datapath).
Non-work-conserving	A property of a scheduling algorithm such that it services packets no sooner than a scheduled departure time, even if this means leaving packets in a FIFO while the link is idle.
Queueing Block	A combination of functional datapath elements that modulates the transmission of packets belonging to a traffic streams and determines their ordering, possibly storing them temporarily or discarding them.
Scheduling algorithm	An algorithm which determines which queue of a set of qyeyes to service next. This may be based on the relative priority of the queues, on a weighted fair bandwidth sharing policy or some other policy. Such an algorithm may be either work-conserving or non-work-conserving.
Shaping	The process of delaying packets within a traffic stream to cause it to conform to some defined traffic profile. Shaping can be implemented using a queue serviced by a non-work-conserving scheduling algorithm.
Traffic Conditioning Block (TCB)	A logical datapath entity consisting of a number of other functional datapath entities interconnected in such a way as to perform a specific set of traffic conditioning functions on an incoming traffic stream. A TCB can be thought of as an entity with one input and one output and a set of control parameters.
Work-conserving	A property of a scheduling algorithm such that it servicess a packet, if one is available, at every transmission opportunity."

3. Conceptual Model

This section introduces a block diagram of a Diffserv router and describes the various components illustrated. Note that a Diffserv core router is assumed to include only a subset of these components: the model presented here is intended to cover the case of both Diffserv edge and core routers.

3.1. Elements of a Diffserv Router

The conceptual model includes abstract definitions for the following:

- Traffic Classification elements.
- Metering functions.
- Traffic Conditioning (TC) actions of Marking, Absolute Dropping, Counting and Multiplexing.
- Queueing elements, including capabilities of algorithmic dropping.
- Certain combinations of traffic classification, traffic conditioning and queueing elements.

The components and combinations of components described in this document form building blocks that need to be manageable by Diffserv configuration and management tools. One of the goals of this document is to show how a model of a Diffserv device can be built using these component blocks. This model is in the form of a connected directed acyclic graph (DAG) of functional datapath elements that describes the traffic conditioning and queueing behaviors that any particular packet will experience when forwarded to the Diffserv router.

The following diagram illustrates the major functional blocks of a Diffserv router:

3.1.1. Datapath

An ingress interface, routing core and egress interface are illustrated at the center of the diagram. In actual router implementations, there may be an arbitrary number of ingress and egress interfaces interconnected by the routing core. The routing core element serves as an abstraction of a router's normal routing and switching functionality. The routing core moves packets between interfaces according to policies outside the scope of Diffserv. The actual queueing delay and packet loss

Figure 1: Diffserv Router Major Functional Blocks

behavior of a specific router's switching fabric/backplane is not modeled by the routing core; these should be modeled using the functional elements described later. The routing core should be thought of as an infinite bandwidth, zero- delay backplane connecting ingress and egress interfaces.

The components of interest on the ingress/egress interfaces are the traffic classifiers, traffic conditioning (TC) components, and the queueing components that support Diffserv traffic conditioning and per-hop behaviors [DSARCH]. These are the fundamental components comprising a Diffserv router and will be the focal point of our conceptual model.

3.1.2. Configuration and Management Interface

Diffserv operating parameters are monitored and provisioned through this interface. Monitored parameters include statistics regarding traffic carried at various Diffserv service levels. These statistics may be important for accounting purposes and/or for tracking compliance to Traffic Conditioning Specifications (TCSs) [DSTERMS] negotiated with

customers. Provisioned parameters are primarily classification rules, TC and PHB configuration parameters. The network administrator interacts with the Diffserv configuration and management interface via one or more management protocols, such as SNMP or COPS, or through other router configuration tools such as serial terminal or telnet consoles.

Specific policy objectives are presumed to be installed by or retrieved from policy management mechanisms. However, diffserv routers are subject to implementation decisions which form a meta- policy that scopes the kinds of policies which can be created.

3.1.3. Optional QoS Agent Module

Diffserv routers may snoop or participate in either per-microflow or per-flow-aggregate signaling of QoS requirements [E2E] e.g. using the RSVP protocol. Snooping of RSVP messages may be used, for example, to learn how to classify traffic without actually participating as a RSVP protocol peer. Diffserv routers may reject or admit RSVP reservation requests to provide a means of admission control to Diffserv-based services or they may use these requests to trigger provisioning changes for a flow-aggregation in the Diffserv network. A flow-aggregation in this context might be equivalent to a Diffserv BA or it may be more fine-grained, relying on a MF classifier [DSARCH]. Note that the conceptual model of such a router implements the Integrated Services Model as described in [INTSERV], applying the control plane controls to the data classified and conditioned in the data plane, as desribed in [E2E].

Note that a QoS Agent component of a Diffserv router, if present, might be active only in the control plane and not in the data plane. In this scenario, RSVP could be used merely to signal reservation state without installing any actual reservations in the data plane of the Diffserv router: the data plane could still act purely on Diffserv DSCPs and provide PHBs for handling data traffic without the normal per-microflow handling expected to support some Intserv services.

3.2. Hierarchical Model of Diffserv Components

This document focuses on the Diffserv-specific components of the router: classification, traffic conditioning and queueing functions. Figure 2 shows a high-level view of ingress and egress interfaces of a router. The diagram illustrates two Diffserv router interfaces, each having an ingress and an egress component. It shows classification, meter, action and queueing elements which might be instantiated on each interface's ingress and egress component. The TC functionality is implemented by a combination of classification, action, meter and queueing elements.

In principle, if one were to construct a network entirely out of two-port routers (in appropriate places connected by LANs or similar media), then it would be necessary for each router to perform four QoS control functions in the datapath on traffic in each direction:

- Classify each message according to some set of rules.

- If necessary, determine whether the data stream the message is part of is within or outside its rate by metering the stream.

- Perform a set of resulting actions, including applying a drop policy appropriate to the classification and queue in question and perhaps additionally marking the traffic with a Differentiated Services Code Point (DSCP) as defined in [DSCP].

- Enqueue the traffic for output in the appropriate queue, which may either shape the traffic or simply forward it with some minimum rate or maximum latency.

If the network is now built out of N-port routers, the expected behavior of the network should be identical. Therefore, this model must provide for essentially the same set of functions on the ingress as on the egress port of the router. Some interfaces will be "edge" interfaces and some will be "interior" to the Differentiated Services domain. The one point of difference between an ingress and an egress interface is that

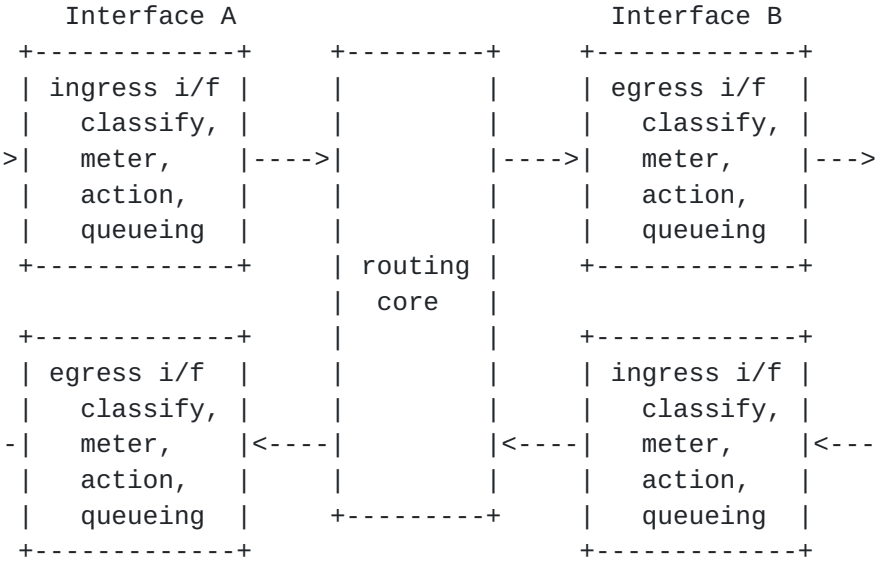

Figure 2. Traffic Conditioning and Queueing Elements

all traffic on an egress interface is queued, while traffic on an ingress interface will typically be queued only for shaping purposes, if at all. Therefore, equivalent functional elements are modelled on both the ingress and egress components of an interface.

Note that it is not mandatory that each of these functional elements be implemented on both ingress and egress components; equally, the model allows that multiple sets of these elements may be placed in series and/or in parallel at ingress or at egress. The arrangement of elements is dependent on the service requirements on a particular interface on a particular router. By modelling these elements on both ingress and egress components, it is not implied that they must be implemented in this way in a specific router. For example, a router may implement all shaping and PHB queueing on the interface egress component or may instead implement it only on the ingress component. Furthermore, the classification needed to map a packet to an egress component queue (if present) need not be implemented on the egress component but instead may be implemented on the ingress component, with the packet passed through the routing core with in-band control information to allow for egress queue selection.

>From a device-configuration and management perspective, the following hierarchy exists:

> At the top level, the network administrator manages interfaces. Each interface consists of an ingress component and an egress component. Each component may contain classifier, action, meter and queueing elements.
>
> At the next level, the network administrator manages groups of functional elements interconnected in a DAG. These elements are organized in self-contained Traffic Conditioning Blocks (TCBs) which are used to implement some desired network policy (see Section 8). One or more TCBs may be instantiated on each ingress or egress component; they may be connected in series and/or in parallel configurations on the multiple outputs of a classifier. The TCB is defined optionally to include classification and queueing elements so as to allow for flexible functionality. A TCB can be thought of as a "black box" with one input and one output in the data path. Each interface (ingress or egress) may have different TCB configurations.
>
> At the lowest level are individual functional elements, each with their own configuration parameters and management counters and flags.

4. Classifiers

4.1. Definition

Classification is performed by a classifier element. Classifiers are 1:N (fan-out) devices: they take a single traffic stream as input and generate N logically separate traffic streams as output. Classifiers are parameterized by filters and output streams. Packets from the input stream are sorted into various output streams by filters which match the contents of the packet or possibly match other attributes associated with the packet. Various types of classifiers are described in the following sections.

We use the following diagram to illustrate a classifier, where the outputs connect to succeeding functional elements:

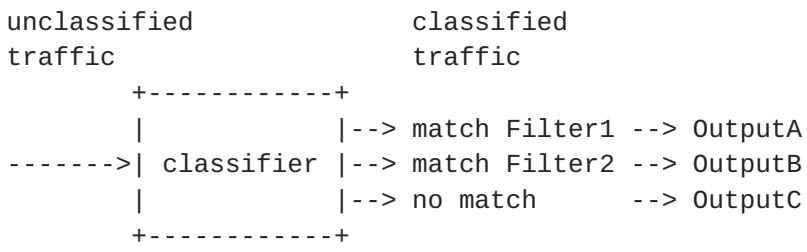

Figure 3. An Example Classifier

Note that we allow a multiplexor (see Section 6.5) before the classifier to allow input from multiple traffic streams. For example, if multiple ingress sub-interfaces feed through a single classifier then the interface number can be considered by the classifier as a packet attribute and be included in the packet's classification key. This optimization may be important for scalability in the management plane. Another example of a packet attribute could be an integer representing the BGP community string associated with the packet's best-matching route.

The following classifier separates traffic into one of three output streams based on three filters:

Filter Matched	Output Stream
Filter1	A
Filter2	B
Filter3 (no match)	C

Where Filters1 and Filter2 are defined to be the following BA filters ([DSARCH], Section 4.2.1):

Filter	DSCP
1	101010
2	111111
3	****** (wildcard)

4.1.1. Filters

A filter consists of a set of conditions on the component values of a packet's classification key (the header values, contents, and attributes relevant for classification). In the BA classifier example above, the classification key consists of one packet header field, the DSCP, and both Filter1 and Filter2 specify exact-match conditions on the value of the DSCP. Filter3 is a wildcard default filter which matches every packet, but which is only selected in the event that no other more specific filter matches.

In general there are a set of possible component conditions including exact, prefix, range, masked, and wildcard matches. Note that ranges can be represented (with less efficiency) as a set of prefixes and that prefix matches are just a special case of both masked and range matches.

In the case of a MF classifier [DSARCH], the classification key consists of a number of packet header fields. The filter may specify a different condition for each key component, as illustrated in the example below for a IPv4/TCP classifier:

Filter	IP Src Addr	IP Dest Addr	TCP SrcPort	TCP DestPort
Filter4	172.31.8.1/32	172.31.3.X/24	X	5003

In this example, the fourth octet of the destination IPv4 address and the source TCP port are wildcard or "don't cares".

MF classification of fragmented packets is impossible if the filter uses transport-layer port numbers e.g. TCP port numbers. MTU-discovery is therefore a prerequisite for proper operation of a Diffserv network that uses such classifiers.

4.1.2. Overlapping Filters

Note that it is easy to define sets of overlapping filters in a classifier. For example:

```
Filter5:
Type:   Masked-DSCP
Value:  111000
```

```
      Mask:   111000

      Filter6:
      Type:   Masked-DSCP
      Value:  000111 (binary)
      Mask:   000111 (binary)
```

A packet containing DSCP = 111111 cannot be uniquely classified by this pair of filters and so a precedence must be established between Filter5 and Filter6 in order to break the tie. This precedence must be established either (a) by a manager which knows that the router can accomplish this particular ordering e.g. by means of reported capabilities, or (b) by the router along with a mechanism to report to a manager which precedence is being used. These ordering mechanisms must be supported by the configuration and management protocols although further discussion of this is outside the scope of this document.

As another example, one might want first to disallow certain applications from using the network at all, or to classify some individual traffic streams that are not Diffserv-marked. Traffic that is not classified by those tests might then be inspected for a DSCP. The word "then" implies sequence and this must be specified by means of precedence.

An unambiguous classifier requires that every possible classification key match at least one filter (possibly the wildcard default) and that any ambiguity between overlapping filters be resolved by precedence. Therefore, the classifiers on any given interface must be "complete" and will often include an "everything else" filter as the lowest precedence element in order for the result of classification to be deterministic. Note that this completeness is only required of the first classifier that incoming traffic will meet as it enters an interface - subsequent classifiers on an interface only need to handle the traffic that it is known that they will receive.

4.2. Examples

4.2.1. Behaviour Aggregate (BA) Classifier

The simplest Diffserv classifier is a behavior aggregate (BA) classifier [DSARCH]. A BA classifier uses only the Diffserv codepoint (DSCP) in a packet's IP header to determine the logical output stream to which the packet should be directed. We allow only an exact-match condition on this field because the assigned DSCP values have no structure, and therefore no subset of DSCP bits are significant.

The following defines a possible BA filter:

```
Filter8:
Type:   BA
Value:  111000
```

4.2.2. Multi-Field (MF) Classifier

Another type of classifier is a multi-field (MF) classifier [DSARCH]. This classifies packets based on one or more fields in the packet (possibly including the DSCP). A common type of MF classifier is a 6-tuple classifier that classifies based on six fields from the IP and TCP or UDP headers (destination address, source address, IP protocol, source port, destination port, and DSCP). MF classifiers may classify on other fields such as MAC addresses, VLAN tags, link-layer traffic class fields or other higher-layer protocol fields.

The following defines a possible MF filter:

```
Filter9:
Type:              IPv4-6-tuple
IPv4DestAddrValue: 0.0.0.0
IPv4DestAddrMask:  0.0.0.0
IPv4SrcAddrValue:  172.31.8.0
IPv4SrcAddrMask:   255.255.255.0
IPv4DSCP:          28
IPv4Protocol:      6
IPv4DestL4PortMin: 0
IPv4DestL4PortMax: 65535
IPv4SrcL4PortMin:  20
IPv4SrcL4PortMax:  20
```

A similar type of classifier can be defined for IPv6.

4.2.3. Free-form Classifier

A Free-form classifier is made up of a set of user definable arbitrary filters each made up of {bit-field size, offset (from head of packet), mask}:

```
Classifier2:
Filter12:    OutputA
Filter13:    OutputB
Default:     OutputC

Filter12:
Type:        FreeForm
```

```
      SizeBits:    3 (bits)
      Offset:      16 (bytes)
      Value:       100 (binary)
      Mask:        101 (binary)

      Filter13:
      Type:        FreeForm
      SizeBits:    12 (bits)
      Offset:      16 (bytes)
      Value:       100100000000 (binary)
      Mask:        111111111111 (binary)
```

Free-form filters can be combined into filter groups to form very powerful filters.

4.2.4. Other Possible Classifiers

Classification may also be performed based on information at the datalink layer below IP (e.g. VLAN or datalink-layer priority) or perhaps on the ingress or egress IP, logical or physical interface identifier. (e.g. the incoming channel number on a channelized interface). A classifier that filters based on IEEE 802.1p Priority and on 802.1Q VLAN-ID might be represented as:

```
      Classifier3:
      Filter14 AND Filter15:  OutputA
      Default:                OutputB

      Filter14:                        -- priority 4 or 5
      Type:        Ieee8021pPriority
      Value:       100 (binary)
      Mask:        110 (binary)

      Filter15:                        -- VLAN 2304
      Type:        Ieee8021QVlan
      Value:       100100000000 (binary)
      Mask:        111111111111 (binary)
```

Such classifiers may be subject of other standards or may be enterprise-specific but are not discussed further here.

5. Meters

Metering is is defined in [DSARCH]. Diffserv network providers may choose to offer services to customers based on a temporal (i.e., rate) profile within which the customer submits traffic for the service. In

this event, a meter might be used to trigger real-time traffic conditioning actions (e.g., marking) by routing a non-conforming packet through an appropriate next-stage action element. Alternatively, it might also be used for out-of-band management functions like statistics monitoring for billing applications.

Meters are logically 1:N (fan-out) devices (although a multiplexor can be used in front of a meter). Meters are parameterized by a temporal profile and by conformance levels, each of which is associated with a meter's output. Each output can be connected to another functional element.

Note that this model of a meter differs slightly from that described in [DSARCH]. In that description the meter is not a datapath element but is instead used to monitor the traffic stream and send control signals to action elements to dynamically modulate their behavior based on the conformance of the packet.

The following diagram illustrates a meter with 3 levels of conformance:

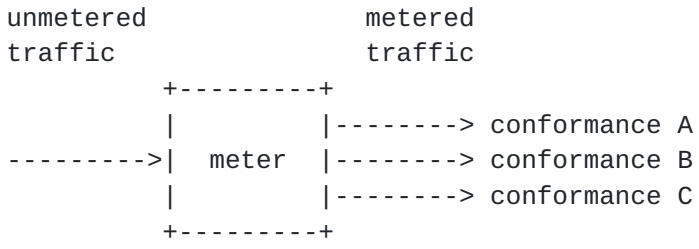

Figure 4. A Generic Meter

In some Diffserv examples, three levels of conformance are discussed in terms of colors, with green representing conforming, yellow representing partially conforming and red representing non-conforming [AF-PHB]. These different conformance levels may be used to trigger different queueing, marking or dropping treatment later on in the processing. Other example meters use a binary notion of conformance; in the general case N levels of conformance can be supported. In general there is no constraint on the type of functional element following a meter output, but care must be taken not to inadvertently configure a datapath that results in packet reordering within an OA.

A meter, according to this model, measures the rate at which packets making up a stream of traffic pass it, compares the rate to some set of thresholds and produces some number (two or more) potential results: a given packet is said to "conform" to the meter if, at the time that the packet is being looked at, the stream appears to be within the meter's limit rate.

The concept of conformance to a meter bears comment. The concept applied in several rate-control architectures, including ATM, Frame Relay, Integrated Services and Differentiated Services, is variously described as a "leaky bucket" or a "token bucket".

A leaky bucket algorithm is primarily used for traffic shaping (handled under Queues and Schedulers in this model): traffic theoretically departs from a device at a rate of one bit every so many time units but, in fact, departs in multi-bit units (packets) at a rate approximating that. It is also possible to build multi-rate leaky buckets, in which traffic departs from the switch at varying rates depending on recent activity or inactivity.

A simple token bucket is usually used in a Meter to measure the behavior of a peer's leaky bucket, for verification purposes. It is, by definition, a relationship between some defined burst_size, rate and interval:

```
      interval = burst_size/rate
   or
      rate = burst_size/interval
```

Multi-rate token buckets (token buckets with both a peak and a mean rate and sometimes more rates) are commonly used. In this case, the burst size for the baseline traffic is conventionally referred to as the "committed burst" and the time interval is as specified by

```
      interval = committed_burst/mean_rate
```

but additional burst sizes (each an increment over its predecessor) are defined, which are conventionally referred to as "excess" burst sizes. The peak rate therefore equals the sum of the burst sizes for any given interval.

A data stream is said to conform to a simple token bucket if the switch receives at most the "burst_size" of data in any time interval of length "interval". In the multi-rate case, the traffic is said to conform at a given level to the token bucket at if its rate does not exceed the sum of the relevant burst sizes in any given interval. Received traffic that arrives pre-classified as one of the "excess" rates (e.g. AF12 or AF13 traffic for a device implementing the AF1x PHB) is only compared to the relevant excess buckets.

<ed: the following paragraphs may need fixing when we can all agree on a stricter vs. looser definition: for now we assume strict schedulers and lenient meters.>

The fact that data is organized into variable length packets introduces some uncertainty in this conformance decision. When used in a Scheduler, a leaky bucket releases a packet only when all of its bits would have been allowed: it does not borrow from future capacity. When used in a Meter, a token bucket accepts a packet if any of its bits would have been accepted and "borrows" any excess capacity required from that allotted to equivalently classified traffic in a previous or subsequent interval. Note that [SRTCM] and [TRTCM] insist on stricter behaviour from a meter than the model here insists on.

Multiple classes of traffic, as identified by the classifier table, may be presented to the same meter. Imagine, for example, that it is desired to drop all traffic that uses any DSCP that has not been publicly defined. A classifier entry might exist for each such DSCP, shunting it to an "accepts everything" meter and dropping all traffic that conforms to only that meter.

It is necessary to identify what is to be done with packets that conform to the meter and with packets that do not. It is also necessary for the meter to be arbitrarily extensible as some PHBs require the successive application of an arbitrary number of meters. The approach taken in this model is to have each meter indicate what action is to be taken for conforming traffic and what meter is to be used for traffic which fails to conform. With the definition of a special type of meter to which all traffic conforms, this has the necessary flexibility.

Note that this definition of a simple token bucket meter requires that the minimal bucket size be at least the MTU of the incoming link and it should also be initialised with sufficient tokens to allow for at least one MTU-sized packet to conform if it arrives at time zero.

5.1. Examples

The following are some examples of possible meters.

5.1.1. Average Rate Meter

An example of a very simple meter is an average rate meter. This type of meter measures the average rate at which packets are submitted to it over a specified averaging time.

An average rate profile may take the following form:

```
Meter1:
Type:               AverageRate
Profile:            Profile1
ConformingOutput:   Queue1
```

```
NonConformingOutput: Counter1

Profile1:
Type:               AverageRate
AverageRate:        120 kbps
Delta:              100 msec
```

A meter measuring against this profile would continually maintain a count that indicates the total number of packets arriving between time T (now) and time T - 100 msecs. So long as an arriving packet does not push the count over 12 kbits in the last 100 msec then the packet would be deemed conforming. Any packet that pushes the count over 12 kbits would be deemed non-conforming. Thus, this meter deems packets to correspond to one of two conformance levels: conforming or non-conforming and sends them on for the appropriate subsequent treatment.

5.1.2. Exponential Weighted Moving Average (EWMA) Meter

The EWMA form of meter is easy to implement in hardware and can be parameterized as follows:

```
avg_rate(t) = (1 - Gain) * avg_rate(t') +  Gain * rate(t)
t = t' + Delta
```

For a packet arriving at time t:

```
if (avg_rate(t) > AverageRate)
   non-conforming
else
   conforming
```

"Gain" controls the time constant (e.g. frequency response) of what is essentially a simple IIR low-pass filter. "rate(t)" measures the number of incoming bytes in a small fixed sampling interval, Delta. Any packet that arrives and pushes the average rate over a predefined rate AverageRate is deemed non-conforming. An EWMA meter profile might look something like the following:

```
Meter2:
Type:               ExpWeightedMovingAvg
Profile:            Profile2
ConformingOutput:   Queue1
NonConformingOutput: AbsoluteDropper1

Profile2:
Type:               ExpWeightedMovingAvg
AverageRate:        25 kbps
```

```
Delta:              10 usec
Gain:               1/16
```

5.1.3. Two-Parameter Token Bucket Meter

A more sophisticated meter might measure conformance to a token bucket (TB) profile. A TB profile generally has two parameters, an average token rate and a burst size. TB meters compare the arrival rate of packets to the average rate specified by the TB profile. Logically, tokens accumulate in a bucket at the average rate, up to a maximum credit which is the burst size. Packets of length L bytes are considered conforming if any tokens are available in the bucket at the time of packet arrival: up to L bytes may then be borrowed from future token allocations. Packets are allowed to exceed the average rate in bursts up to the burst size. Packets which arrive to find a bucket with no tokens in it are deemed non-conforming. A two-parameter TB meter has exactly two possible conformance levels (conforming, non-conforming). TB implementation details are discussed in Appendix A. Note that this is a "lenient" meter that allows some borrowing, as discussed above.

A two-parameter TB meter might appear as follows:

```
Meter3:
Type:               SimpleTokenBucket
Profile:            Profile3
ConformingOutput:   Queue1
NonConformingOutput: AbsoluteDropper1

Profile3:
Type:               SimpleTokenBucket
AverageRate:        200 kbps
BurstSize:          100 kbytes
```

5.1.4. Multi-Stage Token Bucket Meter

More complicated TB meters might define two burst sizes and three conformance levels. Packets found to exceed the larger burst size are deemed non-conforming. Packets found to exceed the smaller burst size are deemed partially conforming. Packets exceeding neither are deemed conforming. Token bucket meters designed for Diffserv networks are described in more detail in [SRTCM, TRTCM, GTC]; in some of these references three levels of conformance are discussed in terms of colors, with green representing conforming, yellow representing partially conforming and red representing non- conforming. Often these multi-conformance level meters can be implemented using an appropriate configuration of multiple two- parameter TB meters.

A profile for a multi-stage TB meter with three levels of conformance might look as follows:

```
Meter4:
Type:                TwoRateTokenBucket
ProfileA:            Profile4
ConformingOutputA:   Queue1
ProfileB:            Profile5
ConformingOutputB:   Marker1
NonConformingOutput: AbsoluteDropper1

Profile4:
Type:                SimpleTokenBucket
AverageRate:         100 kbps
BurstSize:           20 kbytes

Profile5:
Type:                SimpleTokenBucket
AverageRate:         100 kbps
BurstSize:           100 kbytes
```

5.1.5. Null Meter

A null meter has only one output: always conforming, and no associated temporal profile. Such a meter is useful to define in the event that the configuration or management interface does not have the flexibility to omit a meter in a datapath segment.

```
Meter5:
Type:                NullMeter
Output:              Queue1
```

6. Action Elements

The classifiers and meters described up to this point are fan-out elements which are generally used to determine the appropriate action to apply to a packet. The set of possible actions include:

- Marking

- Absolute Dropping

- Multiplexing

- Counting

- Null action - do nothing

The corresponding action elements are described in the following sections.

Diffserv nodes may apply shaping, policing and/or marking to traffic streams that exceed the bounds of their TCS in order to prevent a traffic stream from seizing more than its share of resources from a Diffserv network. Shaping, sometimes considered as a TC action, is treated as a part of the queueing module in this model, as is the use of Algorithmic Dropping techniques - see section 7. Policing is modelled as the combination of either a meter or a scheduler with either an absolute dropper or an algorithmic dropper. These elements will discard packets which exceed the TCS. Marking is performed by a marker, which (in this context) alters the DSCP, and thus the PHB, of the packet to give it a lower-grade treatment at subsequent Diffserv nodes.

6.1. Marker

Markers are 1:1 elements which set a codepoint (e.g. the DSCP in an IP header). Markers may also act on unmarked packets (e.g. those submitted with DSCP of zero) or may re-mark previously marked packets. In particular, the model supports the application of marking based on a preceding classifier match. The mark set in a packet will determine its subsequent treatment in downstream nodes of a network and possibly also in subsequent processing stages within this router.

DSCP Markers for Diffserv are normally parameterized by a single parameter: the 6-bit DSCP to be marked in the packet header.

```
Marker1:
Type:               DSCPMarker
Mark:               010010
```

6.2. Absolute Dropper

Absolute droppers simply discard packets. There are no parameters for these droppers. Because this dropper is a terminating point of the datapath and have no outputs, it is probably desirable to forward the packet through a counter action first for instrumentation purposes.

```
AbsoluteDropper1:
Type:               AbsoluteDropper
```

Absolute droppers are not the only elements than can cause a packet to be discarded: another element is an Algorithmic Dropper element (see Section 6.6). However, since this element's behavior is closely tied the

state of one or more queues, we choose to distinguish it as a separate functional element.

6.3. Multiplexer

It is occasionally necessary to multiplex traffic streams into a 1:1 or 1:N action element or classifier. A M:1 (fan-in) multiplexer is a simple logical device for merging traffic streams. It is parameterized by its number of incoming ports.

```
Mux1:
Type:               Multiplexer
Output:             Queue2
```

6.4. Counter

One passive action is to account for the fact that a data packet was processed. The statistics that result might be used later for customer billing, service verification, or network engineering purposes. Counters are 1:1 functional elements which update a counter by L and a packet counter by 1 every time a L-byte sized packet passes through them. Counters can be used to count packets about to be be dropped by a dropper or a queueing element.

```
Counter1:
Type:               Counter
Output:             Queue1
```

6.5. Null Action

A null action has one input and one output. The element performs no action on the packet. Such an element is useful to define in the event that the configuration or management interface does not have the flexibility to omit an action element in a datapath segment.

```
Null1:
Type:               Null
Output:             Queue1
```

7. Queueing Blocks

Queueing blocks modulate the transmission of packets belonging to the different traffic streams and determine their ordering, possibly storing them temporarily or discarding them. Packets are usually stored either because there is a resource constraint (e.g., available bandwidth) which prevents immediate forwarding, or because the queueing block is being

used to alter the temporal properties of a traffic stream (i.e. shaping). Packets are discarded either because of buffering limitations, because a buffer threshold is exceeded (including when shaping is performed), as a feedback control signal to reactive control protocols such as TCP, because a meter exceeds a configured rate (i.e. policing).

The queueing block in this model is a logical abstraction of a queueing system, which is used to configure PHB-related parameters. There is no conformance to this model. The model can be used to represent a broad variety of possible implementations. However, it need not necessarily map one-to-one with physical queueing systems in a specific router implementation. Implementors should map the configurable parameters of the implementation's queueing systems to these queueing block parameters as appropriate to achieve equivalent behaviors.

7.1. Queueing Model

Queueing is a function a which lends itself to innovation. It must be modelled to allow a broad range of possible implementations to be represented using common structures and parameters. This model uses functional decomposition as a tool to permit the needed lattitude.

Queueing sytems, such as the queueing block defined in this model, perform three distinct, but related, functions: they store packets, they modulate the departure of packets belonging to various traffic streams and they selectively discard packets. This model decomposes the queueing block into the component elements that perform each of these functions. These elements which may be connected together either dynamically or statically to construct queueing blocks. A queueing block is thus composed of of one or more FIFOs, one or more Schedulers and zero or more Algorithmic Droppers.

> <ed: should this be *one* or more? There are valid cases that do not require a dropper but they are exceptional.>

Note that the term FIFO has multiple different common usages: it is sometimes taken to mean, among other things, a data structure that permits items to be removed only in the order in which they were inserted or a service discipline which is non- reordering.

7.1.1. FIFO

In this model, a FIFO element is a data structure which at any time may contain zero or more packets. It may have one or more thresholds associated with it. A FIFO has one or more inputs and exactly one output. It must support an enqueue operation to add a packet to the tail of the queue, and a dequeue operation to remove a packet from the head

of the queue. Packets must be dequeued in the order in which they were enqueued. A FIFO has a current depth, which indicates the number of packets that it contains at a particular time. FIFOs in this model are modelled without inherent limits on their depth - obviously this does not reflect the reality of implementations: FIFO size limits are modelled here by an algorithmic dropper associated with the FIFO, typically at its input. It is quite likely that, every FIFO will be preceded by an algorithmic dropper. One exception might be the case where the packet stream has already been policed to a profile that can never exceed the scheduler bandwidth available at the FIFO's output - this would not need an algorithmic dropper at the input to the FIFO.

This representation of a FIFO allows for one common type of depth limit, one that results from a FIFO supplied from a limited pool of buffers, shared between multiple FIFOs.

> <ed: should we instead model a FIFO as having a single input and use a "multiplexer" at its input if it needs to collect from multiple input sources?>

Typically, the FIFO element of this model will be implemented as a FIFO data structure. However, this does not preclude implementations which are not strictly FIFO, in that they also support operations that remove or examine packets (e.g., for use by discarders) other than at the head or tail. However, such operations MUST NOT have the effect of reordering packets belonging to the same microflow.

In an implementation, packets are presumably stored in one or more buffers. Buffers are allocated from one or more free buffer pools. If there are multiple instances of a FIFO, their packet buffers may or may not be allocated out of the same free buffer pool. Free buffer pools may also have one or more threshold associated with them, which may affect discarding and/or scheduling. Other than this, buffering mechanisms are implementation specific and not part of this model.

A FIFO might be represented using the following parameters:

```
Fifo1:
Type:       FIFO
Output:     Scheduler1
```

Note that a FIFO must provide triggers and/or current state information to other elements upstream and downstream from it: in particular, it is likely that the current depth will need to be used by Algorithmic Dropper elements placed before or after the FIFO. It will also likely need to provide an implicit "I have packets for you" signal to downstream Scheduler elements.

7.1.2. Scheduler

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

The service discipline (also known as a scheduling algorithm) is an algorithm which might take any of the following as its input(s):

a) static parameters such as relative priority associated with each of the scheduler's inputs.

b) absolute token bucket parameters for maximum or minimum rates associated with each of the scheduler's inputs.

c) parameters, such as packet length or DSCP, associated with the packet currently present at its input.

d) absolute time and/or local state.

Possible service disciplines fall into a number of categories, including (but not limited to) first come, first served (FCFS), strict priority, weighted fair bandwidth sharing (e.g., WFQ, WRR, etc.), rate-limited strict priority and rate-based. Service disciplines can be further distinguished by whether they are work-conserving or non-work-conserving (see Glossary). Non-work-conserving schedulers can be used to shape traffic streams to match some profile by delaying packets that might be deemed non-conforming by some downstream node: a packet is delayed until such time as it would conform to a downstream meter using the same profile.

[DSARCH] defines PHBs without specifying required scheduling algorithms. However, PHBs such as the class selectors [DSFIELD], EF [EF-PHB] and AF [AF-PHB] have descriptions or configuration parameters which strongly suggest the sort of scheduling discipline needed to implement them. This memo discusses a minimal set of queue parameters to enable realization of these per- hop behaviors. It does not attempt to specify an all-embracing set of parameters to cover all possible implementation models. A mimimal set includes:

a) a minimum service rate profile which allows rate guarantees for each traffic stream as required by EF and AF without specifying the details of how excess bandwidth between these traffic streams is shared. Additional parameters to control this behavior should be made available, but are dependent on the particular scheduling

algorithm implemented.

b) a service priority, used only after the minimum rate profiles of all inputs have been satisfied, to decide how to allocate any remaining bandwidth.

c) a maximum service rate profile, for use only with a non-work-conserving service discipline.

For an implementation of the EF PHB using a strict priority scheduling algorithm that assumes that the aggregate EF rate has been appropriately bounded to avoid starvation, the minimum rate profile would be reported as zero and the maximum service rate would be reported as line rate. Such an implementation, with multiple priority classes, could also be used for the Diffserv class selectors [DSFIELD].

Alternatively, setting the service priority values for each input to the scheduler to the same value enables the scheduler to satisfy the minimum service rates for each input, so long as the sum of all minimum service rates is less than or equal to the line rate.

For example, a non-work-conserving scheduler, allocating spare bandwidth equally between all its inputs, might be represented using the following parameters:

```
Scheduler1:
Type:           Scheduler2Input

Input1:
MaxRateProfile: Profile1
MinRateProfile: Profile2
Priority:       none

Input2:
MaxRateProfile: Profile3
MinRateProfile: Profile4
Priority:       none
```

A work-conserving scheduler might be represented using the following parameters:

```
Scheduler2:
Type:           Scheduler3Input

Input1:
MaxRateProfile: WorkConserving
MinRateProfile: Profile5
```

```
     Priority:       1

     Input2:
     MaxRateProfile: WorkConserving
     MinRateProfile: Profile6
     Priority:       2

     Input3:
     MaxRateProfile: WorkConserving
     MinRateProfile: none
     Priority:       3
```

7.1.3. Algorithmic Dropper

An Algorithmic Dropper is an element which selectively discards packets that arrive at its input, based on a discarding algorithm. It has one data input and one output. In this model (but not necessarily in a real implementation), a packet enters the dropper at its input and either its buffer is returned to a free buffer pool or the packet exits the dropper at the output.

Alternatively, an Algorithmic Dropper may invoke operations on a FIFO which selectively removes a packet, then return its buffer to the free buffer pool, based on a discarding algorithm. In this case, the operation is modelled as a side-effect on the FIFO upon which it operates, rather than as having a discrete input and output. These two treatments are equivalent and we choose the former here.

The Algorithmic Dropper is modelled as having a single input. However, it is likely that packets which were classified differently by a Classifier in this TCB will end up passing through the same dropper. The dropper's algorithm may need to apply different calculations based on characteristics of the incoming packet e.g. its DSCP. So there is a need, in implementations of this model, to be able to relate information about which classifier element was matched by a packet from a Classifier to an Algorithmic Dropper. This is modelled here as a reverse pointer from one of the drop probability calculation algorithms inside the dropper to the classifier element that selects this algorithm.

There are many formulations of a model that could represent this linkage, other than the one described above: one way would have been to have multiple "inputs" fed from the preceding elements, leading eventually to the classifier elements that matched the packet. Another formulation might have been for the Classifier to (logically) include some sort of "classification identifier" along with the packet along its path, for use by any subsequent element. Yet another could have been to include a classifier inside the dropper, in order for it to pick out the

drop algorithm to be applied. All of these other approaches were deemed to be more clumsy or less useful than the approach taken here.

An Algorithmic Dropper, shown in Figure 5, has one or more triggers that cause it to make a decision whether or not to drop one (or possibly more than one) packet. A trigger may be internal (the arrival of a packet at the input to the dropper) or it may be external (resulting from one or more state changes at another element, such as a FIFO depth exceeding a threshold or a scheduling event). It is likely that an instantaneous FIFO depth will need to be smoothed over some averaging interval. Some dropping algorithms may require several trigger inputs feeding back from events elsewhere in the system e.g. smoothing functions that calculate averages over more than one time interval. Smoothing functions are outside the scope of this document and are not modelled here, we merely indicate where they might be added in the model.

A trigger may be a boolean combination of events (e.g. a FIFO depth exceeding a threshold OR a buffer pool depth falling below a threshold).

The dropping algorithm makes a decision on whether to forward or to discard a packet. It takes as its parameters some set of dynamic parameters (e.g. averaged or instantaneous FIFO depth) and some set of static parameters (e.g. thresholds) and possibly parameters associated

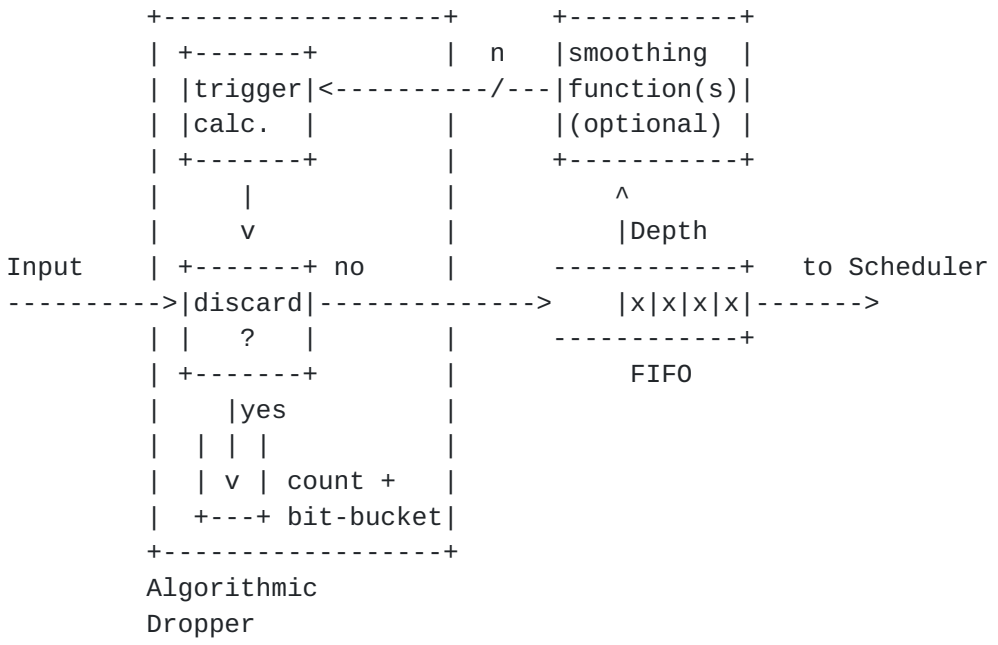

Figure 5. Algorithmic Dropper + Queue

with the packet (e.g. its PHB, as determined by a classifier, which will determine on which of the droppers inputs trhe packet arrives). It may also have internal state and is likely to keep counters regarding the dropped packets (there is no appropriate place here to include a Counter Action element).

RED, RED-on-In-and-Out (RIO) and Drop-on-threshold are examples of dropping algorithms. Tail-dropping and head-dropping are effected by the location of the dropper relative to the FIFO.

Note that, although an Algorithmic Dropper may require knowledge of data fields in a packet, as discovered by a Classifier in the same TCB, it may not modify the packet (i.e. it is not a marker).

> <ed: have rearranged this example so as not to include a Classifier in the Dropper - this leads to needing either multiple inputs or an implicit classification stage to separate the in- and out-of-profile traffic. We have chosen the former representation.>

A dropper which uses a RIO algorithm might be represented using the following parameters:

```
AlgorithmicDropper1:
Type:                   AlgorithmicDropper
Discipline:             RIO
Trigger:                Internal
Output:                 Fifo1

InputA: (in profile)
MinThresh:              Fifo1.Depth > 20 kbyte
MaxThresh:              Fifo1.Depth > 30 kbyte

InputB: (out of profile)
MinThresh:              Fifo1.Depth > 10 kbyte
MaxThresh:              Fifo1.Depth > 20 kbyte

SampleWeight            .002
MaxDropProb             1%
```

Another form of dropper, a threshold-dropper, might be represented using the following parameters:

```
AlgorithmicDropper2:
Type:                   AlgorithmicDropper
Discipline:             Drop-on-threshold
Trigger:                Fifo2.Depth > 20 kbyte
Output:                 Fifo1
```

Yet another dropper which drops all out-of-profile packets whenever the FIFO threshold exceeds a certain depth (this dropper is not part of the larger TCB example) might be represented with the following parameters:

```
AlgorithmicDropper3:
Type:                   AlgorithmicDropper2Input
Discipline:             Drop-out-packets-on-threshold
Output:                 Fifo3

InputA: (in profile)
Trigger:                none
InputB: (out of profile)
Trigger:                Fifo3.Depth > 100 kbyte
```

<ed: this models the dropper without using an embedded Classifier which seems a cleaner model than embedding a classifier here>

7.1.4. Constructing queueing blocks from the elements

A queueing block is constructed by concatenation of these elements so as to meet the meta-policy objectives of the implementation, subject to the grammar rules specified in this section.

Elements of the same type may appear more than once in a queueing block, either in parallel or in series. Typically, a queueing block will have relatively many elements in parallel and few in series. Iteration and recursion are not supported constructs in this grammar. A queueing block must have at least one FIFO, at least one dropper, and at least one scheduler. The following connections are allowed:

1) The input of a FIFO may be the input of the queueing block or it may be connected to the output of a dropper or to an output of a scheduler.

2) Each input of a scheduler may be connected to the output of a FIFO, to the output of a dropper or to the output of another scheduler.

3) The input of a dropper which has a discrete input and output may be the input of the queueing block or it may be connected to the output of a FIFO (e.g., head dropping).

4) The output of the queueing block may be the output of a FIFO element, a discarding element or a scheduling element.

Note, in particular, that schedulers may operate in series such that a packet at the head of a FIFO feeding the concatenated schedulers is serviced only after all of the scheduling criteria are met. For example,

a FIFO which carries EF traffic streams may be served first by a non-work-conserving scheduler to shape the stream to a maximum rate, then by a work-conserving scheduler to mix EF traffic streams with other traffic streams. Alternatively, there might be a FIFO and/or a dropper between the two schedulers.

7.2. Shaping

Traffic shaping is often used to condition traffic such that packets arriving in a burst will be "smoothed" and deemed conforming by subsequent downstream meters in this or other nodes. Shaping may also be used to isolate certain traffic streams from the effects of other traffic streams of the same BA.

In [DSARCH] a shaper is described as a queueing element controlled by a meter which defines its temporal profile. However, this representation of a shaper differs substantially from typical shaper implementations.

In this conceptual model, a shaper is realized by using a non-work-conserving scheduler. Some implementations may elect to have queues whose sole purpose is shaping, while others may integrate the shaping function with other buffering, discarding and scheduling associated with access to a resource. Shapers operate by delaying the departure of packets that would be deemed non-conforming by a meter configured to the shaper's maximum service rate profile. The packet is scheduled to depart no sooner than such time that it would become conforming.

8. Traffic Conditioning Blocks (TCBs)

The classifiers, meters, action elements and queueing elements described above can be combined into traffic conditioning blocks (TCBs). The TCB is an abstraction of a functional element that may be used to facilitate the definition of specific traffic conditioning functionality.

A general TCB might consist of the following four stages:

- Classification stage
- Metering stage
- Action stage
- Queueing stage

where each stage may consist of a set of parallel datapaths consisting of pipelined elements.

Note that a classifier is a 1:N element, metering and actions are typically 1:1 elements and queueing is a N:1 element. The whole TCB should, however, result in a 1:1 abstract element.

TCBs are constructed by connecting elements corresponding to these stages in any sensible order. It is possible to omit stages, to include null elements, or to concatenate multiple stages of the same type. TCB outputs may drive additional TCBs (on either the ingress or egress interfaces).

8.1. An Example TCB

A SLS is presumed to have been negotiated between the customer and the provider which specifies the handling of the customer's traffic by the provider's network. The agreement might be of the following form:

DSCP	PHB	Profile	Treatment
001001	EF	Profile4	Discard non-conforming.
001100	AF11	Profile5	Shape to profile, tail-drop when full.
001101	AF21	Profile3	Re-mark non-conforming to DSCP 001000, tail-drop when full.
other	BE	none	Apply RED-like dropping.

This SLS specifies that the customer may submit packets marked for DSCP 001001 which will get EF treatment so long as they remain conforming to Profile1 and will be discarded if they exceed this profile. The discarded packets are counted in this example, perhaps for use by the provider's sales department in convincing the customer to buy a larger SLS. Packets marked for DSCP 001100 will be shaped to Profile2 before forwarding. Packets marked for DSCP 001101 will be metered to Profile3 with non-conforming packets "downgraded" by being re-marked with a DSCP of 001000. It is implicit in this agreement that conforming packets are given the PHB originally indicated by the packets' DSCP field.

Figures 6 and 7 illustrates a TCB that might be used to handle this SLS at an ingress interface at the customer/provider boundary.

The Classification stage of this example consists of a single BA classifier. The BA classifier is used to separate traffic based on the Diffserv service level requested by the customer (as indicated by the DSCP in each submitted packet's IP header). We illustrate three DSCP filter values: A, B and C. The 'X' in the BA classifier is a wildcard filter that matches every packet not otherwise matched.

The paths for DSCP 001001 and 001101 then include a metering stage. There is a separate meter for each set of packets corresponding to classifier outputs A and C. Each meter uses a specific profile, as specified in the TCS, for the corresponding Diffserv service level. The meters in this example each indicate one of two conforming levels,

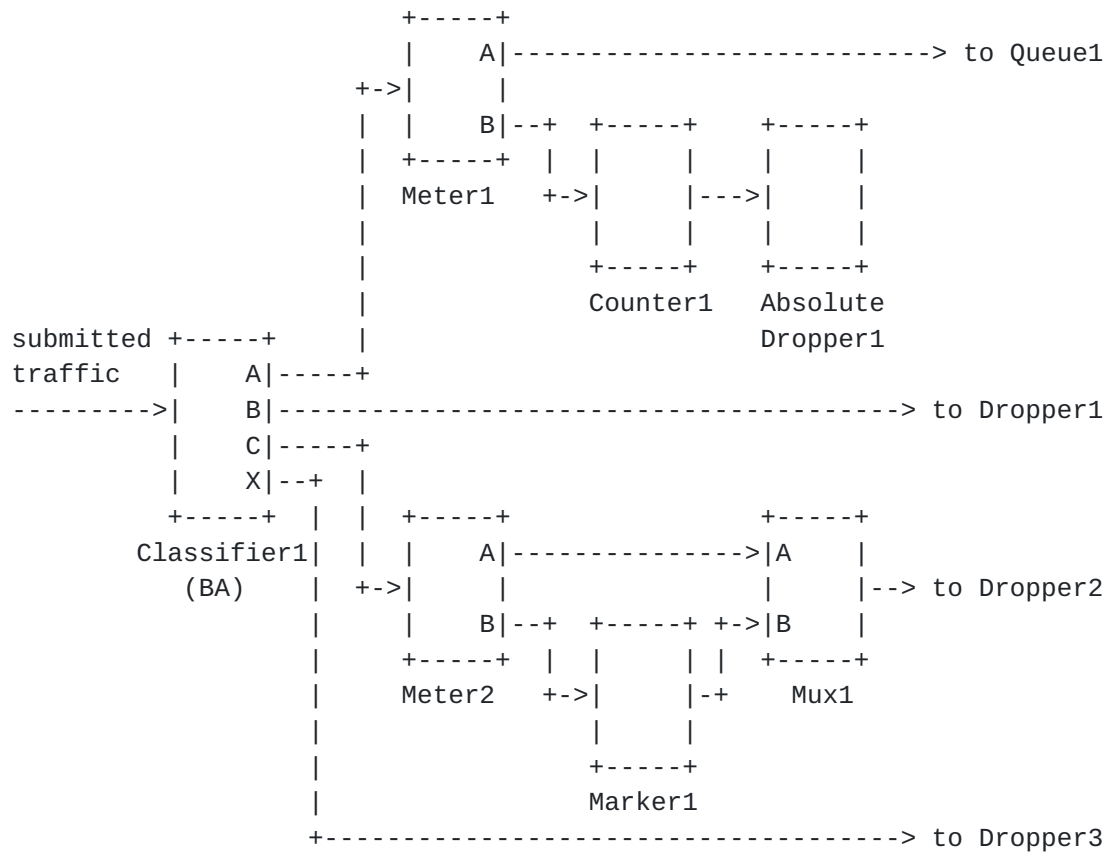

Figure 6: An Example Traffic Conditioning Block (Part 1)

conforming or non-conforming.

Following the Metering stage is the Action stage in the upper and lower branches. Packets submitted for DSCP 001001 that are deemed non-conforming are counted and discarded while packets that are conforming are passed on to Dropper1/Queue1. Packets submitted for DSCP 001101 that are deemed non-conforming are re-marked and then conforming and non-conforming packets are multiplexed together before being passed on to Dropper2/Queue3. Packets submitted for DSCP 001100 are passed straight on to Queue2.

The Queueing stage is realised as follows, shown in figure 6. The conforming 001001 packets are passed directly to Queue1: there is no way, with correct configuration of the scheduler for these to overflow the depth of Queue1 so there is never a requirement for dropping. Packets marked for 001100 must be passed through a tail-dropper, Dropper1, which serves to limit the depth of the following queue, Queue2: packets that arrive to a full queue will be discarded - this is likely to be an error case: the customer is obviously not sticking to

its agreed profile. Similarly, packets from the 001101 stream are passed to Dropper2 and Queue3. Packets marked for all other DSCPs are passed to Dropper3 which is a RED-like algorithmic dropper: based on feedback of the current depth of Queue4, this dropper is likely to discard enough packets from its input stream to keep the queue depth under control.

These four queues are then serviced by a scheduling algorithm in Scheduler1 which has been configured to give each of the queues an appropriate priority and/or bandwidth share. Inputs A and C are given guarantees of bandwidth, as appropriate for the contracted profiles. Input B is given a limit on the bandwidth it can use i.e. a non-work-conserving discipline in order to achieve the desired shaping of this stream. Input D is given no limits or guarantees but a lower priority than the other queues, appropriate for its best-effort status. Traffic then exits the scheduler in a single orderly stream.

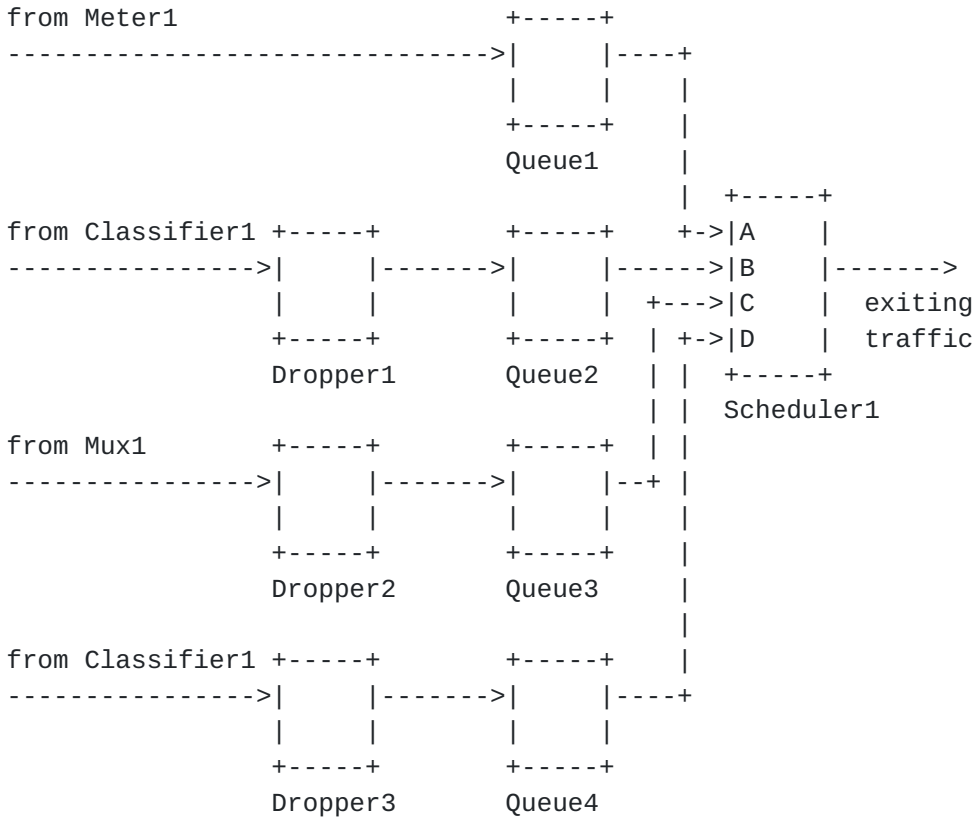

Figure 7: An Example Traffic Conditioning Block (Part 2)

The interconnections of the TCB elements illustrated in Figures 6 and 7 can be represented as follows:

```
      TCB1:

      Classifier1:
      FilterA:              Meter1
      FilterB:              Dropper1
      FilterC:              Meter2
      Default:              Dropper3

      Meter1:
      Type:                 AverageRate
      Profile:              Profile1
      ConformingOutput:     Queue1
      NonConformingOutput: Counter1

      Counter1:
      Output:               AbsoluteDropper1

      Meter2:
      Type:                 AverageRate
      Profile:              Profile3
      ConformingOutput:     Mux1.InputA
      NonConformingOutput: Marker1

      Marker1:
      Type:                 DSCPMarker
      Mark:                 001000
      Output:               Mux1.InputB

      Mux1:
      Output:               Dropper2

      Dropper1:
      Type:                 AlgorithmicDropper
      Discipline:           Drop-on-threshold
      Trigger:              Queue2.Depth > 10kbyte
      Output:               Queue2

      Dropper2:
      Type:                 AlgorithmicDropper
      Discipline:           Drop-on-threshold
      Trigger:              Queue3.Depth > 20kbyte
      Output:               Queue3

      Dropper3:
```

```
      Type:               AlgorithmicDropper
      Discipline:         RED93
      Trigger:            Internal
      Output:             Queue3
      MinThresh:          Queue3.Depth > 20 kbyte
      MaxThresh:          Queue3.Depth > 40 kbyte
         <other RED parms too>

      Queue1:
      Type:               FIFO
      Output:             Scheduler1.InputA

      Queue2:
      Type:               FIFO
      Output:             Scheduler1.InputB

      Queue3:
      Type:               FIFO
      Output:             Scheduler1.InputC

      Queue4:
      Type:               FIFO
      Output:             Scheduler1.InputD

      Scheduler1:
      Type:               Scheduler4Input
      InputA:
      MaxRateProfile:     none
      MinRateProfile:     Profile4
      Priority:           20
      InputB:
      MaxRateProfile:     Profile5
      MinRateProfile:     none
      Priority:           40
      InputC:
      MaxRateProfile:     none
      MinRateProfile:     Profile3
      Priority:           20
      InputD:
      MaxRateProfile:     none
      MinRateProfile:     none
      Priority:           10
```

8.2. An Example TCB to Support Multiple Customers

The TCB described above can be installed on an ingress interface to implement a provider/customer TCS if the interface is dedicated to the customer. However, if a single interface is shared between multiple customers, then the TCB above will not suffice, since it does not differentiate among traffic from different customers. Its classification stage uses only BA classifiers.

The TCB is readily extended to support the case of multiple customers per interface, as follows. First, a TCB is defined for each customer to reflect the TCS with that customer: TCB1, defined above is the TCB for customer 1 and definitions are then added for TCB2 and for TCB3 which reflect the agreements with customers 2 and 3 respectively.

Finally, a classifier is added to the front end to separate the traffic from the three different customers. This forms a new TCB, TCB4, which incorporates TCB1, TCB2, and TCB3 and is illustrated in Figure 8.

A formal representation of this multi-customer TCB might be:

```
TCB4:

Classifier4:
Filter1:     to TCB1
Filter2:     to TCB2
Filter3:     to TCB3
No Match:    AbsoluteDropper4

TCB1:
(as defined above)

TCB2:
```

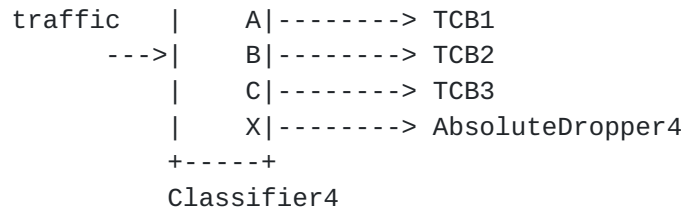

Figure 8: An Example of a Multi-Customer TCB

(similar to TCB1, perhaps with different numeric parameters)

TCB3:
(similar to TCB1, perhaps with different numeric parameters)

TCB4:
(the total TCB)

and the filters, based on each customer's source MAC address, could be defined as follows:

Filter1:
Type: MacAddress
SrcValue: 01-02-03-04-05-06 (source MAC address of customer 1)
SrcMask: FF-FF-FF-FF-FF-FF
DestValue: 00-00-00-00-00-00
DestMask: 00-00-00-00-00-00

Filter2:
(similar to Filter1 but with customer 2's source MAC address as SrcValue)

Filter3:
(similar to Filter1 but with customer 3's source MAC address as SrcValue)

In this example, Classifier4 separates traffic submitted from different customers based on the source MAC address in submitted packets. Those packets with recognized source MAC addresses are passed to the TCB implementing the TCS with the corresponding customer. Those packets with unrecognized source MAC addresses are passed to a dropper.

TCB4 has a Classifier stage and an Action element stage, which consists of either a dropper or another TCB.

8.3. TCBs Supporting Microflow-based Services

The TCB illustrated above describes a configuration that might be suitable for enforcing a SLS at a router's ingress. It assumes that the customer marks its own traffic for the appropriate service level. It then limits the rate of aggregate traffic submitted at each service level, thereby protecting the resources of the Diffserv network. It does not provide any isolation between the customer's individual microflows.

A more complex example might be a TCB configuration that offers additional functionality to the customer. It recognizes individual

customer microflows and marks each one independently. It also isolates the customer's individual microflows from each other in order to prevent a single microflow from seizing an unfair share of the resources available to the customer at a certain service level. This is illustrated in Figure 9.

Suppose that the customer has an SLS which specifices 2 service levels, to be identifed to the provider by DSCP A and DSCP B. Traffic is first directed to a MF classifier which classifies traffic based on miscellaneous classification criteria, to a granularity sufficient to identify individual customer microflows. Each microflow can then be marked for a specific DSCP The metering elements limit the contribution of each of the customer's microflows to the service level for which it was marked. Packets exceeding the allowable limit for the microflow are dropped.

This TCB could be formally specified as follows:

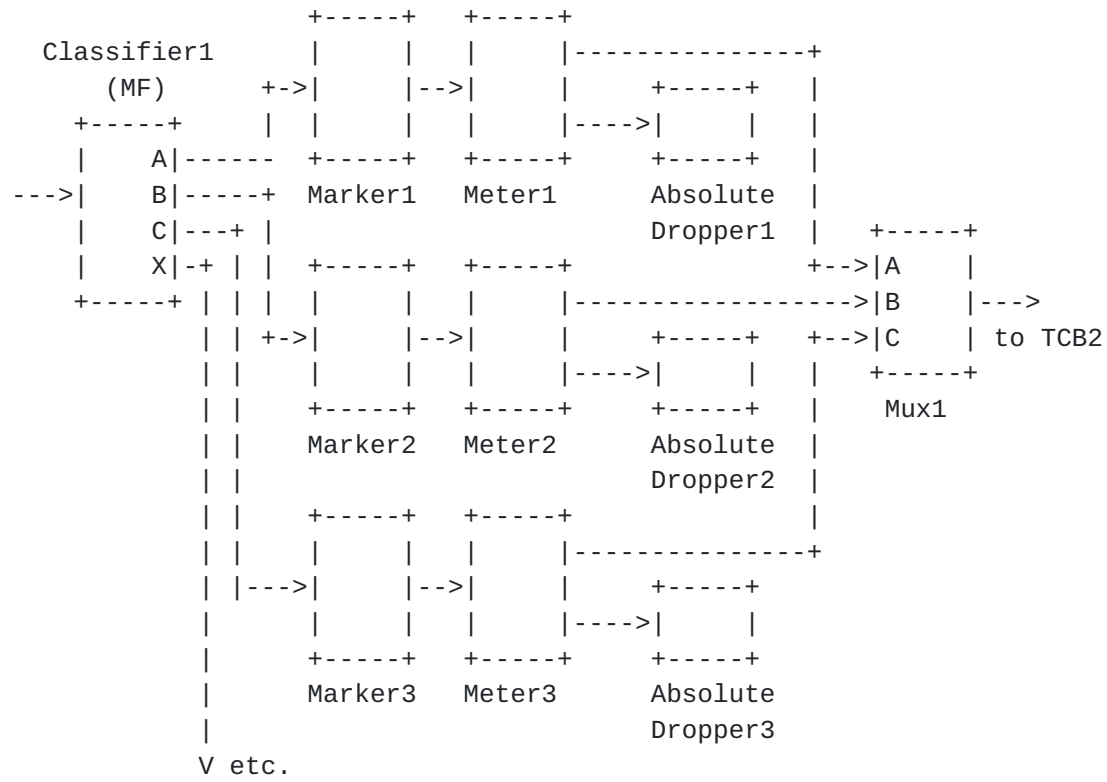

Figure 9: An Example of a Marking and Traffic Isolation TCB

```
      TCB1:
      Classifier1: (MF)
      FilterA:              Marker1
      FilterB:              Marker2
      FilterC:              Marker3
      etc.

      Marker1:
      Output:               Meter1

      Marker2:
      Output:               Meter2

      Marker3:
      Output:               Meter3

      Meter1:
      ConformingOutput:     Mux1.InputA
      NonConformingOutput: AbsoluteDropper1

      Meter2:
      ConformingOutput:     Mux1.InputB
      NonConformingOutput: AbsoluteDropper2

      Meter3:
      ConformingOutput:     Mux1.InputC
      NonConformingOutput: AbsoluteDropper3

      etc.

      Mux1:
      Output:               to TCB2
```

Note that the detailed traffic element declarations are not shown here. Traffic is either dropped by TCB1 or emerges marked for one of two DSCPs. This traffic is then passed to TCB2 which is illustrated in Figure 10.

TCB2 could then be specified as follows:

```
      Classifier2: (BA)
      FilterA:               Meter5
      FilterB:               Meter6

      Meter5:
      ConformingOutput:      Queue1
```

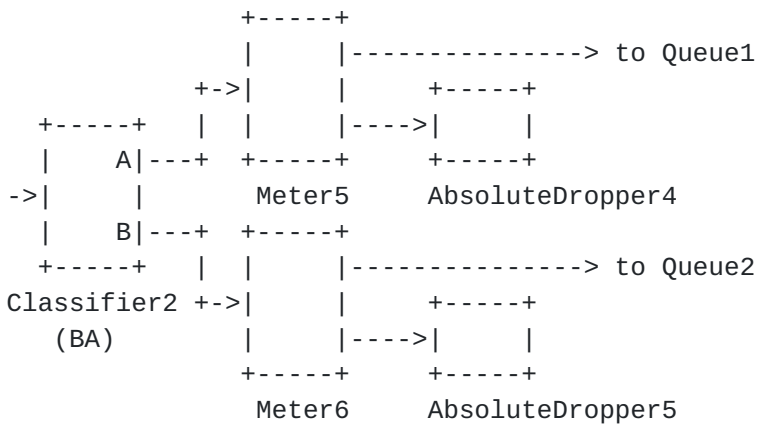

Figure 10: Additional Example: TCB2

NonConformingOutput: AbsoluteDropper4

Meter6:
ConformingOutput: Queue2
NonConformingOutput: AbsoluteDropper5

8.4. Cascaded TCBs

Nothing in this model prevents more complex scenarios in which one microflow TCB precedes another (e.g. for TCBs implementing separate TCSs for the source and for a set of destinations).

9. Open Issues

<ed: this section to be deleted before WG last call and RFC publication. The current stance of this draft is supplied in parentheses.

(1) FIFOs are modelled here as having infinite depth: it is up to any preceding meter/dropper to make sure that they do not overflow - a hard stop on the depth would be modelled, for example, by preceding the FIFO with an Absolute Dropper. Is this appropriate? (Yes)

(2) We must allow algorithmic droppers that apply different dropping behaviour to packets with different classifier matches, with these possibly fed through different meters and actions. Should we model the dropper as a single input element with implicit pointers back to the matching classifier that selects different dropper algorithms/treatments? Or as multiple droppers? Or as having multiple logical inputs? (single input, implicit pointers).

10. Security Considerations

Security vulnerabilities of Diffserv network operation are discussed in [DSARCH]. This document describes an abstract functional model of Diffserv router elements. Certain denial-of-service attacks such as those resulting from resource starvation may be mitigated by appropriate configuration of these router elements; for example, by rate limiting certain traffic streams or by authenticating traffic marked for higher quality-of-service.

One particular theft- or denial-of-service issue may arise where a token-bucket meter, with an absolute dropper for non-conforming traffic, is used in a TCB to police a stream to a given TCS: the definition of the token-bucket meter in section 5 indicates that it should be lenient in accepting a packet whenever any bits of the packet would have been within the profile; the definition of the leaky-bucket scheduler is conservative in that a packet is to be transmitted only if the whole packet fits within the profile. This difference may be exploited by a malicious scheduler either to obtain QoS treatment for more octets than allowed in the TCS or to disrupt (perhaps only slightly) the QoS guarantees promised to other traffic streams.

11. Acknowledgments

Concepts, terminology, and text have been borrowed liberally from [POLTERM], [DSMIB] and [DSPIB]. We wish to thank the authors of those documents: Fred Baker, Michael Fine, Keith McCloghrie, John Seligson, Kwok Chan and Scott Hahn for their contributions.

This document has benefitted from the comments and suggestions of several participants of the Diffserv working group.

12. References

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

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

[DSFIELD]
K. Nichols, S. Blake, F. Baker, and D. Black, "Definition of the

Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.

[DSMIB]
F. Baker, A. Smith, K. Chan, "Differentiated Services MIB", Internet Draft <draft-ietf-diffserv-mib-03.txt>, May 2000.

[DSPIB]
M. Fine, K. McCloghrie, J. Seligson, K. Chan, S. Hahn, and A. Smith, "Quality of Service Policy Information Base", Internet Draft <draft-ietf-diffserv-pib-00.txt>, March 2000.

[DSTERMS]
D. Grossman, "New Terminology for Diffserv", Internet Draft <draft-ietf-diffserv-new-terms-02.txt>, November 1999.

[E2E]
Y. Bernet, R. Yavatkar, P. Ford, F. Baker, L. Zhang, M. Speer, K. Nichols, R. Braden, B. Davie, J. Wroclawski, and E. Felstaine, "Integrated Services Operation over Diffserv Networks", Internet Draft <draft-ietf-issll-diffserv-rsvp-04.txt>, March 2000.

[EF-PHB]
V. Jacobson, K. Nichols, and K. Poduri, "An Expedited Forwarding PHB", RFC 2598, June 1999.

[GTC]
L. Lin, J. Lo, and F. Ou, "A Generic Traffic Conditioner", Internet Draft <draft-lin-diffserv-gtc-01.txt>, August 1999.

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

[POLTERM]
F. Reichmeyer, D. Grossman, J. Strassner, M. Condell, "A Common Terminology for Policy Management", Internet Draft <draft-reichmeyer-polterm-terminology-00.txt>, March 2000

[QOSDEVMOD]
J. Strassner, W. Weiss, D. Durham, A. Westerinen, "Information Model for Describing Network Device QoS Mechanisms", Internet Draft <draft-ietf-policy-qos-device-info-model-00.txt>, April 2000

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

[TRTCM]
J. Heinanen, R. Guerin, "A Two Rate Three Color Marker", RFC 2698, September 1999.

13. Authors' Addresses

Yoram Bernet
Microsoft
One Microsoft Way
Redmond, WA 98052
Phone: +1 425 936 9568
E-mail: yoramb@microsoft.com

Andrew Smith
Extreme Networks
3585 Monroe St.
Santa Clara, CA 95051
Phone: +1 408 579 2821
E-mail: andrew@extremenetworks.com

Steven Blake
Ericsson
920 Main Campus Drive, Suite 500
Raleigh, NC 27606
Phone: +1 919 472 9913
E-mail: slblake@torrentnet.com

Daniel Grossman
Motorola Inc.
20 Cabot Blvd.
Mansfield, MA 02048
Phone: +1 508 261 5312
E-mail: dan@dma.isg.mot.com

Table of Contents

1 Introduction 2
2 Glossary 3
3 Conceptual Model 5
3.1 Elements of a Diffserv Router 5
3.1.1 Datapath 5
3.1.2 Configuration and Management Interface 6
3.1.3 Optional QoS Agent Module 7
3.2 Hierarchical Model of Diffserv Components 7

4 Classifiers 10
4.1 Definition 10
4.1.1 Filters 11
4.1.2 Overlapping Filters 11
4.2 Examples 12
4.2.1 Behaviour Aggregate (BA) Classifier 12
4.2.2 Multi-Field (MF) Classifier 13
4.2.3 Free-form Classifier 13
4.2.4 Other Possible Classifiers 14
5 Meters 14
5.1 Examples 17
5.1.1 Average Rate Meter 17
5.1.2 Exponential Weighted Moving Average (EWMA) Meter 18
5.1.3 Two-Parameter Token Bucket Meter 19
5.1.4 Multi-Stage Token Bucket Meter 19
5.1.5 Null Meter 20
6 Action Elements 20
6.1 Marker 21
6.2 Absolute Dropper 21
6.3 Multiplexer 22
6.4 Counter 22
6.5 Null Action 22
7 Queueing Blocks 22
7.1 Queueing Model 23
7.1.1 FIFO 23
7.1.2 Scheduler 25
7.1.3 Algorithmic Dropper 27
7.1.4 Constructing queueing blocks from the elements 30
7.2 Shaping 31
8 Traffic Conditioning Blocks (TCBs) 31
8.1 An Example TCB 32
8.2 An Example TCB to Support Multiple Customers 37
8.3 TCBs Supporting Microflow-based Services 38
8.4 Cascaded TCBs 41
9 Open Issues 41
10 Security Considerations 42
11 Acknowledgments 42
12 References 42
13 Authors' Addresses 44

14. Full Copyright