Internet Engineering Task Force
Diffserv Working Group
INTERNET-DRAFT
Expires: September 2000

Y. Bernet
Microsoft
A. Smith
Extreme Networks
S. Blake
Ericsson
D. Grossman
Motorola
March 2000

A Conceptual Model for Diffserv Routers

draft-ietf-diffserv-model-02.txt

Status of this Memo

Copyright Notice

Abstract

DISCLAIMER - for reasons outside our control this version has been rushed out with formatting errors and not checked by all authors.

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, replicators, muxes, 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 should serve as a rationale for the design of a Diffserv MIB [DSMIB], as well for various configuration interfaces (such as [PIB]). Since these documents are all evolving simultaneously there are discrepancies between their current revisions; this should be resolved in a future revision of this draft.

Table of Contents

1. Introduction

Differentiated Services (Diffserv) [DSARCH] is a set of technologies which allow network service providers to offer differing levels 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 Sec. 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. We expect that router vendors will demonstrate a great deal of variability in their implementations. To the extent that vendors 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 various implementations.
In Sec. 3 we start by describing the basic high-level functional elements of a Diffserv router and then describe the various components. We then focus on the Diffserv-specific components of the router and describe a hierarchical management model for these.

In Sec. 4 we describe classification elements and in Sec. 5, we discuss the meter elements.

In Sec. 6 we discuss action elements. In Sec. 7 we discuss the basic queueing elements and their functional behaviors (e.g., shaping).

In Sec. 8, we show how the basic classification, meter, action, and queueing elements can be combined to build modules called Traffic Conditioning Blocks (TCBs).

In Sec. 9 we discuss open issues with this document and in Sec. 10 we discuss security concerns.

Appendix A discusses token bucket implementation details.

2. Glossary

Some of the terms used in this draft are defined in [DSARCH] and in [DSTERMS]. We define a few of them here again only to provide additional detail.

Buffer management algorithm	An algorithm used to determine whether an arriving packet should be stored in a queue, or discarded. This decision is usually a function of the instantaneous or average queue occupancy, but also may be a function of the aggregate queue occupancy in a queue set, or of other parameters.
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.
Enqueueing	The process of executing a buffer management algorithm to determine whether an arriving packet should be stored in a queue.
Filter	A set of (wildcard/prefix/masked/range/exact) conditions on the components of a packet's classification key. A filter is said to match only if each condition is satisfied.
Replicating element	A functional datapath element which makes one or more copies of a packet and forwards them on distinct datapaths; for example to a monitoring port.
Monitor	A functional datapath element which updates an octet and a packet counter for every packet which passes through it. Used for collecting statistics.
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 does not necessarily service a packet if available at every transmission opportunity.
Queue	A storage location for packets awaiting transmission or processing by the next functional element in the data-path. The queues represented in this model are abstract elements that may be implemented by multiple physical queues in series and/or in parallel in a specific implementation. Note that we assume that a queue is serviced such as to preserve the required ordering constraint for each Ordering Aggregate (OA) it queues [DSTERMS]. This can be achieved by a FIFO (first in, first out) service policy or by other means (e.g., multiple FIFOs exclusively servicing particular OAs).
Queue set	A set of queues which are serviced by a scheduling algorithm and which may share a buffer management algorithm.
Scheduling algorithm	An algorithm which determines which queue of a queue set to service next. This may be based on the relative priority of the queues, or on a weighted fair bandwidth sharing policy, or some other policy. A scheduling 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 at least one input and output and a set of control parameters.
Work conserving	A property of a scheduling algorithm such that it services a packet if available at every transmission opportunity.

3. Conceptual Model

In this section we introduce a block diagram of a Diffserv router and describe the various components illustrated. Note that a Diffserv core router is assumed to include only a subset of these components: the model we present here is intended to cover the case of both Diffserv edge and core routers.

3.1 Elements of a Diffserv Router

The conceptual model we define includes abstract definitions for the following:

- The basic traffic classification components.
- The basic traffic conditioning components.
- Certain combinations of traffic classification and conditioning components.
- Queueing components.

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:

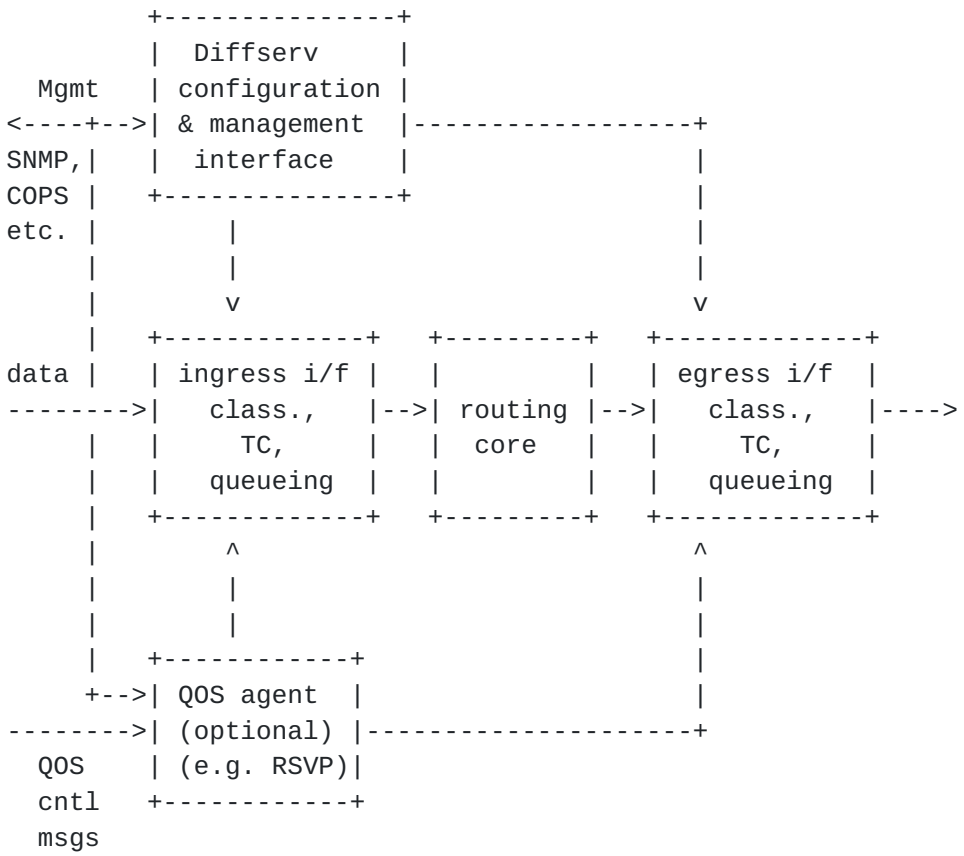

Figure 1: Diffserv Router Major Functional Blocks

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 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 RSVP Module

Diffserv routers may snoop or participate in either per-microflow or per-flow-aggregate signaling of QoS requirements [E2E]. The example discussed here uses 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 starts to look the same as a Integrated Services (intserv) router in its component makeup [E2E].

Note that a RSVP 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 is used strictly as a signaling protocol. The data plane of such a Diffserv router can still act purely on Diffserv DSCPs and PHBs in handling data traffic.

3.2 Hierarchical Model of Diffserv Components

We focus on the Diffserv specific functional components of the router: the classification, traffic conditioning, and queueing functionality. The diagram below is based on the larger block diagram shown above:

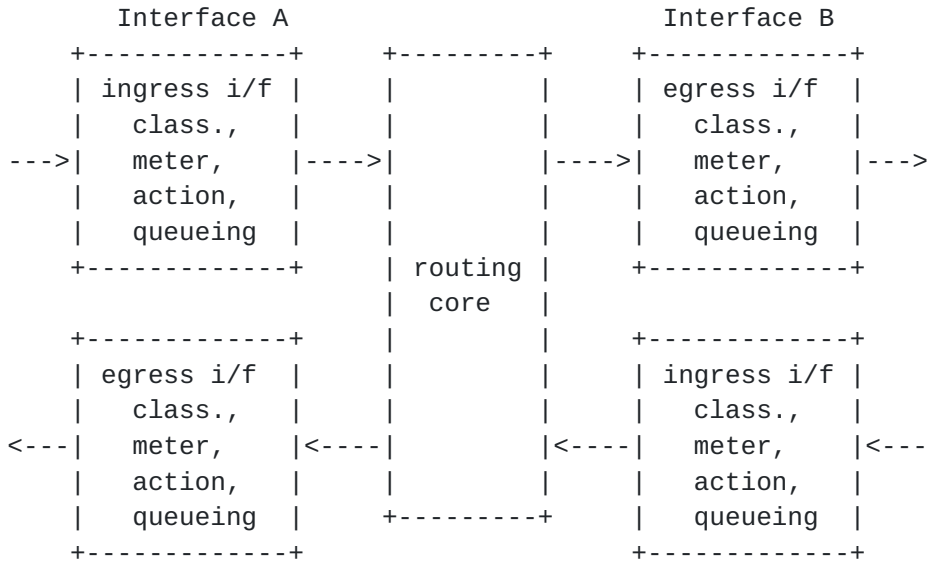

Figure 2. Traffic Conditioning and Queueing Elements

This 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. We show equivalent functional elements on both the ingress and egress components of an interface because we expect an N-port router to display the same Diffserv capabilities as a network of 2-port routers interconnected by LAN media [DSMIB]. Note that it is not mandatory that each of these functional elements be implemented on both ingress and egress components; it is dependent on the service requirements on a particular interface on a particular router. Further, we wish to point out that by showing these elements

on both ingress and egress components we do not mean to imply 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. Further, 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 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 Sec. 8). One or more TCBs may be instantiated on each ingress or egress component, may be connected in series, and/or may be connected in a parallel configuration on the multiple outputs of a classifier. We define the TCB to optionally include classification and queueing elements so as to allow for rich functionality. A TCB can be thought of as a "black box" with a single input and a single output (on the main data path). TCBs can be constructed out of a DAG of other TCBs, recursively. We do not assume the same TCB configuration on every interface (ingress or egress).

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:

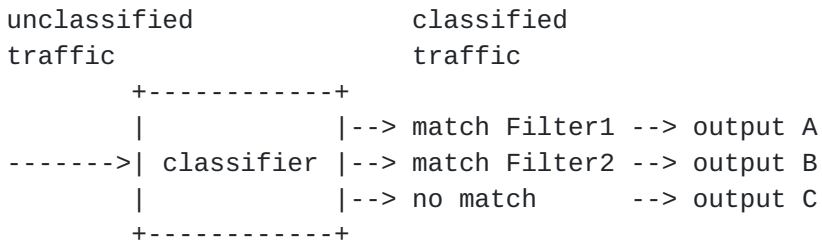

Figure 3. An Example Classifier

Note that we allow a mux (see Sec. 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 possible 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], see Sec. 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 filtering of fragmented packets is impossible. MTU size discovery is therefore prerequisite for proper operation of a diffserv network.

4.1.2 Overlapping Filters

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

```
Filter5:              Filter6:
Type:   Masked-DSCP   Type:   Masked-DSCP
Value:  111000        Value:  000111 (binary)
Mask:   111000        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.

An unambiguous classifier requires that every possible classification key match at least one filter (including the wildcard default), and that any ambiguity between overlapping filters be resolved by precedence.

4.1.3 Filter Groups

Filters may be logically combined. For example, consider the following DestMacAddress filter:

```
Filter7:
Type:        DestMacAddress
Value:       01-02-03-04-05-06
Mask:        FF-FF-FF-FF-FF-FF
```

Classifier0 could then be declared as:

```
Classifier0:
Filter1 and Filter7:          output A
Filter2 and Filter7:          output B
Default (wildcard) filter:    output C
```

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 header (including the DSCP). A common type of MF classifier is a 6-tuple classifier that classifies based on six IP header fields (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
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 IEEE802 MAC Address Classifier

A MacAddress filter is parameterized by a 6-byte {value, mask} pair for either source or destination MAC address. For example, the following classifier sends packets matching either DA = 01-02-03-04-05-06 or SA = 00-E0-2B-XX-XX-XX to output A:

```
Classifier1:
Filter10:     output A
Filter11:     output A
Default:      output B
```

```
Filter10:
Type:        DestMacAddress
Value:       01-02-03-04-05-06 (hex)
Mask:        FF-FF-FF-FF-FF-FF (hex)

Filter11:
Type:        SrcMacAddress
DestValue:   00-E0-2B-00-00-00 (hex)
DestMask:    FF-FF-FF-00-00-00 (hex)
```

4.2.4 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:    output A
Filter13:     output B
Default:     output C

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.

4.2.5 Other Possible Classifiers

```
Classifier3:
Filter14:     output A
Filter15:     output B
Default:      output C

Filter14:
Type:        IEEEPriority
Value:       100 (binary)
Mask:        101 (binary)
```

```
Filter15:
Type:        IEEEVLAN
Value:       100100000000 (binary)
Mask:        111111111111 (binary)
```

Classification may be performed based on implicit information associated with a packet (e.g. the incoming channel number on a channelized interface) or on information derived from a different non-Diffserv classification operation (e.g. the outgoing interface determined by the route lookup operation). Other vendor-specific filter formats are possible. We do not discuss these further here.

4.3 MPLS

It is possible for an MPLS label-switched router (LSR) to function as a Diffserv router [MPLSDS]. The interaction between MPLS and Diffserv is not discussed further in this document.

5. Meters

5.1 Definition

Metering is the function of monitoring the arrival times of packets of a traffic stream and determining the level of conformance of each packet to a pre-established traffic profile. 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 mux 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 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. We find the description here more powerful.

We use the following diagram to illustrate a meter with 3 levels of conformance:

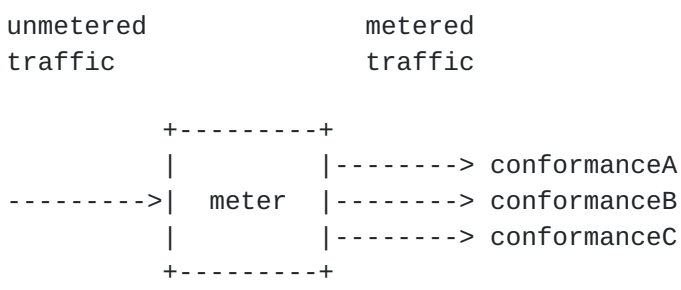

Figure 4. An Example Meter

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 are used to trigger different buffer management actions. 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.

5.2 Examples

The following is a non-exhaustive list of possible meters.

5.2.1 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
Profile1:            output A
NonConforming:       output B

Profile1:
```

```
      Type:                AverageRate
      AverageRate:         120 KBps
      Delta:               1.0 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 - 1.0 msecs. So long as an arriving packet does not push the count over 120 bytes, the packet would be deemed conforming. Any packet that pushes the count over 120 would be deemed non-conforming. Thus, this meter deems packets to correspond to one of two conformance levels: conforming or non-conforming.

5.2.2 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 as follows:

```
Meter2:
Type:                ExpWeightedMovingAvg
Profile2:            output A
NonConforming:       output B

Profile2:
Type:                ExpWeightedMovingAvg
AverageRate:         25 KBps
Delta:               10.0 usec
Gain:                1/16
```

5.2.3 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, a burst size. TB meters compare the arrival rate of packets to the average rate specified by the TB profile. Logically, byte 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 L tokens are available in the bucket at the time of packet arrival. Packets are allowed to exceed the average rate in bursts up to the burst size. Packets

which arrive to find a bucket with insufficient 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.

A two-parameter RB meter profile might look as follows:

```
Meter3:
Type:               SimpleTokenBucket
Profile3:           output A
NonConforming:      output B

Profile3:
Type:               SimpleTokenBucket
AverageRate:        100 KBps
BurstSize:          100 KB
```

5.2.4 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:               MultiTokenBucket
Profile4:           output A
Profile5:           output B
NonConforming:      output C

Profile4:
Type:               SimpleTokenBucket
AverageRate:        100 KBps
BurstSize:          20 KB

Profile5:
Type:               SimpleTokenBucket
AverageRate:        100 KBps
```

BurstSize: 100 KB

5.2.5 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.

6. Action Elements

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

1) Marking
2) Dropping
2) Shaping
3) Replicating
4) Monitoring

The corresponding action elements are described in the following paragraphs.

Policing is a general term for the process of preventing a traffic stream from seizing more than its share of resources from a Diffserv network. Each of the first three actions described above may be used to police traffic. Markers do so by re-marking non-conforming packets to a DSCP value that is entitled to fewer network resources. Shapers and droppers do so by limiting the rate at which a particular traffic stream is submitted to the network.

6.1 Marker

Markers are 1:1 elements which set the DSCP in an IP header (in the case of unlabeled packets). Markers may act on unmarked packets (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 DSCP set in a packet will determine its subsequent treatment in downstream nodes of a network, and possible in subsequent processing stages within the router (depending on configuration).

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

```
ActionElement1:
Type:               Marker
Mark:               010010
```

In the case of a MPLS labeled packet, the marker is parameterized

by a 3-bit EXP value to be marked in the MPLS shim header.

6.2 Dropper

Droppers simply discard packets. There are no parameters for droppers. Because a dropper is a terminating point of the datapath, it may be desirable to forward the packet through a monitor first for instrumentation purposes.

Droppers are not the only elements than can cause a packet to be discarded. The other element is an enqueueing element (see Sec. 6.6). However, since the enqueueing element's behavior is closely tied the state of one or more queues, we choose to distinguish them as separate functional elements.

6.3 Shaper

Shapers are used to shape traffic streams to a certain temporal profile. For example, a shaper can be used to smooth traffic arriving in bursts. In [DSARCH] a shaper is described as a queueing element controlled by a meter which defines its temporal profile. This model of a shaper differs substantially from typical shaper implementations. Further, with the inclusion of queueing elements in the model a separate shaping element becomes confusing. Therefore, the function of a shaper is embedded in a queue and is covered in Sec. 7.

6.4 Replicating Element

It is occasionally desirable to replicate traffic on one or more additional interfaces for data collection purposes. A replicating element is a 1:N (fan-out) element. However, each and every packet follows each output path simultaneously. A replicating element is parameterized by the number of outputs it supports.

6.5 Mux

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

6.6 Monitor

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. Monitors are 1:1 functional elements which update an octet counter by L and a packet counter by 1 every time a L-byte sized packet passes through it. Monitors can also be used to count packets on the verge of being dropped by a dropper.

6.7 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.

7. Queueing block

The queueing block modulates the transmission of packets belonging to the different traffic streams and determines 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 Model

Queuing 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 queuing block is thus composed of of one or more FIFO, one or more scheduler, and one or more discarder. See figure TBA for an example of a queueing block.

Note that the term FIFO is overloaded (i.e., has more than one meaning). In common usage it is taken to mean, among other things, a data structure that permits items to be removed only in the order in which they were inserted, and a service discipline which is non-reordering.

7.1.1 FIFO

A FIFO element is a data structure which at any time may contain zero or more packets. It may have one or more threshold 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 depth, which indicates the number of packets that it contains at a particular time; this is a traffic dependent variable and not used to configure a FIFO.

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 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 buffer. Buffers are allocated from one or more free buffer pool. 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. Otherwise, buffering mechanisms are implementation specific and not part of this model.

A FIFO might be represented using the following parameters:

```
FIFO1:
Type:       FIFO
Input:      QueuingBlock.input1
Output:     Discarder2
Threshold1: 3 packets
```

Another FIFO may be represented using the following parameters:

```
FIFO2:
Type:       FIFO
Input:      Discarder1
Output:     Scheduler1
Threshold1: 3 packets
Threshold2: 1000 octets
Threshold3: 10 packets
Threshold4: 2000 octets
```

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 may take as its inputs static parameters (such as relative priority, and/or absolute token bucket parameters for maximum or minimum rates) associated with each of the scheduler's inputs; parameters (such as packet length or DSCP) associated with the packet present at its input; 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. A work conserving service discipline transmits a packet at every transmission opportunity if one is available. A non-work conserving service discipline transmits packets no sooner than a scheduled departure time, even if it means leaving packets in a FIFO while the link is idle. Non-work conserving schedulers can be used to shape traffic streams by delaying packets that would be deemed non-conforming by some traffic profile. The packet is delayed until such time as it would conform to a meter using the same profile.

[DSARCH] defines PHBs without specifying required scheduling algorithms. However, PHBs such as the class selctors [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 specifies 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. The mimimum set includes a minimum service rate profile, a

service priority and a maximum service rate profile (the latter is for use only with a non-work conserving service discipline). The minimum service rate 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. The service priority is used only after the MinRateProfiles of all inputs have been satisfied in order to decide how to allocate any remaining bandwidth. It could be used for the class selectors. For the EF PHB, using a strict priority scheduling algorithm on some links, and assuming that the aggregate EF rate has been appropriately bounded to avoid starvation, for this scheduler the MinRateProfile would be reported as zero and the MaxRateProfile reported as line rate. Setting the service priority of 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.

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

```
Scheduler1:
Type:           Scheduler

Input1:         Discarder1
MaxRateProfile: Profile1
MinRateProfile: Profile2
Priority:       None

Input2:         Discarder1
MaxRateProfile: Profile3
MinRateProfile: Profile4
Priority:       None
```

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

```
Scheduler2:
Type:           Scheduler

Input1:         Scheduler1,
MaxRateProfile: WorkConserving
MinRateProfile: Profile5
Priority:       1

Input2:         FIFO2
MaxRateProfile: WorkConserving
MinRateProfile: Profile6
Priority:       2
```

```
        Input3:         FIFO3
        MaxRateProfile: WorkConserving
        MinRateProfile: None
        Priority:       3
```

7.1.3 Discarder

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

Alternatively, a discarder may invoke operations on a FIFO which selectively remove packets, then return those packets to the free buffer pool, based on a discarding discipline. In this case, the discarder's operation is modelled as a side-effect on the FIFO upon which it operates, rather than as having a discrete input and output.

A discarder has a trigger that causes the discarder to make a decision whether or not to drop one (or possibly more than one) packet. The trigger may internal (i.e., the arrival of a packet at the input to the discarder), or it may be external (i.e., resulting from one or more state change at another element, such as a FIFO depth exceeding a threshold or a scheduling event). 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 discarding discipline is an algorithm which makes a decision to forward or 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 with the packet (e.g. its PHB, as determined by a classifier). It may also have internal state. RED, RIO, and drop-on-threhold are examples of a discarding discipline. Tail dropping and head dropping are effected by the location of the discarder relative to the FIFO.

Note that although a discarder may need to examine the DSCP or possibly other fields in a packet, it may not modify them (i.e., it is not a marker).

A discarder might be represented using the following parameters:

```
        Discarder1:
        Type:                   Discarder
        Trigger:                Internal
        Input:          QueuingBlock.input2
        Output:         FIFO1
```

```
        Discipline:             RIO

        Parameters:
        In-MinTh:               FIFO1.Threshold1
        In-MaxTh:               FIFO1.Threshold2
        Out-Minth:              FIFO1.Threshold3
        Out-Maxth:              FIFO1.Threshold4
        InClassification:       AFx1_PHB
        OutClassifcation:       AFx2_PHB
        W_q                     .002
        Max_p                   .01

Another discarder might be represented using the following parameters:
        Discarder2:
        Type:                   Discarder
        Trigger:
        Input:          FIFO2
        Output:         Scheduler1.input1
        Discipline:             Drop-on-threshold

        Parameters:
        Threshold               FIFO2.Threshold1

Yet another discarder (not part of the example) might be represented
with the following parameters:
        Discarder3:
        Type:                   Discarder
        Operate_on              FIFO3
        Trigger:                FIFO3.depth > 100 packets
        Discipline:             Drop-all-out-packets

        Parameters:
        Out-DSCP:               AFx2_recommended_DSCP | AFx3_recommended_DSCP
```

7.1.4 Constructing queueing blocks from the elements

A queuing 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 queuing block will have relatively many elements in parallel and few in series. Iteration and recursion are not supported constructs in this grammar. A queuing block must have at least one FIFO, at least one discarder, and at least one scheduler. The following connections are allowed:

The input of a FIFO may be the input of the queueing block, or it may be connected to the output of a discarder or to an output of a scheduler.

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

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

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 discarder between the two schedulers.

7.2 Shaping

Traffic shaping is often used to condition traffic such that packets will be deemed conforming by subsequent meters, e.g., in downstream Diffserv nodes. Shaping may also be used to isolate certain traffic streams from the effects of other traffic streams of the same BA.

A shaper is realized in this model 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.

One of the simplest possible TCBs would consist of the following stages:

1. Classifier stage
2. Enqueueing stage
3. Queueing stage

Note that a classifier is a 1:N element, while an enqueueing stage is a N:1 element and a queue is a 1:1 element. If the classifier split traffic across multiple enqueueing elements then the queueing stage may consist of a hierarchy of queue sets, all resulting in a 1:1 abstract element.

A more general TCB might consists of the following four stages:

1. Classifier stage
2. Metering stage
3. Action stage
4. Queueing stage

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

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). Classifiers and meters are fan-out elements, muxes and enqueueing elements are fan-in elements.

8.1 An Example TCB

The following diagram illustrates an example TCB:

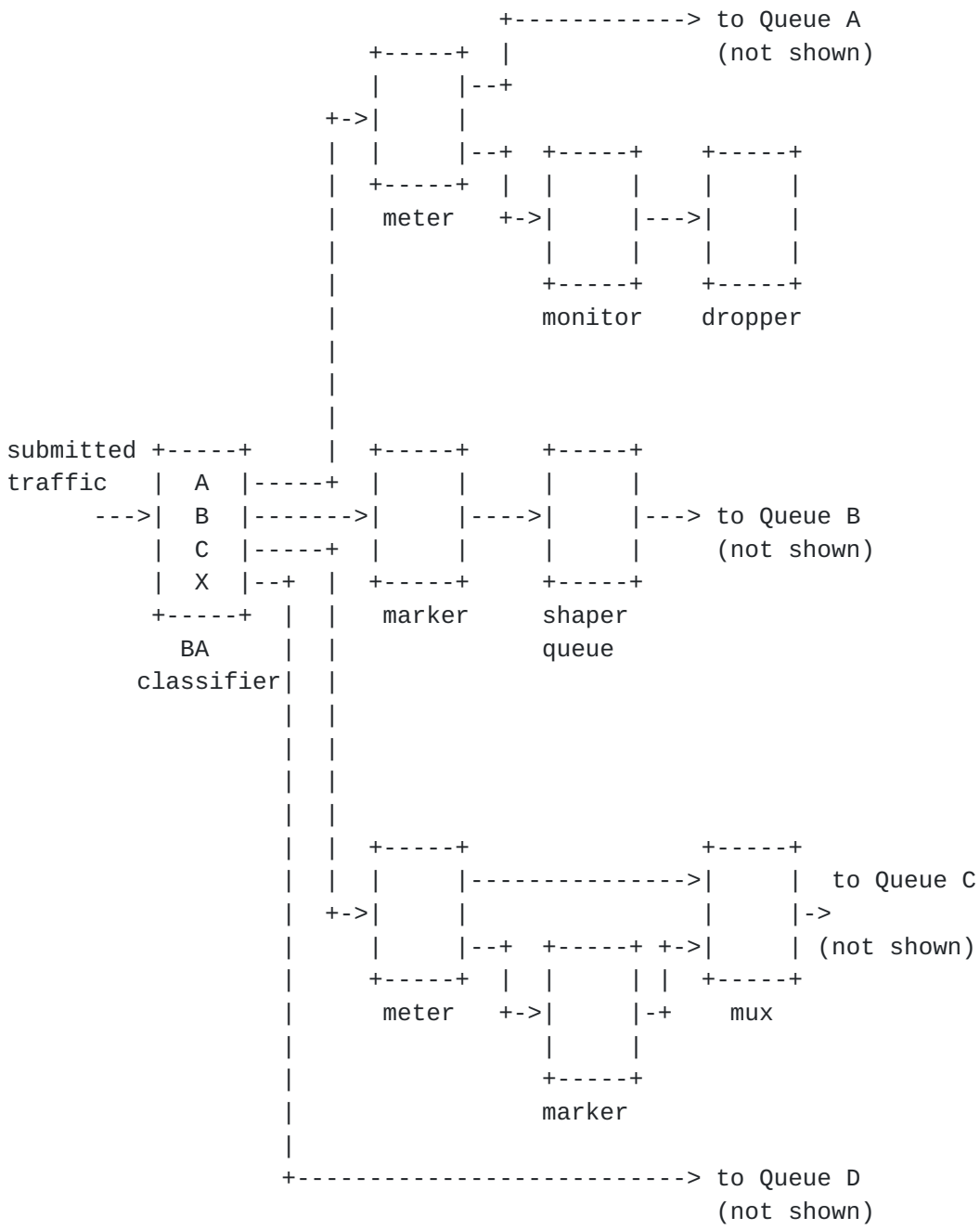

Figure 5: An Example Traffic Conditioning Block

This sample TCB might be suitable for an ingress interface at a customer/provider boundary. 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	Non-Conforming Packets

001001	PHB1	Profile1	Discard
001100	PHB2	Profile2	Wait in shaper queue
001101	PHB3	Profile3	Re-mark to DSCP 001000

It is implicit in this agreement that conforming packets are given the PHB originally indicated by the packets' DSCP field. It specifies that the customer may submit packets marked for DSCP 001001 which will get PHB1 treatment so long as they remain conforming to Profile1 and will be discarded if they exceed this profile. Similar contract rules are applied for 001100 and 001101 traffic.

In this example, the classification stage 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 the default wildcard filter that matches every packet.

A metering stage is next in the upper and lower branches. There is a separate meter for each set of packets corresponding to DSCPs 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 indicate one of two conforming levels, conforming or non-conforming. The middle branch has a marker which re-marks all packets received with DSCP B.

Following the metering stage is the action stage in the upper and lower branches. Packets submitted for DSCP A that are deemed non-conforming and are counted and discarded. Packets that are conforming are passed on to Queue A. Packets submitted for DSCP C that are deemed non-conforming are re-marked, and then conforming and non-conforming packets are muxed together before being forwarded to Queue C. Packets submitted for DSCP B are shaped to Profile2 before being forwarded to Queue B.

The interconnections of the TCB elements illustrated in Fig. 5 can be represented as follows:

```
TCB1:

Classifier1:
Output A --> Meter1
Output B --> Marker1
Output C --> Meter2
Output X --> QueueD

Meter1:
Output A --> QueueA
Output B --> Monitor1

Monitor1:
Output A --> Dropper1
```

```
Marker1:
Output A --> Shaper1
```

```
Shaper1:
Output A --> Queue B

Meter2:
Output A --> Mux1
Output B --> Marker2

Marker2:
Output A --> Mux1

Mux1:
Output A --> Queue C
```

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, we define a TCB for each customer to reflect the TCS with that customer. TCB1, defined above is the TCB for customer 1. We add definitions for TCB2 and for TCB3 which reflect the agreements with customers 2 and 3 respectively.

Finally, we add a classifier which provides a front end to separate the traffic from the three different customers. This forms a new TCB which incorporates TCB1, TCB2, and TCB3, and can be illustrated as follows:

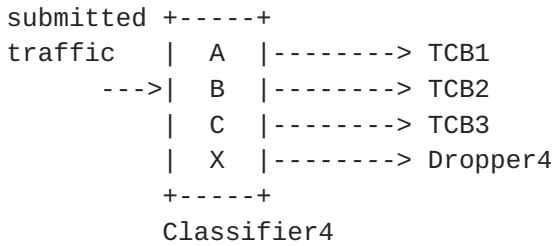

Figure 6: An Example of a Multi-Customer TCB

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

```
TCB1:
(as defined above)
```

TCB2:
(similar to TCB1, perhaps with different numeric parameters)

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

   TCB4:
   (the total TCB)

   Classifier4:
   Output A --> TCB1
   Output B --> TCB2
   Output C --> TCB3
   Output X --> Dropper4
```

Where Classifier2 is defined as follows:

```
   Classifier4:
   Filter1:     Output A
   Filter2:     Output B
   Filter3:     Output C
   No Match:    Output X
```

and the filters, based on each customer's source MAC address, are 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 classification 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 (other than from separated queueing).

Next we present 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 7 below:

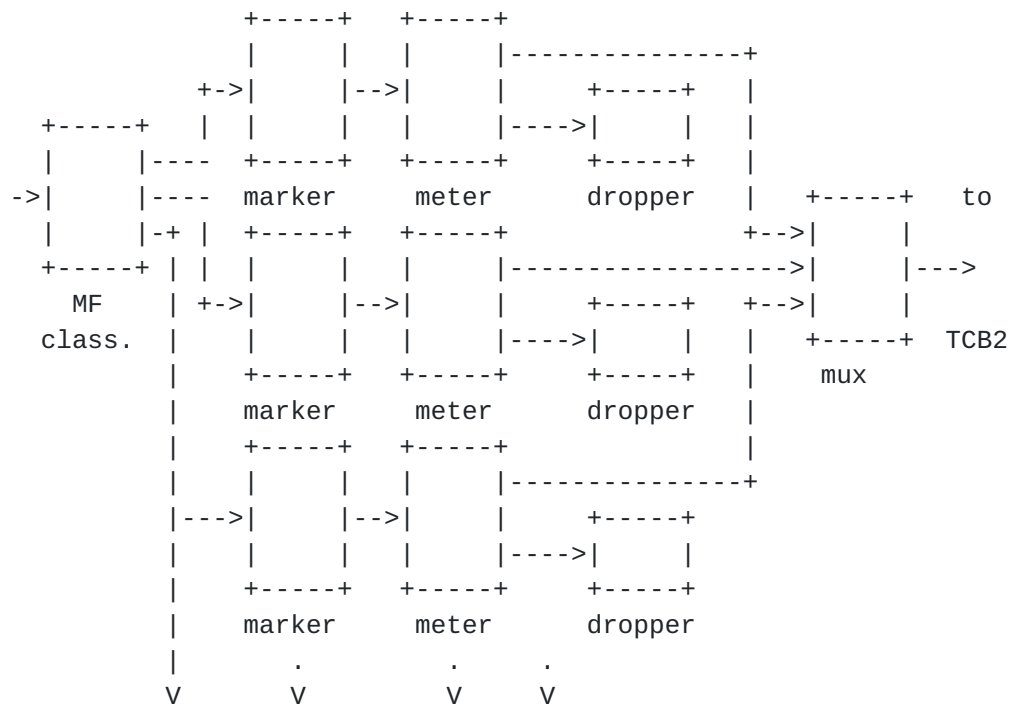

Figure 7: An Example of a Marking and Traffic Isolation TCB

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 (in this particular example we assume that one of two different DSCPs is marked). The metering stage limits 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.

The TCB could be formally specified as follows:

```
TCB1:
Classifier1: (MF)
Output A --> Marker1
Output B --> Marker2
Output C --> Marker3
. . .

Marker1 --> Meter1
Marker2 --> Meter2
Marker3 --> Meter3

Meter1:
Output A --> TCB2
Output B --> ActionElement1 (dropper)

Meter2:
Output A --> TCB2
Output B --> ActionElement2 (dropper)

Meter3:
Output A --> TCB2
Output B --> ActionElement3 (dropper)
```

The actual 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, illustrated below:

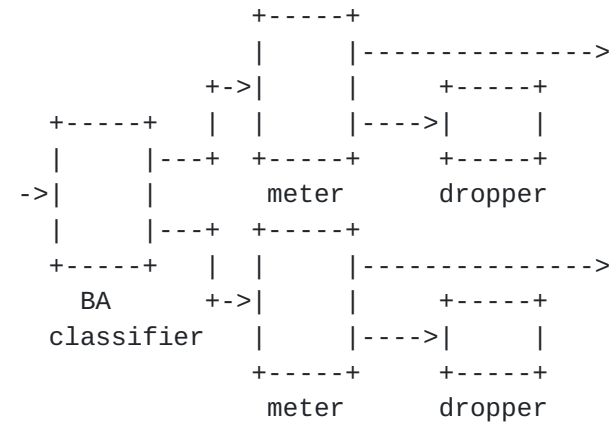

Figure 8: Additional Example TCB

TCB2 would be formally specified as follows:

```
Classifier2: (BA)
Output A --> Meter10
```

```
      Output B --> Meter11
```

```
Meter10:
Output A --> PHBQueueA
Output B --> Dropper10

Meter11:
Output A --> PHBQueueB
Output B --> Dropper11
```

8.4 Cascaded TCBs

Conceptually, nothing prevents more complex scenarios in which one microflow TCB precedes another (for example, TCBs implementing separate TCS's for the source and for a set of destinations).

9. Open Issues

- There is a difference in interpretation of token bucket behavior between this document (Appendix A) and [DSMIB]. Specifically, [DSMIB] allows a packet to conform if any smaller packet would conform.

- The meter in [SRTCM] cannot be precisely modeled using two two-parameter token buckets because its two buckets do not accumulate credits independently. We intended to demonstrate how the [TRTCM] meter could be implemented but ran out of time.

- Are the queue parameters (scheduling and buffer management) parameters defined sufficient?

- Does Queue and Queue Set really belong in the model (and the MIB and PIB?), or should the model stick to the abstract PHB representation and leave the implementation details to the MIB and PIB?

- Should a classifier be part of a TCB? We argue yes. This allows a TCB to be a one input/one output black box element.

- Is the description of a shaper sufficient? Is it overbroad?

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.

11. Acknowledgments

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

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

12. References

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

[DSTERMS] D. Grossman, "New Terminology for Diffserv", Internet Draft <draft-ietf-diffserv-new-terms-00.txt>, October 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-02.txt>, September 1999.

[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.

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

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

[DSMIB] F. Baker, "Differentiated Services MIB", Internet Draft <draft-ietf-diffserv-mib-00.txt>, June 1999.

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

[PIB] M. Fine, K. McCloghrie, J. Seligson, K. Chan, S. Hahn, and A. Smith, "Quality of Service Policy Information Base", Internet Draft <draft-mfine-cops-pib-01.txt>, June 1999.

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

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

[MPLSDS] J. Heinanen, "Differentiated Services in MPLS Networks", Internet Draft <draft-heinanen-diffserv-mpls-00.txt>, June 1999.

Appendix A. Simple Token Bucket Definition

[DSMIB] presents a fairly detailed exposition on the operation of two-parameter token buckets for metering. However, the behavior described does not appear to be consistent with the behavior defined in [SRTCM] and [TRTCM]. Specifically, under the definition in [DSMIB], a packet is assumed to conform to the meter if any of its bytes would have been accepted, while in [SRTCM] and [TRTCM], a packet is assumed to conform only if sufficient tokens are available for every byte in the packet. Further, a packet has no effect on the token occupancy if it does not conform (no tokens are decremented).

The behavior defined in [SRTCM] and [TRTCM] is not mandatory for compliance, but we give here a mathematical definition of two-parameter token bucket operation which is consistent with these documents, and which can be used to define a shaping profile.

Define a token bucket with bucket size BS, token accumulation rate R, and instantaneous token occupancy T(t). Assume that T(0) = BS.

Then after an arbitrary interval with no packet arrivals, T(t) will not change since the bucket is already full of tokens. Assume a packet of size B bytes at time t'. The bucket capacity T(t'-) = BS still. Then, as long as B <= BS, the packet conforms to the meter, and

$$T(t') = BS - B.$$

Assume an interval v = t - t' elapses before the next packet, of size C <= BS, arrives. T(t-) is given by the following equation:

$$T(t-) = \min \{ BS, T(t') + v*R \}$$

(the packet has accumulated v*R tokens over the interval, up to a maximum of BS tokens).

If T(t-) - C >= 0, the packet conforms and T(t) = T(t-) - C. Otherwise, the packet does not conform and T(t) = T(t-).

This function can be used to define a shaping profile. If a packet of size C arrives at time t, it will be eligible for transmission at time te given as follows (we still assume C <= BS):

$$te = \max \{ t, t" \}$$

where

$$t" = (C - T(t') + t'*R)/R.$$

T(t") = C, the time when C credits have accumulated in the bucket,
and when the packet would conform if the token bucket were a meter.
te != t" only if t > t".

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