Network Working Group
Request for Comments: 3670
Category: Standards Track

B. Moore
IBM Corporation
D. Durham
Intel
J. Strassner
INTELLIDEN, Inc.
A. Westerinen
Cisco Systems
W. Weiss
Ellacoya

January 2004

Information Model for Describing Network Device QoS Datapath Mechanisms

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2004). All Rights Reserved.

Abstract

The purpose of this document is to define an information model to describe the quality of service (QoS) mechanisms inherent in different network devices, including hosts. Broadly speaking, these mechanisms describe the properties common to selecting and conditioning traffic through the forwarding path (datapath) of a network device. This selection and conditioning of traffic in the datapath spans both major QoS architectures: Differentiated Services and Integrated Services.

This document should be used with the QoS Policy Information Model (QPIM) to model how policies can be defined to manage and configure the QoS mechanisms (i.e., the classification, marking, metering, dropping, queuing, and scheduling functionality) of devices. Together, these two documents describe how to write QoS policy rules to configure and manage the QoS mechanisms present in the datapaths of devices.

This document, as well as QPIM, are information models. That is, they represent information independent of a binding to a specific type of repository.

Table of Contents

1. Introduction 4
1.1. Policy Management Conceptual Model 6
1.2. Purpose and Relation to Other Policy Work. 7
1.3. Typical Examples of Policy Usage 7
2. Approach 8
2.1. Common Needs Of DiffServ and IntServ 8
2.2. Specific Needs Of DiffServ 9
2.3. Specific Needs Of IntServ. 9
3. Methodology. 10
3.1. Level of Abstraction for Expressing QoS Policies 10
3.2. Specifying Policy Parameters 11
3.3. Specifying Policy Services 12
3.4. Level of Abstraction for Defining QoS Attributes and Classes. 13
3.5. Characterization of QoS Properties 14
3.6. QoS Information Model Derivation 15
3.7. Attribute Representation 16
3.8. Mental Model 17
3.8.1. The QoSService Class 17
3.8.2. The ConditioningService Class. 18
3.8.3. Preserving QoS Information from Ingress to Egress 19
3.9. Classifiers, FilterLists, and Filter Entries 21
3.10. Modeling of Droppers 23
3.10.1. Configuring Head and Tail Droppers 23
3.10.2. Configuring RED Droppers 24
3.11. Modeling of Queues and Schedulers. 25
3.11.1. Simple Hierarchical Scheduler. 25
3.11.2. Complex Hierarchical Scheduler 27
3.11.3. Excess Capacity Scheduler. 29
3.11.4. Hierarchical CBQ Scheduler 31
4. The Class Hierarchy. 33
4.1. Associations and Aggregations. 33
4.2. The Structure of the Class Hierarchies 34
4.3. Class Definitions. 38
4.3.1. The Abstract Class ManagedElement. 38
4.3.2. The Abstract Class ManagedSystemElement. 39
4.3.3. The Abstract Class LogicalElement. 39
4.3.4. The Abstract Class Service 39
4.3.5. The Class ConditioningService. 39
4.3.6. The Class ClassifierService. 40
4.3.7. The Class ClassifierElement. 41

4.3.8. The Class MeterService 42
4.3.9. The Class AverageRateMeterService. 44
4.3.10. The Class EWMAMeterService 44
4.3.11. The Class TokenBucketMeterService. 46
4.3.12. The Class MarkerService. 47
4.3.13. The Class PreambleMarkerService. 47
4.3.14. The Class ToSMarkerService 48
4.3.15. The Class DSCPMarkerService. 49
4.3.16. The Class 8021QMarkerService 49
4.3.17. The Class DropperService 50
4.3.18. The Class HeadTailDropperService 52
4.3.19. The Class REDDropperService. 52
4.3.20. The Class QueuingService 54
4.3.21. The Class PacketSchedulingService. 55
4.3.22. The Class NonWorkConservingSchedulingService . . 56
4.3.23. The Class QoSService 57
4.3.24. The Class DiffServService. 58
4.3.25. The Class AFService. 59
4.3.26. The Class FlowService. 60
4.3.27. The Class DropThresholdCalculationService. . . . 60
4.3.28. The Abstract Class FilterEntryBase 61
4.3.29. The Class IPHeaderFilter 62
4.3.30. The Class 8021Filter 62
4.3.31. The Class PreambleFilter 62
4.3.32. The Class FilterList 63
4.3.33. The Abstract Class ServiceAccessPoint. 63
4.3.34. The Class ProtocolEndpoint 63
4.3.35. The Abstract Class Collection. 65
4.3.36. The Abstract Class CollectionOfMSEs. 65
4.3.37. The Class BufferPool 65
4.3.38. The Abstract Class SchedulingElement 65
4.3.39. The Class AllocationSchedulingElement. 66
4.3.40. The Class WRRSchedulingElement 67
4.3.41. The Class PrioritySchedulingElement. 69
4.3.42. The Class BoundedPrioritySchedulingElement . . . 70
4.4. Association Definitions. 70
4.4.1. The Abstract Association Dependency. 71
4.4.2. The Association ServiceSAPDependency 71
4.4.3. The Association IngressConditioningServiceOnEndpoint 71
4.4.4. The Association EgressConditioningServiceOnEndpoint. 72
4.4.5. The Association HeadTailDropQueueBinding 72
4.4.6. The Association CalculationBasedOnQueue. 73
4.4.7. The Association ProvidesServiceToElement 74
4.4.8. The Association ServiceServiceDependency 74
4.4.9. The Association CalculationServiceForDropper . . 75
4.4.10. The Association QueueAllocation. 75

4.4.11. The Association ClassifierElementUsesFilterList. 76
4.4.12. The Association AFRelatedServices. 77
4.4.13. The Association NextService. 78
4.4.14. The Association NextServiceAfterClassifierElement. 79
4.4.15. The Association NextScheduler. 80
4.4.16. The Association FailNextScheduler. 81
4.4.17. The Association NextServiceAfterMeter. 82
4.4.18. The Association QueueToSchedule. 83
4.4.19. The Association SchedulingServiceToSchedule. . . 84
4.4.20. The Aggregation MemberOfCollection 85
4.4.21. The Aggregation CollectedBufferPool. 85
4.4.22. The Abstract Aggregation Component 86
4.4.23. The Aggregation ServiceComponent 86
4.4.24. The Aggregation QoSSubService. 86
4.4.25. The Aggregation QoSConditioningSubService. . . . 87
4.4.26. The Aggregation ClassifierElementInClassifierService 88
4.4.27. The Aggregation EntriesInFilterList. 89
4.4.28. The Aggregation ElementInSchedulingService . . . 90
5. Intellectual Property Statement. 91
6. Acknowledgements 91
7. Security Considerations. 91
8. References 92
8.1. Normative References. 92
8.2. Informative References 92
9. Appendix A: Naming Instances in a Native CIM Implementation . 94
9.1. Naming Instances of the Classes Derived from Service. . . 94
9.2. Naming Instances of Subclasses of FilterEntryBase 94
9.3. Naming Instances of ProtocolEndpoint. 94
9.4. Naming Instances of BufferPool. 95
9.4.1. The Property CollectionID. 95
9.4.2. The Property CreationClassName 95
9.5. Naming Instances of SchedulingElement 95
10. Authors' Addresses 96
11. Full Copyright Statement 97

1. Introduction

The purpose of this document is to define an information model to describe the quality of service (QoS) mechanisms inherent in different network devices, including hosts. Broadly speaking, these mechanisms describe the attributes common to selecting and conditioning traffic through the forwarding path (datapath) of a network device. This selection and conditioning of traffic in the datapath spans both major QoS architectures: Differentiated Services (see [R2475]) and Integrated Services (see [R1633]).

This document is intended to be used with the QoS Policy Information Model [QPIM] to model how policies can be defined to manage and configure the QoS mechanisms (i.e., the classification, marking, metering, dropping, queuing, and scheduling functionality) of devices. Together, these two documents describe how to write QoS policy rules to configure and manage the QoS mechanisms present in the datapaths of devices.

This document, as well as [QPIM], are information models. That is, they represent information independent of a binding to a specific type of repository. A separate document could be written to provide a mapping of the data contained in this document to a form suitable for implementation in a directory that uses (L)DAP as its access protocol. Similarly, a document could be written to provide a mapping of the data in [QPIM] to a directory. Together, these four documents (information models and directory schema mappings) would then describe how to write QoS policy rules that can be used to store information in directories to configure device QoS mechanisms.

The approach taken in this document defines a common set of classes that can be used to model QoS in a device datapath. Vendors can then map these classes, either directly or using an intervening format like a COP-PR PIB, to their own device-specific implementations. Note that the admission control element of Integrated Services is not included in the scope of this model.

The design of the class, association, and aggregation hierarchies described in this document is influenced by the Network QoS submodel defined by the Distributed Management Task Force (DMTF) - see [CIM]. These hierarchies are not derived from the Policy Core Information Model [PCIM]. This is because the modeling of the QoS mechanisms of a device is separate and distinct from the modeling of policies that manage those mechanisms. Hence, there is a need to separate QoS mechanisms (this document) from their control (specified using the generic policy document [PCIM] augmented by the QoS Policy document [QPIM]).

While it is not a policy model per se, this document does have a dependency on the Policy Core Information Model Extensions document [PCIME]. The device-level packet filtering, through which a Classifier splits a traffic stream into multiple streams, is based on the FilterEntryBase and FilterList classes defined in [PCIME].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [R2119].

1.1. Policy Management Conceptual Model

The Policy Core Information Model [PCIM] describes a general methodology for constructing policy rules. PCIM Extensions [PCIME] updates and extends the original PCIM. A policy rule aggregates a set of policy conditions and an ordered set of policy actions. The semantics of a policy rule are such that if the set of conditions evaluates to TRUE, then the set of actions are executed.

Policy conditions and actions have two principal components: operands and operators. Operands can be constants or variables. To specify a policy, it is necessary to specify:

- the operands to be examined (also known as state variables);
- the operands to be changed (also known as configuration variables);
- the relationships between these two sets of operands.

Operands can be specified at a high-level, such as Joe (a user) or Gold (a service). Operands can also be specified at a much finer level of detail, one that is much closer to the operation of the device. Examples of the latter include an IP Address or a queue's bandwidth allocation. Implicit in the use of operands is the binding of legal values or ranges of values to an operand. For example, the value of an IP address cannot be an integer. The concepts of operands and their ranges are defined in [PCIME].

The second component of policy conditions and actions is a set of operators. Operators can express both relationships (greater than, member of a set, Boolean OR, etc.) and assignments. Together, operators and operands can express a variety of conditions and actions, such as:

```
If Bob is an Engineer...
If the source IP address is in the Marketing Subnet...
Set Joe's IP address to 192.0.2.100
Limit the bandwidth of application x to 10 Mb
```

We recognize that the definition of operator semantics is critical to the definition of policies. However, the definition of these operators is beyond the scope of this document. Rather, this document (with [QPIM]) takes the first steps in identifying and standardizing a set of properties (operands) for use in defining policies for Differentiated and Integrated Services.

1.2. Purpose and Relation to Other Policy Work

This model establishes a canonical model of the QoS mechanisms of a network device (e.g., a router, switch, or host) that is independent of any specific type of network device. This enables traffic conditioning to be described using a common set of abstractions, modeled as a set of services and sub-services.

When the concepts of this document are used in conjunction with the concepts of [QPIM], one is able to define policies that bind the services in a network to the needs of applications using that network. In other words, the business requirements of an organization can be reflected in one set of policies, and those policies can be translated to a lower-level set of policies that control and manage the configuration and operation of network devices.

1.3. Typical Examples of Policy Usage

Policies could be implemented as low-level rules using the information model described in this specification. For example, in a low-level policy, a condition could be represented as an evaluation of a specific attribute from this model. Therefore, a condition such as "If filter = HTTP" would be interpreted as a test determining whether any HTTP filters have been defined for the device. A high-level policy, such as "If protocol = HTTP, then mark with Differentiated Services Code Point (DSCP) 24," would be expressed as a series of actions in a low-level policy using the classes and attributes described below:

1. Create HTTP filter
2. Create DSCP marker with the value of 24
3. Bind the HTTP filter to the DSCP marker

Note that unlike "mark with DSCP 24," these low-level actions are not performed on a packet as it passes through the device. Rather, they are configuration actions performed on the device itself, to make it ready to perform the correct action(s) on the correct packet(s). The act of moving from a high-level policy rule to the correct set of low-level device configuration actions is an example of what [POLTERM] characterizes as "policy translation" or "policy conversion".

2. Approach

QoS activities in the IETF have mainly focused in two areas, Integrated Services (IntServ) and Differentiated Services (DiffServ) (see [POLTERM], [R1633] and [R2475]). This document focuses on the specification of QoS properties and classes for modeling the datapath where packet traffic is conditioned. However, the framework defined by the classes in this document has been designed with the needs of the admission control portion of IntServ in mind as well.

2.1. Common Needs Of DiffServ and IntServ

First, let us consider IntServ. IntServ has two principal components. One component is embedded in the datapath of the networking device. Its functions include the classification and policing of individual flows, and scheduling admitted packets for the outbound link. The other component of IntServ is admission control, which focuses on the management of the signaling protocol (e.g., the PATH and RESV messages of RSVP). This component processes reservation requests, manages bandwidth, outsources decision making to policy servers, and interacts with the Routing Table manager.

We will consider RSVP when defining the structure of this information model. As this document focuses on the datapath, elements of RSVP applicable to the datapath will be considered in the structure of the classes. The complete IntServ device model will, as we have indicated earlier, be addressed in a subsequent document.

This document models a small subset of the QoS policy problem, in hopes of constructing a methodology that can be adapted for other aspects of QoS in particular, and of policy construction in general. The focus in this document is on QoS for devices that implement traffic conditioning in the datapath.

DiffServ operates exclusively in the datapath. It has all of the same components of the IntServ datapath, with two major differences. First, DiffServ classifies packets based solely on their DSCP field, whereas IntServ examines a subset of a standard flow’s addressing 5-tuple. The exception to this rule occurs in a router or host at the boundary of a DiffServ domain. A device in this position may examine a packet’s DSCP, its addressing 5-tuple, other fields in the packet, or even information wholly outside the packet, in determining the DSCP value with which to mark the packet prior to its transfer into the DiffServ domain. However, routers in the interior of a DiffServ domain will only need to classify based on the DSCP field.

The second difference between IntServ and DiffServ is that the signaling protocol used in IntServ (e.g., RSVP) affects the configuration of the datapath in a more dynamic fashion. This is because each newly admitted RSVP reservation requires a reconfiguration of the datapath. In contrast, DiffServ requires far fewer changes to the datapath after the Per Hop Behaviors (PHBs) have been configured.

The approach advocated in this document for the creation of policies that control the various QoS mechanisms of networking devices is to first identify the attributes with which policies are to be constructed. These attributes are the parameters used in expressions that are necessary to construct policies. There is also a parallel desire to define the operators, relations, and precedence constructs necessary to construct the conditions and actions that constitute these policies. However, these efforts are beyond the scope of this document.

2.2. Specific Needs Of DiffServ

DiffServ-specific rules focus on two particular areas: the core and the edges of the network. As explained in the DiffServ Architecture document [R2475], devices at the edge of the network classify traffic into different traffic streams. The core of the network then forwards traffic from different streams by using a set of Per Hop Behaviors (PHBs). A DSCP identifies each PHB. The DSCP is part of the IP header of each packet (as described in [R2474]). This enables multiple traffic streams to be aggregated into a small number of aggregated traffic streams, where each aggregate traffic stream is identified by a particular DSCP, and forwarded using a particular PHB.

The attributes used to manipulate QoS capabilities in the core of the network primarily address the behavioral characteristics of each supported PHB. At the edges of the DiffServ network, the additional complexities of flow classification, policing, RSVP mappings, remarkings, and other factors have to be considered. Additional modeling will be required in this area. However, first, the standards for edges of the DiffServ network need more detail - to allow the edges to be incorporated into the policy model.

2.3. Specific Needs Of IntServ

This document focuses exclusively on the forwarding aspects of network QoS. Therefore, while the forwarding aspects of IntServ are considered, the management of IntServ is not considered. This topic will be addressed in a future document.

3. Methodology

There is a clear need to define attributes and behavior that together define how traffic should be conditioned. This document defines a set of classes and relationships that represent the QoS mechanisms used to condition traffic; [QPIM] is used to define policies to control the QoS mechanisms defined in this document.

However, some very basic issues need to be considered when combining these documents. Considering these issues should help in constructing a schema for managing the operation and configuration of network QoS mechanisms through the use of QoS policies.

3.1. Level of Abstraction for Expressing QoS Policies

The first issue requiring consideration is the level of abstraction at which QoS policies should be expressed. If we consider policies as a set of rules used to react to events and manipulate attributes or generate new events, we realize that policy represents a continuum of specifications that relate business goals and rules to the conditioning of traffic done by a device or a set of devices. An example of a business level policy might be: from 1:00 pm PST to 7:00 am EST, sell off 40% of the network capacity on the open market. In contrast, a device-specific policy might be: if the queue depth grows at a geometric rate over a specified duration, trigger a potential link failure event.

A general model for this continuum is shown in Figure 1 below.

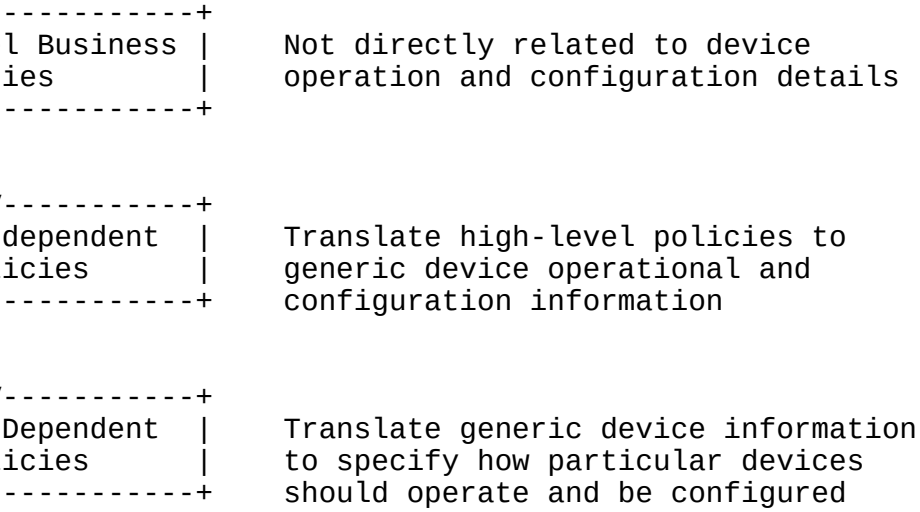


Figure 1. The Policy Continuum

High-level business policies are used to express the requirements of the different applications, and prioritize which applications get "better" treatment when the network is congested. The goal, then, is to use policies to relate the operational and configuration needs of a device directly to the business rules that the network administrator is trying to implement in the network that the device belongs to.

Device-independent policies translate business policies into a set of generalized operational and configuration policies that are independent of any specific device, but dependent on a particular set of QoS mechanisms, such as random early detection (RED) dropping or weighted round robin scheduling. Not only does this enable different types of devices (routers, switches, hosts, etc.) to be controlled by QoS policies, it also enables devices made by different vendors that use the same types of QoS mechanisms to be controlled. This enables these different devices to each supply the correct relative conditioning to the same type of traffic.

In contrast, device-dependent policies translate device-independent policies into ones that are specific for a given device. The reason that a distinction is made between device-independent and device-dependent policies is that in a given network, many different devices having many different capabilities need to be controlled together. Device-independent policies provide a common layer of abstraction for managing multiple devices of different capabilities, while device-dependent policies implement the specific conditioning that is required. This document provides a common set of abstractions for representing QoS mechanisms in a device-independent way.

This document is focused on the device-independent representation of QoS mechanisms. QoS mechanisms are modeled in sufficient detail to provide a common device-independent representation of QoS policies. They can also be used to provide a basis for specialization, enabling each vendor to derive a set of vendor-specific classes that represent how traffic conditioning is done for that vendor’s set of devices.

3.2. Specifying Policy Parameters

Policies are a function of parameters (attributes) and operators (boolean, arithmetic, relational, etc.). Therefore, both need to be defined as part of the same policy in order to correctly condition the traffic. If the parameters of the policy are specified too narrowly, they will reflect the individual implementations of QoS in each device. As there is currently little consensus in the industry on what the correct implementation model for QoS is, most defined attributes would only be applicable to the unique characteristics of a few individual devices. Moreover, standardizing all of these

potential implementation alternatives would be a never-ending task as new implementations continued to appear on the market.

On the other hand, if the parameters of the policy are specified too broadly, it is impossible to develop meaningful policies. For example, if we concentrate on the so-called Olympic set of policies, a business policy like "Bob gets Gold Service," is clearly meaningless to the large majority of existing devices. This is because the device has no way of determining who Bob is, or what QoS mechanisms should be configured in what way to provide Gold service.

Furthermore, Gold service may represent a single service, or it may identify a set of services that are related to each other. In the latter case, these services may have different conditioning characteristics.

This document defines a set of parameters that fit into a canonical model for modeling the elements in the forwarding path of a device implementing QoS traffic conditioning. By defining this model in a device-independent way, the needed parameters can be appropriately abstracted.

3.3. Specifying Policy Services

Administrators want the flexibility to be able to define traffic conditioning without having to have a low-level understanding of the different QoS mechanisms that implement that conditioning. Furthermore, administrators want the flexibility to group different services together, describing a higher-level concept such as "Gold Service". This higher-level service could be viewed as providing the processing to deliver "Gold" quality of service.

These two goals dictate the need for the following set of abstractions:

- a flexible way to describe a service
- must be able to group different services that may use different technologies (e.g., DiffServ and IEEE 802.1Q) together
- must be able to define a set of sub-services that together make up a higher-level service
- must be able to associate a service and the set of QoS mechanisms that are used to condition traffic for that service
- must be able to define policies that manage the QoS mechanisms used to implement a service.

This document addresses this set of problems by defining a set of classes and associations that can represent abstract concepts like "Gold Service," and bind each of these abstract services to a specific set of QoS mechanisms that implement the conditioning that they require. Furthermore, this document defines the concept of "sub-services," to enable Gold Service to be defined either as a single service or as a set of services that together should be treated as an atomic entity.

Given these abstractions, policies (as defined in [QPIM]) can be written to control the QoS mechanisms and services defined in this document.

3.4. Level of Abstraction for Defining QoS Attributes and Classes

This document defines a set of classes and properties to support policies that configure device QoS mechanisms. This document concentrates on the representation of services in the datapath that support both DiffServ (for aggregate traffic conditioning) and IntServ (for flow-based traffic conditioning). Classes and properties for modeling IntServ admission control services may be defined in a future document.

The classes and properties in this document are designed to be used in conjunction with the QoS policy classes and properties defined in [QPIM]. For example, to preserve the delay characteristics committed to an end-user, a network administrator may wish to create policies that monitor the queue depths in a device, and adjust resource allocations when delay budgets are at risk (perhaps as a result of a network topology change). The classes and properties in this document define the specific services and mechanisms required to implement those services. The classes and properties defined in [QPIM] provide the overall structure of the policy that manages and configures this service.

This combination of low-level specification (using this document) and high-level structuring (using [QPIM]) of network services enables network administrators to define new services required of the network, that are directly related to business goals, while ensuring that such services can be managed. However, this goal (of creating and managing service-oriented policies) can only be realized if policies can be constructed that are capable of supporting diverse implementations of QoS. The solution is to model the QoS capabilities of devices at the behavioral level. This means that for traffic conditioning services realized in the datapath, the model must support the following characteristics:

- modeling of a generic network service that has QoS capabilities

- o modeling of how the traffic conditioning itself is defined
- o modeling of how statistics are gathered to monitor QoS traffic conditioning services - this facet of the model will be added in a future document.

This document models a network service, and associates it with one or more QoS mechanisms that are used to implement that service. It also models in a canonical form the various components that are used to condition traffic, such that standard as well as custom traffic conditioning services may be described.

3.5. Characterization of QoS Properties

The QoS properties and classes will be described in more detail in Section 4. However, we should consider the basic characteristics of these properties, to understand the methodology for representing them.

There are essentially two types of properties, state and configuration. Configuration properties describe the desired state of a device, and include properties and classes for representing desired or proposed thresholds, bandwidth allocations, and how to classify traffic. State properties describe the actual state of the device. These include properties to represent the current operational values of the attributes in devices configured via the configuration properties, as well as properties that represent state (queue depths, excess capacity consumption, loss rates, and so forth).

In order to be correlated and used together, these two types of properties must be modeled using a common information model. The possibility of modeling state properties and their corresponding configuration settings is accomplished using the same classes in this model - although individual instances of the classes would have to be appropriately named or placed in different containers to distinguish current state values from desired configuration settings.

State information is addressed in a very limited fashion by QDDIM. Currently, only CurrentQueueDepth is proposed as an attribute on QueuingService. The majority of the model is related to configuration. Given this fact, it is assumed that this model is a direct memory map into a device. All manipulation of model classes and properties directly affects the state of the device. If it is desired to also use these classes to represent desired configuration, that is left to the discretion of the implementor.

It is acknowledged that additional properties are needed to completely model current state. However, many of the properties defined in this document represent exactly the state variables that will be configured by the configuration properties. Thus, the definition of the configuration properties has an exact correspondence with the state properties, and can be used in modeling both actual (state) and desired/proposed configuration.

3.6. QoS Information Model Derivation

The question of context also leads to another question: how does the information specified in the core and QoS policy models ([PCIM], [PCIME], and [QPIM], respectively) integrate with the information defined in this document? To put it another way, where should device-independent concepts that lead to device-specific QoS attributes be derived from?

Past thinking was that QoS was part of the policy model. This view is not completely accurate, and it leads to confusion. QoS is a set of services that can be controlled using policy. These services are represented as device mechanisms. An important point here is that QoS services, as well as other types of services (e.g., security), are provided by the mechanisms inherent in a given device. This means that not all devices are indeed created equal. For example, although two devices may have the same type of mechanism (e.g., a queue), one may be a simple implementation (i.e., a FIFO queue) whereas one may be much more complex and robust (e.g., class-based weighted fair queuing (CBWFQ)). However, both of these devices can be used to deliver QoS services, and both need to be controlled by policy. Thus, a device-independent policy can instruct the devices to queue certain traffic, and a device-specific policy can be used to control the queuing in each device.

Furthermore, policy is used to control these mechanisms, not to represent them. For example, QoS services are implemented with classifiers, meters, markers, droppers, queues, and schedulers. Similarly, security is also a characteristic of devices, as authentication and encryption capabilities represent services that networked devices perform (irrespective of interactions with policy servers). These security services may use some of the same mechanisms that are used by QoS services, such as the concepts of filters. However, they will mostly require different mechanisms than the ones used by QoS, even though both sets of services are implemented in the same devices.

Thus, the similarity between the QoS model and models for other services is not so much that they contain a few common mechanisms. Rather, they model how a device implements their respective services.

As such, the modeling of QoS should be part of a networking device schema rather than a policy schema. This allows the networking device schema to concentrate on modeling device mechanisms, and the policy schema to focus on the semantics of representing the policy itself (conditions, actions, operators, etc.). While this document concentrates on defining an information model to represent QoS services in a device datapath, the ultimate goal is to be able to apply policies that control these services in network devices. Furthermore, these two schemata (device and policy) must be tightly integrated in order to enable policy to control QoS services.

3.7. Attribute Representation

The last issue to be considered is the question of how attributes are represented. If QoS attributes are represented as absolute numbers (e.g., Class AF2 gets 2 Mbs of bandwidth), it is more difficult to make them uniform across multiple ports in a device or across multiple devices, because of the broad variation in link capacities. However, expressing attributes in relative or proportional terms (e.g., Class AF2 gets 5% of the total link bandwidth) makes it more difficult to express certain types of conditions and actions, such as:

```
(If ConsumedBandwidth = AssignedBandwidth Then ...)
```

There are really three approaches to addressing this problem:

- Multiple properties can be defined to express the same value in various forms. This idea has been rejected because of the difficulty in keeping these different properties synchronized (e.g., when one property changes, the others all have to be updated).

- Multi-modal properties can be defined to express the same value, in different terms, based on the access or assignment mode. This option was rejected because it significantly complicates the model and is impossible to express in current directory access protocols (e.g., (L)DAP).

- Properties can be expressed as "absolutes", but the operators in the policy schema would need to be more sophisticated. Thus, to represent a percentage, division and multiplication operators are required (e.g., Class AF2 gets .05 * the total link bandwidth). This is the approach that has been taken in this document.

3.8. Mental Model

The mental model for constructing this schema is based on the work done in the Differentiated Services working group. This schema is based on information provided in the current versions of the DiffServ Informal Management Model [DSMODEL], the DiffServ MIB [DSMIB], the PIB [PIB], as well as on information in the set of RFCs that constitute the basic definition of DiffServ itself ([R2475], [R2474], [R2597], and [R3246]). In addition, a common set of terminology is available in [POLTERM].

This model is built around two fundamental class hierarchies that are bound together using a set of associations. The two class hierarchies derive from the QoSService and ConditioningService base classes. A set of associations relate lower-level QoSService subclasses to higher-level QoS services, relate different types of conditioning services together in processing a traffic class, and relate a set of conditioning services to a specific QoS service. This combination of associations enables us to view the device as providing a set of services that can be configured, in a modular building block fashion, to construct application-specific services. Thus, this document can be used to model existing and future standard as well as application-specific network QoS services.

3.8.1. The QoSService Class

The first of the classes defined here, QoSService, is used to represent higher-level network services that require special conditioning of their traffic. An instance of QoSService (or one of its subclasses) is used to bring together a group of conditioning services that, from the perspective of the system manager, are all used to deliver a common service. Thus, the set of classifiers, markers, and related conditioning services that provide premium service to the "selected" set of user traffic may be grouped together into a premium QoS service.

QoSService has a set of subclasses that represent different approaches to delivering IP services. The currently defined set of subclasses are a FlowService for flow-oriented QoS delivery and a DiffServService for DiffServ aggregate-oriented QoS service delivery.

The QoS services can be related to each other as peers, or they can be implemented as subservient services to each other. The QoSSubService aggregation indicates that one or more QoSService objects are subservient to a particular QoSService object. For example, this enables us to define Gold Service as a combination of two DiffServ services, one for high quality traffic treatment, and one for servicing the rest of the traffic. Each of these

DiffServService objects would be associated with a set of classifiers, markers, etc, such that the high quality traffic would get EF marking and appropriate queuing.

The DiffServService class itself has an AFService subclass. This subclass is used to represent the specific notion that several related markings within the AF PHB Group work together to provide a single service. When other DiffServ PHB Groups are defined that use more than one code point, these will be likely candidates for additional DiffServService subclasses.

Technology-specific mappings of these services, representing the specific use of PHB marking or 802.1Q marking, are captured within the ConditioningService hierarchy, rather than in the subclasses of QoSService.

These concepts are depicted in Figure 2. Note that both of the associations are aggregations: a QoSService object aggregates both the set of QoSService objects subservient to it, and the set of ConditioningService objects that realize it. See Section 4 for class and association definitions.

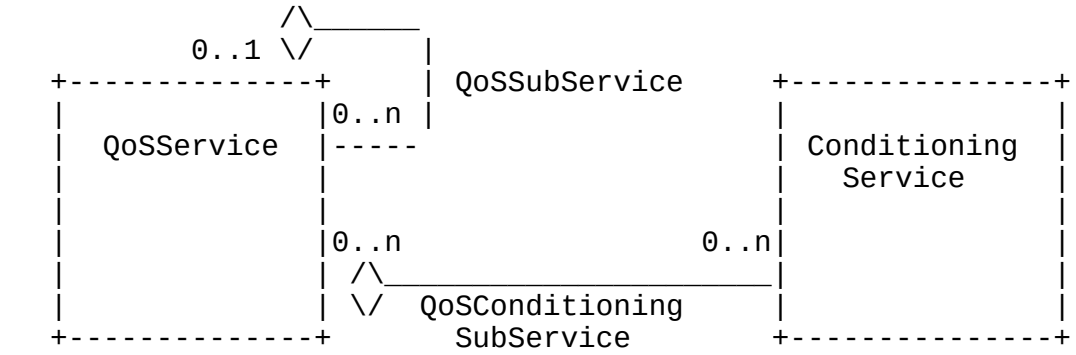


Figure 2. QoSService and its Aggregations

3.8.2. The ConditioningService Class

The goal of the ConditioningService classes is to describe the sequence of traffic conditioning that is applied to a given traffic stream on the ingress interface through which it enters a device, and then on the egress interface through which it leaves the device. This is done using a set of classes and relationships. The routing decision in the device core, which selects which egress interface a particular packet will use, is not represented in this model.

A single base class, ConditioningService, is the superclass for a set of subclasses representing the mechanisms that condition traffic.

These subclasses define device-independent conditioning primitives (including classifiers, meters, markers, droppers, queues, and schedulers) that together implement the conditioning of traffic on an interface. This model abstracts these services into a common set of modular building blocks that can be used, regardless of device implementation, to model the traffic conditioning internal to a device.

The different conditioning mechanisms need to be related to each other to describe how traffic is conditioned. Several important variations of how these services are related together exist:

- o A particular ingress or egress interface may not require all the types of ConditioningServices.
- o Multiple instances of the same mechanism may be required on an ingress or egress interface.
- o There is no set order of application for the ConditioningServices on an ingress or egress interface.

Therefore, this model does not dictate a fixed ordering among the subclasses of ConditioningService, or identify a subclass of ConditioningService that must appear first or last among the ConditioningServices on an ingress or egress interface. Instead, this model ties together the various ConditioningService instances on an ingress or egress interface using the NextService, NextServiceAfterMeter, and NextServiceAfterConditioningElement associations. There are also separate associations, called IngressConditioningServiceOnEndpoint and EgressConditioningServiceOnEndpoint, which, respectively, tie an ingress interface to its first ConditioningService, and tie an egress interface to its last ConditioningService(s).

3.8.3. Preserving QoS Information from Ingress to Egress

There is one important way in which the QDDIM model diverges from the [DSMODEL]. In [DSMODEL], traffic passes through a network device in three stages:

- o It comes in on an ingress interface, where it may receive QoS conditioning.
- o It traverses the routing core, where logic outside the scope of QoS determines which egress interface it will use to leave the device.

o It may receive further QoS conditioning on the selected egress interface, and then it leaves the device.

In this model, no information about the QoS conditioning that a packet receives on the ingress interface is communicated with the packet across the routing core to the egress interface.

The QDDIM model relaxes this restriction, to allow information about the treatment that a packet received on an ingress interface to be communicated along with the packet to the egress interface. (This relaxation adds a capability that is present in many network devices.) QDDIM represents this information transfer in terms of a packet preamble, which is how many devices implement it. But implementations are free to use other mechanisms to achieve the same result.

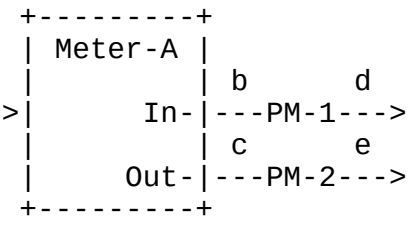


Figure 3: Meter Followed by Two Preamble Markers

Figure 3 shows an example in which meter results are captured in a packet preamble. The arrows labeled with single letters represent instances of either the NextService association (a, d, and e), or of its peer association NextServiceAfterMeter (b and c). PreambleMarker PM-1 adds to the packet preamble an indication that the packet exited Meter A as conforming traffic. Similarly, PreambleMarker PM-2 adds to the preambles of packets that come through it indications that they exited Meter A as nonconforming traffic. A PreambleMarker appends its information to whatever is already present in a packet preamble, as opposed to overwriting what is already there.

To foster interoperability, the basic format of the information captured by a PreambleMarker is specified. (Implementations, of course, are free to represent this information in a different way internally - this is just how it is represented in the model.) The information is represented by an ordered, multi-valued string property FilterItemList, where each individual value of the property is of the form "<type>,<value>". When a PreambleMarker "appends" its information to the information that was already present in a packet preamble, it does so by adding additional items of the indicated format to the end of the list.

QDDIM provides a limited set of <type>'s that a PreambleMarker may use:

- ConformingFromMeter: the value is the name of the meter.
- PartConformingFromMeter: the value is the name of the meter.
- NonConformingFromMeter: the value is the name of the meter.
- VlanId: the value is the virtual LAN identifier (VLAN ID).

Implementations may recognize other <type>'s in addition to these. If collisions of implementation-specific <type>'s become a problem, it is possible that <type>'s may become an IANA-administered range in a future revision of this document.

To make use of the information that a PreambleMarker stores in a packet preamble, a specific subclass PreambleFilter of FilterEntryBase is defined, to match on the "<type>,<value>" strings. To simplify the case where there's just a single level of metering in a device, but different individual meters on each ingress interface, PreambleFilter allows a wildcard "any" for the <value> part of the three meter-related filters. With this wildcard, an administrator can specify a Classifier to select all packets that were found to be conforming (or partially conforming, or non-conforming) by their respective meters, without having to name each meter individually in a separate ClassifierElement.

Once a meter result has been stored in a packet preamble, it is available for any subsequent Classifier to use. So while the motivation for this capability has been described in terms of preserving QoS conditioning information from an ingress interface to an egress interface, a prior meter result may also be used for classifying packets later in the datapath on the same interface where the meter resides.

3.9. Classifiers, FilterLists, and Filter Entries

This document uses a number of classes to model the classifiers defined in [DSMODEL]: ClassifierService, ClassifierElement, FilterList, FilterEntryBase, and various subclasses of FilterEntryBase. There are also two associations involved: ClassifierElementUsesFilterList and EntriesInFilterList. The QDDIM model makes no use of CIM's FilterEntry class.

In [DSMODEL], a single traffic stream coming into a classifier is split into multiple traffic streams leaving it, based on which of an ordered set of filters each packet in the incoming stream matches. A

filter matches either a field in the packet itself, or possibly other attributes associated with the packet. In the case of a multi-field (MF) classifier, packets are assigned to output streams based on the contents of multiple fields in the packet header. For example, an MF classifier might assign packets to an output stream based on their complete IP-addressing 5-tuple.

To optimize the representation of MF classifiers, subclasses of FilterEntryBase are introduced, which allow multiple related packet header fields to be represented in a single object. These subclasses are IPHeaderFilter and 8021Filter. With IPHeaderFilter, for example, criteria for selecting packets based on all five of the IP 5-tuple header fields and the DiffServ DSCP can be represented by a FilterList containing one IPHeaderFilter object. Because these two classes have applications beyond those considered in this document, they, as well as the abstract class FilterEntryBase, are defined in the more general document [PCIME] rather than here.

The FilterList object is always needed, even if it contains only one filter entry (that is, one FilterEntryBase subclass) object. This is because a ClassifierElement can only be associated with a Filter List, as opposed to an individual FilterEntry. FilterList is also defined in [PCIME].

The EntriesInFilterList aggregation (also defined in [PCIME]) has a property EntrySequence, which in the past (in CIM) could be used to specify an evaluation order on the filter entries in a FilterList. Now, however, the EntrySequence property supports only a single value: '0'. This value indicates that the FilterEntries are ANDed together to determine whether a packet matches the MF selector that the FilterList represents.

A ClassifierElement specifies the starting point for a specific policy or data path. Each ClassifierElement uses the NextServiceAfterClassifierElement association to determine the next conditioning service to apply for packets to.

A ClassifierService defines a grouping of ClassifierElements. There are certain instances where a ClassifierService actually specifies an aggregation of ClassifierServices. One practical case would be where each ClassifierService specifies a group of policies associated with a particular application and another ClassifierService groups the application-specific ClassifierService instances. In this particular case, the application-specific ClassifierService instances are specified once, but unique combinations of these ClassifierServices are specified, as needed, using other ClassifierService instances. ClassifierService instances grouping other ClassifierService instances may not specify a FilterList using the

ClassifierElementUsesFilterList association. This special use of ClassifierService serves just as a Classifier collecting function.

3.10. Modeling of Droppers

In [DSMODEL], a distinction is made between absolute droppers and algorithmic droppers. In QDDIM, both of these types of droppers are modeled with the DropperService class, or with one of its subclasses. In both cases, the queue from which the dropper drops packets is tied to the dropper by an instance of the NextService association. The dropper always plays the PrecedingService role in these associations, and the queue always plays the FollowingService role. There is always exactly one queue from which a dropper drops packets.

Since an absolute dropper drops all packets in its queue, it needs no configuration beyond a NextService tie to that queue. For an algorithmic dropper, however, further configuration is needed:

- a specific drop algorithm;
- parameters for the algorithm (for example, token bucket size);
- the source(s) of input(s) to the algorithm;
- possibly per-input parameters for the algorithm.

The first two of these items are represented by properties of the DropperService class, or properties of one of its subclasses. The last two, however, involve additional classes and associations.

3.10.1. Configuring Head and Tail Droppers

The HeadTailDropQueueBinding is the association that identifies the inputs for the algorithm executed by a tail dropper. This association is not used for a head dropper, because a head dropper always has exactly one input to its drop algorithm, and this input is always the queue from which it drops packets. For a tail dropper, this association is defined to have a many-to-many cardinality. There are, however, two distinct cases:

One dropper bound to many queues: This represents the case where the drop algorithm for the dropper involves inputs from more than one queue. The dropper still drops from only one queue, the one to which it is tied by a NextService association. But the drop decision may be influenced by the state of several queues. For the classes HeadTailDropper and HeadTailDropQueueBinding, the rule for combining the multiple inputs is simple addition: if the sum of the lengths of the monitored queues exceeds the dropper’s QueueThreshold value, then

packets are dropped. This rule for combining inputs may, however, be overridden by a different rule in subclasses of one or both of these classes.

One queue bound to many droppers: This represents the case where the state of one queue (which is typically also the queue from which packets are dropped) provides an input to multiple droppers' drop algorithms. A use case here is a classifier that splits a traffic stream into, say, four parts, representing four classes of traffic. Each of the parts goes through a separate HeadTailDropper, then they're re-merged onto the same queue. The net is a single queue containing packets of four traffic types, with, say, the following drop thresholds:

- Class 1 - 90% full
- Class 2 - 80% full
- Class 3 - 70% full
- Class 4 - 50% full

Here the percentages represent the overall state of the queue. With this configuration, when the queue in question becomes 50% full, Class 4 packets will be dropped rather than joining the queue, when it becomes 70% full, Class 3 and 4 packets will be dropped, etc.

The two cases described here can also occur together, if a dropper receives inputs from multiple queues, one or more of which are also providing inputs to other droppers.

3.10.2. Configuring RED Droppers

Like a tail dropper, a RED dropper, represented by an instance of the REDDropperService class, may take as its inputs the states of multiple queues. In this case, however, there is an additional step: each of these inputs may be smoothed before the RED dropper uses it, and the smoothing process itself must be parameterized. Consequently, in addition to REDDropperService and QueuingService, a third class, DropThresholdCalculationService, is introduced, to represent the per-queue parameterization of this smoothing process.

The following instance diagram illustrates how these classes work with each other:

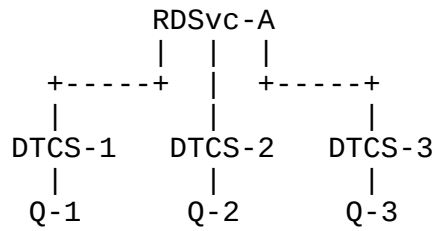


Figure 4. Inputs for a RED Dropper

So REDDropperService-A (RDSvc-A) is using inputs from three queues to make its drop decision. (As always, RDSvc-A is linked to the queue from which it drops packets via the NextService association.) For each of these three queues, there is a (DropThresholdCalculationService) DTCS instance that represents the smoothing weight and time interval to use when looking at that queue. Thus each DTCS instance is tied to exactly one queue, although a single queue may be examined (with different weight and time values) by multiple DTCS instances. Also, a DTCS instance and the queue behind it can be thought of as a "unit of reusability". So a single DTCS can be referred to by multiple RDSvc's.

Unless it is overridden by a different rule in a subclass of REDDropperService, the rule that a RED dropper uses to combine the smoothed inputs from the DTCS's to create a value to use in making its drop decision is simple addition.

3.11. Modeling of Queues and Schedulers

In order to appreciate the rationale behind this rather complex model for scheduling, we must consider the rather complex nature of schedulers, as well as the extreme variations in algorithms and implementations. Although these variations are broad, we have identified four examples that serve to test the model and justify its complexity.

3.11.1. Simple Hierarchical Scheduler

A simple, hierarchical scheduler has the following properties. First, when a scheduling opportunity is given to a set of queues, a single, viable queue is determined based on some scheduling criteria, such as bandwidth or priority. The output of the scheduler is the input to another scheduler that treats the first scheduler (and its queues) as a single logical queue. Hence, if the first scheduler determined the appropriate packet to release based on a priority assigned to each

queue, the second scheduler might specify a bandwidth limit/allocation for the entire set of queues aggregated by the first scheduler.

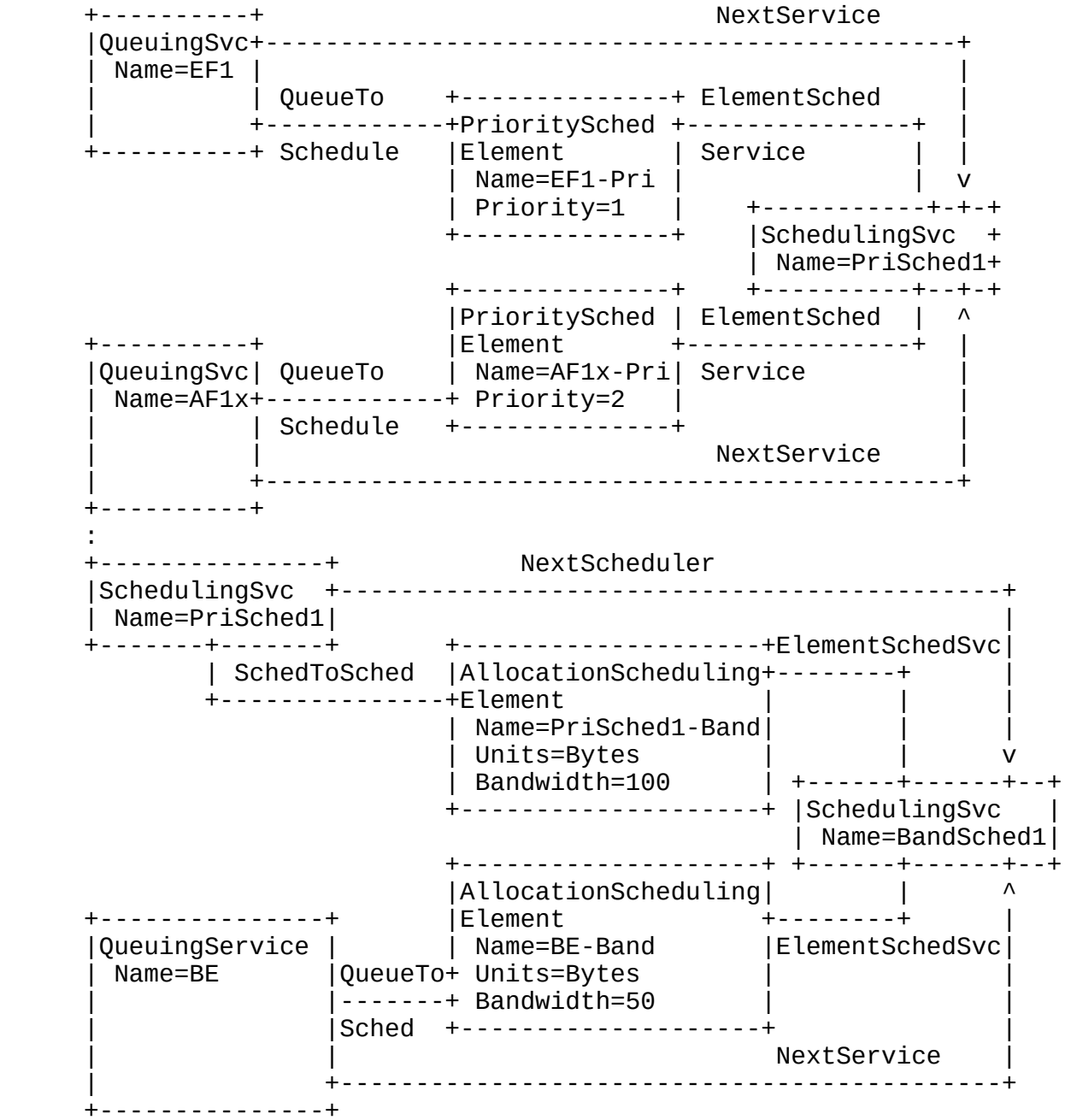


Figure 5. Example 1: Simple Hierarchical Scheduler

Figure 5 illustrates the example and how it would be instantiated using the model. In the figure, NextService determines the first scheduler after the queue. NextScheduler determines the subsequent ordering of schedulers. In addition, the ElementSchedulingService association determines the set of scheduling parameters used by a specific scheduler. Scheduling parameters can be bound either to queues or to schedulers. In the case of the SchedulingElement EF1-Pri, the binding is to a queue, so the QueueToSchedule association is used. In the case of the SchedulingElement PriSched1-Band, the binding is to another scheduler, so the SchedulerToSchedule association is used. Note that due to space constraints of the document, the SchedulingService PRISched1 is represented twice, to show how it is connected to all the other objects.

3.11.2. Complex Hierarchical Scheduler

A complex, hierarchical scheduler has the same characteristics as a simple scheduler, except that the criteria for the second scheduler are determined on a per queue basis rather than on an aggregate basis. One scenario might be a set of bounded priority schedulers. In this case, each queue is assigned a relative priority. However, each queue is also not allowed to exceed a bandwidth allocation that is unique to that queue. In order to support this scenario, the queue must be bound to two separate schedulers. Figure 6 illustrates this situation, by describing an EF queue and a best effort (BE) queue both pointing to a priority scheduler via the NextService association. The NextScheduler association between the priority scheduler and the bandwidth scheduler in turn defines the ordering of the scheduling hierarchy. Also note that each scheduler has a distinct set of scheduling parameters that are bound back to each queue. This demonstrates the need to support two or more parameter sets on a per queue basis.

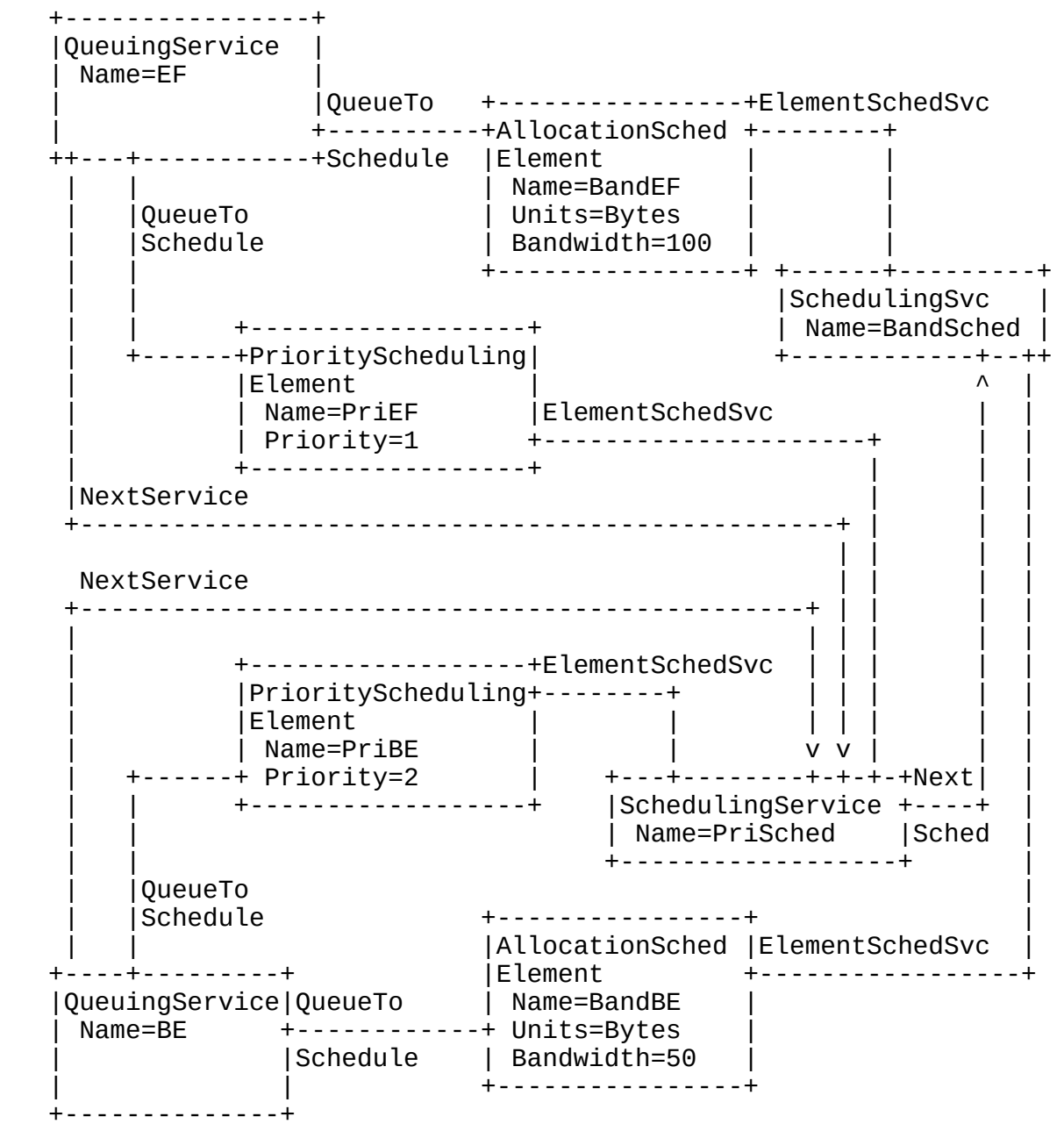


Figure 6. Example 2: Complex Hierarchical Scheduler

3.11.3. Excess Capacity Scheduler

An excess capacity scheduler offers a similar requirement to support two scheduling parameter sets per queue. However, in this scenario the reasons are a little different. Suppose a set of queues have each been assigned bandwidth limits to ensure that no traffic class starves out another traffic class. The result may be that one or more queues have exceeded their allocation while the queues that deserve scheduling opportunities are empty.

The question then is how is the excess (idle) bandwidth allocated. Conceivably, the scheduling criteria for excess capacity are completely different from the criteria that determine allocations under uniform load. This could be supported with a scheduling hierarchy. However, the problem is that the criteria for using the subsequent scheduler are different from those in the last two cases. Specifically, the next scheduler should only be used if a scheduling opportunity exists that was passed over by the prior scheduler.

When a scheduler chooses to forgo a scheduling decision, it is behaving as a non-work conserving scheduler. Work conserving schedulers, by definition, will always take advantage of a scheduling opportunity, irrespective of which queue is being serviced and how much bandwidth it has consumed in the past. This point leads to an interesting insight. The semantics of a non-work conserving scheduler are equivalent to those of a meter, in that if a packet is in profile it is given the scheduling opportunity, and if it is out of profile it does not get a scheduling opportunity. However, with meters there are semantics that determine the next action behavior when the packet is in profile and when the packet is out of profile. Similarly, with the non-work conserving scheduler, there needs to be a means for determining the next scheduler when a scheduler chooses not to utilize a scheduling opportunity.

Figure 7 illustrates this last scenario. It appears very similar to Figure 6, except that the binding between the allocation scheduler and the WRR scheduler is using a FailNextScheduler association. This association is explicitly indicating the fact that the only time the WRR scheduler would be used is when there are non-empty queues that the allocation scheduler rejected for scheduling consideration. Note that Figure 7 is incomplete, in that typically there would be several more queues that are bound to an allocation scheduler and a WRR scheduler.

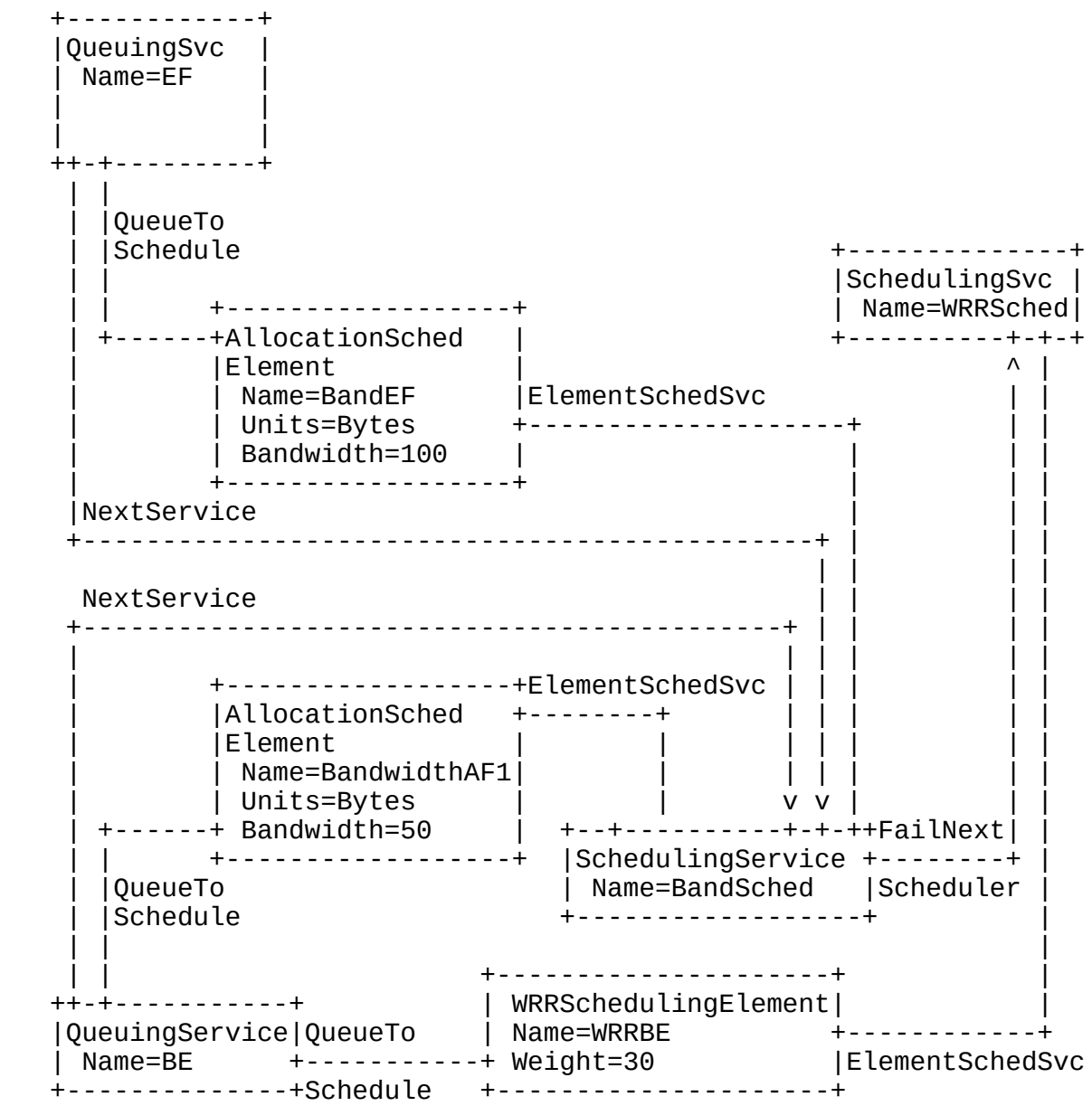


Figure 7. Example 3: Excess Capacity Scheduler

3.11.4. Hierarchical CBQ Scheduler

A hierarchical class-based queuing (CBQ) scheduler is the fourth scenario to be considered. In hierarchical CBQ, each queue is allocated a specific bandwidth allocation. Queues are grouped together into a logical scheduler. This logical scheduler in turn has an aggregate bandwidth allocation that equals the sum of the queues it is scheduling. In turn, logical schedulers can be aggregated into higher-level logical schedulers. Changing perspectives and looking top down, the top-most logical scheduler has 100% of the link capacity. This allocation is parceled out to logical schedulers below it such that the sum of the allocations is equal to 100%. These second tier schedulers may in turn parcel out their allocation across a third tier of schedulers and so forth until the lowest tier that parcels out their allocations to specific queues representing relatively fine-grained classes of traffic. The unique aspect of hierarchical CBQ is that when there is insufficient bandwidth for a specific allocation, schedulers higher in the tree are tested to see if another portion of the tree has capacity to spare.

Figure 8 demonstrates this example with two tiers. The example is split in half because of space constraints, resulting in the CBQTier1 scheduling service instance being represented twice. Note that the total allocation at the top tier is 50 Mb. The voice allocation is 22 Mb. The remaining 23 Mb is split between FTP and Web. Hence, if Web traffic is actually consuming 20 Mb (5 Mb in excess of the allocation). If FTP is consuming 5 Mb, then it is possible for the CBQTier1 scheduler to offer 3Mb of its allocation to Web traffic. However, this is not enough, so the FailNextScheduler association needs to be traversed to determine if there is any excess capacity available from the voice class. If the voice class is only consuming 15 Mb of its 22 Mb allocation, there are sufficient resources to allow the web traffic through. Note that FailNextScheduler is used as the association. The reason is because the CBQTier1 scheduler in fact failed to schedule a packet because of insufficient resources. It is conceivable that a variant of hierarchical CBQ allows a hierarchy for successful scheduling as well. Hence, both associations are necessary.

Note that due to space constraints of the document, the SchedulingService CBQTier1 is represented twice, to show how it is connected to all the other objects.

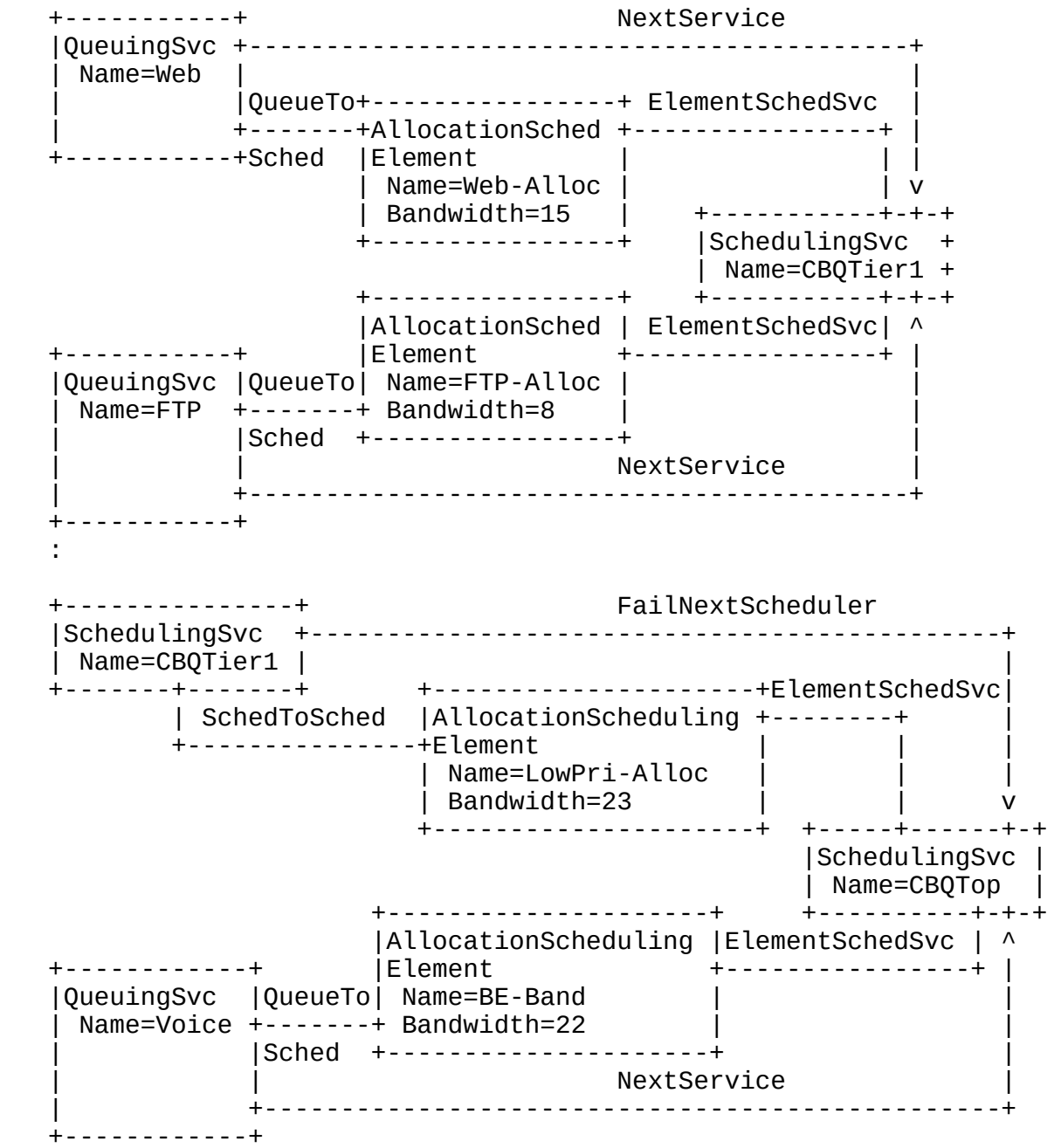


Figure 8. Example 4: Hierarchical CBQ Scheduler

4. The Class Hierarchy

The following sections present the class and association hierarchies that together comprise the information model for modeling QoS capabilities at the device level.

4.1. Associations and Aggregations

Associations and aggregations are a means of representing relationships between two (or theoretically more) objects. Dependency, aggregation, and other relationships are modeled as classes containing two (or more) object references. It should be noted that aggregations represent either "whole-part" or "collection" relationships. For example, aggregation can be used to represent the containment relationship between a system and the components that constitute the system.

Since associations and aggregations are classes, they can benefit from all of the object-oriented features that other non-relationship classes have. For example, they can contain properties and methods, and inheritance can be used to refine their semantics such that they represent more specialized types of their superclasses.

Note that an association (or an aggregation) object is treated as an atomic unit (individual instance), even though it relates/collects/is comprised of multiple objects. This is a defining feature of an association (or an aggregation) - although the individual elements that are related to other objects have their own identities, the association (or aggregation) object that is constructed using these objects has its own identity and name as well.

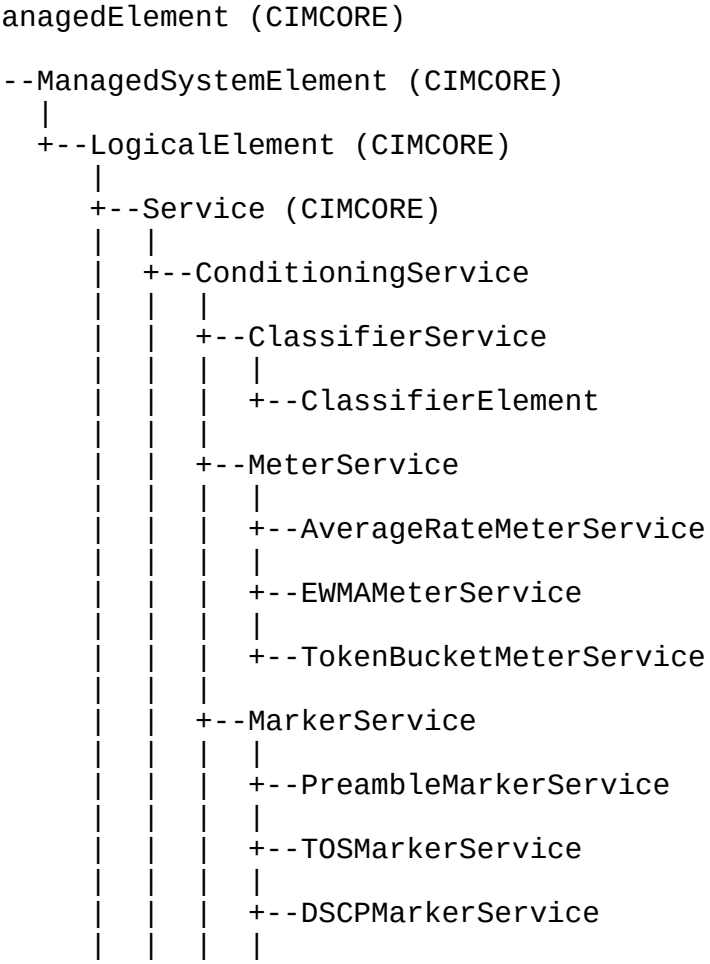
It is important to note that associations and aggregations form an inheritance hierarchy that is separate from the class inheritance hierarchy. Although associations and aggregations are typically bi-directional, there is nothing that prevents higher order associations or aggregations from being defined. However, such associations and aggregations are inherently more complex to define, understand, and use. In practice, associations and aggregations of orders higher than binary are rarely used, because of their greatly increased complexity and lack of generality. All of the associations and aggregations defined in this model are binary.

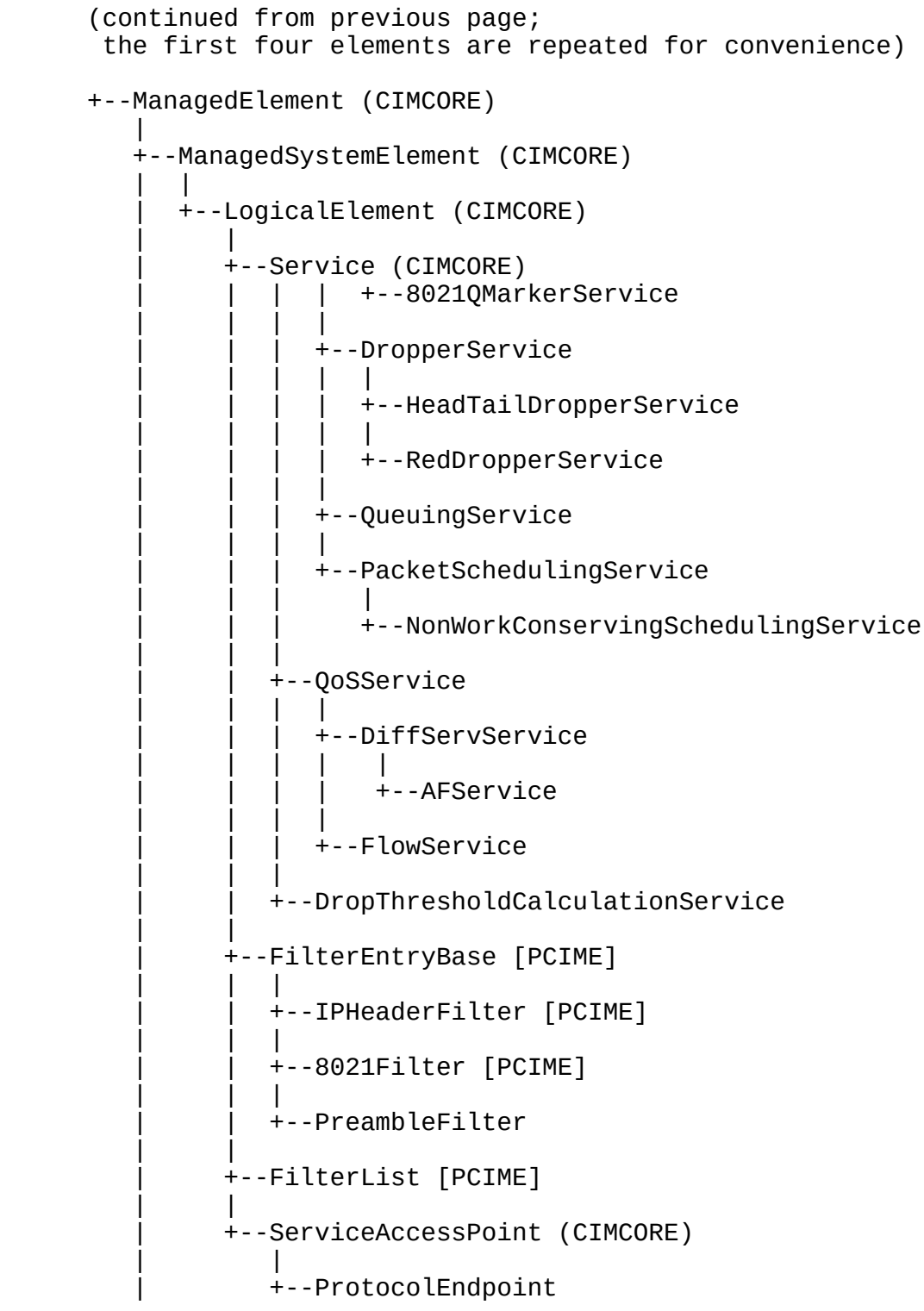
Note also that by definition, associations and aggregations cannot be unary.

Finally, note that associations and aggregations that are defined between two classes do not affect the classes themselves. That is, the addition or deletion of an association or an aggregation does not affect the interfaces of the classes that it is connecting.

4.2. The Structure of the Class Hierarchies

The structure of the class, association, and aggregation class inheritance hierarchies for managing the datapaths of QoS devices is shown, respectively, in Figure 9, Figure 10, and Figure 11. The notation (CIMCORE) identifies a class defined in the CIM Core model. Please refer to [CIM] for the definitions of these classes. Similarly, the notation [PCIME] identifies a class defined in the Policy Core Information Model Extensions document. This model has been influenced by [CIM], and is compatible with the Directory Enabled Networks (DEN) effort.




(continued from previous page;
the first four elements are repeated for convenience)
+--ManagedElement (CIMCORE)
+--ManagedSystemElement (CIMCORE)
+--LogicalElement (CIMCORE)
+--Service (CIMCORE)
+--8021QMarkerService
+--DropperService
+--HeadTailDropperService
+--RedDropperService
+--QueuingService
+--PacketSchedulingService
+--NonWorkConservingSchedulingService
+--QoSService
+--DiffServService
+--AFService
+--FlowService
+--DropThresholdCalculationService
+--FilterEntryBase [PCIME]
+--IPHeaderFilter [PCIME]
+--8021Filter [PCIME]
+--PreambleFilter
+--FilterList [PCIME]
+--ServiceAccessPoint (CIMCORE)
+--ProtocolEndpoint

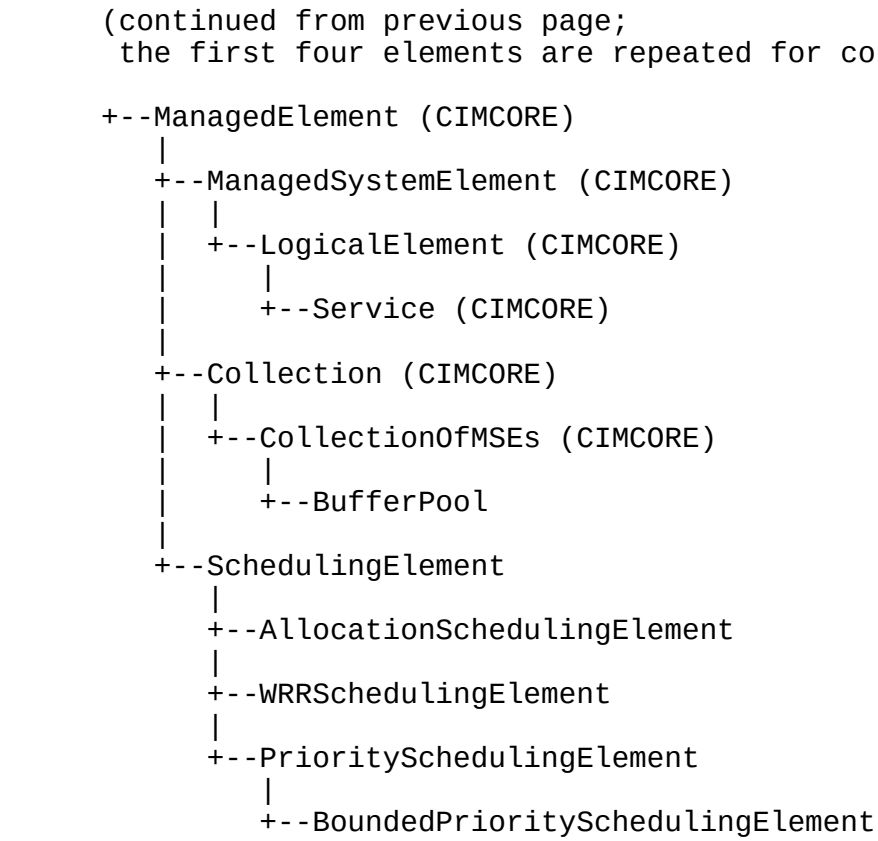


Figure 9. Class Inheritance Hierarchy

The inheritance hierarchy for the associations defined in this document is shown in Figure 10.

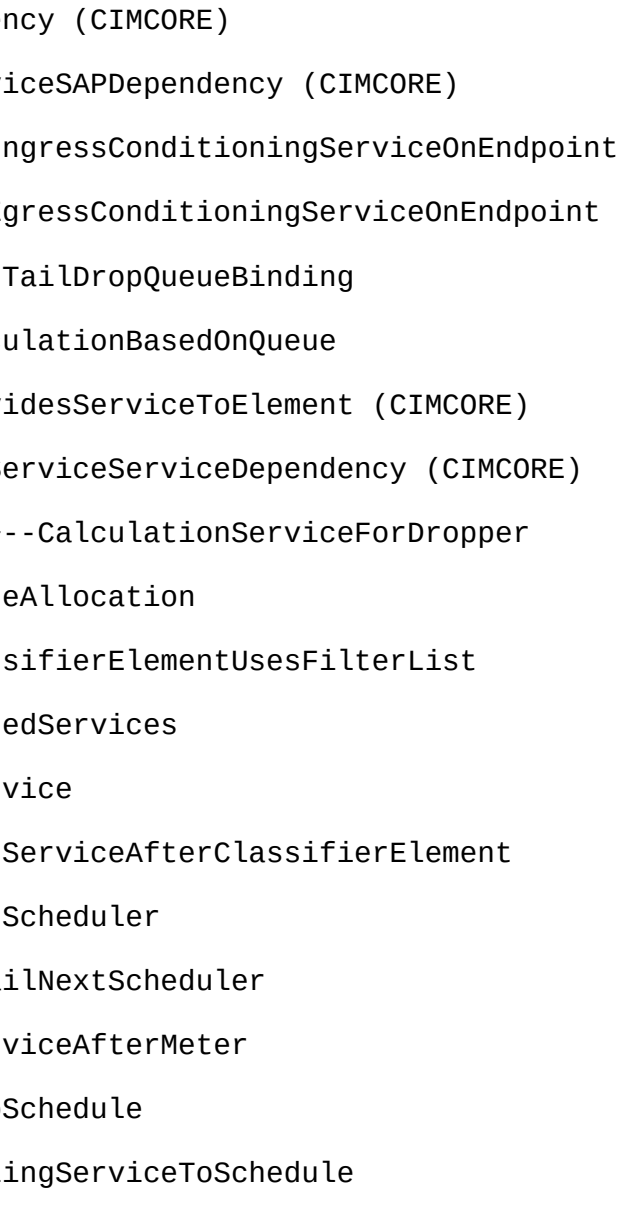


Figure 10. Association Class Inheritance Hierarchy

The inheritance hierarchy for the aggregations defined in this document is shown in Figure 11.

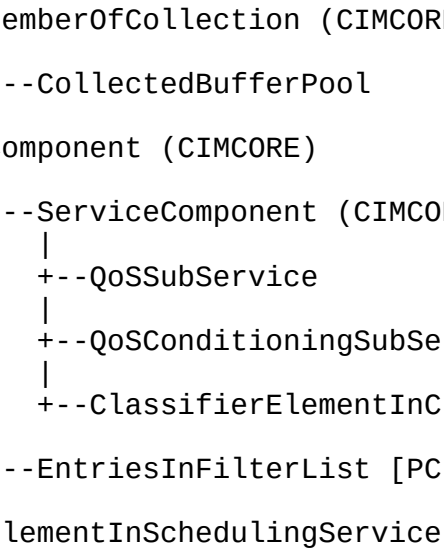


Figure 11. Aggregation Class Inheritance Hierarchy

4.3. Class Definitions

This section presents the classes and properties that make up the Information Model for describing QoS-related functionality in network devices, including hosts. These definitions are derived from definitions in the CIM Core model [CIM]. Only the QoS-related classes are defined in this document. However, other classes drawn from the CIM Core model, as well as from [PCIME], are described briefly. The reader is encouraged to look at [CIM] and at [PCIME] for further information. Associations and aggregations are defined in Section 4.4.

4.3.1. The Abstract Class ManagedElement

This is an abstract class defined in the Core Model of CIM. It is the root of the entire class inheritance hierarchy in CIM. Among the associations that refer to it are two that are subclassed in this document: Dependency and MemberOfCollection, which is an aggregation. ManagedElement’s properties are Caption and Description. Both are free-form strings to describe an instantiated object. Please refer to [CIM] for the full definition of this class.

4.3.2. The Abstract Class ManagedSystemElement

This is an abstract class defined in the Core Model of CIM; it is a subclass of ManagedElement. ManagedSystemElement serves as the base class for the PhysicalElement and LogicalElement class hierarchies. LogicalElement, in turn, is the base class for a number of important CIM hierarchies, including System. Any distinguishable component of a System is a candidate for inclusion in this class hierarchy, including physical components (e.g., chips and cards) and logical components (e.g., software components, services, and other objects).

None of the associations in which this class participates is used directly in the QoS device state model. However, the aggregation Component, which relates one ManagedSystemElement to another, is the base class for the two aggregations that form the core of the QoS device state model: QoSSubService and QoSConditioningSubService. Similarly, the association ProvidesServiceToElement, which relates a ManagedSystemElement to a Service, is the base class for the model's CalculationServiceForDropper association.

Please refer to [CIM] for the full definition of this class.

4.3.3. The Abstract Class LogicalElement

This is an abstract class defined in the Core Model of CIM. It is a subclass of the ManagedSystemElement class, and is the base class for all logical components of a managed System, such as Files, Processes, or system capabilities in the form of Logical Devices and Services. None of the associations in which this class participates is relevant to the QoS device state model. Please refer to [CIM] for the full definition of this class.

4.3.4. The Abstract Class Service

This is an abstract class defined in the Core Model of CIM. It is a subclass of the LogicalElement class, and is the base class for all objects that represent a "service" or functionality in a System. A Service is a general-purpose object that is used to configure and manage the implementation of functionality. As noted above in section 4.3.2, this class participates in the ProvidesServiceToElement association. Please refer to [CIM] for the full definition of this class.

4.3.5. The Class ConditioningService

This is a concrete subclass of the CIM Core class Service; it represents the ability to define how traffic is conditioned in the data-forwarding path of a device. The subclasses of

ConditioningService define the particular types of conditioning that are done. Six fundamental types of conditioning are defined in this document. These are the services performed by a classifier, a meter, a marker, a dropper, a queue, and a scheduler. Other, more sophisticated types of conditioning may be defined in future documents.

ConditioningService is a concrete class because at the time it was defined in CIM, its superclass was concrete. While this class can be instantiated, an instance of it would not accomplish anything, because the nature of the conditioning, and the parameters that control it, are specified only in the subclasses of ConditioningService.

Two associations in which ConditioningService participates are critical to its usage in QoS - QoSConditioningSubService and NextService. QoSConditioningSubService aggregates ConditioningServices into a particular QoS service (such as AF), to describe the specific conditioning functionality that underlies that QoS service in a particular device. NextService indicates the subsequent conditioning service(s) for different traffic streams.

The class definition is as follows:

NAME	ConditioningService
DESCRIPTION	A concrete class to define how traffic is conditioned in the data forwarding path of a host or network device.
DERIVED FROM	Service
TYPE	Concrete
PROPERTIES	(none)

4.3.6. The Class ClassifierService

The concept of a Classifier comes from [DSMODEL]. ClassifierService is a concrete class that represents a logical entity in an ingress or egress interface of a device, that takes a single input stream, and sorts it into one or more output streams. The sorting is done by a set of filters that select packets based on the packet contents, or possibly based on other attributes associated with the packet. Each output stream is the result of matching a particular filter.

The representation of classifiers in QDDIM is closely related to that presented in [DSMIB] and [DSMODEL]. Rather than being linked directly to its FilterLists, a classifier is modeled here as an aggregation of ClassifierElements. Each of these ClassifierElements is then linked to a single FilterList, by the association ClassifierElementUsesFilterList.

A Classifier is modeled as a subclass of ConditioningService so that it can be aggregated into a QoSService (using the QoSConditioningSubService aggregation), and can use the NextService association to identify the subsequent ConditioningService objects for the different traffic streams.

ClassifierService is designed to allow hierarchical classification. When hierarchical classification is used, a ClassifierElement may point to another ClassifierService. When used for this purpose, the ClassifierElement must not use the ClassifierElementUsesFilterList association.

The class definition is as follows:

NAME	ClassifierService
DESCRIPTION	A concrete class describing how an input traffic stream is sorted into multiple output streams using one or more filters.
DERIVED FROM	ConditioningService
TYPE	Concrete
PROPERTIES	(none)

4.3.7. The Class ClassifierElement

The concept of a ClassifierElement comes from [DSMIB]. This concrete class represents the linkage, within a single ClassifierService, between a FilterList that specifies a set of criteria for selecting packets from the stream of packets coming into the ClassifierService, and the next ConditioningService to which the selected packets go after they leave the ClassifierService. ClassifierElement has no properties of its own. It is present to serve as the anchor for an aggregation with its classifier, and for associations with its FilterList and its next ConditioningService.

When a ClassifierElement is associated with a ClassifierService through the NextServiceAfterClassifierElement association, the ClassifierElement may not use the ClassifierElementUsesFilterList association. Further, when a ClassifierElement is associated with a ClassifierService as described above, the order of processing of the associated ClassifierService is a function of the ClassifierOrder property of the ClassifierElementInClassifierService aggregation. For example, lets assume the following:

1. ClassifierService (C1) aggregates ClassifierElements (E1), (E2) and (E3), with relative ClassifierOrder values of 1, 2, and 3.

2. ClassifierElements (E1) and (E3) associations to FilterLists (F1) and (F3) respectively using the ClassifierElementUsesFilterList association.

3. (E1) & (E3) are associated with Meters (M1) and (M3) through their respective NextServiceAfterClassifierElement associations.

4. (E2) is associated with ClassifierService (C2) through its NextServiceAfterClassifierElement association.

5. ClassifierService (C2) aggregates ClassifierElements (E4) and (E5) with relative ClassifierOrder values of 1 and 2.

6. ClassifierElements (E4) and (E5) have associations to FilterLists (F4) and (F5) respectively using the ClassifierElementUsesFilterList association.

In this example, packet processing would match FilterLists in the order of (F1), (F4), (F5), and (F3).

The class definition is as follows:

NAME	ClassifierElement
DESCRIPTION	A concrete class representing the process by which a classifier uses a filter to select packets to forward to a specific next conditioning service.
DERIVED FROM	ClassifierService
TYPE	Concrete
PROPERTIES	(none)

4.3.8. The Class MeterService

This is a concrete class that represents the metering of network traffic. Metering is the function of monitoring the arrival times of packets of a traffic stream, and determining the level of conformance of each packet with respect to a pre-established traffic profile. A meter has the ability to invoke different ConditioningServices for conforming and non-conforming traffic. Traffic leaving a meter may be further conditioned (e.g., dropped or queued) by routing the packet to another conditioning element. Please see [DSMODEL] for more information on metering.

This class is the base class for defining different types of meters. As such, it contains common properties that all meter subclasses share. It is modeled as a ConditioningService so that it can be aggregated into a QoSService (using the QoSConditioningSubService

association), to indicate that its functionality underlies that QoS service. MeterService also participates in the NextServiceAfterMeter association, to identify the subsequent ConditioningService objects for conforming and non-conforming traffic.

The class definition is as follows:

NAME	MeterService
DESCRIPTION	A concrete class describing the monitoring of traffic with respect to a pre-established traffic profile.
DERIVED FROM	ConditioningService
TYPE	Concrete
PROPERTIES	MeterType, OtherMeterType, ConformanceLevels

Note: The MeterType property and the MeterService subclasses provide similar information. The MeterType property is defined for query purposes and for future expansion. It is possible that not all MeterServices will require a subclass to define them. In these cases, MeterService will be instantiated directly, and the MeterType property will provide the only way of identifying the type of the meter.

4.3.8.1. The Property MeterType

This property is an enumerated 16-bit unsigned integer that is used to specify the particular type of meter represented by an instance of MeterService. The following enumeration values are defined:

1 - Other
2 - Average Rate Meter
3 - Exponentially Weighted Moving Average Meter
4 - Token Bucket Meter

Note: if the value of MeterType is not one of these four values, it SHOULD be interpreted as if it had the value '1' (Other).

4.3.8.2. The Property OtherMeterType

This is a string property that defines a vendor-specific description of a type of meter. It is used when the value of the MeterType property in the instance is equal to 1.

4.3.8.3. The Property ConformanceLevels

This property is a 16-bit unsigned integer. It indicates the number of conformance levels supported by the meter. For example, when only "in profile" versus "out of profile" metering is supported, ConformanceLevels is equal to 2.

4.3.9. The Class AverageRateMeterService

This is a concrete subclass of MeterService that represents a simple meter, called an Average Rate Meter. This type of meter measures the average rate at which packets are submitted to it over a specified time. Packets are defined as conformant if their average arrival rate does not exceed the specified measuring rate of the meter. Any packet that causes the specified measuring rate to be exceeded is defined to be non-conforming. For more information, please see [DSMODEL].

The class definition is as follows:

NAME	AverageRateMeterService
DESCRIPTION	A concrete class classifying traffic as either conforming or non-conforming, depending on whether the arrival of a packet causes the average arrival rate to exceed a pre-determined value.
DERIVED FROM	MeterService
TYPE	Concrete
PROPERTIES	AverageRate, DeltaInterval

4.3.9.1. The Property AverageRate

This is an unsigned 32-bit integer that defines the rate used to determine whether admitted packets are in conformance or not. The value is specified in kilobits per second.

4.3.9.2. The Property DeltaInterval

This is an unsigned 64-bit integer that defines the time period over which the average measurement should be taken. The value is specified in microseconds.

4.3.10. The Class EWMAMeterService

This is a concrete subclass of the MeterService class that represents an exponentially weighted moving average meter. This meter is a simple low-pass filter that measures the rate of incoming packets

over a small, fixed sampling interval. Any admitted packet that pushes the average rate over a pre-defined limit is defined to be non-conforming. Please see [DSMODEL] for more information.

The class definition is as follows:

NAME	EWMAMeterService
DESCRIPTION	A concrete class classifying admitted traffic as either conforming or non-conforming, depending on whether the arrival of a packet causes the average arrival rate in a small fixed sampling interval to exceed a pre-determined value or not.
DERIVED FROM	MeterService
TYPE	Concrete
PROPERTIES	AverageRate, DeltaInterval, Gain

4.3.10.1. The Property AverageRate

This property is an unsigned 32-bit integer that defines the average rate against which the sampled arrival rate of packets should be measured. Any packet that causes the sampled rate to exceed this rate is deemed non-conforming. The value is specified in kilobits per second.

4.3.10.2. The Property DeltaInterval

This property is an unsigned 64-bit integer that defines the sampling interval used to measure the arrival rate. The calculated rate is averaged over this interval and checked against the AverageRate property. All packets whose computed average arrival rate is less than the AverageRate are deemed conforming.

The value is specified in microseconds.

4.3.10.3. The Property Gain

This property is an unsigned 32-bit integer representing the reciprocal of the time constant (e.g., frequency response) of what is essentially a simple low-pass filter. For example, the value 64 for this property represents a time constant value of 1/64.

4.3.11. The Class TokenBucketMeterService

This is a concrete subclass of the MeterService class that represents the metering of network traffic using a token bucket meter. Two types of token bucket meters are defined using this class - a simple, two-parameter bucket meter, and a multi-stage meter.

A simple token bucket usually has two parameters, an average token rate and a burst size, and has two conformance levels: "conforming" and "non-conforming". This class also defines an excess burst size, which enables the meter to have three conformance levels ("conforming", "partially conforming", and "non-conforming"). In this case, packets that exceed the excess burst size are deemed non-conforming, while packets that exceed the smaller burst size but are less than the excess burst size are deemed partially conforming. Operation of these meters is described in [DSMODEL].

The class definition is as follows:

NAME	TokenBucketMeterService
DESCRIPTION	A concrete class classifying admitted traffic with respect to a token bucket. Either two or three levels of conformance can be defined.
DERIVED FROM	MeterService
TYPE	Concrete
PROPERTIES	AverageRate, PeakRate, BurstSize, ExcessBurstSize

4.3.11.1. The Property AverageRate

This property is an unsigned 32-bit integer that specifies the committed rate of the meter. The value is expressed in kilobits per second.

4.3.11.2. The Property PeakRate

This property is an unsigned 32-bit integer that specifies the peak rate of the meter. The value is expressed in kilobits per second.

4.3.11.3. The Property BurstSize

This property is an unsigned 32-bit integer that specifies the maximum number of tokens available for the committed rate (specified by the AverageRate property). The value is expressed in kilobytes.

4.3.11.4. The Property ExcessBurstSize

This property is an unsigned 32-bit integer that specifies the maximum number of tokens available for the peak rate (specified by the PeakRate property). The value is expressed in kilobytes.

4.3.12. The Class MarkerService

This is a concrete class that represents the general process of marking some field in a network packet with some value. Subclasses of MarkerService identify particular fields to be marked, and introduce properties to represent the values to be used in marking these fields. Markers are usually invoked as a result of a preceding classifier match. Operation of markers of various types is described in [DSMODEL].

MarkerService is a concrete class because at the time it was defined in CIM, its superclass was concrete. While this class can be instantiated, an instance of it would not accomplish anything, because both the field to be marked and the value to be used to mark it are specified only in subclasses of MarkerService.

MarkerService is modeled as a ConditioningService so that it can be aggregated into a QoSService (using the QoSConditioningSubService association) to indicate that its functionality underlies that QoS service. It participates in the NextService association to identify the subsequent ConditioningService object that acts on traffic after it has been marked by the marker.

The class definition is as follows:

NAME	MarkerService
DESCRIPTION	A concrete class representing the general process of marking a selected field in a packet with a specified value. Packets are marked in order to control the conditioning that they will subsequently receive.
DERIVED FROM	ConditioningService
TYPE	Concrete
PROPERTIES	(none)

4.3.13. The Class PreambleMarkerService

This is a concrete class that models the storing of traffic-conditioning results in a packet preamble. See Section 3.8.3 for a discussion of how, and why, QDDIM models the capability to store these results in a packet preamble. An instance of

PreambleMarkerService appends to a packet preamble a two-part string of the form "<type>,<value>". Section 3.8.3 provides a list of the <type> strings defined by QDDIM. Implementations may support other <type>'s in addition to these.

The class definition is as follows:

NAME	PreambleMarkerService
DESCRIPTION	A concrete class representing the saving of traffic-conditioning results in a packet preamble.
DERIVED FROM	MarkerService
TYPE	Concrete
PROPERTIES	FilterItemList[ ]

4.3.13.1. The Multi-valued Property FilterItemList

This property is an ordered list of strings, where each string has the format "<type>,<value>". See Section 3.8.3 for a list of <type>'s defined in QDDIM, and the nature of the associated <value> for each of these types.

4.3.14. The Class ToSMarkerService

This is a concrete class that represents the marking of the ToS field in the IPv4 packet header [R791]. Following common practice, the value to be written into the field is represented as an unsigned 8-bit integer.

The class definition is as follows:

NAME	ToSMarkerService
DESCRIPTION	A concrete class representing the process of marking the type of service (ToS) field in the IPv4 packet header with a specified value. Packets are marked in order to control the conditioning that they will subsequently receive.
DERIVED FROM	MarkerService
TYPE	Concrete
PROPERTIES	ToSValue

4.3.14.1. The Property ToSValue

This property is an unsigned 8-bit integer, representing a value to be used for marking the type of service (ToS) field in the IPv4 packet header. The ToS field is defined to be a complete octet, so the range for this property is 0..255. Some implementations, however, require that the lowest-order bit in the ToS field always be '0'. Such an implementation is consequently unable to support an odd TosValue.

4.3.15. The Class DSCPMarkerService

This is a concrete class that represents the marking of the differentiated services codepoint (DSCP) within the DS field in the IPv4 and IPv6 packet headers, as defined in [R2474]. Following common practice, the value to be written into the field is represented as an unsigned 8-bit integer.

The class definition is as follows:

NAME	DSCPMarkerService
DESCRIPTION	A concrete class representing the process of marking the DSCP field in a packet with a specified value. Packets are marked in order to control the conditioning that they will subsequently receive.
DERIVED FROM	MarkerService
TYPE	Concrete
PROPERTIES	DSCPValue

4.3.15.1. The Property DSCPValue

This property is an unsigned 8-bit integer, representing a value to be used for marking the DSCP within the DS field in an IPv4 or IPv6 packet header. Since the DSCP consists of 6 bits, the values for this property are limited to the range 0..63. When the DSCP is marked, the remaining two bit in the DS field are left unchanged.

4.3.16. The Class 8021QMarkerService

This is a concrete class that represents the marking of the user priority field defined in the IEEE 802.1Q specification [IEEE802Q]. Following common practice, the value to be written into the field is represented as an unsigned 8-bit integer.

The class definition is as follows:

NAME	8021QMarkerService
DESCRIPTION	A concrete class representing the process of marking the Priority field in an 802.1Q-compliant frame with a specified value. Frames are marked in order to control the conditioning that they will subsequently receive.
DERIVED FROM	MarkerService
TYPE	Concrete
PROPERTIES	PriorityValue

4.3.16.1. The Property PriorityValue

This property is an unsigned 8-bit integer, representing a value to be used for marking the Priority field in the 802.1Q header. Since the Priority field consists of 3 bits, the values for this property are limited to the range 0..7. When the Priority field is marked, the remaining bits in its octet are left unchanged.

4.3.17. The Class DropperService

This is a concrete class that represents the ability to selectively drop network traffic, or to invoke another ConditioningService for further processing of traffic that is not dropped. This is the base class for different types of droppers. Droppers are distinguished by the algorithm that they use to drop traffic. Please see [DSMODEL] for more information about the various types of droppers. Note that this class encompasses both Absolute Droppers and Algorithmic Droppers from [DSMODEL].

DropperService is modeled as a ConditioningService so that it can be aggregated into a QoSService (using the QoSConditioningSubService association) to indicate that its functionality underlies that QoS service. It participates in the NextService association to identify the subsequent ConditioningService object that acts on any remaining traffic that is not dropped.

NextService has special semantics for droppers, in addition to the general "what happens next" semantics that apply to all ConditioningServices. The queue(s) from which a particular dropper drops packets are identified by following chain(s) of NextService associations "rightwards" from the dropper until they reach a queue.

The class definition is as follows:

NAME	DropperService
DESCRIPTION	A concrete base class describing the common characteristics of droppers.
DERIVED FROM	ConditioningService
TYPE	Concrete
PROPERTIES	DropperType, OtherDropperType, DropFrom

Note: The DropperType property and the DropperService subclasses provide similar information. The DropperType property is defined for query purposes, as well as for those cases where a subclass of DropperService is not needed to model a particular type of dropper. For example, the Absolute Dropper defined in [DSMODEL] is modeled as an instance of the DropperService class with its DropperType set to '4' ("Absolute Dropper").

4.3.17.1. The Property DropperType

This is an enumerated 16-bit unsigned integer that defines the type of dropper. Values include:

1 - Other
2 - Random
3 - HeadTail
4 - Absolute Dropper

Note: if the value of DropperType is not one of these four values, it SHOULD be interpreted as if it had the value '1' (Other).

4.3.17.2. The Property OtherDropperType

This string property is used in conjunction with the DropperType property. When the value of DropperType is '1' (i.e., Other), then the name of the type of dropper appears in this property.

4.3.17.3. The Property DropFrom

This is an unsigned 16-bit integer enumeration that indicates the point in the associated queue from which packets should be dropped. Defined enumeration values are:

- unknown(0)
- head(1)
- tail(2)

Note: if the value of DropFrom is '0' (unknown), or if it is not one of the three values listed here, then packets MAY be dropped from any location in the associated queue.

4.3.18. The Class HeadTailDropperService

This is a concrete class that represents the threshold information of a head or tail dropper. The inherited property DropFrom indicates whether a particular instance of this class represents a head dropper or a tail dropper.

A head dropper always examines the same queue from which it drops packets, and this queue is always related to the dropper as the following service in the NextService association.

The class definition is as follows:

NAME	HeadTailDropperService
DESCRIPTION	A concrete class used to describe a head or tail dropper.
DERIVED FROM	DropperService
TYPE	Concrete
PROPERTIES	QueueThreshold

4.3.18.1. The Property QueueThreshold

This is an unsigned 32-bit integer that indicates the queue depth at which traffic will be dropped. For a tail dropper, all newly arriving traffic is dropped. For a head dropper, packets at the front of the queue are dropped to make room for new packets, which are added at the end. The value is expressed in bytes.

4.3.19. The Class REDDropperService

This is a concrete class that represents the ability to drop network traffic using a Random Early Detection (RED) algorithm. This algorithm is described in [RED]. The purpose of a RED algorithm is to avoid congestion (as opposed to managing congestion). Instead of waiting for the queues to fill up, and then dropping large numbers of packets, RED works by monitoring the average queue depth. When the queue depth exceeds a minimum threshold, packets are randomly discarded. These discards cause TCP to slow its transmission rate for those connections that experienced the packet discards. Other TCP connections are not affected by these discards. Please see [DSMODEL] for more information about a dropper.

A RED dropper always drops packets from a single queue, which is related to the dropper as the following service in the NextService association. The queue(s) examined by the drop algorithm are found by following the CalculationServiceForDropper association to find the dropper's DropThresholdCalculationService, and then following the CalculationBasedOnQueue association(s) to find the queue(s) being watched.

The class definition is as follows:

NAME	REDDropperService
DESCRIPTION	A concrete class used to describe dropping using the RED algorithm (or one of its variants).
DERIVED FROM	DropperService
TYPE	Concrete
PROPERTIES	MinQueueThreshold, MaxQueueThreshold, ThresholdUnits, StartProbability, StopProbability

NOTE: In [DSMIB], there is a single diffServRandomDropTable, which represents the general category of random dropping. (RED is one type of random dropping, but there are also types of random dropping distinct from RED.) The REDDropperService class corresponds to the columns in the table that apply to the RED algorithm in particular.

4.3.19.1. The Property MinQueueThreshold

This is an unsigned 32-bit integer that defines the minimum average queue depth at which packets are subject to being dropped. The units are identified by the ThresholdUnits property. The slope of the drop probability function is described by the Start/StopProbability properties.

4.3.19.2. The Property MaxQueueThreshold

This is an unsigned 32-bit integer that defines the maximum average queue length at which packets are subject to always being dropped, regardless of the dropping algorithm and probabilities being used. The units are identified by the ThresholdUnits property.

4.3.19.3. The Property ThresholdUnits

This is an unsigned 16-bit integer enumeration that identifies the units for the MinQueueThreshold and MaxQueueThreshold properties. Defined enumeration values are:

- o bytes(1)
- o packets(2)

Note: if the value of ThresholdUnits is not one of these two values, it SHOULD be interpreted as if it had the value '1' (bytes).

4.3.19.4. The Property StartProbability

This is an unsigned 32-bit integer; in conjunction with the StopProbability property, it defines the slope of the drop probability function. This function governs the rate at which packets are subject to being dropped, as a function of the queue length.

This property expresses a drop probability in drops per thousand packets. For example, the value 100 indicates a drop probability of 100 per 1000 packets, that is, 10%. Min and max values are 0 to 1000.

4.3.19.5. The Property StopProbability

This is an unsigned 32-bit integer; in conjunction with the StartProbability property, it defines the slope of the drop probability function. This function governs the rate at which packets are subject to being dropped, as a function of the queue length.

This property expresses a drop probability in drops per thousand packets. For example, the value 100 indicates a drop probability of 100 per 1000 packets, that is, 10%. Min and max values are 0 to 1000.

4.3.20. The Class QueuingService

This is a concrete class that represents the ability to queue network traffic, and to specify the characteristics for determining long-term congestion. Please see [DSMODEL] for more information about queuing functionality.

QueuingService is modeled as a ConditioningService so that it can be aggregated into a QoSService (using the QoSConditioningSubService association) to indicate that its functionality underlies that QoS service.

The class definition is as follows:

NAME	QueuingService
DESCRIPTION	A concrete class describing the ability to queue network traffic and to specify the characteristics for determining long-term congestion.
DERIVED FROM	ConditioningService
TYPE	Concrete
PROPERTIES	CurrentQueueDepth, DepthUnits

4.3.20.1. The Property CurrentQueueDepth

This is an unsigned 32-bit integer, which functions as a (read-only) gauge representing the current depth of this one queue. This value may be important in diagnosing unexpected behavior by a DropThresholdCalculationService.

4.3.20.2. The Property DepthUnits

This is an unsigned 16-bit integer enumeration that identifies the units for the CurrentQueueDepth property. Defined enumeration values are:

- bytes(1)
- packets(2)

Note: if the value of DepthUnits is not one of these two values, it SHOULD be interpreted as if it had the value '1' (bytes). The

4.3.21. Class PacketSchedulingService

This is a concrete class that represents a scheduling service, which is a process that determines when a queued packet should be removed from a queue and sent to an output interface. Note that output interfaces can be physical network interfaces or interfaces to components internal to systems, such as crossbars or back planes. In either case, if multiple queues are involved, schedulers are used to provide access to the interface.

Each instance of a PacketSchedulingService describes a scheduler from the perspective of the queues that it is servicing. Please see [DSMODEL] for more information about a scheduler.

PacketSchedulingService is modeled as a ConditioningService so that it can be aggregated into a QoSService (using the QoSConditioningSubService association) to indicate that its functionality underlies that QoS service. It participates in the

NextService association to identify the subsequent ConditioningService object, if any, that acts on traffic after it has been processed by the scheduler.

The class definition is as follows:

NAME	PacketSchedulingService
DESCRIPTION	A concrete class used to determine when a packet should be removed from a queue and sent to an output interface.
DERIVED FROM	ConditioningService
TYPE	Concrete
PROPERTIES	SchedulerType, OtherSchedulerType

4.3.21.1. The Property SchedulerType

This property is an enumerated 16-bit unsigned integer, and defines the type of scheduler. Values are:

1 - Other
2 - FIFO
3 - Priority
4 - Allocation
5 - Bounded Priority
6 - Weighted Round Robin Packet

Note: if the value of SchedulerType is not one of these six values, it SHOULD be interpreted as if it had the value '2' (FIFO).

4.3.21.2. The Property OtherSchedulerType

This string property is used in conjunction with the SchedulerType property. When the value of SchedulerType is 1 (i.e., Other), then the type of scheduler is specified in this property.

4.3.22. The Class NonWorkConservingSchedulingService

This class does not add any properties beyond those it inherits from its superclass, PacketSchedulingService. It does, however, participate in one additional association, FailNextScheduler.

The class definition is as follows:

NAME	NonWorkConservingSchedulingService
DESCRIPTION	A concrete class representing a scheduler that is capable of operating in a non-work conserving manner.
DERIVED FROM	PacketSchedulingService
TYPE	Concrete
PROPERTIES	(none)

4.3.23. The Class QoSService

This is a concrete class that represents the ability to conceptualize a QoS service as a set of coordinated sub-services. This enables the network administrator to map business rules to the network, and the network designer to engineer the network such that it can provide different functions for different traffic streams.

This class has two main purposes. First, it serves as a common base class for defining the various sub-services needed to build higher-level QoS services. Second, it serves as a way to consolidate the relationships between different types of QoS services and different types of ConditioningServices.

For example, Gold Service may be defined as a QoSService which aggregates two QoS services together. Each of these QoS services could be represented by an instance of the class DiffServService, one for servicing of very high demand packets (represented by an instance of DiffServService itself), and one for the service given to most of the packets, represented by an instance of AFService, which is a subclass of DiffServService. The high demand DiffServService instance will then use the QoSConditioningSubService aggregation to aggregate together the necessary classifiers to indicate which traffic it applies to, and the appropriate meters for contract limits, the marker to mark the EF PHB in the packets, and the queuing-related conditioning services. The AFService instance will also use the QoSConditioningSubService aggregation, to aggregate its classifiers and meters, the several markers used to mark the different AF PHBs in the packets, and the queuing-related conditioning services needed to deliver the packet treatment.

QoSService is modeled as a type of Service, which is used as the anchor point for defining a set of sub-services that implement the desired conditioning characteristics for different types of flows. It will direct the specific type of conditioning services to be used in order to implement this service.

The class definition is as follows:

NAME	QoSService
DESCRIPTION	A concrete class used to represent a QoS service or set of services, as defined by a network administrator.
DERIVED FROM	Service
TYPE	Concrete
PROPERTIES	(none)

4.3.24. The Class DiffServService

This is a concrete class representing the use of standard or custom DiffServ services to implement a (higher-level) QoS service. Note that a DiffServService object may be just one of a set of coordinated QoSSubServices objects that together implement a higher-level QoS service.

DiffServService is modeled as a subclass of QoSService. This enables it to be related to a higher-level QoS service via QoSSubService, as well as to specific ConditioningService objects (e.g., metering, dropping, queuing, and others) via QoSConditioningSubService.

The class definition is as follows:

NAME	DiffServService
DESCRIPTION	A concrete class used to represent a DiffServ service associated with a particular Per Hop Behavior.
DERIVED FROM	QoSService
TYPE	Concrete
PROPERTIES	PHBID

4.3.24.1. The Property PHBID

This property is a 16-bit unsigned integer, which identifies a particular per hop behavior, or family of per hop behaviors. The value here is a Per Hop Behavior Identification Code, as defined in [R3140]. Note that as defined, these identification codes use the default, recommended, code points for PHBs as part of their structure. These values may well be different from the actual value used in the marker, as the marked value is a domain-dependent value. The ability to indicate the PHB Identification Code associated with a service is helpful for tying the QoS Service to reference documents, and for inter-domain coordination and operation.

4.3.25. The Class AFService

This is a concrete class that represents a specialization of the general concept of forwarding network traffic, by adding specific semantics that characterize the operation of the Assured Forwarding (AF) Service ([R2597]).

[R2597] defines four different AF classes, to represent four different treatments of traffic. A different amount of forwarding resources, such as buffer space and bandwidth, are allocated to each AF class. Within each AF class, IP packets are marked with one of three possible drop precedence values. The drop precedence of a packet determines the relative importance of that packet compared to other packets within the same AF class, if congestion occurs. A congested interface will try to avoid dropping packets marked with a lower drop precedence value, by instead discarding packets marked with a higher drop precedence value.

Note that [R2597] defines 12 DSCPs that together represent the AF Per Hop Behavior (PHB) group. Implementations are free to extend this (e.g., add more classes and/or drop precedences).

The AFService class is modeled as a specialization of DiffServService, which is in turn a specialization of QoSService. This enables it to be related to higher-level QoS services, as well as to lower-level conditioning sub-services (e.g., classification, metering, dropping, queuing, and others).

The class definition is as follows:

NAME	AFService
DESCRIPTION	A concrete class for describing the common characteristics of differentiated services that are used to affect traffic forwarding, using the AF PHB Group.
DERIVED FROM	DiffServService
TYPE	Concrete
PROPERTIES	ClassNumber, DropperNumber

4.3.25.1. The Property ClassNumber

This property is an 8-bit unsigned integer that indicates the number of AF classes that this AF implementation uses. Among the instances aggregated using the QoSConditioningSubService aggregation with an instance of AFService, one SHOULD find markers with as many distinct values as the ClassNumber of the AFService instance.

4.3.25.2. The Property DropperNumber

This property is an 8-bit unsigned integer that indicates the number of drop precedence values that this AF implementation uses. The number of drop precedence values is the number PER AF CLASS. The corresponding droppers will be found in the collection of conditioning services aggregated with the QoSConditioningSubService aggregation.

4.3.26. The Class FlowService

This class represents a service that supports a particular microflow. The microflow is identified by the string-valued property FlowID. In some implementations, an instance of this class corresponds to an entry in the implementation's flow table.

The class definition is as follows:

NAME	FlowService
DESCRIPTION	A concrete class representing a microflow.
DERIVED FROM	QoSService
TYPE	Concrete
PROPERTIES	FlowID

4.3.26.1. The Property FlowID

This property is a string containing an identifier for a microflow.

4.3.27. The Class DropThresholdCalculationService

This class represents a logical entity that calculates an average queue depth for a queue, based on a smoothing weight and a sampling time interval. It does this calculation on behalf of a RED dropper, to allow the dropper to make its decisions whether to drop packets based on a smoothed average queue depth for the queue.

The class definition is as follows:

NAME	DropThresholdCalculationService
DESCRIPTION	A concrete class representing a logical entity that calculates an average queue depth for a queue, based on a smoothing weight and a sampling time interval. The latter are properties of this Service, describing how it operates and its necessary parameters.
DERIVED FROM	Service
TYPE	Concrete
PROPERTIES	SmoothingWeight, TimeInterval

4.3.27.1. The Property SmoothingWeight

This property is a 32-bit unsigned integer, ranging between 0 and 100,000 - specified in thousandths. It defines the weighting of past history in affecting the calculation of the current average queue depth. The current queue depth calculation uses the inverse of this value as its factor, and one minus that inverse as the factor for the historical average. The calculation takes the form:

```
average = (old_average*(1-inverse of SmoothingWeight))
     + (current_queue_depth*inverse of SmoothingWeight)
```

Implementations may choose to limit the acceptable set of values to a specified set, such as powers of 2.

Min and max values are 0 and 100000.

4.3.27.2. The Property TimeInterval

This property is a 32-bit unsigned integer, defining the number of nanoseconds between each calculation of average/smoothed queue depth. If this property is not specified, the CalculationService may determine an appropriate interval.

4.3.28. The Abstract Class FilterEntryBase

FilterEntryBase is the abstract base class from which all filter entry classes are derived. It serves as the endpoint for the EntriesInFilterList aggregation, which groups filter entries into filter lists. Its properties include CIM naming properties and an IsNegated boolean property (to easily "NOT" the match information specified in an instance of one of its subclasses).

Because FilterEntryBase has general applicability, it is defined in [PCIME]. See [PCIME] for the definition of this class.

4.3.29. The Class IPHeaderFilter

This concrete class makes it possible to represent an entire IP header filter in a single object. A property IpVersion identifies whether the IP addresses in an instance are IPv4 or IPv6 addresses. (Since the source and destination IP addresses come from the same packet header, they will always be of the same type.)

See [PCIME] for the definition of this class.

4.3.30. The Class 8021Filter

This concrete class allows 802.1.source and destination MAC addresses, as well as the 802.1 protocol ID, priority, and VLAN identifier fields, to be expressed in a single object

See [PCIME] for the definition of this class.

4.3.31. The Class PreambleFilter

This is a concrete class that models classifying packets using traffic-conditioning results stored in a packet preamble by a PreambleMarkerService. See Section 3.8.3 for a discussion of how, and why, QDDIM models the capability to store these results in a packet preamble. An instance of PreambleFilter is used to select packets based on a two-part string identifying a specific result. The logic for this match is "at least one". That is, a packet with multiple results in its preamble matches a filter if at least one of these results matches the filter.

The class definition is as follows:

NAME	PreambleFilter
DESCRIPTION	A concrete class representing criteria for selecting packets based on prior traffic-conditioning results stored in a packet preamble.
DERIVED FROM	FilterEntryBase
TYPE	Concrete
PROPERTIES	FilterItemList[ ]

4.3.31.1. The Multi-valued Property FilterItemList

This property is an ordered list of strings, where each string has the format "<type>,<value>". See Section 3.8.3 for a list of <type>'s defined in QDDIM, and the nature of the associated <value> for each of these types.

Note that there are two parallel terminologies for characterizing meter results. The enumeration value "conforming(1)" is sometimes described as "in profile," and the value "nonConforming(3)" is sometimes described as "out of profile".

4.3.32. The Class FilterList

This is a concrete class that aggregates instances of (subclasses of) FilterEntryBase via the aggregation EntriesInFilterList. It is possible to aggregate different types of filters into a single FilterList - for example, packet header filters (represented by the IPHeaderFilter class) and security filters (represented by subclasses of FilterEntryBase defined by IPsec).

The aggregation property EntriesInFilterList.EntrySequence is always set to 0, to indicate that the aggregated filter entries are ANDed together to form a selector for a class of traffic.

See [PCIME] for the definition of this class.

4.3.33. The Abstract Class ServiceAccessPoint

This is an abstract class defined in the Core Model of CIM. It is a subclass of the LogicalElement class, and is the base class for all objects that manage access to CIM_Services. It represents the management of utilizing or invoking a Service. Please refer to [CIM] for the full definition of this class.

4.3.34. The Class ProtocolEndpoint

This is a concrete class derived from ServiceAccessPoint, which describes a communication point from which the services of the network or the system's protocol stack may be accessed. Please refer to [CIM] for the full definition of this class.

4.3.35. The Abstract Class Collection

This is an abstract class defined in the Core Model of CIM. It is the superclass for all classes that represent groupings or bags, and that carry no status or "state". (The latter would be more correctly modeled as ManagedSystemElements.) Please refer to [CIM] for the full definition of this class.

4.3.36. The Abstract Class CollectionOfMSEs

This is an abstract class defined in the Core Model of CIM. It is a subclass of the Collection superclass, restricting the contents of the Collection to ManagedSystemElements. Please refer to [CIM] for the full definition of this class.

4.3.37. The Class BufferPool

This is a concrete class that represents the collection of buffers used by a QueuingService. (The association QueueAllocation represents this usage.) The existence and management of individual buffers may be modeled in a future document. At the current level of abstraction, modeling the existence of the BufferPool is necessary. Long term, it is not sufficient.

In implementations where there are multiple buffer sizes, an instance of BufferPool should be defined for each set of buffers with identical or similar sizes. These instances of buffer pools can then be grouped together using the CollectedBuffersPool aggregation.

Note that this class is derived from CollectionOfMSEs, and not from Forwarding or ConditioningService. A BufferPool is only a collection of storage, and is NOT a Service.

The class definition is as follows:

NAME	BufferPool
DESCRIPTION	A concrete class representing a collection of buffers.
DERIVED FROM	CollectionOfMSEs
TYPE	Concrete
PROPERTIES	Name, BufferSize, TotalBuffers, AvailableBuffers, SharedBuffers

4.3.37.1. The Property Name

This property is a string with a maximum length of 256 characters. It is the common name or label by which the object is known.

4.3.37.2. The Property BufferSize

This property is a 32-bit unsigned integer, identifying the approximate number of bytes in each buffer in the buffer pool. An implementation will typically group buffers of roughly the same size together, to reduce the number of buffer pools it needs to manage. This model does not specify the degree to which buffers in the same buffer pool may differ in size.

4.3.37.3. The Property TotalBuffers

This property is a 32-bit unsigned integer, reporting the total number of individual buffers in the pool.

4.3.37.4. The Property AvailableBuffers

This property is a 32-bit unsigned integer, reporting the number of buffers in the Pool that are currently not allocated to any instance of a QueuingService. Buffers allocated to a QueuingService could either be in use (that is, currently contain packet data), or be allocated to a queue pending the arrival of new packet data.

4.3.37.5. The Property SharedBuffers

This property is a 32-bit unsigned integer, reporting the number of buffers in the Pool that have been simultaneously allocated to multiple instances of QueuingService.

4.3.38. The Abstract Class SchedulingElement

This is an abstract class that represents the configuration information that a PacketSchedulingService has for one of the elements that it is scheduling. The scheduled element is either a QueuingService or another PacketSchedulingService.

Among the subclasses of this class, some are defined in such a way that all of their instances are work conserving. Other subclasses, however, may have instances that either are or are not work conserving. In this class, the boolean property WorkConserving indicates whether an instance is or is not work conserving. The range of values for WorkConserving is restricted to TRUE in the subclasses that are inherently work conserving, since instances of these classes cannot be anything other than work conserving.

The class definition is as follows:

NAME	SchedulingElement
DESCRIPTION	An abstract class representing the configuration information that a PacketSchedulingService has for one of the elements that it is scheduling.
DERIVED FROM	ManagedElement
TYPE	Abstract
PROPERTIES	WorkConserving

4.3.38.1. The Property WorkConserving

This boolean property indicates whether the PacketSchedulingService tied to this instance by the ElementInSchedulingService aggregation is treating the input tied to this instance by the QueueToSchedule or SchedulingServiceToSchedule association in a work-conserving manner. Note that this property is writable, indicating that an administrator can change the behavior of the SchedulingElement - but only for those elements that can operate in a non-workconserving mode.

4.3.39. The Class AllocationSchedulingElement

This class is a subclass of the abstract class SchedulingElement. It introduces five new properties to support bandwidth-based scheduling. As is the case with all subclasses of SchedulingElement, the input associated with an instance of AllocationSchedulingElement is of one of two types: either a queue, or another scheduler.

The class definition is as follows:

NAME	AllocationSchedulingElement
DESCRIPTION	A concrete class containing parameters for controlling bandwidth-based scheduling.
DERIVED FROM	SchedulingElement
TYPE	Concrete
PROPERTIES	AllocationUnits, BandwidthAllocation, BurstAllocation, CanShare, WorkFlexible

4.3.39.1. The Property AllocationUnits

This property is a 16-bit unsigned integer enumeration that identifies the units in which the BandwidthAllocation and BurstAllocation properties are expressed. The following values are defined:

- o bytes(1)
- o packets(2)
- o cells(3) -- fixed-size, for example, ATM

Note: if the value of AllocationUnits is not one of these three values, it SHOULD be interpreted as if it had the value '1' (bytes).

4.3.39.2. The Property BandwidthAllocation

This property is a 32-bit unsigned integer that defines the number of units/second that should be allocated to the associated input. The units are identified by the AllocationUnits property.

4.3.39.3. The Property BurstAllocation

This property is a 32-bit unsigned integer that specifies the amount of temporary or short-term bandwidth (in units per second) that can be allocated to an input, beyond the amount of bandwidth allocated through the BandwidthAllocation property. If the maximum actual bandwidth allocation for the input were to be measured, it would be the sum of the BurstAllocation and the BandwidthAllocation properties. The units are identified by the AllocationUnits property.

4.3.39.4. The Property CanShare

This is a boolean property that, if TRUE, enables unused bandwidth from the associated input to be allocated to other inputs serviced by the Scheduler.

4.3.39.5. The Property WorkFlexible

This is a boolean property that, if TRUE, indicates that the behavior of the scheduler relative to this input can be altered by changing the value of the inherited property WorkConserving.

4.3.40. The Class WRRSchedulingElement

This class is a subclass of the abstract class SchedulingElement, representing a weighted round robin (WRR) scheduling discipline. It introduces a new property WeightingFactor, to give some inputs a

higher probability of being serviced than other inputs. It also introduces a property Priority, to serve as a tiebreaker to be used when inputs have equal weighting factors. As is the case with all subclasses of SchedulingElement, the input associated with an instance of WRRSchedulingElement is of one of two types: either a queue, or another scheduler.

Because scheduling of this type is always work conserving, the inherited boolean property WorkConserving is restricted to the value TRUE in this class.

The class definition is as follows:

NAME	WRRSchedulingElement
DESCRIPTION	This class specializes the SchedulingElement class to add a per-input weight. This is used by a weighted round robin packet scheduler when it handles its associated inputs. It also adds a second property to serve as a tie-breaker in the case where multiple inputs have been assigned the same weight.
DERIVED FROM	SchedulingElement
TYPE	Concrete
PROPERTIES	WeightingFactor, Priority

4.3.40.1. The Property WeightingFactor

This property is a 32-bit unsigned integer, which defines the weighting factor that offers some inputs a higher probability of being serviced than other inputs. This property represents this probability. Its minimum value is 0, its maximum value is 100000, and its units are in thousandths.

4.3.40.2. The Property Priority

This property is a 16-bit unsigned integer, which serves as a tiebreaker, in the event that two or more inputs have equal weights. A larger value represents a higher priority. If this property is specified for any of the WRRSchedulingElements associated with a PacketSchedulingService, then it must be specified for all WRRSchedulingElements for that PacketSchedulingService, and the property values for these WRRSchedulingElements must all be different.

While this condition may not occur in some implementations of a weighted round-robin scheduler, many implementations require a priority to resolve an equal-weight condition. In instances where this behavior is not necessary or is undesirable, this property may be left unspecified.

4.3.41. The Class PrioritySchedulingElement

This class is a subclass of the abstract class SchedulingElement. It indicates that a scheduler is taking packets from a set of inputs using the priority scheduling discipline. As is the case with all subclasses of SchedulingElement, the input associated with an instance of PrioritySchedulingElement is of one of two types: either a queue, or another scheduler. The property Priority in PrioritySchedulingElement represents the priority for an input, relative to the priorities of all the other inputs to which the scheduler that aggregates this PrioritySchedulingElement is associated. Inputs to which the scheduler is related via other scheduling disciplines do not figure in this prioritization.

Because scheduling of this type is always work conserving, the inherited boolean property WorkConserving is restricted to the value TRUE in this class.

The class definition is as follows:

NAME	PrioritySchedulingElement
DESCRIPTION	A concrete class that specializes the SchedulingElement class to add a Priority property. This property is used by a SchedulingService that is doing priority scheduling for a set of inputs.
DERIVED FROM	SchedulingElement
TYPE	Concrete
PROPERTIES	Priority

4.3.41.1. The Property Priority

This property is a 16-bit unsigned integer that indicates the priority level of a scheduler input relative to the other inputs serviced by this PacketSchedulingService. A larger value represents a higher priority.

4.3.42. The Class BoundedPrioritySchedulingElement

This class is a subclass of the class PrioritySchedulingElement, which is itself derived from the abstract class SchedulingElement. As is the case with all subclasses of SchedulingElement, the input associated with an instance of BoundedPrioritySchedulingElement is of one of two types: either a queue, or another scheduler. BoundedPrioritySchedulingElement adds an upper bound (in kilobits per second) on how much traffic can be handled from an input. This data is specific to that one input. It is needed when bounded strict priority scheduling is performed.

This class inherits from its superclass PrioritySchedulingElement the restriction of the inherited boolean property WorkConserving to the value TRUE.

The class definition is as follows:

NAME	BoundedPrioritySchedulingElement
DESCRIPTION	This concrete class specializes the PrioritySchedulingElement class to add a BandwidthBound property. This property bounds the rate at which traffic from the associated input can be handled.
DERIVED FROM	PrioritySchedulingElement
TYPE	Concrete
PROPERTIES	BandwidthBound

4.3.42.1. The Property BandwidthBound

This property is a 32-bit unsigned integer that defines the upper limit on the amount of traffic that can be handled from the input. This is not a shaped upper bound, since bursts can occur. It is a strict bound, limiting the impact of the input. The units are kilobits per second.

4.4. Association Definitions

This section details the QoS device datapath associations, including the aggregations, which were shown earlier in Figures 4 and 5. These associations are defined as classes in the Information Model. Each of these classes has two properties referring to instances of the two classes that the association links. Some of the association classes have additional properties as well.

4.4.1. The Abstract Association Dependency

This abstract association defines two object references (named Antecedent and Dependent) that establish general dependency relationships between different managed objects in the information model. The Antecedent reference identifies the independent object in the association, while the Dependent reference identifies the entity that IS dependent.

The association’s cardinality is many to many.

The association is defined in the Core Model of CIM. Please refer to [CIM] for the full definition of this class.

4.4.2. The Association ServiceSAPDependency

This association defines two object references that establish a general dependency relationship between a Service object and a ServiceAccessPoint object. This relationship indicates that the referenced Service uses the ServiceAccessPoint of ANOTHER Service. The Service is the Dependent reference, relying on the ServiceAccessPoint to gain access to another Service.

The association’s cardinality is many to many.

The association is defined in the Core Model of CIM. Please refer to [CIM] for the full definition of this class.

4.4.3. The Association IngressConditioningServiceOnEndpoint

This association is derived from the association ServiceSAPDependency, and represents the binding, in the ingress direction, between a protocol endpoint and the first ConditioningService that processes packets received via that protocol endpoint. Since there can only be one "first" ConditioningService for a protocol endpoint, the cardinality for the Dependent object reference is narrowed from 0..n to 0..1. Since, on the other hand, a single ConditioningService can be the first to process packets received via multiple protocol endpoints, the cardinality of the Antecedent object reference remains 0..n.

The class definition is as follows:

NAME	IngressConditioningServiceOnEndpoint
DESCRIPTION	An association that establishes a dependency relationship between a protocol endpoint and the first conditioning service that processes traffic arriving via that protocol endpoint.
DERIVED FROM	ServiceSAPDependency
ABSTRACT	False
PROPERTIES	Antecedent[ref ProtocolEndpoint[0..n]], Dependent[ref ConditioningService[0..1]]

4.4.4. The Association EgressConditioningServiceOnEndpoint

This association is derived from the association ServiceSAPDependency, and represents the binding, in the egress direction, between a protocol endpoint and the last ConditioningService that processes packets before they leave a network device via that protocol endpoint. (This "last" ConditioningService is ordinarily a scheduler, but it doesn't have to be.) Since there can be multiple "last" ConditioningServices for a protocol endpoint in the case of a fallback scheduler, the cardinality for the Dependent object reference remains 0..n. Since, however, a single ConditioningService cannot be the last one to process packets for multiple protocol endpoints, the cardinality of the Antecedent object reference is narrowed from 0..n to 0..1.

The class definition is as follows:

NAME	EgressConditioningServiceOnEndpoint
DESCRIPTION	An association that establishes a dependency relationship between a protocol endpoint and the last conditioning service(s) that process traffic to be transmitted via that protocol endpoint.
DERIVED FROM	ServiceSAPDependency
ABSTRACT	False
PROPERTIES	Antecedent[ref ProtocolEndpoint[0..1]], Dependent[ref ConditioningService[0..n]]

4.4.5. The Association HeadTailDropQueueBinding

This association is a subclass of Dependency, describing the association between a head or tail dropper and a queue that it monitors to determine when to drop traffic. The referenced queue is the one whose queue depth is compared against the Dropper's threshold. The cardinality is 1..n on the queue side, since a

head/tail dropper must monitor at least one queue. For the classes HeadTailDropper and HeadTailDropQueueBinding, the rule for combining the inputs from multiple queues is simple addition: if the sum of the lengths of the monitored queues exceeds the dropper's QueueThreshold value, then packets are dropped. This rule for combining inputs may, however, be overridden by a different rule in subclasses of one or both of these classes.

The class definition is as follows:

NAME	HeadTailDropQueueBinding
DESCRIPTION	A generic association used to establish a dependency relationship between a head or tail dropper and a queue that it monitors.
DERIVED FROM	Dependency
ABSTRACT	False
PROPERTIES	Antecedent[ref QueuingService[1..n]], Dependent[ref HeadTailDropperService [0..n]]

4.4.6. The Association CalculationBasedOnQueue

This association is a subclass of Dependency, which defines two object references that establish a dependency relationship between a QueuingService and an instance of the DropThresholdCalculationService class. The queue's current depth is used by the calculation service in calculating an average queue depth.

The class definition is as follows:

NAME	CalculationBasedOnQueue
DESCRIPTION	A generic association used to establish a dependency relationship between a QueuingService object and a DropThresholdCalculationService object.
DERIVED FROM	ServiceServiceDependency
ABSTRACT	False
PROPERTIES	Antecedent[ref QueuingService[1..1]], Dependent[ref DropThresholdCalculationService [0..n]]

4.4.6.1. The Reference Antecedent

This property is inherited from the Dependency association, and overridden to serve as an object reference to a QueuingService object (instead of to the more general ManagedElement). This reference identifies the queue that the DropThresholdCalculationService will use in its calculation of average queue depth.

4.4.6.2. The Reference Dependent

This property is inherited from the Dependency association, and overridden to serve as an object reference to a DropThresholdCalculationService object (instead of to the more general ManagedElement). This reference identifies a DropThresholdCalculationService that uses the referenced queue's current depth as one of the inputs to its calculation of average queue depth.

4.4.7. The Association ProvidesServiceToElement

This association defines two object references that establish a dependency relationship in which a ManagedSystemElement depends on the functionality of one or more Services. The association's cardinality is many to many.

The association is defined in the Core Model of CIM. Please refer to [CIM] for the full definition of this class.

4.4.8. The Association ServiceServiceDependency

This association defines two object references that establish a dependency relationship between two Service objects. The particular type of dependency is represented by the TypeOfDependency property; typical examples include that one Service is required to be present or required to have completed for the other Service to operate.

This association is very similar to the ServiceSAPDependency relationship. For the latter, the Service is dependent on an AccessPoint to get at another Service. In this relationship, it directly identifies its Service dependency. Both relationships should not be instantiated, since their information is repetitive.

The association's cardinality is many to many.

The association is defined in the Core Model of CIM. Please refer to [CIM] for the full definition of this class.

4.4.9. The Association CalculationServiceForDropper

This association is a subclass of ServiceServiceDependency, which defines two object references that represent the reliance of a REDDropperService on a DropThresholdCalculationService - calculating an average queue depth based on the observed depths of one or more queues.

The class definition is as follows:

NAME	CalculationServiceForDropper
DESCRIPTION	A generic association used to establish a dependency relationship between a calculation service and a REDDropperSrevice for which it performs average queue depth calculations
DERIVED FROM	ServiceServiceDependency
ABSTRACT	False
PROPERTIES	Antecedent[ref DropThresholdCalculationService[1..n]], Dependent[ref REDDropperService[0..n]]

4.4.9.1. The Reference Antecedent

This property is inherited from the ServiceServiceDependency association, and overridden to serve as an object reference to a DropThresholdCalculationService object (instead of to the more general Service object). The cardinality of the object reference is 1..n, indicating that a RED dropper may be served by one or more calculation services.

4.4.9.2. The Reference Dependent

This property is inherited from the ServiceServiceDependency association, and overridden to serve as an object reference to a REDDropperService object (instead of to the more general Service object). This reference identifies a RED dropper served by a DropThresholdCalculationService.

4.4.10. The Association QueueAllocation

This association is a subclass of Dependency, which defines two object references that establish a dependency relationship between a QueuingService and a BufferPool that provides storage space for the packets in the queue.

The class definition is as follows:

NAME	QueueAllocation
DESCRIPTION	A generic association used to establish a dependency relationship between a QueuingService object and a BufferPool object.
DERIVED FROM	Dependency
ABSTRACT	False
PROPERTIES	Antecedent[ref BufferPool[0..n]], Dependent[ref QueuingService[0..n]] AllocationPercentage

4.4.10.1. The Reference Antecedent

This property is inherited from the Dependency association, and overridden to serve as an object reference to a BufferPool object. This reference identifies the BufferPool in which packets on the QueuingService's queue are stored.

4.4.10.2. The Reference Dependent

This property is inherited from the Dependency association, and overridden to serve as an object reference to a QueuingService object. This reference identifies the QueuingService whose packets are being stored in the BufferPool's buffers.

4.4.10.3. The Property AllocationPercentage

This property is an 8-bit unsigned integer with minimum value of zero and maximum value of 100. It defines the percentage of the BufferPool that should be allocated to the referenced QueuingService. If absolute sizes are desired, this would be accomplished by defining individual BufferPools of the specified sizes, with QueueAllocation.AllocationPercentages set to 100.

4.4.11. The Association ClassifierElementUsesFilterList

This association is a subclass of the Dependency association. It relates one or more ClassifierElements with a FilterList representing the criteria for selecting packets for each of the ClassifierElements to process.

In the QDDIM model, a classifier is always modeled as a ClassifierService that aggregates a set of ClassifierElements. When ClassifierElements use the NextServiceAfterClassifierElement

association to bind to another ClassifierService (to construct a hierarchical classifier), the ClassifierElementUsesFilterList association must not be specified.

The class definition is as follows:

NAME	ClassifierElementUsesFilterList
DESCRIPTION	An association relating a ClassifierElement to the FilterList representing the criteria for selecting packets for that ClassifierElement to process.
DERIVED FROM	Dependency
ABSTRACT	False
PROPERTIES	Antecedent[ref FilterList [0..1]], Dependent[ref ClassifierElement [0..n]]

4.4.11.1. The Reference Antecedent

This property is inherited from the Dependency association, and overridden to serve as an object reference to a FilterList object, instead of to the more general ManagedElement object. Also, its cardinality is restricted to 0 and 1, indicating that a ClassifierElement uses either one FilterList to select packets for it or no FilterList when the ClassifierElement uses the NextServiceAfterClassifierElement association to bind to another ClassifierService to form a hierarchical classifier.

4.4.11.2. The Reference Dependent

This property is inherited from the Dependency association, and overridden to serve as an object reference to a ClassifierElement object, instead of to the more general ManagedElement object. This reference identifies a ClassifierElement that depends on the associated FilterList object to represent its packet-selection criteria.

4.4.12. The Association AFRelatedServices

This association defines two object references that establish a dependency relationship between two AFService objects. This dependency is the precedence of the individual AF drop-related Services within an AF IP packet-forwarding class.

The class definition is as follows:

```
NAME              AFRelatedServices
DESCRIPTION       An association used to establish
                  a dependency relationship between two
                  AFService objects.
DERIVED FROM      Nothing
ABSTRACT          False
PROPERTIES        AFLowerDropPrecedence[ref
                    AFService[0..1]],
                  AFHigherDropPrecedence[ref
                    AFService[0..n]]
```

4.4.12.1. The Reference AFLowerDropPrecedence

This property serves as an object reference to an AFService object that has the lower probability of dropping packets.

4.4.12.2. The Reference AFHigherDropPrecedence

This property serves as an object reference to an AFService object that has the higher probability of dropping packets.

4.4.13. The Association NextService

This association defines two object references that establish a predecessor-successor relationship between two ConditioningService objects. This association is used to indicate the sequence of ConditioningServices required to process a particular type of traffic.

Instances of this dependency describe the various relationships between different ConditioningServices (such as classifiers, meters, droppers, etc.) that are used collectively to condition traffic. Both one-to-one and more complicated fan-in and/or fan-out relationships can be described. The ConditioningServices may feed one another directly, or they may be mapped to multiple "next" Services based on the characteristics of the packet.

The class definition is as follows:

```
NAME              NextService
DESCRIPTION       An association used to establish
                  a predecessor-successor relationship
                  between two ConditioningService objects.
DERIVED FROM      Nothing
ABSTRACT          False
PROPERTIES        PrecedingService[ref
                    ConditioningService[0..n]],
                  FollowingService[ref
                    ConditioningService[0..n]]
```

4.4.13.1. The Reference PrecedingService

This property serves as an object reference to a ConditioningService object that occurs earlier in the processing sequence for a given type of traffic.

4.4.13.2. The Reference FollowingService

This property serves as an object reference to a ConditioningService object that occurs later in the processing sequence for a given type of traffic, immediately after the ConditioningService identified by the PrecedingService object reference.

4.4.14. The Association NextServiceAfterClassifierElement

This association refines the definition of its superclass, the NextService association, in two ways:

- It restricts the PrecedingService object reference to the class ClassifierElement.
- It restricts the cardinality of the FollowingService object reference to exactly 1.

The class definition is as follows:

```
NAME              NextServiceAfterClassifierElement
DESCRIPTION       An association used to establish
                  a predecessor-successor relationship
                  between a single ClassifierElement within
                  a Classifier and the next
                  ConditioningService object that is
                  responsible for further processing of
                  the traffic selected by that
                  ClassifierElement.
```

```
DERIVED FROM      NextService
ABSTRACT          False
PROPERTIES        PrecedingService
                    [ref ClassifierElement[0..n]],
                  FollowingService
                    [ref ConditioningService[1..1]
```

4.4.14.1. The Reference PrecedingService

This property is inherited from the NextService association. It is overridden in this subclass to restrict the object reference to a ClassifierElement, as opposed to the more general ConditioningService defined in the NextService superclass.

This property serves as an object reference to a ClassifierElement, which is a component of a single ClassifierService. Packets selected by this ClassifierElement are always passed to the ConditioningService identified by the FollowingService object reference.

4.4.14.2. The Reference FollowingService

This property is inherited from the NextService association. It is overridden in this subclass to restrict the cardinality of the reference to exactly 1. This reflects the requirement that the behavior of a DiffServ classifier must be deterministic: the packets selected by a given ClassifierElement in a given ClassifierService must always go to one and only one next ConditioningService.

4.4.15. The Association NextScheduler

This association is a subclass of NextService, and defines two object references that establish a predecessor-successor relationship between PacketSchedulingServices. In a hierarchical queuing configuration where a second scheduler treats the output of a first scheduler as a single, aggregated input, the two schedulers are related via the NextScheduler association.

The class definition is as follows:

```
NAME              NextScheduler
DESCRIPTION       An association used to establish
                  predecessor-successor relationships
                  between PacketSchedulingService objects
                  for simple hierarchical scheduling.
DERIVED FROM      NextService
ABSTRACT          False
```

```
PROPERTIES        PrecedingService[ref
                     PacketSchedulingService[0..n]],
                  FollowingService[ref
                     PacketSchedulingService[0..1]]
```

4.4.15.1. The Reference PrecedingService

This property is inherited from the NextService association, and overridden to serve as an object reference to a PacketSchedulingService object (instead of to the more general ConditioningService object). This reference identifies a scheduler whose output is being treated as a single, aggregated input by the scheduler identified by the FollowingService reference. The [0..n] cardinality indicates that a single FollowingService scheduler may bring together the aggregated outputs of multiple prior schedulers.

4.4.15.2. The Reference FollowingService

This property is inherited from the NextService association, and overridden to serve as an object reference to a PacketSchedulingService object (instead of to the more general ConditioningService object). This reference identifies a scheduler that includes among its inputs the aggregated outputs of one or more PrecedingService schedulers.

4.4.16. The Association FailNextScheduler

This association is a subclass of the NextScheduler association. FailNextScheduler represents the relationship between two schedulers when the first scheduler passes up a scheduling opportunity (thereby behaving in a non-work conserving manner), and makes the resulting bandwidth available to the second scheduler for its use. See Sections 3.11.3 and 3.11.4 for examples of where this association might be used.

The class definition is as follows:

```
NAME              FailNextScheduler
DESCRIPTION       This association specializes the
                  NextScheduler association.  It
                  establishes a relationship between a
                  non-work-conserving scheduler and a
                  second scheduler to which it makes
                  available the bandwidth that it elects
                  not to use.
DERIVED FROM      NextScheduler
ABSTRACT          False
```

```
PROPERTIES        PrecedingService[ref
                   NonWorkConservingSchedulingService[0..n]]
```

4.4.16.1. The Reference PrecedingService

This property is inherited from the NextScheduler association, and overridden to serve as an object reference to a NonWorkConservingSchedulingService object (instead of to the more general PacketSchedulingService object). This reference identifies a non-work-conserving scheduler whose excess bandwidth is being made available to the scheduler identified by the FollowingService reference. The [0..n] cardinality indicates that a single FollowingService scheduler may have the opportunity to use the unused bandwidth of multiple prior non-work-conserving schedulers.

4.4.17. The Association NextServiceAfterMeter

This association describes a predecessor-successor relationship between a MeterService and one or more ConditioningService objects that process traffic from the meter. For example, for devices that implement preamble marking, the FollowingService reference (after the meter) is a PreambleMarkerService, to record the results of the metering in the preamble.

It might be expected that the NextServiceAfterMeter association would subclass from NextService. However, meters are 1:n fan-out elements, and require a mechanism to distinguish between the different results/outputs of the meter. Therefore, this association defines a new key property, MeterResult, which is used to record the result and identify the output through which this traffic left the meter. Because of this additional key, NextServiceAfterMeter cannot be a subclass of NextService.

The class definition is as follows:

```
NAME              NextServiceAfterMeter
DESCRIPTION       An association used to establish
                  a predecessor-successor relationship
                  between a particular output of a
                  MeterService and the next
                  ConditioningService object that is
                  responsible for further processing of
                  the traffic.
DERIVED FROM      Nothing
ABSTRACT          False
```

```
PROPERTIES      PrecedingService[ref MeterService[0..n]],
                FollowingService[ref
                  ConditioningService[0..n]],
                MeterResult
```

4.4.17.1. The Reference PrecedingService

The preceding MeterService, 'earlier' in the processing sequence for a packet. Since Meters are 1:n fan-out devices, this relationship associates a particular output of a MeterService (identified by the MeterResult property) to the next ConditioningService that is used to further process the traffic.

4.4.17.2. The Reference FollowingService

The 'next' or following ConditioningService.

4.4.17.3. The Property MeterResult

This property is an enumerated 16-bit unsigned integer, and represents information describing the result of the metering. Traffic is distinguished as being conforming, non-conforming, or partially conforming. More complicated metering can be built either by extending the enumeration or by cascading meters.

The enumerated values are: "Unknown" (0), "Conforming" (1), "PartiallyConforming" (2), "NonConforming" (3).

4.4.18. The Association QueueToSchedule

This is a top-level association, representing the relationship between a queue (QueuingService) and a SchedulingElement. The SchedulingElement, in turn, represents the information in a packet scheduling service that is specific to this queue, such as relative priority or allocated bandwidth.

It cannot be expressed formally with the association cardinalities, but there is an additional constraint on participation in this association. A particular instance of (a subclass of) SchedulingElement always participates either in exactly one instance of this association, or in exactly one instance of the association SchedulingServiceToSchedule.

The class definition is as follows:

```
NAME              QueueToSchedule
DESCRIPTION       This association relates a queue to
                  the SchedulingElement containing
                  information specific to the queue.
DERIVED FROM      Nothing
ABSTRACT          False
PROPERTIES        Queue[ref QueuingService[0..1]],
                  SchedElement[ref
                     SchedulingElement[0..n]]
```

4.4.18.1. The Reference Queue

This property serves as an object reference to a QueuingService object. A QueuingService object may be associated 0 or more SchedulingElement objects.

4.4.18.2. The Reference SchedElement

This property serves as an object reference to a SchedulingElement object. A SchedulingElement is always associated either with exactly one QueuingService or with exactly one upstream scheduler (PacketSchedulingService).

4.4.19. The Association SchedulingServiceToSchedule

This is a top-level association, representing the relationship between a scheduler (PacketSchedulingService) and a SchedulingElement, in a configuration involving cascaded schedulers. The SchedulingElement, in turn, represents the information in a subsequent packet scheduling service that is specific to this scheduler, such as relative priority or allocated bandwidth.

It cannot be expressed formally with the association cardinalities, but there is an additional constraint on participation in this association. A particular instance of (a subclass of) SchedulingElement always participates either in exactly one instance of this association, or in exactly one instance of the association QueueToSchedule.

The class definition is as follows:

```
NAME              SchedulingServiceToSchedule
DESCRIPTION       This association relates a scheduler to
                  the SchedulingElement in a subsequent
                  scheduler containing information specific
                  to this scheduler.
```

DERIVED FROM	Nothing
ABSTRACT	False
PROPERTIES	SchedService[ref PacketSchedulingService[0..1]], SchedElement[ref SchedulingElement[0..n]]

4.4.19.1. The Reference SchedService

This property serves as an object reference to a PacketSchedulingService object. A PacketSchedulingService object may be associated 0 or more SchedulingElement objects.

4.4.19.2. The Reference SchedElement

This property serves as an object reference to a SchedulingElement object. A SchedulingElement is always associated either with exactly one QueuingService or with exactly one upstream scheduler (PacketSchedulingService).

4.4.20. The Aggregation MemberOfCollection

This aggregation is a generic relationship used to model the aggregation of a set of ManagedElements in a generalized Collection object. The aggregation’s cardinality is many to many.

MemberOfCollection is defined in the Core Model of CIM. Please refer to [CIM] for the full definition of this class.

4.4.21. The Aggregation CollectedBufferPool

This aggregation models the ability to treat a set of buffers as a pool, or collection, that can in turn be contained in a "higher-level" buffer pool. This class overrides the more generic MemberOfCollection aggregation to restrict both the aggregate and the part component objects to be instances only of the BufferPool class.

The class definition for the aggregation is as follows:

NAME	CollectedBufferPool
DESCRIPTION	A generic association used to aggregate a set of related buffers into a higher-level buffer pool.
DERIVED FROM	MemberOfCollection
ABSTRACT	False
PROPERTIES	Collection[ref BufferPool[0..1]], Member[ref BufferPool[0..n]]

4.4.21.1. The Reference Collection

This property represents the parent, or aggregate, object in the relationship. It is a BufferPool object.

4.4.21.2. The Reference Member

This property represents the child, or lower level pool, in the relationship. It is one of the set of BufferPools that together make up the higher-level pool.

4.4.22. The Abstract Aggregation Component

This abstract aggregation is a generic relationship used to establish "part-of" relationships between managed objects (named GroupComponent and PartComponent). The association's cardinality is many to many.

The association is defined in the Core Model of CIM. Please refer to [CIM] for the full definition of this class.

4.4.23. The Aggregation ServiceComponent

This aggregation is used to model a set of subordinate Services that are aggregated together to form a higher-level Service. This aggregation is derived from the more generic Component superclass to restrict the types of objects that can participate in this relationship. The association's cardinality is many to many.

The association is defined in the Core Model of CIM. Please refer to [CIM] for the full definition of this class.

4.4.24. The Aggregation QoSSubService

This aggregation represents a set of subordinate QoSService objects (that is, a set of instances of subclasses of the QoSService class) that are aggregated together to form a higher-level QoSService. A QoSService is a specific type of Service that conceptualizes QoS functionality as a set of coordinated sub-services.

This aggregation is derived from the more generic ServiceComponent superclass to restrict the types of objects that can participate in this relationship to QoSService objects, instead of a more generic Service object. It also restricts the cardinality of the aggregate to 0-or-1 (instead of the more generic 0-or-more).

The class definition for the aggregation is as follows:

NAME	QoSSubService
DESCRIPTION	A generic association used to establish "part-of" relationships between a higher-level QoSService object and the set of lower-level QoSServices that are aggregated to create/form it.
DERIVED FROM	ServiceComponent
ABSTRACT	False
PROPERTIES	GroupComponent[ref QoSService[0..1]], PartComponent[ref QoSService[0..n]]

4.4.24.1. The Reference GroupComponent

This property is overridden in this aggregation to represent an object reference to a QoSService object (instead of to the more generic Service object defined in its superclass). This object represents the parent, or aggregate, object in the relationship.

4.4.24.2. The Reference PartComponent

This property is overridden in this aggregation to represent an object reference to a QoSService object (instead of to the more generic Service object defined in its superclass). This object represents the child, or "component", object in the relationship.

4.4.25. The Aggregation QoSConditioningSubService

This aggregation identifies the set of conditioning services that together condition traffic for a particular QoS service.

This aggregation is derived from the more generic ServiceComponent superclass; it restricts the types of objects that can participate in it to ConditioningService and QoSService objects, instead of the more generic Service objects.

The class definition for the aggregation is as follows:

NAME	QoSConditioningSubService
DESCRIPTION	A generic aggregation used to establish "part-of" relationships between a set of ConditioningService objects and the particular QoSService object(s) that they provide traffic conditioning for.
DERIVED FROM	ServiceComponent
ABSTRACT	False

```
PROPERTIES        GroupComponent[ref QoSService[0..n]],
                  PartComponent[ref
                    ConditioningService[0..n]]
```

4.4.25.1. The Reference GroupComponent

This property is overridden in this aggregation to represent an object reference to a QoSService object (instead of to the more generic Service object defined in its superclass). The cardinality of the reference remains 0..n, to indicate that a given ConditioningService may provide traffic conditioning for 0, 1, or more than 1 QoSService objects.

This object represents the parent, or aggregate, object in the association. In this case, this object represents the QoSService that aggregates one or more ConditioningService objects to implement the appropriate traffic conditioning for its traffic.

4.4.25.2. The Reference PartComponent

This property is overridden in this aggregation to represent an object reference to a ConditioningService object (instead of to the more generic Service object defined in its superclass). This object represents the child, or "component", object in the relationship. In this case, this object represents one or more ConditioningService objects that together indicate how traffic for a specific QoSService is conditioned.

4.4.26. The Aggregation ClassifierElementInClassifierService

This aggregation represents the relationship between a classifier and the classifier elements that provide the fan-out function for the classifier. A classifier typically aggregates multiple classifier elements. A classifier element, however, is aggregated only by a single classifier. See [DSMODEL] and [DSMIB] for more about classifiers and classifier elements.

The class definition for the aggregation is as follows:

```
NAME              ClassifierElementInClassifierService
DESCRIPTION       An aggregation representing the
                  relationship between a classifier
                  and its classifier elements.
DERIVED FROM      ServiceComponent
ABSTRACT          False
```

```
PROPERTIES      GroupComponent[ref
                   ClassifierService[1..1]],
                PartComponent[ref
                   ClassifierElement[0..n],
                ClassifierOrder
```

4.4.26.1. The Reference GroupComponent

This property is overridden in this aggregation to represent an object reference to a ClassifierService object (instead of to the more generic Service object defined in its superclass). It also restricts the cardinality of the aggregate to 1..1 (instead of the more generic 0-or-more), representing the fact that a ClassifierElement always exists within the context of exactly one ClassifierService.

4.4.26.2. The Reference PartComponent

This property is overridden in this aggregation to represent an object reference to a ClassifierElement object (instead of to the more generic Service object defined in its superclass). This object represents a single traffic selector for the classifier. A ClassifierElement usually has an association to a FilterList that provides selection criteria for packets from the traffic stream coming into the classifier, and to a ConditioningService to which packets selected by these criteria are next forwarded.

4.4.26.3. The Property ClassifierOrder

Because the filters for a classifier can overlap, it is necessary to specify the order in which the ClassifierElements aggregated by a ClassifierService are presented with packets coming into the classifier. This property is an unsigned 32-bit integer representing this order. Values are represented in ascending order: first '1', then '2', and so on. Different values MUST be assigned for each of the ClassifierElements aggregated by a given ClassifierService.

4.4.27. The Aggregation EntriesInFilterList

This aggregation is a specialization of the Component aggregation; it is used to define a set of filter entries (subclasses of FilterEntryBase) that are aggregated by a FilterList.

The cardinalities of the aggregation itself are 0..1 on the FilterList end, and 0..n on the FilterEntryBase end. Thus in the general case, a filter entry can exist without being aggregated into

any FilterList. However, the only way a filter entry can figure in the QoS Device model is by being aggregated into a FilterList by this aggregation.

See [PCIME] for the definition of this aggregation.

4.4.28. The Aggregation ElementInSchedulingService

This concrete aggregation represents the relationship between a PacketSchedulingService and the set of SchedulingElements that tie it to its inputs.

The class definition for the aggregation is as follows:

NAME	ElementInSchedulingService
DESCRIPTION	An aggregation used to tie a PacketSchedlingService to the configuration information for one of the elements (either a QueuingService or another PacketSchedulingService) that it schedules.
DERIVED FROM	Component
ABSTRACT	False
PROPERTIES	GroupComponent[ref PacketSchedulingService[0..1]], PartComponent[ref SchedulingElement[1..n]

4.4.28.1. The Reference GroupComponent

This property is overridden in this aggregation to represent an object reference to a PacketSchedulingService object (instead of to the more generic Service object defined in its superclass). It also restricts the cardinality of the aggregate to 0..1 (instead of the more generic 0-or-more), representing the fact that a SchedulingElement exists within the context of at most one PacketSchedulingService.

4.4.28.2. The Reference PartComponent

This property is overridden in this aggregation to represent an object reference to a SchedulingElement object (instead of to the more generic Service object defined in its superclass). This object represents a single scheduling element for the scheduler. It also restricts the cardinality of the SchedulingElement to 1..n (instead of the more generic 0-or-more), representing the fact that a PacketSchedulingService always includes at least one SchedulingElement.

5. Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11.

Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

6. Acknowledgements

The authors wish to thank the participants of the Policy Framework and Differentiated Services working groups for their many helpful comments and suggestions. Special thanks to Joel Halpern, who provided some key technical direction during the latter stages of the document's development.

7. Security Considerations

Like [PCIM] and [PCIME], this document defines an information model that cannot be implemented directly. Consequently, security issues do not arise until it is mapped to an actual, implementable data model such as a MIB, PIB, or LDAP schema. See [PCIM] for a general discussion of security considerations for information models. See also [DSMIB] (which in fact is a data model that corresponds to a large extent with the QDDIM information model), for a discussion of the security implications of specific objects in the model.

8. References

8.1. Normative References

[CIM] Common Information Model (CIM) Schema, version 2.5. Distributed Management Task Force, Inc., available at http://www.dmtf.org/standards/cim_schema_v25.php.

[IEEE802Q] Virtual Bridged Local Area Networks, ANSI/IEEE std 802.1Q, 1998 edition. Approved December 8, 1998

[PCIM] Moore, B., Ellesson, E., Strassner, J. and A. Westerinen, "Policy Core Information Model - Version 1 Specification", RFC 3060, February 2001.

[PCIME] Moore, B., Ed., "Policy Core Information Model (PCIM) Extensions", RFC 3460, January 2003.

[R791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[R2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[R2474] Nichols, K., Blake, S., Baker, F. and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.

[R2597] Heinanen, J., Baker, F., Weiss, W. and J. Wroclawski, "Assured Forwarding PHB Group", RFC 2597, June 1999.

[R3140] Black, D., Brim, S., Carpenter, B. and F. Le Faucheur, "Per Hop Behavior Identification Codes", RFC 3140, June 2001.

8.2. Informative References

[DSMIB] Baker, F., Chan, K. and A. Smith, "Management Information Base for the Differentiated Services Architecture", RFC 3289, May 2002.

[DSMODEL] Bernet, Y., Blake, S., Grossman, D. and A. Smith, "An Informal Management Model for DiffServ Routers", RFC 3290, May 2002.

[PIB] Chan, K., Sahita, R., Hahn, S. and K. McCloghrie, "Differentiated Services Quality of Service Policy Information Base", RFC 3317, March 2003.

[POLTERM] Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J. and S. Waldbusser, "Terminology for Policy-Based Management", RFC 3198, November 2001.

[QPIM] Snir, Y., Ramberg, Y., Strassner, J., Cohen, R. and B. Moore, "Policy Quality of Service (QoS) Information Model", RFC 3644, November 2003.

[R1633] Braden, R., Clark, D. and S. Shenker, "Integrated Services in the Internet Architecture: An Overview", RFC 1633, June 1994.

[R2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z. and W. Weiss, "An Architecture for Differentiated Service", RFC 2475, December 1998.

[R3246] Davie, B., Charny, A., Bennet, J.C.R., Benson, K., Le Boudec, J.Y., Courtney, W., Davari, S., Firoiu, V. and D. Stiliadis, "An Expedited Forwarding PHB (Per-Hop Behavior)", RFC 3246, March 2002.

[RED] See http://www.aciri.org/floyd/red.html

9. Appendix A: Naming Instances in a Native CIM Implementation

Following the precedent established in [PCIM], this document has placed the details of how to name instances of its classes in a native CIM implementation here in an appendix. Since Appendix A in [PCIM] has a lengthy discussion of the general principles of CIM naming, this appendix does not repeat that information here. Readers interested in a more global discussion of how instances are named in a native CIM implementation should refer to [PCIM].

9.1. Naming Instances of the Classes Derived from Service

Most of the classes defined in this model are derived from the CIM class Service. Although Service is an abstract class, it nevertheless has key properties included as part of its definition. The purpose of including key properties in an abstract class is to have instances of all of its instantiable subclasses named in the same way. Thus, the majority of the classes in this model name their instances in exactly the same way: with the two key properties CreationClassName and Name that they inherit from Service.

9.2. Naming Instances of Subclasses of FilterEntryBase

Like Service, FilterEntryBase (defined in [PCIME]) is an abstract class that includes key properties in its definition. FilterEntryBase has four key properties. Two of them, SystemCreationClassName and SystemName, are propagated to it via the weak association FilterEntryInSystem. The other two, CreationClassName and Name, are native to FilterEntryBase.

Thus, instances of all of the subclasses of FilterEntryBase, including the PreambleFilter class defined here, are named in the same way: with the four key properties they inherit from FilterEntryBase.

9.3. Naming Instances of ProtocolEndpoint

The class ProtocolEndpoint inherits its key properties from its superclass, ServiceAccessPoint. These key properties provide the same naming structure that we've seen before: two propagated key properties SystemCreationClassName and SystemName, plus two native key properties CreationClassName and Name.

9.4. Naming Instances of BufferPool

Unlike the other classes in this model, BufferPool is not derived from Service. Consequently, it does not inherit its key properties from Service. Instead, it inherits one of its key properties, CollectionID, from its superclass Collection, and adds its other key property, CreationClassName, in its own definition.

9.4.1. The Property CollectionID

CollectionID is a string property with a maximum length of 256 characters. It identifies the buffer pool. Note that this property is defined in the BufferPool class's superclass, CollectionOfMSEs, but not as a key property. It is overridden in BufferPool, to make it part of this class's composite key.

9.4.2. The Property CreationClassName

This property is a string property of with a maximum length of 256 characters. It is set to "CIM_BufferPool" if this class is directly instantiated, or to the class name of the BufferPool subclass that is created.

9.5. Naming Instances of SchedulingElement

This class has not yet been incorporated into the CIM model, so it does not have any CIM naming properties yet. If the normal pattern is followed, however, instances will be named with two properties CreationClassName and Name.

10. Authors' Addresses

Bob Moore
P. O. Box 12195, BRQA/B501/G206
3039 Cornwallis Rd.
Research Triangle Park, NC 27709-2195

Phone: (919) 254-4436
EMail: remoore@us.ibm.com

David Durham
Intel
2111 NE 25th Avenue
Hillsboro, OR 97124

Phone: (503) 264-6232
EMail: david.durham@intel.com

John Strassner
INTELLIDEN, Inc.
90 South Cascade Avenue
Colorado Springs, CO 80903

Phone: (719) 785-0648
EMail: john.strassner@intelliden.com

Andrea Westerinen
Cisco Systems, Bldg 20
725 Alder Drive
Milpitas, CA 95035

EMail: andreaw@cisco.com

Walter Weiss
Ellacoya Networks
7 Henry Clay Dr.
Merrimack, NH 03054

Phone: (603) 879-7364
EMail: walterweiss@attbi.com

11. Full Copyright Statement

Copyright (C) The Internet Society (2004). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assignees.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.