Network Working Group
Request for Comments: 3644
Category: Standards Track

Y. Snir
Y. Ramberg
Cisco Systems
J. Strassner
Intelliden
R. Cohen
Ntear LLC
B. Moore
IBM

November 2003

# Policy Quality of Service (QoS) Information Model

## Status of this Memo



## Copyright Notice



## Abstract

This document presents an object-oriented information model for representing Quality of Service (QoS) network management policies. This document is based on the IETF Policy Core Information Model and its extensions. It defines an information model for QoS enforcement for differentiated and integrated services using policy. It is important to note that this document defines an information model, which by definition is independent of any particular data storage mechanism and access protocol.

Table of Contents

# 1. Introduction

The QoS Policy Information Model (QPIM) establishes a standard framework and constructs for specifying and representing policies that administer, manage, and control access to network QoS resources. Such policies will be referred to as "QoS policies" in this document. The framework consists of a set of classes and relationships that are organized in an object-oriented information model. It is agnostic of any specific Policy Decision Point (PDP) or Policy Enforcement Point (PEP) (see [TERMS] for definitions) implementation, and independent of any particular QoS implementation mechanism.

QPIM is designed to represent QoS policy information for large-scale policy domains (the term "policy domain" is defined in [TERMS]). A primary goal of this information model is to assist human administrators in their definition of policies to control QoS resources (as opposed to individual network element configuration). The process of creating QPIM data instances is fed by business rules, network topology and QoS methodology (e.g., Differentiated Services).

This document is based on the IETF Policy Core Information Model and its extensions as specified by [PCIM] and [PCIMe]. QPIM builds upon these two documents to define an information model for QoS enforcement for differentiated and integrated services ([DIFFSERV] and [INTSERV], respectively) using policy. It is important to note that this document defines an information model, which by definition is independent of any particular data storage mechanism and access protocol. This enables various data models (e.g., directory schemata, relational database schemata, and SNMP MIBs) to be designed and implemented according to a single uniform model.

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 [KEYWORDS].

## 1.1. The Process of QoS Policy Definition

This section describes the process of using QPIM for the definition QoS policy for a policy domain. Figure 1 illustrates information flow and not the actual procedure, which has several loops and feedback not depicted.

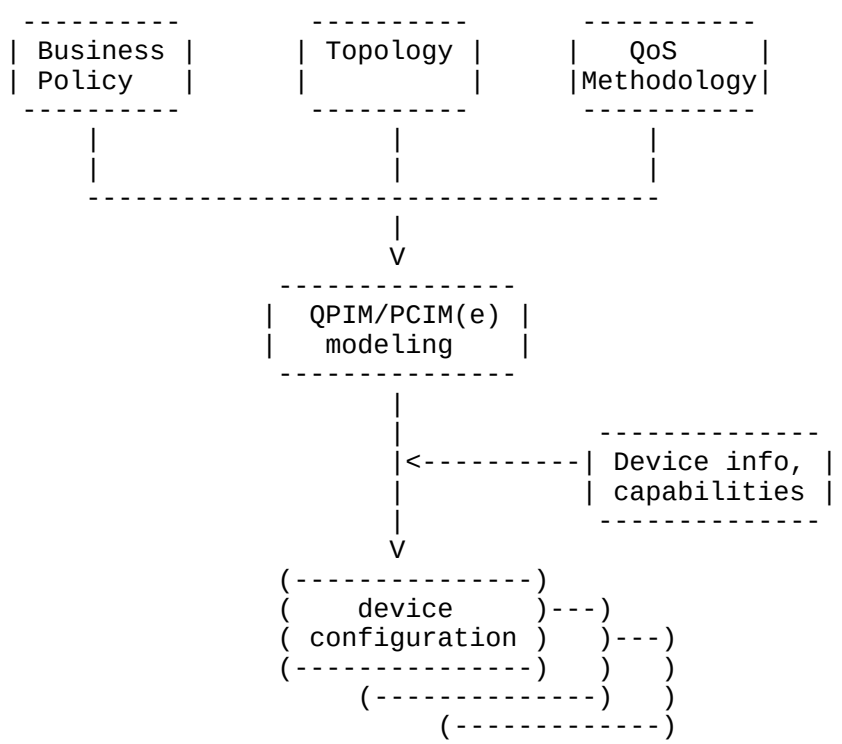



Figure 1: The QoS definition information flow

The process of QoS policy definition is dependent on three types of information: the topology of the network devices under management, the particular type of QoS methodology used (e.g., DiffServ) and the business rules and requirements for specifying service(s) [TERMS] delivered by the network. Both topology and business rules are outside the scope of QPIM. However, important facets of both must be known and understood for correctly specifying the QoS policy.

Typically, the process of QoS policy definition relies on a methodology based on one or more QoS methodologies. For example, the DiffServ methodology may be employed in the QoS policy definition process.

The topology of the network consists of an inventory of the network elements that make up the network and the set of paths that traffic may take through the network. For example, a network administrator may decide to use the DiffServ architectural model [DIFFSERV] and classify network devices using the roles "boundary" and "core" (see [TERMS] for a definition of role, and [PCIM] for an explanation of

how they are used in the policy framework). While this is not a complete topological view of the network, many times it may suffice for the purpose of QoS policy definition.

Business rules are informal sets of requirements for specifying the behavior of various types of traffic that may traverse the network. For example, the administrator may be instructed to implement policy such that VoIP traffic manifests behavior that is similar to legacy voice traffic over telephone networks. Note that this business rule (indirectly) prescribes specific behavior for this traffic type (VoIP), for example in terms of minimal delay, jitter and loss. Other traffic types, such as WEB buying transactions, system backup traffic, video streaming, etc., will express their traffic conditioning requirements in different terms. Again, this information is required not by QPIM itself, but by the overall policy management system that uses QPIM. QPIM is used to help map the business rules into a form that defines the requirements for conditioning different types of traffic in the network.

The topology, QoS methodology, and business rules are necessary prerequisites for defining traffic conditioning. QPIM enables a set of tools for specifying traffic conditioning policy in a standard manner. Using a standard QoS policy information model such as QPIM is needed also because different devices can have markedly different capabilities. Even the same model of equipment can have different functionality if the network operating system and software running in those devices is different. Therefore, a means is required to specify functionality in a standard way that is independent of the capabilities of different vendors' devices. This is the role of QPIM.

In a typical scenario, the administrator would first determine the role(s) that each interface of each network element plays in the overall network topology. These roles define the functions supplied by a given network element independent of vendor and device type. The [PCIM] and [PCIMe] documents define the concept of a role. Roles can be used to identify what parts of the network need which type of traffic conditioning. For example, network interface cards that are categorized as "core" interfaces can be assigned the role name "core-interface". This enables the administrator to design policies to configure all interfaces having the role "core-interface" independent of the actual physical devices themselves. QPIM uses roles to help the administrator map a given set of devices or interfaces to a given set of policy constructs.

The policy constructs define the functionality required to perform the desired traffic conditioning for particular traffic type(s). The functions themselves depend on the particular type of networking technologies chosen. For example, the DiffServ methodology encourages us to aggregate similar types of traffic by assigning to each traffic class a particular per-hop forwarding behavior on each node. RSVP enables bandwidth to be reserved. These two methodologies can be used separately or in conjunction, as defined by the appropriate business policy. QPIM provides specific classes to enable DiffServ and RSVP conditioning to be modeled.

The QPIM class definitions are used to create instances of various policy constructs such as QoS actions and conditions that may be hierarchically organized in rules and groups (PolicyGroup and PolicyRule as defined in [PCIM] and [PCIMe]). Examples of policy actions are rate limiting, jitter control and bandwidth allocation. Policy conditions are constructs that can select traffic according to a complex Boolean expression.

A hierarchical organization was chosen for two reasons. First, it best reflects the way humans tend to think about complex policy. Second, it enables policy to be easily mapped onto administrative organizations, as the hierarchical organization of policy mirrors most administrative organizations. It is important to note that the policy definition process described here is done independent of any specific device capabilities and configuration options. The policy definition is completely independent from the details of the implementation and the configuration interface of individual network elements, as well as of the mechanisms that a network element can use to condition traffic.

## 1.2. Design Goals and Their Ramifications

This section explains the QPIM design goals and how these goals are addressed in this document. This section also describes the ramifications of the design goals and the design decisions made in developing QPIM.

### 1.2.1. Policy-Definition Oriented

The primary design goal of QPIM is to model policies controlling QoS behavior in a way that as closely as possible reflects the way humans tend to think about policy. Therefore, QPIM is designed to address the needs of policy definition and management, and not device/network configuration.

There are several ramifications of this design goal. First, QPIM uses rules to define policies, based on [PCIM] and [PCIMe]. Second, QPIM uses hierarchical organizations of policies and policy information extensively. Third, QPIM does not force the policy writer to specify all implementation details; rather, it assumes that configuration agents (PDPs) interpret the policies and match them to suit the needs of device-specific configurations.

#### 1.2.1.1. Rule-based Modeling

Policy is best described using rule-based modeling as explained and described in [PCIM] and [PCIMe]. A QoS policy rule is structured as a condition clause and an action clause. The semantics are simple: if the condition clause evaluates to TRUE, then a set of QoS actions (specified in the action clause) can be executed. For example, the rule:

> "WEB traffic should receive at least 50% of the available bandwidth resources or more, when more is available"

can be formalized as:

> "<If protocol == HTTP> then <minimum BW = 50%>"

where the first angle bracketed clause is a traffic condition and the second angle bracketed clause is a QoS action.

This approach differs from data path modeling that describes the mechanisms that operates on the packet flows to achieve the desired effect.

Note that the approach taken in QPIM specifically did NOT subclass the PolicyRule class. Rather, it uses the SimplePolicyCondition, CompoundPolicyCondition, SimplePolicyAction, and CompoundPolicyAction classes defined in [PCIMe], as well as defining subclasses of the following classes: Policy, PolicyAction, SimplePolicyAction, PolicyImplicitVariable, and PolicyValue. Subclassing the PolicyRule class would have made it more difficult to combine actions and conditions defined within different functional domains [PCIMe] within the same rules.

#### 1.2.1.2. Organize Information Hierarchically

The organization of the information represented by QPIM is designed to be hierarchical. To do this, QPIM utilizes the PolicySetComponent aggregation [PCIMe] to provide an arbitrarily nested organization of policy information. A policy group functions as a container of

policy rules and/or policy groups. A policy rule can also contain policy rules and/or groups, enabling a rule/sub-rule relationship to be realized.

The hierarchical design decision is based on the realization that it is natural for humans to organize policy rules in groups. Breaking down a complex policy into a set of simple rules is a process that follows the way people tend to think and analyze systems. The complexity of the abstract, business-oriented policy is simplified and made into a hierarchy of simple rules and grouping of simple rules.

The hierarchical information organization helps to simplify the definition and readability of data instances based on QPIM. Hierarchies can also serve to carry additional semantics for QoS actions in a given context. An example, detailed in section 2.3, demonstrates how hierarchical bandwidth allocation policies can be specified in an intuitive form, without the need to specify complex scheduler structures.

#### 1.2.1.3. Goal-Oriented Policy Definition

QPIM facilitates goal-oriented QoS policy definition. This means that the process of defining QoS policy is focused on the desired effect of policies, as opposed to the means of implementing the policy on network elements.

QPIM is intended to define a minimal specification of desired network behavior. It is the role of device-specific configuration agents to interpret policy expressed in a standard way and fill in the necessary configuration details that are required for their particular application. The benefit of using QPIM is that it provides a common lingua franca that each of the device- and/or vendor-specific configuration agents can use. This helps ensure a common interpretation of the general policy as well as aid the administrator in specifying a common policy to be implemented across different devices. This is analogous to the fundamental object-oriented paradigm of separating specification from implementation. Using QPIM, traffic conditioning can be specified in a general manner that can help different implementations satisfy a common goal.

For example, a valid policy may include only a single rule that specifies that bandwidth should be reserved for a given set of traffic flows. The rule does not need to include any of the various other details that may be needed for implementing a scheduler that supports this bandwidth allocation (e.g., the queue length required). It is assumed that a PDP or the PEPs would fill in these details using (for example) their default queue length settings. The policy

writer need only specify the main goal of the policy, making sure that the preferred application receives enough bandwidth to operate adequately.

#### 1.2.2. Policy Domain Model

An important design goal of QPIM is to provide a means for defining policies that span numerous devices. This goal differentiates QPIM from device-level information models, which are designed for modeling policy that controls a single device, its mechanisms and capabilities.

This design goal has several ramifications. First, roles [PCIM] are used to define policies across multiple devices. Second, the use of abstract policies frees the policy definition process from having to deal with individual device peculiarities, and leaves interpretation and configuration to be modeled by PDPs or other configuration agents. Third, QPIM allows extensive reuse of all policy building blocks in multiple rules used within different devices.

##### 1.2.2.1. Model QoS Policy in a Device- and Vendor-Independent Manner

QPIM models QoS policy in a way designed to be independent of any particular device or vendor. This enables networks made up of different devices that have different capabilities to be managed and controlled using a single standard set of policies. Using such a single set of policies is important because otherwise, the policy will itself reflect the differences between different device implementations.

##### 1.2.2.2. Use Roles for Mapping Policy to Network Devices

The use of roles enables a policy definition to be targeted to the network function of a network element, rather than to the element's type and capabilities. The use of roles for mapping policy to network elements provides an efficient and simple method for compact and abstract policy definition. A given abstract policy may be mapped to a group of network elements without the need to specify configuration for each of those elements based on the capabilities of any one individual element.

The policy definition is designed to allow aggregating multiple devices within the same role, if desired. For example, if two core network interfaces operate at different rates, one does not have to define two separate policy rules to express the very same abstract policy (e.g., allocating 30% of the interface bandwidth to a given

preferred set of flows). The use of hierarchical context and relative QoS actions in QPIM addresses this and other related problems.

#### 1.2.2.3. Reusability

Reusable objects, as defined by [PCIM] and [PCIMe], are the means for sharing policy building blocks, thus allowing central management of global concepts. QPIM provides the ability to reuse all policy building blocks: variables and values, conditions and actions, traffic profiles, and policy groups and policy rules. This provides the required flexibility to manage large sets of policy rules over large policy domains.

For example, the following rule makes use of centrally defined objects being reused (referenced):

> If <DestinationAddress == FinanceSubNet> then <DSCP = MissionCritical>

In this rule, the condition refers to an object named FinanceSubNet, which is a value (or possibly a set of values) defined and maintained in a reusable objects container. The QoS action makes use of a value named MissionCritical, which is also a reusable object. The advantage of specifying a policy in this way is its inherent flexibility. Given the above policy, whenever business needs require a change in the subnet definition for the organization, all that's required is to change the reusable value FinanceSubNet centrally. All referencing rules are immediately affected, without the need to modify them individually. Without this capability, the repository that is used to store the rules would have to be searched for all rules that refer to the finance subnet, and then each matching rule's condition would have to be individually updated. This is not only much less efficient, but also is more prone to error.

For a complete description of reusable objects, refer to [PCIM] and [PCIMe].

### 1.2.3. Enforceable Policy

Policy defined by QPIM should be enforceable. This means that a PDP can use QPIM's policy definition in order to make the necessary decisions and enforce the required policy rules. For example, RSVP admission decisions should be made based on the policy definitions specified by QPIM. A PDP should be able to map QPIM policy definitions into PEP configurations, using either standard or proprietary protocols.

QPIM is designed to be agnostic of any particular, vendor-dependent technology. However, QPIM's constructs SHOULD always be interpreted so that policy-compliant behavior can be enforced on the network under management. Therefore, there are three fundamental requirements that QPIM must satisfy:

1. Policy specified by QPIM must be able to be mapped to actual network elements.

2. Policy specified by QPIM must be able to control QoS network functions without making reference to a specific type of device or vendor.

3. Policy specified by QPIM must be able to be translated into network element configuration.

QPIM satisfies requirements #1 and #2 above by using the concept of roles (specifically, the PolicyRoles property, defined in PCIM). By matching roles assigned to policy groups and to network elements, a PDP (or other enforcement agent) can determine what policy should be applied to a given device or devices.

The use of roles in mapping policy to network elements supports model scalability. QPIM policy can be mapped to large-scale policy domains by assigning a single role to a group of network elements. This can be done even when the policy domain contains heterogeneous devices. So, a small set of policies can be deployed to large networks without having to re-specify the policy for each device separately. This QPIM property is important for QoS policy management applications that strive to ease the task of policy definition for large policy domains.

Requirement #2 is also satisfied by making QPIM domain-oriented (see [TERMS] for a definition of "domain"). In other words, the target of the policy is a domain, as opposed to a specific device or interface.

Requirement #3 is satisfied by modeling QoS conditions and actions that are commonly configured on various devices. However, QPIM is extensible to allow modeling of actions that are not included in QPIM.

It is important to note that different PEPs will have different capabilities and functions, which necessitate different individual configurations even if the different PEPs are controlled by the same policy.

### 1.2.4. QPIM Covers Both Signaled And Provisioned QoS

The two predominant standards-based QoS methodologies developed so far are Differentiated Services (DiffServ) and Integrated Services (IntServ). The DiffServ provides a way to enforce policies that apply to a large number of devices in a scalable manner. QPIM provides actions and conditions that control the classification, policing and shaping done within the differentiated service domain boundaries, as well as actions that control the per-hop behavior within the core of the DiffServ network. QPIM does not mandate the use of DiffServ as a policy methodology.

Integrated services, together with its signaling protocol (RSVP), provides a way for end nodes (and edge nodes) to request QoS from the network. QPIM provides actions that control the reservation of such requests within the network.

As both methodologies continue to evolve, QPIM does not attempt to provide full coverage of all possible scenarios. Instead, QPIM aims to provide policy control modeling for all major scenarios. QPIM is designed to be extensible to allow for incorporation of control over newly developed QoS mechanisms.

### 1.2.5. Interoperability for PDPs and Management Applications

Another design goal of QPIM is to facilitate interoperability among policy systems such as PDPs and policy management applications. QPIM accomplishes this interoperability goal by standardizing the representation of policy. Producers and consumers of QoS policy need only rely on QPIM-based schemata (and resulting data models) to ensure mutual understanding and agreement on the semantics of QoS policy.

For example, suppose that a QoS policy management application, built by vendor A writes its policies based on the LDAP schema that maps from QPIM to a directory implementation using LDAP. Now assume that a separately built PDP from vendor B also relies on this same LDAP schema derived from QPIM. Even though these are two vendors with two different PDPs, each may read the schema of the other and "understand" it. This is because both the management application and the PDP were architected to comply with the QPIM specification. The same is true with two policy management applications. For example, vendor B’s policy application may run a validation tool that computes whether there are conflicts within rules specified by the other vendor’s policy management application.

Interoperability of QPIM producers/consumers is by definition at a high level, and does not guarantee that the same policy will result in the same PEP configuration. First, different PEPs will have different capabilities and functions, which necessitate different individual configurations even if the different PEPs are controlled by the same policy. Second, different PDPs will also have different capabilities and functions, and may choose to translate the high-level QPIM policy differently depending on the functionality of the PDP, as well as on the capabilities of the PEPs that are being controlled by the PDP. However, the different configurations should still result in the same network behavior as that specified by the policy rules.

## 1.3. Modeling Abstract QoS Policies

This section provides a discussion of QoS policy abstraction and the way QPIM addresses this issue.

As described above, the main goal of the QPIM is to create an information model that can be used to help bridge part of the conceptual gap between a human policy maker and a network element that is configured to enforce the policy. Clearly this wide gap implies several translation levels, from the abstract to the concrete. At the abstract end are the business QoS policy rules. Once the business rules are known, a network administrator must interpret them as network QoS policy and represent this QoS policy by using QPIM constructs. QPIM facilitates a formal representation of QoS rules, thus providing the first concretization level: formally representing humanly expressed QoS policy.

When a human business executive defines network policy, it is usually done using informal business terms and language. For example, a human may utter a policy statement that reads:

> "human resources applications should have better QoS than simple web applications"

This might be translated to a slightly more sophisticated form, such as:

> "traffic generated by our human resources applications should have a higher probability of communicating with its destinations than traffic generated by people browsing the WEB using non-mission-critical applications"

While this statement clearly defines QoS policy at the business level, it isn’t specific enough to be enforceable by network elements. Translation to "network terms and language" is required.

On the other end of the scale, a network element functioning as a PEP, such as a router, can be configured with specific commands that determine the operational parameters of its inner working QoS mechanisms. For example, the (imaginary) command "output-queue-depth = 100" may be an instruction to a network interface card of a router to allow up to 100 packets to be stored before subsequent packets are discarded (not forwarded). On a different device within the same network, the same instruction may take another form, because a different vendor built that device or it has a different set of functions, and hence implementation, even though it is from the same vendor. In addition, a particular PEP may not have the ability to create queues that are longer than, say, 50 packets, which may result in a different instruction implementing the same QoS policy.

The first example illustrates 'abstract policy', while the second illustrates 'concrete configuration'. Furthermore, the first example illustrates end-to-end policy, which covers the conditioning of application traffic throughout the network. The second example illustrates configuration for a particular PEP or a set thereof. While an end-to-end policy statement can only be enforced by configuration of PEPs in various parts of the network, the information model of policy and that of the mechanisms that a PEP uses to implement that policy are vastly different.

The translation process from abstract business policy to concrete PEP configuration is roughly expressed as follows:

1. Informal business QoS policy is expressed by a human policy maker (e.g., "All executives' WEB requests should be prioritized ahead of other employees' WEB requests")

2. A network administrator analyzes the policy domain's topology and determines the roles of particular device interfaces. A role may be assigned to a large group of elements, which will result in mapping a particular policy to a large group of device interfaces.

3. The network administrator models the informal policy using QPIM constructs, thus creating a formal representation of the abstract policy. For example, "If a packet's protocol is HTTP and its destination is in the 'EXECUTIVES' user group, then assign IPP 7 to the packet header".

4. The network administrator assigns roles to the policy groups created in the previous step matching the network elements' roles assigned in step #2 above.

5. A PDP translates the abstract policy constructs created in step #3 into device-specific configuration commands for all devices effected by the new policy (i.e., devices that have interfaces that are assigned a role matching the new policy constructs' roles). In this process, the PDP consults the particular devices' capabilities to determine the appropriate configuration commands implementing the policy.

6. For each PEP in the network, the PDP (or an agent of the PDP) issues the appropriate device-specific instructions necessary to enforce the policy.

QPIM, PCIM and PCIMe are used in step #3 above.

## 1.4. Rule Hierarchy

Policy is described by a set of policy rules that may be grouped into subsets [PCIMe]. Policy rules and policy groups can be nested within other policy rules, providing a hierarchical policy definition. Nested rules are also called sub-rules, and we use both terms in this document interchangeably. The aggregation PolicySetComponent (defined in [PCIMe] is used to represent the nesting of a policy rule or group in another policy rule.

The hierarchical policy rule definition enhances policy readability and reusability. Within the QoS policy information model, hierarchy is used to model context or scope for the sub-rule actions. Within QPIM, bandwidth allocation policy actions and drop threshold actions use this hierarchal context. First we provide a detailed example of the use of hierarchy in bandwidth allocation policies. The differences between flat and hierarchical policy representation are discussed. The use of hierarchy in drop threshold policies is described in a following subsection. Last but not least, the restrictions on the use of rule hierarchies within QPIM are described.

### 1.4.1. Use of Hierarchy Within Bandwidth Allocation Policies

Consider the following example where the informal policy reads:

> On any interface on which these rules apply, guarantee at least 30% of the interface bandwidth to UDP flows, and at least 40% of the interface bandwidth to TCP flows.

The QoS Policy information model follows the Policy Core information model by using roles as a way to specify the set of interfaces on which this policy applies. The policy does not assume that all interfaces are run at the same speed, or have any other property in

common apart from being able to forward packets. Bandwidth is allocated between UDP and TCP flows using percentages of the available interface bandwidth. Assume that we have an available interface bandwidth of 1 Mbits/sec. Then this rule will guarantee 300Kbits/sec to UDP flows. However, if the interface bandwidth was instead only 64kbits/sec, then this rule would correspondingly guarantee 19.2kb/sec.

This policy is modeled within QPIM using two policy rules of the form:

```
If (IP protocol is UDP) THEN (guarantee 30% of available BW) (1)
If (IP protocol is TCP) THEN (guarantee 40% of available BW) (2)
```

Assume that these two rules are grouped within a PolicySet [PCIMe] carrying the appropriate role combination. A possible implementation of these rules within a PEP would be to use a Weighted-Round-Robin scheduler with 3 queues. The first queue would be used for UDP traffic, the second queue for TCP traffic and the third queue for the rest of the traffic. The weights of the Weighted-Round-Robin scheduler would be 30% for the first queue, 40% for the second queue and 30% for the last queue.

The actions specifying the bandwidth guarantee implicitly assume that the bandwidth resource being guaranteed is the bandwidth available at the interface level. A PolicyRoleCollection is a class defined in [PCIMe] whose purpose is to identify the set of resources (in this example, interfaces) that are assigned to a particular role. Thus, the type of managed elements aggregated within the PolicyRoleCollection defines the bandwidth resource being controlled. In our example, interfaces are aggregated within the PolicyRoleCollection. Therefore, the rules specify bandwidth allocation to all interfaces that match a given role. Other behavior could be similarly defined by changing what was aggregated within the PolicyRoleCollection.

Normally, a full specification of the rules would require indicating the direction of the traffic for which bandwidth allocation is being made. Using the direction variable defined in [PCIMe], the rules can be specified in the following form:

```
If (direction is out)
    If (IP protocol is UDP) THEN (guarantee 30% of available BW)
    If (IP protocol is TCP) THEN (guarantee 40% of available BW)
```

where indentation is used to indicate rule nesting. To save space, we omit the direction condition from further discussion.

Rule nesting provides the ability to further refine the scope of bandwidth allocation within a given traffic class forwarded via these interfaces. The example below adds two nested rules to refine bandwidth allocation for UDP and TCP applications.

```
If (IP protocol is UDP) THEN (guarantee 30% of available BW) (1)
   If (protocol is TFTP) THEN (guarantee 10% of available BW) (1a)
   If (protocol is NFS) THEN (guarantee 40% of available BW) (1b)
If (IP protocol is TCP) THEN (guarantee 40% of available BW) (2)
   If (protocol is HTTP) THEN guarantee 20% of available BW) (2a)
   If (protocol is FTP) THEN (guarantee 30% of available BW) (2b)
```

Subrules 1a and 1b specify bandwidth allocation for UDP applications. The total bandwidth resource being partitioned among UDP applications is the bandwidth available for the UDP traffic class (i.e., 30%), not the total bandwidth available at the interface level. Furthermore, TFTP and NFS are guaranteed to get at least 10% and 40% of the total available bandwidth for UDP, while other UDP applications aren't guaranteed to receive anything. Thus, TFTP and NFS are guaranteed to get at least 3% and 12% of the total bandwidth. Similar logic applies to the TCP applications.

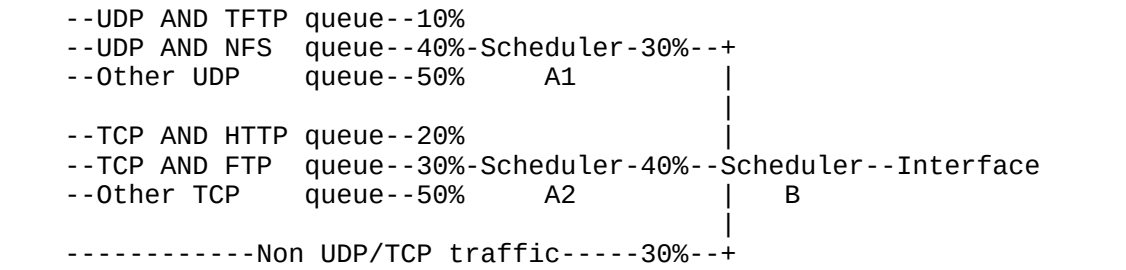
The point of this section will be to show that a hierarchical policy representation enables a finer level of granularity for bandwidth allocation to be specified than is otherwise available using a non-hierarchical policy representation. To see this, let's compare this set of rules with a non-hierarchical (flat) rule representation. In the non-hierarchical representation, the guaranteed bandwidth for TFTP flows is calculated by taking 10% of the bandwidth guaranteed to UDP flows, resulting in 3% of the total interface bandwidth guarantee.

```
If (UDP AND TFTP) THEN (guarantee 3% of available BW) (1a)
If (UDP AND NFS) THEN (guarantee 12% of available BW) (1b)
If (other UDP APPs) THEN (guarantee 15% of available BW) (1c)
If (TCP AND HTTP) THEN guarantee 8% of available BW) (2a)
If (TCP AND FTP) THEN (guarantee 12% of available BW) (2b)
If (other TCP APPs) THEN (guarantee 20% of available BW) (2c)
```

Are these two representations identical? No, bandwidth allocation is not the same. For example, within the hierarchical representation, UDP applications are guaranteed 30% of the bandwidth. Suppose a single UDP flow of an application different from NFS or TFTP is running. This application would be guaranteed 30% of the interface bandwidth in the hierarchical representation but only 15% of the interface bandwidth in the flat representation.

A two stage scheduler is best modeled by a hierarchical representation whereas a flat representation may be realized by a non-hierarchical scheduler.

A schematic hierarchical Weighted-Round-Robin scheduler implementation that supports the hierarchical rule representation is described below.




Scheduler A1 extracts packets from the 3 UDP queues according to the weight specified by the UDP sub-rule policy. Scheduler A2 extracts packets from the 3 TCP queues specified by the TCP sub-rule policy. The second stage scheduler B schedules between UDP, TCP and all other traffic according to the policy specified in the top most rule level.

Another difference between the flat and hierarchical rule representation is the actual division of bandwidth above the minimal bandwidth guarantee. Suppose two high rate streams are being forwarded via this interface: an HTTP stream and an NFS stream. Suppose that the rate of each flow is far beyond the capacity of the interface. In the flat scheduler implementation, the ratio between the weights is 8:12 (i.e., HTTP:NFS), and therefore HTTP stream would consume 40% of the bandwidth while NFS would consume 60% of the bandwidth. In the hierarchical scheduler implementation the only scheduler that has two queues filled is scheduler B, therefore the ratio between the HTTP (TCP) stream and the NFS (UDP) stream would be 30:40, and therefore the HTTP stream would consume approximately 42% of the interface bandwidth while NFS would consume 58% of the interface bandwidth. In both cases both HTTP and NFS streams got more than the minimal guaranteed bandwidth, but the actual rates forwarded via the interface differ.

The conclusion is that hierarchical policy representation provides additional structure and context beyond the flat policy representation. Furthermore, policies specifying bandwidth allocation using rule hierarchies should be enforced using hierarchical schedulers where the rule hierarchy level is mapped to the hierarchical scheduler level.

### 1.4.2. Use of Rule Hierarchy to Describe Drop Threshold Policies

Two major resources govern the per hop behavior in each node. The bandwidth allocation resource governs the forwarding behavior of each traffic class. A scheduler priority and weights are controlled by the bandwidth allocation policies, as well as the (minimal) number of queues needed for traffic separation. A second resource, which is not controlled by bandwidth allocation policies, is the queuing length and drop behavior. For this purpose, queue length and threshold policies are used.

Rule hierarchy is used to describe the context on which thresholds act. The policy rule's condition describes the traffic class and the rule's actions describe the bandwidth allocation, the forwarding priority and the queue length. If the traffic class contains different drop precedence sub-classes that require different thresholds within the same queue, the sub-rules actions describe these thresholds.

Below is an example of the use of rule nesting for threshold control purposes. Let's look at the following rules:

```
   If (protocol is FTP) THEN (guarantee 10% of available BW)
                             (queue length equals 40 packets)
                             (drop technique is random)

      if (src-ip is from net 2.x.x.x) THEN min threshold = 30%
                                           max threshold = 70%

      if (src-ip is from net 3.x.x.x) THEN min threshold = 40%
                                           max threshold = 90%

      if (all other)                  THEN min threshold = 20%
                                                 max threshold = 60%
```

The rule describes the bandwidth allocation, the queue length and the drop technique assigned to FTP flows. The sub-rules describe the drop threshold priorities within those FTP flows. FTP packets received from all networks apart from networks 2.x.x.x and 3.x.x.x are randomly dropped when the queue threshold for FTP flows accumulates to 20% of the queue length. Once the queue fills to 60%, all these packets are dropped before queuing. The two other sub rules provide other thresholds for FTP packets coming from the specified two subnets. The Assured Forwarding per hop behavior (AF) is another good example of the use of hierarchy to describe the different drop preferences within a traffic class. This example is provided in a later section.

### 1.4.3. Restrictions of the Use of Hierarchy Within QPIM

Rule nesting is used within QPIM for two important purposes:

1) Enhance clarity, readability and reusability.

2) Provide hierarchical context for actions.

The second point captures the ability to specify context for bandwidth allocation, as well as providing context for drop threshold policies.

When is a hierarchy level supposed to specify the bandwidth allocation context, when is the hierarchy used for specifying the drop threshold context, and when is it used merely for clarity and reusability? The answer depends entirely on the actions. Bandwidth control actions within a sub-rule specify how the bandwidth allocated to the traffic class determined by the rule's condition clause should be further divided among the sub-rules. Drop threshold actions control the traffic class's queue drop behavior for each of the sub-rules. The bandwidth control actions have an implicit pointer saying: the bandwidth allocation is relative to the bandwidth resources defined by the higher level rule. Drop threshold actions have an implicit pointer saying: the thresholds are taken from the queue resources defined by the higher level rule. Other actions do not have such an implicit pointer, and for these actions hierarchy is used only for reusability and readability purposes.

Each rule that includes a bandwidth allocation action implies that a queue should be allocated to the traffic class defined by the rule's condition clause. Therefore, once a bandwidth allocation action exists within the actions of a sub-rule, a threshold action within this sub-rule cannot refer to thresholds of the parent rule's queue. Instead, it must refer to the queue of the sub-rule itself. Therefore, in order to have a clear and unambiguous definition, refinement of thresholds and refinements of bandwidth allocations within sub-rules should be avoided. If both refinements are needed for the same rule, threshold refinements and bandwidth refinements rules should each be aggregated to a separate group, and these groups should be aggregated under the policy rule, using the PolicySetComponent aggregation.

### 1.5. Intended Audiences

QPIM is intended for several audiences. The following lists some of the intended audiences and their respective uses:

1. Developers of QoS policy management applications can use this model as an extensible framework for defining policies to control PEPs and PDPs in an interoperable manner.

2. Developers of Policy Decision Point (PDP) systems built to control resource allocation signaled by RSVP requests.

3. Developers of Policy Decision Points (PDP) systems built to create QoS configuration for PEPs.

4. Builders of large organization data and knowledge bases who decide to combine QoS policy information with other networking policy information, assuming all modeling is based on [PCIM] and [PCIMe].

5. Authors of various standards may use constructs introduced in this document to enhance their work. Authors of data models wishing to map a storage specific technology to QPIM must use this document as well.

## 2. Class Hierarchies

### 2.1. Inheritance Hierarchy

QPIM's class and association inheritance hierarchies are rooted in [PCIM] and [PCIMe]. Figures 2 and 3 depict these QPIM inheritance hierarchies, while noting their relationships to [PCIM] and [PCIMe]classes. Note that many other classes used to form QPIM policies, such as SimplePolicyCondition, are defined in [PCIM] and [PCIMe]. Thus, the following figures do NOT represent ALL necessary classes and relationships for defining QPIM policies. Rather, the designer using QPIM should use appropriate classes and relationships from [PCIM] and [PCIMe] in conjunction with those defined below.


(continued on the next page)

(continued from the previous page)

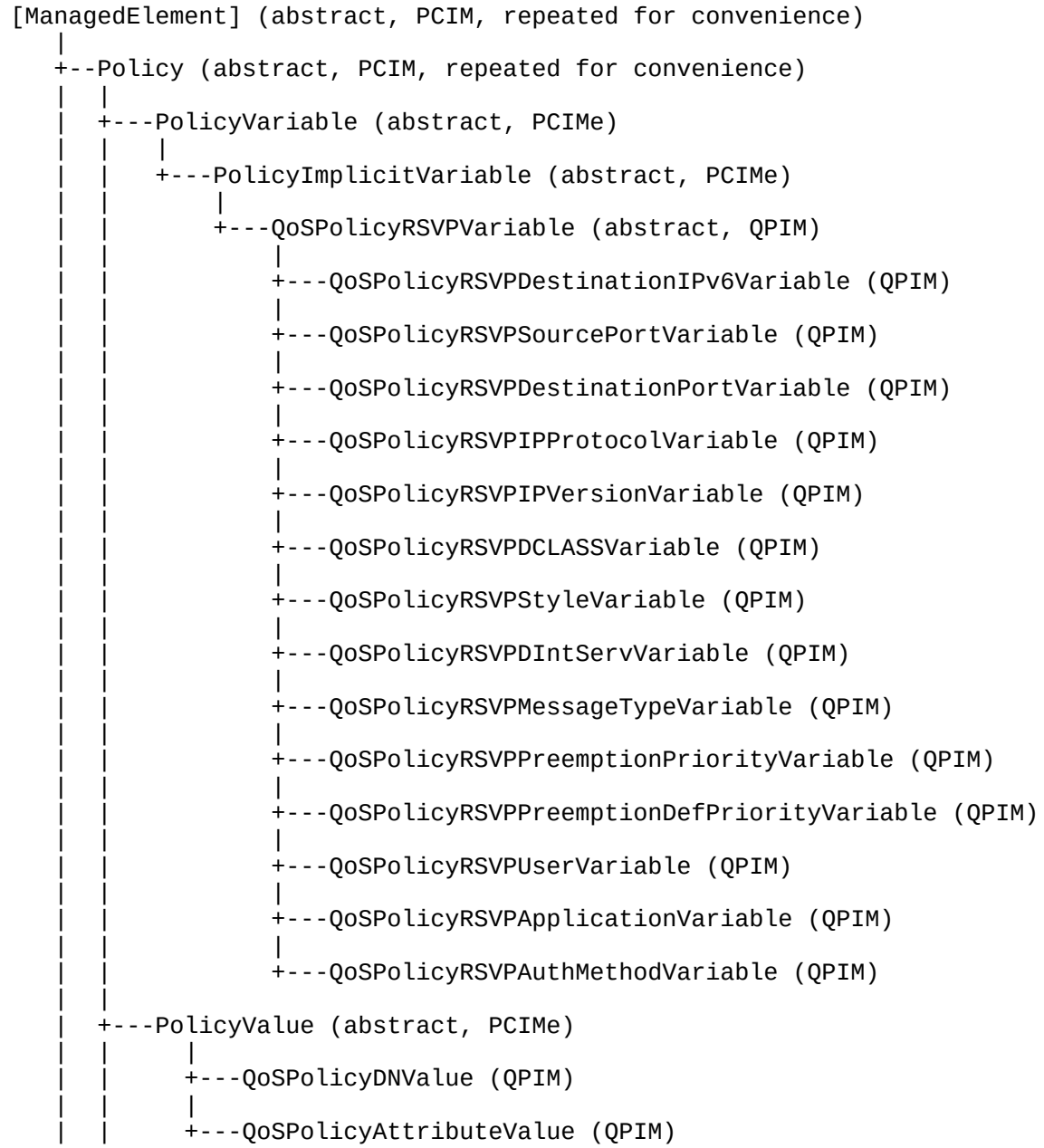



Figure 2. The QPIM Class Inheritance Hierarchy

## 2.2. Relationship Hierarchy

Figure 3 shows the QPIM relationship hierarchy.

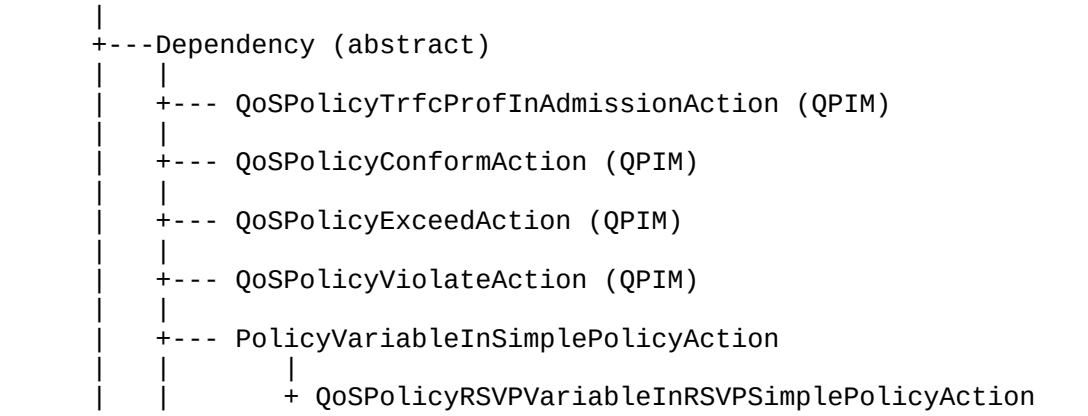



Figure 3. The QPIM Association Class Inheritance Hierarchy

# 3. QoS Actions

This section describes the QoS actions that are modeled by QPIM. QoS actions are policy enforced network behaviors that are specified for traffic selected by QoS conditions. QoS actions are modeled using the classes PolicyAction (defined in [PCIM]), SimplePolicyAction (defined in [PCIMe]) and several QoS actions defined in this document that are derived from both of these classes, which are described below.

Note that there is no discussion of PolicyRule, PolicyGroup, or different types of PolicyCondition classes in this document. This is because these classes are fully specified in [PCIM] and [PCIMe].

## 3.1. Overview

QoS policy based systems allow the network administrator to specify a set of rules that control both the selection of the flows that need to be provided with a preferred forwarding treatment, as well as specifying the specific set of preferred forwarding behaviors. QPIM provides an information model for specifying such a set of rules.

QoS policy rules enable controlling environments in which RSVP signaling is used to request different forwarding treatment for different traffic types from the network, as well as environments where no signaling is used, but preferred treatment is desired for

some (but not all) traffic types. QoS policy rules also allow controlling environments where strict QoS guarantees are provided to individual flows, as well as environments where QoS is provided to flow aggregates. QoS actions allow a PDP or a PEP to determine which RSVP requests should be admitted before network resources are allocated. QoS actions allow control of the RSVP signaling content itself, as well as differentiation between priorities of RSVP requests. QoS actions allow controlling the Differentiated Service edge enforcement including policing, shaping and marking, as well as the per-hop behaviors used in the network core. Finally, QoS actions can be used to control mapping of RSVP requests at the edge of a differentiated service cloud into per hop behaviors.

Four groups of actions are derived from action classes defined in [PCIM] and [PCIMe]. The first QoS action group contains a single action, QoSPolicyRSVPSimpleAction. This action is used for both RSVP signal control and install actions. The second QoS action group determines whether a flow or class of flows should be admitted. This is done by specifying an appropriate traffic profile using the QoSPolicyTrfcProf class and its subclasses. This set of actions also includes QoS admission control actions, which use the QoSPolicyAdmissionAction class and its subclasses. The third group of actions control bandwidth allocation and congestion control differentiations, which together specify the per-hop behavior forwarding treatment. This group of actions includes the QoSPolicyPHBAction class and its subclasses. The fourth QoS action is an unconditional packet discard action, which uses the QoSPolicyDiscardAction class. This action is used either by itself or as a building block of the QoSPolicyPoliceAction.

Note that some QoS actions are not directly modeled. Instead, they are modeled by using the class SimplePolicyAction with the appropriate associations. For example, the three marking actions (DSCP, IPP and CoS) are modeled by using the SimplePolicyAction class, and associating that class with variables and values of the appropriate type defined in [PCIMe].

## 3.2. RSVP Policy Actions

There are three types of decisions a PDP (either remote or within a PEP) can make when it evaluates an RSVP request:

1. Admit or reject the request
2. Add or modify the request admission parameters
3. Modify the RSVP signaling content

The COPS for RSVP [RFC2749] specification uses different Decision object types to model each of these decisions. QPIM follows the COPS for RSVP specification and models each decision using a different action class.

The QoSPolicyRSVPAdmissionAction controls the Decision Command and Decision Flags objects used within COPS for RSVP. The QoSPolicyRSVPAdmissionAction class, with its associated QoSPolicyIntServTrfcProf class, is used to determine whether to accept or reject a given RSVP request by comparing the RSVP request's TSPEC or RSPEC parameters against the traffic profile specified by the QoSPolicyIntServTrfcProf. For a full description of the comparison method, see section 4. Following the COPS for RSVP specification, the admission decision has an option to both accept the request and send a warning to the requester. The QoSPolicyRSVPAdmissionAction can be used to limit the number of admitted reservations as well.

The class QoSPolicyRSVPSimpleAction, which is derived from the PolicySimpleAction class [PCIMe], can be used to control the two other COPS RSVP decision types. The property qpRSVPActionType designates the instance of the class to be either of type 'REPLACE', 'STATELESS', or both ('REPLACEANDSTATELESS'). For instances carrying a qpRSVPActionType property value of 'REPLACE', the action is interpreted as a COPS Replace Decision, controlling the contents of the RSVP message. For instances carrying a qpRSVPActionType property value of 'STATELESS', the action is interpreted as a COPS Stateless Decision, controlling the admission parameters. If both of these actions are required, this can be done by assigning the value REPLACEANDSTATELESS to the qpRSVPActionType property.

This class is modeled to represent the COPS for RSVP Replace and Stateless decisions. This similarity allows future use of these COPS decisions to be directly controlled by a QoSPolicySimpleAction. The only required extension might be the definition of a new RSVP variable.

### 3.2.1. Example: Controlling COPS Stateless Decision

The QoSPolicyRSVPSimpleAction allows the specification of admission parameters. It allows specification of the preemption priority [RFC3181] of a given RSVP Reservation request. Using the preemption priority value, the PEP can determine the importance of a Reservation compared with already admitted reservations, and if necessary can preempt lower priority reservations to make room for the higher priority one. This class can also be used to control mapping of RSVP requests to a differentiated services domain by setting the

QoSPolicyRSVPDCLASSVariable to the required value. This instructs the PEP to mark traffic matching the Session and Sender specifications carried in an RSVP request to a given DSCP value.

#### 3.2.2. Example: Controlling the COPS Replace Decision

A Policy system should be able to control the information carried in the RSVP messages. The QoSPolicyRSVPSimpleAction allows control of the content of RSVP signaling messages. An RSVP message can carry a preemption policy object [RFC3181] specifying the priority of the reservation request in comparison to other requests. An RSVP message can also carry a policy object for authentication purposes. An RSVP message can carry a DCLASS [DCLASS] object that specifies to the receiver or sender the particular DSCP value that should be set on the data traffic. A COPS for RSVP Replacement Data Decision controls the content of the RSVP message by specifying a set of RSVP objects replacing or removing the existing ones.

### 3.3. Provisioning Policy Actions

The differentiated Service Architecture [DIFFSERV] was designed to provide a scalable QoS differentiation without requiring any signaling protocols running between the hosts and the network. The QoS actions modeled in QPIM can be used to control all of the building blocks of the Differentiated Service architecture, including per-hop behaviors, edge classification, and policing and shaping, without a need to specify the datapath mechanisms used by PEP implementations. This provides an abstraction level hiding the unnecessary details and allowing the network administrator to write rules that express the network requirements in a more natural form. In this architecture, as no signaling between the end host and the network occurs before the sender starts sending information, the QoS mechanisms should be set up in advance. This usually means that PEPs need to be provisioned with the set of policy rules in advance.

Policing and Shaping actions are modeled as subclasses of the QoS admission action. DSCP and CoS marking are modeled by using the SimplePolicyAction ([PCIMe]) class associated with the appropriate variables and values. Bandwidth allocation and congestion control actions are modeled as subclasses of the QpQPolicyPHBAction, which is itself a subclass PolicyAction class ([PCIM])

#### 3.3.1. Admission Actions: Controlling Policers and Shapers

Admission Actions (QoSPolicyAdmissionAction and its subclasses) are used to police and/or shape traffic.

Each Admission Action is bound to a traffic profile (QoSPolicyTrfcProf) via the QoSPolicyTrfcProfInAdmissionAction association. The traffic profile is used to meter traffic for purposes of policing or shaping.

An Admission Action carries a scope property (qpAdmissionScope) that is used to determine whether the action controls individual traffic flows or aggregate traffic classes. The concepts of "flow" and "traffic class" are explained in [DIFFSERV] using the terms 'microflow' and 'traffic stream'. Roughly speaking, a flow is a set of packets carrying an IP header that has the same values for source IP, destination IP, protocol and layer 4 source and destination ports. A traffic class is a set of flows. In QPIM, simple and compound conditions can identify flows and/or traffic classes by using Boolean terms over the values of IP header fields, including the value of the ToS byte.

Thus, the interpretation of the scope property is as follows: If the value of the scope property is 0 (per-flow), each (micro) flow that can be positively matched with the rule's condition is metered and policed individually. If the value of the scope property is 1 (per-class), all flows matched with the rule's condition are metered as a single aggregate and policed together.

The following example illustrates the use of the scope property. Using two provisioned policing actions, the following policies can be enforced:

- Make sure that each HTTP flow will not exceed 64kb/s

- Make sure that the aggregate rate of all HTTP flows will not exceed 512Kb/s

Both policies are modeled using the same class QoSPolicyPoliceAction (derived from QoSPolicyAdmissionAction). The first policy has its scope property set to 'flow', while the second policy has its scope property set to 'class'. The two policies are modeled using a rule with two police actions that, in a pseudo-formal definition, looks like the following:

```
If (HTTP) Action1=police, Traffic Profile1=64kb/s, Scope1=flow
          Action2=police, Traffic Profile2=512kb/s, Scope2=class
```

The provisioned policing action QoSPolicyPoliceAction has three associations, QoSPolicyConformAction, QoSPolicyExceedAction and QoSPolicyViolateAction.

To accomplish the desired result stated above, two possible modeling techniques may be used: The two actions can be part of a single policy rule using two PolicyActionInPolicyRule [PCIM] associations. In this case the ExecutionStrategy property of the PolicyRule class [PCIMe] SHOULD be set to "Do All" so that both individual flows and aggregate streams are policed.

Alternatively, Action1 and Action2 could be aggregated in a CompundPolicyAction instance using the PolicyActionInPolicyAction aggregations [PCIMe]. In this case, in order for both individual flows and aggregate traffic classes to be policed, the ExecutionStrategy property of the CompoundPolicyAction class [PCIMe] SHOULD be set to "Do All".

The policing action is associated with a three-level token bucket traffic profile carrying rate, burst and excess-burst parameters. Traffic measured by a meter can be classified as conforming traffic when the metered rate is below the rate defined by the traffic profile, as excess traffic when the metered traffic is above the normal burst and below the excess burst size, and violating traffic when rate is above the maximum excess burst.

The [DIFF-MIB] defines a two-level meter, and provides a means to combine two-level meters into more complex meters. In this document, a three-level traffic profile is defined. This allows construction of both two-level meters as well as providing an easier definition for three-level meters needed for creating AF [AF] provisioning actions.

A policing action that models three-level policing MUST associate three separate actions with a three-level traffic profile. These actions are a conforming action, an exceeding action and a violating action. A policing action that models two-level policing uses a two-level traffic profile and associates only conforming and exceeding actions. A policing action with a three-level traffic profile that specifies an exceed action but does not specify a violate action implies that the action taken when the traffic is above the maximum excess burst is identical to the action taken when the traffic is above the normal burst. A policer determines whether the profile is being met, while the actions to be performed are determined by the associations QoSPolicyXXXAction.

Shapers are used to delay some or all of the packets in a traffic stream, in order to bring the stream into compliance with a traffic profile. A shaper usually has a finite-sized buffer, and packets may be discarded if there is not sufficient buffer space to hold the delayed packets. Shaping is controlled by the QoSPolicyShapeAction

class. The only required association is a traffic profile that specifies the rate and burst parameters that the outgoing flows should conform with.

### 3.3.2. Controlling Markers

Three types of marking control actions are modeled in QPIM: Differentiated Services Code Point (DSCP) assignment, IP Precedence (IPP) assignment and layer-2 Class of Service (CoS) assignment. These assignment actions themselves are modeled by using the SimplePolicyAction class associated with the appropriate variables and values.

DSCP assignment sets ("marks" or "colors") the DS field of a packet header to a particular DS Code Point (DSCP), adding the marked packet to a particular DS behavior aggregate.

When used in the basic form, "If <condition> then 'DCSP = ds1'", the assignment action assigns a DSCP value (ds1) to all packets that result in the condition being evaluated to true.

When used in combination with a policing action, a different assignment action can be issued via each of the 'conform', 'exceed' and 'violate' action associations. This way, one may select a PHB in a PHB group according to the state of a meter.

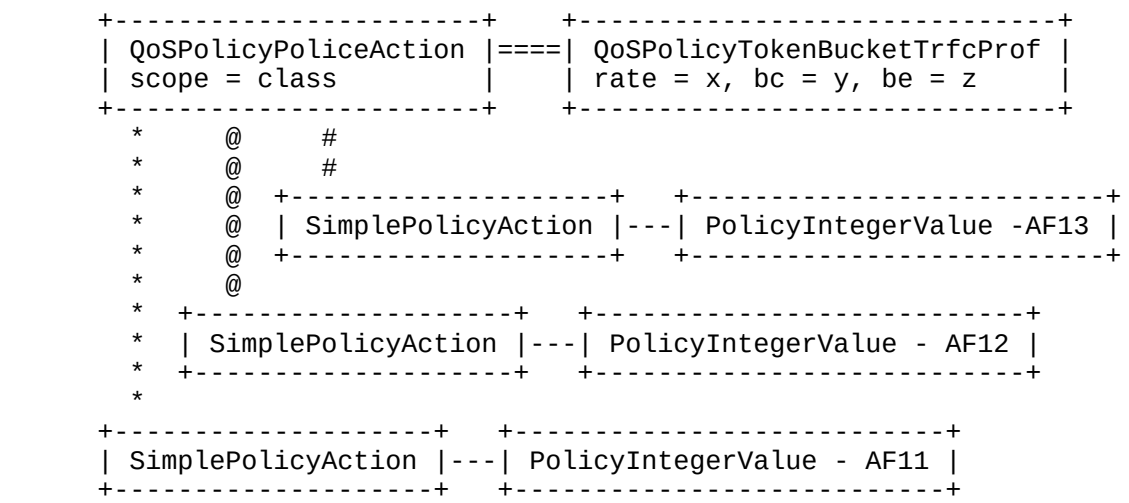
The semantics of the DSCP assignment is encapsulated in the pairing of a DSCP variable and a DSCP value within a single SimplePolicyAction instance via the appropriate associations.

IPP assignment sets the IPP field of a packet header to a particular IPP value (0 through 7). The semantics of the IPP assignment is encapsulated in the pairing of a ToS variable (PolicyIPTosVariable) and a bit string value () (defined in [PCIMe]) within a single SimplePolicyAction instance via the appropriate associations. The bit string value is used in its masked bit string format. The mask indicates the relevant 3 bits of the IPP sub field within the ToS byte, while the bit string indicates the IPP value to be set.

CoS assignments control the mapping of a per-hop behavior to a layer-2 Class of Service. For example, mapping of a set of DSCP values into a 802.1p user priority value can be specified using a rule with a condition describing the set of DSCP values, and a CoS assignment action that specifies the required mapping to the given user priority value. The semantics of the CoS assignment is encapsulated in the pairing of a CoS variable and a CoS value (integer in the range of 0 through 7) within a single SimplePolicyAction instance via the appropriate associations.

### 3.3.3. Controlling Edge Policies - Examples

Assuming that the AF1 behavior aggregate is enforced within a DS domain, policy rules on the boundaries of the network should mark packets to one of the AF1x DSCPs, depending on the conformance of the traffic to a predetermined three-parameter traffic profile. QPIM models such AF1 policing action as defined in Figure 4.




Association and Aggregation Legend:

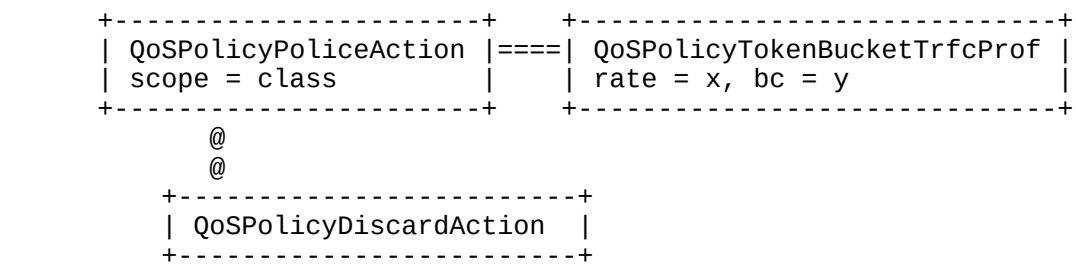
```
****  QoSPolicyConformAction
@@@@  QoSPolicyExceedAction
####  QoSPolicyViolateAction
====  QoSTrfcProfInAdmissionAction
----  PolicyValueInSimplePolicyAction ([PCIMe])
&&&&  PolicyVariableInSimplePolicyAction ([PCIMe], not shown)
```

Figure 4. AF Policing and Marking

The AF policing action is composed of a police action, a token bucket traffic profile and three instances of the SimplePolicyAction class. Each of the simple policy action instances models a different marking action. Each SimplePolicyAction uses the aggregation PolicyVariableInSimplePolicyAction to specify that the associated PolicyDSCPVariable is set to the appropriate integer value. This is done using the PolicyValueInSimplePolicyAction aggregation. The three PolicyVariableInSimplePolicyAction aggregations which connect the appropriate SimplePolicyActions with the appropriate DSCP

Variables, are not shown in this figure for simplicity. AF11 is marked on detecting conforming traffic; AF12 is marked on detecting exceeding traffic, and AF13 on detecting violating traffic.

The second example, shown in Figure 5, is the simplest policing action. Traffic below a two-parameter traffic profile is unmodified, while traffic exceeding the traffic profile is discarded.




Association and Aggregation Legend:
**** QoSPolicyConformAction (not used)
@@@@ QoSPolicyExceedAction
#### QoSPolicyViolateAction (not used)
==== QoSTrfcProfInAdmissionAction

Figure 5. A Simple Policing Action

## 3.4. Per-Hop Behavior Actions

A Per-Hop Behavior (PHB) is a description of the externally observable forwarding behavior of a DS node applied to a particular DS behavior aggregate [DIFFSERV]. The approach taken here is that a PHB action specifies both observable forwarding behavior (e.g., loss, delay, jitter) as well as specifying the buffer and bandwidth resources that need to be allocated to each of the behavior aggregates in order to achieve this behavior. That is, a rule with a set of PHB actions can specify that an EF packet must not be delayed more than 20 msec in each hop. The same rule may also specify that EF packets need to be treated with preemptive forwarding (e.g., with priority queuing), and specify the maximum bandwidth for this class, as well as the maximum buffer resources. PHB actions can therefore be used both to represent the final requirements from PHBs and to provide enough detail to be able to map the PHB actions into a set of configuration parameters to configure queues, schedulers, droppers and other mechanisms.

The QoSPolicyPHBAction abstract class has two subclasses. The QoSPolicyBandwidthAction class is used to control bandwidth, delay and forwarding behavior, while the QoSPolicyCongestionControlAction

class is used to control queue size, thresholds and congestion algorithms. The qpMaxPacketSize property of the QoSPolicyPHBAction class specifies the packet size in bytes, and is needed when translating the bandwidth and congestion control actions into actual implementation configurations. For example, an implementation measuring queue length in bytes will need to use this property to map the qpQueueSize property into the desired queue length in bytes.

### 3.4.1. Controlling Bandwidth and Delay

QoSPolicyBandwidthAction allows specifying the minimal bandwidth that should be reserved for a class of traffic. The property qpMinBandwidth can be specified either in Kb/sec or as a percentage of the total available bandwidth. The property qpBandwidthUnits is used to determine whether percentages or fixed values are used.

The property qpForwardingPriority is used whenever preemptive forwarding is required. A policy rule that defines the EF PHB should indicate a non-zero forwarding priority. The qpForwardingPriority property holds an integer value to enable multiple levels of preemptive forwarding where higher values are used to specify higher priority.

The property qpMaxBandwidth specifies the maximum bandwidth that should be allocated to a class of traffic. This property may be specified in PHB actions with non-zero forwarding priority in order to guard against starvation of other PHBs.

The properties qpMaxDelay and qpMaxJitter specify limits on the per-hop delay and jitter in milliseconds for any given packet within a traffic class. Enforcement of the maximum delay and jitter may require use of preemptive forwarding as well as minimum and maximum bandwidth controls. Enforcement of low max delay and jitter values may also require fragmentation and interleave mechanisms over low speed links.

The Boolean property qpFairness indicates whether flows should have a fair chance to be forwarded without drop or delay. A way to enforce a bandwidth action with qpFairness set to TRUE would be to build a queue per flow for the class of traffic specified in the rule’s filter. In this way, interactive flows like terminal access will not be queued behind a bursty flow (like FTP) and therefore have a reasonable response time.

### 3.4.2. Congestion Control Actions

The QoSPolicyCongestionControlAction class controls queue length, thresholds and congestion control algorithms.

A PEP should be able to keep in its queues qpQueueSize packets matching the rule's condition. In order to provide a link-speed independent queue size, the qpQueueSize property can also be measured in milliseconds. The time interval specifies the time needed to transmit all packets within the queue if the link speed is dedicated entirely for transmission of packets within this queue. The property qpQueueSizeUnit determines whether queue size is measured in number of packets or in milliseconds. The property qpDropMethod selects either tail-drop, head-drop or random-drop algorithms. The set of maximum and minimum threshold values can be specified as well, using qpDropMinThresholdValue and qpDropMaxThresholdValue properties, either in packets or in percentage of the total available queue size as specified by the qpDropThresholdUnits property.

### 3.4.3. Using Hierarchical Policies: Examples for PHB Actions

Hierarchical policy definition is a primary tool in the QoS Policy information model. Rule nesting introduced in [PCIMe] allows specification of hierarchical policies controlling RSVP requests, hierarchical shaping, policing and marking actions, as well as hierarchical schedulers and definition of the differences in PHB groups.

This example provides a set of rules that specify PHBs enforced within a Differentiated Service domain. The network administrator chose to enforce the EF, AF11 and AF13 and Best Effort PHBs. For simplicity, AF12 is not differentiated. The set of rules takes the form:

```
If (EF) then do EF actions
If (AF1) then do AF1 actions
    If (AF11) then do AF11 actions
    If (AF12) then do AF12 actions
    If (AF13) then do AF13 actions
If (default) then do Default actions.
```

EF, AF1, AF11, AF12 and AF13 are conditions that filter traffic according to DSCP values. The AF1 condition matches the entire AF1 PHB group including the AF11, AF12 and AF13 DSCP values. The default rule specifies the Best Effort rules. The nesting of the AF1x rules within the AF1 rule specifies that there are further refinements on how AF1x traffic should be treated relative to the entire AF1 PHB group. The set of rules reside in a PolicyGroup with a decision strategy property set to 'FirstMatching'.

The class instances below specify the set of actions used to describe each of the PHBs. Queue sizes are not specified, but can easily be added to the example.

The actions used to describe the Best Effort PHB are simple. No bandwidth is allocated to Best Effort traffic. The first action specifies that Best Effort traffic class should have fairness.

```
QoSPolicyBandwidthAction  BE-B:
  qpFairness: TRUE
```

The second action specifies that the congestion algorithm for the Best Effort traffic class should be random, and specifies the thresholds in percentage of the default queue size.

```
QoSPolicyCongestionControlAction  BE-C:
  qpDropMethod: random
  qpDropThresholdUnits %
  qpDropMinThreshold:  10%
  qpDropMaxThreshold:  70%
```

EF requires preemptive forwarding. The maximum bandwidth is also specified to make sure that the EF class does not starve the other classes. EF PHB uses tail drop as the applications using EF are supposed to be UDP-based and therefore would not benefit from a random dropper.

```
QoSPolicyBandwidthAction  EF-B:
  qpForwardingPriority: 1
  qpBandwidthUnits: %
  qpMaxBandwidth  50%
  qpFairness: FALSE

QoSPolicyCongestionControlAction  EF-C:
  qpDropMethod: tail-drop
  qpDropThresholdUnits packet
  qpDropMaxThreshold:  3 packets
```

The AF1 actions define the bandwidth allocations for the entire PHB group:

```
QoSPolicyBandwidthAction  AF1-B:
  qpBandwidthUnits: %
  qpMinBandwidth: 30%
```

The AF1i actions specifies the differentiating refinement for the AF1x PHBs within the AF1 PHB group. The different threshold values provide the difference in discard probability of the AF1x PHBs within the AF1 PHB group.

```
QoSPolicyCongestionControlAction  AF11-C:
  qpDropMethod: random
  qpDropThresholdUnits packet
  qpDropMinThreshold:  6 packets
  qpDropMaxThreshold:  16 packets

QoSPolicyCongestionControlAction  AF12-C:
  qpDropMethod: random
  qpDropThresholdUnits packet
  qpDropMinThreshold:  4 packets
  qpDropMaxThreshold:  13 packets

QoSPolicyCongestionControlAction  AF13-C:
  qpDropMethod: random
  qpDropThresholdUnits packet
  qpDropMinThreshold:  2 packets
  qpDropMaxThreshold:  10 packets
```

## 4. Traffic Profiles

Meters measure the temporal state of a flow or a set of flows against a traffic profile. In this document, traffic profiles are modeled by the QoSPolicyTrfcProf class. The association QoSPolicyTrfcProf InAdmissionAction binds the traffic profile to the admission action using it. Two traffic profiles are derived from the abstract class QoSPolicyTrfcProf. The first is a Token Bucket provisioning traffic profile carrying rate and burst parameters. The second is an RSVP traffic profile, which enables flows to be compared with RSVP TSPEC and FLOWSPEC parameters.

### 4.1. Provisioning Traffic Profiles

Provisioned Admission Actions, including shaping and policing, are specified using a two- or three-parameter token bucket traffic profile. The QoSPolicyTokenBucketTrfcProf class includes the following properties:

1. Rate measured in kbits/sec
2. Normal burst measured in bytes
3. Excess burst measured in bytes

Rate determines the long-term average transmission rate. Traffic that falls under this rate is conforming, as long as the normal burst is not exceeded at any time. Traffic exceeding the normal burst but still below the excess burst is exceeding the traffic profile. Traffic beyond the excess burst is said to be violating the traffic profile.

Excess burst size is measured in bytes in addition to the burst size. A zero excess burst size indicates that no excess burst is allowed.

## 4.2. RSVP traffic profiles

RSVP admission policy can condition the decision whether to accept or deny an RSVP request based on the traffic specification of the flow (TSPEC) or the amount of QoS resources requested (FLOWSPEC). The admission decision can be based on matching individual RSVP requests against a traffic profile or by matching the aggregated sum of all FLOWSPECs (TSPECs) currently admitted, as determined by the qpAdmissionScope property in an associated QoSPolicyRSVPAdmissionAction.

The QoSPolicyIntservTrfcProf class models both such traffic profiles. This class has the following properties:

1. Token Rate (r) measured in bits/sec
2. Peak Rate (p) measured in bits/sec
3. Bucket Size (b) measured in bytes
4. Min Policed unit (m) measured in bytes
5. Max packet size (M) measured in bytes
6. Resv Rate (R) measured in bits/sec
7. Slack term (s) measured in microseconds

The first five parameters are the traffic specification parameters used in the Integrated Service architecture ([INTSERV]). These parameters are used to define a sender TSPEC as well as a FLOWSPEC for the Controlled-Load service [CL]. For a definition and full explanation of their meanings, please refer to [RSVP-IS].

Parameters 6 and 7 are the additional parameters used for specification of the Guaranteed Service FLOWSPEC [GS].

A partial order is defined between TSPECs (and FLOWSPECs). The TSPEC A is larger than the TSPEC B if and only if rA>rB, pA>pB, bA>bB, mA<mB and MA>MB. A TSPEC (FLOWSPEC) measured against a traffic profile uses the same ordering rule. An RSVP message is accepted only if its TSPEC (FLOWSPEC) is either smaller or equal to the traffic profile. Only parameters specified in the traffic profile are compared.

The GS FLOWSPEC is compared against the rate R and the slack term s. The term R should not be larger than the traffic profile R parameter, while the FLOWSPEC slack term should not be smaller than that specified in the slack term.

TSPECs as well as FLOWSPECs can be added. The sum of two TSPECs is computed by summing the rate r, the peak rate p, the bucket size b, and by taking the minimum value of the minimum policed unit m and the maximum value of the maximum packet size M. GS FLOWSPECs are summed by adding the Resv rate and minimizing the slack term s. These rules are used to compute the temporal state of admitted RSVP states matching the traffic class defined by the rule condition. This state is compared with the traffic profile to arrive at an admission decision when the scope of the QoSPolicyRSVPAdmissionAction is set to 'class'.

## 5. Pre-Defined QoS-Related Variables

Pre-defined variables are necessary for ensuring interoperability among policy servers and policy management tools from different vendors. The purpose of this section is to define frequently used variables in QoS policy domains.

Notice that this section only adds to the variable classes as defined in [PCIMe] and reuses the mechanism defined there.

The QoS policy information model specifies a set of pre-defined variable classes to support a set of fundamental QoS terms that are commonly used to form conditions and actions and are missing from the [PCIMe]. Examples of these include RSVP related variables. All variable classes defined in this document extend the QoSPolicyRSVPVariable class (defined in this document), which itself extends the PolicyImplictVariable class, defined in [PCIMe]. Subclasses specify the data type and semantics of the policy variables.

This document defines the following RSVP variable classes; for details, see their class definitions:

RSVP related Variables:

1. QoSPolicyRSVPSourceIPv4Variable - The source IPv4 address of the RSVP signaled flow, as defined in the RSVP PATH SENDER_TEMPLATE and RSVP RESV FILTER_SPEC [RSVP] objects.

2. QoSPolicyRSVPDestinationIPv4Variable - The destination port of the RSVP signaled flow, as defined in the RSVP PATH and RESV SESSION [RSVP] objects (for IPv4 traffic).

3. QoSPolicyRSVPSourceIPv6Variable - The source IPv6 address of the RSVP signaled flow, as defied in the RSVP PATH SENDER_TEMPLATE and RSVP RESV FILTER_SPEC [RSVP] objects.

4. QoSPolicyRSVPDestinationIPv6Variable - The destination port of the RSVP signaled flow, as defined in the RSVP PATH and RESV SESSION [RSVP] objects (for IPv6 traffic).

5. QoSPolicyRSVPSourcePortVariable - The source port of the RSVP signaled flow, as defined in the RSVP PATH SENDER_TEMPLATE and RSVP RESV FILTER_SPEC [RSVP] objects.

6. QoSPolicyRSVPDestinationPortVariable - The destination port of the RSVP signaled flow, as defined in the RSVP PATH and RESV SESSION [RSVP] objects.

7. QoSPolicyRSVPIPProtocolVariable - The IP Protocol of the RSVP signaled flow, as defined in the RSVP PATH and RESV SESSION [RSVP] objects.

8. QoSPolicyRSVPIPVersionVariable - The version of the IP addresses carrying the RSVP signaled flow, as defined in the RSVP PATH and RESV SESSION [RSVP] objects.

9. QoSPolicyRSVPDCLASSVariable - The DSCP value as defined in the RSVP DCLASS [DCLASS] object.

10. QoSPolicyRSVPStyleVariable - The reservation style (FF, SE, WF) as defined in the RSVP RESV message [RSVP].

11. QoSPolicyRSVPIntServVariable - The type of Integrated Service (CL, GS, NULL) requested in the RSVP Reservation message, as defined in the FLOWSPEC RSVP Object [RSVP].

12. QoSPolicyRSVPMessageTypeVariable - The RSVP message type, either PATH, PATHTEAR, RESV, RESVTEAR, RESVERR, CONF or PATHERR [RSVP].

13. QoSPolicyRSVPPreemptionPriorityVariable - The RSVP reservation priority as defined in [RFC3181].

14. QoSPolicyRSVPPreemptionDefPriorityVariable - The RSVP preemption reservation defending priority as defined in [RFC3181].

15. QoSPolicyRSVPUserVariable - The ID of the user that initiated the flow as defined in the User Locator string in the Identity Policy Object [RFC3182].

16. QoSPolicyRSVPApplicationVariable - The ID of the application that generated the flow as defined in the application locator string in the Application policy object [RFC2872].

17. QoSPolicyRSVPAuthMethodVariable - The RSVP Authentication type used in the Identity Policy Object [RFC3182].

Each class restricts the possible value types associated with a specific variable. For example, the QoSPolicyRSVPSourcePortVariable class is used to define the source port of the RSVP signaled flow. The value associated with this variable is of type PolicyIntegerValue.

## 6. QoS Related Values

Values are used in the information model as building blocks for the policy conditions and policy actions, as described in [PCIM] and [PCIMe]. This section defines a set of auxiliary values that are used for QoS policies as well as other policy domains.

All value classes extend the PolicyValue class [PCIMe]. The subclasses specify specific data/value types that are not defined in [PCIMe].

This document defines the following two subclasses of the PolicyValue class:

| | |
|---|---|
| QoSPolicyDNValue | This class is used to represent a single or set of Distinguished Name [DNDEF] values, including wildcards. A Distinguished Name is a name that can be used as a key to retrieve an object from a directory service. This value can be used in comparison to reference values carried in RSVP policy objects, as specified in [RFC3182]. This class is defined in Section 8.31. |
| QoSPolicyAttributeValue | A condition term uses the form "Variable matches Value", and an action term uses the form "set Variable to Value" ([PCIMe]). This class is used to represent a single or set of property values for the "Value" term in either a condition or an action. This value can be used in conjunction with reference values carried in RSVP objects, as specified in [RFC3182]. This class is defined in section 8.12. |

The property name is used to specify which of the properties in the QoSPolicyAttributeValue class instance is being used in the condition or action term. The value of this property or properties will then

be retrieved. In the case of a condition, a match (which is dependent on the property name) will be used to see if the condition is satisfied or not. In the case of an action, the semantics are instead "set the variable to this value".

For example, suppose the "user" objects in the organization include several properties, among them:

- First Name
- Last Name
- Login Name
- Department
- Title

A simple condition could be constructed to identify flows by their RSVP user carried policy object. The simple condition: Last Name = "Smith" to identify a user named Bill would be constructed in the following way:

A SimplePolicyCondition [PCIMe] would aggregate a QoSPolicyRSVPUserVariable [QPIM] object, via the PolicyVariableInSimplePolicyCondition [PCIMe] aggregation.

The implicit value associated with this condition is created in the following way:

A QoSPolicyAttributeValue object would be aggregated to the simple condition object via a PolicyValueInSimplePolicyCondition [PCIMe]. The QoSPolicyAttributeValue attribute qpAttributeName would be set to "last name" and the qpAttributeValueList would be set to "Smith".

Another example is a condition that has to do with the user’s organizational department. It can be constructed in the exact same way, by changing the QoSPolicyAttributeValue attribute qpAttributeName to "Department" and the qpAttributeValueList would be set to the particular value that is to be matched (e.g., "engineering" or "customer support"). The logical condition would than be evaluated to true if the user belong to either the engineering department or the customer support.

Notice that many multiple-attribute objects require the use of the QoSPolicyAttributeValue class to specify exactly which of its attributes should be used in the condition match operation.

# 7. Class Definitions: Association Hierarchy

The following sections define associations that are specified by QPIM.

## 7.1. The Association "QoSPolicyTrfcProfInAdmissionAction"

This association links a QoSPolicyTrfcProf object (defined in section 8.9), modeling a specific traffic profile, to a QoSPolicyAdmissionAction object (defined in section 8.2). The class definition for this association is as follows:

| | |
|---|---|
| NAME | QoSPolicyTrfcProfInAdmissionAction |
| DESCRIPTION | A class representing the association between a QoS admission action and its traffic profile. |
| DERIVED FROM | Dependency (See [PCIM]) |
| ABSTRACT | FALSE |
| PROPERTIES | Antecedent[ref QoSPolicyAdmissionAction [0..n]]<br>Dependent[ref QoSPolicyTrfcProf [1..1]] |

### 7.1.1. The Reference "Antecedent"

This property is inherited from the Dependency association, defined in [PCIM]. Its type is overridden to become an object reference to a QoSPolicyAdmissionAction object. This represents the "independent" part of the association. The [0..n] cardinality indicates that any number of QoSPolicyAdmissionAction object(s) may use a given QoSPolicyTrfcProf.

### 7.1.2. The Reference "Dependent"

This property is inherited from the Dependency association, and is overridden to become an object reference to a QoSPolicyTrfcProf object. This represents a specific traffic profile that is used by any number of QoSPolicyAdmissionAction objects. The [1..1] cardinality means that exactly one object of the QoSPolicyTrfcProf can be used by a given QoSPolicyAddmissionAction.

## 7.2. The Association "PolicyConformAction"

This association links a policing action with an object defining an action to be applied to conforming traffic relative to the associated traffic profile. The class definition for this association is as follows:

| | |
|---|---|
| NAME | PolicyConformAction |
| DESCRIPTION | A class representing the association between a policing action and the action that should be applied to traffic conforming to an associated traffic profile. |
| DERIVED FROM | Dependency (see [PCIM]) |
| ABSTRACT | FALSE |
| PROPERTIES | Antecedent[ref QoSPolicyPoliceAction[0..n]]<br>Dependent[ref PolicyAction [1..1]] |

#### 7.2.1. The Reference "Antecedent"

This property is inherited from the Dependency association. Its type is overridden to become an object reference to a QoSPolicyPoliceAction object. This represents the "independent" part of the association. The [0..n] cardinality indicates that any number of QoSPolicyPoliceAction objects may be given the same action to be executed as the conforming action.

#### 7.2.2. The Reference "Dependent"

This property is inherited from the Dependency association, and is overridden to become an object reference to a PolicyAction object. This represents a specific policy action that is used by a given QoSPolicyPoliceAction. The [1..1] cardinality means that exactly one policy action can be used as the "conform" action for a QoSPolicyPoliceAction. To execute more than one conforming action, use the PolicyCompoundAction class to model the conforming action.

### 7.3. The Association "QoSPolicyExceedAction"

This association links a policing action with an object defining an action to be applied to traffic exceeding the associated traffic profile. The class definition for this association is as follows:

| | |
|---|---|
| NAME | QoSPolicyExceedAction |
| DESCRIPTION | A class representing the association between a policing action and the action that should be applied to traffic exceeding an associated traffic profile. |
| DERIVED FROM | Dependency (see [PCIM]) |
| ABSTRACT | FALSE |
| PROPERTIES | Antecedent[ref QoSPolicePoliceAction[0..n]]<br>Dependent[ref PolicyAction [1..1]] |

### 7.3.1. The Reference "Antecedent"

This property is inherited from the Dependency association. Its type is overridden to become an object reference to a QoSPolicyPoliceAction object. This represents the "independent" part of the association. The [0..n] cardinality indicates that any number of QoSPolicyPoliceAction objects may be given the same action to be executed as the exceeding action.

### 7.3.2. The Reference "Dependent"

This property is inherited from the Dependency association, and is overridden to become an object reference to a PolicyAction object. This represents a specific policy action that is used by a given QoSPolicyPoliceAction. The [1..1] cardinality means that a exactly one policy action can be used as the "exceed" action by a QoSPolicyPoliceAction. To execute more than one conforming action, use the PolicyCompoundAction class to model the exceeding action.

## 7.4. The Association "PolicyViolateAction"

This association links a policing action with an object defining an action to be applied to traffic violating the associated traffic profile. The class definition for this association is as follows:

| | |
|---|---|
| NAME | PolicyViolateAction |
| DESCRIPTION | A class representing the association between a policing action and the action that should be applied to traffic violating an associated traffic profile. |
| DERIVED FROM | Dependency (see [PCIM]) |
| ABSTRACT | FALSE |
| PROPERTIES | Antecedent[ref QoSPolicePoliceAction[0..n]]<br>Dependent[ref PolicyAction [1..1]] |

### 7.4.1. The Reference "Antecedent"

This property is inherited from the Dependency association. Its type is overridden to become an object reference to a QoSPolicyPoliceAction object. This represents the "independent" part of the association. The [0..n] cardinality indicates that any number of QoSPolicyPoliceAction objects may be given the same action to be executed as the violating action.

### 7.4.2. The Reference "Dependent"

This property is inherited from the Dependency association, and is overridden to become an object reference to a PolicyAction object. This represents a specific policy action that is used by a given QoSPolicyPoliceAction. The [1..1] cardinality means that exactly one policy action can be used as the "violate" action by a QoSPolicyPoliceAction. To execute more than one violating action, use the PolicyCompoundAction class to model the conforming action.

## 7.5. The Aggregation "QoSPolicyRSVPVariableInRSVPSimplePolicyAction"

A simple RSVP policy action is represented as a pair {variable, value}. This aggregation provides the linkage between a QoSPolicyRSVPSimpleAction instance and a single QoSPolicyRSVPVariable. The aggregation PolicyValueInSimplePolicyAction links the QoSPolicyRSVPSimpleAction to a single PolicyValue.

The class definition for this aggregation is as follows:

```
NAME            QoSPolicyRSVPVariableInRSVPSimplePolicyAction
DERIVED FROM    PolicyVariableInSimplePolicyAction
ABSTRACT        FALSE
PROPERTIES      GroupComponent[ref QoSPolicyRSVPSimpleAction
                  [0..n]]
                PartComponent[ref QoSPolicyRSVPVariable [1..1] ]
```

### 7.5.1. The Reference "GroupComponent"

The reference property "GroupComponent" is inherited from PolicyComponent, and overridden to become an object reference to a QoSPolicyRSVPSimpleAction that contains exactly one QoSPolicyRSVPVariable. Note that for any single instance of the aggregation class QoSPolicyRSVPVariableInRSVPSimplePolicyAction, this property is single-valued. The [0..n] cardinality indicates that there may be 0, 1, or more QoSPolicyRSVPSimpleAction objects that contain any given RSVP variable object.

### 7.5.2. The Reference "PartComponent"

The reference property "PartComponent" is inherited from PolicyComponent, and overridden to become an object reference to a QoSPolicyRSVPVariable that is defined within the scope of a QoSPolicyRSVPSimpleAction. Note that for any single instance of the association class QoSPolicyRSVPVariableInRSVPSimplePolicyAction, this property (like all reference properties) is single-valued. The

[1..1] cardinality indicates that a QoSPolicyRSVPVariableInRSVPSimplePolicyAction must have exactly one RSVP variable defined within its scope in order to be meaningful.

# 8. Class Definitions: Inheritance Hierarchy

The following sections define object classes that are specified by QPIM.

## 8.1. The Class QoSPolicyDiscardAction

This class is used to specify that packets should be discarded. This is the same as stating that packets should be denied forwarding. The class definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyDiscardAction |
| DESCRIPTION | This action specifies that packets should be discarded. |
| DERIVED FROM | PolicyAction (defined in [PCIM]) |
| ABSTRACT | FALSEFALSE |
| PROPERTIES | None |

## 8.2. The Class QoSPolicyAdmissionAction

This class is the base class for performing admission decisions based on a comparison of a meter measuring the temporal behavior of a flow or a set of flow with a traffic profile. The qpAdmissionScope property controls whether the comparison is done per flow or per class (of flows). Only packets that conform to the traffic profile are admitted for further processing; other packets are discarded. The class definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyAdmissionAction |
| DESCRIPTION | This action controls admission decisions based on comparison of a meter to a traffic profile. |
| DERIVED FROM | PolicyAction (defined in [PCIM]) |
| ABSTRACT | FALSEFALSE |
| PROPERTIES | qpAdmissionScope |

### 8.2.1. The Property qpAdmissionScope

This attribute specifies whether the admission decision is done per flow or per the entire class of flows defined by the rule condition. If the scope is "flow", the actual or requested rate of each flow is compared against the traffic profile. If the scope is set to "class", the aggregate actual or requested rate of all flows matching the rule condition is measured against the traffic profile. The property is defined as follows:

| | |
|---|---|
| NAME | qpAdmissionScope |
| DESCRIPTION | This property specifies whether the admission decision is done per flow or per the entire class of flows. |
| SYNTAX | Integer |
| VALUE | This is an enumerated integer. A value of 0 specifies that admission is done on a per-flow basis, and a value of 1 specifies that admission is done on a per-class basis. |

## 8.3. The Class QoSPolicyPoliceAction

This is used for defining policing actions (i.e., those actions that restrict traffic based on a comparison with a traffic profile). Using the three associations QoSPolicyConformAction, QoSPolicyExceedAction and QoSPolicyViolateAction, it is possible to specify different actions to take based on whether the traffic is conforming, exceeding, or violating a traffic profile. The traffic profile is specified in a subclass of the QoSPolicyTrfcProf class. The class definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyPoliceAction |
| DESCRIPTION | This action controls the operation of policers. The rate of flows is measured against a traffic profile. The actions that need to be performed on conforming, exceeding and violating traffic are indicated using the conform, exceed and violate action associations. |
| DERIVED FROM | QoSPolicyAdmissionAction (defined in this document) |
| ABSTRACT | FALSEFALSE |
| PROPERTIES | None |

## 8.4. The Class QoSPolicyShapeAction

This class is used for defining shaping actions. Shapers are used to delay some or all of the packets in a traffic stream in order to bring a particular traffic stream into compliance with a given traffic profile. The traffic profile is specified in a subclass of the QoSPolicyTrfcProf class. The class definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyShapeAction |
| DESCRIPTION | This action indicate that traffic should be shaped to be conforming with a traffic profile. |
| DERIVED FROM | QoSPolicyAdmissionAction (defined in this document) |
| ABSTRACT | FALSEFALSE |
| PROPERTIES | None |

### 8.5. The Class QoSPolicyRSVPAdmissionAction

This class determines whether to accept or reject a given RSVP request by comparing the RSVP request's TSPEC or RSPEC parameters against the associated traffic profile and/or by enforcing the pre-set maximum sessions limit. The traffic profile is specified in the QoSPolicyIntServTrfcProf class. This class inherits the qpAdmissionScope property from its superclass. This property specifies whether admission should be done on a per-flow or per-class basis. If the traffic profile is not larger than or equal to the requested reservation, or to the sum of the admitted reservation merged with the requested reservation, the result is a deny decision. If no traffic profile is specified, the assumption is that all traffic can be admitted.

The class definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyRSVPAdmissionAction |
| DESCRIPTION | This action controls the admission of RSVP requests. Depending on the scope, either a single RSVP request or the total admitted RSVP requests matching the conditions are compared against a traffic profile. |
| DERIVED FROM | QoSPolicyAdmissionAction (defined in this document) |
| ABSTRACT | FALSEFALSE |
| PROPERTIES | qpRSVPWarnOnly, qpRSVPMaxSessions |

#### 8.5.1. The Property qpRSVPWarnOnly

This property is applicable when fulfilling ("admitting") an RSVP request would violate the policer (traffic profile) limits or when the maximum number session would be exceeded (or both).

When this property is set to TRUE, the RSVP request is admitted in spite of the violation, but an RSVP error message carrying a warning is sent to the originator (sender or receiver). When set to FALSE, the request would be denied and an error message would be sent back to the originator. So the meaning of the qpWarnOnly flag is: Based on property's value (TRUE or FALSE), determine whether to admit but warn the originator that the request is in violation or to deny the request altogether (and send back an error).

Specifically, a PATHERR (in response to a Path message) or a RESVERR (in response of a RESV message) will be sent. This follows the COPS for RSVP send error flag in the Decision Flags object. This property is defined as follows:

| | |
|---|---|
| NAME | qpRSVPWarnOnly |
| SYNTAX | Boolean |
| Default | FALSE |
| VALUE | The value TRUE means that the request should be admitted AND an RSVP warning message should be sent to the originator. The value of FALSE means that the request should be not admitted and an appropriate error message should be sent back to the originator of the request. |

### 8.5.2. The Property qpRSVPMaxSessions

This attribute is used to limit the total number of RSVP requests admitted for the specified class of traffic. For this property to be meaningful, the qpAdmissionScope property must be set to class. The definition of this property is as follows:

| | |
|---|---|
| NAME | qpRSVPMaxSessions |
| SYNTAX | Integer |
| VALUE | Must be greater than 0. |

## 8.6. The Class QoSPolicyPHBAction

This class is a base class that is used to define the per-hop behavior that is to be assigned to behavior aggregates. It defines a common property, qpMaxPacketSize, for use by its subclasses (QoSPolicyBandwidthAction and QoSPolicyCongestionControlAction). The class definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyPHBAction |
| DESCRIPTION | This action controls the Per-Hop-Behavior provided to behavior aggregates. |
| DERIVED FROM | PolicyAction (defined in [PCIM]) |
| ABSTRACT | TRUE |
| PROPERTIES | qpMaxPacketSize |

### 8.6.1. The Property qpMaxPacketSize

This property specifies the maximum packet size in bytes, of packets in the designated flow. This attribute is used in translation of QPIM attributes to QoS mechanisms used within a PEP. For example, queue length may be measured in bytes, while the minimum number of packets that should be kept in a PEP is defined within QPIM in number of packets. This property is defined as follows:

| | |
|---|---|
| NAME | qpMaxPacketSize |
| SYNTAX | Integer |
| Value | Must be greater than 0 |

### 8.7. The Class QoSPolicyBandwidthAction

This class is used to control the bandwidth, delay, and forwarding behavior of a PHB. Its class definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyBandwidthAction |
| DESCRIPTION | This action controls the bandwidth, delay, and forwarding characteristics of the PHB. |
| DERIVED FROM | QoSPolicyPBHAction (defined in this document) |
| ABSTRACT | FALSE |
| PROPERTIES | qpForwardingPriority, qpBandwidthUnits, qpMinBandwdith, qpMaxBandwidth, qpMaxDelay, qpMaxJitter, qpFairness |

#### 8.7.1. The Property qpForwardingPriority

This property defines the forwarding priority for this set of flows. A non-zero value indicates that preemptive forwarding is required. Higher values represent higher forwarding priority. This property is defined as follows:

| | |
|---|---|
| NAME | qpForwardingPriority |
| SYNTAX | Integer |
| VALUE | Must be non-negative. The value 0 means that preemptive forwarding is not required. A positive value indicates the priority that is to be assigned for this (set of) flow(s). Larger values represent higher priorities. |

#### 8.7.2. The Property qpBandwidthUnits

This property defines the units that the properties qpMinBandwidth and qpMaxBandwidth have. Bandwidth can either be defined in bits/sec or as a percentage of the available bandwidth or scheduler resources. This property is defined as follows:

| | |
|---|---|
| NAME | qpBandwidthUnits |
| SYNTAX | Integer |
| VALUE | Two values are possible. The value of 0 is used to specify units of bits/sec, while the value of 1 is used to specify units as a percentage of the available bandwidth. If this property indicates that the bandwidth units are percentages, then each of the bandwidth properties expresses a whole-number percentage, and hence its maximum value is 100. |

### 8.7.3. The Property qpMinBandwidth

This property defines the minimum bandwidth that should be reserved for this class of traffic. Both relative (i.e., a percentage of the bandwidth) and absolute (i.e., bits/second) values can be specified according to the value of the qpBandwidthUnits property. This property is defined as follows:

| | |
|---|---|
| NAME | qpMinBandwidth |
| SYNTAX | Integer |
| VALUE | The value must be greater than 0. If the property qpMaxBandwidth is defined, then the value of qpMinBandwidth must be less than or equal to the value of qpMaxBandwidth. |

### 8.7.4. The Property qpMaxBandwidth

This property defines the maximum bandwidth that should be allocated to this class of traffic. Both relative (i.e., a percentage of the bandwidth)and absolute (i.e., bits/second) values can be specified according to the value of the qpBandwidthUnits property. This property is defined as follows:

| | |
|---|---|
| NAME | qpMaxBandwidth |
| SYNTAX | Integer |
| VALUE | The value must be greater than 0. If the property qpMaxBandwidth is defined, then the value of qpMinBandwidth must be less than or equal to the value of qpMaxBandwidth. |

### 8.7.5. The Property qpMaxDelay

This property defines the maximal per-hop delay that traffic of this class should experience while being forwarded through this hop. The maximum delay is measured in microseconds. This property is defined as follows:

| | |
|---|---|
| NAME | qpMaxDelay |
| SYNTAX | Integer (microseconds) |
| VALUE | The value must be greater than 0. |

### 8.7.6. The Property qpMaxJitter

This property defines the maximal per-hop delay variance that traffic of this class should experience while being forwarded through this hop. The maximum jitter is measured in microseconds. This property is defined as follows:

| | |
|---|---|
| NAME | qpMaxJitter |
| SYNTAX | Integer (microseconds) |
| VALUE | The value must be greater than 0. |

### 8.7.7. The Property qpFairness

This property defines whether fair queuing is required for this class of traffic. This property is defined as follows:

| | |
|---|---|
| NAME | qpFairness |
| SYNTAX | Boolean |
| VALUE | The value of FALSE means that fair queuing is not required for this class of traffic, while the value of TRUE means that fair queuing is required for this class of traffic. |

## 8.8. The Class QoSPolicyCongestionControlAction

This class is used to control the characteristics of the congestion control algorithm being used. The class definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyCongestionControlAction |
| DESCRIPTION | This action control congestion control characteristics of the PHB. |
| DERIVED FROM | QoSPolicyPBHAction (defined in this document) |
| ABSTRACT | FALSE |
| PROPERTIES | qpQueueSizeUnits, qpQueueSize, qpDropMethod, qpDropThresholdUnits, qpDropMinThresholdValue, qpDropMaxThresholdValue |

### 8.8.1. The property qpQueueSizeUnits

This property specifies the units in which the qpQueueSize attribute is measured. The queue size is measured either in number of packets or in units of time. The time interval specifies the time needed to transmit all packets within the queue if the link speed is dedicated entirely to transmission of packets within this queue. The property definition is:

| | |
|---|---|
| NAME | qpQueueSizeUnits |
| SYNTAX | Integer |
| VALUE | This property can have two values. If the value is set to 0, then the unit of measurement is number of packets. If the value is set to 1, then the unit of measurement is milliseconds. |

### 8.8.2. The Property qpQueueSize

This property specifies the maximum queue size in packets or in milliseconds, depending on the value of the qpQueueSizeUnits (0 specifies packets, and 1 specifies milliseconds). This property is defined as follows:

| | |
|---|---|
| NAME | qpQueueSize |
| SYNTAX | Integer |
| VALUE | This value must be greater than 0. |

### 8.8.3. The Property qpDropMethod

This property specifies the congestion control drop algorithm that should be used for this type of traffic. This property is defined as follows:

| | |
|---|---|
| NAME | qpDropMethod |
| SYNTAX | Integer |
| VALUES | Three values are currently defined. The value 0 specifies a random drop algorithm, the value 1 specifies a tail drop algorithm, and the value 2 specifies a head drop algorithm. |

### 8.8.4. The Property qpDropThresholdUnits

This property specifies the units in which the two properties qpDropMinThresholdValue and qpDropMaxThresholdValue are measured. Thresholds can be measured either in packets or as a percentage of the available queue sizes. This property is defined as follows:

| | |
|---|---|
| NAME | qpDropThresholdUnits |
| SYNTAX | Integer |
| VALUES | Three values are defined. The value 0 defines the units as number of packets, the value 1 defines the units as a percentage of the queue size and the value 2 defines the units in milliseconds. If this property indicates that the threshold units are percentages, then each of the threshold properties expresses a whole-number percentage, and hence its maximum value is 100. |

### 8.8.5. The Property qpDropMinThresholdValue

This property specifies the minimum number of queuing and buffer resources that should be reserved for this class of flows. The threshold can be specified as either relative (i.e., a percentage) or absolute (i.e., number of packets or millisecond) value according to the value of the qpDropThresholdUnits property. If this property

specifies a value of 5 packets, then enough buffer and queuing resources should be reserved to hold 5 packets before running the specified congestion control drop algorithm. This property is defined as follows:

| | |
|---|---|
| NAME | qpDropMinThresholdValue |
| SYNTAX | Integer |
| VALUE | This value must be greater than or equal to 0. If the property qpDropMaxThresholdValue is defined, then the value of the qpDropMinThresholdValue property must be less than or equal to the value of the qpDropMaxThresholdValue property. |

### 8.8.6. The Property qpDropMaxThresholdValue

This property specifies the maximum number of queuing and buffer resources that should be reserved for this class of flows. The threshold can be specified as either relative (i.e., a percentage) or absolute (i.e., number of packets or milliseconds) value according to the value of the qpDropThresholdUnits property. Congestion Control droppers should not keep more packets than the value specified in this property. Note, however, that some droppers may calculate queue occupancy averages, and therefore the actual maximum queue resources should be larger. This property is defined as follows:

| | |
|---|---|
| NAME | qpDropMaxThresholdValue |
| SYNTAX | Integer |
| VALUE | This value must be greater than or equal to 0. If the property qpDropMinThresholdValue is defined, then the value of the qpDropMinThresholdValue property must be less than or equal to the value of the qpDropMaxThresholdValue property. |

## 8.9. Class QoSPolicyTrfcProf

This is an abstract base class that models a traffic profile. Traffic profiles specify the maximum rate parameters used within admission decisions. The association QoSPolicyTrfcProfInAdmissionAction binds the admission decision to the traffic profile. The class definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyTrfcProf |
| DERIVED FROM | Policy (defined in [PCIM]) |
| ABSTRACT | TRUE |
| PROPERTIES | None |

### 8.10. Class QoSPolicyTokenBucketTrfcProf

This class models a two- or three-level Token Bucket traffic profile. Additional profiles can be modeled by cascading multiple instances of this class (e.g., by connecting the output of one instance to the input of another instance). This traffic profile carries the policer or shaper rate values to be enforced on a flow or a set of flows. The class definition is as follows:

```
NAME          QoSPolicyTokenBucketTrfcProf
DERIVED FROM  QoSPolicyTrfcProf (defined in this document)
ABSTRACT      FALSE
PROPERTIES    qpTBRate, qpTBNormalBurst, qpTBExcessBurst
```

#### 8.10.1. The Property qpTBRate

This is a non-negative integer that defines the token rate in kilobits per second. A rate of zero means that all packets will be out of profile. This property is defined as follows:

```
NAME        qpTBRate
SYNTAX      Integer
VALUE       This value must be greater than to 0
```

#### 8.10.2. The Property qpTBNormalBurst

This property is an integer that defines the normal size of a burst measured in bytes. This property is defined as follows:

```
NAME        qpTBNormalBurst
SYNTAX      Integer
VALUE       This value must be greater than to 0
```

#### 8.10.3. The Property qpTBExcessBurst

This property is an integer that defines the excess burst size measured in bytes. This property is defined as follows:

```
NAME        qpTBExcessBurst
SYNTAX      Integer
VALUE       This value must be greater than or equal to
            qpTBNormalBurst
```

### 8.11. Class QoSPolicyIntServTrfcProf

This class represents an IntServ traffic profile. Values of IntServ traffic profiles are compared against Traffic specification (TSPEC) and QoS Reservation (FLOWSPEC) requests carried in RSVP requests.

The class definition is as follows:

```
NAME          QoSPolicyIntServTrfcProf
DERIVED FROM  QoSPolicyTrfcProf (defined in this document)
ABSTRACT      FALSE
PROPERTIES    qpISTokenRate, qpISPeakRate, qpISBucketSize,
              qpISResvRate, qpISResvSlack, qpISMinPolicedUnit,
              qpISMaxPktSize
```

### 8.11.1. The Property qpISTokenRate

This property is a non-negative integer that defines the token rate parameter, measured in kilobits per second. This property is defined as follows:

```
NAME        qpISTokenRate
SYNTAX      Integer
VALUE       This value must be greater than or equal to 0
```

### 8.11.2. The Property qpISPeakRate

This property is a non-negative integer that defines the peak rate parameter, measured in kilobits per second. This property is defined as follows:

```
NAME        qpISPeakRate
SYNTAX      Integer
VALUE       This value must be greater than or equal to 0
```

### 8.11.3. The Property qpISBucketSize

This property is a non-negative integer that defines the token bucket size parameter, measured in bytes. This property is defined as follows:

```
NAME        qpISBucketSize
SYNTAX      Integer
VALUE       This value must be greater than or equal to 0
```

### 8.11.4. The Property qpISResvRate

This property is a non-negative integer that defines the reservation rate (R-Spec) in the RSVP guaranteed service reservation. It is measured in kilobits per second. This property is defined as follows:

| | |
|---|---|
| NAME | qpISResvRate |
| SYNTAX | Integer |
| VALUE | This value must be greater than or equal to 0 |

### 8.11.5. The Property qpISResvSlack

This property is a non-negative integer that defines the RSVP slack term in the RSVP guaranteed service reservation. It is measured in microseconds. This property is defined as follows:

| | |
|---|---|
| NAME | qpISResvSlack |
| SYNTAX | Integer |
| VALUE | This value must be greater than or equal to 0 |

### 8.11.6. The Property qpISMinPolicedUnit

This property is a non-negative integer that defines the minimum RSVP policed unit, measured in bytes. This property is defined as follows:

| | |
|---|---|
| NAME | qpISMinPolicedUnit |
| SYNTAX | Integer |
| VALUE | This value must be greater than or equal to 0 |

### 8.11.7. The Property qpISMaxPktSize

This property is a positive integer that defines the maximum allowed packet size for RSVP messages, measured in bytes. This property is defined as follows:

| | |
|---|---|
| NAME | qpISMaxPktSize |
| SYNTAX | Integer |
| VALUE | This value must be a positive integer, denoting the number of bytes in the largest payload packet of an RSVP signaled flow or class. |

## 8.12. The Class QoSPolicyAttributeValue

This class can be used for representing an indirection in variable and value references either in a simple condition ("<x> match <y>") or a simple action ("<x> = <y>"). In both cases, <x> and <y> are known as the variable and the value of either the condition or action. The value of the properties qpAttributeName and qpAttributeValueList are used to substitute <x> and <y> in the condition or action respectively.

The substitution is done as follows: The value of the property qpAttributeName is used to substitute <x> and the value of the property qpAttributeValueList is used to substitute <y>.

Once the substitution is done, the condition can be evaluated and the action can be performed.

For example, suppose we want to define a condition over a user name of the form "user == 'Smith'", using the QoSPolicyRSVPUserVariable class. The user information in the RSVP message provides a DN. The DN points to a user objects holding many attributes. If the relevant attribute is "last name", we would use the QoSPolicyAttributeValue class with qpAttributeName = "Last Name", qpAttributeValueList = {"Smith"}.

The class definition is as follows:

```
NAME           QoSPolicyAttributeValue
DERIVED FROM   PolicyValue (defined in [PCIMe])
ABSTRACT       FALSE
PROPERTIES     qpAttributeName, qpAttributeValueList
```

### 8.12.1. The Property qpAttributeName

This property carries the name of the attribute that is to be used to substitute <x> in a simple condition or simple condition of the forms "<x> match <y>" or "<x> = <y>" respectively. This property is defined as follows:

```
NAME       qpAttributeName
SYNTAX     String
```

### 8.12.2. The Property qpAttributeValueList

This property carries a list of values that is to be used to substitute <y> in a simple condition or simple action of the forms "<x> match <y>" or "<x> = <y>" respectively.

This property is defined as follows:

```
NAME       qpAttributeValueList
SYNTAX     String
```

## 8.13. The Class "QoSPolicyRSVPVariable"

This is an abstract class that serves as the base class for all implicit variables that have to do with RSVP conditioning. The class definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyRSVPVariable |
| DESCRIPTION | An abstract base class used to build other classes that specify different attributes of an RSVP request |
| DERIVED FROM | PolicyImplicitVariable (defined in [PCIMe]) |
| ABSTRACT | TRUE |
| PROPERTIES | None |

## 8.14. The Class "QoSPolicyRSVPSourceIPv4Variable"

This is a concrete class that contains the source IPv4 address of the RSVP signaled flow, as defined in the RSVP PATH SENDER_TEMPLATE and RSVP RESV FILTER_SPEC [RSVP] objects. The class definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyRSVPSourceIPv4Variable |
| DESCRIPTION | The source IPv4 address of the RSVP signaled flow, as defined in the RSVP PATH SENDER_TEMPLATE and RSVP RESV FILTER_SPEC [RSVP] objects.<br><br>ALLOWED VALUE TYPES: PolicyIPv4AddrValue |
| DERIVED FROM | QoSPolicyRSVPVariable (defined in this document) |
| ABSTRACT | FALSE |
| PROPERTIES | None |

## 8.15. The Class "QoSPolicyRSVPDestinationIPv4Variable"

This is a concrete class that contains the destination IPv4 address of the RSVP signaled flow, as defined in the RSVP PATH SENDER_TEMPLATE and RSVP RESV FILTER_SPEC [RSVP] objects. The class definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyRSVPDestinationIPv4Variable |
| DESCRIPTION | The destination IPv4 address of the RSVP signaled flow, as defined in the RSVP PATH and RESV SESSION [RSVP] objects.<br><br>ALLOWED VALUE TYPES: PolicyIPv4AddrValue |
| DERIVED FROM | QoSPolicyRSVPVariable (defined in this document) |
| ABSTRACT | FALSE |
| PROPERTIES | None |

### 8.16. The Class "QoSPolicyRSVPSourceIPv6Variable"

This is a concrete class that contains the source IPv6 address of the RSVP signaled flow, as defined in the RSVP PATH SENDER_TEMPLATE and RSVP RESV FILTER_SPEC [RSVP] objects. The class definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyRSVPSourceIPv6Variable |
| DESCRIPTION | The source IPv6 address of the RSVP signaled flow, as defined in the RSVP PATH SENDER_TEMPLATE and RSVP RESV FILTER_SPEC [RSVP] objects.<br><br>ALLOWED VALUE TYPES: PolicyIPv6AddrValue |
| DERIVED FROM | QoSPolicyRSVPVariable (defined in this document) |
| ABSTRACT | FALSE |
| PROPERTIES | None |

### 8.17. The Class "QoSPolicyRSVPDestinationIPv6Variable"

This is a concrete class that contains the destination IPv6 address of the RSVP signaled flow, as defined in the RSVP PATH SENDER_TEMPLATE and RSVP RESV FILTER_SPEC [RSVP] objects. The class definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyRSVPDestinationIPv6Variable |
| DESCRIPTION | The destination IPv6 address of the RSVP signaled flow, as defined in the RSVP PATH and RESV SESSION [RSVP] objects.<br><br>ALLOWED VALUE TYPES: PolicyIPv6AddrValue |
| DERIVED FROM | QoSPolicyRSVPVariable (defined in this document) |
| ABSTRACT | FALSE |
| PROPERTIES | None |

### 8.18. The Class "QoSPolicyRSVPSourcePortVariable"

This class contains the source port of the RSVP signaled flow, as defined in the RSVP PATH SENDER_TEMPLATE and RSVP RESV FILTER_SPEC [RSVP] objects. The class definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyRSVPSourcePortVariable |
| DESCRIPTION | The source port of the RSVP signaled flow, as defined in the RSVP PATH SENDER_TEMPLATE and RSVP RESV FILTER_SPEC [RSVP] objects.<br><br>ALLOWED VALUE TYPES: PolicyIntegerValue (0..65535) |

| | |
|---|---|
| DERIVED FROM | QoSPolicyRSVPVariable (defined in this document) |
| ABSTRACT | FALSE |
| PROPERTIES | None |

## 8.19. The Class "QoSPolicyRSVPDestinationPortVariable"

This is a concrete class that contains the destination port of the RSVP signaled flow, as defined in the RSVP PATH SENDER_TEMPLATE and RSVP RESV FILTER_SPEC [RSVP] objects. The class definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyRSVPDestinationPortVariable |
| DESCRIPTION | The destination port of the RSVP signaled flow, as defined in the RSVP PATH and RESV SESSION [RSVP] objects.<br><br>ALLOWED VALUE TYPES: PolicyIntegerValue (0..65535) |
| DERIVED FROM | QoSPolicyRSVPVariable (defined in this document) |
| ABSTRACT | FALSE |
| PROPERTIES | None |

## 8.20. The Class "QoSPolicyRSVPIPProtocolVariable"

This is a concrete class that contains the IP Protocol number of the RSVP signaled flow, as defined in the RSVP PATH and RESV SESSION [RSVP] objects. The class definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyRSVPIPProtocolVariable |
| DESCRIPTION | The IP Protocol number of the RSVP signaled flow, as defined in the RSVP PATH and RESV SESSION [RSVP] objects.<br><br>ALLOWED VALUE TYPES: PolicyIntegerValue |
| DERIVED FROM | QoSPolicyRSVPVariable (defined in this document) |
| ABSTRACT | FALSE |
| PROPERTIES | None |

## 8.21. The Class "QoSPolicyRSVPIPVersionVariable"

This is a concrete class that contains the IP Protocol version number of the RSVP signaled flow, as defined in the RSVP PATH and RESV SESSION [RSVP] objects. The well-known version numbers are 4 and 6. This variable allows a policy definition of the type:

"If IP version = IPv4 then ...".

The class definition is as follows:

```
NAME           QoSPolicyRSVPIPVersionVariable
DESCRIPTION    The IP version number of the IP Addresses carried the
               RSVP signaled flow, as defined in the RSVP PATH and
               RESV SESSION [RSVP] objects.

               ALLOWED VALUE TYPES: PolciIntegerValue

DERIVED FROM   QoSPolicyRSVPVariable (defined in this document)
ABSTRACT       FALSE
PROPERTIES     None
```

## 8.22. The Class "QoSPolicyRSVPDCLASSVariable"

This is a concrete class that contains the DSCP value as defined in the RSVP DCLASS [DCLASS] object. The class definition is as follows:

```
NAME           QoSPolicyRSVPDCLASSVariable
DESCRIPTION    The DSCP value as defined in the RSVP DCLASS [DCLASS]
               object.

               ALLOWED VALUE TYPES: PolicyIntegerValue,
                                    PolicyBitStringValue

DERIVED FROM   QoSPolicyRSVPVariable (defined in this document)
ABSTRACT       FALSE
PROPERTIES     None
```

## 8.23. The Class "QoSPolicyRSVPStyleVariable"

This is a concrete class that contains the reservation style as defined in the RSVP STYLE object in the RESV message [RSVP]. The class definition is as follows:

```
NAME           QoSPolicyRSVPStyleVariable
DESCRIPTION    The reservation style as defined in the RSVP STYLE
               object in the RESV message [RSVP].

               ALLOWED VALUE TYPES:  PolicyBitStringValue,
                                     PolicyIntegerValue (Integer has
                                     an enumeration of
                                     { Fixed-Filter=1,
                                      Shared-Explicit=2,
                                      Wildcard-Filter=3}
```

DERIVED FROM QoSPolicyRSVPVariable (defined in this document)
ABSTRACT FALSE
PROPERTIES None

## 8.24. The Class "QoSPolicyIntServVariable"

This is a concrete class that contains the Integrated Service requested in the RSVP Reservation message, as defined in the FLOWSPEC RSVP Object [RSVP]. The class definition is as follows:

NAME QoSPolicyRSVPIntServVariable
DESCRIPTION The integrated Service requested in the RSVP Reservation message, as defined in the FLOWSPEC RSVP Object [RSVP].

ALLOWED VALUE TYPES: PolicyIntegerValue (An enumerated value of { CL=1 , GS=2, NULL=3}

DERIVED FROM QoSPolicyRSVPVariable (defined in this document)
ABSTRACT FALSE
PROPERTIES None

## 8.25. The Class "QoSPolicyRSVPMessageTypeVariable"

This is a concrete class that contains the RSVP message type, as defined in the RSVP message common header [RSVP] object. The class definition is as follows:

NAME QoSPolicyRSVPMessageTypeVariable
DESCRIPTION The RSVP message type, as defined in the RSVP message common header [RSVP] object.

ALLOWED VALUE TYPES: Integer (An enumerated value of {PATH=1 , PATHTEAR=2, RESV=3, RESVTEAR=4, RESVERR=5, CONF=6, PATHERR=7}

DERIVED FROM QoSPolicyRSVPVariable (defined in this document)
ABSTRACT FALSE
PROPERTIES None

## 8.26. The Class "QoSPolicyRSVPPreemptionPriorityVariable"

This is a concrete class that contains the RSVP reservation priority, as defined in [RFC3181] object. The class definition is as follows:

NAME QoSPolicyRSVPPreemptionPriorityVariable
DESCRIPTION The RSVP reservation priority as defined in [RFC3181].

```
                ALLOWED VALUE TYPES: PolicyIntegerValue

   DERIVED FROM  QoSPolicyRSVPVariable (defined in this document)
   ABSTRACT      FALSE
   PROPERTIES    None
```

8.27. The Class "QoSPolicyRSVPPreemptionDefPriorityVariable"

This is a concrete class that contains the RSVP reservation defending priority, as defined in [RFC3181] object. The class definition is as follows:

```
   NAME          QoSPolicyRSVPPreemptionDefPriorityVariable
   DESCRIPTION   The RSVP preemption reservation defending priority as
                 defined in [RFC3181].

                 ALLOWED VALUE TYPES: PolicyIntegerValue

   DERIVED FROM  QoSPolicyRSVPVariable (defined in this document)
   ABSTRACT      FALSE
   PROPERTIES    None
```

8.28. The Class "QoSPolicyRSVPUserVariable"

This is a concrete class that contains the ID of the user that initiated the flow as defined in the User Locator string in the Identity Policy Object [RFC3182]. The class definition is as follows:

```
   NAME          QoSPolicyRSVPUserVariable
   DESCRIPTION   The ID of the user that initiated the flow as defined
                 in the User Locator string in the Identity Policy
                 Object [RFC3182].

                 ALLOWED VALUE TYPES: QoSPolicyDNValue,
                                      PolicyStringValue,
                                      QoSPolicyAttributeValue

   DERIVED FROM  QoSPolicyRSVPVariable (defined in this document)
   ABSTRACT      FALSE
   PROPERTIES    None
```

8.29. The Class "QoSPolicyRSVPApplicationVariable"

This is a concrete class that contains the ID of the application that generated the flow as defined in the application locator string in the Application policy object [RFC2872]. The class definition is as follows:

```
NAME          QoSPolicyRSVPApplicationVariable
DESCRIPTION   The ID of the application that generated the flow as
              defined in the application locator string in the
              Application policy object [RFC2872].

              ALLOWED VALUE TYPES: QoSPolicyDNValue,
                                   PolicyStringValue,
                                   QoSPolicyAttributeValue

DERIVED FROM  QoSPolicyRSVPVariable (defined in this document)
ABSTRACT      FALSE
PROPERTIES    None
```

## 8.30. The Class "QoSPolicyRSVPAuthMethodVariable"

This is a concrete class that contains the type of authentication used in the Identity Policy Object [RFC3182]. The class definition is as follows:

```
NAME          QoSPolicyRSVPAuthMethodVariable
DESCRIPTION   The RSVP Authentication type used in the Identity
              Policy Object [RFC3182].

              ALLOWED VALUE TYPES: PolicyIntegerValue (An enumeration
                                   of { NONE=0, PLAIN-TEXT=1,
                                   DIGITAL-SIG = 2, KERBEROS_TKT=3,
                                   X509_V3_CERT=4, PGP_CERT=5}

DERIVED FROM  QoSPolicyRSVPVariable (defined in this document)
ABSTRACT      FALSE
PROPERTIES    None
```

## 8.31. The Class QoSPolicyDNValue

This class is used to represent a single or set of Distinguished Name [DNDEF] values, including wildcards. A Distinguished Name is a name that can be used as a key to retrieve an object from a directory service. This value can be used in comparison to reference values carried in RSVP policy objects, as specified in [RFC3182]. The class definition is as follows:

```
NAME           QoSPolicyDNValue
DERIVED FROM   PolicyValue
ABSTRACT       FALSE
PROPERTIES     qpDNList
```

### 8.31.1. The Property qpDNList

This attribute provides an unordered list of strings, each representing a Distinguished Name (DN) with wildcards. The format of a DN is defined in [DNDEF]. The asterisk character ("*") is used as wildcard for either a single attribute value or a wildcard for an RDN. The order of RDNs is significant. For example: A qpDNList attribute carrying the following value:

"CN=*, OU=Sales, O=Widget Inc., *, C=US" matches:

"CN=J. Smith, OU=Sales, O=Widget Inc, C=US"

and also matches:

"CN=J. Smith, OU=Sales, O=Widget Inc, L=CA, C=US".

The attribute is defined as follows:

| | |
|---|---|
| NAME | qpDNList |
| SYNTAX | List of Distinguished Names implemented as strings, each of which serves as a reference to another object. |

## 8.32. The Class QoSPolicyRSVPSimpleAction

This action controls the content of RSVP messages and the way RSVP requests are admitted. Depending on the value of its qpRSVPActionType property, this action directly translates into either a COPS Replace Decision or a COPS Stateless Decision, or both as defined in COPS for RSVP. Only variables that are subclasses of the QoSPolicyRSVPVariable are allowed to be associated with this action. The property definition is as follows:

| | |
|---|---|
| NAME | QoSPolicyRSVPSimpleAction |
| DESCRIPTION | This action controls the content of RSVP messages and the way RSVP requests are admitted. |
| DERIVED FROM | SimplePolicyAction (defined in [PCIMe]) |
| ABSTRACT | FALSE |
| PROPERTIES | qpRSVPActionType |

### 8.32.1. The Property qpRSVPActionType

This property is an enumerated integer denoting the type(s) of RSVP action. The value 'REPLACE' denotes a COPS Replace Decision action. The value 'STATELESS' denotes a COPS Stateless Decision action. The value REPLACEANDSTATELESS denotes both decision actions. Refer to [RFC2749] for details.

| | |
|---|---|
| NAME | qpRSVPActionType |
| DESCRIPTION | This property specifies whether the action type is for COPS Replace, Stateless, or both types of decisions. |
| SYNTAX | Integer |
| VALUE | This is an enumerated integer. A value of 0 specifies a COPS Replace decision. A value of 1 specifies a COPS Stateless Decision. A value of 2 specifies both COPS Replace and COPS Stateless decisions. |

## 9. Intellectual Property Rights Statement



## 10. Acknowledgements

The authors wish to thank the input of the participants of the Policy Framework working group, and especially the combined group of the PCIMe coauthors, Lee Rafalow, Andrea Westerinen, Ritu Chadha and Marcus Brunner. In addition, we'd like to acknowledge the valuable contribution from Ed Ellesson, Joel Halpern and Mircea Pana. Thank you all for your comments, critique, ideas and general contribution.

## 11. Security Considerations

The Policy Core Information Model [PCIM] describes the general security considerations related to the general core policy model. The extensions defined in this document do not introduce any additional considerations related to security.

## 12. References

### 12.1. Normative References

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

[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 Extensions", RFC 3460, January 2003.

### 12.2. Informative References

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

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

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

[RSVP] Braden, R., Ed., Zhang, L., Berson, S., Herzog, S. and S. Jamin, "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification", RFC 2205, September 1997.

[RFC2749] Herzog, S., Ed., Boyle, J., Cohen, R., Durham, D., Rajan, R. and A. Sastry, "COPS usage for RSVP", RFC 2749, January 2000.

[RFC3181] Herzog, S., "Signaled Preemption Priority Policy Element", RFC 3181, October 2001.

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

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

[CL] Wroclawski, J., "Specification of the Controlled-Load Network Element Service", RFC 2211, September 1997.

[RSVP-IS] Wroclawski, J., "The Use of RSVP with IETF Integrated Services", RFC 2210, September 1997.

[GS] Shenker, S., Partridge, C. and R. Guerin, "Specification of the Guaranteed Quality of Service", RFC 2212, September 1997.

[DCLASS] Bernet, Y., "Format of the RSVP DCLASS Object", RFC 2996, November 2000.

[RFC3182] Yadav, S., Yavatkar, R., Pabbati, R., Ford, P., Moore, T., Herzog, S. and R. Hess, "Identity Representation for RSVP", RFC 3182, October 2001.

[RFC2872] Bernet, Y. and R. Pabbati, "Application and Sub Application Identity Policy Element for Use with RSVP", RFC 2872, June 2000.

[DNDEF] Wahl, M., Kille, S. and T. Howes, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997.

## 13. Authors' Addresses

Yoram Ramberg
Cisco Systems
4 Maskit Street
Herzliya Pituach, Israel 46766

Phone: +972-9-970-0081
Fax: +972-9-970-0219
EMail: yramberg@cisco.com

Yoram Snir
Cisco Systems
300 East Tasman Drive
San Jose, CA 95134

Phone: +1 408-853-4053
Fax: +1 408 526-7864
EMail: ysnir@cisco.com

John Strassner
Intelliden Corporation
90 South Cascade Avenue
Colorado Springs, Colorado 80903

Phone: +1-719-785-0648
Fax: +1-719-785-0644
EMail: john.strassner@intelliden.com

Ron Cohen
Ntear LLC

Phone: +972-8-9402586
Fax: +972-9-9717798
EMail: ronc@lyciumnetworks.com

Bob Moore
IBM Corporation
P. O. Box 12195, BRQA/501/G206
3039 Cornwallis Rd.
Research Triangle Park, NC 27709-2195

Phone: +1 919-254-4436
Fax: +1 919-254-6243
EMail: remoore@us.ibm.com

## 14. Full Copyright Statement



## Acknowledgement

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