INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A POLICY-BASED MODEL FOR IP NETWORK MANAGEMENT IN SUPPORT OF QOS

by

ELIZA CLAUDIA CELENTI

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2002

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

UMI Number: 3063812

UMI®

UMI Microform 3063812

Copyright 2002 by ProQuest Information and Learning Company.
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

This manuscript has been read and accepted for the Graduate Faculty in computer science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

9/17/2002

Date

Chair of Examining Committee

9/17/2002

Date

Executive Officer

Prof. Christina M. D. Zamfirescu

Prof. Ted Brown

Prof. Miriam Tausner

Supervisory Committee

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Abstract

A POLICY-BASED MODEL FOR IP NETWORK MANAGEMENT IN SUPPORT OF QOS

By

ELIZA CLAUDIA CELENTI

Adviser: Professor Christina M.D. Zamfirescu

The rapid development of IP network technologies offers service providers the potential to greatly expand the range and customization of IP connectivity services. However, these technologies are diverse, complex and ever changing. If they are to be relied upon, they must be managed to ensure that a consistent and coherent service is reliably delivered.

The *Policy-Based Model for IP Network Management* (PBNM) is an effort to provide scalable management of heterogeneous multi-service networks. The model is characterized by the creation of a distinct policy "layer", separate from the data transmission layer, which allows administrators to manage services using policy rules. It parses and verifies configurations for semantic correctness and consistency, and automatically distributes policies to various network devices, while hiding (to whatever extent possible) translations of service level directives into device configuration commands.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In addition, the policy layer collaborates in aggregating device related information, coordinating management information across various sub-systems, and - in some cases - “closing the loop” between monitoring and control. To perform these roles, the policy layer includes a set of functions that provide configuration, management and control of the IP connectivity attributes. Such attributes include *Quality of Service (QoS)*, and routing policy.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ACKNOWLEDGEMENTS

A few years ago, my life turned completely up side down. First, it just looked very different. Then, I realized that it took a giant leap for the better. This could not have been possible if it had not been for ... My Faith and, of course, a number of people who acted as catalysts and eventually shortened a journey that was eventually going to happen anyway ....

For making my Ph.D. work as exciting and painless as possible, for their constant good advise, encouragement and support, my thanks should first go to my advisor, Professor Christina Zamfirescu and the former and current directors of the Graduate Program in CS at CUNY, Professors Stanley Habib and Ted Brown. I would also like to thank Professor Miriam Tausner for the challenging suggestions and support. There were also others, faculty at the Graduate Center, Hunter and Baruch Colleges, my AT&T Labs colleagues Pramila Mullan, Raju Rajan and Yzhak Ronen for stimulating my interest in this field and also my friends. I also need to thank my son Eric, for the inspiration he gave me even before he was born, my mother for her constant help and support, my father for inspiring my love for computers and -- last but not least -- my husband Dan for having the courage to play God with my life, being so good at anticipating (most of) my needs, and putting up with me all this time.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

TABLE OF CONTENTS

1 INTRODUCTION 1

2 PREMISES AND OBJECTIVES 8

2.1 Assumptions and Restrictions 8

2.2 Objectives and Motivation 9

2.2.1 State of the Art and Limitations *9*

2.2.2 Objectives/Goals *11*

3 QOS: THE DIFFERENTIATED SERVICES ARCHITECTURE OVERVIEW 13

3.1 What is QoS? 13

3.2 Why is QoS important? 13

3.3 How is QoS evaluated? 14

3.4 Congestion Avoidance 15

3.4.1 TCP Rate Control *16*

3.4.2 Random Early Detection *18*

3.4.3 Traffic Shaping Non-Adaptive Flows *18*

3.5 Diffserv QoS model 20

4 SERVICE LEVEL 28

4.1 Service Level Agreements 31

4.2 Quantitative vs. Qualitative Specifications 33

4.3 Service Level Specifications for Bilateral QoS Negotiation 35

4.3.1 The Negotiation Process *35*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.3.2 Service Level Specifications – Information Model 39

4.3.3 Terms and Usage 40

4.3.4 SLS Components 41

4.3.5 Schema 48

4.3.6 Use Case Scenario - VLL Implementation and Specifications 55

4.3.7 Security Considerations 59

5 MODELING THE SERVICE LEVEL SPECIFICATIONS - MAPPING QUALITY PARAMETERS INTO QOS MECHANISMS **60**

5.1 IP QoS – A SYSTEMATIC APPROACH 60

5.2 IMPACT OF QoS MECHANISMS ON THE TCP FLOWS 63

5.2.1 Committed Access Rate (CAR) Policing Qos mechanism 73

5.2.2 Weighted Fair Queuing (WFQ) QoS Mechanism 77

5.2.3 Weighted Random Early Discard (WRED) QoS Mechanism 83

5.2.4 The synergy of applying a combination of QoS mechanisms 96

5.2.5 Test results and future work 98

6 QOS LEVEL **99**

6.1 DETECTION AND RESOLUTION OF CONFLICTS AMONG RULES IN A POLICY BASED NETWORK MANAGEMENT SYSTEM 102

6.1.1 Conflict Detection Algorithms using Multidimensional Range Searching 104

FOR THE TIME PERIOD CONDITIONS ATTACHED TO A RULE, THE FOLLOWING SYNTAX VALIDATIONS MUST BE PERFORMED: 107

6.2 VALIDATION OF WELL FORMED SEQUENCES OF QoS MECHANISMS 128

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.2.1 Validation of Well-Formed Sequences of Actions – Algorithm Description *133*

6.2.2 Cycle Detection in Sequences of Actions – Algorithm Description *137*

7 TRANSLATING POLICY RULES INTO DEVICE SPECIFIC COMMANDS **139**

7.1.1 Levels of Policy *139*

7.1.2 Policy Functions *143*

7.1.3 Toolkit *150*

7.1.4 System Operation *155*

7.1.5 Common Information Model *156*

7.1.6 Use Case for User Requesting QoS Services *183*

8 CONCLUDING REMARKS **189**

9 FUTURE WORK **189**

10 APPENDICES **192**

10.1 CONFLICT DETECTION ALGORITHMS AND THEIR JAVA IMPLEMENTATIONS 192

10.2 VALIDATING WELL-FORMED SEQUENCES OF QOS MECHANISMS ALGORITHMS AND THEIR JAVA IMPLEMENTATIONS 211

11 BIBLIOGRAPHY **215**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

LIST OF ILLUSTRATIONS, CHARTS AND DIAGRAMS

Figure 1 Policy Based Network Management - Levels of Abstraction........................ 7

Figure 2: Managing the network over a single provider 29

Figure 3 : Example Architecture for Service Negotiation and Fulfillment................. 37

Figure 4: Example of the Service Negotiation Process... 38

Figure 5 : Lab setup for testing 64

Figure 6: Class of Service 2: Telnet traffic 65

Figure 7: Telnet script setup.. 66

Figure 8 : Class of Service 3: HTTP traffic 67

Figure 9 : HTTP script setup... 68

Figure 10 : Class of Service 4: FTP traffic.. 69

Figure 11 : FTP script setup.. 70

Figure 12: Throughput of generated traffic – no controls... 71

Figure 13 : Transaction rate of generated traffic... 72

Figure 14 : Response time of generated traffic 73

Figure 15 : Traffic behavior after applying CAR QoS mechanism 75

Figure 16 : Traffic behavior prior to applying WFQ mechanism 81

Figure 17 : Traffic behavior after applying WFQ mechanism.................................... 82

Figure 18 : Traffic generation without any QoS controls 87

Figure 19 : Applying WRED QoS mechanism with various drop probabilities......... 88

Figure 20 : Applying WRED QoS mechanism with different minimum and maximum threshold parameters 94

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 21: Impact of adjusting the drop probability on the traffic behavior............... 95

Figure 22: The synergy of applying combined QoS controls 97

Figure 23 : The Structure of a Policy Rule.. 101

Figure 24 : Rectangle intersection problem corresponding to bi-dimensional rules. 117

Figure 25 : The Structure of Sequences of Actions 129

Figure 26: Policy Action Sequence: Example 1.. 129

Figure 27 : Policy Action Sequence: Example 2.. 130

Figure 28: Well formed sequences of actions represented by paths in a directed graph 131

Figure 29 : One-Step and Multiple-Step Cycles in Sequences of Actions 132

Figure 30 : PBNM Functions 144

Figure 31 : Policy component view 154

Figure 32: Service Template Information Model.. 159

Figure 33 : Service Level Template supporting classes... 160

Figure 34: Network Level Template Class 168

Figure 35 : Role Level Template class.. 182

Figure 36 : Policy Action Classes 183

LIST OF TABLES

Table 1 (continued on the successive pages 173-179):.. 161

Table 2 (continued on the successive pages 182 and 183) :...................................... 169

Table 3 (continued on the successive pages 184-193) :... 172

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 1: Service Level Template Class **161**

Table 2 : Network Level Template Class **169**

Table 3 : Role Level Template Class **172**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ACRONYMS

ATM = Asynchronous Transfer Mode

CAR = Committed Access Rate

CBB = Common BackBone

CBWFQ = Class Based Weighted Fair Queuing

CLI = Command Line Interface

CMTS = Cable Modem Termination Equipment

COPS = Common Open Policy Service

DEN = Directory Enabled Network

DiffServ = Differentiated Services

DMTF = Distributed Management Task Force

DQoS = Dynamic Quality of Service

DOCSIS = Data Over Cable Service Interface Specification

DSCP = DiffServ Code Point

EF PHB = Expedited Forwarding Per Hop Behavior

Ethernet = A 10-Mbps, coaxial standard for LANs, initially developed by Xerox and later refined by Digital, Intel and Xerox (DIX). All nodes connect to the cable where they contend for access via CSMA/CD. Also slang for the coaxial cable that carries the standard.

Frame Relay = A faster form of packet switching that is accomplished with smaller packet size and less error checking

IETF = Internet Engineering Task Force

IP = Internet Protocol

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

IPSec = IP Security Protocol

LDAP = Lightweight Directory Access Protocol

MIB = Management Information Base

MPLS = Multiprotocol Label Switching

MTU = Maximum Transmission Unit

NTP = Network Time Protocol

PBN = Policy-Based Networking

PBNM = Policy-Based Network Management

PDP = Policy Decision Point

PEP = Policy Enforcement Point

PIB = Policy Information Base

PSC = Packet Stream Condition

PVC = Permanent Virtual Circuit

PQ = Priority Queuing

QoS = Quality of Service

RADIUS = Remote Authentication Dial-In User Service

RPSL = Routing Policy Specification Language

RSVP = (ReSerVation Protocol)

SAP = Service Access Point

SIP = Session Initiation Protocol

SIS = Session Initiation Specification

SLA = Service Level Agreements

SLO = Service Level Objects

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

SLS = Service Level Specification

STS = Service Template Specification

SNMP = Simple Network Management Protocol

SONET = a physical layer technology designed to provide a universal transmission and multiplexing scheme, with transmission rates in the gigabit per second range, and a sophisticated operations and management system. This technology is standardized by the American National Standards Institute (ANSI)

UDP = User Datagram Protocol

VLL = Virtual Leased Line

VOIP = Voice Over Internet Protocol

VPrN = Virtual Provisioned Network

VPrP = Virtual Provisioned Pipe

WFQ = Weighted Fair Queuing

WRED = Weighted Random Early Discard

XML = Extensible Markup Language

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1 Introduction

The task of administering IP (Internet Protocol) networks becomes increasingly difficult with their growing size and complexity. For Internet service providers, the challenge of operating a fast-growing high-speed IP backbone is compounded by the introduction of new technologies such as RSVP (ReSerVation Protocol), DiffServ (Differentiated Services Standard), MPLS (Multiprotocol Label Switching), and IPSec (IP Security Protocol), new access methods such as cable, wireless and satellite, as well as diversified service and product offerings within the network.

In general, *Quality of Service* (QoS) refers to the ability of a network to provide better service to selected network traffic over various underlying technologies such as frame relay, ATM, Ethernet, SONET, and IP-routed networks. QoS capabilities are fundamental components of an intelligent network. The industry is transitioning from a less sophisticated physical infrastructure to a more intelligent infrastructure. A major component of that intelligence is the ability of the network to provide different classes of services based on application and user requirements. QoS are the means to providing that differentiation. There are various ways to provide QoS guarantees and it is a matter of tradeoffs between overhead, strictness of guarantees, and efficient resource utilization, but in this work we exclusively focus on *IP QoS*, by using the *DiffServ* (*Diffserv* is a standard used to help solve the IP quality problem. It operates at Layer 3 – i.e. network layer - only and does not deal with lower layers. It relies on traffic conditioners sitting at the edge of the network to indicate each packet's

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

requirements. *Diffserv* uses the IP *TOS* -- type of service -- field, renamed "*the DS (DiffServ) byte*", to carry information about IP packet service requirements.)

Policy based network management (PBNM) is an effort to provide scalable management of heterogeneous multi-service networks. The approach is characterized by the creation of a distinct "policy layer", separate from the data transmission layer, which allows administrators to manage services, while hiding (to whatever extent possible) translations from service level directives to device configuration commands. In addition, the policy layer also collaborates in monitoring the network, aggregating device-state information, coordinating event notifications across various sub-systems, and in many cases, "closing the loop" between monitoring and control. Differentiated Services architecture is characterized by high scalability, flexibility and interoperability. The network architecture needed to support the services is defined in terms of functional blocks (policing, classification, marking, buffering and scheduling) and of their placement in the network.

The purpose of a policy system is to manage and control a network as a whole, so that network operations conform to the business goals of the organization that operates the network. Ultimately, achieving such control requires altering the behavior of the individual entities that comprise the network. One approach is to alter the behavior of these entities individually by using a centralized network management application. Iterating through a list of network entities, a management application achieves control

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of the network by manipulating the operational parameters of each network entity separately.

To effectively control a network, network management software must have explicit knowledge of the management interfaces of each entity it endeavors to control, as well as knowledge of the capabilities of each of these entities. As a result, network management software is often forced to manage only those features controlled by the management interfaces common to the majority of the entities in the network.

The research work presented in this document consists of: definig a lexicon for systematically describing the service level specifications, reducing the scope of the open problem regarding mapping quality parameters into QoS parameters, results regarding optimization of TCP traffic using QoS mechanisms, definition and implementation of algorithms for detection and resolution of policy conflicts, defining a grammar for validating the semantic of well-formed sequences of QoS mechanisms using graph theory, defining cycle detection algorithms for sequences of QoS mechanisms, and implementing a 4-tier architecture in order to translate policy rules into router specific commands.

The document is structured as follows: the first chapter states the premises and the objectives of this thesis, the state-of-art and the limitations in the field of interest (Network Management). The second chapter describes the context of the current work, the assumptions and characteristics of the approach to find a QoS solution. The

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

third chapter describes the research module that defines a lexicon (information system) for specifying service requests in a general, abstract manner. This module is needed for hiding the heterogeneous, multi-vendor nature of the network from the higher-level systems and processes. This capability formalizes the information needed to define a QoS request and it is expected to be able to match the input information into the information understood by the underlying networking equipment. It is necesary to homogenize concepts and values that are specific for various protocols and have different formats.

The next chapter, called "Modelling the service level specification - Mapping quality parameters into device specific commands" describes the research approach taken toward automating the translation of the service request into QoS parameters. This section enumerates the challenges of finding and formalizing the dependencies between quality parameters and the QoS mechanisms parameters and it offers lab test results giving guidelines in setting the parameters of specific QoS mechanisms (CAR, WRED, WFQ) as well as presenting the synergy of using them together. In addition, it provides experimental conclusions regarding specific considerations for optimising TCP traffic. It also outlines this complex problem and its very dynamic environment, delimitating its boundaries.

The next research module decribed in chapter 6 of this document represents a proof of concept demonstrator for detection and resolution of conflicts among policy rules

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

using multi-dimensional range searching and for semantic validation of sequences of QoS mechanisms.

The challenge for a policy-based network management system is to acquire, translate in a device-specific language and deploy *Policy Rules* into devices in a **conflict-free environment**. However, in an effort to define *Policy Rules*, the user might specify rules that imply two distinct subjects within the same rule. The result is either a rule that makes no sense among the existent rules, or one that leads to the development of conflicting device-specific rules. A policy conflict occurs when the conditions associated with two or more *Policy Rules* are simultaneously satisfied, but not all of the actions associated with the *Policy Rules* can be performed together. In other words, it occurs when the definition of the rules is ambiguous and when they do not define disjoint sets of conditions associated with actions.

The first part of this module (described in section 6.1.) is intended to be a proof-of-concept demonstrator for conflict detection among rules in the context of a Policy-based Network Management system, using multidimensional range searching techniques.

In addition, the sequences of actions associated with the rules represent mechanisms to be applied in a distributed fashion on the network. An ill-formed sequence of actions that violates at least some constraints on well-formedness, can create a deadlock on the network, with severe consequences and with causes almost impossible to trace or detect at the time when they are deployed into the network devices.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The aim of this work is to approach policy-based network management from a more challenging perspective, which is **deadlock prevention through conflict detection.** The programming language that was chosen for this task is *Java* to work on a PC workstation, operating under Windows NT.

The last part of this module (described in section 6.2) creates a specification of a language that can be used for both analysis and synthesis of well-formed sequences of QoS mechanisms. This can be viewed as a transducer between linguistic surface strings and the semantic representations of their meaning and it will also include loop detection in their structure for the purpose of deadlock avoidance.

The next module described in the chapter called "Translating policy rules into device specific commands" is concerned with the ingenious multi-layered and distributed architecture of this proof of concept model. It also specifies the information and the object model and the progess of implemetation of all functional modules in a top-down approach. This document ends with two chapters regarding concluding remarks and suggested future work.

The following figure shows the levels of abstraction implemented by this Policy Based Network Management model. The sections of this document also follow them.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

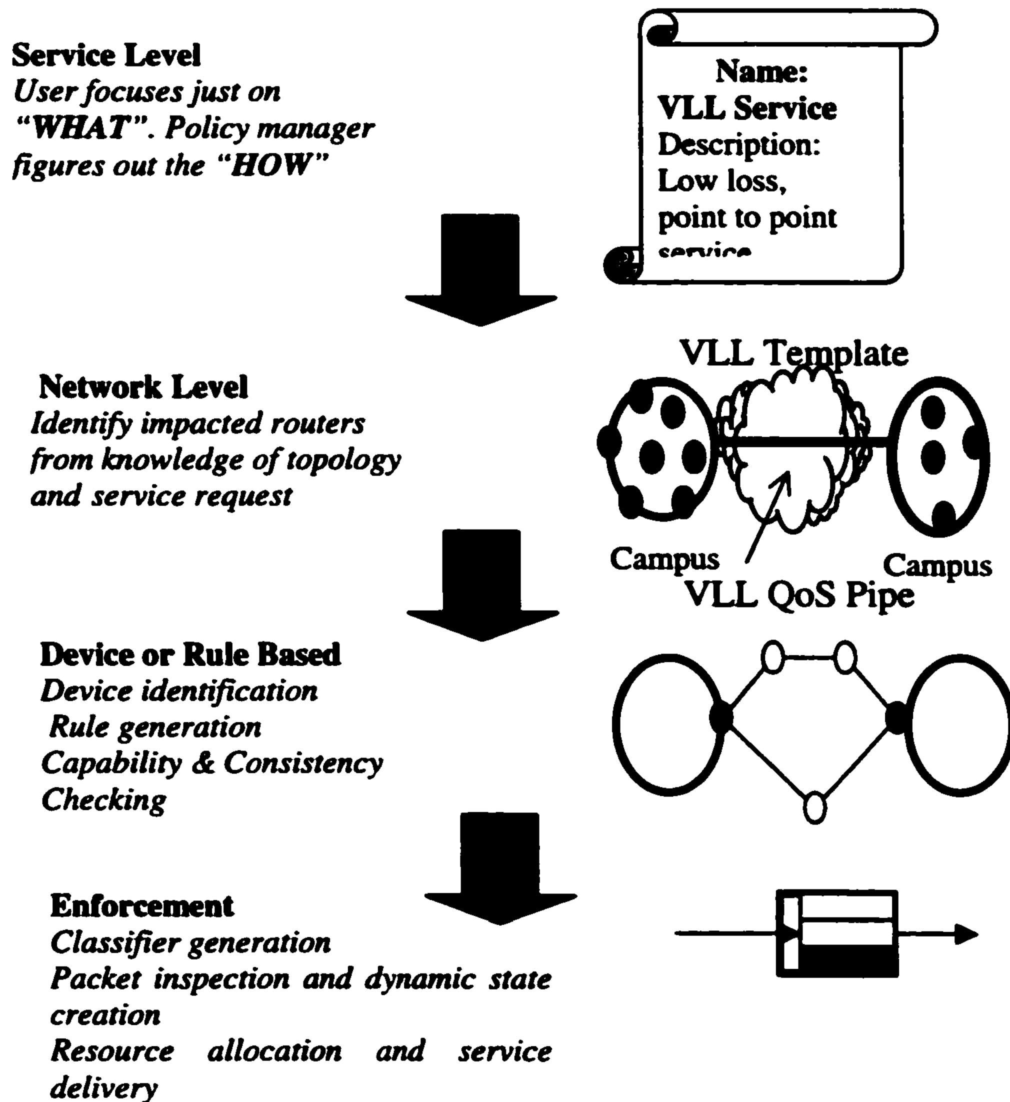


Figure 1 Policy Based Network Management - Levels of Abstraction

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2 Premises and Objectives

2.1 Assumptions and Restrictions

There are various ways to provide QoS guarantees and doing it is a matter of tradeoffs between overhead, strictness of guarantees, and efficient resource utilization. This proof of concept is exclusively focused on IP QoS within the framework of Differentiated Services model.

The project deals only with IP network (below Layer 3) and it is intended to be applied first to one provider's common backbone (CBB) whose facilities are non-ATM. Therefore, we will refer further in this document only at the architecture for a Layer 3 network with no attempt to map Layer 3 services into Layer 2 QoS mechanisms.

A motivation for introducing differentiated IP services into the router networks with less emphasis on ATM and other Layer 2 switches (considering that some traffic engineering has been provided already by ATM networks) can be revealed by the following comparison:

In an ATM network, allocating a certain amount of bandwidth for a specific virtual circuit (VC) can provide QoS. Traffic engineering is usually done by computing the routes offline and then downloading the configuration statically into the ATM switches on an hourly or daily basis. Per permanent virtual circuit (PVC) traffic statistics of the current configuration provide accurate traffic information for

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

computing the routes for the next configuration. The advantages of ATM networks over router networks **without** differentiated services or MPLS are:

- Per-PVC traffic statistics are available.
- QoS and some sort of traffic engineering are provided.
- ATM networks are currently faster in data forwarding.

Traditional disadvantages of ATM networks are:

- ATM cell header overhead is large.
- Routers must be used at the boundary of the network. With both switches and routers present in the network, two sets of configurations are required: one for routers and the other one for switches.

With differentiated IP service, router networks can also provide QoS and traffic engineering. This can be done without a large header overhead and two sets of configurations, i.e. eliminating the traditional disadvantages of ATM networks.

2.2 Objectives and Motivation

2.2.1 State of the Art and Limitations

Management information in a large network is usually distributed between the MIBs (Management Information Base) of network elements as well as widely disparate databases representing partial aspects of the network state such as configuration, performance, faults. Currently, an administrator uses his mental model of the whole system and device/domain specific tools (Telnet/CLI – Command Line Interface,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

SNMP – Simple Network Management Protocol) to perform consistent and coherent changes across the system. Only in very few cases are QoS techniques deployed on the devices automatically and usually the only validation that is made corresponds to syntax checking of router commands.

Furthermore, standards in this area are in their infancy. There has been a spate of efforts in a number of forums such as DEN (Directory Enabled Network), IETF (Internet Engineering Task Force) and DMTF (Distributed Management Task Force), focussed on standardizing protocols and schema for policy based management of networking devices. The emphasis has been solely on role level policies. Specifically, protocols such as COPS (Common Open Policy Service) and DIAMETER have been introduced in the IETF for communication between a networking device and the management system. COPS is for RSVP (standards track) and DiffServ (in progress). Diameter and Radius (Remote Authentication Dial-In User Service) protocols are for authentication and dial-up use. DQoS (Dynamic Quality of Service) standards in DOCSIS (Data Over Cable Service Interface Specification) uses SIP (Session Initiation Protocol) for initial signaling with COPS as a preferred protocol from policy server to CMTS (Cable Modem Termination Equipment). Other domains in which policy-based management efforts have been initiated have been in IPSec Policy and in Routing (RPSL - Routing Policy Specification Language). These are outside the scope of our current work.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.2.2 Objectives/Goals

Given the current context of management systems and practices, vendors of network management tools can only make incremental progress. The focus of their work is limited to policy-based element management in the campus/enterprise space. As a result, there are a lot of open problems and questions that remain unanswered. The aim of this work is to approach policy-based network management from a service provider perspective, corresponding to a more challenging task of having to offer quality services across the Internet, of which element management is only a small part. Some of the issues that we address in our research attempt are:

- Translation/mapping of quality parameters onto device specific attributes
- Implementation and control of quality parameters (e.g. delay, jitter, loss, etc)
- Definition of specific language for validation of Service Level Specifications
- Global conflict detection among policy rules using multi-dimensional range searching
- Validation of sequences of policy actions that represent QoS mechanisms. This include checking for well-formedness, one step loops and many-step loops in their definition
- A distributed, heterogeneous policy management application (4-tier architecture)

Challenges – General:

- Rationalize and automate network management practices
- Centralize administration while distributing monitoring and control
- Provide homogeneous service over heterogeneous network

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Challenges – Specific:

- Create a policy layer above the network layer
- Enable customers/users to manage their own VPN (Virtual Private Networks) via user-friendly interface
- Allow providers to administer network-wide provisioning and peering policies
- Create multiple levels of abstraction
- Map services to policies
- Implement a way to check creation, validation, distribution and enforcement of policies

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3 QoS: The Differentiated Services Architecture Overview

3.1 What is QoS?

Quality of Service is the term that describes the methods for introducing differentiated service in the networks. All applications used over the network are not created equal. Some applications need more predictable service than others do, as for instance interactive applications such as telnet. Some applications are more sensitive to delay or delay jitter than others are, as for instance telephony over the Internet. Some applications are very sensitive to packet loss, as for instance router configuration messages via ICMP. It is clear that we might get our network to work better if we somehow could organize all this instead of just sending everything into the network on a first come first served basis. QoS is the term used for this organization. By differentiating one type of traffic from another we could provide them with different service levels.

3.2 Why is QoS important?

Good functioning of the applications used on the company intranet is crucial for business. A company cannot afford badly functioning business-critical applications. Traffic on company intranets is increasing very fast. This is probably due to the introduction of a large amount of new applications in the market offering services such as videoconferencing, multimedia, and other bandwidth-hungry services. It may also be due to the increasing use of the Internet. Network equipment and links are costly resources in a corporate network. The solution for supporting higher amounts

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of traffic might not always be to increase the size of the pipe once the network gets congested. As we must deal with the network congestion in a cost-efficient way, this thesis explores an alternative solution. Apart from supporting the needs for the applications, there is also a policy point of view. Business policy might aim to give certain business-critical applications higher priority. Such a management facility is commonly called "controlled link-sharing".

3.3 How is QoS evaluated?

Packets in a flow from a sender to one or more receivers will be affected by network characteristics on the way. There are four very important characteristics of a packet flow; bandwidth, delay, jitter, and reliability, as described in [42].

Bandwidth is the maximal data transfer rate available to a flow between a sender and a receiver. The upper bound of the bandwidth available is the physical link capacity of the link with the lowest capacity on the path between the end-points in a simple topology. In more complex topologies several links could run in parallel between end-points and the upper bound on the bandwidth gets more complicated to calculate. Other flows may also be using parts of the path between the end-points, reducing the bandwidth available to the flow in question.

Delay is the time it takes for a packet to travel from a sender, through the network, to the receiver. Delay is due not only to transmission links, but is also increased by router holding time. If the packet has to be queued in the router it will experience higher delay.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Jitter is the variation of the delay. Mathematically it is the absolute value of the first derivative of the sequence of individual delay measurements.

Reliability measures the probability that the data arrives properly at the receiver. Errors can be introduced either on the physical link layer where a bit or a number of bits get changed during transmission, (bit-errors and burst-errors). Or, routing and protocol processing in the system can introduce degradation in reliability, as the order of the packets can be changed (packet reordering) or packets can be lost (packet loss).These are characteristics that are directly measurable and that can be modeled mathematically.

Now we have to remember that behind each flow is a transfer protocol and an application. Different types of protocols and applications behave differently when encountering the limitations described above. Taking the jitter parameter as an example: A user doing a file transfer would not experience any degradation in quality, although the TCP protocol might work a bit inefficiently. A user talking on the telephone over IP, on the other hand, would experience degradation in quality due to the loss of signal.

3.4 Congestion Avoidance

The standard way an IP-network is normally configured is for best-effort traffic. This means that no measures are taken to separate one type of traffic from another. All packets are competing on the same basis. Buffering is done at the intermediate

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

stations (e.g., routers) and packets are forwarded in a FIFO manner. Routing mechanisms try to make sure that the packet travels the best way through the network. It should be noted that there are fields in the IPv4 header to specify QoS levels in terms of delay, throughput, and reliability [43]. These are bits 3-5 in the TOS field and they can be used for specifying the delay to be normal or low; the throughput or the reliability to be normal or high. However, very few applications set these QoS parameters.

The best-effort scenario with FIFO forwarding works fine until the point when the network or a link gets congested due to traffic overload. At this point the routers will start to drop packets, since their queues are finite. This might lead to an even worse situation where applications try to retransmit data and possibly introduce an even larger load on the network. However, there are already some mechanisms that deal with this problem.

3.4.1 TCP Rate Control

The TCP protocol [44] has some basic mechanisms to avoid introducing more congestion to the network when losing packets. It is called TCP rate control.

The key parameter for this behavior is the congestion window (cwnd). This parameter defines the number of segments that a sender can transmit without receiving an acknowledgement (ACK) from the receiver. The first control function is the slow start of a TCP connection. When initializing a TCP connection the sender transmits only one segment. Each time the sender receives an ACK from the receiver, the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

congestion window (cwnd) is increased by one segment size. This effectively doubles the transmission rate for each round trip time (RTT) cycle. A TCP connection starts with an initial cwnd value of 1, and a single segment is sent into the network. The sender then awaits the reception of the matching ACK from the receiver. When received, the cwnd is increased from 1 to 2, allowing two packets to be sent. When each ACK is received from these two segments, the congestion window is incremented. The value of cwnd is then 4, allowing 4 packets to be sent, and so on.

The other mechanism is called congestion avoidance. In the event of packet loss, as signaled by the reception of duplicate ACK's, the value of cwnd is halved, and this value is saved as the threshold value to terminate the slow start algorithm (ssthresh). When cwnd exceeds this threshold value, the window is increased in a linear fashion, opening the window by one segment size in each RTTinterval. The value of cwnd is reduced to 1 when the end-to-end signaling collapses and the sender imes out waiting for ACK packets from the receiver. Since the value of cwnd is below the ssthreshvalue, TCP switches to slow start control mode, doubling the congestion window with every RTTinterval if the ACKs are getting back.

The intent of the algorithm is to reach a steady state where the sender injects a new segment into the network at the same rate at which the receiver accepts a segment from the network. The algorithm works fine for longer data flows such as a file transfer. Shorter flows, such as Web browsing via HTTP, are not likely to reach this point. The major disadvantage of this scheme is that when several TCP connections

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

are competing over a congested line, they all could experience packet loss at approximately the same time, which leads to what is called global synchronization. All flows decrease their transmission rates and invoke the TCP slow start mode.

3.4.2 *Random Early Detection*

RED [45] is a queue managing mechanism used for lowering the risk of global synchronization. The idea is to start dropping packets when a queue reaches a threshold value instead of waiting until it is full and drop the tail of it. Instead of dropping just any packets, RED selects randomly individual TCP flows and drops their packets. The result is that the sender of those flows invokes the TCP slow start mode. The advantage is that this does not happen to all flows at the same time, thus avoiding the global synchronization.

It should be noted that RED is a very simple queuing mechanism. It does not require much computational overhead as other queuing techniques do. With RED, it is simply a matter of deciding who gets into the queue in the first place - no packet reordering or queue management takes place. When packets are placed into the outbound queue, they are transmitted in the order in which they are queued.

3.4.3 *Traffic Shaping Non-Adaptive Flows*

TCP transport protocol is an adaptive protocol due to its use of feedback. As described above, the TCP sender slows down its transmission rate in case of congestion. However, there are applications using other protocols than TCP above IP. UDP [46] is the transport protocol used by many real-time multimedia applications. UDP is a connectionless transport protocol and is not concerned about recovering lost

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

packets or reordering packets. It transmits the data when received by the application and does not slow down because of packet loss. It is easy to imagine that this type of protocol can easily generate congestion in a network unless some precautions are taken. What could be done to those flows is to shape the rate of the traffic when it enters the network. This way the flow can be limited to a certain maximum bandwidth. Traffic shaping can also be used to decrease the jitter in the flow, by making sure to release the packets with the same time space in between them.

A decrease in jitter generates an increase in delay, since the shaping is realized by delaying. A very common model for doing traffic shaping is the leaky bucket model [47]. The user's offered load to the network is modeled as what is poured in to the bucket. The load that is accepted by the network is represented by what is coming out of the hole in the bottom of the bucket. If the user offers more loads than what is accepted by the network, he will start to fill up his bucket, which is synonymous with storing his data packets in a queue. When the bucket is full, it will flow over. This is represented by a full queue with packet drops as a consequence. The user can never exceed a certain transfer rate represented by the size of the hole in the bottom of the bucket. Another similar model is the token-bucket model [48]. In this model the bucket contains tokens that are produced at a fixed rate. For each packet or for a certain volume that a user offers to the network he must use a token. If he does not offer a load to the network, the bucket will fill up with tokens that are not used, until it is full. The full bucket represents the maximum burst size that the network will accept from the user. If the user tries to transfer data at a higher speed than tokens are

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

generated, the bucket will eventually be emptied. When there are no tokens to use for an offered packet, it is discarded.

The difference from the leaky bucket model is that the user is allowed to transfer at a speed higher than the transfer rate represented by the token generation speed, but the mean value of the accepted load will always be equal to or below this value.

3.5 Diffserv QoS model

The above mechanisms help us use the network efficiently, but they cannot differentiate certain traffic from another. Differentiation would be preferable, since different users or applications have different needs. The following paragraphs discuss the Diffserv QoS model for introducing different levels of service for network traffic.

3.5.1. Differentiated Services

There is an IETF Working Group called DiffServ [49] that is working with a differentiated services model on the Internet [50]. Differentiated services aims to give some traffic flows better service than others. To distinguish between flows, a way to mark packets with a priority is needed. The priority can be used either to let some traffic flows have precedence over others by reordering packets in the queue so that higher priority packets get sent first. Or, the priority could serve as a discard preference. A packet with a lower priority level would be discarded more easily in the event of congestion than a packet with higher priority.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The most common way to mark packets is via IP Precedence, as described in the initial Internet Protocol RFC [43]. A sub-field of the TOS (Type of Service) octet in the IPv4 header is used for this. The three leftmost bits in the TOS field are used, giving a possible maximum of **eight different priority levels. The priorities are set as 0 for lowest priority and 7 for highest.** The IETF Working Group proposes as an extension to use the complete TOS-field as priority field. They propose to call the field for the Differentiated Services (DS) field, [51]. Six bits are used for setting priority, giving a possible maximum of 64 different priority levels. The two leftover bits are reserved for future use. The idea of differentiated services is to set the priority level of the packet when it enters the network. A packet classifier does this. A packet can be classified as a certain priority by examining its source and destination IP address and/or by other information found in the packet header such as a TCP or UDP port number.

The network manager has to define a policy for what traffic should be classified as what priority level, then the core network has only to implement scheduling algorithms to queue and forward the packets to their destinations. It is especially important to apply those mechanisms at network bottlenecks, as it is in these areas where congestion is most likely to occur. The basic modification of the single level FIFO queuing algorithm to enable differentiated services is to divide traffic into a number of categories, and then provide resources to each category in accordance with a predetermined allocation structure, implementing some form of proportional resource allocation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A basic modification of the FIFO structure is to introduce **Priority Queues**. The idea is to create a number of distinct queues for each interface and associate a relative priority level with each one. Packets are scheduled from a particular priority queue in FIFO order only when all queues of a higher priority are empty. In such a model, *the highest priority traffic receives minimal delay, but all other priority levels may experience resource starvation if the highest precedence traffic queue remains occupied.*

The low priority traffic gets completely starved by the two higher priority traffic flows. This model is *simple to implement*, but to ensure that all traffic receives some level of service we need more sophisticated scheduling algorithms.

A more sophisticated method would be to classify each traffic flow as belonging to its own queue. Differentiating flows could do this by criteria such as source and destination address, source and destination port, Protocol ID, and TOS field. The router assigns each flow its own queue. It then applies its scheduling mechanism to these queues, so that the packets gets scheduled on a per flow basis. Each queue will be serviced in order to its relative weight. This approach is called **General Processor Sharing**, GPS. The equation for calculating the bandwidth received for each flow would be:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$\mathbf{BWFlow_i} = \frac{\mathbf{IPPrecFlow_i + 1}}{\mathbf{\sum_i (IPPrec_i + 1)}} * \mathbf{TotalBW}$$

$BWFlow_i$ = The share of the bandwidth allotted to the flow in question

$IPPrecFlow_i$ = The IP precedence set for the flow in question

TotalBW = The total bandwidth available on the link

A **weighted round-robin** scheduling algorithm can be used to service each queue. As an example let us consider four queues with priority 3, 1, 0, and 0. The weighted round robin schedule will send four packets from queue number one, two packets from queue number two and one packet each from queues number three and four. Then it will start over by sending packets from queue number one again.

When packets are equally large, this mechanism fairly shares the bandwidth between all flows on our link. When packet sizes vary, a flow with larger packets could occupy more bandwidth than a flow with smaller packets of the same priority. To avoid a situation like this, a deficit weighted round-robin algorithm can be used, which modifies the round robin algorithm to use a service quantum unit. This is also known as a bit-wise round-robin algorithm. A packet is scheduled from the head of a weighted queue only if the packet size minus the per-queue deficit counter is less than the weighted quantum value. The next packet in the queue is tested using a weighted

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

quantum value, which has been reduced by the size of the scheduled packet. When the test fails, the remaining weighted quantum size is added to the per-queue deficit counter and the scheduler moves to the next queue. This algorithm performs with an average allocation that corresponds to the relative weights of each queue on a bandwidth basis.

Another fair queuing algorithm with the possibility to weight packets is **Weighted Fair Queuing, WFQ**. WFQ introduces the concept of finishing time of a packet. Instead of serving each queue in around-robin fashion, it serves the packet with the smallest finishing time first. Finishing time is calculated as follows, if a flow is active (i.e., there are already packets in the queue for this flow.):

$FT(Pkt_{k+1}) = FT(Pkt_k) + Size(Pkt_{k+1})$*(transmission time in secs/byte)*(active flows)

Pkt_{k+1} = first packet in this queue

Pkt_k = last packet that got sent away from this queue.

$FT(Pkt_x)$ = calculated finish time for packet x.

Otherwise:

$FT(Pkt_0) = Now + Size(Pkt_0)$*(transmission time in secs/byte)*(active flows)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

I have normalized the **Size(Pktk+1)*(transmission time in secs/byte)*(active flows)** to the numbers between brackets to avoid a too complicated equation. Note that some smaller packets can be scheduled before others, although they arrived later.

This is not done by assigning more bandwidth to these flows. The mechanism only prevents smaller packets from getting stuck after large packets, as they could do in a round robin scheduling discipline. Also, the benefit for a low bit rate flow is that when a packet belonging to this flow arrives, the WFQ mechanism will schedule it quite soon, since the last packet in its flow was scheduled quite some time ago.

It has been proved by Parekh [52] that this algorithm provides an absolute upper bound on the network delay on multi hop networks.

Given that WFQ is used at every hop for a particular data flow and the traffic injection source conforms to certain token bucket model assumptions, it has been shown that the worst case queuing delay is bounded within a network. An important benefit of this is that WFQ can be used to provide strong guarantees for a given data flow within a heterogeneous network. Apparently there seem to be different definitions of WFQ - those who take priority into account and those who do not. The above equation does not take priority into account, but there is a Cisco implementation of WFQ [53] that does.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The Cisco WFQ model checks for the IP Precedence bits in the TOS field of the IP packet and assigns bandwidth according to the priority of the packet. A flow with higher precedence will receive more bandwidth than flows with lower precedence. To achieve this, the equation for the finishing time must be changed. Cisco implements this by the following equations: if a flow is active (i.e., there are already packets in the queue for this flow.):

FT(Pktk+1) = FT(Pktk) + Size(Pktk+1)*(transmission time in secs/byte)*4096/(Prec + 1)

Pkt_{k+1}= first packet in this queue.

Pkt_k = last packet that got sent away from this queue

FT(Pkt_x) = calculated finish time for packet x.

Prec = precedence set for the packet.

Otherwise:

FT(Pkt_0) = Now + Size(Pkt_0)*(transmission time in secs/byte)*4096/(Prec + 1)

Cisco puts the flows into a hash table, using the classification criteria described above. The value 4096 is the maximum number of entries in this hash table. This value can be set between 16 and 4096, where 256 is the default, to tune the functionality of the WFQ mechanism. The result of the differentiated service model is that some traffic gets treated better than other traffic. It should be remembered,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

though, that there are no guarantees for a certain service level. The share of the bandwidth received is still dependent on what other traffic is competing on the network.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4 Service Level

The high level goal of this thesis module is to enable the customer to define, validate, monitor, and manage network elements associated with managed connectivity services. This capability interacts with the underlying networking equipment, speaks the specific protocols and formats understood by each element, maps them into a common form, interacts with the directory and security components as appropriate. In general, this capability hides the heterogeneous, multi-vendor nature of the network from higher-level systems and processes.

A basic scenario of the service offer corresponds to the creation and management of Virtual Provisioned Networks (VPrN) over a single service provider QoS enabled backbone network. Customers have one or more sites attached to the backbone at service access points (SAP) – see Figure 1. Customer sites are assumed to have best-effort IP connectivity at all times. A customer may create a Virtual Private Network (VPrN) spanning multiple sites by creating on-demand SAP-to-SAP, SAP-to-site or site-to-site Virtual Provisioned Pipes (VPrP) at different QoS levels. In addition, the customer may choose to monitor the VPrN at different levels of granularity and frequency. Usage and performance information is fed back to the customer, to enable regulation or behavior modification based on network signals.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

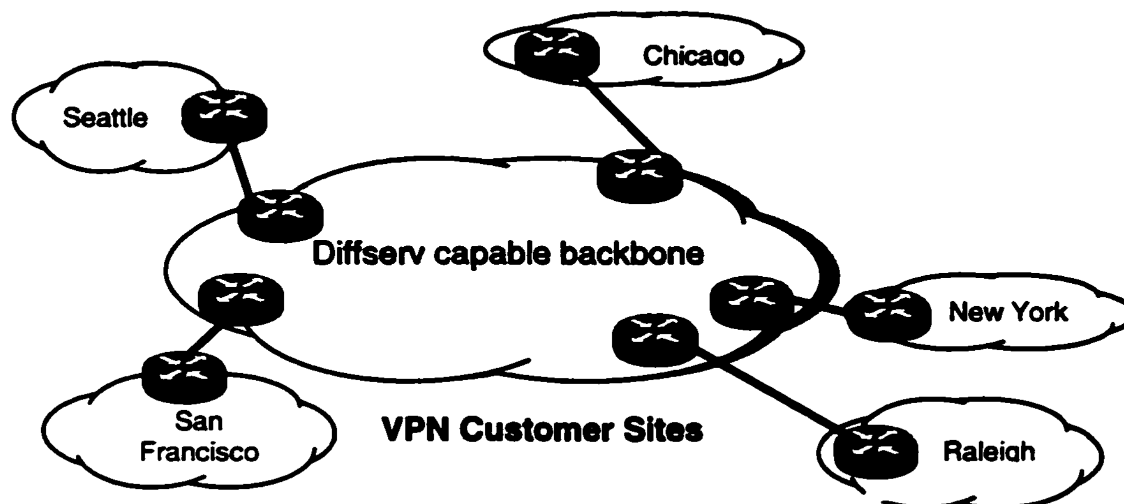


Figure 2: Managing the network over a single provider

Currently, the Internet delivers only one type of service, best effort, to all IP traffic. Differentiating packets and giving them different quality of service treatment across the IP network can be achieved – however - by marking the Differentiated Services byte of the IP packet header. As a result, a major component of the versatility afforded by QoS is the ability of the network to differentiated treatment of the traffic flows based on application and user requirements.

This section describes a structure for defining Service Level Specifications for QoS negotiation over IP networks. The high level schema described in this section is intended for use during the process of QoS negotiation between a customer entity and a provider entity. The purpose of this effort is to provide a vendor-independent lexicon and extensible information structure that can be used to describe services and instantiate them.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The challenge is motivated by the need to automate the creation and re-negotiation of QoS services, i.e., the need for customers to create, modify and manage QoS pipes or hoses across the provider's backbone on the fly. Irrespective of whether or not such negotiations are human initiated, network management systems must be capable of participation in negotiation, translation of negotiated agreements to device configurations, and monitoring the network for conformance. The emphasis should be more on the component of automation relating to the interaction of the customer and provider domains rather than on network configuration and monitoring within a domain.

This lexicon, named Service Level Specification (SLS), is intended to be a protocol independent representation of a set of technical parameters and their associated semantics that describe traffic flows and their corresponding service levels. The Service Level Specification is a language. The SLS is used to create specific documents called Service Level Objects (SLO). An SLO is a protocol dependent instantiation of a Service Level Specification, that is to say, it contains the parameters and their values that describe the transport service a specified flow is to receive over the transport domain. A Service Template Specification (STS) is an SLO sent from the provider to the customer describing the broad parameters of the service. A Service Instantiation Specification (SIS) is an SLO used by the customer to specify the exact levels of service required. The result of the negotiation process is a Service Level Agreement (SLA), i.e., a contractual document that a subscriber and a service provider have agreed upon [11].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.1 Service Level Agreements

At each Diffserv customer/provider boundary, the technical aspects of the service provided is defined in the form of a contract called Service Level Agreement (SLA) which specifies the overall features and performance that can be expected by the customer.

The SLA consists in essence of some service specifications that describe

- Quality parameters such as: expected throughput, drop probability, latency;
- Constraints on the ingress and egress points at which the service is provided, indicating the 'scope' of the service;
- Traffic profiles which must be adhered to for the requested service to be provided, such as token bucket parameters;
- Disposition of traffic submitted in excess of the specified profile;
- Actions to by applied to the service provided, i.e. shaping, marking, etc.

In addition to the above mentioned details, the SLS may specify more general service characteristics such as:

- availability/reliability, which may include behavior in the event of failures resulting in rerouting of traffic
- encryption services
- routing constraints
- authentication mechanisms
- mechanisms for monitoring and auditing the service

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- responsibilities such as location of the equipment and functionality, action if the contract is broken, support capabilities
- pricing and billing mechanisms

Currently, some data carriers offer a standard SLA consisting of the following items.

- Average monthly latency - for example, no more than 85ms roundtrip within provider's network at location A and of no more than 120ms between location B and provider's international gateway hub at location C.

  How is latency actually measured? Sometimes, data is collected from certain routers in the provider's backbone at certain intervals (in minutes.) The monthly latency value is the average of all these values. If, for two consecutive months, the average latency is greater than specified in the SLA, the provider could decide to credit the equivalent of one or more days of service fees.

- Packet loss - it provides a guarantee concerning the network-wide packet loss (for example, no more than 1% for a calendar month).

  For measuring the packet loss, data is collected by sending packets between city-pairs (round trip), then counting the packets that do not return. This statistic is updated every 30 minutes, then averaged to yield a monthly loss percentage.

- Network availability (network reliability) - as an example, the data carrier may guarantee 100% network availability (typically within the carrier's backbone).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

This may also include a time bound on when scheduled maintenance will occur and notification.

- Proactive network notification - guarantee of notification for a network outage within specified amount of time.

4.2 Quantitative vs. Qualitative Specifications

The Diffserv architecture can support a broad spectrum of different kinds of service. Categorizing these services provides some constraints on the corresponding service specifications that can be offered for the service.

Some services can be clearly categorized as qualitative or quantitative depending on the type of performance parameters offered.

Examples of *qualitative services* are as follows:

1. Traffic offered at service level A will be delivered with low latency.
2. Traffic offered at service level B will be delivered with low loss.

The assurances offered in the above examples are relative and can only be verified by comparison.

Examples of *quantitative services* are as follows:

1. 90% of the traffic delivered at service level C will experience no more than 50 msec latency.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2. 95% of the traffic delivered at service level D will be delivered.

The above examples both provide concrete guarantees that could be verified by suitable measurements on the example service irrespective of any other services offered in parallel with it.

There are also services which are not readily categorized as qualitative or quantitative as in the following examples:

1. Traffic offered at service level E will be allotted twice the bandwidth of traffic delivered at service level F.
2. Traffic with drop precedence DP1 has a higher probability of delivery than traffic with drop precedence DP2.

In example 1 above, the provider is quantifying the relative benefit of submitting traffic at service level E vs. service level F, but the customer cannot expect any particular quantifiable throughput. This situation can be described as a 'Relative Quantification Service'.

Determining how to monitor and audit the delivery of a qualitative or relative quantification service is nevertheless a challenging task.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.3 Service Level Specifications for Bilateral QoS Negotiation

4.3.1 The Negotiation Process

A corporate manager wishes to configure and manage corporate connectivity. The manager does this by connecting to a provider e-service web site and entering the security credentials necessary to authenticate her/him and provide access to the capabilities associated with her/his "management" role. The amount of information accessed depends on the set of credentials that were validated. Consequently, the view of a customer that uses the application might be different from what an administrator can access and from a provider's perspective.

After credentials validation, the customer is then able to configure and manage the connectivity, security, and QoS between the various corporate sites. This includes dedicated access for major locations and dial access for small locations and remote users. The MIS manager can add new sites, update the list of authorized users (based on automatic linkages to corporate HR databases or manually), and set policies controlling varying levels of access and quality of service. The QoS and levels of access are drawn from a set of options provided by the provider.

Why are multiple SLOs needed?

A simple model for the negotiation process, presented in Figure 2, is to consider independent policy servers in the customer and provider ([11]) networks that negotiate the attributes of service instantiation. (Naturally, "customer" and "provider" are roles played by negotiating entities in this process, and are not to be confused with

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

"end-users" or "carriers". For instance, two carriers negotiating peering agreements may each play the role of the other's customer.)

The customer declares its QoS requirements and the provider decides whether or not it can fulfill them. During this process (that corresponds to the service invocation defined in [11]), each entity takes responsibility for checking the correctness of the transaction, and for configuring and monitoring devices in its domain, say by translating negotiated parameters into policy rules to be evaluated by the Policy Decision Point (PDP) and hence distributing them to enforcement points (see [10] for policy concepts).

There are many ways that this transaction may be structured. As an illustration to help motivate the concepts in this document, we present one negotiation model in Figure 3. In this example, the provider first presents the attributes of the offered service to the customer in the form of a Service Template Specification (STS). The STS carries the provider-specified attributes of the service, instructs the customer about which attributes are customer specified, and lays some limitations on what values of the latter are acceptable. For instance, the provider may specify that the delay encountered by packets of a Gold service will not exceed 20 milliseconds, and restrict the bandwidth choices of the customer to a maximum of 5 Gigabytes/second.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

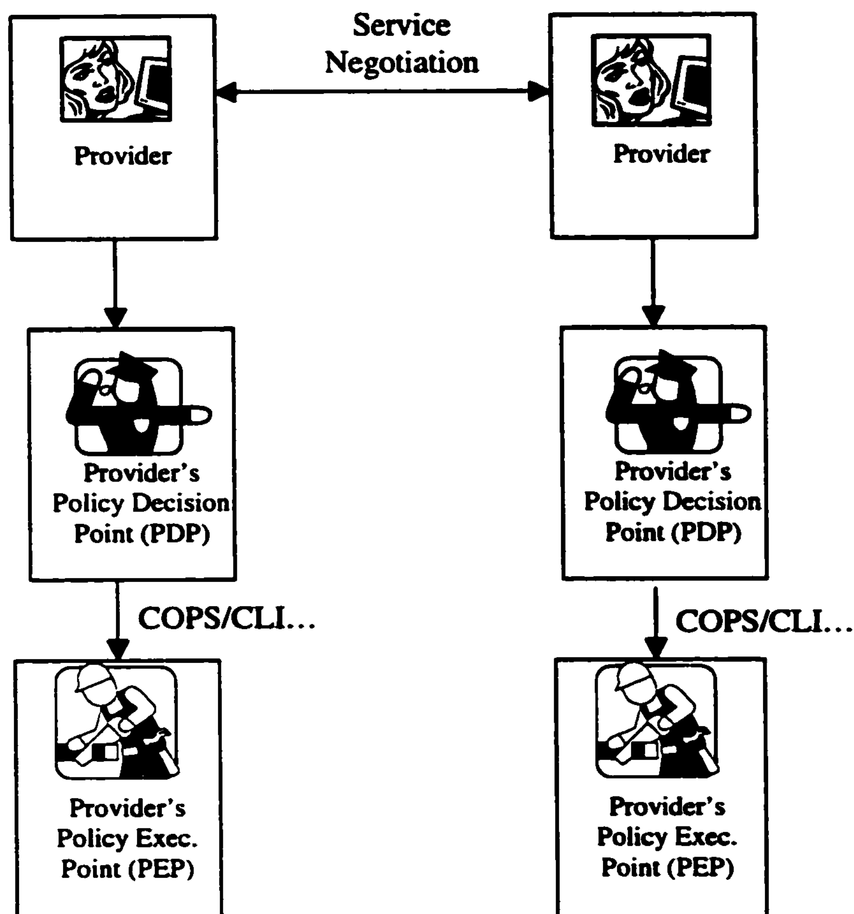


Figure 3 : Example Architecture for Service Negotiation and Fulfillment

The customer fills in the parameters needed to complete the service description and sends the request back to the provider as a Service Instance Specification (SIS.) For instance, the customer may declare in the SIS the need for 3 Mbps of bandwidth. The provider is now able to accept or to reject the customer's request. The provider may generate further updates about the state of the service or its performance.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

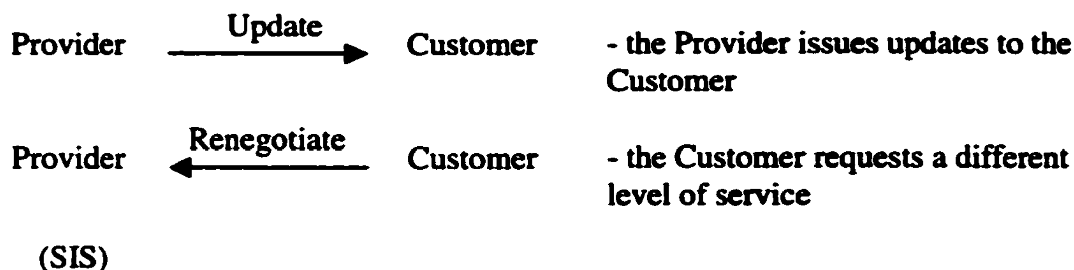


Figure 4: Example of the Service Negotiation Process

Needless to mention, the above process can be too complex or too simplistic depending on the negotiating entities and operational environment. For instance, a VOIP gateway configured to deliver high-quality service to a particular user may dispense with all but the SIS carrying the bandwidth desired. Another service may dispense with updates from the provider. Yet another service may not be re-negotiable. And so on.

Irrespective of complexity of the service negotiation process ([11]), there are a number of concepts, entities and attributes that can be used and reused in different aspects of the service negotiation process. The Service Level Specification (SLS) is intended as a lexicon that defines parameters, and also a schema that assigns semantics to valid combinations of these parameters. Information models and schema representing them are required for systematically understanding and documenting QoS service requirements and implementation, for automating the service negotiation process to whatever degree desired, and for making use of directories/databases to store persistent information about customer service subscription and network

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

configurations. The following sub-section presents an approach toward structuring the information needed for defining Service Level Specifications for QoS negotiation.

4.3.2 Service Level Specifications – Information Model

The design of the SLS in this section is based on the following principles:

1. The SLS should allow different instantiations in different languages. For instance, an XML representation would be useful for a web-based transaction, while COPS PIBs may be ideal if the negotiating entities are themselves PDPs.

2. The SLS should allow service negotiations at different levels of complexity, and be friendly to as many different negotiating protocols as possible. Examples of such protocols are HTTP, COPS, Diameter and RSVP.

3. The SLS should NOT standardize services. Vendors and customers should. Inasmuch as there are standard services such as those based on the Assured Forwarding PHB [7] or the Expedited Forwarding PHB [8], the SLS should be capable of representing them. The SLS should be extensible to allow vendors to define their own unique offerings.

4. The SLS should allow a simple STS or SIS to be represented simply, while being up to the task of representing a STS or SIS based on complex semantics.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.3.3 Terms and Usage

A. DiffServ SLS

This work is exclusively focused on IP QoS enabled services on the IP network, by building on the DiffServ model as a starting point. As a result, it only discusses requirements for a Layer 3 network only, without any attempt to map Layer 3 Services to Layer 2 QoS mechanisms. This does not lose extensibility or applicability.

B. STS and SIS

The SLS provides a unified syntax, semantics and schema for use in the STS, SIS and other aspects of service negotiation, re-negotiation and reporting.

C. Customer and Provider

Customer - a customer is the negotiating entity that offers data to the provider at one or more service access points (SAPs) or end-points. I assumed the existence of communication channels between customer and provider. The term "customer" refers to a role in the process, and is not restricted to end-users.

Provider - a provider is a data carrier who is capable of offering different levels of QoS across its network. The provider assumes responsibility to transport data packets belonging to the customer according to the SIS which both the customer and the provider agreed on previously.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

D. Tagging

The parameters and semantic units of the SLS are meant for use in the STS, SIS or other aspects of the negotiation. Depending on the service, each parameter may have a pre-defined value or configurable or unused. Even if the parameter is configurable, there may be additional constraints such as the data format, units used and its set of acceptable values. In order to facilitate such flexibility, each attribute or unit of the SLS may be optionally tagged with any or all of the following fields –

i) Specification: CUSTOMER_SPECIFIED or PROVIDER_SPECIFIED; used to describe who supplies the value for the attribute during negotiation.

ii) Type: Describes the format of the field, whether it is a string, integer or real.

iii) Description: Reserved for text describing the syntax or semantics of the field for ease of use. This would be particularly relevant to negotiations that are partially automated, for instance, on the provider side alone.

iv) Values Acceptable: Range or sequence of values that is allowed for this field.

4.3.4 SLS Components

The information contained in each SLS is structured into the following description units and sub-units:

- Common Unit
    - Customer/Provider/Service instance descriptors
        - Validity sub-unit
- Topology Unit
        - SAP sub-unit
        - Graph sub-unit

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- QoS Unit
    - Scope sub-unit
    - Traffic descriptor sub-unit
    - Load descriptor sub-unit
    - Qos parameters sub-unit
- Monitoring Unit
    - Scope sub-unit
    - Reporting parameters sub-unit

4.3.4.1 The Common Unit

The Common Unit contains information describing the general terms of the service offering. There is at most one Common Unit defined per SLS. The common unit contains fields that identify the provider, customer, service type and time. (These identifiers could contain names as well as signatures for non-repudiability). Further, the common unit contains a Validity sub-unit that identifies the period of applicability of the SLA.

4.3.4.2 The Topology Unit

The Topology Unit describes the number and nature of the end points that a service instance can have, and the relationship of traffic generation and consumption amongst them. The Topology unit is composed of one SAP sub-unit, and one or more Graph sub-units.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.3.4.2.1 SAP Sub-Unit

The SAP sub-unit is a list of end-points that are used in constructing the topology. The reason for keeping this list separate is that end-point descriptions may be complex -- using IP addresses or provider/customer specific attributes. There is little need to repeat these descriptions in every topology unit or in the QoS unit. Using the SAP sub-unit allows us to assign document specific serial numbers to each end-point, which can be re-used wherever needed.

4.3.4.2.2 Graph Sub-unit

When used in describing a topology, an end-point may be a source, destination or both. A source end point generates but does not consume traffic, while a destination consumes traffic without generating it. The graph sub-unit comprises of a list of sources and destinations, with special semantics that describes their inter-relationship. Further, we allow one or more Graph sub-units to be included in a single Topology unit. In order to better understand the design consider that service topologies can range from the simplest (single source or destination) to highly complex (several multiple-source to multiple- destination hoses.)

Some sample topologies are:

A "hose" with one source and any destination

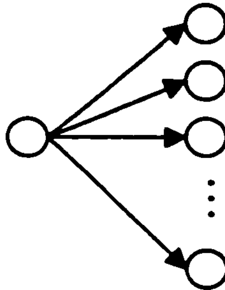
A "hose" with one destination and any source

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

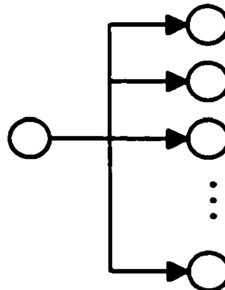
A bi-directional "hose" with one end point generating and receiving traffic from any other end point.

A uni-directional "pipe" between two end-points: a source and a destination

A uni-directional "pipe" between two end-points: a source and a destination



A point-to-multipoint "pipe" with one source and multiple destinations.



A point-to-multipoint "funnel" where the traffic generated by the source is arbitrarily distributed among the end points.

Given the variety of scenarios that need to be represented in a general form, I used the following principles in modeling topologies:

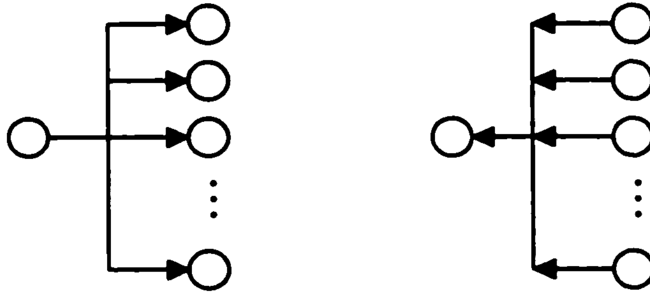
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

a) Allow different types of basic topologies to be represented in a unit.

b) Allow multiple instances of basic topology units that can be super-posed to represent complex topologies. The three basic topology types that I suggest using are the following:

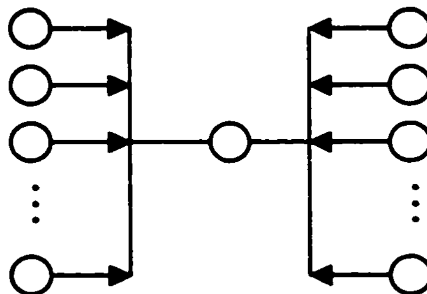
PIPE

FUNNEL



OR

HOSE:



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The end-points of the pipe, funnel or hose can be specific IP addresses, or the wildcard ANY that specifies that traffic going to any host is part of the topology. Note that the funnel topology is more general than pipe, and that the hose is more general than the first two. The reason for specifying them separately is to allow the right model for the task.

4.3.4.3 The QoS Unit

QoS Unit is a complex description block used to describe the traffic streams that are the subject of the SLA, and the nature and extent of service differentiation provided to them. The QoS unit describes quantitative or qualitative levels of service to some or all parts of the topology unit. A number of attributes and concepts in this unit may be shared with the Policy Working Group specifications [1]. In designing the schema it is important to be aware of the following issues:

a) Not all QoS parameters are used in describing every service. For instance, many services do not specify jitter.

b) QoS parameters may be pre-specified by the provider (in the STS) or be configurable by the customer (in the SLS).

c) QoS parameters may apply to all the traffic described by the topology unit (for example, a maximum delay for all traffic), or be specific to one Graph sub-unit (mean delay for a point-to-point pipe), or be specific to one end-point within a Graph sub-unit (leaky bucket descriptor for traffic offered by one source).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Keeping in view the above issues, the QoS unit was designed with three components:

Scope: This unit describes the topology unit, graph sub-unit or end-point to which this QoS unit applies. We provide for the general case where the QoS unit has parameters that apply to the entire topology, as well as more specific cases. If there are multiple QoS units, the more specific instances override the more general ones.

Traffic Descriptor: This is used to identify the packet streams (within the scope) that are subject to the SLA. In particular, the traffic descriptor includes the port numbers and protocol number that is subject to agreement.

Load Descriptor: This describes the quantity of offered traffic described through a leaky bucket (for instance) that is considered within profile, as well as the treatment of excess or out-of-profile traffic.

QoS Parameters: These include delay, jitter and loss parameters that describe the characteristics of the packet or flow transport to be provided under the agreement.

4.3.4.4 The Monitoring Unit

The Monitoring Unit has a structure similar to the QoS Unit and it defines a set of parameters that need to be collected and reported back to the customer in order to be compared with the Service Level Agreement (SLA) ones. This specification of the Monitoring unit is preliminary, and is included here as a placeholder.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.3.5 Schema

This section represents a detailed description of the previously mentioned entities, their attributes and their sources.

A brief word on the notation below. The name of the field or unit is followed in square parentheses [] by the cardinality and an indication on whether this particular attribute is provider specified (PS) or customer specified (CS) or either (PS/CS).

The Common Unit – systematically describes the customer and service ID as well as its validity period.

4.3.5.1 Topology Unit [0..1:PS/CS]

The customer, subject to restrictions placed by the provider usually specifies the topology. The topology unit consists of one SAP sub-unit and one or more Graph sub-units.

4.3.5.1.1 SAP Sub-unit

[1:PS/CS] has one occurrence in a Topology Unit. It consists of:

- Number Of SAP Items [1:PS/CS] A field denoting the cardinality of the list.

- SAP Item [1..N:PS/CS] Identifies a SAP end-point. Each Sap Item further consists of

- Serial Number [1: PS/CS] non-negative integer uniquely assigned to this end point for identification within this transaction. Serial numbers must be unique within the Sap Item. Serial numbers 0 and 1 are reserved as explained below.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- SAP Identifier [1: PS/CS] String used to identify the service end-point. Two reserved names that have special semantics are:

+ ANY, a name that refers to any service end-point; this end point uses Serial Number 0

+ THIS, a term used to refer to the service end-point that can be identified from context (for instance from the source IP address of the entity originating the SIS). This end-point uses Serial Number 1.

The SAP Identifier may be of different types. For instance, it can be an IP address of the customer interface attached to the provider network; a layer 2 identifier; or a string unique within the provider-customer context. The type is carried in the optional TAG.

4.3.5.1.2 Graph Sub-unit [0..N:PS/CS]

- The tag "Type" is used to describe the basic topological structure chosen to represent the relationship among end points. Allowed values are PIPE, FUNNEL and HOSE. PIPE implies that the number of sources and destinations must each be 1. A FUNNEL implies that either the number of sources or the number of destinations must be 1. A HOSE has no such restriction.

- Graph Identifier [1: PS/CS] Non-negative integer that uniquely identifies this instance of a graph sub-unit within the transaction, for reference from other units. The value 0 is reserved for reference to ALL the Graph sub-units.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- Number of Sources [1: PS/CS] Positive integer that specifies the number of sources for a particular topology unit description.

- Number of Destinations [1:PS/CS] Positive integer that specifies the number of sources for a particular topology unit description

- Source Item [1..N:PS/CS] Non-negative integer that refers to a end-point serial number in the SAP sub-unit.

- Destination Item [1..N:PS/CS] Non-negative integer that refers to a end-point serial number in the SAP sub-unit.

4.3.5.2 QoS Unit [1..N:PS/CS]

4.3.5.2.1 Scope Sub-Unit [1:PS/CS]

Specifies the scope of this QoS unit.

-Graph Identifier : Positive integer that refers to one of the Graph sub-units in the topology. If the Graph Identifier is 0, then the specifications are global and apply to all the end-points and graph sub-units. A larger Graph Identifier over-rides a smaller one.

-SAP Identifier [0..N:PS/CS]: Non-negative integer that refers to one of the end-points used in the Graph Identifier given above. This field cannot be used if the Graph Identifier is 0.

4.3.5.2.2 Traffic Descriptor Sub-Unit [1:PS/CS]

Describes the packet streams for which the QoS Unit attributes are defined. This sub-unit has considerable overlap with the Packet Stream Condition specified by the Policy Working Group in [1].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Its attributes are:

-DSCP [0..1:PS/CS] is a string defining the diffServCodePoint associated with the flow defined by the scope sub-unit.

-sourcePort [0..N:PS/CS] Define a set of port numbers that the traffic originates from. It is expressed as a set of integers.

-destinationPort [0..N:PS/CS] Specify a set of port numbers that the traffic is destined to. It is expressed as a set of integers.

-protocol [0..1:PS/CS] An integer that defines the protocol number that the traffic uses.

-Layer2Specification [0..1:PS/CS]

4.3.5.2.3 Load Descriptor Sub-Unit [0..N:PS/CS]

Describes the offered or received conformant traffic, as well as the actions to be taken for the out-of-profile traffic. It consists of the following attributes:

- DescriptorType [1:PS/CS] is a string from an enumerated list that describes the policing mechanism used to control the traffic. Currently, we define one value - LEAKYBUCKET used to indicate bucket policing.
- MTUSize [0..1:PS/CS] integer specifying the Maximum Transmission Unit size
- MeanRate [0..1:PS/CS] is a positive number that describes the average rate of arrival of conformant traffic as measured over the time interval. It applies to Leaky Bucket mechanism.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- TimeInterval [0..1:PS/CS] is a positive integer that defines the time interval over which the mean rate measurement is valid. It is specific to Leaky Bucket mechanism.
- BurstSize [0..1:PS/CS] is a positive integer that describes the tolerance to burst over the time intervals. It is specific to Leaky Bucket mechanism.
- ExcessTrafficTreatment [1:PS/CS] is a string from an enumerated list that specifies the actions to be applied to the out-of-profile traffic. Its value can be DROP if excess traffic is to be dropped, REMARK:XXX if excess traffic is to be treated as a different priority class with DSCP XXX, RESHAPE if excess traffic is to be reshaped to the submitted load descriptor.

If REMARK is chosen as the excess traffic treatment, another nested load descriptor sub-unit may be defined for the remarked traffic. It will include the new DSCP and the new set of attributes for that traffic to conform with.

4.3.5.2.4 QoS Parameters Sub-Unit [1:PS/CS]

Describes the quality parameters to be provided for the traffic described by the scope sub-unit. The exact definition of each of the following terms and the semantics of their combinations is yet to be decided.

- Delay Descriptor [0..1:PS/CS] specifies the delay imparted to conformant packets by the network. It can be specified by:

-delayPriority[0..1:PS/CS] is a string specifying the relative delay priority of this traffic.

-maxDelay [0..1:PS/CS] is a positive integer specifying in milliseconds an upper bound on the delay seen by each conformant packet of the flow

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

-maxRTT [0..1:PS/CS] is a positive integer specifying in milliseconds the maximum round trip time for the flow, taking the forward and reverse paths into account

-Percentile [0..1:PS/CS] is a positive integer less than 100 defining the percentile associated with maxDelay or maxRTT. It describes the portion of packets that will meet the standard delay.

- Loss Descriptor [0..1:PS/CS] describes the packet loss characteristics of the conformant traffic:

- meanLoss [0..1:PS/CS] is a positive integer that describes the average loss suffered by packets of a conformant flow. This attribute is used for quantitative description of the service performance.

- lossPriority [0..1:PS/CS] is a string that describes the relative experience of loss between different flow aggregates. A smaller loss priority implies a higher drop rate. This attribute is used for qualitative description of the service performance.

- Jitter Descriptor [0..1:PS/CS] describes the jitter that will be experienced by a conformant flow

-maxJitter [0..1:PS/CS] is a positive integer that describes
the maximum jitter that will be experienced by conformant packets.

-meanJitter [0..1:PS/CS] is a positive integer that describes the average jitter seen by a conformant flow. It is used for quantitative description of the service performance.

-JitterPriority [0..1:PS/CS] is a string that describes the relative experience of jitter between different flow aggregates. It is used for qualitative description of the service performance.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

-Percentile [0..1:PS/CS] is a positive integer less than 100 that specifies the percentile of conformant packets for which the maxJitter and meanJitter parameters apply.

- Service Availability Descriptor [0..1:PS/CS] Describes service down-times and exceptional conditions during which service terms are not applicable.

-Maximum Down-Time [0..1:PS] is a positive number that indicates the maximum time that the service will be down.

-Measurment Interval [0..1:PS] is a positive number that indicates the interval over which the Maximum Down-Time is measured.

-linkFailureTolerance [0..1:PS/CS] is a positive number that indicates the number of simultaneous link failures against which this service instance should be protected.

-nodeFailureTolerance [0..1:PS/CS] is a positive number that indicates the number of simultaneous node failures against which this service instance should be protected.

4.3.5.3 Monitoring Unit [1..N:PS/CS]

Describes the quality parameters that are to be reported for a topology unit, graph sub-unit or end-point. It is structured similarly to the QoS Unit, consisting of a scope sub-unit and quality parameters specifications included in the Reporting Parameters sub-unit.

4.3.5.3.1 Scope Sub-Unit [1:PS/CS]

Specifies the scope of this Monitoring unit.

-Graph Identifier : Positive integer that refers to one of the Graph sub-units in the topology. If the Graph Identifier is 0, then the specifications are global and apply to

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

all the end-points and graph sub-units. A larger Graph Identifier over-rides a smaller one.

-SAP Identifier [0..N:PS/CS]: Non-negative integer that refers to one of the end-points used in the Graph Identifier given above. This field cannot be used if the Graph Identifier is 0.

4.3.5.3.2 Reporting Parameters Sub-Unit

-Delay Frequency [0..1:PS/CS] is a positive integer that describes the frequency of delay updates.

-Jitter Frequency [0..1:PS/CS] is a positive integer that describes the frequency of jitter updates.

-Loss Frequency [0..1:PS/CS] is a positive integer that describes the frequency of loss updates.

4.3.6 Use Case Scenario - VLL Implementation and Specifications

This sub-section describes the implementation and the specifications needed for a Virtual Leased Line service, as an example.

4.3.6.1 VLL features and Implementation

The Virtual Leased Line (VLL)/ Gold service is targeted for applications and customers that require predictable point-to-point performance. A virtual leased line is a point-to-point pipe with a guaranteed peak transmission rate. No packets are lost due to network congestion; delay and jitter are very low. Appropriate applications include Voice over IP, transaction processing, and multimedia applications that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

require low queuing delay and jitter. VLL offers a simple, well-understood performance model (approximating leased line performance), and so should be appealing for any application, including web applications, where predictable performance is highly valued.

The reserved rate may be renegotiated, and so vary over time. Packets arriving at a rate exceeding R are either dropped or buffered at the point of origination of the flow. Furthermore, no packet of a flow is dropped due to congestion (losses, however, may occur due to other factors).

The traffic is classified and marked with DSCP 100110 on the customer premises. The provider polices incoming traffic and drops out of profile packets.

4.3.6.2 Specifications

A sample Service Instance Specification for this service is as below.

A. Common Unit

- Service name: Virtual Leased Line

- Provider Identifier: XXX

- Customer Identifier: YYY

- Time Stamp : 010120001352

- Validity Sub-Unit

    - DateFrom :11/01/2000

    - DateTo: 11/01/2001

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- TimeFrom: 9:00
- TimeTo: 17:00

B. Topology Unit

B1. SAP Sub-Unit

- Number of SAP Items :2 #(only point to point pipes are supported for VLL)
  - SAP Item 1
    - Serial number: 1
    - SAP Identifier: 130.16.120.51
  - SAP Item 2
    - Serial number: 2
    - SAP Identifier: 125.15.136.20

B2. Graph Sub-Unit

- Type : PIPE
- Graph Identifier: 1
- Number of Sources: 1 #(unidirectional flow)
- Number of Destinations: 1
- Source Item: 1 #(Source-SAP serial number)
- Destination Item: 2 #(Destination-SAP serial number)

C. QoS Unit

C1. Scope Sub-Unit

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- Graph Identifier: 1 #End-point is left unspecified

C2. Traffic Descriptor Sub-Unit

- DSCP 10010

C3. Load Descriptor Sub-Unit

- DescriptorType: Leaky Bucket
- MeanRate: 5000 #The units tag specifies kbps, say.
- BurstSize: 1MTU #provider specified and ingress interface specific
- PeakRate: 10000 #Provider Specified
- ExcessTrafficTreatment: DROP #Provider Specified

C4. QoS Parameters Sub-Unit

-Delay Descriptor

-maxDelay: 50 ms #provider specified

-Percentile: 90 #provider specified

-Loss Descriptor

-lossPriority: 3 #Low loss, provider specified

-Availability Descriptor

-maxDownTime: 1 #provider specified in units of hours

-measurement Interval: 1 #(provider specified in units of years

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.3.7 Security Considerations

Given the nature of information included in a SLS, some security related procedures are required prior to the initiation of any negotiation process.

Both parties involved in a negotiation process (the customer and the provider of services) should be identified for non-reputability of the transaction.

Independent of the service offer resolution (whether the service request was accepted or rejected) the confidentiality of exchanged information should be preserved using encryption.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5 Modeling the Service Level Specifications - Mapping Quality Parameters Into QoS Mechanisms

5.1 IP QoS – A Systematic approach

The scope of this chapter is defining an approach toward a methodology for offering a completely automated QoS solution. This goal of this module is to create an expert ruler and to sort out decisions about assigning applications into classes and setting up QoS mechanisms and their corresponding parameters in an automated fashion, based on the customer requirements. It can only be finalized in future work, given the complexity of all the factors involved.

The knowledge and the experience of offering QoS are still in their infancy and consequently, the aim of this chapter is not to exhaustively sort out QoS mechanisms and their exact behavior, but to raise questions about this multi-faceted problem and to make suggestions toward a possible solution.

Deploying QoS on the network should be approached systematically. Before rolling out an extensive set of QoS policies, one needs to identify all the details regarding the network the applications running on it, like direction of traffic, patterns of each application (for example if it always terminates at the same server), the size of the packets, if the traffic is delay sensitive, number of packets per second that the application generates per active session, number of actions typically active simultaneously, burstiness of the traffic, frequency of congestion, and the reasons

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

behind it. Prior to deploying QoS, a network analyst must identify the mission critical traffic, and assess which applications suffer from congestion or latency based performance issues. Just because a particular traffic is the most prolific on the WAN doesn't mean that it's the most important.

Also, it is important to determine if the traffic is differentiated based on specific users or groups or user.

QoS is deployed primarily at the LAN-WAN boundary, but it can also be applied in a LAN environment to ensure that localized LAN congestion does not impact traffic. It is necessary only on network segments that suffer from high delay because of serialization or congestion.

Also, a special consideration must be given to TCP-based traffic, which tries to decrease the window size to avoid oversubscribing, while UDP/IP traffic simply blasts away.

TCP is adaptive, rate based, and connection-oriented. It behaves politely when oversubscribed. TCP applications run as fast as they can, but gracefully back down when faced with congestion. UDP applications often don't have feedback, because the traffic is sent in one direction. When faced with congestion, the packets don't back off – frames are just dropped, degrading the quality of what is received. Thus, existing business traffic may yield to ill-mannered multimedia traffic. Section

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.2 of this document elaborates on the questions and problems raised by this feature of the TCP traffic when applying QoS.

The network analysis and diagnostic (including the need to offer QoS) with its complex characteristics and variables is not an automated process yet and it represents in itself an open research problem.

Another area in the Policy Based QoS systems that is not currently explored is translating quality parameters (as delay, jitter, loss) into QoS parameters (bandwidth percentage, number of queues for each router, normal burst size, excess burst size, etc.)

For example, voice traffic needs less than 30 milliseconds of jitter and about 150 milliseconds of latency. This type of traffic can be identified, marked and constrained along the way, but there is no exact formula for the parameters that express the best those quality parameters in terms of queuing or policing parameters, link speed, etc. Not even the exact path between the end points of traffic is known, as routers function based on Per Hop Behavior, deciding at each step where to go forward.

When QoS is deployed, it should give the desired behavior from end-to-end, which requires complex, cross-network configuration updates. In addition, optimizing one flow can have a negative impact to an unacceptable level on another application.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Even when QoS is used to improve network transmission, without any exact quantification of the improvement, complex decisions must be taken. There exist a variety of QoS schemes and parameters today. Each QoS scheme has its own terminology and tuning peculiarities, which makes the process hard to manage automatically. Many network devices and applications are potentially involved. Mismatches in device setup can occur at any of them. The large "cross-product" of potential problems makes setup particularly error-prone.

Policy-based network management today is in its infancy. Hardware and software from different manufacturers don't necessarily interoperate well. It's hard to predict the effects of policies, which result in QoS configuration changes. The next section describes the considerations that must be taken when using QoS mechanisms for TCP flows and the last section presents some test results and the formulae derived for setting specific QoS mechanisms' parameters.

5.2 Impact of QoS mechanisms on the TCP flows

Several recent studies [54, 55, 56] suggest that it is difficult to guarantee requested throughput for TCP flows, because of the flow and congestion control mechanism used by TCP which results in bursty traffic. This section gives an overview of the dynamics of TCP flows and discusses the nature of associated challenges.

For evaluating the impact of QoS mechanisms on the TCP flows, Chariot traffic generator application was used to send TCP flows over an uncongested link.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The following figure shows the network configuration, the QoS mechanisms applied and their locations.

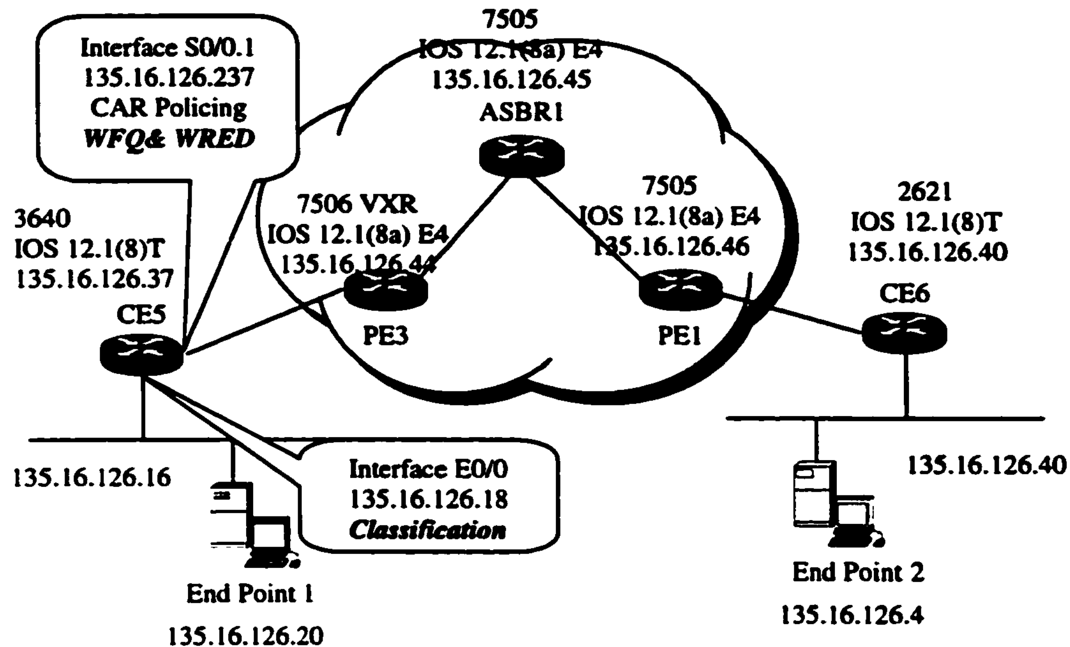


Figure 5 : Lab setup for testing

The traffic was generated between the End Point 1 (135.16.126.20) to the End Point 2 (135.16.126.4) using Ganymede software. It contained three main applications: telnet, http and ftp traffic (all using TCP protocol).

The telnet application was associated with the class of service 2, http application was assigned to class of service 3 and ftp traffic was mapped to class of service 4. Class of service 1 is usually reserved for real time applications and that is the reason why it was not used here as a generic class.

The traffic generated for telnet application had the following features:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 6: Class of Service 2: Telnet traffic

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Script Editor - Telnet.scr

File Edit Insert Help

Telnet

Line	Endpoint 1	Endpoint 2
2	time = initial_delay (0)	
3	CONNECT_INITIATE	CONNECT_ACCEPT
4	port = source_port (23)	port = destination_port (23)
5	LOOP	LOOP
6	count = number_of_timing_records (500)	count = number_of_timing_records (500)
7	START_TIMER	
8	LOOP	LOOP
9	count = transactions_per_record (50)	count = transactions_per_record (50)
10	SEND	RECEIVE
11	size = size_of_record_to_send (50)	size = size_of_record_to_send (50)
12	buffer = receive_buffer_size (DEFAULT)	buffer = receive_buffer_size (DEFAULT)

Variable Name	Current Value	Default Value	Comment
number_of_timing_records	500	50	How many timing records to generate
transactions_per_record	50	50	Transactions per timing record
size_of_record_to_send	50	1	Amount of data to be sent
receive_buffer_size	DEFAULT	DEFAULT	How many bytes of data in each RECEIVE
delay_before_responding	0	0	Milliseconds to wait before responding
user_delay	0	0	Pause before answering
transaction_delay	0	0	Milliseconds to pause
send_datatype	trans.cmp	trans.cmp	What type of data to send
send_data_rate	UNLIMITED	UNLIMITED	How fast to send data
destination_port	23	AUTO	What port to use for Endpoint 2
close_type	Reset	Reset	How connections are terminated

***Figure 7*:** Telnet script setup

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The traffic generated for http traffic had the following features:

Edit an Endpoint Pair

Pair comment: COS3

Endpoint 1 to Endpoint 2

Endpoint 1 network address

Endpoint 2 network address

135.16.126.4

Network protocol

TCP

Service quality

Edit This Script

HTTPtext.scr

HTTP Text Transfer

Select Script

Cancel

Help

Figure 8 : Class of Service 3: HTTP traffic

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Script Editor HTTPtext.scr

File Edit Insert Help

HTTP Text Transfer

Line	Endpoint 1	Endpoint 2
2	time = initial_delay (0)	
3	LOOP	LOOP
4	count = number_of_timing_records (500)	count = number_of_timing_records (500)
5	START_TIMER	
6	LOOP	LOOP
7	count = transactions_per_record (10)	count = transactions_per_record (10)
8	CONNECT_INITIATE	CONNECT_ACCEPT
9	port = source_port (80)	port = destination_port (AUTO)
10	SEND	RECEIVE
11	size = size_of_record_to_send (300)	size = size_of_record_to_send (300)
12	buffer = size_of_record_to_send (300)	buffer = size_of_record_to_send (300)

Variable Name	Current Value	Default Value	Comment
initial_delay			
number_of_timing_records	500	50	How many timing records to generate
transactions_per_record	10	10	Transactions per timing record
size_of_record_to_send	300	300	Amount of data to be sent
file_size	1000	1000	How many bytes in the transferred file
send_buffer_size	DEFAULT	DEFAULT	How many bytes of data in each SEND
receive_buffer_size	DEFAULT	DEFAULT	How many bytes of data in each RECEIVE
delay_before_responding	0	0	Milliseconds to wait before responding
transaction_delay	0	0	Milliseconds to pause
send_datatype	news.cmp	news.cmp	What type of data to send
control_datatype	trans.cmp	trans.cmp	What type of control data to send
send_data_rate	UNLIMITED	UNLIMITED	How fast to send data

Figure 9 : HTTP script setup

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The traffic generated to simulate the ftp application had the following characteristics:

Edit an Endpoint Pair

Pair comment: COS4

Endpoint 1 to Endpoint 2

Endpoint 1 network address

Endpoint 2 network address

135.16.126.4

Network protocol: TCP

Service quality

Edit This Script | FTPput.scr

Select Script | FTP Put

Cancel | Help

Figure 10 : Class of Service 4: FTP traffic

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Script Editor - FTPput.scr

File Edit Insert Help

FTP Put

Line	Endpoint 1	Endpoint 2
2	time = initial_delay (0)	
3	LOOP	LOOP
4	count = number_of_repetitions (1)	count = number_of_repetitions (1)
5	CONNECT_INITIATE	CONNECT_ACCEPT
6	port = source_port (21)	port = destination_port (21)
7	RECEIVE	SEND
8	size = login_size (15)	size = login_size (15)
9	buffer = control_buffer_size (DEFAULT)	buffer = control_buffer_size (DEFAULT)
10		type = control_datatype (trans.cmp)
11		rate = send_data_rate (UNLIMITED)
12	SLEEP	

Variable Name	Current Value	Default Value	Comment
number_of_repetitions	1	1	How many times to repeat the script
number_of_timing_records	500	100	How many timing records to generate
transactions_per_record	1	1	Transactions per timing record
size_of_record_to_send	100000	100000	Amount of data to be sent
user_delay	0	0	Pause before answering
transaction_delay	0	0	Milliseconds to pause
delay_before_responding	0	0	Milliseconds to wait before responding
file_control_size	30	30	How many bytes are in the control flows
login_size	15	15	How many bytes are in the login flows
control_buffer_size	DEFAULT	DEFAULT	Buffer size for control flows
send_buffer_size	4096	4096	How many bytes of data in each SEND

Figure 11* : *FTP script setup

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The following figures shows the behavior of all generated applications in terms of used bandwidth (throughput), transaction rate and response time prior to applying any QoS controls.

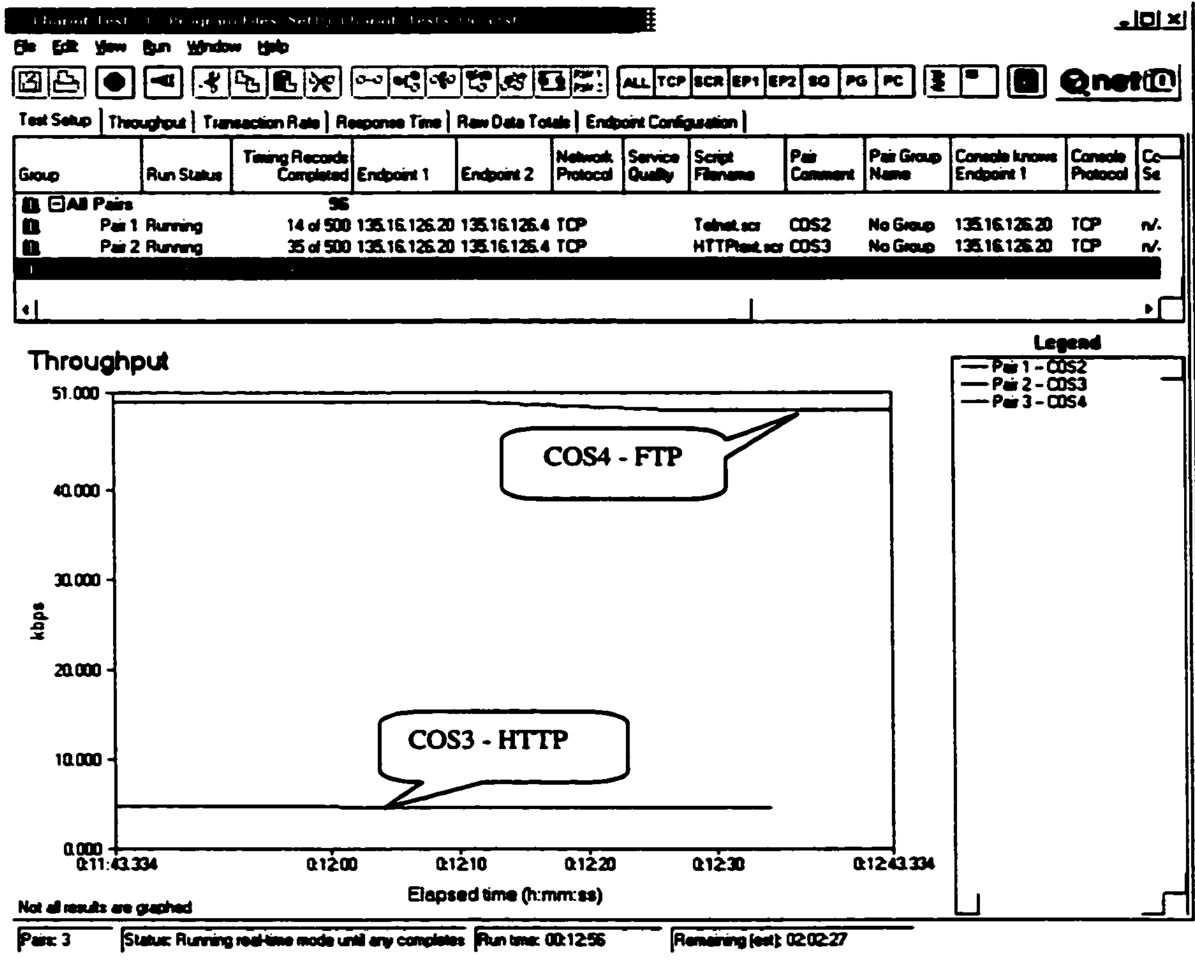


Figure 12*: *Throughput of generated traffic – no controls

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

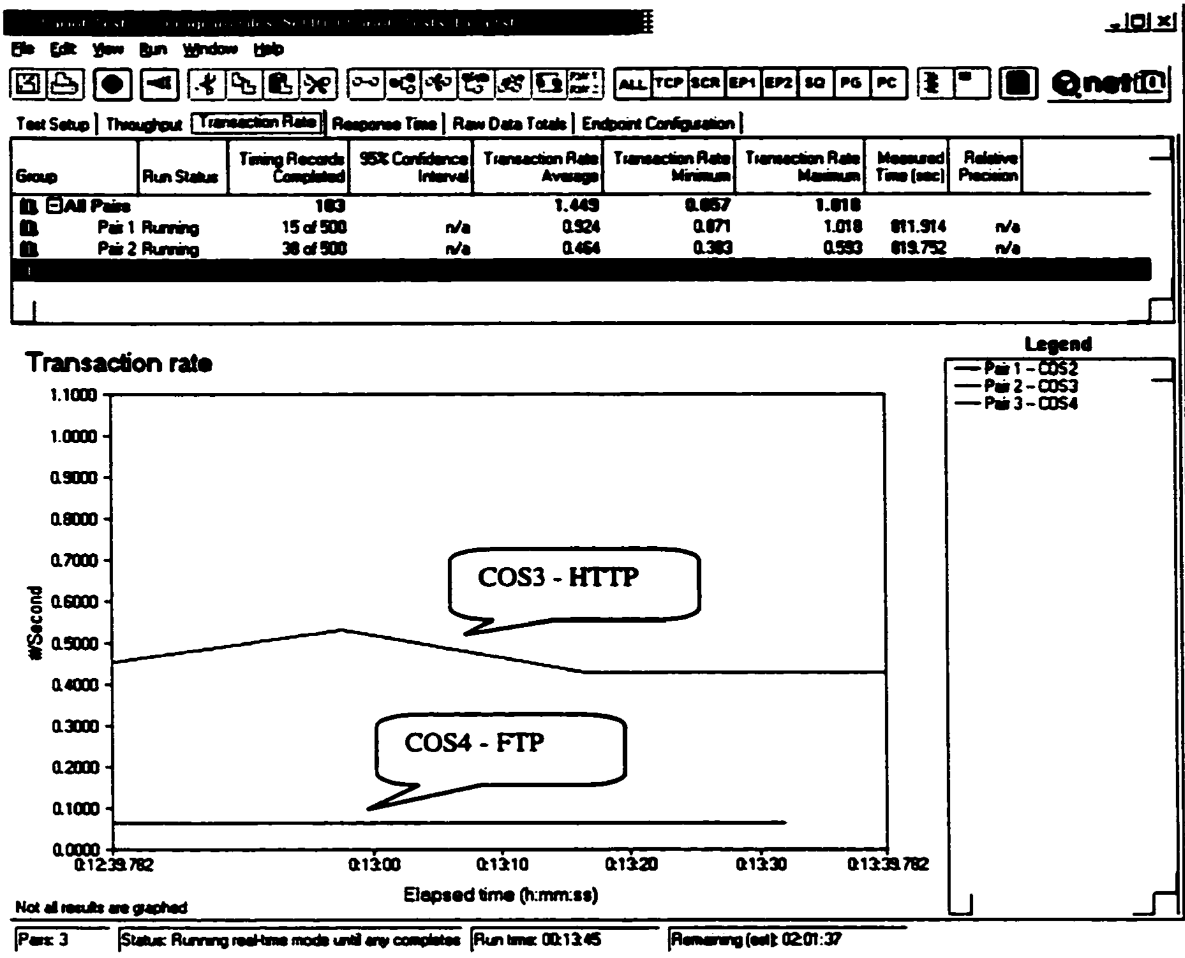


Figure 13* : *Transaction rate of generated traffic

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

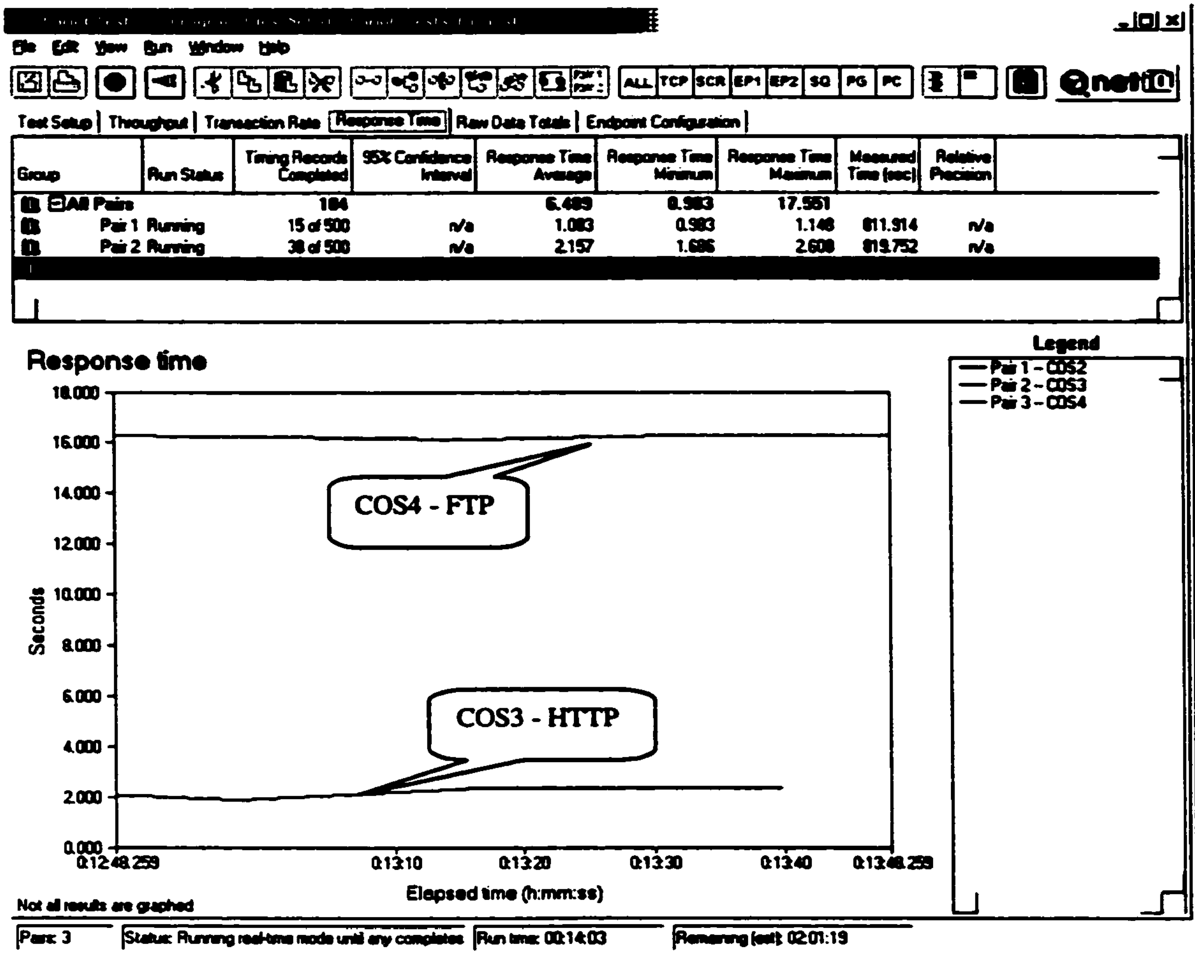


Figure 14* : *Response time of generated traffic

5.2.1 Committed Access Rate (CAR) Policing Qos mechanism

CAR policing Qos mechanism was applied at the egress interface of the first router encountered.

As stated above, TCP reacts when dropped packets are detected. If the transmitter recognizes a lost packet, it falls into either the congestion control phase or the slow start phase, depending on the number of contiguous packets lost. This behavior dramatically affects TCP performance. The first experiment was designed to illustrate

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

this impact, by demonstrating the behavior of a relatively small TCP transfer (8 MB), with different transmission rates in combination with CAR. Increasing the normal burst size results in a higher short-term bandwidth, until CAR's exceeding policy comes into play.

This experiment demonstrated effectively the behavior of TCP's slow-start feature. As soon as the token bucket is empty, the router starts dropping packets and often drops consecutive packets. The TCP transmitter recognizes that consecutive packets are lost and reacts to this by shrinking the congestion window to two, which has an enormous short-term impact on the actual throughput.

From this, one can conclude that exceeding the actual rate limit causes TCP to decrease performance drastically. Transmitting packets faster than the QoS rate limit has a negative impact on overall performance. Note that TCP might submit a whole socket buffer in one burst as permitted by the offered receiver window and the congestion window. The actual CAR configuration must be able to handle those bursts without dropping packets. Currently the normal burst size is limited to 2 MB. This limit has the side effect of shrinking the maximum supported window size to 2 MB. Cisco has mentioned that future versions of CAR might be able to handle deeper token buckets.

Besides the short-term behavior of TCP streams, it is important to analyze a stream with a longer duration. For that reason several long-term TCP sessions were

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

monitored. In this case, when the TCP session exceeding its rate limit, the normally constant throughput starts oscillating. As soon as CAR starts dropping packets, the transmitter reacts with TCP's slow-start feature. This reduces the transmission rate drastically. As the router's token bucket begins filled again, the transmitter is able to exceed its limits again (if enforced by the application) and hence resumes its oscillating behavior.

The following figure shows the behavior of the traffic after applying CAR Policing QoS mechanism.

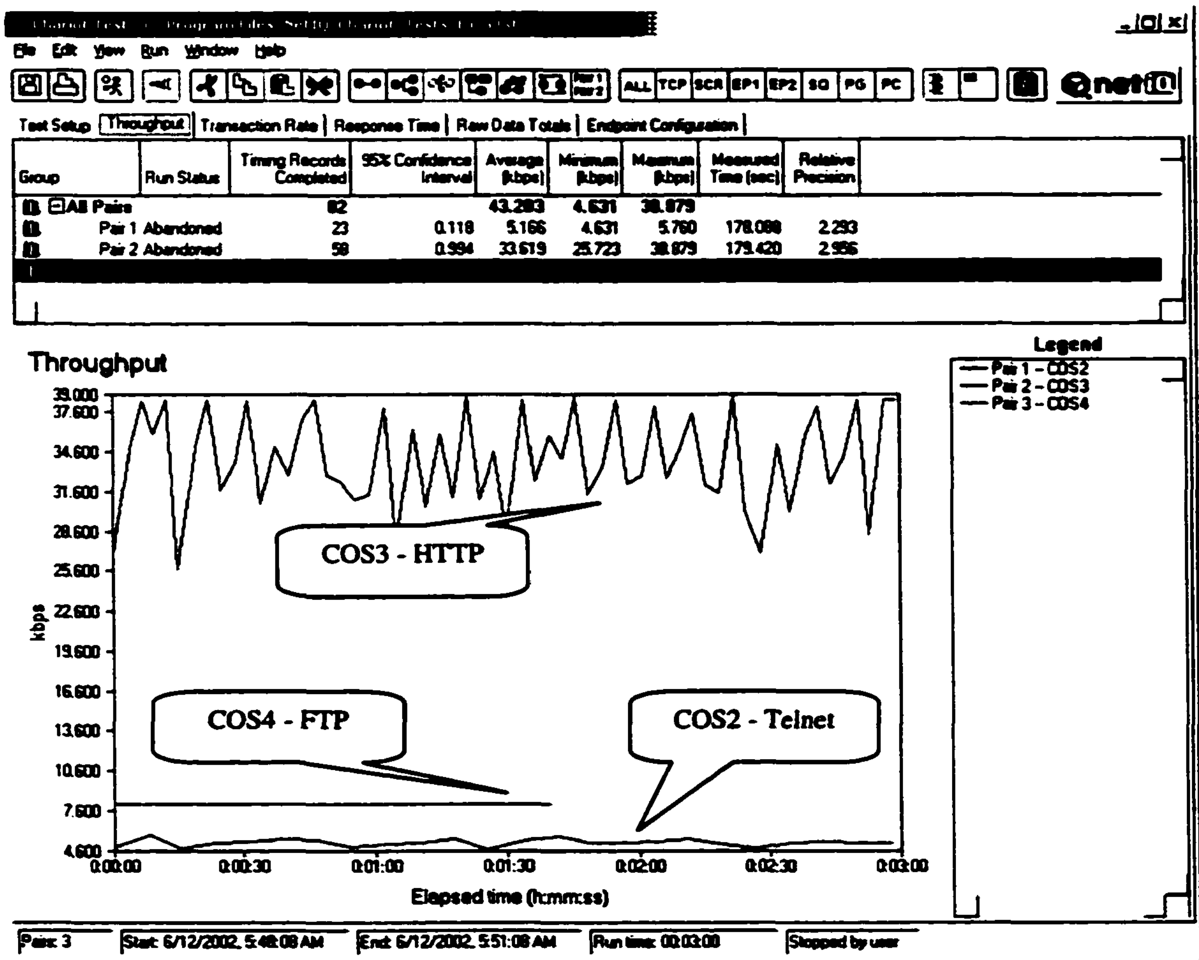


Figure 15 : Traffic behavior after applying CAR QoS mechanism

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The following command shows that the CAR mechanism is enabled on the sub-interface called Serial0/0.1 and some statistics available into the router:

```
CE5#sh int rate-limit
Serial0/0.1
 Output
  matches: access-group 100
   params:  32000 bps, 8000 limit, 8000 extended limit
   conformed 560 packets, 55559 bytes; action: set-dscp-transmit 26
   exceeded 0 packets, 0 bytes; action: drop
   last packet: 668ms ago, current burst: 68 bytes
   last cleared 00:02:18 ago, conformed 3000 bps, exceeded 0 bps
  matches: access-group 101
   params:  16000 bps, 8000 limit, 8000 extended limit
   conformed 1186 packets, 126372 bytes; action: set-dscp-transmit 18
   exceeded 0 packets, 0 bytes; action: drop
   last packet: 20920ms ago, current burst: 0 bytes
   last cleared 00:02:18 ago, conformed 7000 bps, exceeded 0 bps
  matches: access-group 102
   params:  8000 bps, 8000 limit, 8000 extended limit
   conformed 56 packets, 83050 bytes; action: set-dscp-transmit 0
   exceeded 39 packets, 58344 bytes; action: drop
   last packet: 12408ms ago, current burst: 0 bytes
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

last cleared 00:02:18 ago, conformed 4000 bps, exceeded 3000 bps

5.2.2 Weighted Fair Queuing (WFQ) QoS Mechanism

Cisco's WFQ implementation divides nearly the whole available amount of queue space to the aggregate specific queues based on their weight. Hence, a best-effort aggregate with a guarantee of 2% of bandwidth under congestion receives only a small amount of queue space. Following the idea of a premium service, it does not make sense to provide a significant amount of queue space to the premium flow, because it would result in a potentially higher latency. Consequently, an implementation of the WFQ QoS mechanism should change the default buffering dramatically by providing a significant amount of queue space to the best-effort queue.

The next test used the CBQ (Custom Based Queuing) mechanism for TCP flows. It showed that part of the bandwidth is lost in overhead and that the effective throughput is closer to 82% of the total bandwidth.

When UDP stream were run concurrently with TCP streams and no management controls were applied, the UDP stream overwhelmed the TCP stream and grabbed as much of the available bandwidth as it could. TCP throughput level dropped about 10% of potential. When CBQ QoS mechanism was applied, both UDP and TCP streams were able to flow to the level of the bandwidth reserved for them.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In all the TCP testing, no errors or dropped packets were detected. With CBQ enabled, TCP streams quickly reached steady state because of the controlled rate at which the packets were output/forwarded.

CBQ's rate limiting behavior demonstrates that connectionless protocols do not "cheat" by grabbing more than their allotments of available bandwidth than was configured for them. Concurrent TCP streams appear to oscillate when FIFO queues are applied at the router egress interface. As the router queue congests and packets are dropped, the data source employs TCP congestion control and throttles the flow rate. As one flow backs off, the other flow adapts to the increased bandwidth by increasing it's flow rate until it detects congestion and then it backs off. CBQ rate control permits concurrent TCP flows to reach steady-state prior to interface congestion.

In conclusion, based on the effective tests, when applying QoS mechanisms to TCP flows, the following suggestions should be followed. When specifying required bandwidth for CAR, the packet size should be taken into account; packets that exceed the rate-limit should be dropped.

The burst size parameter should be set to the bandwidth (in bytes/second) multiplied by the assumed maximum round trip time. It is recommended to assign 99% of the available bandwidth of the slowest link (BW) to the premium class of WFQ. This is not as extreme as it may seem at first glance. If CAR policing mechanism is used to

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

limit the rate of premium flows, premium traffic will never actually take 99% of the bandwidth except, perhaps, for very short bursts. However, providing 99% to the queue for premium traffic means that WFQ will serve packets very quickly after they arrive in the queue, thus minimizing the queuing and therefore decreasing the delay that the premium traffic encounters.

However, guaranteeing a smaller amount of bandwidth to the premium aggregate would not prevent a premium flow from using up to 99% of the available bandwidth if there is no congestion or rate-limiting. This is because WFQ comes into play only if there is congestion. The following proof shows that the proposed configuration minimizes the queuing time for packets.

If all ingress interfaces guarantee 99% of the rate BW, the maximum queuing time introduced for the premium traffic can be actually computed. Let **x *BW** be the amount of premium bandwidth allowed by the policy (e.g, the average rate limit of CAR), where $0 < x < 1$. Furthermore, let **rtt** be the round-trip time with no competing traffic and **qt** the maximum queuing time for the premium traffic, introduced by bursts.

Note that **qt** is the time the last packet of a burst remains in the router queue until it gets transmitted. Assuming that there is no additional queuing delay for the acknowledgments, the maximum estimated round-trip time used by CAR to calculate the token bucket depth is rtt+qt. Assuming this round-trip time CAR actually limits

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the maximum size of a burst to $x*BW*(rtt + qt)$. Because the bandwidth guaranteed for premium traffic is of $0.99*BW$, the queuing time is

$$qt = \frac{x*BW*(rtt + qt)}{0.99*BW}$$

Assuming $x < 0.99$ it follows that

$$qt = \frac{rtt*x}{0.99 - x}$$

For example, limiting the allowed premium bandwidth to 33% of the network capacity ($x = 0.33$) results in a maximum increase of latency of rtt/2.

The buffer size allocated to the WFQ queues should leave a significant amount of space. This could be used on demand by either the best-effort class or the premium class. This configuration minimizes the impact on handling traditional best-effort traffic under congestion. It is important to note that the usage of the overall queue limit is available only with the standard tail-drop behavior of the class of queue, and not with activating WRED on WFQ queues.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The following figures show that applying WFQ QoS mechanism and allocating 99% of the bandwidth to the premium class doesn't have much of a impact on the other flows. This setting becomes effective when there is congestion.

Before applying the WFQ mechanism, the ftp traffic was using the most of the available bandwidth.

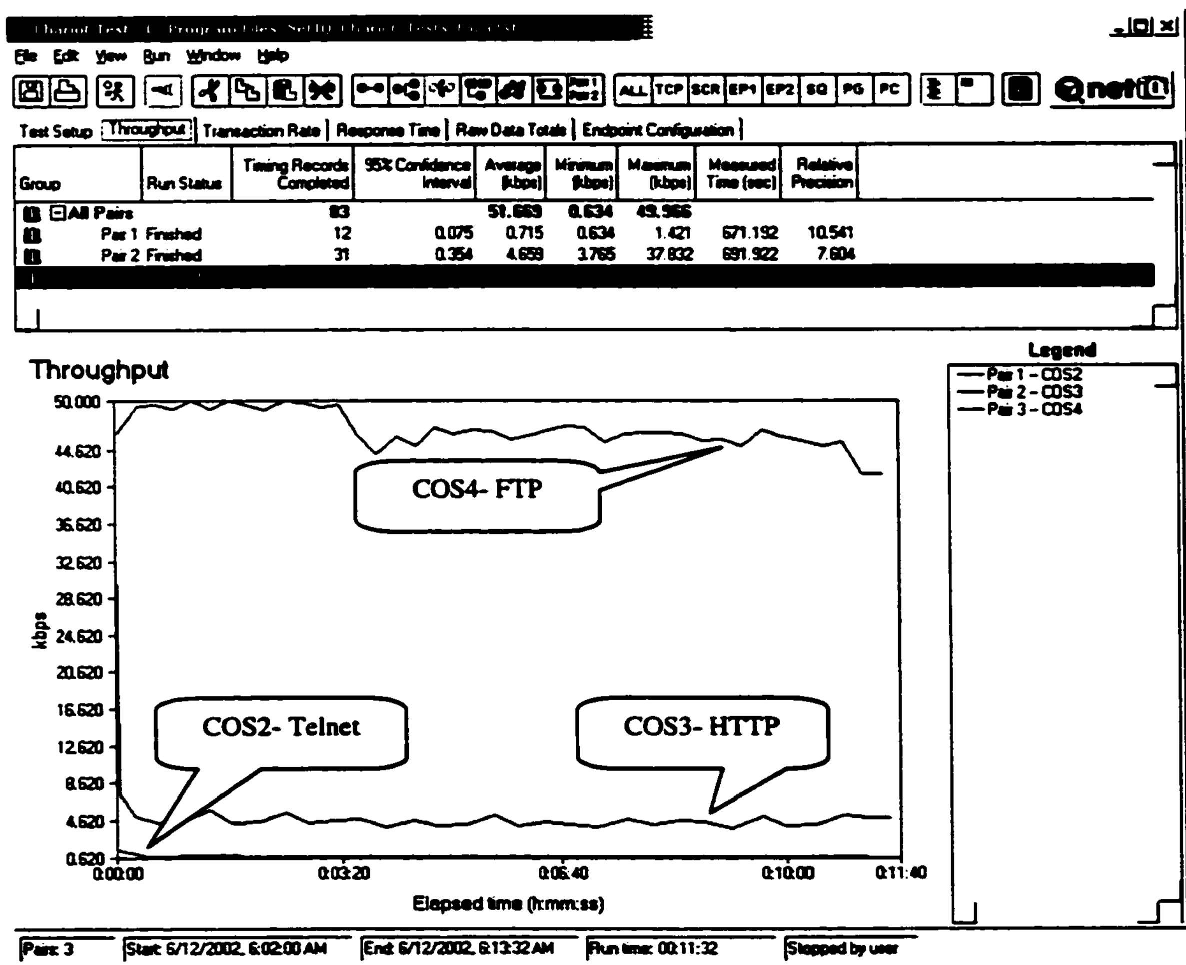


Figure 16 : Traffic behavior prior to applying WFQ mechanism

After applying WFQ mechanism, the proportion in which the classes used the bandwidth was the same. Thus, setting the bandwidth percentage to 99% for the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

premium class does not have a major impact on the other classes when there is no congestion. This is because the premium traffic (telnet session) needs quick response time, but it doesn't actually use a large piece of bandwidth, while the ftp traffic would take whatever bandwidth is available and for that reason it should be limited.

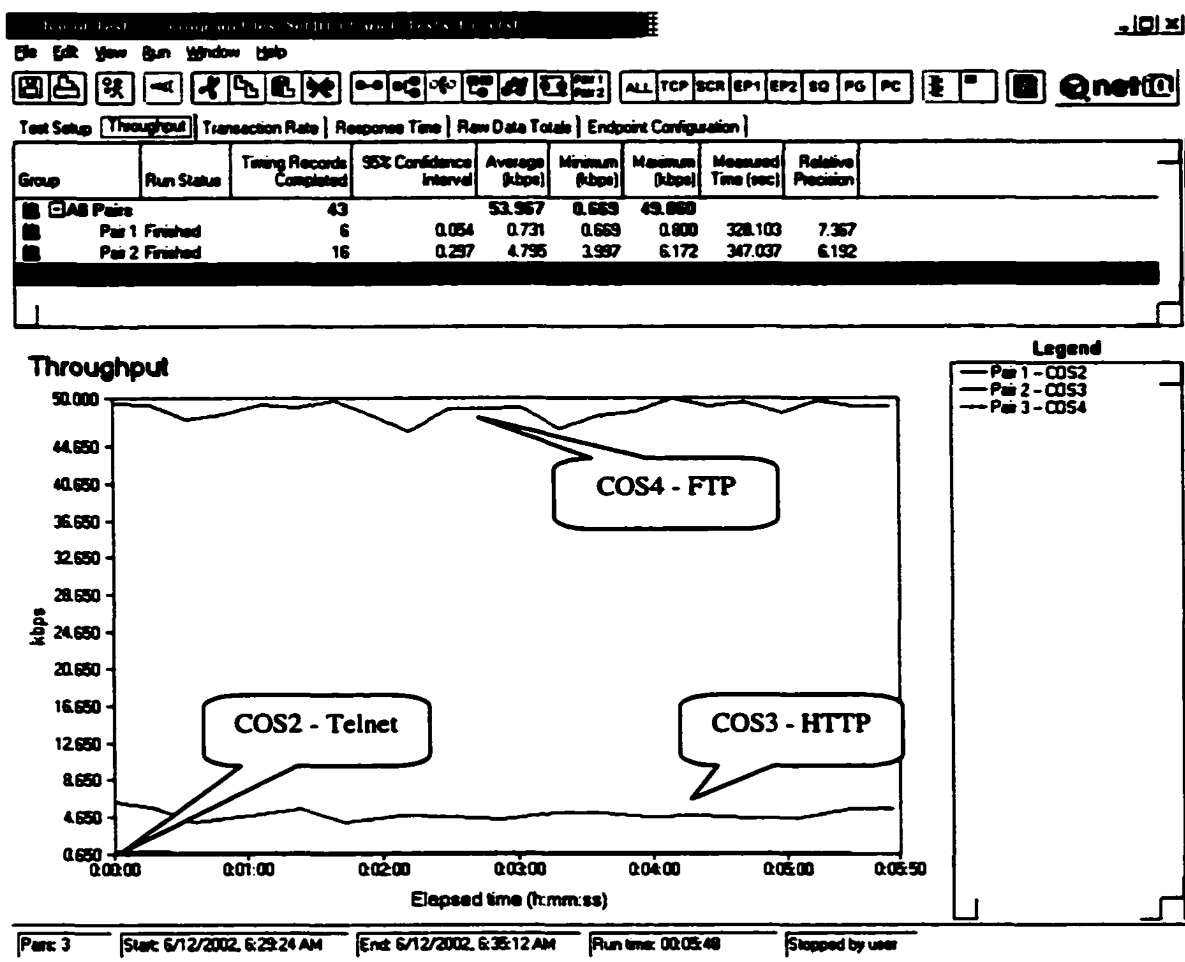


Figure 17 : Traffic behavior after applying WFQ mechanism

The following command shows that the WFQ mechanism is enabled on the interface called Serial0/1 and some statistics available into the router

CE5#sh queueing fair

Current fair queue configuration:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Interface	Discard threshold	Dynamic queues	Reserved queues	Link queues	Priority queues
Serial0/1	64	256	0	8	1
BRI2/0	64	16	0	8	1
BRI2/0:1	64	16	0	8	1
BRI2/0:2	64	16	0	8	1

```
CE5#sh queueing interface serial0/1
Interface Serial0/1 queueing strategy: fair
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: weighted fair
  Output queue: 0/1000/64/0 (size/max total/threshold/drops)
     Conversations  0/0/256 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 1536 kilobits/sec
```

5.2.3 Weighted Random Early Discard (WRED) QoS Mechanism

The objective of the tests performed applying WRED mechanism was to understand the impact of each parameter to differentiate the traffic and the bandwidth share among different classes of traffic, where each class was identified by different values of IP precedence.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

When a packet belonging to a queue for which WRED is enabled arrives, the following actions take place: The Average Queue Size (AQS) is calculated, and if its value is less than the minimum WRED threshold the packet is enqueued. Otherwise, accordingly to the Drop Probability of the packet within a WRED class, the packet is dropped or enqueued. This behavior can be adjusted by setting the WRED parameters. They can be set for each aggregate of packets (class of service) and these parameters are:

- **Minimum Threshold:** number of packets above which the Drop Probability is increased linearly
- **Maximum Threshold**: number of packets above which all packets of that class are discared
- **Max. Drop Probability**: maximum value for the Drop Probability when the Average Queue size equals the Max. Threshold
- **Exponential Weighting Factor**: unique for all WRED classes, influences the way the Average Queue size is calculated

The Average Queue Size is an exponential weighted moving average of the instantaneous queue size that depends on the **exponential weighting factor** value ("**n**") that can be set in Cisco Router. The formula for the calculation of the Average Queue Size (AQS) from the Instantaneous Queue Size (Qsize) is the following:

$$W_q=1/(2^n)$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$AQS_i = (1 - W_q) * AQS_{i-1} + W_q * Qsize$$

W_q = Weighted queue

AQS = Average Queue Size

n = exponential weighting factor

The more the weight W_q is increased the more the previous values of the instantaneous queue size are taken into account, and the system has more capacity to absorb burst. At the same time W_q should not be too low otherwise WRED cannot react quickly to congestion. The choice of W_q determines the "time constant" for the averaging for the queue size. If its value is too low, than the estimated average queue size is responding too slowly to transient congestion. If its value is too high, than the estimated average queue size too closely tracks the instantaneous queue size.

The following formulae are derived for the purpose of mapping the WRED parameters into quality parameters. Since Min. and Max. Thresholds are given in unit of packets, a conversion from time to packets must be performed using the following formula.

$$T = \frac{\text{Bandwidth available (Mbps)}}{\text{MTU (bytes/pkt)} * 8 \text{ bits/byte}}$$

, where

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

T = throughput

MTU = maximum transmission unit

$$MinTresh_i = 30 \text{ ms} * T$$

$$MaxTresh_i = 100 \text{ ms} * T$$

$$n = \frac{\log_{10}(10/T)}{\log_{10}(2)}$$

For example if the maximum average queue delay accepted is 100ms, first the value of T must be determined. It depends on the link speed of the sub-interface, and then MaxThreshold=100ms * T. A good rule for Min Threshold selection is to set it smaller than or equal to 1/2 of the Max Threshold. For the testing scenarios it was chosen 1/3.

The purpose of the tests was to identify the dependencies between the bandwidth share and the WRED parameters (minimum and maximum threshold and drop probability). The results showed that the relative bandwidth allocated to each class was not dependent on the drop probability assigned to that, but only dependent on the minimum and maximum threshold values. If WRED is not enabled, the bandwidth share varies only according to the number of flows.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The following picture represents the traffic behavior without any QoS mechanisms applied.

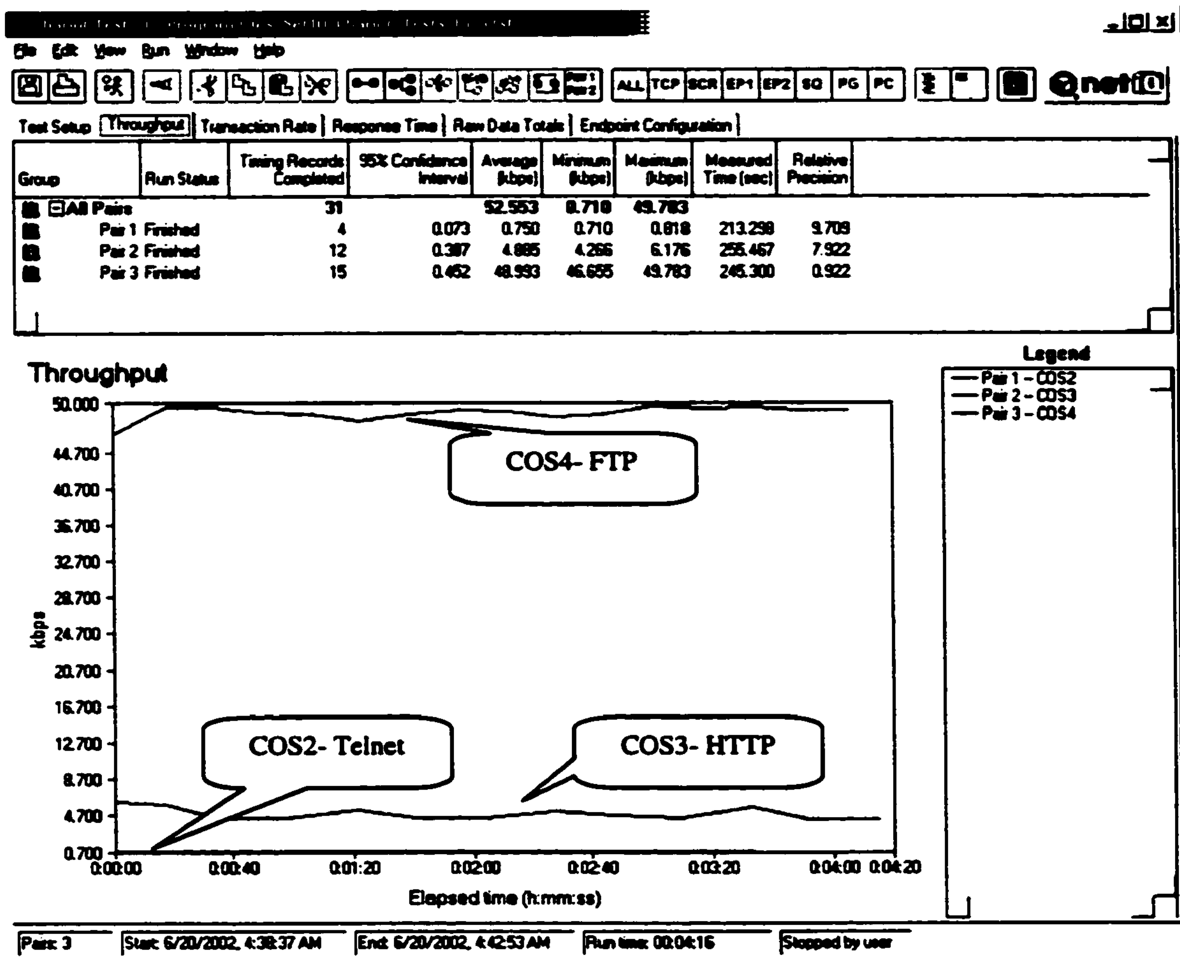


Figure 18 : Traffic generation without any QoS controls

5.2.3.1 Adjusting the drop probability

The next test was performed with the same traffic, applying WRED QoS mechanism, using the same minimum and maximum threshold values and varying only the drop probability as follows 2 for COS2 class, 5 for COS3 class, and 10 for COS4 respectively. Minimum and maximum threshold values considered were 15 and 30.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The following figure illustrates the test results.

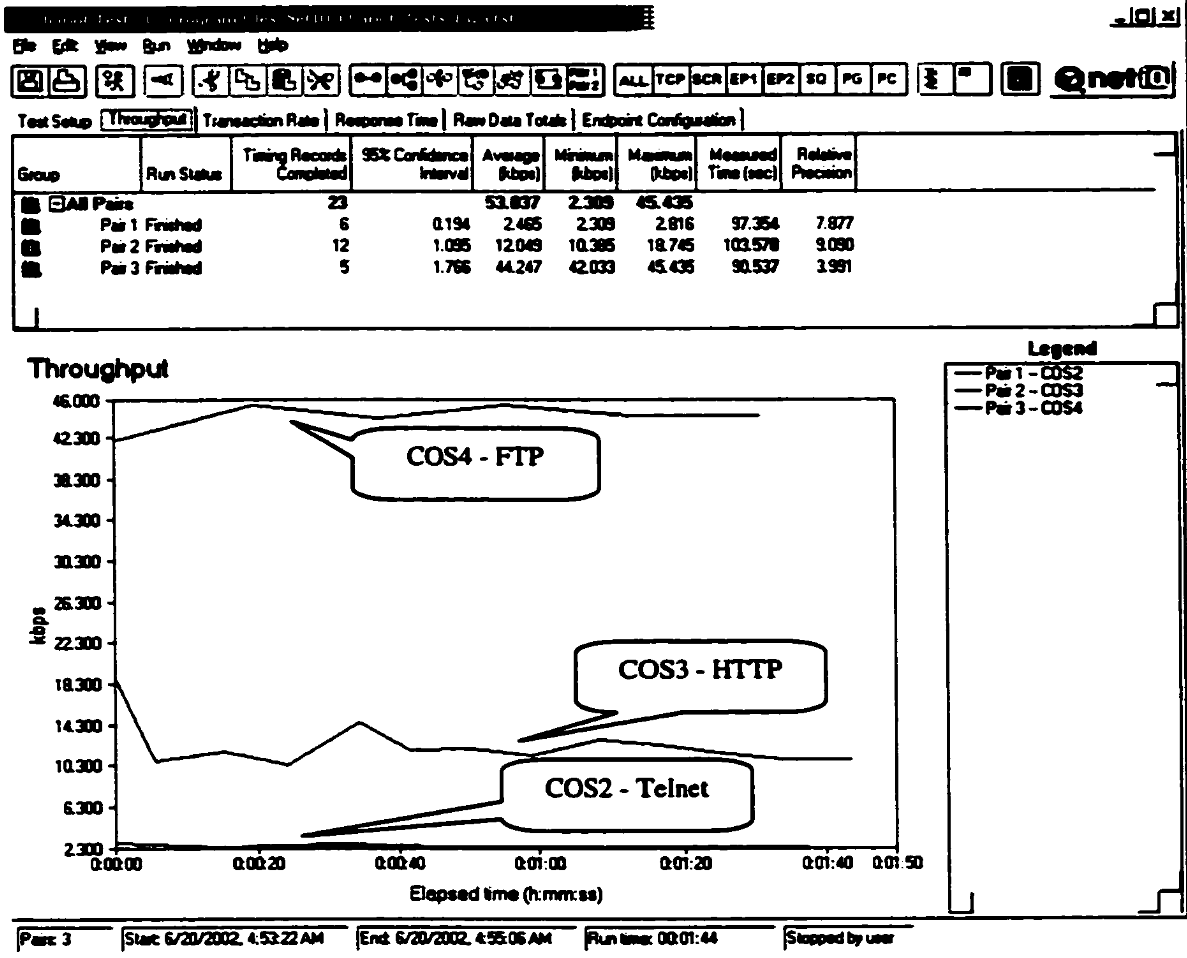


Figure 19 : Applying WRED QoS mechanism with various drop probabilities

The conclusion of this testing scenario was that the traffic was not improved or differentiated based on the drop probability values.

The next testing scenario used the same drop probability for all classes, adjusting only the minimum and maximum threshold values as follows: COS2 class was assigned with minimum threshold 200 and maximum threshold 300, COS3 class was assigned

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

with minimum threshold 100 and maximum threshold 200, and COS4 class was assigned with default WRED parameters. The drop probability value used for all classes was 10. The router configuration of the WRED mechanism that describes the above mentioned scenario is the following:

```
policy-map CEPHB
 class COS2
  bandwidth percent 97
 class COS3
  bandwidth percent 1
  random-detect dscp-based
  random-detect exponential-weighting-constant 1
  random-detect dscp 18   100   200   10
 class COS4
  bandwidth percent 1
  random-detect dscp-based
  random-detect exponential-weighting-constant 1
  random-detect dscp 0   1    10    10
```

The router statistics showing that the WRED mechanism became active on interface Serial0/0.1 are the following:

```
CE5#sh policy-map interface s0/0.1
 Serial0/0.1: DLCI 200 -
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
Service-policy output: CEPHB
 Class-map: COS2 (match-all)
  867 packets, 81156 bytes
  5 minute offered rate 0 bps, drop rate 0 bps
  Match: ip dscp 26
  Weighted Fair Queueing
   Output Queue: Conversation 25
   Bandwidth 97 (%)
   Bandwidth 27 (kbps) Max Threshold 64 (packets)
   (pkts matched/bytes matched) 574/53806
   (depth/total drops/no-buffer drops) 0/0/0

 Class-map: COS3 (match-all)
  1708 packets, 180188 bytes
  5 minute offered rate 0 bps, drop rate 0 bps
  Match: ip dscp 18
  Weighted Fair Queueing
   Output Queue: Conversation 26
   Bandwidth 1 (%)
   Bandwidth 0 (kbps)
   (pkts matched/bytes matched) 926/108040
   (depth/total drops/no-buffer drops) 0/0/0
    exponential weight: 1
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

mean queue depth: 0

dscp	Transmitted pkts/bytes	Random drop pkts/bytes	Tail drop pkts/bytes	Minimum thresh	Maximum thresh	Mark pro
af11	0/0	0/0	0/0	32	40	1/1
af12	0/0	0/0	0/0	28	40	1/1
af13	0/0	0/0	0/0	24	40	1/1
af21	1708/180188	0/0	0/0	100	200	1/1
af22	0/0	0/0	0/0	28	40	1/1
af23	0/0	0/0	0/0	24	40	1/1
af31	0/0	0/0	0/0	32	40	1/1
af32	0/0	0/0	0/0	28	40	1/1
af33	0/0	0/0	0/0	24	40	1/1
af41	0/0	0/0	0/0	32	40	1/1
af42	0/0	0/0	0/0	28	40	1/1
af43	0/0	0/0	0/0	24	40	1/1
cs1	0/0	0/0	0/0	22	40	1/1
cs2	0/0	0/0	0/0	24	40	1/1
cs3	0/0	0/0	0/0	26	40	1/1
cs4	0/0	0/0	0/0	28	40	1/1
cs5	0/0	0/0	0/0	30	40	1/1
cs6	0/0	0/0	0/0	32	40	1/1
cs7	0/0	0/0	0/0	34	40	1/1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ef	0/0	0/0	0/0	36	40	1/1
rsvp	0/0	0/0	0/0	36	40	1/1
default	0/0	0/0	0/0	20	40	1/10

Class-map: COS4 (match-all)

1172 packets, 1124692 bytes

5 minute offered rate 13000 bps, drop rate 0 bps

Match: ip dscp 0

Weighted Fair Queueing

Output Queue: Conversation 27

Bandwidth 1 (%)

Bandwidth 0 (kbps)

(pkts matched/bytes matched) 758/856114

(depth/total drops/no-buffer drops) 0/154/0

exponential weight: 1

mean queue depth: 0

dscp	Transmitted pkts/bytes	Random drop pkts/bytes	Tail drop pkts/bytes	Minimum thresh	Maximum thresh	Mark pro
af11	0/0	0/0	0/0	32	40	1/1
af12	0/0	0/0	0/0	28	40	1/1
af13	0/0	0/0	0/0	24	40	1/1
af21	0/0	0/0	0/0	32	40	1/1
af22	0/0	0/0	0/0	28	40	1/1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

af23	0/0	0/0	0/0	24	40	1/1
af31	0/0	0/0	0/0	32	40	1/1
af32	0/0	0/0	0/0	28	40	1/1
af33	0/0	0/0	0/0	24	40	1/1
af41	0/0	0/0	0/0	32	40	1/1
af42	0/0	0/0	0/0	28	40	1/1
af43	0/0	0/0	0/0	24	40	1/1
cs1	0/0	0/0	0/0	22	40	1/1
cs2	0/0	0/0	0/0	24	40	1/1
cs3	0/0	0/0	0/0	26	40	1/1
cs4	0/0	0/0	0/0	28	40	1/1
cs5	0/0	0/0	0/0	30	40	1/1
cs6	0/0	0/0	0/0	32	40	1/1
cs7	0/0	0/0	0/0	34	40	1/1
ef	0/0	0/0	0/0	36	40	1/1
rsvp	0/0	0/0	0/0	36	40	1/1
default	1018/977238	154/147454	0/0	1	10	1/10

Class-map: class-default (match-any)

53 packets, 13828 bytes

5 minute offered rate 0 bps, drop rate 0 bps

Match: any

They correspond to the following graphic results:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

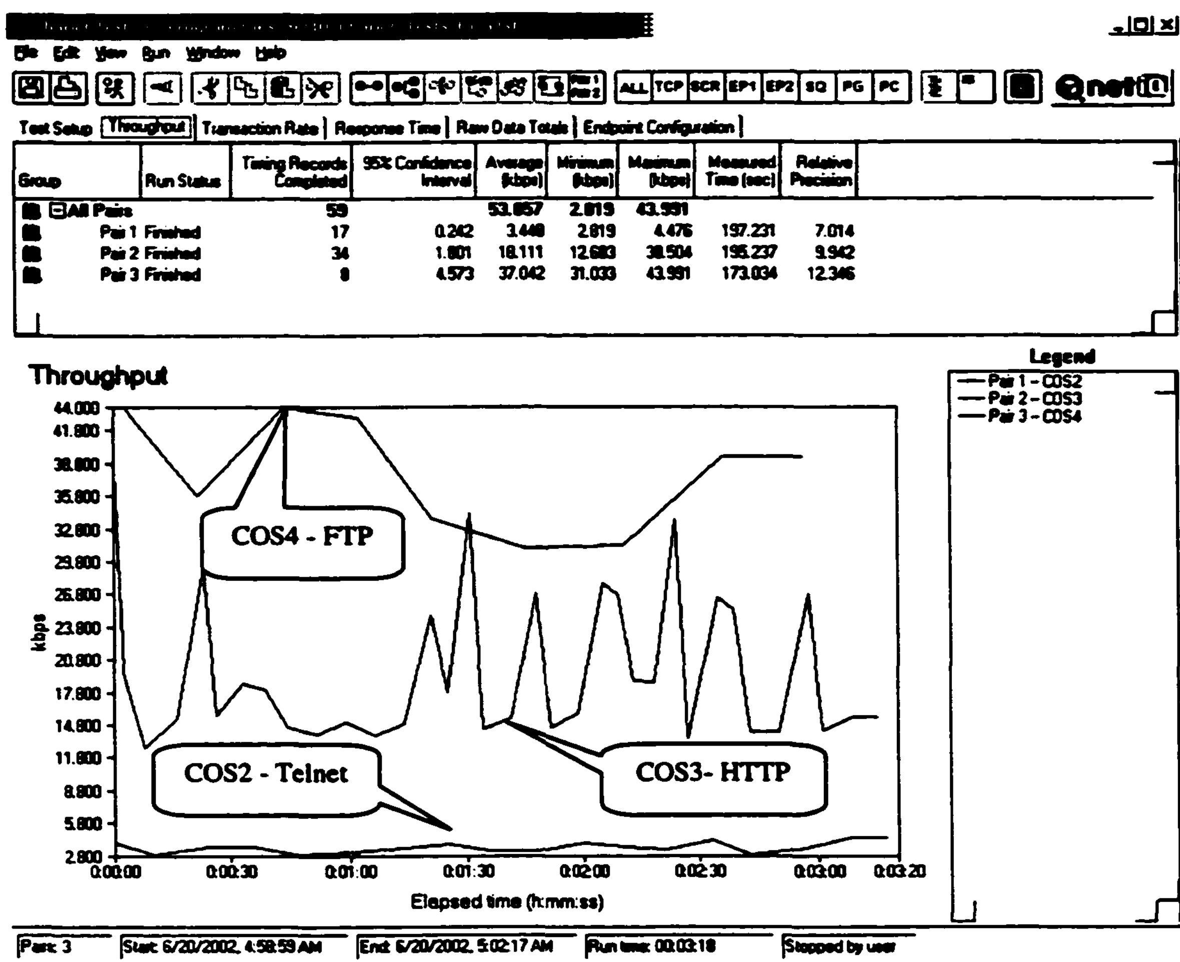


Figure 20 : Applying WRED QoS mechanism with different minimum and maximum threshold parameters

It can be noticed from the previous picture that the COS4 traffic dropped dramatically, while the COS3 acquired more bandwidth, as well as COS2.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.2.3.2 Adjusting the drop probability and the minimum and maximum threshold WRED parameters

When the drop probability parameter was adjusted in conjunction with setting the minimum and maximum threshold parameters, the traffic behavior was not impacted, as the following figure shows:

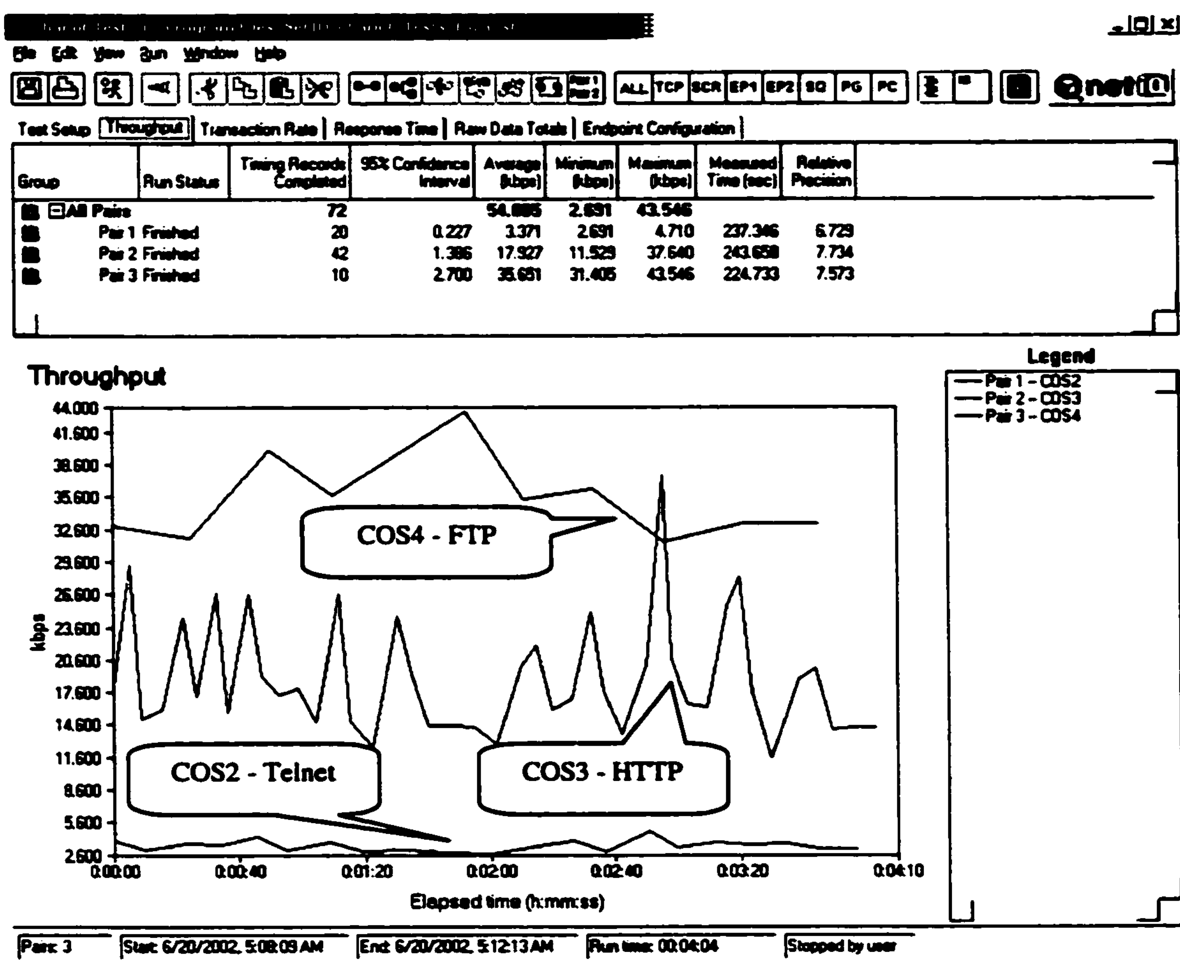


Figure 21: Impact of adjusting the drop probability on the traffic behavior

The router WRED configuration corresponding to this scenario is the following:

class COS2

bandwidth percent 97

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
 random-detect dscp-based
 random-detect exponential-weighting-constant 1
 random-detect dscp 26   200   300   2
class COS3
 bandwidth percent 1
 random-detect dscp-based
 random-detect exponential-weighting-constant 1
 random-detect dscp 18   100   200   5
class COS4
 bandwidth percent 1
 random-detect dscp-based
 random-detect exponential-weighting-constant 1
 random-detect dscp 0   1     10    10
```

5.2.4 The synergy of applying a combination of QoS mechanisms

The following picture shows the impact of all QoS mechanisms combined: classification for marking the traffic on ingress interface, CAR policing for limiting the traffic, WRED discard mechanism and WFQ queuing mechanism on egress interface. It can be seen that prior to applying the QoS controls the traffic is competing over the bandwidth, but as soon as they are active, the traffic separates into distinct classes with different priorities and behavior.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

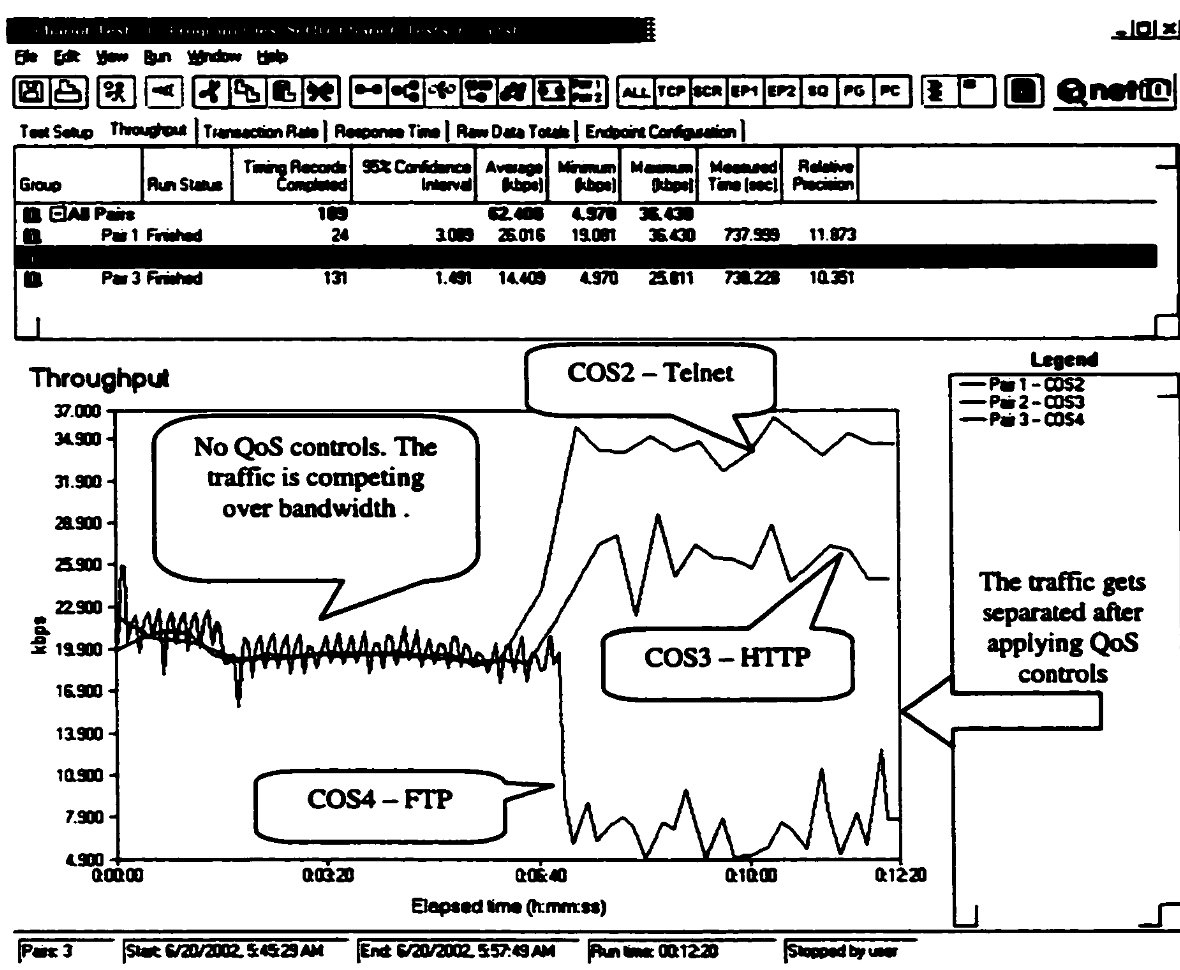


***Figure 22*: The synergy of applying combined QoS controls**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.2.5 Test results and future work

In conclusion, the test results described in this section presented a quantitative evaluation of QoS implementation for the Diffserv model, regarding high-performance TCP- flows and demonstrated that QoS can be delivered to such flows if mechanisms are configured carefully. The basic challenge when dealing with TCP flow is the burstiness introduced by the TCP's sliding window. This must be addressed by appropriate policies at the edge routers that must support bursts corresponding to the TCP window size.

Burstiness also introduces problems on the interior interfaces, because accumulating aggregate bursts might exceed the available bandwidth. For that reason the implementation of the QoS should avoid queuing by over provisioning the guaranteed amount of bandwidth for the premium class as much as possible. The tests also have shown that CAR policing and WFQ queuing mechanisms can be used to implement a Diffserv architecture. Policing at edge routers is done based on the applied bandwidth and the estimated round-trip time.

Implementing the QoS mechanism for the outgoing traffic by guaranteeing 99% (WFQ mechanism) of the available bandwidth to the premium class minimizes the latency of premium traffic, while the impact of this configuration on best-effort traffic in the absence of premium traffic is negligible.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The results also showed that the relative bandwidth allocated to each class was not dependent with the drop probability assigned to that class, but directly proportional to the minimum and maximum threshold values. If WRED is not enabled, the bandwidth share varies only according to the number of flows.

The last test showed that when the QoS controls are applied together, the effect is much greater then the sum of each one of them. In the absence of QoS controls the traffic was competing over bandwidth. When the QoS mechanisms became active, the traffic separated according to the different priorities assigned to each class and it was constrained to follow a more predictable and desired behavior.

There are multiple challenges in trying to automate the process of setting the QoS mechanisms and translating qualitative parameters into router commands. This tests described in this chapter are meant to reveal the fact that this is a multi-faceted and very complex problem. They should be considered not a solution but rather a starting point in evaluating each specific QoS mechanism, their implications and outcome.

6 QoS Level

A policy system shifts the focus from configuring individual devices to managing network in aggregate, and controlling device behavior through network policies. The system developed within this project implements by centralizing control functions into a single software application.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

At the center of such a policy system is the policy rule. Policy rules may be general and abstract or specific and concrete. In either case, policy rules represent a pairing of conditions and actions that are intended to be device and vendor independent.

The schema described in [4] defines the composition of policy rules, along with some of the characteristics of devices that are being controlled by policy rules. It also specifies the format and the organization of the storage for policy rules, as well as the data that characterize the devices being controlled by policy rules. Other characteristics of devices, used to capture the semantics and relationships between different objects being managed, define how the conditions and actions represented in a Policy rule are interpreted and what effect they have on the functions of the device. These are described in [1]. The module described in this chapter presents a context for the schema and semantic definitions.

A policy rule is a specification of a set of optionally - sequenced actions to be initiated when a specified set of conditions is satisfied. A Policy rule takes the form:

IF *<set of conditions to be met>* **THEN** *<set or sequence of actions to be taken>*

and it is designed to specify the behavior of a specific device.

The ***<condition>*** expression may be a compound expression and it may be related to entities such as hosts, applications, protocols, users, other system sub-components,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

etc. The *<action>* may be a set of actions that specify services to grant or deny or other parameters to be input to the provisioning of one or more services.

There are two types of conditions: conditions that specify characteristics of the traffic flow (called Packet Stream Conditions) and conditions that describe time validity for the rule (called Time Period Conditions). One rule can have multiple Packet Stream Conditions, but only one Time Period Condition attached to it.

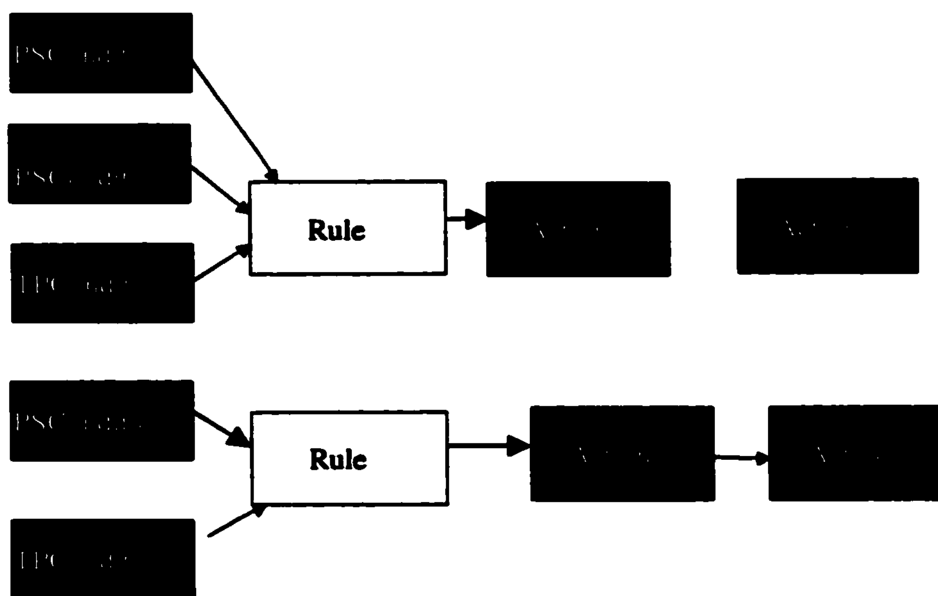


Figure 23 : The Structure of a Policy Rule

The policy rules specify the logic used to deliver the prescribed service and service levels. They are interpreted, validated and then mapped to the underlying policy mechanisms of the device.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The policy rule serves as the point of interoperability between entities participating in the policy system. A policy system built upon the expression of rules must demonstrate at least the following abilities:

- The ability to enable a user to define and update policy rules.
- The ability to store and retrieve policy rules.
- The ability to interpret, implement and enforce policy rules.
- The ability to validate the rules for the purpose of creating a conflict free environment before translating and deploying rules into the devices

The challenge for the QoS level of this project is to ***acquire***, ***translate*** in a device-specific language and ***deploy*** Policy rules into devices in a ***conflict-free environment***.

Functionally, interpreting rules is separate from the implementation of the rule, which is the evaluation of conditions and the execution of actions. The translation of a policy rule represents an analogous expression of the rule, but in a more device-specific form.

6.1 Detection and Resolution of Conflicts Among Rules in a Policy Based Network Management System

The growth in the number of users and multimedia content has caused the Internet to undergo fundamental changes in the demand for bandwidth and new services, beyond the traditional best effort service. This has fueled the demand for routers able to handle large traffic volumes measured in millions of packets per second. At the same time, router technology is advancing from simple destination based forwarding to

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

incorporate a number of new capabilities, which affect the forwarding process. Successful deployment of these technologies and services is crucial for the successful evolution of the Internet towards a full service network.

In order for the Internet to transform itself from today's chaotic logjam to the full service network of tomorrow, it is important to provide a solution to the conflict detection problem among policy rules used to define classification filters.

Policy rules conflicts can lead to ambiguities in packet classification. This is possible because a packet might match multiple criteria defined in policy rules, each with a different associated action. One goal of this research work is to address the issue of consistency amongst policy rules or policy rules conflict detection and resolution.

This can be divided into two distinct sub-problems. First, how does one detect such conflicts? Second, given a set of conflicting rules, how does one resolve these conflicts? The aim of this work is to find a solution to both problems, and a fast solution optimized for commonly occurring type of rules. It is important to consider policy rules conflict resolution in any scheme involving policy based network management, since if they are not handled correctly, they can cause packets to be subject to the wrong actions. For example, incorrectly matching packets to filters in firewalls can cause security problems.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The main goal of this work is for the conflicts to be detected at the policy level, which is designed above the network layer. Consequently, when packet classification algorithms are performed into the router, on a per-packet basis, the conditions defining the flows are known to be conflict-free.

One approach in solving the conflicts among rules can be prioritizing conflicting rules and letting the one with a higher priority to override the others. However, this scheme does not always work and that is the reason why my implementation performs the conflict detection algorithm every time a new rule is attached to the pool of existing rules.

I also propose a new scheme for solving policy rules conflicts by defining another rule for the domain where the rules overlap. This is another option of solving policy rule conflicts used in the proof of concept software module.

6.1.1 Conflict Detection Algorithms using Multidimensional Range Searching

A ***policy conflict*** occurs when the conditions associated with two or more policy rules are simultaneously satisfied, but not all of the actions associated with the policy rules can be performed together. In other words, when the definition of the rules is ***ambiguous*** and when they don't define ***disjoint sets of conditions*** associated with actions.

The most common policy conflict is expressed by rules that imply two distinct subjects within the same rule.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

As an example, the following Policy Rule could be defined: "Traffic between Point A and Point B should receive ***Gold*** type of service". This could be translated into the following two policy rules:

First Rule:

Packet Stream Condition:

IP address source = 135.16.126.0

IP address source mask = 255.255.255.0

IP address destination = 108.19.35.0

IP address destination mask = 255.255.255.0

Protocol = 0 (any)

Port = 0 (any)

DSCP = 101110 (the actions to be performed for the ***Gold*** type of service are enabled through the marking of packets with the Differentiated Service Code Point - DSCP - of 101110)

Direction = incoming

Time Period Condition = Default, all the time.

The actions attached to this rule are specific to the Gold type of service and they can include marking/remarking, policing, buffering and queuing.

Second Rule:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Packet Stream Condition:

IP address source = 108.19.35.0

IP address source mask = 255.255.255.0

IP address destination = 135.16.126.0

IP address destination mask = 255.255.255.0

Protocol = 0 (any)

Port = 0 (any)

DSCP = 101110 (the actions to be performed for the ***Gold*** type of service are enabled through the marking of packets with the Differentiated Service Code Point - DSCP - of 101110)

Direction = incoming

Time Period Condition = Default, all the time;

The actions attached to this rule are the same as the actions attached to the first rule.

In this example, the network has been configured to treat packets with a DSCP 101110 as packets that receive Gold treatment. Thus these rules apply to the ingress interface for the network, on either an end system or a router, where packets will be marked.

The consistency-checking algorithm performs checking of a policy rule against all the rules that are to be applied on the same interface at the same time or all the rules already applied on the interface, and returns the results of this checking. Multiple kinds of checking should be performed, as described in the following sections.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.1.1.1 Validation of the data types of the terms of the specified policy rule

For the Packet Stream Conditions attached to a rule, the following syntax validations must be performed:

- Valid IP Address and mask (numeric values for each octet, between 0-255)
- Integer values for port, protocol, and dscp
- String value for traffic direction
- Non-empty values for each field; for some fields default values are pre-entered
- Address Type must be CIDR

For the Time Period Conditions attached to a rule, the following syntax validations must be performed:

- Valid date and time values
- The start date should be earlier than the end date

For the Policy Actions attached to a rule, the following syntax validations must be performed:

- If all the parameters corresponding to each type of action are valid in type
- If edit/set action results in a new next action, then ensure that next action exists.

The following syntax validations must be performed for each rule:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- Parameters valid in type (string name, numeric rule priority)
- If there is no Time Period Condition attached to it, attach the default one
- If there is another Time Period Condition already attached to this rule then reject or choose between them
- The user must define a Packet Stream Condition for each rule if none was attached already. This checking is done at the time when a rule is ready to be deployed into a device
- When attaching an action to a rule, check whether there exists another action attached to this rule then reject or choose between them.

6.1.1.2 Validation of the semantics of the Policy Rule

This validation is related to ensuring that the construction of a policy rule, and its conditions and actions, from a set of pre-defined building blocks, is meaningful and valid. Policy rules can be syntactically correct yet they can make no sense. For example, a rule may be defined stating that "Traffic from 135.16.126.35 to 135.16.126.35 should be marked as Bronze type of traffic". Its translation into Packet Stream Conditions and actions is syntactically valid, but semantically wrong since it specifies the same source and destination address.

Also, if there were multiple Packet Stream Conditions associated with a single rule, they should overlap, more exactly, there would be an inclusion relation among them. The procedure that performs this checking is called pscIntersectsRule (cond, rulename). It is performed every time a new Packet Stream Condition is to be

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

attached to a rule. The function is implemented in Java, as the rest of the project. It takes as parameters the condition and the rule name.

The Packet Stream Conditions already attached to the rule are intersected, and their resulted intersection is compared against the new Packet Stream Condition that is about to be attached to the rule.

The procedure that reports the intersection among Packet Stream Conditions attached to a rule, as well as the procedure that investigates the intersection between two Packet Stream Conditions are described in the next sub-section, as part of the generic intersection algorithm. The procedure that looks for conflicts between a new Packet Stream Condition that is to be attached to a rule and the Packet Stream Conditions already attached to that rule (pscIntersectsRule)is shown in Appendix 10.1.

The ObjectGateway is a tool (architectural block) that knows how to identify and locate objects by their name. It is described in section 6.3. The procedure takes as an input the name of the new Packet Stream Condition that is to be attached to a rule, and the name of the rule. In case the rule is not created yet, the user will be notified by an exception. Otherwise, the Packet Stream Condition object (condStruct) will be compared against the reported intersection among the Packet Stream Conditions alredy attached to that rule (pscIntersection). The absence of any overlaps within a rule between the new Packet Stream Condition and the existent Packet Stream Conditions corresponds to a ***conflict***.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Another component of a policy rule is the action sequence. Any time an action is attached to a rule, a checking algorithm is performed to see whether the action is allowed to be the first action in the action sequences. This algorithm is part of a more complex process that validates well formed sequences of actions. Details about it are described in section 6.2.

6.1.1.3 Validation of a policy rule against all other existing rules

This function investigates whether or not a newly entered policy conflicts with other policies. This type of conflict detection is not bound to any specific device, subnet, or network. It checks for static conflicts derived from Policy Rules whose conditions are simultaneously satisfied, but whose actions conflict with those of currently existing rules. For example, an administrator may define two rules stating that "A maximum of 10 video conference channels allowed on Network A", and that "Eight video conference lines are dedicated to Marketing Department on Wednesdays from 9-10am". If a third rule provisioning 3 video conference lines for Finance Department every day at 9-10 am were to be added to the rule set, a conflict should be detected.

The administrator is attempting to provision for 11 video channels (versus the maximum of 10 channels allowed). Not all policy conflicts can be detected by the consistency checking function. Rules may be based on dynamic state information. These rules may indeed conflict with others. But, these conflicts may only be detected at the time that the rule becomes valid and enforcement actions are attempted. For example, one may have policy rules that apply in normal, congested,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and business - critical (e.g., financial crisis, take away all bandwidth from everywhere to support this) conditions. On the surface, they appear to conflict with each other. However, in reality, they don't, since they are meant to apply in non-overlapping time periods and conditions. The validation performed in this component is also called off-line validation, meaning that it is not performed at the same time as the execution of the policy.

As an example regarding dynamic rules validation, let us consider the following rules:

Rule 1: If there is overall congestion in the network, then drop packets received from Network A.

Rule 2: In there is not overall congestion in the network, then accept and process packets received from Network A.

"Overall network congestion" in these conditions does not indicate a single interface's or single device's understanding of the current state of the network. Instead, it refers to an understanding of the state of the network as a whole, which might involve a monitoring application (the "congestion application") that interacts with various probes in the network, and/or introduces artificial traffic into the network and measures the progress of this traffic. In this simplified example, this application would need to provide a Boolean answer ("Yes, the network is congested" or "No, the network is not congested") to the Device Configuration tool, designated to evaluate and initiate rules deployment.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Based on whether or not the network is currently experiencing congestion, the Device Configuration Tool acts. If the network is congested, then the policy rule is deployed into the router as a set of configuration parameters that will cause it to drop packets from Network A. If the network is not congested, then the Device Configuration Tool downloads a different set of configuration parameters, that cause the device / router to process packets from Network A.

This initial configuration download isn't the end of the Device Configuration Tool's responsibilities in this case. After this, it must continue to interact with the congestion application, and be ready to download new configuration parameters to the when the network becomes congested, or when it ceases to be congested. This scenario is beyond the scope of this document, but it is considered an interesting topic for future work.

It is important to detect conflicts prior to deploying policy rules. There are cases when the policy condition(s) cannot be evaluated by the same entity that executes the action(s). So, the information stored in the policy action must, for certain cases, be device-specific. This work accommodates both device-specific as well as device-independent policies. The following example gives a motivation for the functional split between the evaluation and the execution of a policy rule.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Suppose an organization has a set of game servers, and wants to limit access to these servers to periods of the day outside normal working hours. A policy rule governing access to the servers could be written in two ways:

- It could specify time conditions, and an action indicating that access to the servers should be enabled or disabled;
- It could specify the same time conditions, but the action could contain directives specific to the device where the policy is to be deployed.

The purpose of detecting rule conflicts is to make it easier for the device to execute policy rule semantics without evaluating.

6.1.1.4 Consistency Checking Algorithm

In order to ensure that the enforced mechanism described by the policy rule can consistently work as an "integrated" whole, a complex algorithm called "consistency checking" or "conflict detection algorithm" is performed. It detects whether a new rule is conflicting with the policy rules already supported at the same time by a specific interface.

A policy conflict occurs when the conditions of two or more Policy Rules are concurrently satisfied but the actions that they mandate produce inconsistent results with each other. For example, a policy rule specifying that "All engineers get Bronze type of service" is in conflict with another rule defining that "The lead engineer gets gold service". This is a direct conflict, since there are directly identifiable terms in each Policy Rule that conflict. However, there are also indirect conflicts, such as with

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

this third rule: "All FTP traffic gets best effort". This conflicts only if an engineer decides to send FTP traffic.

Each rule consists of conditions defining the traffic flow (packet stream conditions), conditions defining the period of time when the rule becomes valid (time period conditions), and actions to be followed for the packets that match the conditions.

Each packet stream condition range from a 1-tuple to 7-tuple definition including :Source IP Address, Source IP Mask (used for wildcards), Destination IP Address, Destination IP Mask, protocol number, source port, destination port, DSCP (DiffServ Code Point) values, and traffic flow direction.

Each time period condition is identified by 4 values: start date, end date, start time, end time.

Typical actions include accept/deny the traffic, mark outgoing DSCP, assign to class of service X, forbid reservations above a certain limit, queuing and buffering mechanisms.

The following is an example of a typical policy rule:

Rule 1: Mark all HTTP packets from addresses 129.1.x.x to 130.1.1.1 on destination port 20 with DSCP byte 101.

Packet Stream condition:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

SourceAddress : 129.1.0.0 to 129.1.255.255

Source Mask: 255.255.0.0

DestinationAddress: 130.1.1.1

Destination Mask: 255.255.255.255

Protocol: 80

DestinationPort: 20

Source Port:0 (any)

DSCP : 101

Direction: outgoing

Action: Marking action followed by transmit.

Time Period Condition: All the time

It can be noticed that a policy rule is a collection of d-dimensional ranges $[l_i^1, r_i^1] \times \ldots \times [l_i^d, r_i^d]$ or sets (as in the case of port numbers and protocol numbers), time specifications and actions.

The precise nature of the action is not relevant here except that we can determine if two actions A_i and A_j are in ***conflict*** (for example, is A_i is to allow the packets to be transmitted while A_j is to drop them, there is a conflict of action).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The role of the conflict detection algorithm is thus to determine the existent intersection among rules, which are regarded as d-dimensional objects, whose values are ranges or sets. The Packet Stream conditions typically specify IP Address ranges as an IP address $a_1 \ldots a_{32}$ and a mask of a certain number n of bits, that is $[a_1 \ldots a_n 00..000,\ a_1 \ldots a_n 11..111]$. So these are not arbitrary ranges. Instead they are hierarchical, that is, if two ranges intersect, one is completely contained in the other. All our results will work for arbitrary ranges in each dimension, although some of our algorithms can be made simpler for implementation purposes if the ranges are hierarchical.

Existing IP routers use destination based routing, that is, use filters with d=1 specifying ranges of destination IP addresses. As the Internet evolves from being the best effort network as it is now to provide differentiated services, a filter may specify two or more IP header fields.

Figure 24 represents the problem of rectangle intersection, corresponding to the bi-dimensional representation of rules, e.g., IP source Address , IP Destination Address.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

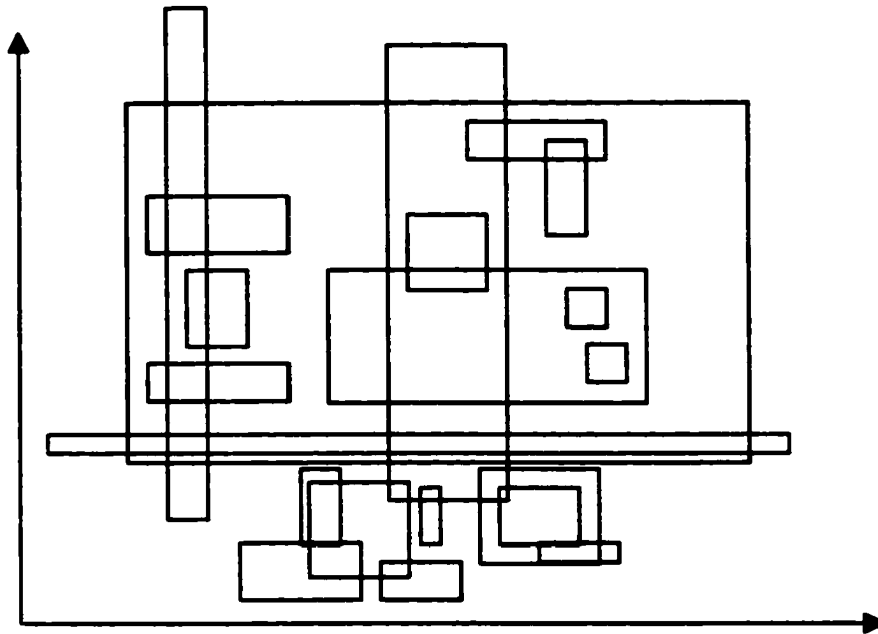


Figure 24* : *Rectangle intersection problem corresponding to bi-dimensional rules

Algorithm description

The validation algorithms are performed every time a new rule is created, or when an existing rule is modified or deleted. The consistency checking algorithm is used whenever a rule is to be deployed on a interfaceThis strategy ensures that the current pool of rules is conflict-free.

The main function called for the conflict detection algorithm is **CheckConsistency** (shown in Appendix 10.1). It takes as parameters the new rule and a list of existing rules already associated with that interface or device and it returns the vector of conflicting rules, if any. That list of conflicting rules will be used further for defining

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

additional solving rules for the areas of overlap. The list of existing rules can be a group of rules associated with a particular interface, device, domain or customer. This function uses another one, called **ruleConflict(newRuleName, ruleName)** that compares two rules for detecting conflicts based on overlaps. It takes as arguments the names of the rules, it gathers the objects from the storage locations and it compares them.

Two rules are conflicting if their packet stream conditions overlap, if their time period conditions also overlap, if they have the same priority and if the actions to be followed by each one of them are different. If one of them does not meet any one of these conditions, the checking is stoped. The algorithm implementation is shown in Appendix 10.1.

The comparison between two rules and the checking for overlaps follows a generalized intersection detection algorithm. It decomposes each rule into its components: Packet Stream Conditions, Time Period Conditions, and Actions. Each Packet Stream Condition is considered further as a collection (vector) of sets and ranges. (This structure will be defined later, within the PSCStruct class description).

For each pair of rules, the intersection of ALL Packet Stream Conditions within a rule is reported back and then the two resulting intersections from both rules are compared against each other. The algorithm that computes and returns the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

intersection among ALL the Packet Stream Conditions attached to a rule is described in the appendix 10.1.

The object returned by this function is also of type Packet Stream Condition.

The Time Period Condition attached to the rule is also decomposed into two ranges, the time range and the date range.

6.1.1.5 The generalized algorithm for Packet Stream Conditions intersection

Each packet stream condition object contains the following attributes:

- Source address
- source mask
- destination address
- destination mask
- protocol
- source port
- destination port
- DSCP
- direction

The algorithm that determines if two such objects intersect, as well as the algorithm that returns the actual intersection of such objects is based on the general range intersection or set intersection concept and it does not compare specific objects of this type with previously known values. For example, it does not take into consideration the fact that source and destination IP addresses are given in the form of four octets.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

On the contrary, it transforms all the attributes into ranges or sets. In this case, an IP address would be represented and acknowledged as an interval of long integers. For each given IP address, the beginning of the interval is obtained by computing a logical AND between the IP address and the corresponding mask, and the end of the interval is the result of the logical AND performed between the IP address and the maximum possible mask (which has all octets 1, or 255.255.255.255).

In a similar manner, a specific port number or ANY port number (represented in a Packet Stream Condition by the value 0), the direction attribute, specific protocol types or ANY protocol type, will be converted into sets. The following table describes the conversion of a Packet Stream Condition into ranges and sets.

Src Addr	Source Mask	Dest Addr	Dest Mask	Protocol	Src Port	Dest Port	DSCP	Direction
Range		Range		Set	Set	Set	Set	Set

The resulted collection of ranges and sets that represents the attributes of a Packet Stream Condition constitutes the class PSCStruct, describedin the Appendix 10.1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

This class takes an object of type Packet Stream Condition, it converts it into a collection of ranges and sets and defines the functions to be performed in order to determine whether two such collections intersect and what the actual intersection is, if any.

The function called "intersects" which takes as an argument another PSCStruct returns a boolean (TRUE if the current PSCStruct intersects with the PSCStruct given as a parameter and FALSE otherwise).

The function called "intersection" which takes as an argument another PSCStruct object returns another object of type PSCStruct that represents the actual intersection between the current PSCStruct and the PSCStruct object given as a parameter.

This class uses two different types of structures that correspond to the classes RangeStruct and SetStruct. Those classes define actions specific for range intersection and set intersection. The algorithm that performs set intersection is obviously different from the one checking range intersection. The two auxiliary classes are described in appendix 10.1.

The function called "intersects" which takes as an argument another object of the same type (parent class being CollectionStruct, for both ranges and sets) returns a boolean (TRUE if the current range intersects with the range given as a parameter and FALSE otherwise).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The function called "intersection" takes as an argument another object of the same type and returns another object of type CollectionStruct that represents the actual intersection between the current ranges the range given as a parameter.

The existence of a parent class, called CollectionStruct was necessary in order to combine both structures in the same object.

The class corresponding to the set structure and procedures is called SetStruct and it is described in Appendix 10.1.

The function called "intersects" which takes as an argument another object of the same type (parent class being CollectionStruct, for both ranges and sets) returns a boolean (TRUE if the current set intersects with the set given as a parameter and FALSE otherwise).

The function called "intersection" which takes as an argument another object of the same typereturns another object of type CollectionStruct that represents the actual intersection between the current set the set given as a parameter.
The challenging part of the sets intersection was a way to represent the infinite or universal set or the void set, as well as not producing duplicate elements within the resulting set (reunion or intersection).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The parent class, CollectionStruct is the following:

```
package com.policyarena.util;

public abstract class CollectionStruct {

  public void CollectionStruct() {  }
  public boolean intersects(CollectionStruct another) throws Exception{
   System.out.println("This is CollectionStruct's method");
   return false;
  }

  public CollectionStruct intersection(CollectionStruct another)throws Exception
{
  return null;
  }
  public void out() {
  }
}
```

6.1.1.6 The generalized algorithm for Time Period Conditions Intersection

Each Time Period Condition object contains the following attributes:

- Start Date
- End Date

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- Start Time
- End Time

The algorithm that determines if two such objects intersect, as well as the algorithm that returns the actual intersection of such objects is based on the general range intersection concept. It does not compare specific objects of this type (Time Period Condition) with previously known values (of type date or time, available in Java). The algorithm transforms all the attributes into ranges, one range for the date interval and one range for the time interval.

The date intervals are ordered, but the time intervals can be defined with a Start Time value lower than the End Time value. For this particular case, instead of having one range for the time range, the time attributes of the Time Period Condition will be converted into a reunion of two ordered ranges.

For example, if the Start time is 11 p.m. and End Time is 9 a.m., then this time interval will be converted into two ranges: 11 p.m. to 12 p.m. and 0 a.m. to 9 a.m.

Also, another challenge in determining intersection among these types of objects is the fact that some time intervals can start at one particular date and to continue in time to the next day, which is not specified in the date attributes. Therefore, the first ending interval (time or date) must take effect and override the other interval that is not finished according to the Time Period Condition Definition.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

For example, if one rule is valid from March 1st to March 3rd, between 9 p.m. to 1 a.m., the rule must become invalid on March 3rd at 12 p.m., although the time validity ends at 1 a.m., March 4th.

In addition, a default Time Period Conditions should be defined for "ALL THE TIME" specification. The Start Date and Start Time attributes in this case will be the current date and the current time when the Time Period Condition was created. The End Date and End Time attributes are specified as being definite values, much later in time.

The following table describes the conversion of a Time Period Condition into ranges:

Start Date	End Date	Start Time	End Time
Range		One or two Ranges	

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The collection of ranges resulted after converting the date and time attributes of a Time Period Condition into long integers constitutes the class TPCStruct, described in Appendix 10.1.

This class takes an object of type Time Period Condition, it converts it into a collection of ranges and defines the functions to be performed in order to determine whether two such collections intersect and what the actual intersection is, if any.

The function called "intersects" which takes as an argument another TPCStruct returns a boolean (TRUE if the current TPCStruct intersects with the TPCStruct given as a parameter and FALSE otherwise).

The function called "intersection" which takes as an argument another TPCStruct object returns another object of type TPCStruct that represents the actual intersection between the current TPCStruct and the TPCStruct object given as a parameter.

This class uses one type of structure that corresponds to the class RangeStruct, described in a previous section (6.1.1.5).

6.1.1.7 Future work

The validity-checking algorithm can be improved by adding a set of checks to ensure that the resources needed by a policy, in isolation from all other local policies, are available in the devices to which this policy applies. For example, suppose that a policy requires that a certain set of paths through the network provide a certain specific queuing behavior. Suppose further that on one of the paths at one of the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

interfaces, no advanced queuing mechanisms are available. This would mean that the needs of the policy are not satisfied. Thus, the policy itself is not satisfied, implying that this policy cannot be implemented in these devices. Also, the role of each interface could be validated (e.g. policing for ingress, buffering/queuing for egress, etc), for a more intelligent automation of the logical provisioning system.

Policy interpretation refers to the process of evaluating the policy in the context in which it is to be enforced. Consequently, another constraint to be taken into account could be semantic checking from an interface perspective. This could be done in terms of supported mechanisms and admitted configurable values, resources needed for rule implementation, and number of rules supported by each interface.

Feasibility checking could be performed to compare the available services of the network with respect to the full set of policies that want to use those services. Feasibility checking will most likely require post-policy deployment checking that is sensitized to the particular network elements involved as well as the nature and effects of the deployed policies.

Also, event notification is required for the network devices once the policy server is active and the policies are deployed.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.2 Validation of Well Formed Sequences of QoS Mechanisms

Validity checking is done at various levels, in order to detect errors before the policy rules are redirected, translated into device specific commands or used in any other ways, which could make the cause of error almost impossible to identify later.

The first type of checking performed is syntactic checking, which includes validation of data types of a policy's terms (described in section 6.1.1.1). In addition to that, semantic lookup must be done (i.e. whether the mentioned IP address exists, if the source and destination addresses are overlapping, etc.) each time a rule is created or modified. This way, the insertion of valid attributes for describing policy rules is guaranteed.

Each QoS mechanism is represented by an action attached to a rule, which is performed if the conditions attached to the same rule are met. Each object of the type Policy Action has at least one attribute called Next Action, unless the Action is a Concluding Action. The Next Action is the mechanism that follows after the current Action. Some Actions have just one Next Action, others have 2 Next Actions. The following figure shows the structure that forms a sequence of Actions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

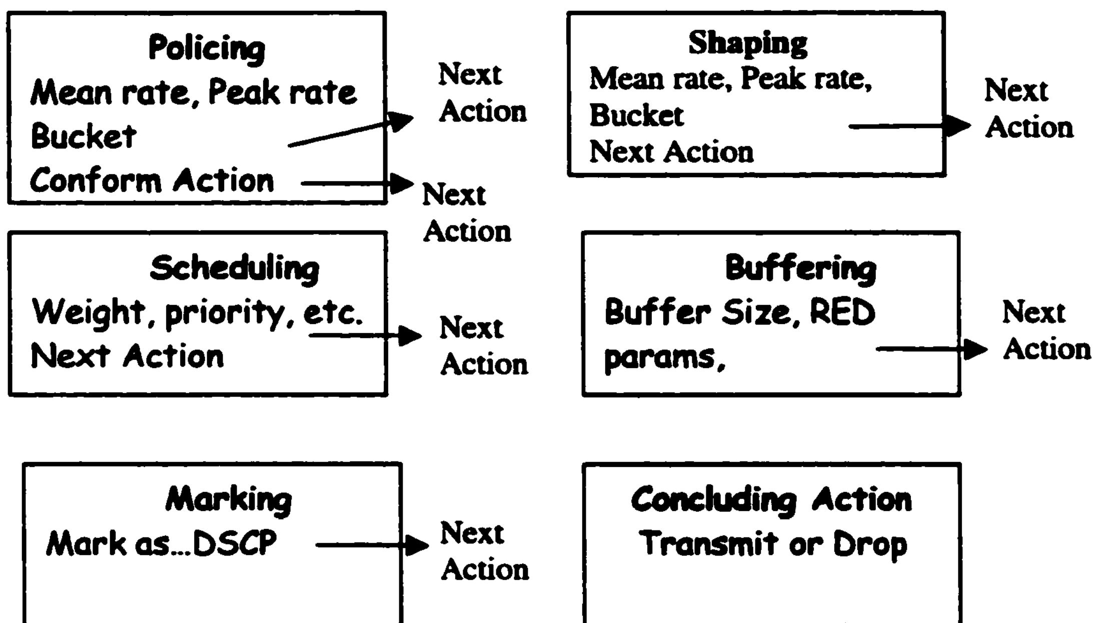


Figure 25 : The Structure of Sequences of Actions

The next two figures show examples of more complex sequences of Actions and their associated parameters:

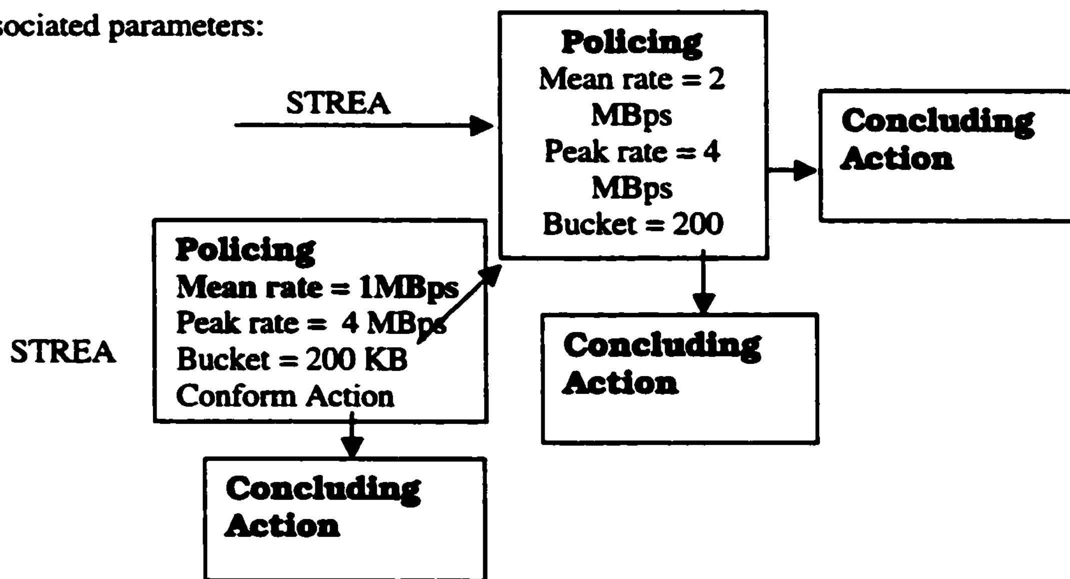


Figure 26: Policy Action Sequence: Example 1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

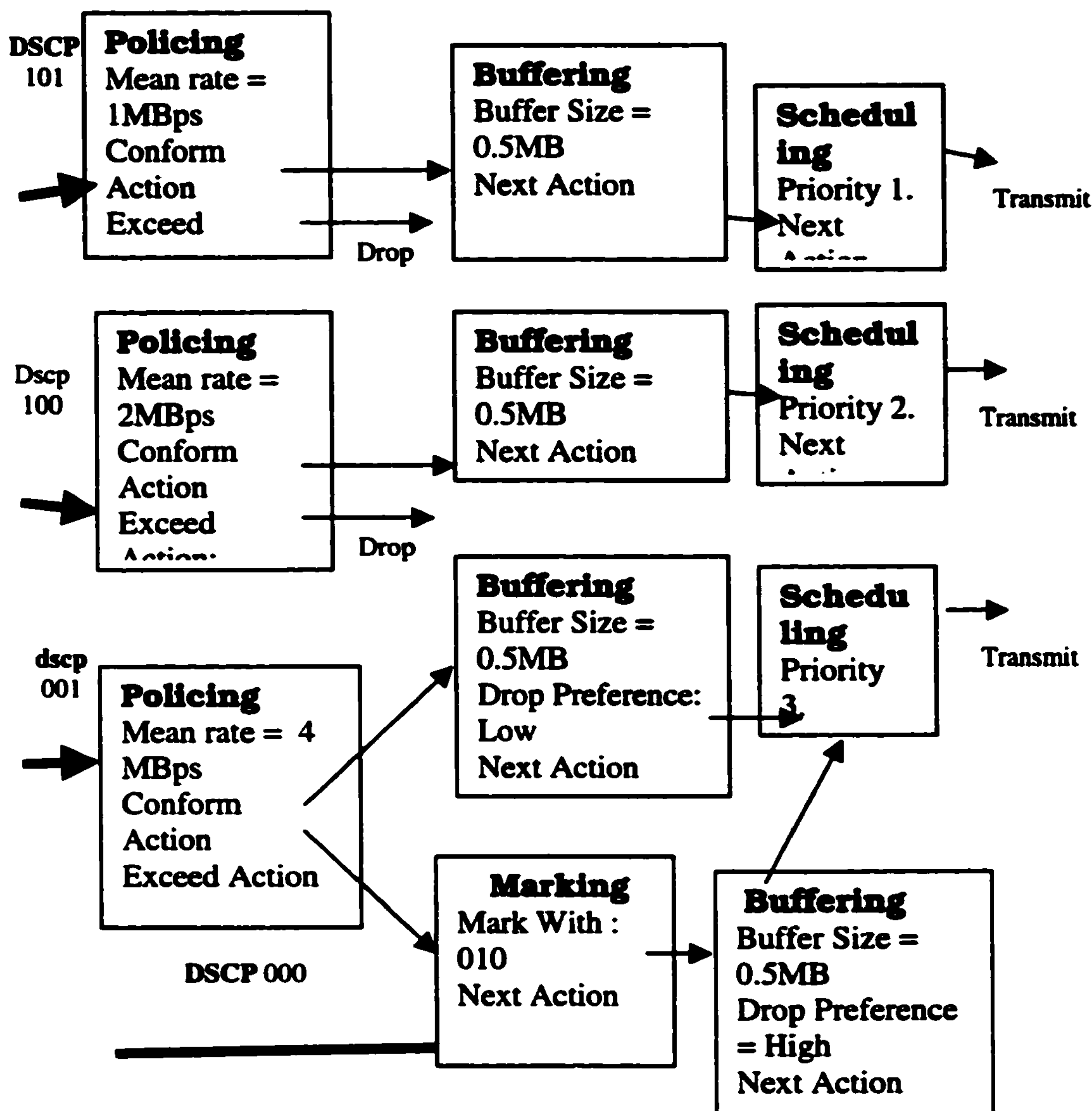


Figure 27 : Policy Action Sequence: Example 2

A special feature of the capability checking module is the ability to ensure well-formed sequences of actions within policy rules. This is done by using a meta-representation language within a multi-perspective environment for a class level checking combined with a loop detection algorithm implemented at the object level.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

These algorithms use graph representation (and the associated adjacency matrix) of accepted precedence relations among actions, as shown in Figure 27.

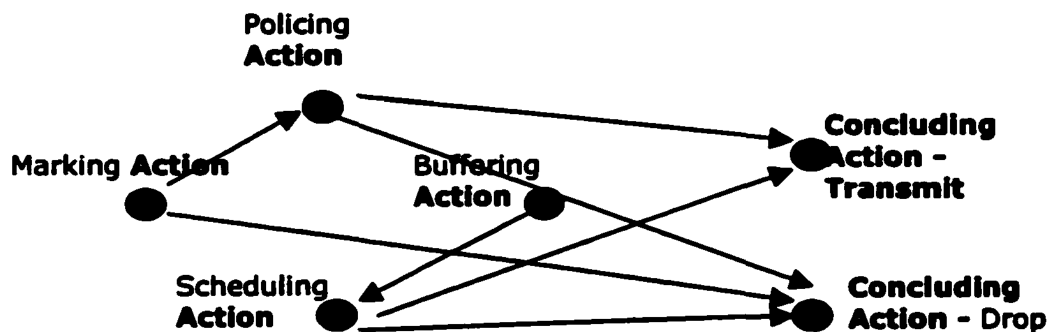


***Figure 28**: Well formed sequences of actions represented by paths in a directed graph*

It can be noticed that nodes representing a concluding action are always sinks, as no other action can follow them.

The sequences of policy actions associated with the rules represent mechanisms to be applied in a distributed fashion on the network. An ill-formed sequence of actions that violates at least some constraints on well-formedness, can create a deadlock on the network, with severe consequences and with causes almost impossible to trace or detect at the time when they are deployed into the network devices.

This semantic checking module regarding well-formed sequences of actions is called ***capability checking*** and it acts like an expert ruler. It is performed each time a new action is created, modified or deleted. One part of it consists of checking whether one action can be followed by an action of a different type. This type of checking is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

concerned with the type of policy actions, the type of object. Another part of it consists of checking the sequence of action for the purpose of detecting one-step and multiple-step loops or cycles in some particular instances of action sequences. This validation is concerned with object instances, not with the types of actions and the following figures explains in an example the difference between the object level and the instance level of policy actions.

The following figure shows sequences of actions that contain one-step cycles and multiple-step cycles.

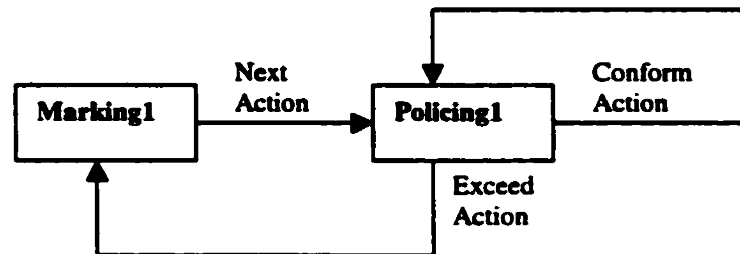


Figure 29 : One-Step and Multiple-Step Cycles in Sequences of Actions

Although an action of type marking is allowed to follow an action of type policing, in this case the sequence forms a **two-step cycle** that must be detected and prevented from being deployed into the network. The reason behind it is the fact that the instance of the marking action that follows the action called Policing1 is the same as the instance of the marking action that precedes the action Policing1. Also, although an action of type policing is allowed to follow another action of the same type, in this case this thing is prohibited, because the action instance called Policing1 forms a **one-step cycle** with itself, as the conformant action.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.2.1 Validation of Well-Formed Sequences of Actions – Algorithm Description

The functions that check the validity of policy action sequences are called each time an object of type policy action is created, modified or deleted. This is because by removing a link in a chain of actions, its semantic might be changed to an invalid one. For example, although a function of type marking can be followed by another function of type marking at a distance greater than two, it makes no sense to have a marking function followed directly by another marking function. Thus, by removing the action between the two marking actions, the sequence of actions becomes invalid.

The functions written for the purpose of checking the validity of sequences of actions are described in Appendix 10.2.

This function, called verify action attachment parses the sequence of actions after getting the names from an intermediary tool (called ObjectGateway) that knows how to retrieve actual objects from the storage based on their name.

This function receives as arguments the names of two actions that are linked or that are to be linked together. In the list of arguments, the **dsActionTo** is the next action for the **dsActionFrom** action.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The first thing checked on that link is the existence of one step cycles and the semantic validity in one step, and this is done by calling the function **checkForwardOneHop,** described in the next section.

Then all possible next paths from that particular action are checked forward one step and multiple steps, through a recursive mechanism. The function that describes the multiple-step checking is called **checkForwardManyHops** and it validates the sequence of actions for the existence of cycles, as well as the semantic meaning.

With regard to the graph representation of the sequences of actions, checking whether one action A of one type X is allowed to follow in one step an action B of a different type Y is equivalent to checking whether the existence of an arc from the node X to the node Y, where X and Y are types of actions, is allowble.

Checking whether one action A of one type X is allowed to follow in multiple steps an action B of a different type Y is equivalent to checking whether the existence of a path from the node X to the node Y, where X and Y are types of actions, is allowable.

The adjacency matrix corresponding to the capability graph mentioned earlier in this section is defined in the ToolsGlobals class as follows:

```
public static int[][] AdjActionAllowed =
// OTHER   POLICING MARKING QUEUING   RESHAP  SCHED   DENY    PERMIT
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
OTHER*/  {{WARNING, WARNING, WARNING, WARNING, WARNING, WARNING,
WARNING, WARNING},
POLICING*/ {WARNING, OK,   OK,   OK,   FAIL,  FAIL,  FAIL,  OK},
MARKING*/  {WARNING, FAIL,  FAIL,  OK,   FAIL,  FAIL,  FAIL,  OK},
QUEUING*/  {WARNING, FAIL,  FAIL,  FAIL,  OK,   OK,   FAIL,  OK},
RESHAPING*/ {WARNING, FAIL,  FAIL,  FAIL,  FAIL,  FAIL,  FAIL,  OK},
SCHEDULING*/{WARNING, FAIL,  FAIL,  FAIL,  FAIL,  FAIL,  FAIL,  OK},
DENY*/    {WARNING, FAIL,  FAIL,  FAIL,  FAIL,  FAIL,  FAIL,  FAIL},
PERMIT*/   {WARNING, FAIL,  FAIL,  FAIL,  FAIL,  FAIL,  FAIL,  FAIL}};
```

If an element a_{xy} in the adjacency matrix has the value "OK", then an action of type x can be followed by an action of type y. If an element a_{xy} in the adjacency matrix has the value "FAIL", then an action of type x cannot be followed by an action of type y. If an element a_{xy} in the adjacency matrix has the value "WARNING", then an action of type x can be followed by an action of type y, although it might not be useful for real network environment.

Also, an action of type Policing can have two next actions, one conform action and one exceed action. Even though both paths would be successors of a policing actions, some action types are suited for being a conform action but not an exceed action and the other way around. That is why for this particular type of action, Policing, two different next action validity sets were created, one for the action types suited to be a conform action and another for the valid exceed actions. These sets are constants in the ToolGlobals class and are defined as follows:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
public static int[] ConformActionAllowed =
    {MARKING, QUEUING, SCHEDULING, CONCLUDING_DENY,
CONCLUDING_PERMIT};

public static int[] ExceedActionAllowed =
    {MARKING, CONCLUDING_PERMIT, CONCLUDING_DENY};
```

The function **VerifyFirstAction** (shown in Appendix 10.2) checks the validity of starting one sequence of actions with a particular action. It might be the case that even though a particular succession of actions is valid, when it is attached is some point of the chain to a policy rule, it becomes invalid because it might not be reasonable for that rule to be the first action performed by a rule. The function is called each time a rule is associated with an action.

The actions that can be the first ones attached to a rule are enumerated in a set and the following function scans through that set for validity. That set is defined as a constant in the ToolsGlobals class as follows:

```
public static int[] FirstActionAllowed =
    {MARKING, POLICING, RESHAPING, CONCLUDING_DENY,
CONCLUDING_PERMIT};
```

In the case when a particular action cannot be attached to a rule as the first action in a sequence of actions, an exception will be thrown with the following message:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

"FAILED: Action *actionName* cannot be first action"

6.2.2 Cycle Detection in Sequences of Actions – Algorithm Description

The function **checkForwardOneHop** (described in Appendix 10.2) receives as arguments the name of the two linked actions, the predecessor and the successor actions. The first thing checked for validity is the one step cycle, which corresponds to the situation when both the predecessor and the successor actions are identical. If that is the case, an exception is raised, with the following error message:

" One step cycle. Action: *action name* cannot follow itself "

The next step in the capability checking is the semantic validity. The action types of the predecessor and successor actions are identified through the Object Gateway tool and then, the adjacency matrix defining admissible combination of rules is accessed. If the action types are incompatible (which corresponds to the case where in the adjacency matrix there is no arc from the predecessor's type to the successor's type of action), an exception is raised, with the following message:

" Action of type *predecessor action name* cannot precede an action of type *successor action name*"

The following function performs the many step validity checking of a sequence of actions:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The function **checkForwardManyHops** (shown in Appendix 10.2) receives as arguments the name of two directly linked actions, the predecessor and the successor actions. The function checks backward and forward the validity of the action sequence, using a recursive algorithm.

The function checks first whether in the sequence of actions there are potential many step cycles created by the association between these two actions. This checking corresponds to identifying the same instances of actions previously included in the sequence. If that is the case, an exception is raised, with the following error message:

"Many steps cycle. Action *predecessor action name* **is forming a cycle with action** *successor action name"*

The next step in this function is the semantic validity. The action types of the predecessor and successor actions are identified through the Object Gateway tool and then, the adjacency matrix defining admissible combination of rules is accessed. If the action types are incompatible (which corresponds to the case where in the adjacency matrix there is no path from the predecessor's type to the successor's type of action), an exception is raised, with the following message:

" Action of type ***predecessor action name*** **cannot precede an action of type** ***successor action name"***

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

7 Translating Policy Rules into Device Specific Commands

The ***policy rules deployment*** is the action of placing the network (or a part of it) in a desired state using a set of management commands. When this definition is applied to network elements, these management commands change the configuration of the device(s) using one or more mechanisms.

7.1.1 Levels of Policy

In this model, policies exist at four levels of abstraction – service level policy, network level policy, role level policy and device specific policy. In order to translate network administration requests into formal policy definitions and consequently into detailed device actions, the use of a Policy Definition Language will be involved. This requires translation to the vendor-specific device configuration language.

Service Level Policy systematically describes the properties of a particular QoS service. It is a template designed to systematically detail and sort service attributes into *provider-specified attributes* and *customer-specified attributes, and restrictions on what constitutes a valid service request.* In other words, the service template is best understood as a collection of attributes, which can be customized to give rise to the parameter set that describes a particular service. Such a template for service description may specify attributes like the service name (i.e. VLL), a textual description of it (i.e. "Low loss, point to point service"), the customer name, the source and destination end points, the peak rate, etc.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Network Level Policy is used to describe an instance of a customer request. Thus, a network level policy is created out of the template that describes the service, from which the useful information is extracted for the purpose of QoS Provisioning. We also use the term *Virtual Provisioned Pipe (VPrP)* to describe a network level policy. A network level policy is created, modified or deleted, whenever a customer subscribes to a particular (instance of) service, or unsubscribes from it.

Role Level Policy is a description of the changes required on the configuration of a router in order to implement the network level policy (i.e., to instantiate a VPrP),. *It is important to note that the role level policy depends only on the ROLE of the router in implementing a particular Network Level Policy (an access role vs. a backbone role, etc.), and not the implementation of the router (Cisco router vs. a Nortel router).* Role level policies have the familiar – IF <conditions> THEN <actions> -- format standardized in the IETF and are the targets for compatibility and consistency checking, as well as resource availability lookups, in an implementation independent manner.

Device Specific Policy is a systematic way of documenting the configlets that are to be downloaded on the routers. It helps the translations from higher level policies to device specific commands to be made in a consistent manner across the network.

Those levels of abstractions can be managed and realized by designing different management tools and implementing them as Java packages that can be flexibly

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

instantiated at any server in the system. They will encode the logic required to create, modify and delete different objects, mediating between users (client applications) and the policy system.

Policy based networking (PBN) is an approach to scalable management networks, characterized by a distinct policy-based network management layer, distinct from the data transmission layer. The policy layer: allows administrators to configure services using policy rules, parses and checks configurations for correctness and consistency, distributes policies to various network devices, and monitors their enforcement.

To perform these roles, the policy layer includes a set of functions that provide monitoring, configuration and control of both the attributes of services and the policies governing them.

The policy layer captures users' requirements, device independent configuration, aggregates device-state information, coordinates event notifications across various sub-systems, and in some cases "closes the loop" between monitoring and control.

The policy layer defines and manages **policies,** which "set the rules" determining how the network service is implemented, and how resources are given to users, rather than simple configuration.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

One of the major advantages of a policy-based networking approach is to reduce the degree to which centralized network management applications need to have direct, real-time control over the operation of the network. The application of policies to the network allows the management applications to set the rules by which the network and services must function, and how the network must respond to events. Embedded functions within the network (such as routing, AAA, etc.) may then respond to events and to network states, by taking actions in accordance with pre-set policies, without interaction with management applications.

The following design principles are prominent in the PBN effort:

- Allowing the same management information to be used across different kinds of network elements. An administrator will manage a vendor independent model of a router rather than having to manage the explicit details of each vendor's router in the system. For example, vendor A's and vendor B's router will need different configuration details to achieve the same behavior.
- Allowing administration of multiple elements at once, by grouping devices together, thus removing the burden of manipulation of individual elements each time a management change is to be made. The importance of facilitating self service administration to users, while hiding (to whatever extent needed) device and protocol configuration.
- The use of *policy rules* -- atomic injunctions used to control the dynamic behavior of the managed system. Policy rules may exist at different levels of abstraction,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

from high service or network level directives to lower device or interface level rules. The policy layer automates translations and integrity checks between different levels of abstraction, as needed.

7.1.2 Policy Functions

It is generally agreed that implementing network policy involves three main functions:

- *Decision-making*—evaluating the policy to determine which actions need to be applied.
- *Enforcement*—implementing a set of actions, usually control and management actions, in response to the state of the network, to bring the network into the desired state.
- *Policing*—ongoing examination of the state of the network for checking network health, whether policies are being satisfied, and whether clients are taking unfair advantage of network services.

The following figure shows how policy management could be implemented within one policy domain. The roles of the various elements and functions are described below. This is a simplified picture, which may suit certain contexts.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

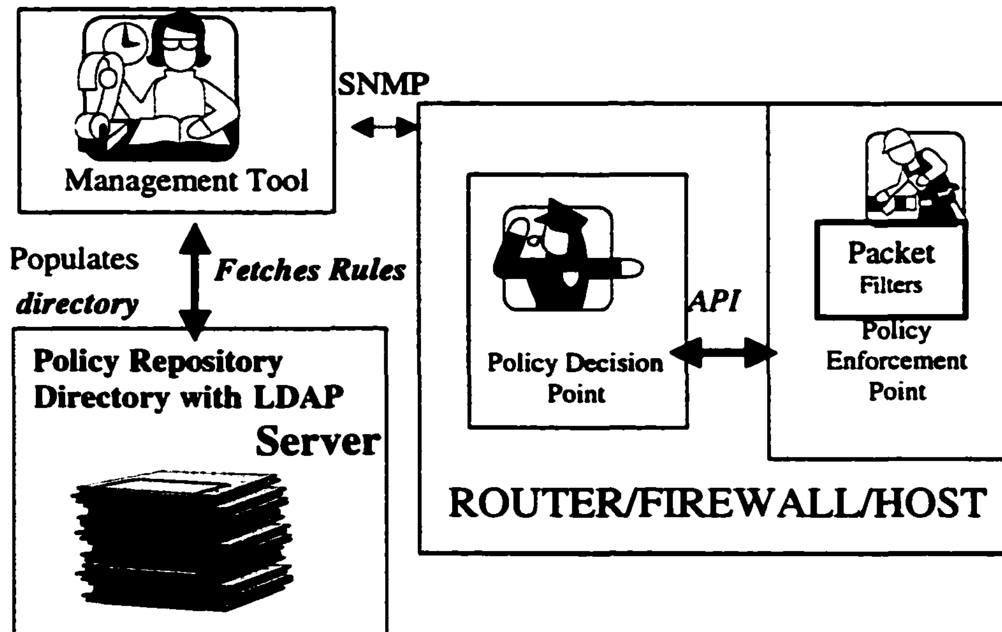


Figure 30 : PBNM Functions

In this diagram:

- The Policy Administration Point receives a request to create, modify, or delete a network policy. This would require an authentication and authorization check, but it is assumed that this will be performed by a separate component of the architecture. The Policy Administration Point constructs a formal statement of the policy, according to the policy schema, and updates the policy repository with the new or updated policy.

- The Policy Administration Point may also (a) attempt to detect any conflicts in the policy before placing it in the database, (b) send an interrupt to the relevant policy servers to alert them to the presence of a new policy, and may send a copy of the policy directly.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

policy servers to alert them to the presence of a new policy, and may send a copy of the policy directly.

- The policy server, either immediately or at a scheduled time, retrieves the policy from the database.
- The policy server then needs to interpret the policy in the context in which it is to be applied. This function may be null, but may require the combining of the static aspects of policy (as described by the schema) with other network information (such as topology, network state information, time, etc.). The result may be termed a Policy Lease, which will be valid for a certain period.
- Policy decisions will be made in many cases in response to network events, which might be caused by the receipt of a signaling message, a user dialing into the network, some change in state of the network, and other factors. In this case, the network device (on which the policy enforcement function resides) will query the policy server for a decision to be made. This query would be communicated by means of a policy protocol. The policy server would then run a process to make the policy decision, which may involve invoking external functions, such as user authentication and network admission control. The result of this decision process will be a set of actions as described in the policy. The policy server would then respond to the network device (policy enforcement point) with the desired actions to be taken.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- Alternatively, the policy server could evaluate the policy and determine the necessary actions, as a configuration mode, without the prompting of network events.

Policy management is likely to address a number of policy areas (QoS, routing, security, etc.), but a single picture for policy management that covers all cases is unlikely. The simple framework picture described in Figure 29 does not take into account all policy-enabled network functions. In summary, the following items note a number of exceptions and additions:

- The Policy Decision Point (PDP) and Policy Enforcement point functions may reside on the same device.
- Policy Decision Points may not be "policy-aware." Therefore, proxy and/or translation capability is required.
- Proxies may be required for timeliness and scaling. Proxies could act as Policy Decision Points, or could act simply as distributors and/or translators of policies.

The Policy Administration Function is a logically centralized entity that has control over the definition and installation of policies. It acts as a hub for interaction between the Policy Servers and the clients. The Policy Administration Function also includes a number of specific functions including:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Policy Definition. The primary task of the Policy Administration Point is to translate network administration requests into formal policy definitions. In many cases, this will involve the use of a Policy Definition Language.

Global Policy Conflict Detection. Policy conflicts may occur when the conditions of two or more policies can be satisfied simultaneously, but the actions of at least one of the policies cannot be executed simultaneously (see section 5).

Initial Admission Control. For many simple policies, conditions may exist under which it is not possible to implement the policy. Examples include the application of a policy for provisioned QoS, to be scheduled for a future time. However, this might be excluded due to lack of resources, knowing the sum total of scheduled QoS requests.

There is a fine line between policy management and configuration. In this context, the only distinction drawn between the two is that the configuration is for immediate and long-term application to the network. Configuration may be implemented directly onto the network element. However, it is essential that coherence be maintained between direct configuration and policy, in the same functional area. For this reason, it is likely that direct configuration is implemented such that the Policy Administration Function is in control of both policy and configuration.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The Policy Server is a vital part of a policy framework. In response to resource requests from the network elements, network events, or certain network conditions, the policy server is responsible for retrieving the policy and interpreting it in the context in which it is to be enforced. In many cases, the Policy Server is responsible for making policy decisions (that is, it also acts as a Policy Decision Point) by evaluating the policy and deciding which actions must be taken. It complements the policy database by adding statefulness and applying the policy to the deployment context.

In general, there will be multiple Policy Servers, which may be application-specific, and there are a number of functions involved, which are usually implemented on the Policy Server. However, the specific functions resident on a policy server are application-dependent. The functions that may be required on a policy server are:

Policy Retrieval. The Policy Server must locate and retrieve the relevant policies from the policy database. The protocol used for this is LDAP, and that the Policy Server has the capability of sending unsolicited queries to the directory to retrieve new or updated policies. To support this, the information model must enable flexible referencing of policies, so that policies can be located from a number of "views." Policy servers, for example, may need to locate policies based on topology, user or group identifier, policy type, service type, etc. For "legacy" devices, the Policy Administration Point must trigger the Policy Server to query the directory server.

Policy Interpretation. Policy interpretation refers to the process of evaluating the policy in the context in which it is to be enforced. To do this, the Policy Server must

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

combine the static policy definition with other information required for evaluating the policy. This includes:

- parameters of the event or request that triggered the policy decision
- other static information referred to by the policy, for example, user or device profiles
- timing information (for example, from NTP sources)
- network state information, for example, performance and utilization statistics

Policy Caching. Having interpreted the policy by taking into account contextual parameters, the Policy Server needs to cache the instance of the policy for future decisions. Such a cache would need to have a lifetime, and would also be capable of revocation, if network conditions changed.

Scheduling. Many policies will involve a scheduling component for activation and deactivation. The Policy Server may need to operate a scheduling function to manage such policies.

Policy Decision Process. The Policy Decision Process may be resident on the policy server, or on the network device, or split between the two. The Policy Decision Process determines the actions that are required, using the policy as input together with other inputs to evaluate the policy conditions. Policy Decisions may be prompted by Policy Decision Requests from network, or may be server initiated (for example, scheduled deployment of a QoS profile). The former will generally require a

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

response in real time. In general, the Policy Decision Process will be application dependent, and will not be wholly determined by the policy conditions.

Policy Translation. In some cases, the Policy Server is acting as a Policy Decision Point (PDP). In this case, it must translate the actions prescribed by the policy decision into detailed device actions the Policy Enforcement Point (PEP) must enforce. This will often involve translation to the vendor-specific device configuration language.

Policy Enforcement Point. The Policy Enforcement function describes the execution of actions or configurations that have been prescribed by the policy process. The Policy Enforcement Point (PEP) function is typically implemented on a network device in the data path. Typically, such a device would not be "policy capable" since it cannot "understand" policy language, and it does not have the capability of polling a directory server. The PEP must capture the actions determined by the Policy Decision Point (PDP), and enforce those actions in the network.

7.1.3 Toolkit

The prototype has a set of tools that expose the various aforementioned policy levels. The tools running on the *Policy* server are responsible for various tasks like service level policy *management*, network level *management*, device level management, and user *management*. These tools provide functionality to capture, interpret and manage policy and to manage customer and network data.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- **Service Administration Tool** - enables the provider to configure service templates for Qos services they are willing to offer. The service templates include a validity template, service endpoint template, and a QoS template. It also enables the provider to instantiate specific service instances if it so desires;

- **Network Administration Tool** - enables the provider to configure a network topology, and to assign control and monitoring agents to devices.

- **Organization Administration Tool** - enables the provider to configure organizations, sites, SAPs, users, user roles and control users' access to tools;

- **Device Configuration Tool** - enables the provider to configure policy rules or role level policies. This includes the ability to manage conditions, policy actions, policy rules, to attach conditions and actions to rules, to attach rules to router interfaces and/or interface roles, and to deploy policy rules to affected devices.

- **Object Gateway Tool** - is a distributed object-oriented view that implements the Common Information Model (CIM) proposed by DMTF. It provides a controlled point of access to the persistent data and enables remote access to data through Java RMI (Remote Method Invocation). It is the module that enables product integration and selects the interfaces that are made public.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The goal of the object gateway is to hold in memory object representations of some persistent data, to create new persistent data, to provide a single controlled point of access to this persistent data (although this control might be implemented in the data gateway that it uses rather than in the implementation of these interfaces) and enable remote access to data through traditional O.O. distributed systems such as Java RMI or CORBA.

- **Data Gateway** - is a simple and distributed data adapter API that allows the object layer to interact with any set of databases and/or directories that are used to store the data. The Data Gateway offers a simple data-driven, rather than functionality driven, API (get, set, create, delete and search) to access entities. An entity is modeled as a set of attribute-value pairs and represents a set of related data. This approach allows easy extension to information accessed without change to the API. The Data Gateway is implemented using an LDAP directory to hold meta information about each entity. Hence the role of the data gateway is to handle the heterogeneity of data sources, locations, and data access drivers. The goal is to allow relocation and integration of data (from physical locations to storage type) through configuration of the meta-directory, rather than modifying the implementation of the object gateway and re-compiling.

PBNM manages several entities, including provider, customer, policy and virtual link. Each entity consists of attributes. Some of these attributes may be optional. In addition, new attributes may be associated with entities over time. Entity

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

attributes belong to two categories: static and dynamic. The values of static attributes remain the same for long periods of time, e.g., customer name. The values of dynamic attributes change over time, e.g., number of bytes sent from one point to another in the context of a virtual pipe. Static attributes are stored in an LDAP directory. Dynamic attributes are stored in a database management system or a file system, depending on the transactional characteristics of these attributes.

The role of the data gateway is to handle the heterogeneity of data sources, locations, data access drivers on behalf of the object gateway, this might be implemented using a meta-directory that stores the manner (as a String representing a parameterized SQL statement for example) by which a particular data entry is extracted and introduced into the persistent data store(s). The goal of it is to allow relocation of data (from physical locations to storage type) through configuration of the meta-directory rather than modifying the implementation of the object gateway and re-compilation. The interface it offers to the object gateway comprises a get/set/create/remove method that require the fully qualified name of the data that is to be manipulated to be able to locate it, a single set of this methods is used to access all different data types defined in the common information model that Policy manages.

For each PBMN entity, a meta-entry describing this entity exists in an LDAP directory. The meta-entry contains information about the storage requirements of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the various attributes associated with this entity. In addition, the meta-entry contains information about the location of an entity instance in an LDAP naming context.

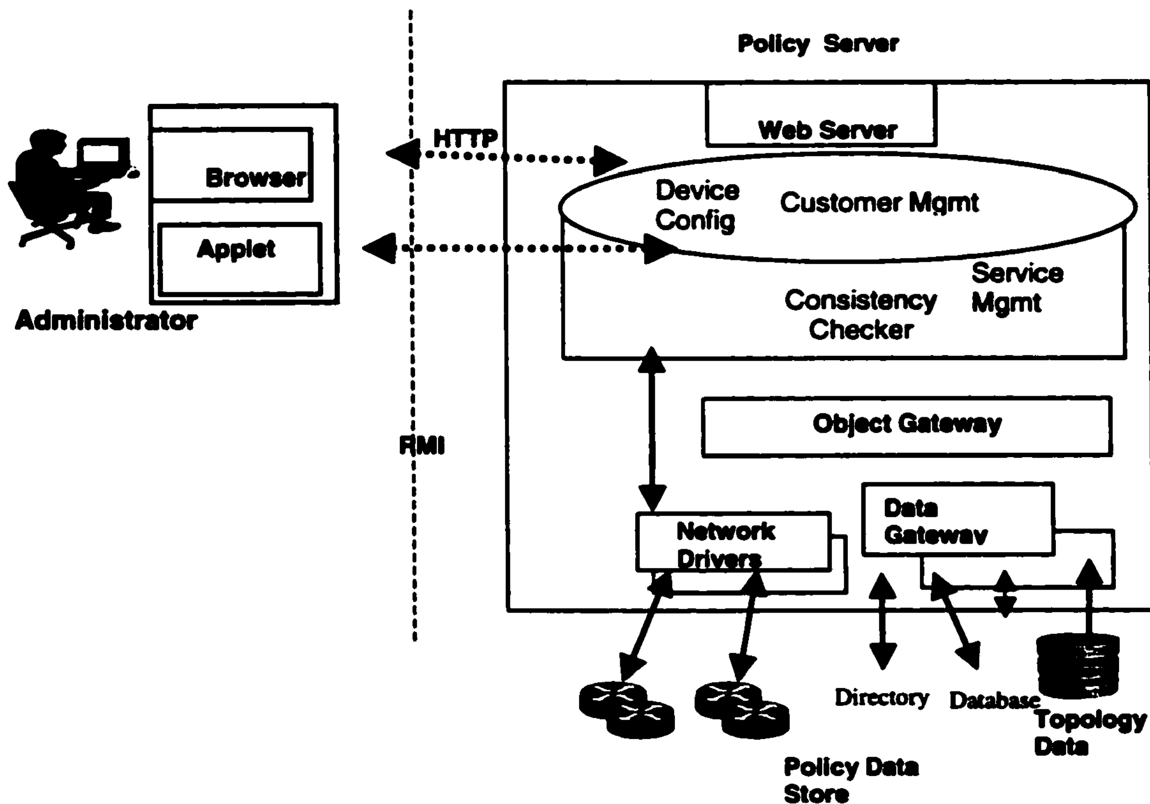


Figure 31 : Policy component view

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

7.1.4 System Operation

This section describes how a customer's request for QoS connectivity between two or more sites is handled within *Policy* server.

A customer uses a web browser to login to the *Policy* server. Once the authentication phase is over, the customer then selects the Service Administration Tool to create a new QoS connectivity between two or more sites. In this phase the customer needs to enter the parameters particular to their service level policy request (e.g. class of service, source and destination site names, application and bandwidth required). The completed request is then translated into network level policy rules.

Each interface of a device has an associated role that it plays in the network (e.g. provider-edge-ingress). The request, together with the topology information is refined into a set of role level policies for each interface in the QoS service pipe. These rules are now service independent and are ready for deployment. The role level policies are deployed to each agent as appropriate to the role of the router interfaces they manage. Upon receiving a new policy the agent performs conflict analysis that checks if the set of policies that apply to this device are mutually consistent (see section 5). If conflict is detected, the rule is rejected and the whole request is rolled-back. The agent then performs further checks to ensure the device it manages is capable of supporting the new rule. For example, the queuing mechanism on a router may not be capable of supporting all configurations that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

can be described by a policy rule. Finally the role level policy is translated by the appropriate network driver into device specific commands and downloaded to the device using the appropriate management protocol (Telnet/CLI for now). If the Policy server successfully completes all the above steps, a confirmation is returned to the customer.

7.1.5 Common Information Model

This section presents information model for policy objects. It also describes the service-level, network-level and role-level policy classes. In this sub-section the information model is presented generally, without reference to the particular set of services that we deploy.

7.1.5.1 Service Level Policy Information Model

The service provider makes available a number of *service templates* to the customer depending on their type of service (Real Time Assured, Assured, etc). The customer chooses a particular template, (the customer-care agent) fills it out, and submits the request to create a new network policy (i.e., a VPrP) or to modify the attributes of a pre-existing one. Each template consists of (a) a set of attributes pre-specified by the provider, (b) another set to be specified by the customer, and (c) restrictions on what constitutes a valid service request. In other words, the service template is best understood as a collection of attributes, which can be customized to give rise to the parameter set that describes a particular service.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The central class in the service template information model is the *Template* (see figure below) which contains *ValidityTemplate*, *QoSTemplate* and *TrafficTemplate*. *ValidityTemplate* describes the time periods when the service applies, *QoSTemplate* describes the DSCP, delay, loss, jitter and excess traffic treatment parameters that characterize the service, while *TrafficTemplate* details the endpoints of the service together with the amount of traffic that emanates from each end point. The information objects are all customizable, as explained below.

Service Menu: The service menu is a grouping class for templates. It contains the customer specific request.

Validity Template: The validity template contains information regarding the period of time during which the service is to be instantiated. While the customer usually specifies the time period information, there are occasions where the provider may wish to restrict or specify these attributes. The validity template allows the service can be instantiated for a certain period of time (eg. 11/09/99 1000 EST to 11/11/99 1200 EST), or for regularly recurring intervals of time (Mondays 0900 EST to 1700 EST). The Validity Template has a number of attributes – *TimeOfDay* (e.g., 0900 to 1700), *DaysOfWeek* (Mon,Tues,Fri), *MonthsOfYear*, *StartTime (0500EST June 19, 2000)* and *EndTime*[1]. Note that all these parameters are in the *NameDescTagString* format. This format allows the provider customize this object further. For instance, if for the VLL service, we only wish to permit requests that are non-recurring (eg. 11/09/99 1000 EST to 11/11/99 1200 EST.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ServiceEndPoint Template: The ServiceEndPointTemplate objects describe the topology of the service instance, using attributes that relate to the number and nature of the end points. An end point is a notion that generalizes a site, a Service Access Point (SAP) or an Internet gateway router. An end point may be categorized as a SOURCE of traffic, a DESTINATION or consumer of traffic or BOTH. By controlling the maxServiceEndPoints attribute, and perhaps by specifying some of the end points, the provider controls the topology of the service. The customer is allowed to specify end

points together with the amount of traffic that is offered from each end point (in case the end point is a SOURCE or BOTH) or destined to an end point (in case it is a DESTINATION or BOTH). This traffic description is represented through the loadDescriptor attribute. For instance, it could be instantiated as a unidirectional pipe (two end points, one SOURCE one DESTINATION with one traffic descriptor attached to the SOURCE), bidirectional pipe (two end points with one traffic descriptor), point-to-multipoint (several end points with one traffic descriptor) and so on. More complex topologies are represented by using multiple ServiceEndPointTemplates associated with the same service instance. For instance, a hub and spoke topology with bi-directional traffic (towards and away from the center) is represented through two templates, one a multi-point to point template and the other a point-to-multi-point template.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

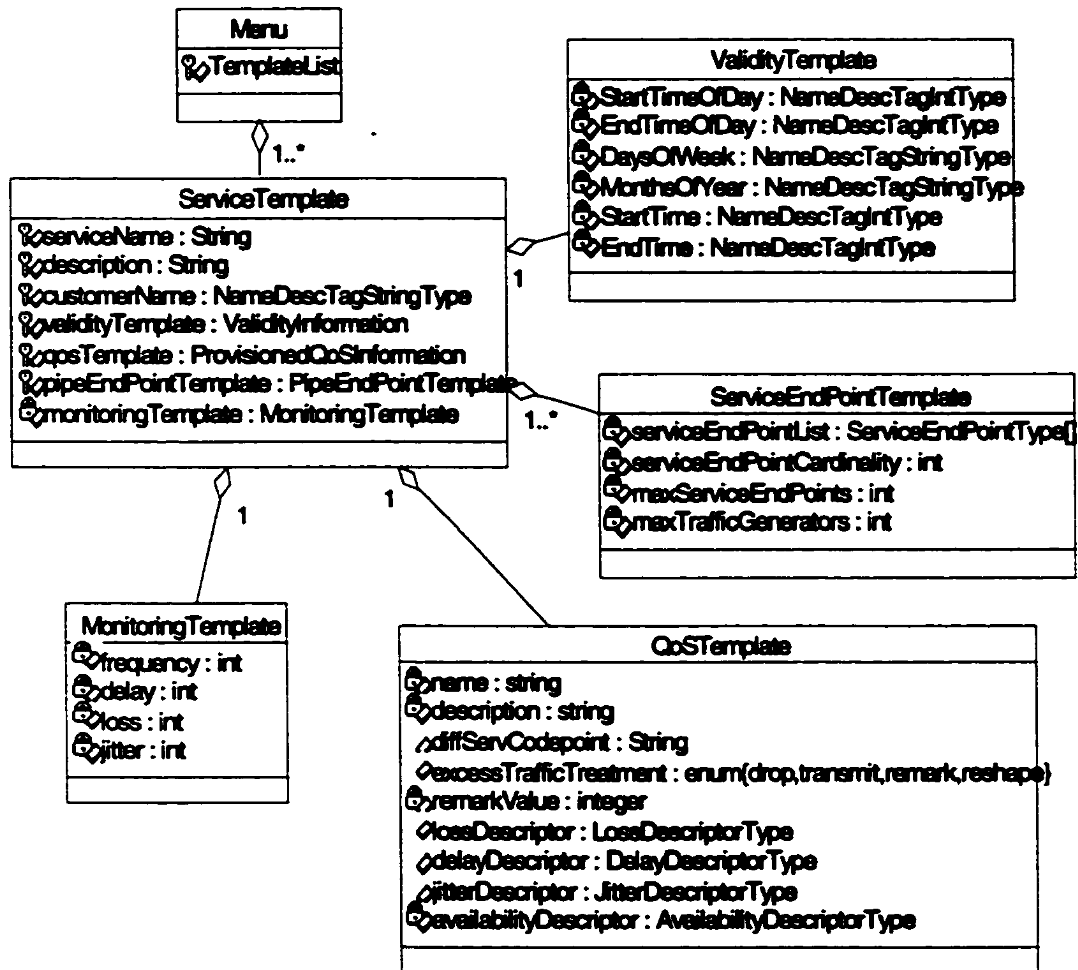


Figure 32: Service Template Information Model

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

QoS Template: The QoS template describes aspects of the service that apply uniformly, irrespective of the topology of the service instance. This includes the DSCP used for the service, the treatment of excess traffic, as well as the delay, loss and jitter attributes of the service.

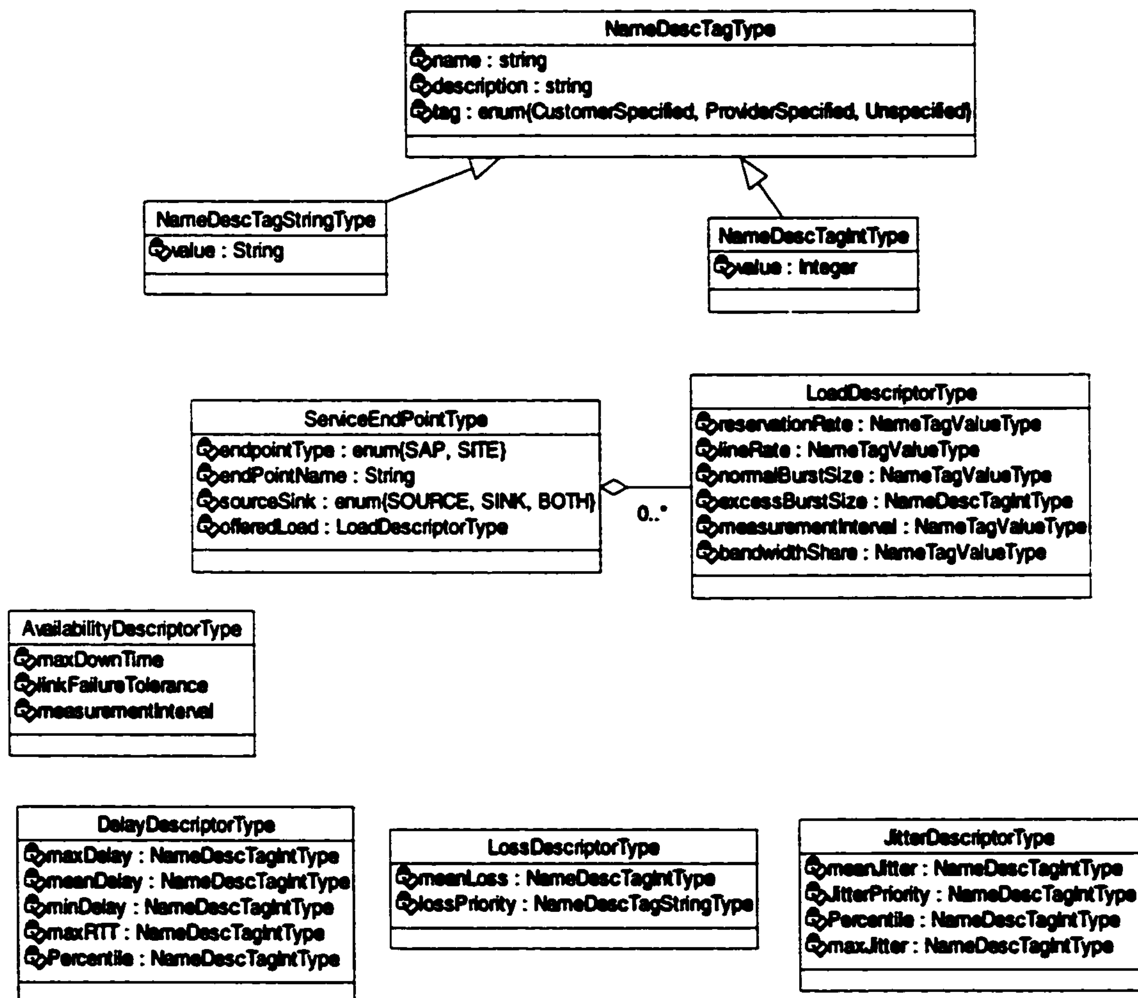


Figure 33 : Service Level Template supporting classes

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 1 (continued on the successive pages 173-179): Service Level Template Class

CLASS NAME	ATTRIBUTE	EXPLANATION
Menu		Grouping class for service templates
	ServiceList	Vector of pointers, each to a service template.
Service Template		Core service template class
	serviceName	Provider specified name of service
	description	Provider specified description of service
	customerName	Provider or customer specified. Has to be unique in the provider's space.
	validityTemplate	Pointer to ValidityTemplate
	qosTemplate	Describes QoS characteristics of the service
	serviceEndPoints Template	It describes the topology of the pipe and offered load from various end points.
	monitoringTempl ate	It describes the monitoring attributes at the service level
Validity Template		One instance per service template. This class describes the time period the service should be active. All the specifications below are "AND"ed, for example, DayOfWeek and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

		MonthOfYear have to be simulatneously satisfied. The list below is incomplete. Use the IETF policy documents to get a more comprehensive list with the right syntax.
	StartTime	Time (including time zone) at which the service begins.
	EndTime	Time at which service ends.
	StartDate	Date at which the service begins.
	EndDate	Date at which the service ends.
Service End Point Template		1 or more instances per Service Template. This class describes the number and nature of end points, as well as the traffic generated by each SOURCE end point. Each Service End Point Template describes a portion of the topology of the service instance being configured.
	Max ServiceEndPoints s	The maximum number of end points (Sites or SAPS) that a service instance can have.
	Max Traffic Generators	The maximum number of end points (Sites or SAPS) that may be sources of traffic. (All other end points are treated as traffic sinks.)
	ServiceEndPoint	The actual number of end points of this

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	Cardinality	service instance.
	ServiceEndPoint List	Vector of ServiceEndPointType, each describing an end point, as well as the traffic it generates if it is a SOURCE.
ServiceEndPointType		Class that describes an end point, whether it is a SOURCE, SINK or BOTH, whether it is a SITE or a SAP, and the traffic generated if it is a source.
		Class that describes the end point.
	EndpointType	Describes the type of the end point. VALUES: SITE or SAP.
	SourceSink	SOURCE indicates that the end point generates but does not consume traffic, SINK indicates that it is a destination but does not generate traffic, BOTH indicates that it generates and consumes traffic.
	Endpoint name	The site name or sap name that is unique in the provider space.
LoadDescriptorType		A struct or complex type used to describe offered or received load. In case that the end point is a SOURCE the load descriptor corresponds to generated traffic. If the end point is a SINK the load descriptor

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

		corresponds to consumed traffic. If the end point is BOTH then the loadDescriptor represents both generated and consumed traffic. Note that the load descriptor may or may not be mandatory, depending on the service requirements.
	Line Rate	The rate of the connection from the SAP
	Normal Burst Size	The normal burst size tolerance that is requested.
	Excess Burst Size	The Excess burst size parameter (> normal burst size) required for the CAR algorithm.
	MEASUREMENT INTERVAL	The duration of time over which the policing is measured
	Bandwidth Share	Fraction of the line rate that is requested for this service instance.
QoS Template		One instance per service template. Describes QoS attributes of the service that applies to the service instance.
	name	Name of the QoS service. The same QoS service can be "bundled" with a different monitoring or charging structure, or be branded differently during a different validity

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

		period.
	description	A user friendly description meant to be presented to the customer.
	diffServCodePoint	DSCP associated with this service. Provider specified.
	excessTrafficTreatment	DROP if excess traffic is to be dropped, TRANSMIT if excess traffic is to be sent into the network, REMARK if excess traffic is to be treated as a different priority class (see remarkValue); RESHAPE, if excess traffic is to be reshaped to the submitted load descriptor.
	remarkValue	DSCP to be marked if excessTrafficTreatment is REMARK.
	lossDescriptor	The lossDescriptor object which describes loss characteristics for this service.
	delayDescriptor	The delayDescriptor object which describes delay characteristics for this service.
	jitterDescriptor	The jitterDescriptor object which describes jitter characteristics of this service.
	availabilityDescriptor	The availabilityDescriptor which describes the characteristics of availability of the service.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Loss Descriptor		One instance per QoS template. Describes the packet loss characteristics of this service. All these are provider specified in the current version.
	meanLoss	It describes the average loss suffered by packets of a conformant flow.
	lossPriority	A smaller loss priority implies a higher drop rate.
Jitter Descriptor	maxJitter	One instance per QoS template. Describes the maximum jitter that will be experienced by a conformant flow
	meanJitter	It describes the average jitter seen by a conformant flow.
	JitterPriority	Describes the relative experience of jitter between different flow aggregates.
	Percentile	The above descriptors apply to this percentile of conformant packets.
Delay Descriptor		One instance per QoS template. Describes the delay imparted to conformant packets by the network
	maxDelay	An upper bound on the delay seen by each conformant packet of the flow.
	meanDelay	The average delay seen by packets.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	minDelay	The minimum (eg. Propagation) delay to be experienced by the flow.
	maxRTT	Maximum round trip time for the flow, taking the forward and reverse paths into account
	Percentile	Associated with meanDelay, maxDelay or minDelay, and describes the portion of packets that will meet the delay standard.
Availability Descriptor		One instance per QoS template. Describes the availability characteristics of the service
	maxDownTime	The maximum time that the service will be down. Usually provider specified.
	measurementInter val	MaxDownTime is measured over this interval.
	linkFailureTolera nce	How many simultaneous link failures should this service instance be protected against?
	nodeFailureToler ance	How many simultaneous node failures should this service instance be protected against?

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

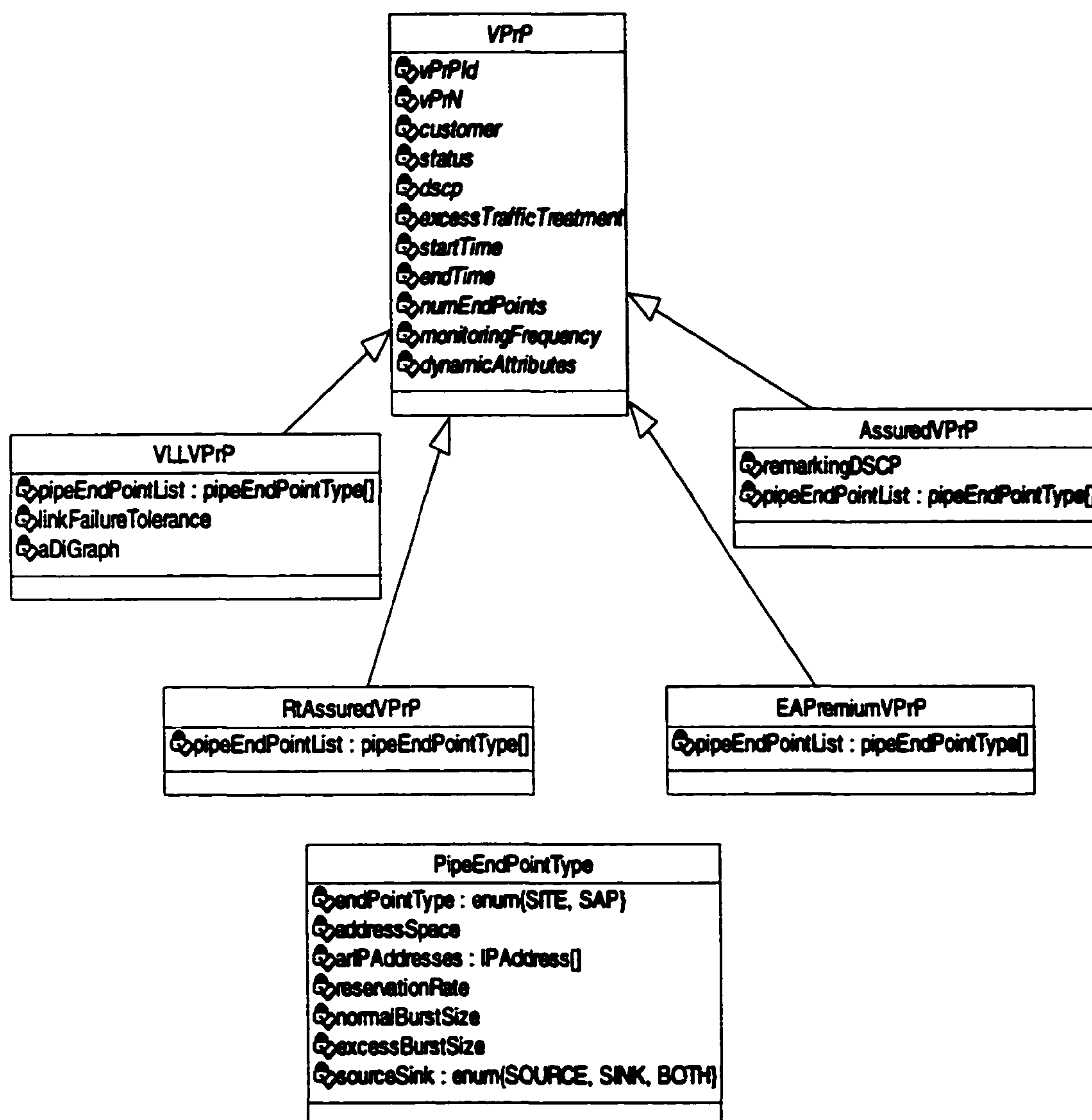


Figure 34: Network Level Template Class

7.1.5.2 Network Level Policy Information Model

While the service level policy may be regarded as a blank form to be filled by the customer (or the service rep.), the network level policy is the VPrP that is instantiated in response to the filled in request. There is an abstract VPrP class from which are

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

derived VPrP classes specific to particular services. The attributes of each service-specific VPrP class are based broadly on the customer-specified attributes of the service.

Table 2 (continued on the successive pages 182 and 183) :
Network Level Template Class

CLASS NAME	ATTRIBUTE	EXPLANATION
VPrP		An abstract class that describes characteristics common to different kinds of network level policies (VPrPs).
	VprPId	A unique identifier for this VPrP. The scope of uniqueness is TBD. Assumed that this is at least unique within the VPrN scope, or Customer scope.
	VPrN	Refers to a customer friendly grouping class. For instance, the customer may wish to group their VPrPs into Eastern and Western Division VPrPs.
	customer	String referring to the owning customer's name. Unique within the providers scope.
	status	PENDING indicates that the provisioning is underway, ACTIVE indicates that the VPrP is in commission, DENIED indicates that the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

		provisioning has failed.
	DSCP	DiffServ code point for this service instance. Copied from the template.
	excessTrafficTreatment	Treatment of excess traffic. Copied from the template.
	remarkingDSCP	If traffic in excess of the load descriptor is to be remarked then this DSCP is used.
	startTime	Time of initiation of service. Provisioning system can override customer specified time
	endTime	Time of termination of service. Provisioning system can override customer specified time
	numEndPoints	The number of end points that this service instance comprises. This will be the number of SAPs in the current service offerings. Later on, it could be the number of SAPs or sites.
	monitoringFrequency	Frequency at which monitoring is done in 100 ms units. Copied from template. This information may be relayed to the customer, or exist for alarm generation purposes.
	delay	Delay experienced by packets of this VPrP as reported from monitoring.
	jitter	Jitter experienced by packets of this VPrP as reported from monitoring.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	loss	Loss experienced by packets of this VPrP as reported from monitoring.

7.1.5.3 Role Level Policy Information Model

This class is modeled from the Policy Core Information Model document of the IETF. Not all attributes from that model are represented below, only those that are needed for foreseeable uses. The policyCondition and policyAction classes from the Core Information Model have been extended for the purposes of representing IP packet classification and DiffServ per-hop behaviors.

Policy rules may be informally represented through the simple paradigm

Policy Rule:

IF (Policy Condition) THEN (Policy Action) (see section 5)

Policy groups are containers used to bunch together policy rules for a variety of purposes -- by interface, by role, by organization, etc.

More formally, they are modeled as below:

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 3 (continued on the successive pages 184-193) :
Role Level Template Class

CLASS OR ATTRIBUTE NAME	TYPE	EXPLANATION
PolicyGroup	CLASS	Container for policy rules. The most common use for this is to group together configuration policies for an interface or group of interfaces with the same role.
name	String	Unique identifier for grouping objects
policyRuleList	vector of pointers	List of pointers to policy rules.
PolicyRule		Models a role level policy injunction.
name	String	String used to uniquely identify policy rule.
keywords	vector of string	Used to store the role to which policy applies, or other keywords for searching.
enabled	ENABLED, DISABLED, ENABLEDFORDEBUG.	Current status of this policy rule.
priority	Int	The priority of this rule. A higher number indicates greater priority.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

usage	CONFIGURATION, USAGE	CONFIGURATION is used for rules that apply irrespective of service instance, eg. configuration of queues in the backbone interfaces. USAGE indicates that this rule is created or deleted on the fly, eg. policing rules at access interfaces.
policyActions	PolicyAction	List of policy actions.
PolicyCondition	CLASS	Each policy condition identifies packets that the policy rule applies to and the validity period.
name	String	Name of this policy condition
policyRule	String	The policy rule that this condition is part of.
PolicyPacketStreamCondition	CLASS	Identifies a sub-stream of packets that the policy applies to, based on the packet header and other attributes of packet processing.
sourceAddressRange	IPAddressRange	Defines an Ipv4 address that the traffic originates from.
sourceMask	IPAdressRange	Identifies a range or a sub-net for the originating source of traffic.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

destinationAddressRange	IPAddressRange	Defines either a range, sub-net or set of IPv4 addresses that the traffic is destined to.
destination Mask	IPAdressRange	Identifies a range or a sub-net for the destination of traffic.
sourcePorts	IntegerSet	Defines a range or set of port numbers that the traffic originates from.
destinationPorts	IntegerSet	Defines a range or set of port numbers that the traffic is destined to.
protocol	Integer	Defines the protocol number that the traffic uses.
dscps	vector of strings	List of DiffServ Code Points.
direction	INCOMING, OUTGOING	Direction of packets with respect to the router or interface that this rule applies to.
PolicyTimePeriodCondition	CLASS	
StartTime	Time	The time at which the policy condition takes effect
EndTime	Time	The time at which the policy condition is not actual anymore.
StartDate	Date	The date at which the policy condition takes effect

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

EndDate	Date	The date at which the validity of the policy condition ends.
PolicyAction	CLASS	Actions to be performed on the sub-stream of packets identified by PolicyCondition.
name	String	Unique identifier for policyAction.
DiffServAction	CLASS	Subclass of PolicyAction that is further sub-classed for various purposes. This is a "grouping" class and has no attributes.
Policing	CLASS	Subclass of DiffServAction that holds policing parameters.
policingType	CAR, LEAKYBUCKET	What type of policing mechanism is being performed. CAR indicates Committed Access Rate; LEAKYBUCKET is the more traditional leaky bucket policer (put here as a placeholder).
conformAction	DiffServAction	Identifies the next action to be performed if the policer is passed successfully by the packet.
exceedAction	DiffServAction	Identifies the next action to be performed if the policer fails the packet.
CARPolicing	CLASS	Describes Cisco Committed Access Rate.
rate	Int	CAR rate parameter.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

normalBurst	Int	CAR parameter for normal burst.
excessBurst	Int	CAR parameter for excess burst tolerance.
LeakyBucketPolicing		The more traditional leaky bucket policing.
meanRate		The average rate of arrival of conformant traffic as measured over the timeInterval.
timeInterval		Time interval over which meanRate.measurement is valid.
burstSize		Burst size tolerance over the timeInterval.
Marking	CLASS	Subclass of DiffServAction that holds marking parameters. For generality mask and mark bitvectors are described, to deal with ToS bits or DSCP bits.
mask	Bitvector	A bit mask has 1s in digits that must be ignored and 0s in digits that are meaningful.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

mark	Bitvector	This holds mandatory 0s in the digits that should not be changed (i.e., digits that are 1 in the mask); the remaining digits are marked on the packet. The simplest way to describe the action of the mask and mark are: NewPktHeader = (PktHeader && mask) || mark where && indicates bitwise AND and || indicates bitwise OR.
nextAction	DiffServAction	Indicates the next action to be performed
Queuing		Subclass of PolicyAction that holds buffer management parameters.
queuingType	TAILDROP, WRED	TAILDROP is a buffer that only has maxThreshold as its parameter; all packets queued beyond this threshold are dropped. WRED is weighted Random Early Discard.
nextAction	DiffServAction	The next action to be performed after placing the packet in the queue.
TailDropQueuing	CLASS	Subclass of Queuing. Describes a simple buffer of size maxThreshold after which all packets are dropped.
maxThreshold	Int	The maximum queue size.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

WREDQueueing	CLASS	Subclass of Queuing. Describes the weighted random early discard mechanism.
minThreshold	Int	No packets are dropped below this threshold
maxThreshold	Int	All packets are dropped above this threshold.
probability	Int	The probability of loss increases linearly from the minThreshold with this maximum value at the maxThreshold.
exponent	Int	Used in the moving average to compute weighed queue lengths. A higher exponent gives greater weight to past queue lengths.
Scheduling		Subclass of PolicyAction that holds queuing parameters.
schedulingType	DWFQ,MDRR	DWFQ is distributed weighted fair queuing, and MDRR is modified deficit round robin.
nextAction	DiffServAction	Identifies the next action to be performed after the packet is queued.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

MDRRScheduling	CLASS	Modified Deficit Round Robin has one low latency high priority (LLHP) queue and a number of other round-robin queues. The latter are served using a deficit round robin mechanism. Arbitration between the LLHP queues and all the round-robin queues can be set to STRICT PRIORITY for the LLHP queue over the others or ALTERNATE PRIORITY where the scheduler alternates between the LLHP queue and one of the others. It may be desirable to split this class in two, one for the LLHP queue and another for the round-robin queues.
mdrrType	STRICTPRIORITY, ALTERNATEPRIORITY	This should be consistent across all the queuing action objects for a single interface.
queueType	LLHP, OTHER	LLHP for the high priority queue, and OTHER for the round-robin queues. One and only one queue per interface may be designated as LLHP.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

quantum	Int	The weight associated with a round robin queue. Is set to be 0 if this is an LLHP queue.
DWFQScheduling	Class	Distributed weighted fair queuing. A Cisco queuing mechanism.
weight	Int	Weight associated with this class. This should be between 1 and 100. The weights on all queues should add up to 100 on each interface.
Reshaping	CLASS	Subclass of PolicyAction that holds reshaping parameters. Some cells are delayed during reshaping in order to fit the given traffic profile. This is different from policing because reshaping usually requires a queuing action before it, and it does not have two outputs – conforming and exceeding packets -- as does remarking.
ReshapingType	TOKENBUCKET, OTHER	TOKENBUCKET is a simple reshaper similar to the token bucket policer.
nextAction	DiffServAction	The action to be performed on cells leaving the reshaper.
TokenBucketReshaping	CLASS	Smooths the flow to a mean rate, measured over the timeInterval.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

meanRate	Int	Desired average rate in bps of departing packets as measured over timeInterval
timeInterval	Int	The time period over which the meanRate is measured.
Concluding Action	CLASS	This class is currently a subclass of DiffServAction, though it is shared with a number of different action types including security actions or ACLs.
type	DROP, TRANSMIT	DROP indicates drop this packet, TRANSMIT indicates transmit this packet.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

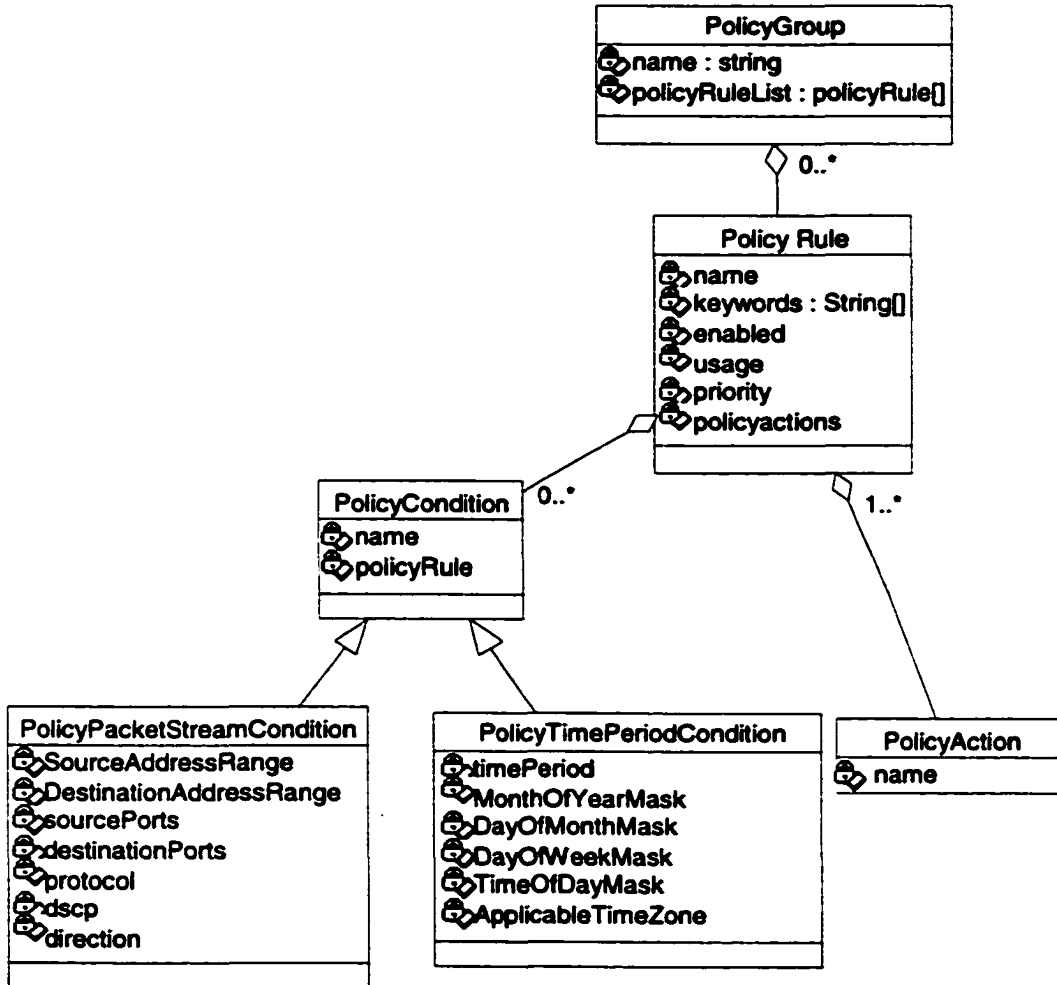


Figure 35 : Role Level Template class

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

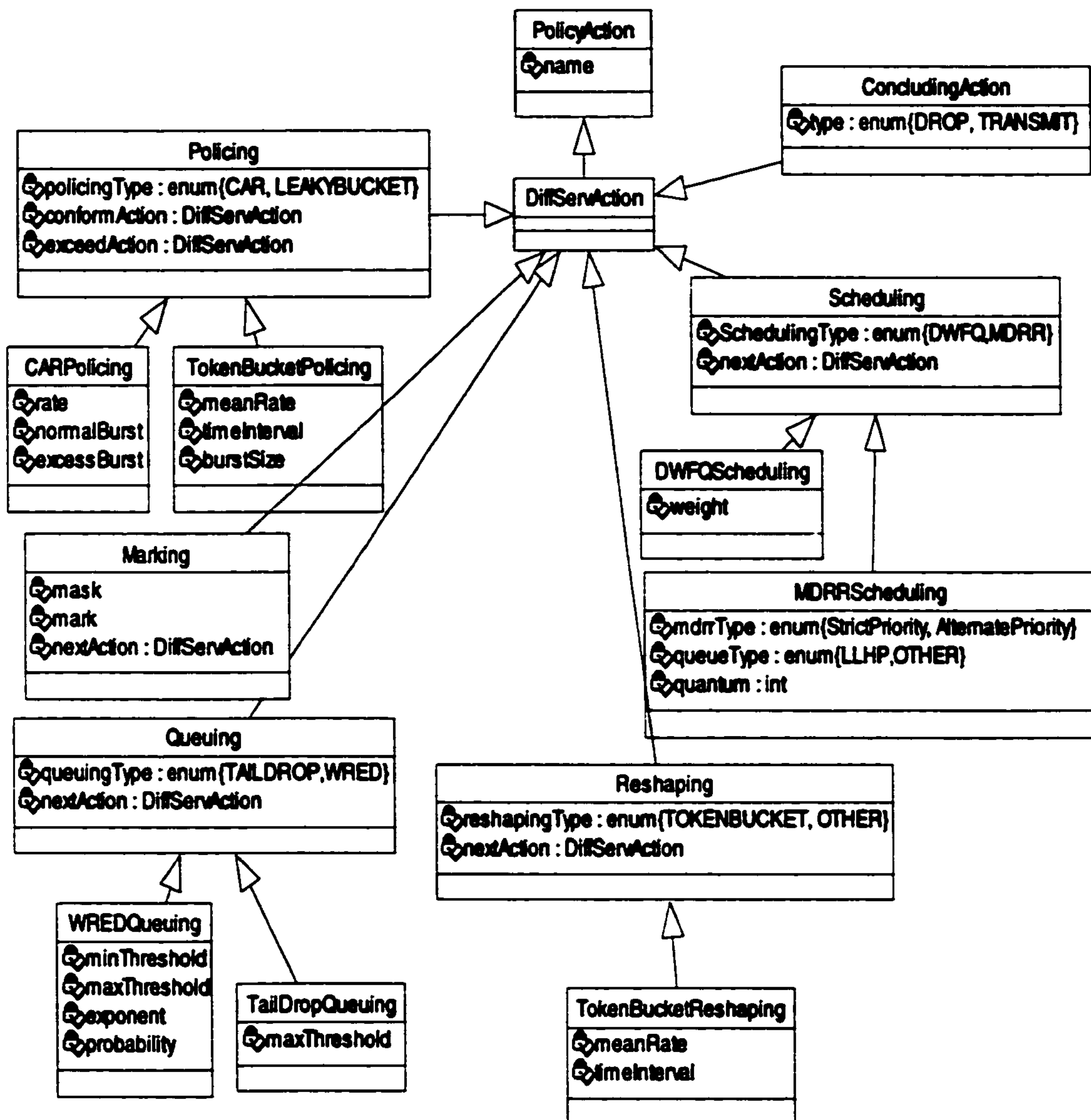


Figure 36 : Policy Action Classes

7.1.6 Use Case for User Requesting QoS Services

This section describes the scenario of a user requesting service and the management process done by the tools through all the policy levels.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Step 1: User or Customer Care Agent fills out a service template (a.k.a., the service level policy object), filling in all customer specifiable information, and submits it to the Policy Configurator module of the Policy server.

Step 2: The Policy Configurator checks the syntax of the service template and retrieves information relevant to the request – SAP, site, address spaces, customer credentials, account information, etc., and sanity checks the request. If the request is admissible, the Policy Configurator returns a request identifier to be passed on to the customer, creates one or more Virtual Provisioned Pipe (VPrP) objects (a.k.a., a network level policy rule) with the request identifier, and marks the request as pending. The request is entered into the policy repository and simultaneously handed off to the policy server.

Step 3: The Policy Translator module of the Policy server resolves each VPrP into a number of point-to-multipoint pipes/hoses, and asks the Topology Server for a path lookup on each of these[2]. The topology server returns an annotated directed graph (ADiG), which identifies the links (and consequently, interfaces/logical ports of devices) that will be affected by the request. The structure is a directed graph as it goes from one origin point to multiple termination points in an acyclic manner. Each segment on this graph (or each branching point) is annotated with the proportion of traffic that is expected to traverse that segment. Second order annotations, such as backup paths on link failure, are also feasible. This step may either involve multiple

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

queries to the topology server, or (more likely) the API may be enhanced to derive all the directed graphs in a single shot, in which case the operation of the Topology Server may be summarized through the input (VPrP, NumFailures) with output (ADiG List). At the end of this step, we have a list of all interfaces/logical ports that will be impacted by the VPrP, an indication of the load they carry in supporting the VPrP, as well as the role each plays (for example access or backbone). In terms of the information model, we assume that the ADiGs will "hang off" the VPrP from this stage on.

Step 4: Further, the Policy Translator returns the policy rules (a.k.a., rule level policies) associated with each *Role*. For instance, for a VLL VPrP the policy rule associated with the INGRESS_ROLE, i.e. with incoming traffic on the access router interface/logical port would involve policing, while that associated with the EGRESS_ROLE, i.e., with outgoing traffic on a backbone router or access router interface would involve a rule assigning traffic to a particular queuing/scheduling resource or per-hop-behavior. In the latter case the amount/nature of resources consumed would also be clearly demarcated. At the end of this step, the policy rules are hanging off the ADiG at the appropriate points.

Step 5: The Policy server hands off the role level policies to the Consistency Checker which queries, per interface/logical port, a Capability Checking function, that will ensure that the interface/logical port can support the set of new role level policies. The input to the capability checking function is the interface/logical port name and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the list of policies. The return value is SUCCESS or FAILURE with explanations. The capability checking function will use the data repository to obtain information about the capabilities of the interface. This step can be accomplished in one shot by handing off the entire VPrP for checking. (The capability check is useful in determining that the QoS service does not traverse non-QoS or ATM links. It has other uses in the context of security services, for instance, to check that the encryption type requested is available on the tunnel-originating interface.)

Step 6: The Conflict Detector queries, per interface/logical port, a Consistency Checking function, with input parameters (Interface/Logical port, New Role Level Policy List). The Consistency Checking function returns CONSISTENT or INCONSITENT with explanations. Global rule validation is considered necessary and the input to the module would be (VPrP) with the same outputs.

Step 7: The Conflict Detector queries the Bandwidth broker Function with input (VPrP, CHECK_RESOURCES) and obtains a SUCCESS or FAILURE with explanation. For all services described in this document, the Bandwidth broker Function is a very simple function. It compares the resource requested on each interface with the available resources, considering the reservation successful if the comparison succeeds for all interfaces. The Bandwidth broker Function writes to the data store, transferring resources from the *Available* to the *Blocked* column in the resource table on each interface, and returns SUCCESS. Otherwise it returns a FAILURE, with explanations. The reason for blocking the resource is, obviously, to

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

minimize race conditions, as the actual commitment of resources has to wait for configuration success.

Step 8: The policy server ascertains the list of Element Managers that manage the interfaces associated with the VPrP and hands off the list of policy rules associated with each interface. The input to this call is a vector of (Interface, Policy Rule Group). The Element Manager returns a Request Handle that is used to refer thereafter to the policy rule group.

Step 9: The Element Manager translates the rules to a form that is suitable for the router in question, and downloads the policies to the router. If the reservations succeed, (Request Handle, SUCCESS) is returned to the Policy Server. Otherwise, (Request Handle, FAILURE, Explanations) is returned.

Step 10: If all element managers are successful in configuring the routers then the bandwidth broker function is instructed with the input (VPrP, COMMIT_RESOURCES) upon which injunction, the latter re-allocates resources from the *Blocked* column to the *Committed* column of the resource table corresponding to each interface. After this call returns with a SUCCESS, the Policy Server marks the VPrP as active (from the pending state). All this information is read from and written to the data store.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Step 11: If one or more element managers return FAILURE, then all the policy rules have to be reversed. In this case the Policy Server enjoins all the element managers to teardown(Request Handle), where the handle is the one returned during the reservation phase.

Step 12: When all element managers return SUCCESS to the teardown( ) call, the bandwidth broker Function is instructed with input (VPrP, ROLLBACK) to free up the blocked resources, and return them to the available resource list.

NOTES

1. *The order of steps 6, 7 & 8 is interchangeable. The order chosen here has the following rationale: Capability checking is fairly lightweight, and is a preliminary sanity check. Bandwidth computations if performed before consistency checking, will result in having to release blocked resources whenever the latter fails – a step that temporarily causes resources to become unavailable.*
2. *There is a dependency between the Topology Server and the Bandwidth broker function in how the ADiGs are annotated.*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

8 Concluding Remarks

This thesis describes the challenges and strategies required for designing a proof of concept, used to rationalize and automate the network management practices in the context of the Policy Based Model, in support of QoS.

This research work gives answers to the following problems:

- Definition of a specific language to validate Service Level Specifications;
- Global conflict detection among policy rules using multidimensional range searching
- Validation of sequences of policy actions that represent QoS mechanisms, including checking for well-formedness, one step loops and many-step loops;
- Translation of policy rules into device specific commands;
- Creation of a distributed, 4-tier architecture, policy management application.

It also proposes open problems in the area of setting QoS parameters according to high-level quality requirements, presenting the risks and challenges of controlling various types of traffic.

9 Future Work

Of further interest within the aim of *Policy Based Network Management* research is to derive a coherent network management architecture encompassing different operational environments, networking technologies and services, based on the vision

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of policy based networking. There are a number of dimensions in which the *Policy system* could be extended in support of this objective:

Policy Based Network Management Model	**Future extensions**
Provisioned QoS (using Diffserv)	RSVP support
Datagram Routing	Path pining using MPLS. Combined with above this allows QoS tunnel overlay through MPLS and RSVP
Provisioning of Backbone Network	Enterprise provisioning integrated with backbone provisioning
Automation of enterprise provider boundary allowing single provider VPrNs	Automating provider to provider VPrP requests (bandwidth broker) allowing multi-provider VPrNs
Elementary security using ACLs	Policy based remote access (using Radius/AAA) and full Virtual Private Networks (using IPSec)

Given the wide scope of policy based networking it is important to focus on a subset of important environments and services, as well as to approach problems one step at a time. In keeping with such a staged approach *Policy Based Network Management* research work aimed to transfer ideas of policy based networking to a focussed

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

reference implementation to prove their viability. This is the reason why initially the prototype was centered on a suite of QoS management applications.

In the near future, the aim of this research activity is to evolve to achieve tomorrow's promise of end-to-end management of new services across multiple domains. In the longer time frame, the goal is to address security and remote access as well, thus evolving to secure QoS VPNs with mobile membership.

Many aspects of the Policy Based Management vision are still to be explored, in particular monitoring to close the management loop, whereby management policy is triggered as a result of a change in the state of the network.

Also, setting QoS parameters for various mechanisms in an automated manner following exact formulae for the purpose of achieving a specific quality of the traffic is a long-term goal, which can be achieved step by step.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

10 Appendices

10.1 Conflict detection Algorithms and their Java implementations

```
public static boolean pscIntersectsRule (String conditionName, String
ruleName)throws Exception, ObjectdoesNotExistException{
ObjectGateway og = PolicyArenaGlobals.getObjectGWRef();
PolicyPktStreamCondition psc = (PolicyPktStreamCondition)
og.getObject(Utilities.str2Fqdn(conditionName));
PSCStruct condStruct = new PSCStruct(psc);
PSCStruct pscIntersection = pscIntersectionInRule(ruleName);
if (condStruct.intersects(pscIntersection))
  return true;
else
  return false;
}
public static Vector CheckConsistency(String newRuleName, Vector oldRuleList)
throws Exception
  {
    Vector returnVec = new Vector(0);
    for(int i=0; i<oldRuleList.size();i++){
      String ruleName = (String)oldRuleList.elementAt(i);
  System.out.println("CheckConsistency between rule "+newRuleName + " and
"+ruleName);
      if (ruleConflict(newRuleName, ruleName))
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
    {
     System.out.println("Conflict exists for rule "+ruleName+" and
"+newRuleName);
     returnVec.addElement(ruleName);
    }
   }
  return returnVec;
 }

package com.policyarena.util;
import java.util.*;
import com.policyarena.exceptions.*;
import java.lang.*;
import com.policyarena.ToolGlobals;

public class RangeStruct extends CollectionStruct
{
long begin;
long end;

public RangeStruct (long begin, long end) throws NumberException{
if ((begin<0) || (end < 0))
  throw new NumberException(" Ill formed range: ");
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
if (begin>end)
   throw new NumberException(" End interval is less than beginning interval: ");
this.begin = begin;
this.end = end;
}

public boolean intersects(CollectionStruct another) throws Exception{
if ((this==null)||(another==null)||((((RangeStruct)another).end <
this.begin)||(((RangeStruct)another).begin> this.end)))
return false;
if (!(another.getClass().toString().equals(this.getClass().toString())))
  throw new Exception("Cannot intersect range with non-range collection");
if ((((RangeStruct)another).end < this.begin)||(((RangeStruct)another).begin>
this.end)){
  return false;
}
else  return true;
}

public CollectionStruct intersection(CollectionStruct another) throws Exception {
long maxbegin, minend;
if (!(another.getClass().toString().equals(this.getClass().toString())))
  {throw new Exception("Cannot intersect range with non-range collection"); }
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
if (this.intersects(another)){
  if  (this.begin <= ((RangeStruct)another).begin){
    maxbegin = ((RangeStruct)another).begin;
  }
  else{
    maxbegin = this.begin;
  }
  if (this.end <= ((RangeStruct)another).end){
    minend = this.end;
  }
  else{
    minend = ((RangeStruct)another).end;
  }
  return new RangeStruct(maxbegin, minend);
 }
 else return null;
}
public void out() {
 System.out.print(" range begin " + Utilities.IPAddressLongToString(begin)+" ");
 System.out.println(" range end " +Utilities.IPAddressLongToString(end)+" ");
}
}
package com.policyarena.util;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
import com.policyarena.objectgateway.*;
import com.policyarena.exceptions.*;

public class TPCStruct {

CollectionStruct[] value;

public TPCStruct (PolicyTimePeriodCondition tpc)throws Exception{
   value = new CollectionStruct[2];
   int tempval[];
   tempval = new int[1];
   long dateStart, dateEnd, timeStart, timeEnd;

   String startDate = tpc.getStartDatePeriod();
   String endDate = tpc.getEndDatePeriod();
   String startTime = tpc.getStartTime();
   String endTime = tpc.getEndTime();

   int beginindex = startDate.indexOf("/");
   int endindex = startDate.lastIndexOf("/");
   String tempStartDate =
startDate.substring(endindex+1)+startDate.substring(0,beginindex)+startDate.substrin
g(beginindex+1,endindex);
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
dateStart = Long.parseLong(tempStartDate);

beginindex = endDate.indexOf("/");
endindex = endDate.lastIndexOf("/");
String tempEndDate =
endDate.substring(endindex+1)+endDate.substring(0,beginindex)+endDate.substring(
beginindex+1,endindex);
dateEnd = Long.parseLong(tempEndDate);

RangeStruct r = new RangeStruct(dateStart,dateEnd);
this.value[0] = r;

int index =  startTime.indexOf(":");
String tempStartTime = startTime.substring(0,
index)+startTime.substring(index+1);
timeStart = Long.parseLong(tempStartTime);
index =  endTime.indexOf(":");
String tempEndTime = endTime.substring(0, index)+endTime.substring(index+1);
timeEnd = Long.parseLong(tempEndTime);

r = new RangeStruct(timeStart,timeEnd);
this.value[1] = r;
}
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
public void output() {
  int index;
          for(index=0;index<2;index++) {
    System.out.print("value[ "+index+ " ] = ");
     if (value[index] != null)
       value[index].out();
     else
       System.out.println("NULL");
          }
  }

public boolean intersects(TPCStruct another)throws Exception{
  boolean result = true;
  int index = 0;
  while ((result == true)&&(index<2)){
  result =  this.value[index].intersects(another.value[index]);
  index++;
  }
  return result;
 }

public TPCStruct intersection(TPCStruct another)throws Exception{
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
TPCStruct result = another;
for(int index=0;index<2;index++)
{
result.value[index] = this.value[index].intersection(another.value[index]);
}
return result;
}
}
public static boolean ruleConflict (String ruleName1, String ruleName2)throws
Exception, ObjectdoesNotExistException
{
  System.out.println("Conflict checking between rule
"+Utilities.getNamefromCimString(ruleName1) + " and
"+Utilities.getNamefromCimString(ruleName2));
  ObjectGateway og = PolicyArenaGlobals.getObjectGWRef();

  PolicyRule iPolicyRule1 = (PolicyRule)(new CIMName(ruleName1)).getRef();
  PolicyRule iPolicyRule2 = (PolicyRule)(new CIMName(ruleName2)).getRef();
  if (iPolicyRule1.getPriority().intValue() != iPolicyRule2.getPriority().intValue())
    return false;

  RuleConditionAssocContainer rulecondcont =
(RuleConditionAssocContainer)og.getContainer("RuleConditionAssociation");
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
String tpcName1 = rulecondcont.getTimePeriodConditionForRule(ruleName1);
PolicyTimePeriodCondition tpc1 = (PolicyTimePeriodCondition)
og.getObject(Utilities.str2Fqdn(tpcName1));
TPCStruct tpcStruct1 = new TPCStruct(tpc1);
String tpcName2 = rulecondcont.getTimePeriodConditionForRule(ruleName2);
PolicyTimePeriodCondition tpc2 = (PolicyTimePeriodCondition)
og.getObject(Utilities.str2Fqdn(tpcName2));
TPCStruct tpcStruct2 = new TPCStruct(tpc2);

boolean tpcOverlap = tpcStruct1.intersects(tpcStruct2);
if (!tpcOverlap){
  System.out.println("TPC's are NOT intersecting between the two rules ");
  return false;
}
else{
  PSCStruct pscIntersection1 = pscIntersectionInRule(ruleName1);
  PSCStruct pscIntersection2 = pscIntersectionInRule(ruleName2);
  boolean pscOverlap = pscIntersection1.intersects(pscIntersection2);
  if (!pscOverlap){
    System.out.println("PSC's are NOT intersecting between the two rules ");
    return false;
  }
  else{
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
      RuleActionAssocContainer ruleactcont =
(RuleActionAssocContainer)og.getContainer("RuleActionAssociation");
      String firstActionName1 = ruleactcont.getFirstActionForRule(ruleName1);
      String firstActionName2 = ruleactcont.getFirstActionForRule(ruleName2);
      if (firstActionName1.equals(firstActionName2)){
        //System.out.println("The first actions are of the same type");
        return false;
      }}}
  return true;
}

public static PSCStruct pscIntersectionInRule (String ruleName)throws Exception,
ObjectdoesNotExistException{
  PSCStruct inters;
  Vector conditionList = new Vector(0);
  ObjectGateway og = PolicyArenaGlobals.getObjectGWRef();
  RuleConditionAssocContainer rulecondcont =
(RuleConditionAssocContainer)og.getContainer("RuleConditionAssociation");
  conditionList = rulecondcont.getAllPscConditionsForRule(ruleName);

  if (conditionList.size()==0){
    return null;
  }
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
 else{
  String conditionName = (String)conditionList.elementAt(0);
  PolicyPktStreamCondition psc = (PolicyPktStreamCondition)
og.getObject(Utilities.str2Fqdn(conditionName));
  inters = new PSCStruct(psc);
  for(int i=1; i<conditionList.size();i++){
    conditionName = (String)conditionList.elementAt(i);
    psc = (PolicyPktStreamCondition)
og.getObject(Utilities.str2Fqdn(conditionName));
    PSCStruct currentPSC = new PSCStruct(psc);
    inters = inters.intersection(currentPSC);
      if (inters == null)
        return null;
   }
  }
return inters;
}
package com.policyarena.util;
import com.policyarena.objectgateway.*;
import com.policyarena.exceptions.*;


public class PSCStruct {
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
CollectionStruct[] value;


public PSCStruct (PolicyPktStreamCondition psc)throws Exception{
    value = new CollectionStruct[7];
    int tempval[];
    tempval = new int[1];
    SetStruct sport, dport, prot;


    Long maxmask=Utilities.IPAddressStringToLong("255.255.255.255");
    long srcaddr = psc.getSrcAddr().getSubnetNumber();
    long srcmask = psc.getSrcAddr().getSubnetMask();
    long destaddr = psc.getDestAddr().getSubnetNumber();
    long destmask = psc.getDestAddr().getSubnetMask();
    int srcport = psc.getSrcPort().intValue();
    int destport = psc.getDestPort().intValue();
    int protocol = psc.getProtocol().intValue();
    String dscp = (String)psc.getDscp();
    int direction = psc.getDirection().intValue();


    if ((srcaddr == 0) && (srcmask == maxmask))
    srcmask = 0;
    if ((destaddr == 0) && (destmask == maxmask))
    destmask = 0;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
long startSrcAddr = srcaddr&srcmask;
long endSrcAddr = srcaddr&maxmask;
RangeStruct r = new RangeStruct(startSrcAddr,endSrcAddr);
this.value[0] = r;
long startDestAddr = destaddr&destmask;
long endDestAddr = destaddr&maxmask;
r = new RangeStruct(startDestAddr,endDestAddr);
this.value[1] = r;
if (srcport == 0)
  {sport = new SetStruct();}
else {
  tempval[0] = srcport;
  sport = new SetStruct(1,tempval);
}
this.value[2] = sport;
if (destport == 0)
  {dport = new SetStruct();}
else {
  tempval[0] = destport;
  dport = new SetStruct(1,tempval);
}
this.value[3] = dport;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
if (protocol == 0)
  {prot = new SetStruct();}
else {
  tempval[0] = protocol;
  prot = new SetStruct(1,tempval);
 }
this.value[4] = prot;
tempval[0] = Integer.parseInt(dscp);
SetStruct dscpSet = new SetStruct(1,tempval);
this.value[5] = dscpSet;
tempval[0] = direction;
SetStruct dir = new SetStruct(1,tempval);
this.value[6] = dir;
}
public void output() {
int index;
        for(index=0;index < 7;index++) {
  System.out.print("value[ "+index+ " ] = ");
    if (value[index] != null)
      value[index].out();
    else
      System.out.println("NULL");
        }
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
 }
public boolean intersects(PSCStruct another)throws Exception{
 boolean result = true;
 int index = 0;
 while ((result == true)&&(index<7)){
 result =  this.value[index].intersects(another.value[index]);
 index++;
 }
 return result;
}

public PSCStruct intersection(PSCStruct another)throws Exception{
 PSCStruct result = another;
 for(int index=0;index<7;index++)
 {
 result.value[index] = this.value[index].intersection(another.value[index]);
 }
return result;
}
}
package com.policyarena.util;
import java.util.*;
import com.policyarena.exceptions.*;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
import java.lang.*;

import com.policyarena.ToolGlobals;


public class SetStruct extends CollectionStruct{

public int cardinality;

private int[] Set;

public boolean universalSet;

public static final int MAXELEMENTS = 1000000;


public SetStruct(int numElements, int [] values){

 if (numElements >= MAXELEMENTS){

  universalSet = true;

  cardinality = this.MAXELEMENTS;

 }

 else{

   universalSet = false;

   cardinality = numElements;

   Set = new int[cardinality];

   int i = 0;

          while (i < cardinality)

   {

           if (noDuplicates(values[i],i)){

           this.Set[i] = values[i];
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
i++;
}
else
System.out.println(" Duplicate elements in one set ");
}
}
}

public SetStruct(){
this.universalSet = true;
this.cardinality = this.MAXELEMENTS;
}

public boolean noDuplicates(int newElement, int currentEnd) {
    int index;
    for (index=0;index < currentEnd;index++) {
        if (Set[index] == newElement) return(false);
        }
    return(true);
    }

public void out() {
    int index;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
if (this.universalSet == true) System.out.println("{Universal Set}");
        else if (cardinality == 0) System.out.println("{ }");
        else
          {
          System.out.print("{" + Set[0]);
          for(index=1;index < cardinality;index++) {
             System.out.print("," + Set[index]);
   }
          }
        }


 public CollectionStruct intersection(CollectionStruct another) throws Exception{
        int index1, index2, index3=0;
   int[] tempval;
   tempval = new int[100];
if (!(another.getClass().toString().equals(this.getClass().toString())))
 throw new Exception("Cannot intersect set with non-set collection");
if (this.universalSet == true){
      SetStruct set3 = (SetStruct) another;
             return(set3);
}
else if (((SetStruct)another).universalSet == true){
    SetStruct set3 = this;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
            return(set3);
    }
    else{
            for(index1=0;index1 < cardinality;index1++) {
              for(index2=0;index2 < ((SetStruct) another).cardinality;index2++) {
                if (Set[index1] == ((SetStruct) another).Set[index2]) {
                      tempval[index3] = Set[index1];
                      index3++;
            }
                      }
                  }
    }
          SetStruct set3 = new SetStruct(index3,tempval);
          return(set3);
     }

    public boolean intersects(CollectionStruct another) throws Exception{
    if (!(another.getClass().toString().equals(this.getClass().toString())))
      throw new Exception("Cannot intersect set with non-set collection");
    if (((((SetStruct)this).cardinality) == 0)||((((SetStruct)another).cardinality) == 0))
      return false;
    CollectionStruct set3 = this.intersection(another);
    if ((((SetStruct)set3).cardinality) == 0)
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
 return false;
else return true;
}
}
```

10.2 Validating Well-formed sequences of QoS mechanisms algorithms and their Java implementations

```
package com.policyarena.util;
import java.util.Vector;
import java.util.Date;
import java.util.Calendar;
import java.util.*;
import java.text.*;
import com.policyarena.objectgateway.*;
import com.policyarena.ToolGlobals;
import com.policyarena.PolicyArenaGlobals;
import com.policyarena.exceptions.*;

public class PolicyUtilities {

public static void VerifyActionAttachment (String dsActionFrom, int xindex, String
dsActionTo)throws MalformedActionSequenceException, Exception
 {
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
ObjectGateway og = PolicyArenaGlobals.getObjectGWRef();
PolicyActionContainer pac =
(PolicyActionContainer)og.getContainer("PolicyActionContainer");

PolicyAction x = (PolicyAction) og.getObject(Utilities.str2Fqdn(dsActionFrom));
PolicyAction y = (PolicyAction) og.getObject(Utilities.str2Fqdn(dsActionTo));

checkForwardOneHop(dsActionFrom, xindex, dsActionTo);

Vector yNext = y.getNextActions();
  if (yNext==null)
  yNext = new Vector(0);

Vector precActions = new Vector(1);
precActions.addElement(dsActionFrom);
size::"+precActions.size());
for (int j=0; j<precActions.size(); j++)
{

  String xname = (String)precActions.elementAt(j);
  x = (PolicyAction) og.getObject(Utilities.str2Fqdn(xname));
  if (yNext != null && yNext.size()>0 )
  {
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
 for (int i=0;i<yNext.size();i++)
 {
   String nextaction=(String)yNext.elementAt(i);
   checkForwardManyHops(xname,nextaction);
 }
}
 Vector temp = pac.getPreviousActions(xname);
   precActions = Utilities.mergeTwoVectors(precActions,temp);
 }
}

public static void VerifyFirstAction (String actName)throws
MalformedActionSequenceException, Exception
 {
  ObjectGateway og = PolicyArenaGlobals.getObjectGWRef();
  PolicyAction act = (PolicyAction) og.getObject(Utilities.str2Fqdn(actName));
  for (int i=0; i<ToolGlobals.NUM_FIRSTACTION_ALLOWED; i++)
  {
  if (ToolGlobals.FirstActionAllowed[i] == act.getActionInfo())
  return;
  }
  throw new MalformedActionSequenceException("FAILED: Action " +
ToolGlobals.actionName[act.getActionInfo()] + " cannot be first action");
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
}

public static void checkForwardOneHop (String xName, int indexx, String yName)
throws MalformedActionSequenceException, Exception
{
    try{
  if (xName.equals(yName))
  throw new Exception(" One step cycle. Action: "+ xName + "cannot follow itself
");

  ObjectGateway og = PolicyArenaGlobals.getObjectGWRef();
PolicyAction y = (PolicyAction) og.getObject(Utilities.str2Fqdn(yName));
int  indexy=y.getActionInfo();
indexy::"+indexy);
  if (ToolGlobals.AdjActionAllowed[indexx][indexy] == ToolGlobals.FAIL)
  throw new MalformedActionSequenceException(" Action of type " +
ToolGlobals.actionName[indexx] + " cannot precede an action of type " +
ToolGlobals.actionName[indexy]);
  }
  catch(Exception exp){
  exp.printStackTrace();
  }
  }
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
public static void checkForwardManyHops (String str1, String str2) throws
MalformedActionSequenceException, Exception
{

ObjectGateway og = PolicyArenaGlobals.getObjectGWRef();
PolicyAction y = (PolicyAction) og.getObject(Utilities.str2Fqdn(str2));

 if (str1.equals(str2))
  throw new MalformedActionSequenceException(" Many steps cycle. Action "+
str2 + " is forming a cycle with action "+str1);

Vector vec = y.getNextActions();
if (!(vec == null))
{
 for (int i=0; i<vec.size();i++)
 {
  checkForwardManyHops (str1, (String)vec.elementAt(i));
 }}}
```

11 Bibliography

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1. "Policy Core Information Model - Version 1 Specification", B. Moore, E. Ellison, J. Strassner, and A. Westerinen. July 2000, draft-ietf-policy-core-info-model-07.txt.

2. "Service Level Specification Semantics, Parameters and negotiation requirements.", D. Goderis, Y. Tjoens, C. Zaccone, C. Jacquenet, G. Memenios, G. Pavlou, R. Egan, D. Griffin, P. Georgatsos, L. Georgiadis. July, 2000 http://search.ietf.org/internet-drafts/draft-tequila-diffserv-sls-00.txt

3. "Policy Framework QoS Information Model", Y. Snir, Y. Ramberg, J. Strassner, R. Cohen, draft-ietf-policy-qos-info-model-01.txt

4. "QoS Policy Schema", Y. Snir, Y. Ramberg, J. Strassner, R.Cohen, Feb 2000, http://www.ietf.org/internet-drafts/draft-ietf-policy-qos-schema-01.txt

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

6. "QoS Policy Information Model", internet draft , Y. Snir, Y Ramberg, J. Strassner, R. Cohen , qos-policy-info-model-00.txt.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

7. "Assured Forwarding PHB Group", F. Baker, J. Heinanen, W. Weiss, J. Wroclawski, RFC 2597, www.ietf.org/rfc/rfc2597.txt.

8. "An Expedited Forwarding PHB", V.Jacobson, K.Nichols, K.Poduri, www.ietf.org/rfc/rfc2598.txt

9. "An Architecture for Differentiated Services", S. Blake, D. Black, Carlson,E.Davies,Z.Wang,W.Weiss, www.ietf.org/rfc/rfc2475.txt

10. "Policy Terminology", A. Westerinen, J. Schnizlein, J. Strassner, Mark Scherling, Bob Quinn, Jay Perry, Shai Herzog, An-Ni Huynh, Mark Carlson http://www.ietf.org/internet-drafts/draft-ietf-policy-terminology-00.txt

11. "Service Level Specification and Usage Framework", Y. T'Joens, D. Goderis, R. Rajan, S. Salsano, C. Jacquenet, G. Memenios, G. Pavlou, R. Egan, D. Griffin, P. Vanheuven, P. Georgatsos, L. Georgiadis, draft-manyfolks-sls-framework-00.txt

12. "The COPS (Common Open Policy Service) Protocol" D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, A. Sastry. January 2000. RFC2748, http://www.ietf.org/rfc/rfc2748.txt

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

13. "COPS Usage for RSVP", S. Herzog, J. Boyle, R . Cohen, D. Durham, R. Rajan, A. Sastry, RFC2749, http:/ /www.ietf.org/rfc/rfc2749.txt

14. "Framework Policy Information Base", M. Fine, K. McCloghrie, J. Seligson, K. Chan, S. Hahn, R. Sahita, A. Smith, F. Reichmeyer, draft-ietf-rap-frameworkpib-02.txt

15. "An Informal Management Model for Diffserv Routers", Y Bernet, A. Smith, S. Blake, D. Grossman, draft-ietf-diffserv-model-04.txt

16. "Computational Geometry - An Introduction", Franco P. Preparata, Michael Ian Shamos, Springer-Verlag 175 Fifth Ave, NY, 1985, ISBN 0-387-96131-3, ISBN 3-540-96131-3

17. "Cisco IOS 12.0 Quality of Service" - documentation from the Cisco IOS Reference Library, ISBN 1-57870-161-9

18. "OSPF - Anatomy of an Internet Routing Protocol", John T. Moy, Addison Wesley 1998, ISBN 0-201-63472-4

19. "Computer Networking : A top-down Approach Featuring the Internet" , J.F. Kurose, K.W. Ross, http://occ.awonline.com/bookbind/pubbooks/kurose-ross1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

20. "BGP4 - Interdomain Routing in the Internet", J.W. Stewart III, Addison Wesley 1999, ISBN 0-201-37951-1

21. "Topics in Intersection Graph Theory", Terry A. McKee, F.R. McMorris, SIAM Monographs on Discrete Mathematics and Applications, April 1999, ISBN-89871-430-3

22. "Graph Theory", Frank Harary, Perseus Publishing, 1994, ISBN 0-201-41033-8

23. "Graph Theory and its applications", Jonathan Gross, J. Yellen, CRC Press, LLC/December 1998, ISBN 0-8493-3982-0

24. "Connection digraphs and second order line digraphs" L.W. Beineke, C.M. Zamfirescu, Discrete Mathematics, 39 (1982) 237-254

25. "Total digraphs", G. Chartrand, M. J. Stewart, Canadian Math. Bull. 9 (1966) 171-176

26. "The representation of a graph by set intersection", P. Erdos, A. Goodman, L. Posa, Can. J. Math 18 (1996)106-112

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

27. "Some properties of line digraphs", F. Harary, R.Z. Norman, Rend. Circ. Mat. Palermo 9 (1960) 161-168

28. "Sur une certaine correspondence entre graphs", C. Heuchene, Bull. Soc. Roy. Sci. Liege 33 (1964)743-753

29. "Interval digraphs – an analogue of interval graphs", M. Sen, S. Das, A. B. Roy, D. B. West, J. Grpah Theory 13 (1989) 189-202

30. "Local and global characterization of middle digraphs, Theory and Applications of Graphs", C. Zamfirescu, Ed. G. Chartrand, 1981, John Wiley and Sons, 595-607

31. "QBone Architecture (v1.0) ", Internet2 QoS Working Group Draft, http://www.internet2.edu/qos/wg/papers/qbArch/1.0/draft-i2-qbone-arch-1.0.html
August, 1999

32. "A Generic Traffic Conditioner", L. Lin, J. Lo, draft-lin-diffserv-gtc-01.txt

33. "Packet Classsification on Multiple Fields", Pankaj Gupta, Nick McKeown, Computer Systems Laboratory, Stanford University Stanford, CA 94305-

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

9030, pankaj@stanford.edu, nickm@stanford.edu, Proc. ACM SIGCOMM 1999, pp147-160

34. "Packet Classification using Hierachical, Intelligent Cuttings", Pankaj Gupta, Nick McKeown, Computer Systems Laboratory, Stanford University Stanford, CA 94305-9030, pankaj@stanford.edu, nickm@stanford.edu, Hot Interconnects VII, Aug. 1999

35. "Internet Packet Filter Management and Rectangle Geometry", D. Epstein (Dept. Inf.& Comp Science Unic. Of California, Irvine, CA 92697-3425 eppstein@ics.uci.edu), S. Muthukrishnan, AT&T Labs Research (Shannon Laboratory, Florham Park, NJ 07932 muthu@research.att.com) – Proprietary Information

36. "Detecting and Resolving Packet Filter Conflicts", Adisehu Hari, Subash Suri, Guru Parulkar, Bell Laboratories, 101 Crawfords Corner Road Box 1045, Holmdel, NJ 07733, hari@bell-labs.com – AT&T Proprietary information

37. "A modular approach to packet classification: Algorithms and Results", Thomas Woo, Bell Laboratories, Lucent Technologies. Woo@research.bell-labs.com – AT&T Proprietary information

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

38. "Tradeoffs for Packet Classification", Anja Feldman, S. Muthukrishnan, AT&T Labs Research Shannon Laboratories, Florham Park, fanja@research.att.com, muthug@research.att.com, INFOCOM, Mar. 2000

39. "Detecting and Resolving Packet Filter Conflicts" A. Hari, S. Suri, Guru Parulkar (Bell Labs & Washington University), 101 Crawfords Corner RoadBox 1045, Holmdel, NJ07733 hari@bell-labs.com (AT&T Proprietary information)

40. "Fast packet classification using tuple space search", V. Srinivasan, G. Varghese, S. Suri, ACM SIGCOMM, Sept 1999, pp135-146

41. "Routing on longest matching prefixes", W. Doeringer, G. Karjoth, M. Nassehi, IEEE/ACM Transactions on Networking, vol. 4, no. 1, pp 86-97, Feb. 1996

42. "Quality of Service in the Internet: Fact, Fiction, or Compromise", P. Ferguson and G. Huston. INET'98. August, 1998.

43. "Internet Protocol", J. Postel. RFC 791. IETF, September 1981.

44. "Transmission Control Protocol", J. Postel. RFC 793. IETF, September 1981

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

45. "Random Early Detection Gateways for Congestion Avoidance", S. Floyd and V. Jacobson.. IEEE/ACM Transactions on Networking, Vol. 1, No. 4, August 1993.

46. "User Datagram Protocol", J. Postel. RFC 768. IETF, August 1980.

47. "Data and Computer Communications. 5th Edition", W. Stallings. Prentice Hall, Upper Saddle River, NJ, USA, 1997.

48. "Gigabit Networking", C. Partridge. Addison Wesley, Reading, MA, USA, 1994.

49. IETF DiffServ Working Group. Available at: http://www.ietf.org/html.charters/ diffserv-charter.html>, March 1999.

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

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

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

52. "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks", A. Parekh. Technical Report LIDS-TR-2089, Laboratory for Information and Decision Systems, MIT, MA, USA, 1992.

53. "Cisco IOS(TM) Software Quality of Service Solutions", Cisco Systems. Available at: <http://www.cisco.com/warp/public/732/net_enabled/qosio_wp.htm>, January 1999.

54. "Realizing Throughput Guarantees in a Differentiated Services Network", I. Yeom and N. Reddy. In Proceedings of the ICMCS, June 99. 1999.

55. "Modeling TCP Behavior in a Differentiated-Services Network", Ikjun Yeom and A. L. Narasimha Reddy. Technical report, TAMU ECE, 1999.

56. "Adaptive Packet Marking for Providing Differentiated Services in the Internet", W. Feng, D. Kandlur, D. Saha, and K. Shin. In Proceedings of the International Confrerence on Network Protocols. 1998.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.