ABSTRACT

ARUNACHALAM, CHIDAMBARAM. Implementation and Validation of Network Policy Services. (Under the direction of Dr. Mladen A.Vouk and Dr. Ioannis Viniotis)

The widespread use of Internet Protocol to deliver voice, video and data to end users has made the deployment of some forms of Quality of Service (QoS) mechanisms an essential requirement in today's campus, enterprise, and service-provider networks. Network administrators need to protect and guarantee QoS elements such as bandwidth, delay and jitter to mission critical applications. At the networking level, QoS can be provided using Differentiated services, Integrated services or some other mechanisms.

Policy based networking provides the network administrator with the ability to define and deploy network polices that control QoS mechanisms. Some of the challenges in this area are the difficulty in specifying, managing and deploying complex, interrelated and inherited policies, detection of policy conflicts, and the deployment of dynamic policies.

The objective of this thesis was to develop and experimentally study a traffic-aware, standards-based policy server and policy management toolset that allows network administrators to define and deploy policies at different levels of granularity, and with the highest level of flexibility. The server uses the IETF's Policy Core Information Model (PCIM) and it stores network policies in a Lightweight Directory Access Protocol (LDAP) directory, it obtains real-time network traffic information from Resource Usage Monitor (a distributed traffic monitoring tool developed at NCSU), and then it uses a Configuration Server to interact with the networking devices to disseminate and implement any changes required in (measurement–based) polices. The servers and the toolset were deployed in an emulated campus environment, and were then evaluated by enforcing network policies that control bandwidth usage and provision QoS to (simulated) voice and video traffic. As part of the evaluation measurement-driven dynamic policy changes were deployed pro-actively to assess the ability of the system to protect QoS of mission critical applications. Experiments show that adaptive policy based management of end-to-end QoS is feasible, and may be very useful in the context of complex service level agreements.

IMPLEMENTATION AND VALIDATION OF NETWORK POLICY SERVICES

by
ARUNACHALAM CHIDAMBARAM

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

COMPUTER NETWORKING

Raleigh

2002

APPROVED BY:

Dr. Douglas Reeves	Dr. Ioannis Viniotis, Co-Chair

Dr. Mladen A.Vouk
Chair of Advisory Committee

DEDICATION

To my late grand father M.Chidambaram

To my late uncle R.Ramanathan

Who have made my higher education dreams come true.

BIOGRAGPHY

Arunachalam Chidambaram was born in Chennai, TamilNadu, India. He did his undergraduate program in Sri Venkateswara College of Engineering and received B.Tech in Information Technology from University of Madras in 2000. He joined North Carolina State University in the same year to pursue M.S in Computer Networking. He worked as Research Assistant for Dr. Mladen A. Vouk in the field of Policy Based Networking, for about a year. As part of his M.S co-operative education program, he has worked in Nortel Networks (Voice over IP simulation) and Cisco Systems (Voice over IP Implementation and Security) in Research Triangle Park, Raleigh.

ACKNOWLEDGEMENTS

I would like to thank my Advisors Dr. Mladen A. Vouk and Dr. Ioannis Viniotis for their invaluable guidance, motivation, and encouragement and for providing a great opportunity to work with them. I would like to thank Dr. Douglas Reeves for accepting to be a committee member and providing comments on this thesis.

I would like to thank my parents for their incredible support and encouragement who have made this work possible. I would like to thank my cousin Annamalai for his help and guidance. I would like to thank my sister Vallikannu and my friends Nachiappan and Karthik Chandrasekhar for their help and encouragement.

My thanks are due to Marhn Fullmer, Ruben Lobo and Nikola Vouk who have helped me a lot in setting up my network test bed. I would like to thank Brian Goff for his help and quick responses to all my questions. I would like to thank John Toebes, my manager at Cisco Systems, for allowing me to work four days a week, in order to have more time for my thesis work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

This chapter presents the motivational background and goals of this thesis. It also discusses the problems in achieving the goals, issues related to construction of an effective solution, and challenges in implementing the solution.

1.1 Voice, Video and Data over IP Network

Originally, Internet Protocol (IP) was used to carry only data. However, development of real-time solutions such as voice- and video-specific packetization techniques, echo cancellation techniques, efficient compression algorithms (G.711, G.723.1, G.726, G.729) and H.323 [1] suite standards made the deployment of low to medium bandwidth voice and video over IP a reality. Similarly, MPEG (Moving Pictures Expert Group) standards developed by the International Telecommunications Union (ITU) and the International Standards Organization (ISO) for compression of audio, video and data encouraged deployment of medium to high quality video streaming ranging from 1.5 Mbps (MPEG1 [2]) to 15Mbps (MPEG2 [3]). Interactive, almost appliance-like, multimedia and video tele-conferencing, using H.323 and/or MPEG are now routine. More recently MPEG4 [4] standard has been gaining in popularity in the context of transmission of video over wireless hand-held devices (e.g., cell-phones). The integration of voice, video and data services over IP networks has an interesting and beneficial network management side-effect. Network administrators now need to manage only a single IP network, instead of managing an IP data network plus a traditional voice telephone network, plus a possibly separate video network. This integration also enables new Internet-based applications that provide or depend on unified voice, video and data services. A good example is video conferencing. However, this same convergence of voice, video and data has made Quality of Service (QoS) an essential requirement in communication networks. Voice, video and data streams may tolerate different loss rates, may have to be synchronized to a different degree, and may have different bandwidth and delay requirements.

1.2 The Need for QoS Mechanisms

QoS of a network refers to the properties of the network that directly contribute to the degree of satisfaction that users perceive relative to the network's performance [5]. Network bandwidth, delay, jitter and packet loss are some of the factors that affect and reflect QoS received by an application, and hence may impact end-user quality of service. As mentioned earlier, different network-based applications require different levels of QoS in order to function properly and provide a good end user experience. For example, a Voice over IP (VoIP) stream requires less bandwidth than a video stream, but both need to be synchronized to within 40 ms or less if they refer to a simultaneous event [6]. Similarly, human interactive applications like IP Telephony and Telnet require low network delay, usually not exceeding 100 to 200 ms, in order to retain its natural usability level [7].

Delay. In G.114 recommendation the International Telecommunication Union (ITU) has specified a range of 0-150 milliseconds as the acceptable network delay range for voice applications. There are two components that contribute to network delays, namely fixed delay and variable delay. Packetization, compression and serialization delays add to fixed delay. Variable delays are caused by queuing/buffering in routing and switching elements of the network. It is important to note that in practice it is easier to provide some delay guarantees by manipulating the variable delay component than it is by manipulating the "fixed" delay component.

Jitter. Another parameter of interest is jitter. Jitter is defined as a variation in the delay of packets at the receiving end. In the case where, at the sending side, the packets are sent in a continuous stream and with even spacing between the packets, i.e., initially with a uniform inter-packet delay, network path may cause the delay between each packet to vary.

Losses. There may also be losses of packets along the way. This is not a good thing if the application is sensitive to loss, and both voice and high definition video are in this category. For example, Table 1.1. illustrates the impact of jitter and loss on the end-quality [8],

Table 1.1 QoS Requirements for a VoIP Application

Quality	Packet Loss (%)	Jitter (milliseconds)
Perfect	0	0
Good	3	75
Medium	10	125
Poor	25	225

Required bandwidth for a voice application depends, among other things, on the compression algorithm used. For example, Real Time Protocol (RTP [9]) header compression and Voice Activity Detection [10] techniques can be used to reduce the bandwidth consumed by a typical voice call into the range from 11.2 kbps to 50 kbps [11]. Video applications may require higher bandwidth connections, from 20 to 64 Kbps for low-end quality and resolution, to as much as 386 Kbps or more for mid-range quality, and up to 15 Mbps for broadcast quality. Optimal network delay for real-time video application is in the range of 125-150 milliseconds.

1.3 Some QoS Mechanisms

Differentiated Services [12,13] (DiffServ) and Integrated Services [14,15] are the two approaches/models used to provide QoS to end user applications. In the DiffServ model, IP packets are classified and marked to receive a particular per-hop forwarding behavior (PHB) on nodes along their path. In the case of DiffServ, classification, marking, policing and shaping operations are performed at network boundaries or nodes (such as ingress or egress routers). The nodes at the core of the network are also provisioned to provide required QoS (per-hop behaviors) for different classes of traffic. For example, the type of congestion control mechanism to be used for an aggregated DiffServ traffic class may be provisioned at the core network. IP Packets are marked using the Differentiated Services (DS) field in the IPv4 or IPv6 headers. This provides packet "flows" that can be clearly distinguished both at the edges and in the core of the network. The ones that have higher priority (and appropriate resource privileges) can then be given preferential treatment from one end to the other end of a DiffServ-enabled network. Of course, the preferences and PHBs along an end-to-end path also need to be coordinated. This is done through definition

and maintenance of per-domain behaviors (PDB) using, for example, policy-based methods.

Integrated Services model can be used to provide guaranteed QoS and controlled-load services to application flows. These services are requested by client applications using, for example, Resource Reservation Protocol (RSVP [16]), or some other such reservation setup protocols. Guaranteed QoS service provides an assured bandwidth and a maximum bound on the end-to-end delay for all conforming packets of the client application flow. The controlled-load service provides the client application flow with a QoS closely approximating the QoS that same flow would receive from an unloaded network element. This approach uses capacity (admission) control to assure that appropriate service is received even in high-load conditions. Network devices use admission control algorithms to decide whether to accept or deny a QoS reservation request from a client application. Such algorithms are used to regulate the load in a network and provide bounded delay services for real-time applications. The admission decision may be based on matching individual RSVP requests against a specified traffic profile, or on matching against the aggregated sum of all QoS resources currently allocated. Traffic profile of a flow, or a group of flows (aggregated traffic class), can be specified using policy-based methods.

1.4 Some issues in providing end-to-end QoS

One important set of issues is a) specifying what QoS mechanisms to use, b) when to use a particular QoS mechanism, and c) which users or applications should receive what levels of QoS. In an environment, which is NOT best-effort (BE) based, different users have different privileges and permissions to access different network services at different times. Consequently, the level of QoS provided to a flow from the same application (e.g., video) may vary depending on the type of end-user and the time at which the application is used.

For example,

- In an enterprise, video and web traffic of an executive may be provided guaranteed bandwidth, while the traffic of an employee not engaged in a high-priority task may be serviced using best-effort.
- In a university, the maximum bandwidth usage of student peer-to-peer applications may be

restricted to 5 Mbps per dorm during peak hours and to 10 Mbps per dorm during non-peak hours.

- Mission critical tasks, like desktop VoIP phones, or H.323 video/voice carrying distance education class in real-time during a specified class slot, may be provided guaranteed bandwidth and delay, irrespective of the end-user.

Another problem associated with providing QoS is the scalability of methods for configuration of network devices on an end-to-end path. Since all network devices that affect a path must be correctly configured to provide the appropriate level of QoS to an end-user application, configuration may become a difficult task when the number of network devices is large. For example, there may be hundreds of network devices in a campus network.

In order to provide QoS effectively and efficiently, properties of a network must be pro-actively protected from violation. The violation may or may not be intentional. For example, a program that, in effect, performs a Denial of Service (DoS) violation, may be downloaded to a user's computer with or without consent of the end-user (e.g., peer-to-peer applications). It can then, consume bandwidth with (deliberate serving of peer-to-peer content) or without (e.g., a misconfiguration that leads to unintentional serving of peer-to-peer content, or activation of a DoS virus software) the knowledge of the user. The problem is not an easy one to solve unless network administrator has direct control to the very edge of the network (effectively to the desktop). For instance, although the QoS mechanisms at the edge devices may drop the packets of the violating hosts (e.g. by using a policer at the network edge router), the packets of the violating hosts still consume bandwidth in their path to the network ingress. Such violating hosts increase the contention for bandwidth in their local area network and affect QoS provided to other hosts.

1.5 Policy Based Networking

Policy Based Networking is a way of providing network administrators with a systematic ability to define and deploy reactive and pro-active rules that control a networking environment. It is an effective solution to the problems discussed in section 1.4. A policy consists of one or more rules that describe the action(s) that must occur when specific condition(s) exist. Policies are classified into the following categories based on

their purpose and intent.

Configuration Policies

Configuration Policies define the setup of a managed entity. For example, the per-hop default forwarding behaviors of a router can be specified as a configuration policy.

Installation Policies

These policies specify what can and cannot be installed on a system, and the configuration of the mechanisms that perform the installation. They typically represent specific administrative permissions.

Error and Event Policies

These policies specify the type of actions that a subject must perform on occurrence of an event. For example, a policy can be specified as "If the bandwidth consumption of a host, alternatively the number of flows emanating from a host, is above the maximum threshold level, shutdown the port to which the host is connected". This happens to be a very powerful policy that can be used to prevent Denial of Service attacks.

Security Policies

Security policies deal with verifying that the client is actually who the client purports to be by selecting and applying appropriate authentication mechanisms, permitting or denying resources, etc. For example, an access list policy is a security policy that defines the list of users that are permitted or denied to enter a network.

Service Policies

These policies are used to characterize network and other services. Service policies describe the services available in the network. For example, a service policy may specify that "All interfaces in the core network must provide weighted random early detect (RED) congestion control mechanism".

Usage Policies

Usage policies describe the particular binding of a client (application flow) to services available in the network. For example, usage policies may specify "Apply Tail-Drop congestion control mechanism to all VoIP traffic"; "Provide guaranteed bandwidth of 10Mbps to Vice-Chancellor X".

1.6 Challenges in Policy Based Networking

1.6.1 Mapping of business policies into network policies

Business policies are used to express the economic characteristics (and thus requirements) of different applications. They prioritize which applications get "better" treatment when the network is congested. They are not directly related to network device operation and configuration details, but they do dictate the "What" part of the QoS, i.e., how much and when resources should be given to a certain application flow. For example, a simple high-level business policy may specify that all employees in the Engineering department will have high priority access to Enterprise Application Server.

The business policy is translated into device-independent "How" or policy. Device independent policies specify a set of QoS mechanisms such as DiffServ marking, weighted Round Robin scheduling, Random Early Detect dropping, etc., that are to be enabled on the device elements in order to enforce the business policy. Typically, the network administrator performs this translation. This translation may be a complex process and is one of the main challenges in the Policy Based Networking when one deals with complex business policies

Service Level Agreement (SLA) is a service contract between a customer and a service provider that specifies the forwarding service a customer should receive. SLA can be specified as high-level business policies. Research has been done in this area to define functions that map the SLA requirements into combinations of QoS controls at the network devices. For example, Haddad [17] discusses an analysis of mapping Service level agreements into DiffServ [12,13] QoS controls. Device independent policies are translated into device dependent policies that specify configuration details according to the type of network

device (router, switch etc.) and the manufacturer of the network device.

1.6.2 Specification of Policies

This challenge deals with specification of policies in a simple, intuitive and efficient manner. Policy languages and information models such as Ponder [18], Path Based Policy Language [19], Policy Description Language [20], and IETF Policy Core Information model [21, 22] have been developed in this regard. A policy language must enhance policy readability, expressiveness and reusability. A basic requirement for a policy language is to have the ability to define complex policies using simple policies as building blocks. The language should also allow policy writers to specify policy at appropriate level of granularity. Section 2.1 discusses the policy languages proposed by various research groups.

1.6.3 Management of Policies

Policy Management deals with the monitoring of the policy execution, creation of new policies, and modification and deletion of existing policies. The challenge associated with policy management is to provide the network administrator with the highest level of flexibility. This can be achieved by providing a network administrator with the ability to enable or disable a top-level policy or component policy at any point in time. Also, it means allowing the user to specify validity time periods of a policy to avoid the need to manually enable and disable policies. For example, a policy may specify that high priority should be given to H.323 traffic from/to classroom in Daniels 331 at NC State every Tuesday and Thursday 14.00 to 17:00 (except during official University holidays) starting after August 16th and ending December 17th). Section 3.4.3 discusses how such time conditions of a policy are specified in the IETF Policy Core Information Model.

Policy Conflict detection

Another major issue is that policies that are to be deployed at the network devices must be verified before being deployed. This applies to all policy-levels (Business, device-independent, and device-dependent) to ensure that there are no policy conflicts. Policy conflicts are of two types namely intra-policy conflicts and inter-policy conflicts [19]. Intra-policy conflicts occur when the conditions of at least two policies are

simultaneously satisfied, but the execution of the actions of these policies cannot be executed at the same time. For example there may be two policies with same condition "source_port == video_port", and one of them specifies a tail drop congestion mechanism and the other specifies a Random Early Detect congestion mechanism to be applied for matching packets. These two policies when deployed will result in an intra-policy conflict, because both policy rules will be matched at the same time, but their actions are conflicting. Inter-policy conflicts occur when two or more policies result in conflicting configuration commands/requests on one or more networking device (PHBs). For example, two different policies when deployed may result in the number of queues in one network device not matching the number of queues allocated in the second device supporting the same traffic flow. This may result in packet losses along the way.

Policy conflicts can be resolved using

- Simple priority mechanisms in which only the highest priority policy of the conflicting policies or rules is executed. The IETF Policy Core Information Model [21,22] uses this approach to resolve conflicts.
- Knowledge (metadata) to determine which rule must be applied. This method is more sophisticated than the simple priority mechanism. The Ponder Policy Specification language uses this method [18]. The metadata that specifies what policies can co-exist is stored as a meta-policy.

A number of research groups are currently working in IPSec Policy correctness, conflict detection and resolution [23]. The Policy management tool (Figure 1.1) developed as part of this thesis provides intra-policy conflict detection and simple priority based conflict resolution at device-independent level by using a Role based policy conflict detection algorithm (Section 4.2.4). It currently does not handle Inter policy conflicts and IPSec policy conflicts.

1.6.4 Deployment of Policies

Policy deployment can either be strongly distributed or weakly distributed. In a strongly distributed policy deployment model, the policy server, software that evaluates and initiates distribution of policies, sends

enabled policies to the policy enforcement entities that are closer to the network devices. The enforcement entities are distributed throughout the network and are responsible for ensuring that the policies are enforced. In a weakly distributed model, a set of monitors provides triggers, or network events, to the central policy server. The central policy server then performs both the evaluation and enforcement of the policies.

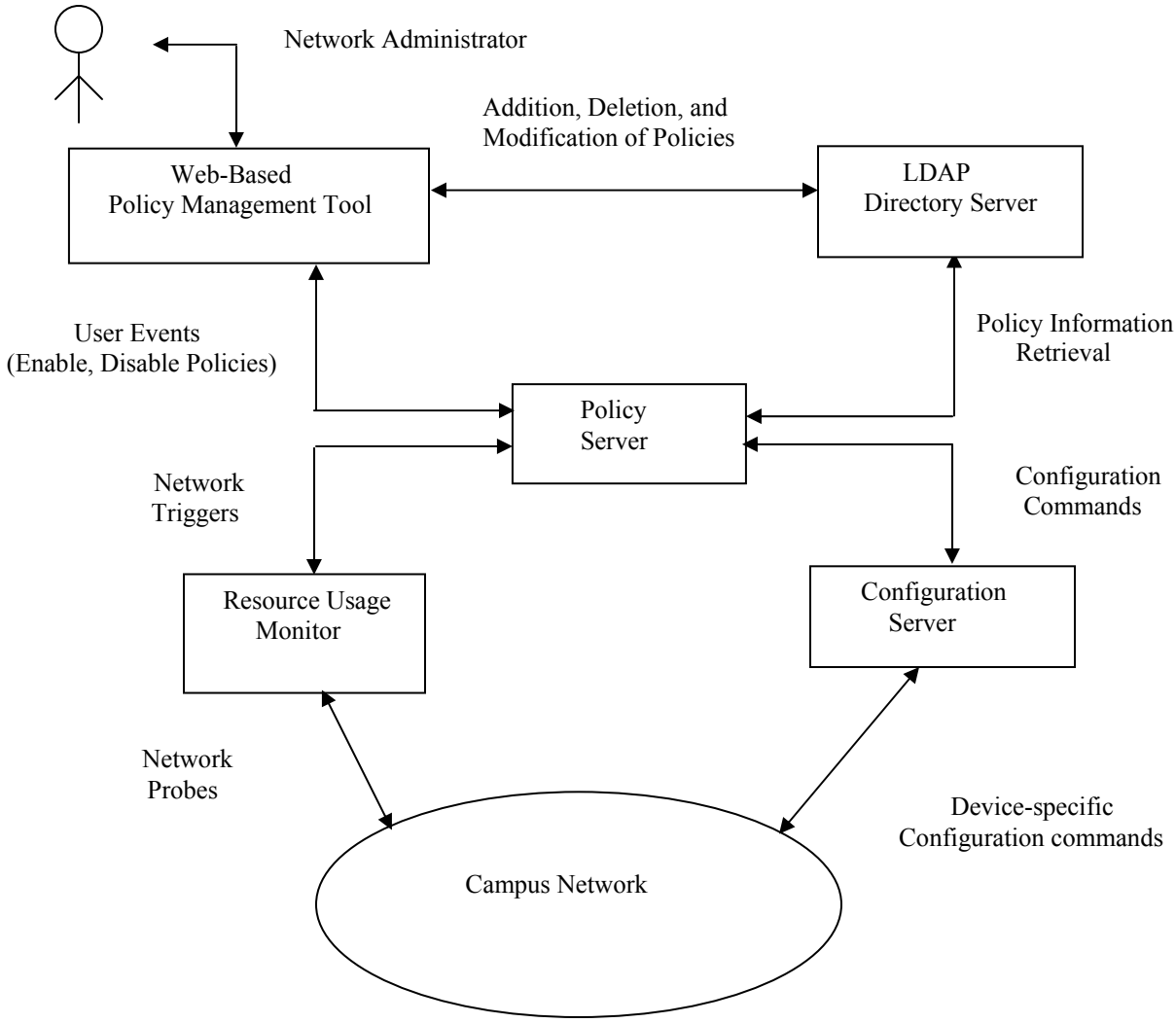


Figure 1.1: Policy Server Architecture

Figure 1.1 shows the architecture of the policy server developed as part of this thesis. The challenge in the weakly distributed model is to efficiently set triggers in the network monitors in order to reduce the processing overhead at the network monitors, and to provide the ability to configure multiple devices at the

same time and do so quickly. The Policy Server (Figure 1.1) developed as part of this thesis uses a weakly distributed policy deployment model. The Policy Server uses a network monitor called Resource Usage Monitor (Figure 1.1) to set network triggers corresponding to policy conditions specified by the network administrator. Other trigger entities can be used as well, provided they are registered with the server and communicate with the deployed server through its API.

1.6.5 Enforcement of Policies

Policy specification is independent of the type of network device, but policy enforcement is device specific. Device independent policies are translated into device specific configuration commands and are then applied to the affected network devices.

The challenge is to provide the ability to perform translation of device-independent policies in a networking environment that has different network devices (routers, switches etc.) with different capabilities. The network may also have network devices from multiple vendors. The translation is performed by a software module, which has knowledge of the QoS capabilities of the network device. In this thesis, the Configuration Server (section 4.3.4, Figure 1.1), a component of the Policy Server, performs this translation. The interface to the network device can be a command line interface, simple network management protocol (SNMP) or Common open policy services [24] (COPS) protocol. Currently, the Configuration Server uses command line interface.

1.7 Thesis Layout

Chapter Two of this thesis reviews the State of the Art in Policy Based Networking, and it discusses the objective and value of the research issues tackled by this work. Chapter Three explains the IETF Policy Core Information Model (PCIM) and gives examples of its use. Chapter Four provides a detailed description of the Policy Server Implementation. The design goals of the implementation and the principles used to achieve the goals are presented in Chapter Four. Chapter Five focuses on deployment issues and experimental functional evaluation of the system. It explains how QoS and pro-active network management policies are specified and deployed in the implementation and discusses this implementation in the context

of IETF PCIM. Chapter six summarizes the work, provides conclusion, and identifies areas for future research. A complete set of policy variables, policy values, policy actions, policy rules, and policy groups specified for the implemented Policy Server evaluations is documented in the Appendix.

2. STATE OF THE ART IN POLICY BASED NETWORKING

Work in the area of Policy-based Networking can be classified into two broad sub-areas, namely Policy Languages, and Implementation of Policy Servers.

2.1 Policy Languages and Models

Policy languages are developed to address the policy specification challenge as discussed in section 1.6.2. The following subsections provide a brief overview of different policy languages that are currently used.

2.1.1 Ponder Policy Specification Language

Ponder [18], is a declarative object-oriented language for specifying security and management policies for distributed systems. It was developed in the Imperial College of Science, Technology and Medicine (London, UK). In the following text, the term "subject" is used to refer to a human or computer agent that performs actions and the term "target" is used to refer to the device that receives the action (e.g. firewall, computer, router etc.). This language provides three basic policies,

- Authorization policies. Those are target based policies that specify what a subject is permitted or NOT permitted to do to a target. These policies are used to protect the target from unauthorized users.
- Obligation policies. Those are subject based policies that specify what a subject must to do on occurrence of an event. For example, these policies can be used to specify actions to be performed when a network node becomes non-operational.
- Delegation Policies. Those are policies that specify what actions a subject can delegate to another subject. For example, a policy may specify that a Network Administrator who has permission to reset, enable and disable network devices can only delegate the enable action to another network administrator, e.g., when he/she is taking time-off from work. This policy will allow another administrator to bring up a network node on account of failure, but restricts him/her to reset or disable a correctly working network device.

Ponder uses the concept of roles to group subjects based on the organizational structures. Examples of the roles are Network Administrator, Employee, Executive, etc. A specialty of this language is its ability to detect modality and semantic policy conflicts.

Modality conflicts are caused by an overlap of subjects, targets and actions. For example, there can be two policies of which one policy allows a subject to access a target, while the other denies access to the same target. Modality conflicts are detected using syntactic analysis.

Semantic conflicts are categorized into three types a) separation of duty, b) self-management (for example, a manager cannot authorize his own expenses) and c) conflict for resources (for example, no more than 25 users can access an Enterprise Application Server at the same time). Semantic conflicts are detected by specifying constraints on a set of policies. These constraints specify what policies can coexist. The constraints are stored as meta-policies (policies about policies).

The Ponder policy compiler performs syntax and semantic analysis and translates the policies into run-time representations like Java classes, XML documents, and Firewall filters. These runtime representations are then sent to policy enforcement entities. A policy server implementation using Ponder language is discussed in section 2.2.1

In my opinion, the specification of policies must be simple and intuitive for a network administrator. Using complex policy languages like Ponder may make the process of analyzing policies for consistency difficult and may lead to intractable policy decision algorithms. A simpler and more limited language that specifies policy as a set of conditions and set of actions, where the policy conditions are used to index the set of actions to be performed, is a better solution since it allows us to specify policies that are easier to analyze and understand. A simpler language also enables and eases storage of policies in a common repository like LDAP directory, thereby providing better interoperability.

2.1.2 Path-based Policy Language

Path based Policy language [19] (PPL) is designed to support policies that can be applied to both the Differentiated Services and Integrated Services QoS models. By using this language, QoS and security policies can be associated with an explicit path through the network. It is flexible enough to support both path and non-path traffic flows. A Path based policy may specify an absolute path that a particular traffic must take, for example a PPL policy can specify that all out-going traffic must be forwarded to a specific node acting as a firewall. Policy conflicts are detected by translating PPL policies into formal logic and then a theorem prover is used for problem detection. The priority of the creator (user) is used to resolve policy conflict.

A summary of PPL constructs.

policyID<userID>@{paths} {target} {conditions} [{action_items}]

policyID –unique policy identification token

userID –user ID of policy creator

paths –network paths the policy affects

target –target class of network traffic

conditions –any global conditions (items are ANDed)

action_items –for setting parameters (e.g., policy priority)

action_item = [{condition}:] {actions}

For example, lets consider a PPL policy,

Policy1 <net_manager> @ {<1,2,5>}

{class = {faculty}} {*} {priority := 1}

This is a policy specified by a user named “net_manager”. The policy states that the path starting at node 1, traversing to node 2, and ending at node 5 will provide high priority for user class “faculty” users. This example shows the capability of the language to specify an explicit path for a traffic flow. Another example,

Policy2 <Betty> @ {<1, *, 5>}

{traffic_class = {accounting}}

{day != Friday : priority: = 5}

This policy specifies that on all paths from node 1 to node 5, accounting class traffic will be lowered to priority 5 unless it is a Friday. Path based polices can reduce the number of policies associated with a domain by providing the ability (via “wild card notation”) to assign a single policy to multiple paths with one statement. For example, path <1, *, 6> specifies all possible paths from node 1 to node 6.

A disadvantage of PPL is that it does not support hierarchical policies and PPL policies cannot be grouped together. The concept of roles is not used in Path based language. This language uses the name of the policy creator (userid) to resolve policy conflict. Hence, this approach can only resolve conflicts among policies that are specified by different network administrators (users), it cannot be used to resolve policy conflicts among policies specified by the same network administrator.

However, path-based policies provide a complexity advantage. When a policy server associates policies with nodes rather than paths, a valid path must be constructed for each new request from a policy enforcement point (e.g. a router). This construction not only uses node connectivity information to build the possible paths, but applies the policy information from each node as well. When a path generation request involves a combination of service constraints, such as acceptable delay and acceptable throughput, this becomes an NP-complete problem. Path-based policies associated with a completely instantiated path are analogous to static routes. Rather than calculating a route through the network, a valid route may be specified ahead of time. When a path is specified in advance with the proper policy constraints, it will accelerate the response to a path request.

2.1.3 Policy Description Language

Policy Description Language [20] (PDL) is developed at Bell Labs for specifying policies that control packet voice gateways and soft switches in packet telephony networks. It is an event-based language. Events can be of primitive or complex type. Primitive events generated by the environment are called System defined events. Events that are triggered by the policies according to policy defined event propositions are called Policy defined primitive events. Policies are specified as conditional statement: "event **causes** action **if** condition" i.e. perform the action if the event occurs under the specified condition.

Complex events can be specified using primitive events. For example, a sequence of primitive event "loginFail", "loginFail", "and loginFail" can be used to form a complex event that represents "three consecutive un-successful attempts to login". This is a major advantage of this language.

2.1.4 Policy Core Information Model

Policy Core Information Model [21] (PCIM) provides a simple object-oriented information model for representing policy information. PCIM is under development in the IETF Policy Framework Working Group and Distributed Management Task Force (DMTF).

PCIM model defines two hierarchies of object classes, namely Structural classes for representing policy information, and Association classes that indicate how instances of the structural classes are related to each other. The policy specification has been designed to assist human administrators in their definition of policies to control a networking environment. The PCIM specification has been developed to be independent of any particular data storage mechanism and access protocol.

PCIM defines policy conditions, policy actions, policy groups and policy rules. The Policy Core Information Model Extensions [22] (PCIMe) define new policy classes like Simple Policy Condition, Compound Policy conditions (Conditions with sub-conditions), Simple policy actions, Compound policy actions (Actions with sub-tasks) and Nested rules. PCIMe also introduces the concept of Policy variables and Policy values. A detailed explanation of PCIM classes and examples are provided in Chapter 3.

2.1.5 QoS Policy Information Model

Policy QoS Information model [25] (QPIM) extends PCIM to QoS Policy Management. QPIM is used for representing policies that administer, manage, and control access to network QoS resources. QPIM uses hierarchical organizations of policies and policy information extensively. The hierarchical policy rule definition enhances clarity, policy readability and reusability. It provides hierarchical context for actions. Within QPIM, hierarchy is used to model context or scope for the sub-rule actions (e.g. bandwidth allocation policy actions and drop threshold actions use this hierarchical context). Complex policies are represented as nested rules that consist of one or more simple rules (building blocks).

QPIM assumes that the QoS policies are independent of any particular device or vendor, thereby enabling networks made up of different devices with different capabilities to be managed and controlled using a single standard set of polices. The use of roles enables a policy definition to be targeted to the network function of a network element, rather than to the element's type and capabilities. The roles provide an efficient and simple method for compact and abstract policy definition. A given abstract policy may be mapped to a group of network elements without the need to specify configuration for each of those elements based on the capabilities of any one individual element. Multiple devices can be aggregated within the same role. For example, if two core network interfaces operate at different rates, one does not have to define two policy rules to express the very same abstract policy (such as allocating 50% bandwidth of the interface bandwidth to a given preferred set of flows).

QPIM also facilitates interoperability among policy systems such as Policy Decision Points (PDPs) and policy management applications by standardizing the representation of policies. Producers and consumers of a QoS policy need only rely on QPIM-based schemata to ensure mutual understanding and agreement on the semantics of the QoS policy. QPIM focuses on both the Differentiated Services and the Integrated Services based QoS policies.

The Policy Server and Policy Management Tool developed as part of this thesis use the PCIM and QPIM to store reusable and non-reusable QoS policies. Differentiated Services actions such as marking, policing and

shaping are specified using QPIM Policy action classes. The use of standardized policy information model enables the Policy Server to be interoperable.

2.2 Currently Implemented Policy Servers

This section provides a brief overview of some relevant Policy Server implementations and shows how various research groups have addressed the Policy management, deployment and enforcement challenges discussed in sections 1.6.3, 1.6.4 and 1.6.5 respectively.

2.2.1 Ponder Policy Server [26]

Researchers at Imperial College have developed a Policy Server to deploy policies that are specified using the Ponder language (Section 2.1.1). A central policy server maintains the policy classes created by the Ponder Policy compiler. The policy object maintains the state of the policy and co-ordinates all policy operations acting as a single point for managing concurrent, and possibly conflicting, requests from multiple policy administrators and from domain objects to which the policy applies. Policy objects can be loaded, unloaded, enabled and disabled. The policy server obtains events corresponding to the changes in the memberships of a domain. For example, a new router can be added to a domain, in which then all policies that apply to that network device must be enforced.

The implementation uses the strongly distributed policy deployment model, i.e. policy server sends the policy objects to policy enforcement entities that are closer to the target. Access controllers and Policy Management Agents are used as Policy enforcement entities for enforcing authorization and obligation policies respectively.

Recently (around June 2002), Ponder language [27] has been extended to provide a policy framework for specifying Differentiated Services actions (such as classifying, marking, policing and shaping) and provisioning of per-hop behaviors at the core network devices. But the Ponder implementation does not support pro-active network management policies (policies that are based on variables such as current

bandwidth utilization of a host, maximum number of flows per host). Ponder Policy servers do not support direct interoperability between different policy server implementations since they use their own language (Ponder), and there needs to be a mapping from Ponder to standard information model, like the IETF PCIM, to achieve interoperability.

2.2.2 Policy Server for Centralized Administration of Packet Voice Gateways [20]

This Policy server is developed at Bell labs and uses the PDL discussed in section 2.1.3. The policy server is embedded in a "soft switch", a next generation switch for circuit and packet telephony networks. It has been used to implement policies for detecting alarm conditions, fail-over, device configuration and provisioning, service class configuration and congestion control. Policies are compiled into Java classes and are stored in a directory server. The administrator manually transfers the policy to the policy enforcers (i.e. soft switch). Policy objects can be added to or deleted from the directory server.

The Policy server performs policy evaluations based on the stream of primitive event instances observed. Streams of event instances are called Event histories. Each set of primitive events occurring simultaneously in a stream is called an epoch. Primitive events can be composed of basic events. A basic event is either an intersection expression of the form e_1 &. ……&e_n representing the occurrence of all instances e_1 through e_n in the current epoch, or a union expression of the form e1 |…….|en representing the occurrence of an instance of one or more of the e_is in the current epoch.

This Policy Server compiles the policies into Java classes and stores them in a Directory server. It does not use a standardized information model to store policy information. Hence, the Policy Server will not be able to interoperate with other Policy Servers implemented by other vendors, research groups. Our Policy Server currently does not support policy evaluations based on stream of events.

2.2.3 Routing and Traffic Engineering Server (Bell laboratories, Lucent Technologies)

Routing and Traffic Engineering Server [28] (RATES) is a MPLS Traffic engineering tool. The RATES implementation consists of a policy and flow database, a browser-based interface for policy definition and entering resource provisioning requests, and a Common Open Policy Service [24] (COPS) protocol server-client implementation for communicating paths and resource information on edge routers.

Policy server is a component of RATES server that is used to communicate with ingress nodes to initiate a Label Switched Path (LSP) setup using explicit routing. The policies are in the form of packet classifiers that filter IP packets and redirect them into LSP tunnel bypassing the routing table lookup at the ingress router. The RATE server computes the route for an LSP demand using minimal non-interference routing. Once the path has been determined, the route server communicates with the source of the route and spawns off signaling from the source to the destination for route setup. RATE server uses COPS for managing policy-related state between policy decision point (policy server) and a policy enforcement point (e.g. ingress routers).

RATES uses a relational database as its information store instead of LDAP [29] directories mainly because of the fact that RATES data model requires both data access and updates. Directories provide only efficient data access and are not efficient for data updates when compared to relational databases.

The policies that are supported by RATES server are not time based, thereby requiring the network administrator to provision and un-provision the network resources every time a LSP is setup. For example, it is possible that a LSP may be required only during some time period say from 9:00 AM to 5:00 PM every day, by releasing the LSP the resource allocated to it can be used by other LSPs. The network resource (bandwidth) can be efficiently utilized if the LSP is setup at 9:00 AM and teared down at 5:00 PM automatically without human interruption. Such time-based policies can be specified and deployed by using the Policy Server developed as part of this thesis. The Policy Server component of RATES does not support role based policy management.

2.2.4 Policy Server for Domain Based Internet Security Control [30]

Policy Based Networking has also been used successfully to manage communication security policies associated with multiple network domains and to resolve the policies into security requirements for inter-domain communication.

Security gateways fragment the Internet into security domains, each of which enforces different security policies. The use of security protocols necessitates the need for establishing security associations among communicating end points and en-route security gateways based on the security requirements of communication. This [30] paper proposes a distributed policy management system that serves this purpose.

The distributed system consists of Security Policy Servers, Policy Clients and Policy Databases. Security policy servers maintain, exchange and process security policies associated with security domains in order to determine necessary security associations to protect inter-domain communication. Policy clients and Policy server communicate using Security Policy Protocol [30]. A Policy Client sends query messages to a pre-configured local policy server to determine the security policies required for a particular communication. The local policy server communicates with other policy servers and obtains answer for the policy client's query. The Security policy protocol works like a Domain Naming Service (DNS) protocol.

Policy Servers also resolve their local policies with the policies received in reply messages from other policy servers. The resolution process identifies and resolves ambiguities in how the policies must be enforced. After the resolution process, the policy server updates its local database and sends appropriate message to policy client and policy enforcement entities (security gateways). The authors of this paper [30] have also developed a Security policy specification language for specifying IPSec and ISAKMP policies.

It is worth noting that the Policy Server implemented as part of the current research has the ability to share policy information (using a common information model), and thus can be used to form a distributed system consisting of QoS Policy Servers, each managing a particular domain. Such a system can be used to deploy QoS policies across multiple domains.

2.3 Objectives and value of the work in the thesis

To the best of my knowledge, it appears that currently publicly and commercially available Policy servers are capable of handling only **static** and simple policies. Policy servers that are capable of handling **dynamic** policies (policies based on real-time traffic parameters, such as current load in the network, delay along a path, etc.) are required to perform pro-active network management. Though the specification of QoS policies has progressed greatly, not much work has been done in the deployment of dynamic and hierarchical (complex) QoS policies. The objective of the work reported in this thesis was to "look into the future" and develop and evaluate a traffic-aware, standards-based Policy Server and Policy management tool that allows network administrators to define and deploy policies at the appropriate level of granularity and with highest level of flexibility (which includes dynamic and hierarchical policies).

The Policy Server and the Policy Management Tool developed as part of this work support the following policy characteristics,

1. Simple and Complex Policies
2. Static and Time-Based QoS Policies
3. Dynamic Measurement-based Policies for Pro-active Network Management
4. Re-usable Policies for efficient Policy Management

The toolset developed as part of the work reported here is based on the PCIM [21]. PCIM is described in the next chapter. Following chapters show how the system was implemented and how it can be used, how the functionality of items 1 through 4 above was evaluated, and they discuss practical issues with the IETF PCIM implementation.

3 IETF POLICY CORE INFORMATION MODEL (PCIM)

3.1 Why the IETF model?

There are a number of good reasons to choose the IETF model [21,22] for specifying and storing policies. Principal reasons are its simplicity and ability to use simple policies as building blocks for specifying complex policies. Another big advantage is that it is a standardized model so we can expect the implementers of other policy servers/solutions to use this model, thereby making interoperability a potential reality. Standardized policy model provides policy servers developed by different research centers and vendors, each specialized in a particular domain - say QoS or Security, the ability to share the information model and co-ordinate data and policy elements. Furthermore, the Directory Enabled Networking (DEN) Initiative of Distributed Management Task Force (DMTF) has done an excellent job of mapping CIM [31] to Lightweight Directory Access Protocol (LDAP [29]) schema. The PCIM [21, 22] information model when integrated with CIM provides the network administrator with the highest level of flexibility in assigning policies to different network elements. Another reason for selecting the PCIM model is the development of QPIM [25] model that uses PCIM to specify QoS Policies. QPIM models policies that apply to Differentiated Services (DiffServ) and Integrated Services (IntServ) environment. The Policy Server developed as part of this thesis supports DiffServ QoS policies.

This chapter first introduces atomic components of a simple policy, and then it focuses on using simple policies as building blocks to define complex policies. The assignment of policies to network elements, policy repositories and reusable policies are also discussed.

3.2 What is a Policy Rule

A policy rule consists of a set of **conditions** and set of **actions**. The set of conditions associated with a policy rule specifies when the policy rule is applicable. The set of conditions can be expressed as either an OR-ed set of AND-ed sets of condition statements (Disjunctive Normal Form, DNF) or an AND-ed set of OR-ed sets of statements (Conjunctive Normal Form, CNF). The actions of a policy rule are executed when

the set of conditions associated with the policy rule evaluates to true. Policy condition and policy action form the atomic components of a policy.

For example,

"((A==50 & B==100) | (C==50&D==100))" is a DNF condition, and

"((A==50 | B==100) & (C==50|D==100))" is a CNF condition.

In PCIM, a policy rule is modeled by "PolicyRule" class, which has the following properties,

PolicyRuleName → Name of the policy rule

Enabled → Identifies whether the policy rule is enabled or not

ConditionListType → Specifies whether the list of policy conditions associated with this policy rule is DNF or CNF form.

SequencedActions → Specifies whether the ordering of action must be followed.

Mandatory → Specifies whether the evaluation of rule is mandatory.

In PCIM, conditions and actions are associated with a policy rule using "PolicyConditionInPolicyRule" and "PolicyActionInPolicyRule" aggregation classes.

The PolicyConditionInPolicyRule class contains a property called "groupNumber" which is used to group conditions. Two or more conditions that are to be grouped together are specified the same "groupNumber" property value. The name of the condition is also a property of this class.

For example, lets say that A==50 is called "Condition1" and B==100 is called "Condition2". In order to group Condition 1 and 2 together (i.e. to specify "A=50 & B=100" or "A=50 | B=100"), two PolicyConditionInPolicyRule aggregation objects are formed with the name property set to corresponding condition name and "groupNumber" property set to, say, 1 (same value). The logical operator "&" or "|" is determined by the "conditionListType" property of the "PolicyRule" object.

Similarly, the PolicyActionInPolicyRule class has an action name property and another property called "actionOrder" that specifies the order of the action.

For example, to define two actions A1 and A2 with a rule, two PolicyActionInPolicyRule aggregation objects are formed with the name property set to "A1" and "A2" and "actionOrder" property set to 1 and 2 respectively. This specifies that action A1 must be executed before action A2.

3.3 Policy Variable and Value

A variable represents information that changes. Policy conditions and actions abstract information as Policy variables. Policy variables can be evaluated in logical expressions or can be set by an action.

Implicitly defined policy variables are variables that are evaluated outside the context of PCIM schema and its modeling constructs. Examples of PolicyImplictVariables are PolicySourceIPv4Address, PolicyDestinationIPv4Address, PolicySourcePort and PolicyDestinationPort. Implicit Policy variables have a multi-valued property called "ValueTypes". This attribute is used to specify the type of values that are valid for an implicit variable.

Policy values are used for modeling values and constants that are used in policy conditions and actions. In PCIM, it is modeled as a "PolicyValue" class. The subclasses of the "PolicyValue" abstract class are PolicyIPv4AddrValue, PolicyStringValue, PolicyBooleanValue, PolicyMACAddrValue, PolicyIntegerValue and PolicyBitStringValue. These subclasses define three basic types of values: scalars, ranges and sets. Named policy variables and policy values are reusable.

Let's consider PolicyIPv4AddrValue class. This class has a property called "IPv4AddrList". The value of the "IPv4AddrList" property is specified in ABN form as

```
IPv4address = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv4prefix  = IPv4address "/" 1*2DIGIT
```

IPv4range = IPv4address"-"IPv4address

IPv4maskedaddress = IPv4address","IPv4address

For example, to specify “152.1.2.2 “ as the value of “PolicyIPv4AddrValue” object, the “IPv4AddrList” property of the object is set to “IPv4address = 152.1.2.2”. To specify the subnet mask along with the IP address, the IPv4addrList property of the PolicyIPv4AddrValue object is set to “IPv4maskedaddress=152.1.2.2,255.255.255.255”.

3.4 Policy Conditions

There are three types of conditions namely Simple conditions, Compound conditions and Time conditions.

3.4.1 Simple Conditions

Simple conditions are of the form {policy variable; logical relation; policy value}. For example “SourcePort == SIP_PORT” is a simple condition with “SourcePort” as the Policy variable and “SIP_PORT” as the Policy Value. An “equal-to” logical operator is used in this example.

3.4.2 Compound Conditions

Simple conditions form the basic building blocks of compound conditions. A Compound Policy condition represents a Boolean combination of simpler conditions. In PCIM, a compound condition is modeled as a “PolicyCompoundCondition” class. A property of this class called “conditionListType” is used to specify whether the condition is a CNF or a DNF of its sub conditions. A policy condition is aggregated in a compound condition, using “PolicyConditionInCondition” aggregation object. This object is similar to “PolicyConditionInPolicyRule” object and has the same “groupNumber” property.

For example, “(A == 50 & B==50)” is a compound condition, with two simple conditions A==50 and B==50.

3.4.3 Time Condition

In PCIM, the “PolicyTimePeriodCondition” class provides a means of representing the time period during which a policy rule is valid. At all times that fall outside these time periods, the policy rule has no effect. A policy rule is treated as valid at all times if it does not specify a time condition.

The “PolicyTimePeriodCondition” class has the following properties,

- *TimePeriod*: This property identifies an overall range of calendar dates and times over which a policy rule is valid. It is formatted as a string consisting of a start date and time, then a colon (':'), and followed by an end date and time. The first date indicates the beginning of the range, while the second date indicates the end. Thus, the second date and time must be later than the first. Dates are expressed as sub strings of the form "yyyymmddhhmmss". For example, a value 20020609091700:200206102091700” represents an overall validity range from “09-Jun-2002 9 Hrs 17Mins“ to “20-Jun-2002 9 Hrs 17 mins”.
- *MonthOfYearMask*: Specifies the months for which the policy is valid.
- *DayOfMonthMask*: Specifies the days in a month for which the policy is valid.
- *DayOfWeekMask*: Specifies the days in a week for which the policy is valid.
- *TimeOfDayMask*: Specifies the range of time periods in a given day during which the policy is valid. For example, a policy rule may be valid from 6:00 AM to 6:00 PM in all its valid days.

3.5 Policy Actions

There are three types of policy actions namely Simple Actions, Compound Actions and Vendor-Specific Actions.

3.5.1 Simple Actions

In PCIM, the “SimplePolicyAction” class models the elementary set operation (SET <variable> TO <value>). A good example for a Simple policy action is specifying a DiffServ marker action. For example,

"Set DSCP To HIGH_PRIORITY" is a simple action that specifies that DSCP variable (an object of PolicyDSCPVariable class) must be set to the policy value HIGH_PRIORITY. The value stored in HIGH_PRIORITY policy value may be "EF".

3.5.2 Compound Actions

A compound action is used to represent a sequence of actions that are to be applied as a single atomic action within a policy rule. A compound action class ("CompoundPolicyAction") has a property called "PolicyExecutionStrategy" that specifies whether all sub-actions are executed (Do-All) or sub-actions are executed until success (Do-until-success) or sub-actions are executed until failure (Do-until-failure).

3.5.3 Vendor-Specific Actions

Vendor-specific actions are used to represent policy actions that have not been modeled in PCIM. The "VendorPolicyAction" class has two properties ActionData and ActionEncoding that are used to define the content and format of the action.

3.6 Nested Policy Rules

Nested policy rule provides the ability to define policy rules with in rules. A nested policy rule has a parent rule and one or more sub-rule(s). Nested policy rules are used to define complex policies. The parent rule must evaluate to TRUE for the sub-rules to be evaluated. If the parent rule evaluates to false, the sub-rules are not evaluated.

For example, let's consider the following nested policy rule,

If (IPv4Address==EXEC_IP_ADDRESS) then Permit_Access_Action

If (IPProtocol == UDP && SourcePort ==SIP_PORT)	Set DSCP 6
If (IPProtocol == TCP && SourcePort ==TELNET_PORT)	Set DSCP 3
If (IPProtocol ==TCP | IPProtocol == UDP)	Set DSCP 1

This rule specifies that an Executive's Flow is allowed to enter the network (specified by the parent rule by Permit_Access_Action) and it also specifies the type of priority (DSCP value) to be set depending on the source port of the flow.

3.7 Policy Groups

Policy rules are aggregated to form policy groups. A policy group can also have one or more policy group(s). A policy group contains either a set of rules or a set of policy groups but not both. Nested policy groups are used to represent hierarchy of policies. For example, all access-list policy rules that apply to the interfaces of an ingress network device can be grouped as "Ingress-Access-Policies" policy group.

3.8 Priorities, Decision and Execution Strategies

Every policy rule and policy group has a unique system priority. The components of a policy group are associated with another priority within the scope of the policy group. This priority is used to specify the order of evaluation of rules contained in the policy group. Higher priority rules in the policy group will be evaluated first. The decision strategy specifies the type of evaluation for a policy. A "First Matching" strategy specifies that the components of the policy group be evaluated until a matching rule (i.e. conditions of the rule is true) is found. An "All-Matching" strategy specifies that all components of the policy group must be evaluated irrespective of whether a matching rule has been found or not. The use of Priorities and Decision strategies make the Policy Evaluation more predictable and deterministic. Decision strategies also apply to policy rules, if they contain sub-rules (Nested policy rule).

Execution strategies are used to specify the order of execution of actions. A "Do until success" strategy specifies that the actions of a rule be performed until the execution of an action is successful. A "Do All" strategy specifies that all actions of the rule be executed. A "Do until failure" strategy specifies that all actions of the rule be performed until the execution of an action is unsuccessful. Execution strategies are very useful in specifying device independent policies. For example, let's consider a network consisting of both DiffServ routers and non-diffserv routers. To specify that a particular traffic be marked with expedited priority (EF), a single rule with two actions "Set DSCP EF" and "Set IPPRECEDENCE

CRITICAL_PRECEDENCE" can be specified with a decision "Do until success" strategy. While configuring a non-diffserv router, the policy server will first try to set DSCP (unsuccessful action) and then sets IPPRECEDENCE (successful action). By using the action execution strategy, we have avoided the need to specify and manage two different rules, one for DiffServ router and other for non-DiffServ router.

3.9 Policy Roles

A role is a type of attribute that is used to select one or more policies for a set of entities and/or components from among a much larger set of available policies. For example, Network devices and their interfaces can be assigned to one or more roles. The concept of roles makes the process of assignment of policies to network elements simple and efficient. For example, suppose if a policy for N routers has changed, instead of modifying N policies, the administrator will be required to change only one policy if this policy is associated with a role to which these network devices are associated. The concept of roles also reduces the complexity of Policy Conflict Detection problem (Section 4.2.4).

A role-combination is a set of attributes that are used to select one or more policies for a set of entities and/or components from among a much larger set of available policies. The process is equivalent to a logical ANDing of each attribute, such that the set of policies that are selected is defined by the intersection of all the roles in the role-combination. For example, all policies that apply to Ethernet interfaces in the "Main Campus" network can be assigned a role-combination "Main_Campus&&Ethernet". There may be other policies that specify policies for roles "Main_Campus" and "Ethernet" separately.

3.10 Reusable Policy Elements

In order to avoid redundancy in storing policies and to make effective use of the Information model, all policy elements are reusable (i.e. policy variable, policy values, policy conditions, policy actions, policy rules and policy groups). Reusable policies are stored separately in the Policy repository. Reusable elements can be grouped together, for example all reusable policy variables can be grouped and placed in a separate reusable container (part of the repository). To retrieve a reusable element, we need to search for

the policy elements only in the corresponding reusable container thereby making the retrieval process faster.

3.11 Advantages and Disadvantages

Advantages

- Simple and intuitive
- Simple actions can be aggregated to form compound actions
- Action Execution strategy (do-all, do-until-success, do until failure) for multiple actions associated with a policy rule can be specified. Alternative actions can be specified using do-until-success strategy.
- Differentiated Services and Integrated Services actions are modeled
- Deterministic policy evaluation can be achieved by specifying First matching or All matching decision strategies
- Role based Policy Management i.e policies are mapped to network devices using roles
- Simple Priority based conflict resolution
- Policies are specified independent of type and manufacturer of network device
- Provides the ability to group interrelated policies
- Provides the ability to specify hierarchical/complex polices using simple policies as building blocks
- Reusable policy elements provide efficient policy management
- Independent of policy repository (policies can be stored in LDAP, relational databases, as XML schemas depending upon the requirements)

Disadvantages

- Specifies policies only for a single administrative domain, policies that span multiple administrative domains are not addressed.
- Simple priority mechanism is not sufficient to resolve all policy conflicts. For example, knowledge about which policies can co-exist (policy constraints) is necessary to resolve

semantic policy conflicts. The model must allow the network administrator to specify such information.

This chapter discussed the reasons for choosing the IETF Policy Core information model (PCIM). It illustrated the information model with examples and showed how simple policy rules can be combined to form nested rules that represent complex policies. This chapter also discussed the advantages and disadvantages of IETF PCIM. A BNF grammar of the IETF Policy Core information model and QoS Policy core information model is provided in Appendix section. The next chapter describes the policy server architecture, and provides implementation details that show how the design goals were achieved.

4 ARCHITECTURE

4.1 Design

This chapter describes the design of the policy serving system developed as part of this work. Policy Management Tool and Policy Server are the main components of the system. The general system architecture is shown in Figure 4.1.

The Policy Manager (which has a web-based user) communicates with an LDAP-based server that stores policies and other meta-data, and with the Policy server that reconciles the policies and acts on the decisions. Networking node-specific configuration commands are delivered to the managed devices via the configuration server. Events that server needs to react to (such as sudden congestion information, a DoS attack signature, etc.) are delivered to the server from one or more event servers. The Figure 4.1 shows NCSU Resource Usage Monitor interfacing through HTTP based interface.

The Policy Server has four sub-systems namely Event Handler, Policy Evaluator, Action Manager and Configuration Server (Figure 4.11). **Event Handler**, through its web interface, handles user events like Enable, Disable, Modify, Delete Policies. Timer Handler is used to manage time-based policies. It triggers internal events corresponding to start and stop validity of time periods for policies. Trigger Manager is used to configure triggers and process trigger notifications from external resources like network monitors (e.g. Resource Usage Monitors).

a) **Policy Evaluator** performs evaluation of enabled policies based on user events, timer events, triggers that are received from corresponding event handlers.
b) The actions and roles of matched policies are sent to **Action Manager**. Action Manager uses roles to find the network elements to which the actions are to be applied. The action manager maintains one configuration thread per network device.
c) Configuration threads use **Configuration Server** to communicate with different network devices like switches, routers with different capabilities and from different vendors.

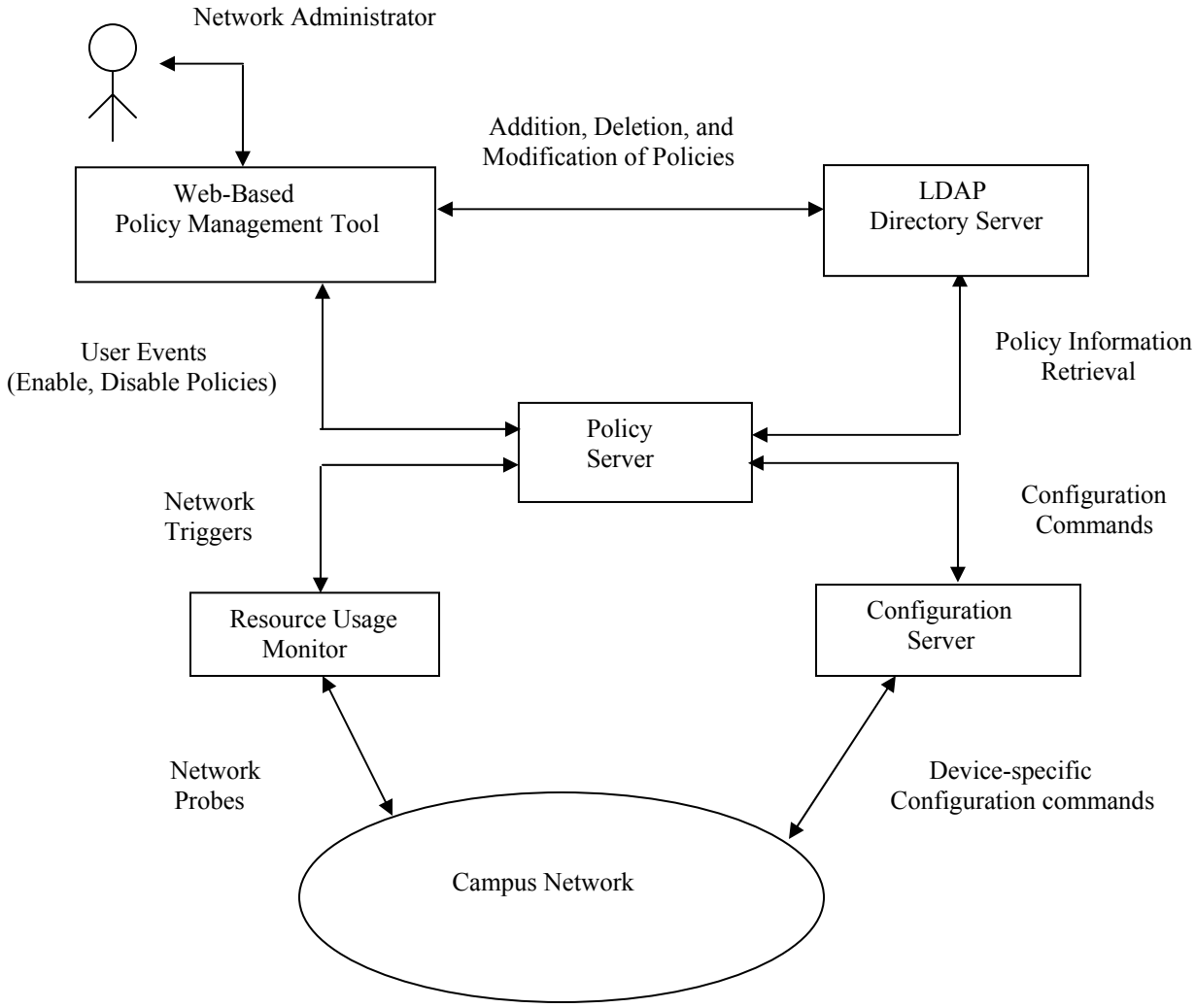


Figure 4.1 Policy Server Architecture

The principal design goals of the system include

- ***Good Performance***

  The performance of a Policy Server is very important from the Administrator's perspective. Some of the performance measures are how quick the policy server responds to a trigger, the ability to enforce policies in a large network, etc. Some of the factors that determine the performance are amount of memory available, CPU power, time needed to evaluate policies, and so on. Global policy table and Event aggregation methods were used to achieve this goal. Sections 4.3.1.1,

4.3.1.2 and 4.3.1.3 explains the how this goal was achieved.

- ***Modular Design to support Future Extensions***

  The policy server is designed to support future extensions. For example, at some point in time, we may want to use COPS to configure network devices rather than using the command-line interface. This goal was achieved by developing a Configuration Server, which takes the responsibility of configuring the network devices according to the policy actions of the matching rule. The configuration server provides an API that the policy server can use to issue configuration requests. Any changes to the communication protocol with the network device (SNMP/COPS/Command line interface) will require code changes only at the configuration server. The configuration server may use different protocols for communicating with different devices.

- ***Support for Complex Policies***

  One of our goals was to provide the administrator the ability to define complex policies using simple policies as building blocks. This goal was achieved by using the concept of nested rules and policy groups defined in Policy Core Information Model Extensions (PCIMe). [22]

- ***Interaction with Multiple Network Monitors***

  The policy server must be able to interact and gather network information from multiple network monitors, one that monitors bandwidth usage, another that monitors security intrusion, and the other that monitors delay, jitter of a particular flow.

- ***Minimal Configuration Updates***

  Another goal was to reduce the number of configuration updates made to network devices. The network devices may not be able to handle very frequent configuration changes. Section 4.3.1.2 and 4.3.1.3 discuss how this goal was achieved.

- ***Intelligent Threshold Setting***

  It is also essential that the policy server does not affect the performance of network monitors. For example, if there are N policies that are based on same trigger, then the policy server should be able to identify that and set only one trigger rather than N triggers. Section 4.3.1.3 discusses how this goal was achieved.

- ***Concurrent Network Device Configuration***

  The Policy Server must be able to configure multiple network devices at the same time. Configuring devices sequentially instead of concurrently will impact the performance of the Policy Server (Policy enforcement time will be high). This goal was achieved by using multiple threads called Configuration threads that are managed by the Action Manager. Section 4.3.3 and 4.3.3.1 discuss implementation details in regard.

- ***Security***

  Security was also one of major concerns. The objective was to develop a secured interface to Policy Management tool and to retrieve network information from network monitors securely. Section 4.2.5 and 4.3.1.3 discusses how security issues were handled.

4.2 Policy Management Tool

One of the main requirements of a policy based management system was the ability to create, modify, delete, enable and disable the policies. The Policy Management tool was developed to satisfy this requirement. The Policy Management tool stores the Policy information in a Lightweight Directory Access Protocol directory (LDAP [29]), and communicates with the Policy Server. It provides a web-based interface for network administrators and policy writers.

4.2.1 LDAP Policy Repository

The Policy Management Tool and Policy Server use an LDAP directory as their policy repository. There are several of reasons for choosing LDAP. First, it provides a flexible directory schema and supports multi-

valued attributes. LDAP access protocol can be used to perform sophisticated directory searches using attribute-value based queries. LDAP directory provides superior performance than traditional database systems for search operations.

Although an LDAP directory provides faster lookups/searches, it is not good at modifying existing data. It is suitable only for applications that would require only infrequent modifications to its data. In a Policy Management System, typically policies, once defined, are not changed very frequently by network administrators. But the lookups/searches performed by the policy server are more frequent. Hence, the amount of time taken to retrieve information about a policy and its components is a major factor that affects the performance of the policy server. Using an LDAP directory thus will increase the performance of the policy server.

Another reason for using LDAP is that it is widely used to store information about users and network resources. The policy server needs to know information about users or network devices to identify what policies belong to which user or network interface. For example there can be policies like "Give Gold Service to users in Computer Science Department", "Apply Priority Queuing at all Ethernet Interfaces", in which case the policy server needs to find users who belong to Computer Science department and network devices that have Ethernet interfaces. Hence by using LDAP Directory as our common repository to store user, network device information, policies, we can eliminate the need to manage different types of informational databases (For example relational databases).

In this project, OpenLDAP's stand-alone LDAP server was used to store policy information. IETF's Policy Core LDAP [34] schema was used as directory schema. It only specified the LDAP schema for Policy Core Information Model (PCIM [21]). New LDAP class definitions were added to support Policy Variables, Policy Values, Nested Rules, Compound conditions, Compound actions and other extensions as specified by PCIMe [22]. OpenLDAP server runs in both Linux and Windows Platforms. Novell's JLDAP (LDAP Class libraries) was used to retrieve policy information from the OpenLDAP server.

4.2.2 Management of Simple Rules, Nested Rules and Policy Groups

The Policy Management Tool provides a web-based interface for network administrators to manage policies. Users can create new policies, modify existing policies and delete existing policies. A policy can be a simple rule or a nested rule or a policy group.

Creating a Simple Rule:

While creating a simple rule the administrator specifies the role(s) or role combination(s) to which the policy applies. The user can also select the type of the condition combination (Disjunctive Normal Form (DNF) or Conjunctive Normal Form (CNF)), execution strategy, decision strategy and other rule properties (Figure 4.2)

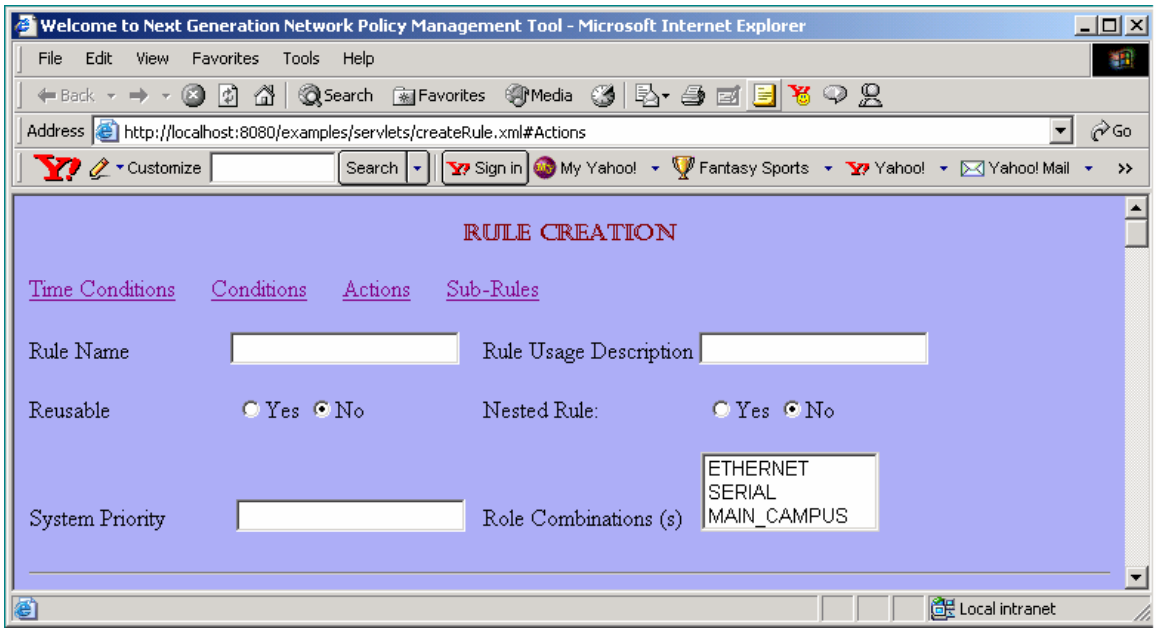


Figure 4.2: Specifying properties of a rule using Policy Management Tool

The user can either create a new condition or reuse a condition. If the user wishes to create a new condition, the policy variable and policy values must be provided. Pre-defined reusable policy variables and policy are reused in conditions and actions (Figure 4.3). Actions and conditions can either be of simple or compound type.

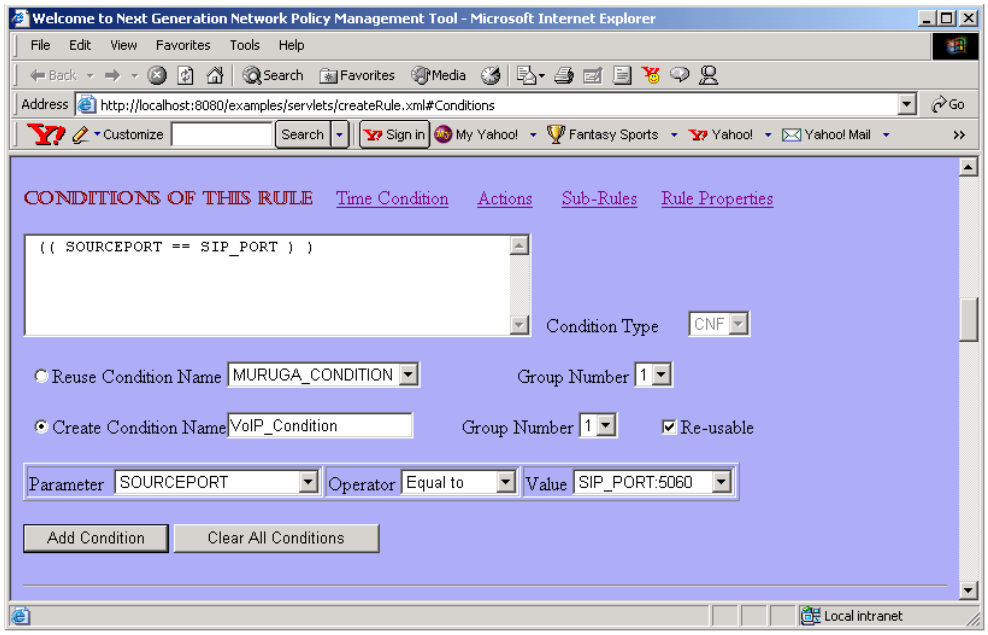


Figure 4.3: Specifying conditions of a rule using Policy Management Tool

Time conditions can be used to specify an overall validity range and specific active time periods within overall validity range (Figure 4.4).

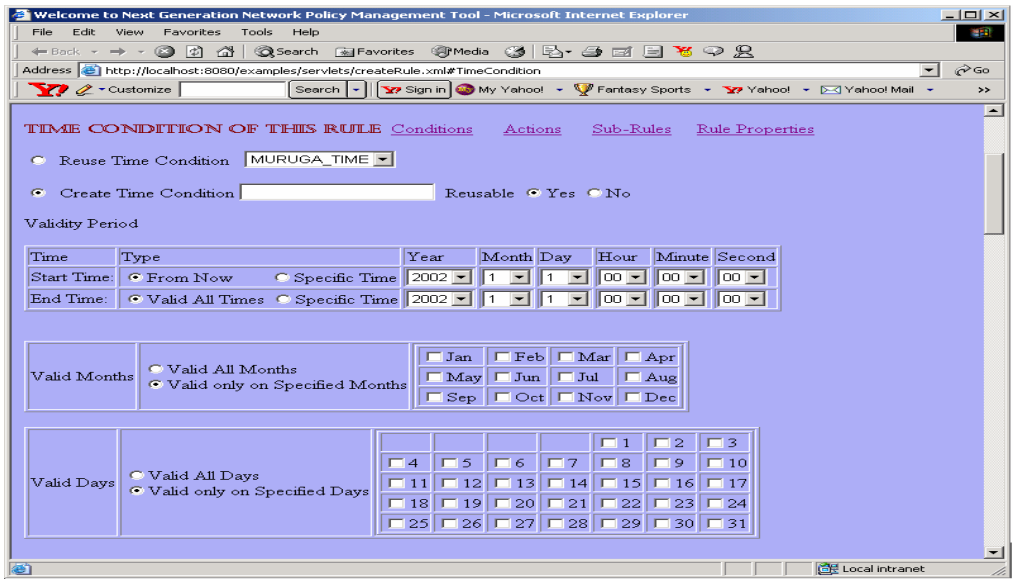


Figure 4.4: Specifying Time Conditions of a Rule using Policy Management Tool

Multiple actions can also be specified for a rule. The user can specify the order of actions by specifying an action order number for each action. This order number need not be unique or continuous. As the user adds an action or a condition to a rule, the rule is updated and displayed to the user in a human readable text

form. After specifying the rule, the user saves the rule by pressing the save button. The user can specify either "Do until success", "Do until failure" or "Do All" action execution strategy (Figure 4.5).

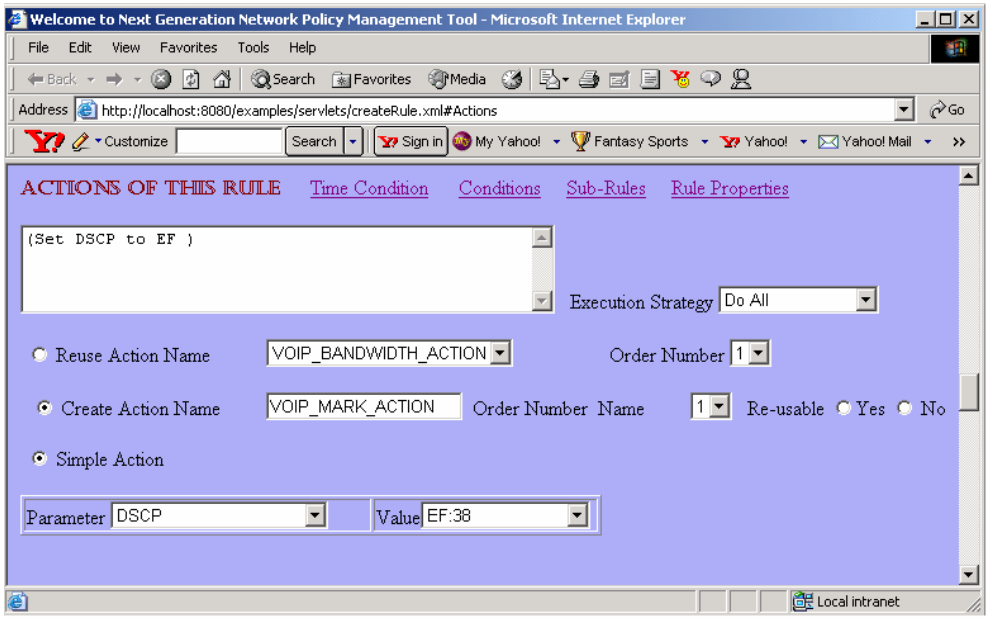


Figure 4.5: Specifying actions of a rule using Policy Management Tool

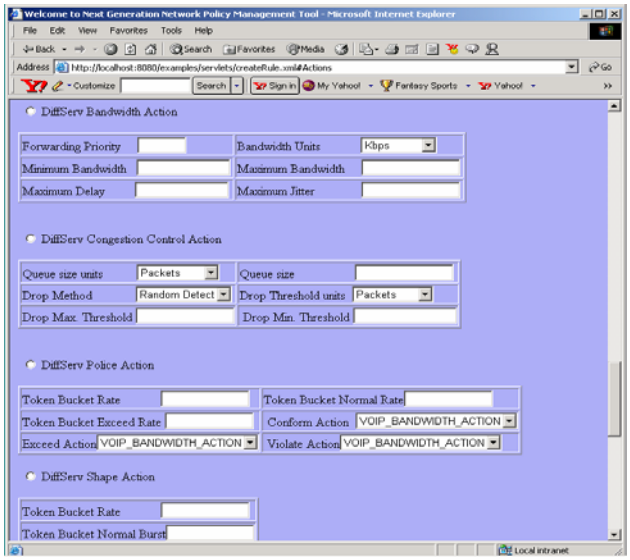


Figure 4.6: Specifying DiffServ Actions using Policy Management Tool

The user interface provides the ability to create simple policy actions and Diffserv policy actions. DiffServ Marking action is specified as simple policy action. Policy actions that specify Bandwidth allocation and Congestion control mechanisms can also be created (Figure 4.6)

Creating a Nested Rule:

The user is presented with a web page similar to Simple Rule creation web page. The sub-rules of a nested rule are pre-defined reusable policy rules. The user can select either First-Matching or All-Matching decision strategy for the nested rule. The user also specifies the name and unique priority for each sub-rule. This unique priority value refers to the priority of the sub-rule within the scope of this nested rule. The user is presented the parent rule and the sub-rules (in descending order of priority) in a human readable text form.

Creating a Policy Group:

The user specifies the role or role combinations to which the policy group applies and its decision strategy (First-Matching or All-Matching). The name of each component of the policy group is specified along with a unique priority (Figure 4.7).

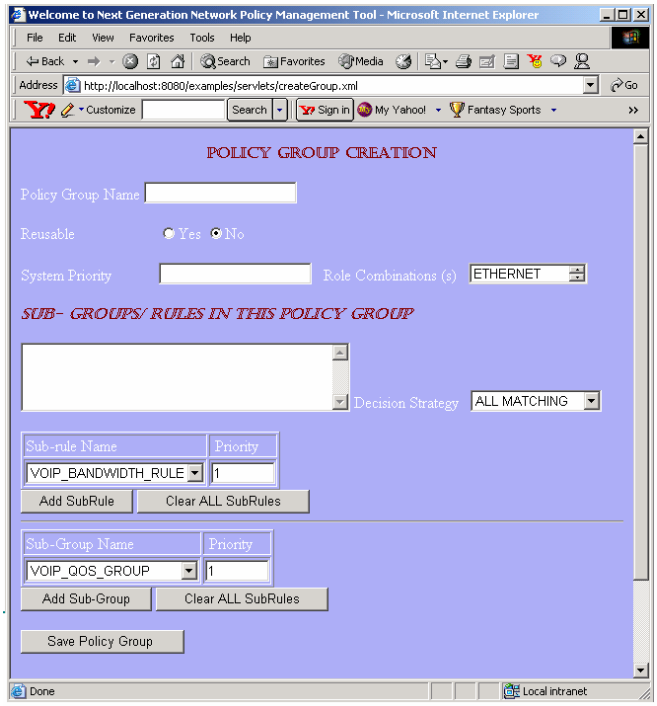


Figure 4.7: Creating Policy Groups using Policy Management Tool

Re-usable Policy Components:

While creating a policy condition, policy variable, policy value, policy rule, policy group the user is provided the option of specifying it as reusable. All reusable policy components are stored in their respective reusable LDAP policy containers. To increase performance, separate reusable containers are used. For example, when the policy server needs to retrieve a reusable policy variable, it searches only in the corresponding reusable variables container rather than searching the entire directory.

Using Policy Browser to Enable/Disable/Delete/Modify Policies:

All policies specified by the user are displayed in a hierarchical form in Policy Browser web page (Figure 4.8). Users can navigate policies by selecting the appropriate links. The user can also view a policy in a human readable text form by selecting the "View " link. The user can enable or disable or modify or delete a policy. When a policy is enabled or disabled a system message is sent to Web Handler component (Section 4.3.1.1) of the policy server. System message is sent only when an enabled policy is modified or deleted.

Modification of Policies:

The user can modify existing policy rules and policy groups. In the case of policy rules, time conditions, actions, conditions and rule properties (decision strategy, action strategy, roles) can be modified. If an action/condition is reusable, the user cannot modify the condition/action, but can add/remove them to/from the existing policy rule. The user can modify the policy variables and policy values used in non-reusable conditions and actions, this provides the user with the flexibility of completely changing the set of conditions and actions associated with a policy rule. In the case of policy group, the user can add or remove policy rules in the group at any point in time. The user can also modify the priority associated with each rule in the policy group thereby the user has the ability to change the order of evaluation of policy rules in the policy group. Figure 4.9 and 4.10 show how the user can modify conditions and actions associated with a policy rule respectively.

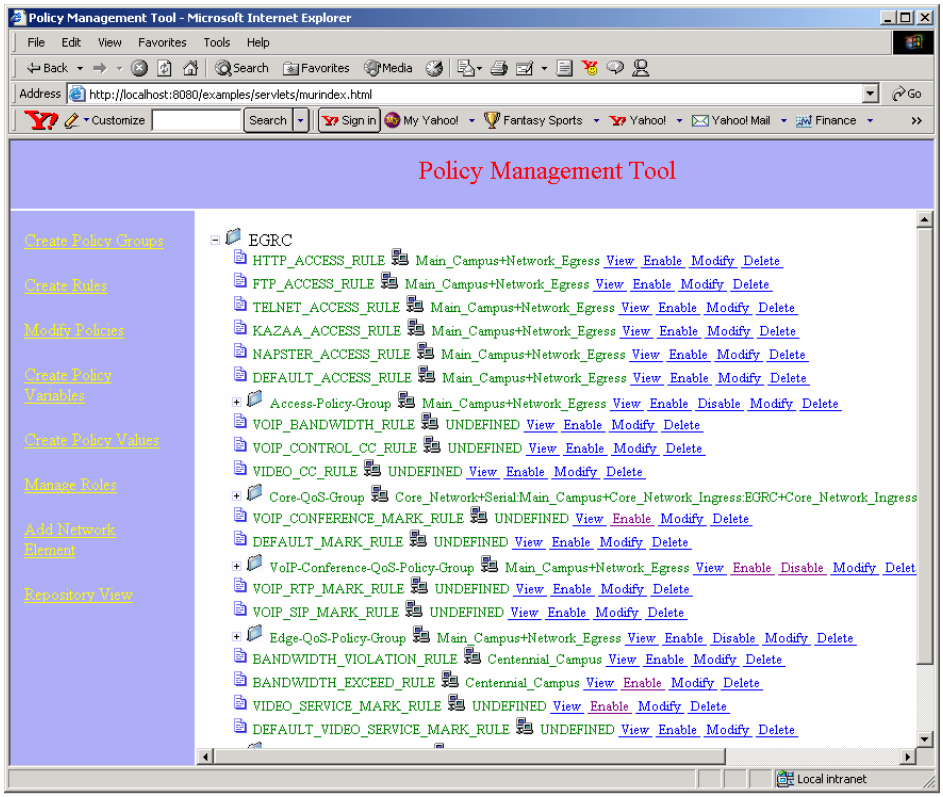


Figure 4.8: Policy Browser

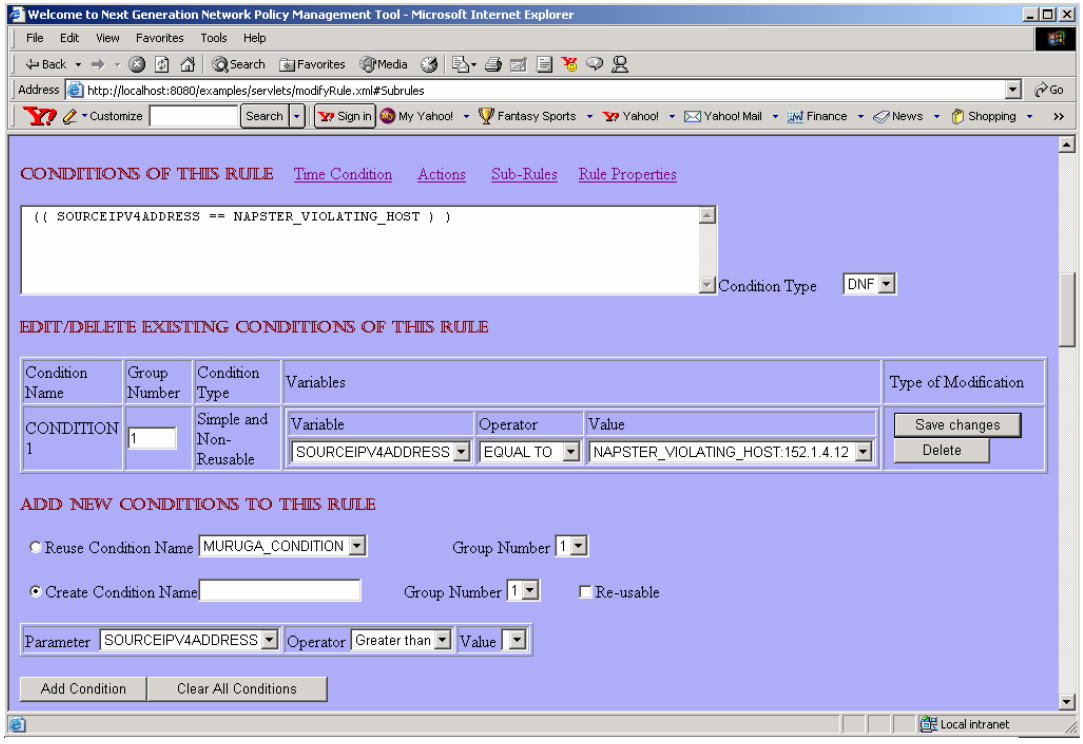


Figure 4.9: Modifying existing conditions of a Policy rule using Policy Management Tool

As the user modifies the policy rule, Policy management tool displays the current conditions, actions associated with a policy rule in a text box immediately. This feature was implemented using XML and JavaScript (XML for SCRIPT parser). When the user wants to modify a policy, the management servlet retrieves the policy from LDAP repository and sends it to the policy client as XML. Policy client uses this XML to display the conditions/actions associated with the policy, it then makes modifications to this XML according to the modifications performed by the user. Once the user has finished all modifications, it sends the modified XML to the Servlet. The management Servlet processes the modified XML and performs required actions i.e. it creates/deletes/modifies condition, actions, rule properties.

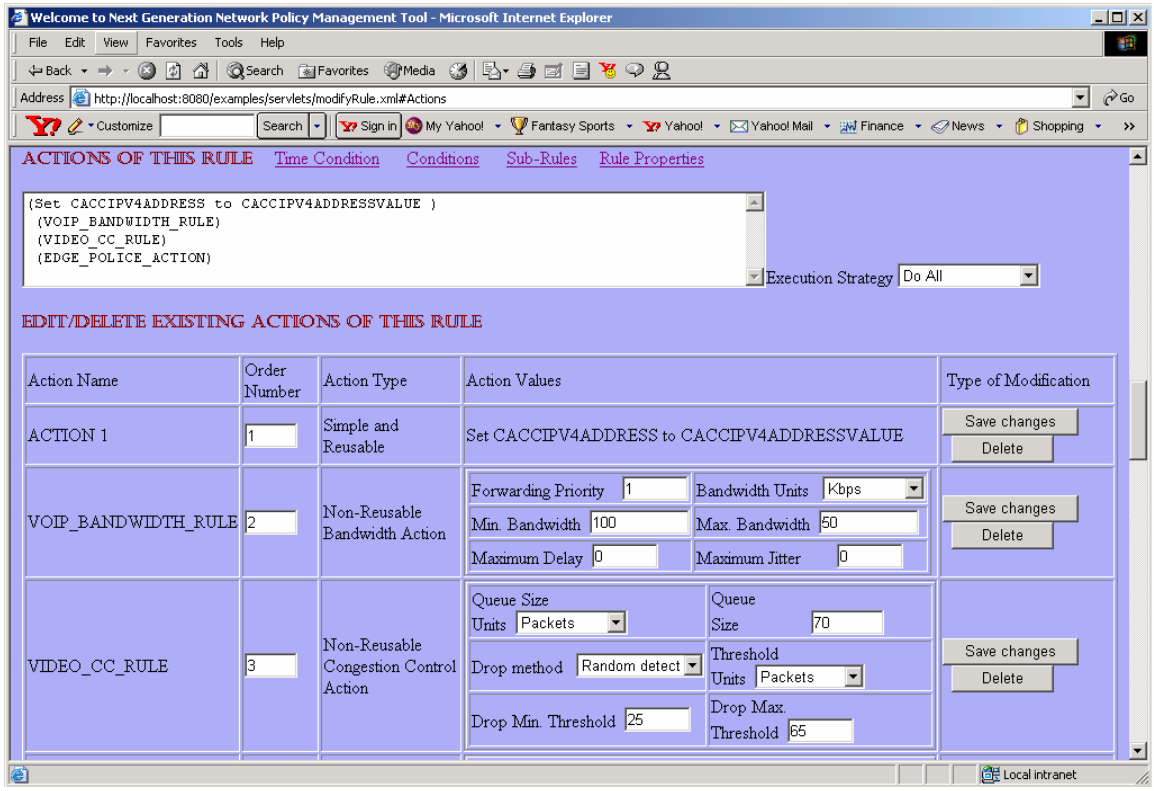


Figure 4.10: Modifying existing actions of a Policy rule using Policy Management Tool

4.2.3 Management of Network Elements using Directory-Enabled Networking

The Policy Management Tool provides the Network Administrator the ability to assign roles to network devices and their interfaces. The concept of Directory-Enabled Networking (DEN) [32] is used for storing information about network devices. The Distributed Management Task Force has defined an object-oriented information model that models the network elements and services in a managed environment.

4.2.3.1 Modeling Network Device and Its Interfaces using CIM [31] and DEN LDAP Schema [33]:

ComputerSystem:

The ComputerSystem class of Common Information model (CIM) is used to model a network device like router, switch etc. The property called "dedicated" identifies the type of computer system (Switch = 5, Router=4). The property called "nameFormat" is used to specify the format of the device name.

ServiceAccessPoint:

The router interfaces and switch ports are modeled as network services. ServiceAccessPoint is an abstract class that is used to represent that a service is made available to other entities for use. “ProtocolEndpoint” is a derived class of ServiceAccessPoint that is used to model router interfaces and switch ports.

IPProtocolEndPoint:

“IPProtocolEndPoint” class is a derived class of “ProtocolEndPoint” abstract class and is used to model interfaces or ports that support IP Protocol. An “IPProtocolEndPoint” object has address, subnet mask and IP version attributes. The “name” and “nameFormat” attributes are inherited from the ProtocolEndPoint base class. The name format in our current implementation is <interface Name>@<device Name>. The Appendix section provides the list of network interfaces modeled in the test bed.

LANEndPoint:

LANEndPoint class is a derived class of ProtocolEndPoint abstract class and is used to model ports of a switch. Local Area Network (LAN) ID, LAN type, MAC address, alias address, multicast address are the properties of this class. The “name” and “nameFormat” attributes are inherited from the

“ProtocolEndPoint” base class. The name format in our current implementation is <switch port Name>@<device Name>.

HostedAccessPoint:

HostedAccessPoint class represents the association between a “ServiceAccessPoint” object and the ”ComputerSystem” object. It is used to associate IPProtocolEndPoint object that represents an interface, with a ComputerSystem object representing a router. It is also used to associate LANEndPoint object that representing a port, with a ComputerSystem object representing a switch. The “HostedAccessPointDependentRef” attribute of this class contains name reference(s) to IPPProtocolEndPoint or LANEndPoint objects.

For example, let’s consider the modeling of a router named "egrc-router” that has 2 Ethernet (eth0, eth1) interfaces with IP addresses 152.1.213.110 and 152.1.212.110 and sub-net mask 255.255.255.0, using DEN LDAP schema.

The router is modeled as a “ComputerSystem” object with "dedicated" attribute set to 4 (value to represent a router) and its name attribute set to "egrc-router". Interface eth0 is modeled as “IPProtocolEndPoint” object with Address attribute set to "152.1.213.110", “SubnetMask” attribute set to "255.255.255.0" and name attribute set to "eth0@ egrc-router". Interface eth1 is modeled as “IPProtocolEndPoint” object with Address attribute set to "152.1.212.110", “SubnetMask” attribute set to "255.255.255.0" and name attribute set to "eth1@ egrc-router". The two IPProtocolEndPoint objects are associated with the router by creating a “HostedAccessPoint” auxiliary association object with HostedAccessPointDependentRef attribute set to "eth0@ router1”, “eth1@router2". This auxiliary object is then attached to ComputerSystem object that models "egrc-router" router (Figure 4.11).

4.2.3.2 Modeling Network Device Settings using DEN LDAP Schema and MgmtAccessSettings

The “**Setting”** class of the DEN specification represents configuration-related and operational parameters for one or more managed system element(s). Each “Setting” class has a setting ID. A managed system element may have multiple setting objects associated with it.

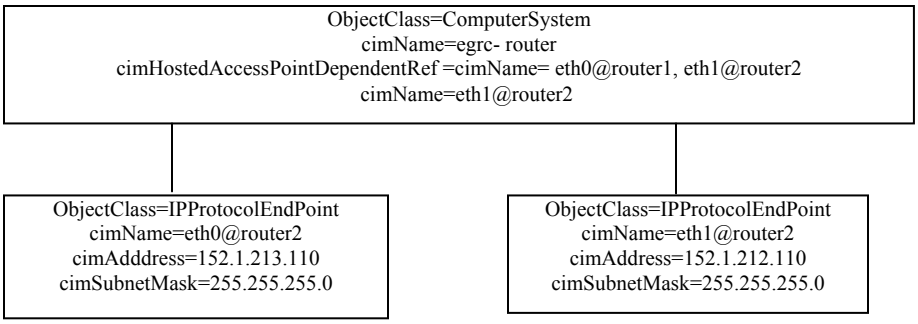


Figure 4.11: Modeling of a Network Device using DEN

The "**ElementSetting"** class represents the association between managed system elements and the setting class(s) defined for them. "ElementSetting" class is used to represent either association between N Setting objects and a managed element or to represent the association between a Setting object and N managed elements. It has two attributes namely "ElementSettingSettingRef" and "ElementSettingElementRef".

The attribute "ElementSettingSettingRef" contains DN reference(s) to "Setting" objects that apply to a specific managed element. This is used when "ElementSetting" class is used to represent association between N settings object and a Managed Element. The attribute "ElementSettingElementRef" contains DN reference(s) to managed network elements to which a "Setting" object apply. This is used when "ElementSetting" class is used to represent association between N managed elements and a Settings object (Figure 4.12).

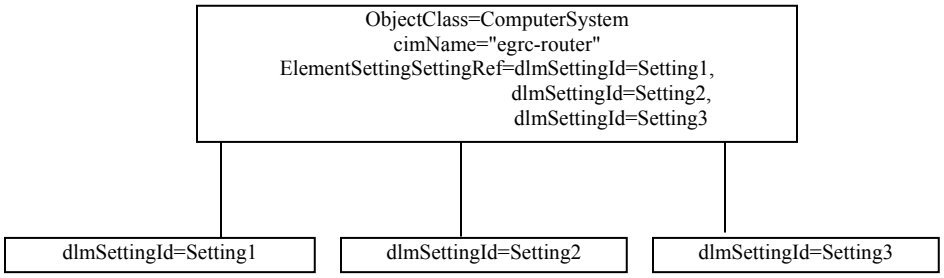


Figure 4.12: Modeling Network Device Settings

New MgmtAccessSettings Auxiliary Class:

A new class called "MgmtAccessSettingsAuxClass" was created to store login information of a network device to perform management/configuration through CLI/SNMP. This class is derived from Setting class. The attributes of this class are MgmtIPAddress, MgmtModelName, MgmtUserName, MgmtPassword, MgmtSNMPReadComString and MgmtSNMPReadWriteComString.

Managing Network Device Information:

The Policy Management Tool can also be used to add information about the network devices to which the policies are applied. A web page called "Add Network Device" is created for this purpose. Network device information obtained from the network administrator is stored as ”ComputerSystem” objects; interface or switch ports are modeled as “IPProtocolEndPoint” objects or as “LANEndpoint” objects. Associations between the interfaces/switch-ports with the network device are stored as described above. Information like telnet user name, telnet password, SNMP read and write strings for the network devices are stored in “MgmtAccessSettings” object and are associated with respective network elements as described above.

Roles can be assigned to network devices and to their interfaces/ports either at the time of adding the network device to the LDAP repository or at any time later (Figure 4.13). A role is stored in LDAP policy repository as “PCIMRoleCollection” object. Roles can be added or removed from the network device at any point in time.

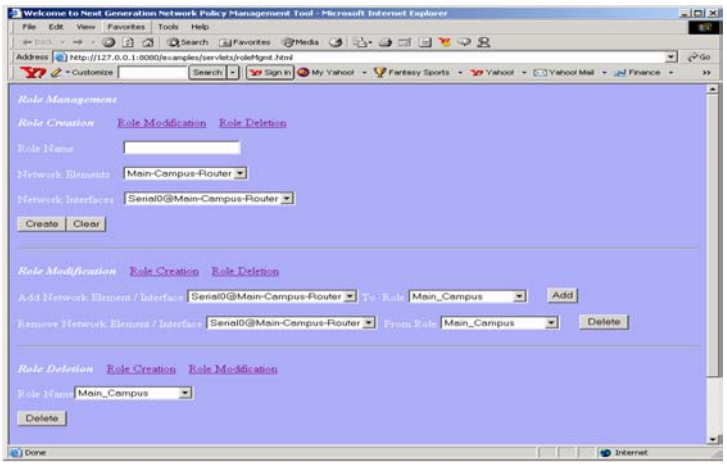


Figure 4.13: Role Management using Policy Management Tool

4.2.4 Role Based Policy Conflict Detection

The following sub-sections describe new algorithm(s) that are used to detect **intra-specific** policy conflicts. These algorithms do not detect semantic and inter-specific conflicts (Section 1.6.3). First a smaller problem is considered and then the solution to the smaller problem is applied to the larger problem.

4.2.4.1 Algorithm to detect conflict between simple policy rules

A simple policy rule is a rule with no sub rules.

Policy Rule De-correlation:

1) The condition of the policy rule is de-correlated into DNF form i.e. the conditions are reduced to a form like (a & b& c) | (a &e & f). After de-correlating, each part of the DNF form i.e. (a & b& c), (a & e & f) is referred to as a de-correlated rule. A policy rule has at least one de-correlated rule. The de-correlated rules are stored in an array. A rule evaluates to TRUE if one or more of its de-correlated rule evaluates to TRUE. For example lets consider the rule R with condition ((a == 1) & (b==1|c==1) & (d==1)). The rule's de-correlated rules are (a==1 & b==1 & d==1) and (a==1 & c==1 & d==1), where a, b, c, d are policy variables, "==" is the sign associated with policy variables a, b, c and d, 1 is the value associated with policy variables a, b, c and d.

2) In order to uniquely identify each de-correlated rule of all policy rules that are processed in conflict detection algorithm, the policy rule and its de-correlated rules are given a unique id. The de-correlated rule id is formed by <policy rule id>"."< Index of the de-correlated rule in its array>. For example if the de-correlated rule ID is 1.1, then policy rule id is 1 and this de-correlated rule is in index 1 of the array.

3) Each de-correlated rule will have one or more policy variables. Each policy variable will be associated with an operator sign and a policy value.

Policy Variable Tree Algorithm

Objective: Given a de-correlated rule R with policy variable V, find the set of all de-correlated rules having policy variable V whose values conflict with value of policy variable V in rule R.

A tree is maintained for policy variable V. Each node in the tree is associated with a positive value. The right sub tree is maintained in ascending order and the left sub tree is maintained in descending order. All policy variables with ">" or "==" sign are inserted in the right sub-tree. All policy variables with "<" sign are inserted in the left sub-tree.

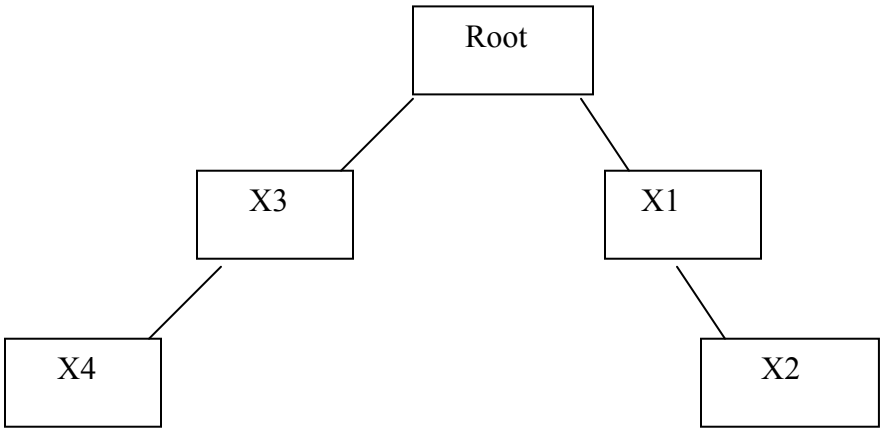


Figure 4.14 Policy Variable Tree

Each node in the right sub-tree rule has two rule lists namely equal rules and right rules. Let's consider the node with value X1 in the Figure 4.14. The equal rules of the node X1 consists of the ids of all the de-correlated rules that were inserted in this policy tree with the sign associated with policy variable being "=="and value equal to X1. Similarly, the right rules of X1 contains de-correlated rules that were inserted in this policy tree with the sign associated with policy variable being ">" and its value greater than X1 but less than X2. Each node in the left sub-tree rule has one rule list called left rules. Let's consider the node with value X4 in the Figure 4.14. The left rules of the node with value X4 consists of the ids of all the de-correlated rules that were inserted in this policy tree with the sign associated with policy variable being "<"and value equal to X4 and less than X3.

A new node is formed in the policy variable right sub-tree if and only if the value associated with the policy variable to be inserted is greater than X2 and the associated sign is either ">" or "==". Similarly, a new

node is formed in the policy variable left sub-tree if and only if the value associated with the policy variable to be inserted is less than X4 and the associated sign is "<". This reduces the size of the tree.

Lets assume that the policy variable tree is populated with de-correlated rules having policy variable V. Lets assume that a new de-correlated rule containing policy variable V with value equal to X5 is inserted in to the tree. The set of de-correlated rules that will be in conflict with X5 depends on the value X5 and the sign associated with policy variable and the sub-tree in which it will be inserted.

If a node is inserted in the left sub-tree, possible conflicts are

1) All rules in the left sub-tree (left rules of the nodes in the left sub-tree)
2) All rules in the right sub-tree of nodes with values less than X5 (both right and equal rules of each node in the right sub tree).

If a node is inserted in the right sub-tree and the sign associated with the policy variable is ">", possible conflicts are

1) All rules in the right sub-tree (right and equal-to rules)
2) All left rules in left sub-tree with value greater than X5

If a node is inserted in the right sub-tree and the sign associated with the policy variable is "==", possible conflicts are

1) All right rules (no equal-to rules) in the right sub-tree with values less than X5.
2) All left rules in left sub-tree with values greater than X5

By this algorithm, the set of possible conflicting de-correlated rules for a de-correlated rule with policy variable of value X5 and of a sign can be determined.

Algorithm to find conflict between simple rules (Policy Rule Conflict Detector)

Objective: Given a rule R, find the set of all rules that are in conflict with R.

1) For each de-correlated rule in the policy rule R,

For each policy variable,

1) A check is performed to determine whether a policy variable tree corresponding to this policy variable has been formed. If a policy variable tree is not formed, a new policy variable tree is formed and the policy variable is inserted in to the tree.

2) If a policy variable tree is found, the set of all possible conflicting de-correlated rules are determined using the policy variable tree algorithm. A bin (list) is maintained for each Policy Variable Tree that maintains the list of conflicting de-correlated rules found in a policy variable tree. The bin will contain the id of the de-correlated rule(s) that are in conflict with.

Now the set of rules to which the de-correlated rules in the bins belong to, are the possible conflicting rules. But we cannot be sure that they are conflicting now, we need to do some more processing to determine whether these rules are really conflicting.

Two rules are conflicting if they get evaluated at the same time and if they result in conflicting actions. The approach taken here is to find set of rules that have conditions that will be matched at the same time.

2) The next step in our detection process is to eliminate all rules in the bins that are non-time conflicting (i.e. they are not valid at the same time) by evaluating the time conditions of the policy rule to which the de-correlated rule belongs.

3) Now the rules in the bins occur at the same time. We need to now determine whether the conflicting rules have any other variables. For e.g. if R is (a = 50 & c=50) and a conflicting rule is (a = 50 & e=50),

then both these rules are not conflicting, but they will we in the bin, because "a=50" is present in both rules. In order to eliminate rules like this the following is performed

1) For each de-correlated rule in policy Rule R

   For each de-correlated rule in each bin B

   1) Let's say that the de-correlated rule in R as DR1 and de-correlated rule in bin B has DR2.

   2) It is first checked, whether DR1 is a subset of DR2 (i.e. all variables in DR1 are in DR2) or DR2 is a subset of DR1 (all variables in DR2 are in DR1).

   3) If none of them are subset of each other, then both rules are not conflicting with each other, hence the policy rule of the conflicting de-correlated rule is not added to the Final Conflict Rules List.

   4) If one de-correlated rule is subset of another, the policy variables in de-correlated rule that is a subset of the other, is compared with the policy variables in the other de-correlated rule.

   To determine whether the two de-correlated rules conflict, the value(s) of each policy variable in the subset rule, is compared to the value(s) of the corresponding policy variable in the other de-correlated rule.

   If they conflict, the **policy rule** id of the de-correlated rule is added to set of Final Conflict Rule List.

4) We still cannot be sure that the rules in the Final Conflict Rule list are conflicting because they will be in conflict, if and only if they have conflicting actions. In this step, the actions associated with policy rule R is

compared with actions of each policy rule in the final conflict rule list. If the actions are in conflict, then both rules are **in Conflict.** If not, both rules are not in conflict.

It is possible for all our bins to be empty in step 1or 2 and the Final conflict rule list to be empty, in which case there are no conflict rules and hence policy rule R is not in conflict.

5) If the all rules in Final Conflicting Rules list do not have action conflict, then Policy rule R1 is free from conflicts.

4.2.4.2 Algorithm to detect conflict among nested rules

This algorithm uses the algorithm described above (Section 4.2.4.1). There is an un-solved problem in both the previous algorithm and this algorithm. The problem is how to populate the Policy Variable tree. In the previous algorithm we assumed that it is already populated.

Suppose if we want to detect conflict for a Policy P1. Which policies must be de-correlated and be inserted in the policy variable tree?

The use of roles makes things easier. The set of conflicting role combinations of Policy P1 is found and all Policies that have been assigned to one or more of the conflicting role combinations are selected, de-correlated and inserted in the policy variable tree. (Populated).

Another interesting problem with nested rules is that the sub-rules of the nested rules are in conflict if and only if their parent rule is in conflict. Hence the policy P1 must first be compared with the parent rule of the nested rule and then with their components, this requires us to keep track which rule is a sub-rule of another, note a reusable rule can be a sub-rule of two or more nested rules.

A simplest solution to this is to combine the parent rule and the child rule in to a single rule using AND condition.

E.g.: If (a == 50) then A1

If (b==50) then A2

This can be converted to two rules (a==50), (a==50) AND (b==50)

Algorithm:

1) First determine the set of conflicting role combination
2) If the input policy rule is a nested rule convert to simple form (there will be N rules). These rules are called input rules.
3) Find policies that are assigned to one or more role combinations in the conflicting Role Combination Set
4) If any policy in step 3, is a nested rule convert to simple form.
5) Maintain a Policy Rule Conflict Detector for each role combination.
6) The policies in step 4 are used to populated policy variable trees in their respective role combinations.
7) Policy Rule Conflict Detector (s) corresponding to the input rule's conflict role combinations processes each de-correlated rule of the rule in Step 2. Note: Each rule may have different role combinations.
8) If all the rules in step 2 pass all conflict detection tests, the nested rule is free from conflict. If not it is in conflict.

4.2.4.3 Algorithm to detect Conflict between Policy Groups having simple and nested rules

The above-described algorithm (Section 4.2.4.2) is used for detecting conflicts across policy groups. The set of input rules in Step 2 of the above algorithm (for simple and nested rules) will be as follows,

1) If a component is a simple rule, it will be added directly to the set of input rule
2) If a component is a nested rule, the parent rule and the sub-rules will be combined to form N rules (refer section 4.2.4.2) and these N rules are added to the set of input rule.

4.2.5 Web based Interface

The web-based interface was developed using Java Servlets and Java scripts. The Policy Management Servlet processes requests from web-clients i.e. from network administrators and policy writers.

The policy browser is implemented using Netscape's JavaScript Tree objects. The servlet accesses policy information from LDAP, translates the policy information in to Java script objects, which is then embedded in Hypertext mark-up language (HTML) files. The Html stream is then sent to the web-client.

The Policy management tool can also provide detailed view of a policy. On receiving a "View" request, it retrieves all the conditions and actions of the policy. Sub-rules of a nested rule or a policy group are also retrieved. The tool displays the policies in the order of their evaluations i.e. high priority policies will be at the top of the policy tree. Actions are showed in the order of execution.

A new XML schema was created to represent policies in a structured and hierarchical manner. The policy management tool converts the policy information obtained from LDAP in to a XML file. The XML stream is then sent to the web-client. Extended Style sheets (XSL) are used to display the policy data in XML form. Using XML and XSL offers one main advantage, the policy retrieval and translator components of the policy management tool need not be modified when the user interface of the policy management tool needs to be changed.

The policy management tool can be accessed from any computer that has Internet Browser software with support for Java-scripts, XML and XSL. The major browser software providers namely Microsoft and Netscape provide support for all three languages.

Security Considerations:

Although a web-based interface for the Policy Management Tool provides a number of advantages, it makes the policy management tool less secure. The policy management tool is susceptible to security attacks like Denial of Service and Eavesdropping.

Typically, Policy Management Systems are deployed in large networks in which the network is divided in to number of domains. For example, in our university we can consider the network in Main campus, Centennial, Vet school as separate domains. Each domain may have different policies and different administrator(s). Different levels of privileges may be assigned to network administrator(s). For example, the senior administrator will have access to all network policies and other administrator's will have access only to the network that they manage. Hence the policy management tool must be able to identify the type of user, and more importantly it needs to be able authenticate the user to prevent intended or unintended modification of policies by administrators of other domains.

In order to satisfy the requirement of authentication and encryption of data, the Policy Management Tool is deployed using Apache Tomcat web server Version 4.1. The Apache Tomcat server is a reference implementation of Java Servlets 2.3 specification. It provides support for encryption of data using Secured Sockets Layer (SSL). HTTP Basic authentication is used to authenticate end-users of the policy management tool.

It is important to note that we need authentication both at the Policy Management tool and Policy LDAP Repository. A network administrator can be provided access to only specific parts of the LDAP directory. This will prevent the modification of policies by administrators belonging to other domains. OpenLDAP implementation provides Access controls for this purpose. This leads to having the same user database at two locations (one in LDAP directory and the other in Apache web server).

By having only one user database in the LDAP Policy Repository and making the Web server communicate with LDAP server whenever it needs to authenticate a user, we can avoid this redundancy of data. This will not add to performance degradation, because LDAP Servers provide high performance for retrieval of data. The Apache web server has in-built software (JNDI-Realm) to support this functionality. The web server also allows us to assign different roles to the user. In addition to having a valid user name and password, the user must belong to the specified role in order to access information. The role information is also stored in the LDAP repository.

We have used this functionality, to assign different roles to administrator. A role is created for each network domain and is assigned to administrators who manage the domain. A senior administrator can be assigned to more than one role, so that he/she can access multiple network domain information.

4.3 Policy Server

This section describes the components of Policy Server in detail. The Policy Server was developed using Java Software Development Kit version 1.3. The Configuration Server is implemented using Expect (a tool for automating interactive applications like telnet, ftp, rlogin etc.). Expect libraries in C are used in Configuration Server implementation.

4.3.1 Event Processor/Handler(s)

The Policy Server is developed using an Event-Based Software Architecture. There are two types of events namely internal and external events. Events that are generated by the Policy Server like timer events are called internal events. Events that are generated by external resources like network monitors, network administrators are called external events. The event generation and processing of events by Policy server is explained in the following sub-sections.

4.3.1.1 Events from Web Clients

These events correspond to creation, modification, deletion, enabling and disabling of policies by the Network administrator. The Policy Management Tool (Policy Servlet) keeps track of the user actions and sends appropriate system events to the Web Handler component of the Policy server.

The Policy Management Tool generates events

- When a user enables a policy
- When a user disables an enabled policy
- When a user modifies an enabled policy
- When a user deletes an enabled policy

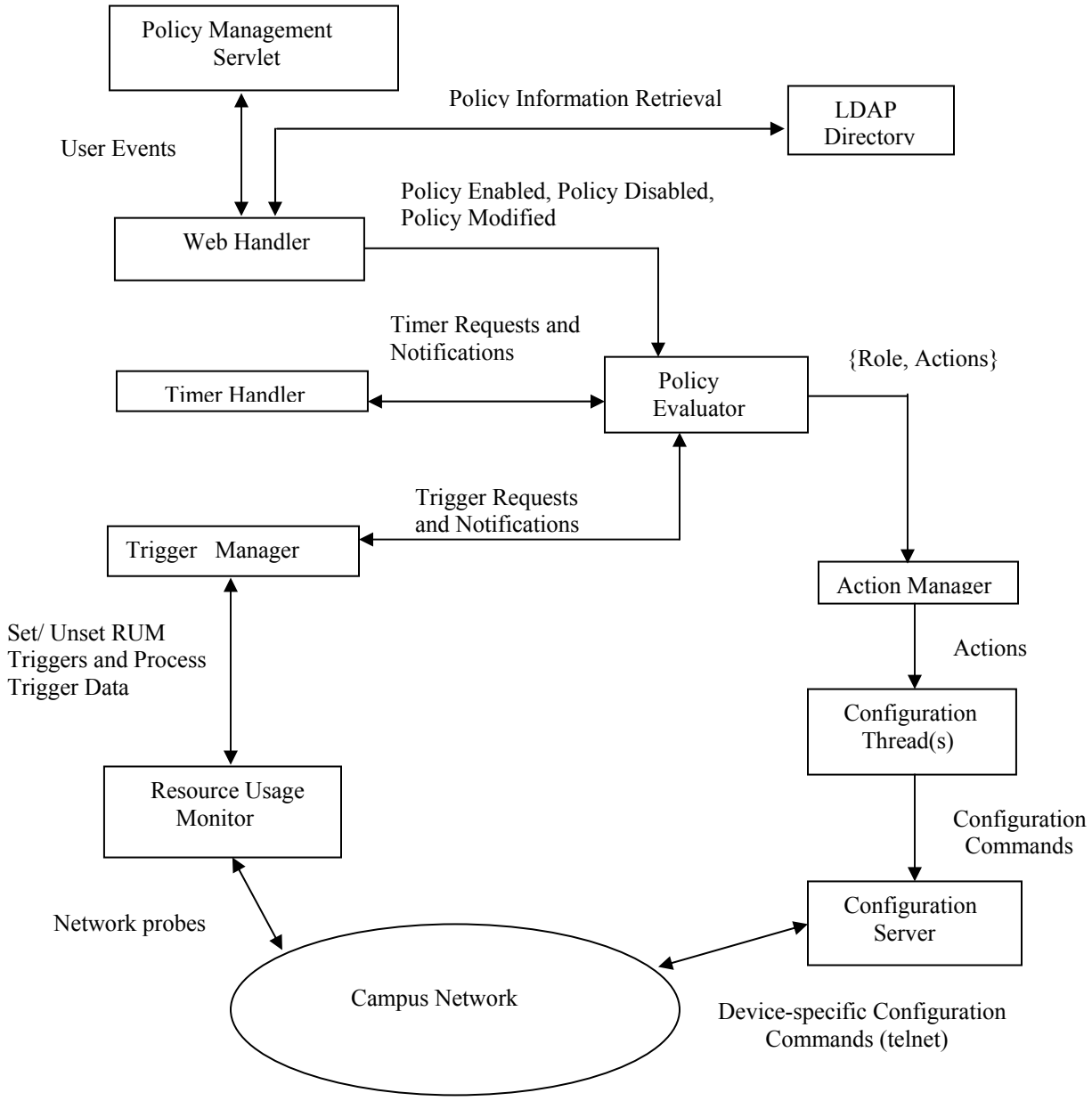


Figure 4.15 Policy Server and its Components

A policy can be a policy group, a policy rule or a nested policy rule. A policy group contains a group of interrelated policy rules or policy groups. A nested rule consists of a parent rule and sub-rules. One or more sub-rules in a nested rule/policy group and one or more policy groups in a policy group are referred as "components" in the following sections.

System Event Messages:

Add / Enable event message corresponds to the Addition / Enabling of a policy.

Re-evaluate event message corresponds to the modification of an enabled policy

Delete event message corresponds to the Deletion / Disabling of an enabled policy

Web Handler:

The Web Handler processes the events generated by the Policy Management tool (PMT). The Web Handler is a thread that listens on a configured port. The policy management tool sends the user events to web handler's configured port.

Policy Information is stored as Policy Objects in a table called Global Policy table. This table is implemented as a Hash Table with {Policy name, Policy object} as the {key, value} for each entry in the table.

Storing information of a policy i.e. it's conditions, its actions, its validity periods takes large amount of memory. The amount of memory consumption is reduced by having only one object for a policy irrespective of whether it is a component of a policy or N policies or is not a component of any other policies. This will be very useful when a reusable policy is used as sub-component in two or more policies. It is for this purpose a global policy table is maintained.

The Web Handler is responsible for updating the global Policy table. When a new policy is enabled, a policy object is formed by retrieving policy information from LDAP Policy repository and is added to global policy table. The policy information includes information about the enabled policy and its enabled components. (A component of a policy may or may not be enabled when the policy is enabled).

The web handler also performs a Policy Expiration check. The Policy Expiration check is a procedure by which the Web Handler identifies whether the time at which the policy is enabled (i.e. current time) is after the end time of overall validity range of the enabled policy (Time Conditions: Section 3.4.3). If the enabled

policy has expired, the policy object is not added to the Global policy table and an error message is sent to the Policy Management tool.

The policy information is retrieved from the LDAP repository in a hierarchical sequence. When an expired policy is processed, its components (if any) are not retrieved from the policy repository irrespective of whether the components are valid or invalid. The Web Handler also keeps track of the policy names that were retrieved. If a reusable policy is part of two or more policies, information about the reusable policy is retrieved only once. This reduces the number of LDAP search queries, thereby improving the performance of the Policy Server.

After updating the policy table, the web-handler sends a Policy Evaluation Message to the Policy Evaluator. When a new policy is enabled it sends "Add/Enable" message to Policy evaluator. When an already enabled policy is modified it sends a "Re-Evaluate" message to the Policy evaluator.

When the web-handler receives a policy delete event from Policy Management Tool, it sends a "Delete" message to the policy evaluator. In this case, the web handler does not make any changes to the Global Policy Table. After processing the delete event, the Policy Evaluator removes the policy object corresponding to the deleted policy from the Global Policy table.

This component serves one of our Goals of Policy Server Design (**Quick Response Time**) in the following ways:

1) The process of policy information retrieval from LDAP policy directory is isolated from Policy Evaluation. As a result, the Policy Evaluator will not be affected by the delays (if any) caused by the LDAP directory server during the directory searches performed in the information retrieval process. The Policy Evaluator can handle other events like timer events and other external events associated with other policies without waiting for the policy information of the enabled or modified policy, from the LDAP Directory server.

2) By remembering the names of the policies retrieved, it reduces the number of searches performed on a LDAP directory, thereby reducing the delay associated with information retrieval process when reusable policies are used.
3) Reduces amount of memory consumption by maintaining a Global Policy Table.

4.3.1.2 Timer Handler

A policy may have time conditions associated with it. These conditions specify the time periods in which the policy is valid (Time Conditions: Section 3.4.3). A time condition specifies the overall validity range of a policy and other specific information like the valid months, valid days and valid times in a day. A policy may or may not be valid through out the entire time period specified in the overall validity range. The network administrator can specify specific valid time periods within the overall validity range.

This results in the following three states for a time based policy

- A policy is valid and active

  When the current time is within the overall validity range and within the specified specific valid time periods within the overall range.

- A policy is valid but not active

  When the current time is within the overall validity range but not within the specified specific valid time periods within the overall range.

- A policy is invalid

  There are three cases associated with this state,

  1) A network administrator enables a policy before the start time of the overall validity range. Such policies will be valid in the future but are invalid now. For example, a network administrator in a university can enable a QoS policy for a spring semester graduate class in December. Though the policy is not valid in December, it will become valid in January.

2) A network administrator commits a mistake by enabling a policy after the end time of the overall validity range, in this case the policy has expired and the Web-Handler component will detect this by the Policy Expiration Check and sends the error message to the policy management tool.
3) A policy becomes invalid after being enabled i.e. when the current time is not within the overall validity range.

The timer handler component is developed to handle all of the above states of a policy except the second case of the invalid state (handled by web-handler). The timer events are stored in a linked list and ordered by the time at which they occur. Ordering is maintained using relative times rather than absolute times. Such a list is called Delta list and was first developed by Comer. The timer handler performs insertion and deletion of timer events in the delta list. This list is referred as timer events list in this section.

Types of Timer Events

There are two types of timer events. The start timer event corresponds to the start of an active time period of a policy. The stop timer events correspond to the end of an active time period of a policy.

Subscribe/Notify Model

The timer handler is designed using a subscribe/notify model. The policy evaluator subscribes to the timer handler and timer handler notifies/sends timer events to the policy evaluator.

When a policy is enabled, the policy evaluator subscribes for timer events notification of the policy and its sub-components. On receiving the subscribe request, the Timer Event Handler determines the time state of the policy and inserts appropriate event in the timer event list in the following manner,

- If the state of the policy is “valid and active”, insert a stop timer event.
- If the state of the policy is “valid and inactive”, insert a start timer event
- If the state of the policy is “invalid, but will be valid in the future”, insert a start timer event.

When a timer event has occurred, the timer handler determines the type and time of next timer event to be inserted in the timer list. If a stop timer event has occurred, the next active period is determined and a start timer event is inserted. If a start timer event has occurred, the stop time of this active period is determined and a stop timer event is inserted.

Before inserting a new start timer event into the timer events list, a validity check is performed. This is used to verify that the time at which the new timer event will occur is before the end time of the overall validity range. It is possible for a policy's active time period to end before the overall validity range and the start of the next active period to occur after the end time of the overall validity range. The timer event is inserted only if the validity check is passed.

For example, lets consider a policy with an overall validity range 01-Jan-2002 6:00 AM to 31-Dec-2002 10:00 PM and is valid on all days from 8:00 AM to 5:00 PM. On 31-Dec-2002 a stop timer event will be generated at 5:00 PM, but the next active period which starts on 1-Jan-2003 8:00 AM is invalid because the overall validity range ends on 31-Dec-2002 10:00 PM.

Similarly, the validity check is also performed when inserting a stop timer event. If the next stop timer event occurs after the end of the overall validity range, the time at which this event occurs is modified to the end time of the overall validity range and is inserted into the timer events list.

Subscription to timer events by policy evaluator is implemented using serialized system messages. Each subscription request message has the name of the policy, an object containing the valid time information of the policy, the name of the "top-level" policy to which this policy belongs to and the priority of the "top-level" policy are passed to timer handler. A "top-level" policy is a policy that is not a component of any other policy.

Policy Evaluation Requirements

1) A policy group, or an active nested policy rule with one or more policy rules (components) having

different time conditions, needs to be re-evaluated whenever a component becomes active or inactive.

2) When a policy rule is a component of two or more top-level policies, these top-level policies must be evaluated in the order of their priority, when the component policy rule is enabled or disabled.

3) A component policy is active if

   a) The policy that contains the component policy is active AND

   b) The component policy itself is active according to its time conditions.

Implementation

In order to satisfy the Policy Evaluation requirements, Policy Evaluator subscribes for timer events as described below. On receipt of timer events, it performs appropriate policy evaluations (Section 4.3.2). When an enable event corresponds to a "top-level" policy, the policy evaluator traverses the "top-level" policy and subscribes timer event notification for all its enabled components.

1) If the top-level policy is a simple policy rule,

   a) Policy Evaluator subscribes for timer event notifications of the top-level policy

2) If the top-level policy is a nested rule,

   a) Policy Evaluator subscribes for timer event notifications of the parent rule in the top- level policy.

   b) Policy Evaluator subscribes for timer event notifications for the enabled sub-rules if and only if the parent rule is active.

3) If the top-level policy is a policy group,

   a) The Policy Evaluator subscribes for timer event notifications of its enabled components.

   b) If the component is a simple rule, Policy Evaluator subscribes for timer event notifications of the component.

   c) If the component is a nested rule, Policy Evaluator subscribes for timer event notifications of the parent rule of the nested rule. Policy Evaluator subscribes for timer event notifications for the enabled sub-rules of this component if and only if the parent rule is active.

Maintaining timer events for a policy that is reused in two or more policies

A reusable policy rule can be a component of two or more nested policy rules or policy groups. In such cases, the Timer Handler receives more than one subscription requests for the reusable policy rule from the Policy Evaluator.

Instead of inserting a timer event in the timer events list for each such request, Timer Handler maintains only one timer event in the timer events list for a policy irrespective of the number of requests associated with the policy. The timer handler maintains two tables namely Timer Event Subscription Table and Timer Events Subscriber Table for this purpose (Table 4.1, 4.2).

Each entry in the subscription table refers to the name of the policy that is handled by the timer handler and the number of timer event subscription requests that were received for this policy. Each "top-level" policy is considered as a Timer events subscriber. Each entry in the Timer Events Subscriber table consists of the name of the subscriber (top-level policy), its priority and the list of handled policies that are associated with the subscriber. This table is maintained in descending priority order i.e. subscribers with top priority will be at the top of the list (to simplify order of evaluation).

Table 4.1: Timer Event Subscription Table

Policy Name	Number of subscriptions

Table 4.2: Timer Events Subscriber Table

Top-level Policy Group Name (Subscriber)	Priority	Name of the component policies.

On receiving the subscribe request, the Timer Handler first checks if the policy for which the timer event notification is requested, is currently handled by performing a lookup in Timer Event subscription table in Table 4.1

- If the timer handler does not handle the policy, a corresponding start/stop timer event is inserted in the timer events list and a new entry is added to subscription table.
- If the timer handler is currently managing the timer events associated with this policy, a corresponding start/stop timer event is not inserted in the timer events list and the "Number of subscriptions" in the subscription table (Table 4.1) is incremented.

The timer handler performs a lookup in the Timer Events Subscriber table (Table 4.2) and updates the sub-component policies list. A new entry is created for new subscribers.

Sending timer event notification to Policy Evaluator

For each timer event that has occurred, the name of the policy to which the timer event is associated and the type of timer event are collected in a list. This list is the set of timer event notifications to be sent.

The subscriber(s) to which timer events are to be notified are determined by traversing the subscriber table (Table 4.2) and checking if the subscriber has requested for notification of one or more timer events in the timer event notifications set.

All timer notifications associated with a subscriber are aggregated in a single "Re-evaluate" message and the message is sent to policy evaluator with the name of the subscriber (the name of the top- level policy group). The purpose of this aggregation is to reduce the number of re-evaluations performed by the policy evaluator, thereby reducing the number of configuration updates. The policy evaluator reevaluates the top-level policy tree identified by its name in the "Re-evaluate" message.

Timer events are sent to policy evaluator in the order of priority of subscribers i.e. a high priority top-level policy will be notified first and evaluated. This will ensure that the top-level policies are evaluated in order of their priority thereby satisfying policy evaluation requirement #2.

Un-subscribing Timer Event Notification from Timer Handler

When the Policy Evaluator receives a "Delete" Message from the Web handler, it un-subscribes timer event notifications associated with the deleted policy and its components. (Policy Evaluator: Section 4.3.2).

This component serves one of our Goals of Policy Server Design (**Minimal Configuration Updates**) by,

- Aggregation of timer events corresponding to policies that are components of a "top-level" policy. By this way, the number of re-evaluations of the top-level policy is reduced, thereby minimizing the number of configuration updates at the network devices.

4.3.1.3 Trigger Manager

The Policy server handles dynamic policies by setting appropriate triggers in external sources such as network monitors. Dynamic policies consist of policy variables such as bandwidth utilization of a host and number of flows per host. The Policy Evaluator subscribes for trigger notifications by using the Trigger Manager. The trigger notification request consists of policy variables and values of the policy that represent the threshold levels of the triggers and role(s) to which the policy applies.

The Trigger manager is a component of the Policy server that manages the setting/un-setting of external triggers and processing of trigger notifications from network monitors. The trigger manager communicates with a Distributed Resource Monitoring Tool called Resource Usage Monitor [43] (RUM) using its web-based interface.

It is designed to minimize the number of external triggers that are set. For example, lets assume that two polices A and B require a trigger to be generated when the bandwidth utilization of a host 152.1.2.2 is greater than 6 Mbps. Instead of setting two triggers, the trigger manager sets only one trigger. This reduces

the amount of processing in the network monitor. The subscribe/notify model described in Section 4.3.1.2 is used for this purpose. Trigger Manager maintains a Trigger subscription table (Table 4.3) and Trigger Subscriber table (Table 4.4) in regard. The entries in the trigger subscription table contain {Trigger name, Trigger object} as {key, value} pairs. The subscriber name (name of the top-level policy), its priority and its associated triggers are maintained in the subscription table.

For each trigger request, a unique report id and trigger id is generated. The policy variables and values (threshold levels) in the trigger request from the policy evaluator are translated to RUM parameters. Trigger Manager checks whether a source IP address or a destination IP address policy variable is present in the trigger request. If so, it forms appropriate RUM parameters that specify RUM to apply threshold levels only to the specified source or destination host. If no such variables are defined, it automatically translates the roles to which this policy applies in to a network/VLAN/subnet name(s) and sets the trigger for each for all network/VLAN/subnet to which this policy applies. The role(s) information is also stored in the Trigger object for future reference.

The information regarding the threshold levels and the network/VLAN/subnet or IP address of a host to be monitored are stored in RUM by sending a "save report" request (HTTP GET) to RUM. The threshold levels are then associated with a trigger by sending a "make trigger" request (HTTP GET) to RUM. The "make trigger" request consists of the id of the report that specifies the threshold levels associated with the sources to be monitored, the IP address and port number to which the trigger notifications are to be sent.

Trigger manager includes a thread that waits for trigger notifications from RUM. RUM sends the names of matching triggers in each sampling interval, in a UDP Packet. On receiving the UDP packet, the trigger manager traverses its subscriber table. It identifies the triggers that have been generated for each subscriber and retrieves that data corresponding to the trigger from RUM. The trigger data consists of information about host(s) or flow(s) that matched the specified threshold levels of a trigger request. The data is securely retrieved from RUM using HTTPS.

Trigger Manager also performs the translation of RUM variables into policy variables. The role(s) (network/ VLAN /subnet name) to which this trigger data is associated is obtained from the trigger object stored in the subscription table and added to trigger data. This will enable the policy evaluator to evaluate components of the top-level policy that apply only to this role.

Data of all matching triggers that are associated with a subscriber are translated and aggregated together and sent in the message body of a single "Re-evaluate" message to the policy evaluator. This aggregation of data reduces the number of re-evaluate messages that are sent to the Policy Evaluator. Also sending a single re-evaluate message reduces the number of configuration updates sent to the network devices (Policy Evaluator: Section 4.3.2).

Triggers are sent to policy evaluator in the order of priority of subscribers i.e. a high priority top-level policy will be notified first and evaluated. This will ensure that the top-level policies are evaluated in order of their priority.

When a trigger has no associated subscribers, trigger manager automatically sends the "remove trigger" (HTTP GET) request to unset the trigger and "remove report" to delete the associated report in the Resource Usage Monitor.

Table 4.3: Trigger Subscription Table

Trigger Name	Trigger Object

Table 4.4: Trigger Subscriber Table

Top Policy Group Name (Subscriber)	Priority	Name of the trigger names associated with this subscriber

This component serves our Goals of Policy Server Design (**Minimal Configuration Update and Minimization of Processing at Network Monitors, Security**) in the following way:

- By aggregating the trigger information in a single policy "Re-evaluate" message, the number of number of configuration updates made to network devices in response to the N triggers is only one instead of N.
- By setting only one trigger instead of N triggers when the same trigger is required by different policies, the policy server minimizes the processing at the external sources like network monitors such as RUM.
- By retrieving network traffic information from RUM using SSL and HTTP Basic Authentication, network data is protected from eavesdropping etc.

4.3.2 Policy Evaluator

The Policy Evaluator is the main component in the policy server. It processes System messages sent by the Web-Handler, Timer handler and Trigger manager. These messages are sent to the policy evaluator using UDP sockets. The policy evaluator is a thread that waits for system messages at a pre-configured port.

There are three types of System Event Messages:

1) Policy Enabled Message
2) Policy Re-Evaluate Message
3) Policy Disabled Message.

Policy Enabled Message:

Web handler sends Policy Enabled Message when the user of the Policy Management Tool enables a policy. The message has a name field that identifies the name of the policy being enabled.

Policy Re-Evaluate Message:

This message is sent by,

- Web-handler when a user modifies an enabled policy.
- Timer event Handler when a policy has become active or inactive.
- Trigger manager when a trigger is notified by RUM

The message has a name field that identifies the name of the policy to which the timer events are associated.

Policy Disabled Message:

When the user of the Policy management Tool disables an enabled policy, Web handler sends Policy Disabled Message to policy evaluator. The message has a name field that identifies the name of the policy being disabled.

Data Structure for Policies:

Each policy is represented using a General Tree data structure.

1) A simple policy rule is a tree with a root node and no child nodes.

2) A nested policy rule is a tree with a root node and one or more child nodes. Each child node can either be a simple policy rule or a nested policy rule.

3) A policy group is a tree with a root node and one or more child nodes. Each child node can either be a simple policy rule or a nested policy rule or a policy group with the condition that “all child nodes must either be of a policy rule or a policy group but not both”.

A Policy node will be referred as Policy Tree Node (PTNode) in this section. The root of the policy tree has two tables, PolicyObjectReference table and RoleCombinations set.

Each entry in the PolicyObjectReference Table has {"policy name", object references list} as {key, value} pair. The object references contain references to PTNodes of a policy with name "policy name". This list will contain more than one reference when a policy is reused more than once in another policy (i.e. a policy tree has two or more instances of a reusable policy). This is used to quickly identify the location of a policy component in the policy tree.

The role-combinations set contains the unique set of role combinations to which the policies in a policy tree apply. The evaluator uses this set to determine whether a policy tree needs to be re-evaluated when another policy is modified, enabled or disabled.

A policy tree is re-evaluated due to the modification, addition or deletion of another policy if and only if one or more of the roles of the policy being modified, added or deleted belongs to the set of role-combinations of the policy tree i.e. if the policies of the policy tree and the added/deleted/modified policy applies to one or more common roles. The objective is to re-evaluate all policies that might have been affected by the modification, deletion of an existing policy or addition of a new policy.

Policy Evaluation:

Evaluation of Trigger-Based Policies:

The evaluation of trigger based policies that have policy variables like bandwidth utilization of host, number of flows per host is performed by comparing the trigger data with the specified rule.

For example, let's consider the rule

if (bandwidth > 6 Mbps && No. Of Flows > 75) Action 1

This rule will be evaluated when a trigger notification message containing one or more trigger data record (Host information) is received. Each data record in the trigger data will have the bandwidth and number of flows information. The evaluator compares the value of these variables in the data record and the rule to determine whether the rule matched.

Evaluation of Non-Trigger Based Policies:

The evaluation of non-trigger based policies is different from the above. For a non-trigger based policy evaluation, the policy evaluator has no data record that it can use to determine whether the rule matched. These policies are actually evaluated by network devices. The policy server translates the valid non-trigger based policies into network device specific command and configures the network devices depending on the role to which the policy applies. In this case, the administrator can specify only decision strategies that are supported by network devices. A number of network devices support only First Matching decision strategy.

For example, let's consider the rule

If (IPv4Address == 152.1.2.2) Permit_Access_Action

In this example, the policy server translates the rule into an access-list configuration command and configures the network devices that belong to the specified role of this rule.

Steps in evaluating a policy:

The evaluation method is passed the following arguments,

1) Policy tree to be evaluated
2) A set of role combinations to be matched
3) Trigger data record (one flow or host information)

The trigger data set will be empty when evaluating a Non-trigger based policy. The set of role combinations specify what policies are to be evaluated. It is used to restrict the policies that will be evaluated. For example when a policy A with role R1 is modified, it is enough that we re-evaluate only policies that have Role R1, in which case "R1" will be passed as the set of role combinations to be matched.

1) Role Combination Check:

   This is used to determine whether the role(s) or role combinations of the policy to be evaluated belongs to the set of role combinations to be matched.

2) Time Condition Check:

   If the policy is a simple rule or nested rule, the policy is checked to determine whether the policy is active. If a policy is not active, the policy and its components are not evaluated.

3) Trigger-based Policy Evaluation:

   If the policy is a policy rule and has parameters like bandwidth utilization of a host, number of flows in a host, the trigger data record is used to evaluate the policy rule. Each policy rule may have one or more de-correlated rules (Section 4.2.4.1). Each de-correlated rule is checked for a match. If a match occurs, the rule is matched. The trigger data set and actions (action tree) of this rule are added to the list of actions to be sent to Action Manager.

4) Non-Trigger based Policy Evaluation

   The policy variables, their values and actions (action tree) of the rule are added to the list of actions to be sent to Action Manager.

5) If the policy is a policy group or a valid nested rule

   Evaluate its components (Go to Step 1)

   a) If the evaluation of the component is successful (i.e component rule matched),

      (i) If the component is a non-trigger based rule, continue evaluation of components irrespective of the decision strategy,

      (ii) If the component is a non-trigger based rule

      Check the decision strategy of parent policy, if the strategy is First-matching,

      Stop evaluation of components; continue evaluation of components for an All-Matching strategy.

6) The list of actions to be executed and corresponding policy roles are sent as a Java serialized object to Action Manager.

Handling an Enable event from web-handler:

The buildPolicyTree () method is called when a new policy is enabled. The information about the new policy is obtained from Global Policy table and a new node called PTNode (Policy tree node) is formed. The components of the new policy (if it is a policy group or a nested policy rule) are inserted in the tree as child PTNode nodes. The child nodes are ordered from left to right in the order of priority. By maintaining this ordering, policy evaluator can evaluate a policy tree in the order of priority just by performing a traversal starting at the root of the policy tree and traversing the children (if any) from left to right.

The policy evaluator determines whether the new enabled policy is a component of an already enabled policy by performing a LDAP search. If the policy is a top-level policy i.e. it is not a component of any other policies, the buildPolicyTree () method is used to build the policy tree and to set appropriate timer events and triggers. The policy tree is then added to policy tree table.

The policy evaluator keeps track of the roles to which the policies in this new policy tree apply. The policy evaluator identifies and evaluates all policy trees that have one or more roles in common to the newly added policy tree.

The buildPolicyTree () performs the following,

1) Inheritance of Policy Roles from Parent policy

   If the policy is a component of another policy, the roles of the parent policy are implicitly inherited by the component.

2) If the policy is a policy rule (either a simple or nested rule), the policy rule's time conditions are evaluated, and a flag "isActive" is set, if time condition is true.

   a) If there is no parent policy rule for this policy, the timer events notifications Subscription request is sent to the Timer Handler. If the rule has real time policy variables like bandwidth utilization of a host, number of flows of a host, a trigger notification request is sent to the trigger manager.

a) If there is a parent policy rule, the timer events and triggers subscription requests are sent if and only if the parent policy is valid

3) A PTNode object containing the policy information is formed and attached to its parent (if any).
4) The policy name and the PTNode are added to PolicyObjectReference table and the role combinations (after inheritance) of this policy are added to RoleCombinations table.
5) If the policy has components, buildPolicyTree () method is called recursively to process the components of the policy

If the new policy is not a top-level policy, but is a component of one or more enabled policies, the policy tree(s) containing the parent rule(s) of the new policy are modified.

Handling a Modify Event from Web Handler:

When a top-level policy is modified, all the timer events and trigger subscription requests associated with the root of the policy tree and its children are un-subscribed and the existing policy tree of this top-level policy is completely deleted. The policy evaluator keeps track of the roles to which the policies in this deleted policy tree applied. The policy tree is then rebuilt using buildPolicyTree (). The policy evaluator keeps track of the roles to which the policies in this new policy tree apply. The policy evaluator then identifies and evaluates all policy trees that have one or more roles in common to the newly added policy tree or in the deleted policy tree.

If the policy is not a top-level policy, the policy trees to which the modified policy is a component are identified. The object reference list of the modified policy is obtained from PolicyObjectReference table. Each element in the object reference list will refer to a sub tree of a tree. This sub tree is processed in the same way as that of modifying a root of the policy tree as described above.

Handling a Delete Event from Web Handler:

The procedure is same as that of handling a modify event, except that new policy trees are not built. The policy evaluator keeps track of the roles to which the policies in this deleted policy tree apply. The policy

evaluator identifies and re-evaluates all policy trees that have one or more roles in common to the roles in the deleted policy tree.

Handling Timer events from Timer Handler:

The timer handler sends one or more timer events in a "Policy Re-evaluate" message. The message contains the name of the top-level policy to which the timer-events are associated (this means that either the top-level policy or its components have been enabled/disabled). The policy evaluator uses the policy name in the "Re-evaluate" message to select the policy tree to be updated. A function called updatePolicyTree () is called. This method sets/unsets the flag "isActive" of the policy object according to the timer-events received.

If a parent policy rule has become inactive, all the triggers associated with the parent policy rule are un-subscribed. All the timer events and triggers associated with its components are un-subscribed.

If a parent policy rule has become active, all the triggers associated with the parent policy rule are subscribed. The timer events request for all its components are subscribed. If the component is active, the trigger subscription is sent to trigger manager.

The policy evaluator keeps track of roles of the policies that were activated or deactivated by the timer events. The policy evaluator identifies and re-evaluates all policy trees that have one or more roles in common to the roles of the policies that were activated or deactivated

Handling of Trigger Notifications from Timer Handler:

Re-evaluate message from Trigger Manager contains trigger data (flow or host specific information like source IP address, destination IP address, source port, destination port, bandwidth utilization, number of flows) with the name of the role to which the trigger data belongs.

The policy tree to which the trigger is associated is evaluated N + 1 times for a trigger notification with N trigger data record. Each evaluation is performed with a trigger data record. The N+1th iteration is performed to evaluate default policies. For example, let's consider a policy group with the First Matching decision strategy

if (bandwidth > 6 Mbps && No. of Flows > 75) Action 1 Priority = 6 …….Rule 1

if (bandwidth > 4 Mbps && bandwidth < 6 && No. of Flows > 50 and No. of Flows < 75)
Action 2 Priority = 4 ……..Rule 2

if (protocol = tcp | protocol = udp) Action 3 Priority = 1 ……. Rule 3

Assume that there are M hosts, and we get triggers notification corresponding to rule 1 and rule 2. Assume that there are N trigger data records in the "Re-Evaluate" message (one flow or host information). In this case the N data records will either match rule 1 or rule 2 and hence Action 1 or Action 2 will be applied to the N hosts. But the M-N hosts (well-behaving hosts) will have no trigger data record in the trigger and hence will not be evaluated and no actions will be applied to M-N hosts. It is because of this reason; the N+1th evaluation is performed with no trigger data so that the default policy will get evaluated, since rule 1 and rule 2 will not match because of absence of trigger data.

4.3.3 Action Manager

The Action Manager has been designed to handle configuration of a number of Network devices at the same time. The Action Manager is a thread-based implementation. The action manager spawns one Configuration thread per network device. On receiving a Java Serialized object of the Action table from the Policy Evaluator, the Action manager performs a LDAP search to determine the network devices and the interfaces that belong to a role combination. It also retrieves the "MgmtAccessSettingsClass" (Section 4.2.3.2) that contains login information of the network device.

A thread table is maintained with {Thread Name, Thread object} as {key, value} pair. The Action manager performs a lookup in this table to check if a configuration thread to the network device to be configured is active. A new configuration thread is spawned when a corresponding thread is not found. It reuses an existing configuration thread, if found.

4.3.3.1 Configuration Threads

Actions to be performed at the network device are passed to configuration thread as Action objects. The action object contains the data set and the action tree. For each data record in the set, configuration thread performs the actions specified in the action tree. One or more actions can be associated with a rule. Each action can be a compound action or a simple policy action. The action tree is maintained according to the action order. The Configuration threads support Do-All, Do until success and Do until Failure execution strategies. The Configuration threads translate QPIM [25] Differentiated Services actions (Figure 4.16) into device specific configuration commands. Figure 4.16 provides a graphical representation of QPIM Differentiated Service actions and their parameters.

The configuration thread opens a TCP/IP session with the configuration server. Network Device information like IP address of the network device, the model type of the network device, telnet password and root password of the network device are sent during the session initiation. It forms the configuration server specific commands by using the data records in the data set and using action tree. The data record contains either the trigger data record or the policy variables and values of the rule.

After completing all the actions assigned to it by the action manager, the Configuration thread sleeps for 10 seconds and then it wakes up again to check if any new actions have been assigned. The Action Manager can also interrupt the thread when the thread is in the sleeping state. An idle timer is maintained which is incremented whenever the thread goes to sleep state, on receiving one or more actions the idle timer is reset. When the idle timer reaches 2 minutes, i.e. the thread is idle for two minutes; the thread removes its object reference from Trigger Table maintained by the action manager and quits. The value of the idle-timer is configurable.

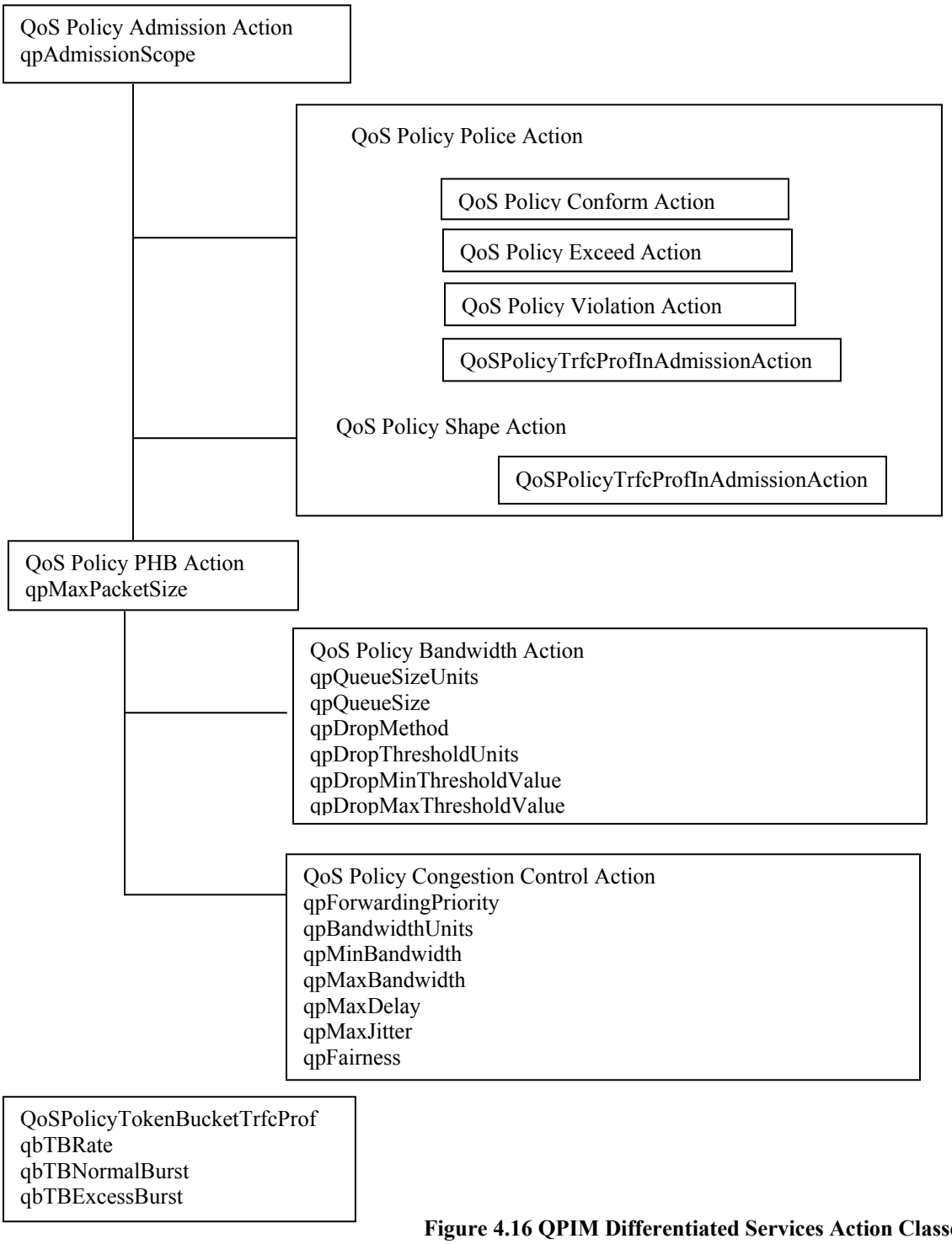


Figure 4.16 QPIM Differentiated Services Action Classes

4.3.4 Configuration Server

The purpose of this component is to provide device independent interface for configuring Network devices having different capabilities and manufactured by multiple vendors. The server provides a TCP/IP based command-line interface to its clients. Clients can open a session and issue a sequence of commands to the Configuration server. The server currently supports Cisco 2500 Diffserv Router and Catalyst 2900 Switch. It configures the Cisco routers and switches using Cisco Modular QoS command Line Interface [44].

The server is capable of configuring the following,

1) IP-Based Access-lists
2) Priority Queuing
3) Custom Queuing
4) Differentiated Service Configuration
    i. Policer
    ii. Marker
    iii. Classifier
    iv. Shaper
5) Congestion Control
    i. Tail-Drop
    ii. Random Early Detection
    iii. DSCP/ IP Precedence based RED.
6) Bandwidth Configuration (kbps or percent)

The configuration server is implemented using Expect, a tool for automating interactive applications. The configuration server opens telnet session to the network device and configures the network device using the command-line interface. The one main disadvantage with command-line interface is that the program needs to be changed when the command-line interface is changed. But vendors of network devices are more careful in not modifying their command-line interface. In future, Common Open Policy Services [24] (COPS) protocol can be used to configure network devices.

Adding New Network Elements with different Command Line Interface language

The Configuration Server is the only component that needs to be modified, when a new network device from a different vendor is incorporated into an existing network. The Configuration server API must be extended to specify new commands corresponding to new features in the network devices. Appropriate translation algorithms must be implemented in the Configuration server when a new device or a new action is introduced.

This chapter presented the design goals of the policy server and policy management toolset. It described the components of the policy server in terms of their functionality and showed how they co-ordinate and serve in achieving the policy server design goals. It also explained the different features supported by the policy management toolset and showed snapshots of its web based graphical user interface. The role based conflict detection algorithm used by the management toolset to detect intra-policy conflicts was also described. Chapter 5 explains how the policy server was deployed and validated in different QoS policy scenarios. The results of the evaluation are also presented in the next chapter.

5. DEPLOYMENT AND VALIDATION OF NETWORK POLICY SERVICES

This chapter describes and discusses the Differentiated Services (DiffServ)[12] test-bed used in this work, the different QoS policies (test cases) that were employed to assess the Policy Server developed as part of this thesis, and the results of applying these validation tests.

5.1 Test-Bed Setup

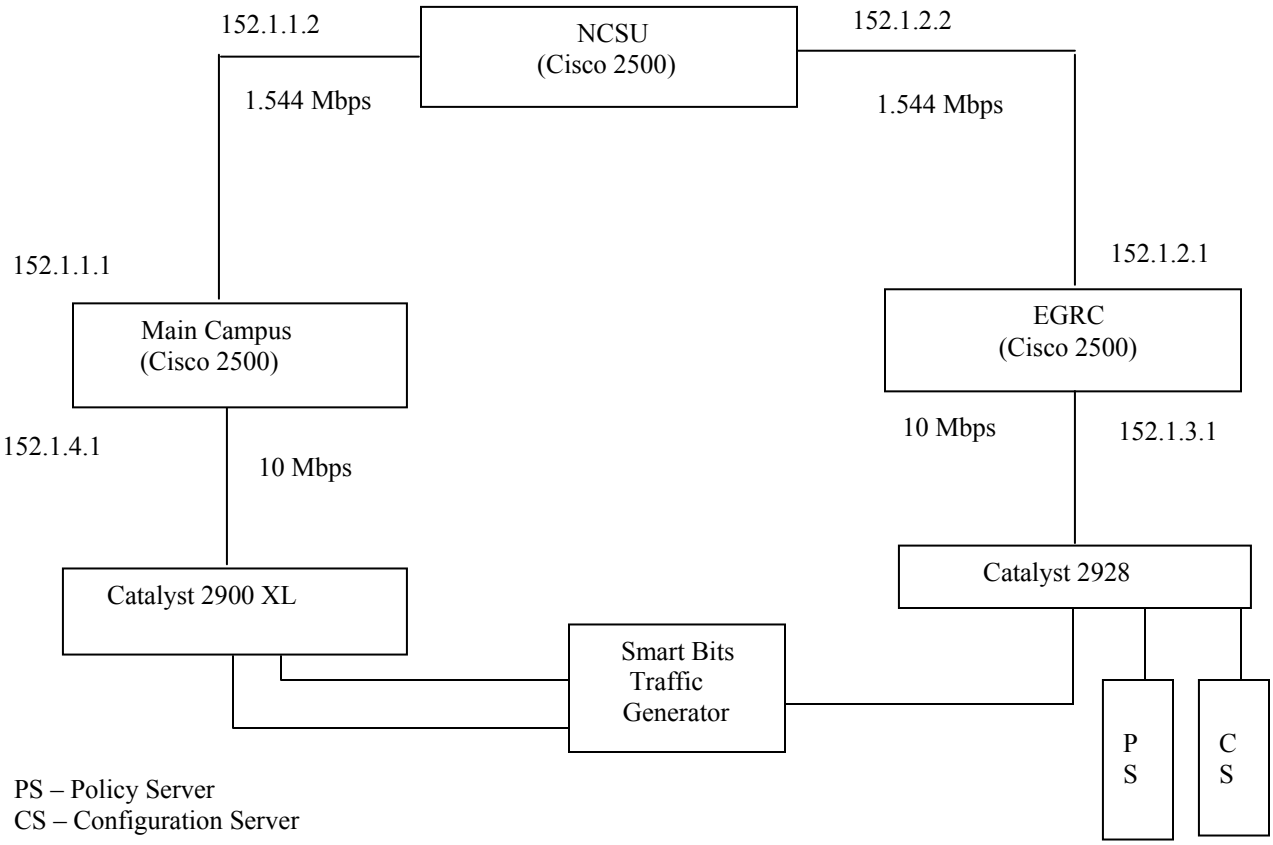


Figure 5.1: Test-Bed Setup

The test-bed models a Campus Network. The "NCSU" box represents the core or backbone network. "Main-Campus" and "EGRC" represent two networks that serve users in two separate locations on the campus. Traffic between the two networks is routed through the core network. The Policy Server and the Configuration server are installed in the EGRC network. In this test-bed, the routers for the "Main

Campus" and the "EGRC" are connected to the core router (NCSU) using a serial interface. The traffic in the networks was generated using Smart Bits 200 test device. Ports 1 and 2 of the test device were connected to a switch in the "Main Campus" network. Port 3 of the test device was connected to a switch in the "EGRC" Network. The devices used in constructing this test-bed are listed below. The network itself was considered to be capable of providing DiffServ as supported by the Cisco devices and the code loads employed.

Network Device Information:

Switches	→ Catalyst 2900 XL and Catalyst 2928
Routers	→ Cisco 2500 DiffServ capable routers (IOS v12.2)

5.2 Policy Scenarios

This section provides examples/test-cases of static, dynamic and time-based policies used in the experiments. The examples/test-cases provided here are part of the test case suit used for the evaluation of the policy server functionality. We distinguish static and dynamic policies. Access-list policies, Traffic marking/classification policies, and Core network provisioning policies were deployed to verify the support of static policies. Bandwidth violation and Network protection policies were tested as part of dynamic policy validations. Support for time-based policy was evaluated by deploying policies that provide guaranteed QoS to VoIP and video traffic at some user specified time intervals. These test cases were designed to demonstrate the ability of the policy server to maintain different QoS policies and to express the level of flexibility achieved by the use of IETF Policy Core Information Model.

In the following test cases, policies are presented using tables. The header of the table provides properties of the policy group, i.e. its name, decision strategy and the role, or role combination, to which the policy group is applied. A horizontal line separates the policy rules and properties of a policy group. Policy rules are listed in the order of evaluation, i.e. high priority rules are in the top of the table. The conditions and properties, i.e. execution strategy and priority of each policy rule, are marked in bold characters. The action(s) of the policy rule are shown on the line after the policy rule's property. If a policy rule has more

than one action, “Action order” property is displayed adjacent to the action. Simple policy action are represented as “policy variable=policy value”, e.g. DSCP=EF where DSCP is policy variable and EF is policy value. Policy actions that have more than one parameter are represented by their class name, followed by the parameters and values. For example, an action of type QoSBandwidthAction is represented as,

QoSBandwidthAction (Action class name)

qpForwardingPriority = 1 (Action parameters and values)

qpBandwidthUnits = 1

qpMaxBandwidth = 30

5.2.1 Static Policies

Static policies apply a fixed set of actions in a pre-determined way according to a set of pre-defined parameters that define how the policy is used. These policies are enabled and disabled by the administrator only. Static policies do change infrequently.

5.2.1.1 Access-List Policies

In this test case, traffic flows of three applications namely Kazaa, Napster, EDonkey (identified by port numbers 1214, 5555, 4661 respectively) were restricted from entering the core network by defining appropriate policies. The policy was specified as a policy group consisting of a set of policy rules (Table 5.1). Each policy rule was used to specify the type of access provided to application traffic. A default policy rule that applies to all application traffic that was not explicitly specified by the policy rules in this policy group was also defined.

The order of evaluation of the rules was specified by the priority of each rule. Higher priority rules were evaluated first and their actions performed before lower priority rules. The decision strategy of this non-trigger based policy group depends on the evaluation strategies supported by the network device. In most cases, network devices support only First matching strategy. The Policy Server translated the specified policy rules into corresponding network device configuration commands.

The access-list policy actions were defined as simple policy actions i.e. actions that either set or unset a variable to a specified value. A policy variable "Access_Type" and two policy values "Permit_Access" and "Deny_Access" were created for this purpose. Policy rules that permit an application traffic were specified with an action "Set Access_Type=Permit_Access" and policy rules that block an application traffic were specified with an action "Set Access_Type=Permit_Deny".

Policy rules that apply to common network applications like HTTP, FTP and Telnet were specified at the top of the access-list in order to avoid large number of IP packets passing through the entire IP access list at the edge router. Each policy rule consists of the IP address and/or the tcp/udp port numbers of the application that needs to be restricted. The policy group was deployed to control the incoming traffic at the Ethernet interface of the "Main-Campus" router. The policy group and Ethernet interface were assigned to policy role combination "Main-Campus&&Network_Egress" for this purpose. Table 5.1 shows the policy that was specified using the Policy management tool.

Table 5.1: Access List Policy

Policy-Group: Access-Policy-Group	**Decision Strategy: First-Matching**	**Role= Main_Campus&&Network_Egress**
if (IP_Protocol==TCP && Source_Port==HTTP && Flow_Direction==INCOMING)		**Priority=7**
Access_Type =Permit_Access		
if (IP_Protocol==TCP && SourcePort==FTP && Flow_Direction==INCOMING)		**Priority=6**
Access_Type =Permit_Access		
if (IP_Protocol ==TCP && Source_Port == TELNET && Flow_Direction==INCOMING)		**Priority=5**
Access_Type =Permit_Access		
if (IP_Protocol==TCP && SourcePort == KAZAA_PORT && Flow_Direction == INCOMING)		**Priority=4**
Access_Type =Deny_Access		
if (IP_Protocol==TCP && SourcePort == NAPSTER_PORT && Flow_Direction == INCOMING)		**Priority=3**
Access_Type =Deny_Access		

if (IP_Protocol==TCP && SourcePort == EDONKEY_PORT && Flow_Direction == INCOMING) **Priority=2**

Access_Type =Deny_Access

if((IPProtocol=TCP | IPProtocol=UDP) && Flow_Direction == INCOMING) **Priority=1**

Access_Type =Permit_Access

The Default policy that permits all traffic from all hosts is also specified. This Default policy is required; otherwise all traffic from all hosts other than HTTP, FTP and Telnet will be denied access. The Administrator groups these policies in a Policy Group called Access-Policy-Group.

The policy server correctly translated the policy and configured corresponding Extended IP access lists in the Main-Campus router. The policy was successfully deployed and the required application(s) traffic was restricted from entering the network. The results of the policy deployment is shown graphically in Figure 5.2 and 5.3

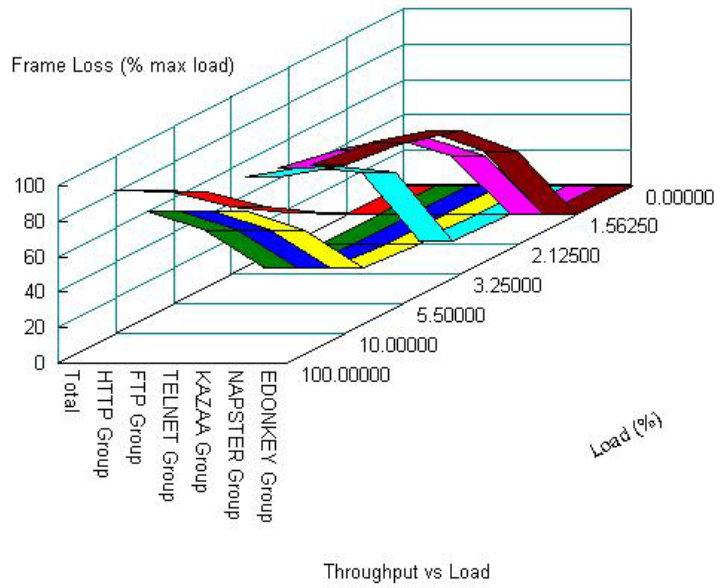


Figure 5.2 Throughput vs. Load before Access-list Policy Deployment

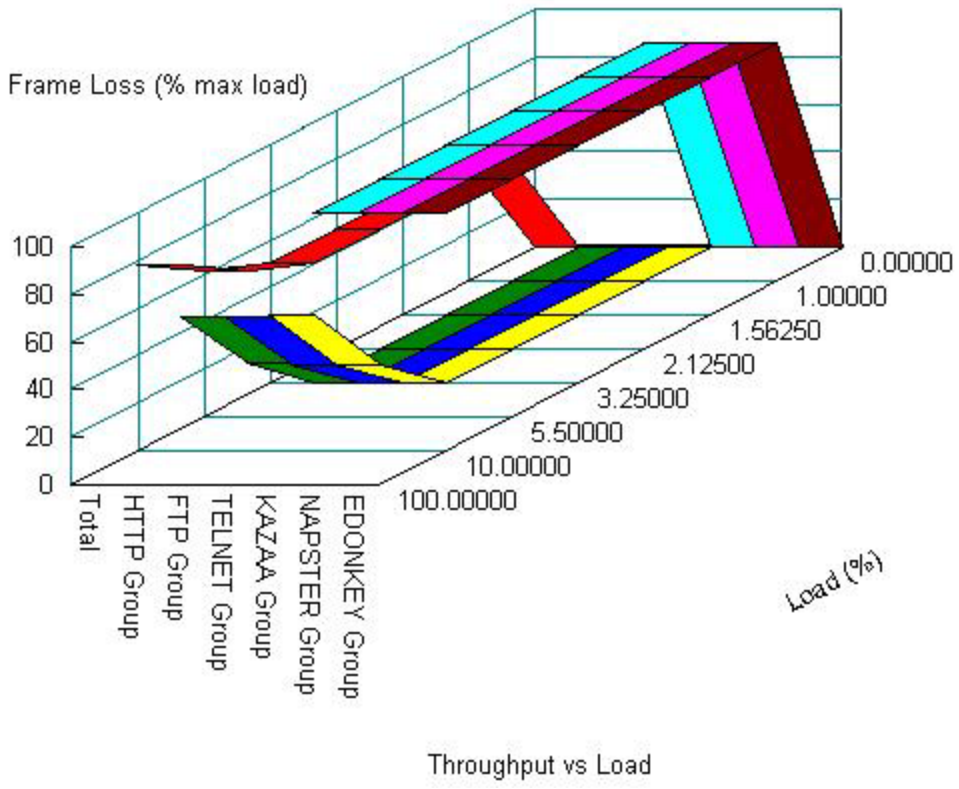


Figure 5.3 Throughput vs. Load after Access-list Policy Deployment

Figure 5.2 shows the packet (frame) losses of each application. As part of this test case, a total of six application traffic flows were generated (Http, Ftp, Telnet, Kazaa, Napster and Edonkey). Before the deployment of access-list policy, http, ftp and telnet applications experienced packet losses in the range of 65-70%. After deployment of access-list policy, napster, kazaa, and eDonkey applications experienced 100% packet losses as shown in Figure 5.3. The deployment of access-list policy resulted in dropping of packets from kazaa, napster and eDonkey applications at the Ethernet interface of the "Main-Campus" router thereby restricting these violating applications from entering the network and decreasing the bandwidth contention in the local area network. This also decreased packet losses (56 %) of Http, ftp and telnet applications.

5.2.1.2 Priority-Based QoS Policies

In a non-best effort (BE) based network, different users have different privileges and permissions to access different network services at different times. The level of QoS provided to a traffic flow(s) are indicated by

marking the packet using the Differentiated Services code point (DSCP) or IP precedence field in the IP header of the packets belonging to the flow.

A test case was designed in order to evaluate the policy server in the context of this scenario. In this test case, VoIP packets leaving a local area network were marked with EF (Expedited Forwarding) DSCP code point or Critical IP precedence (5). The policy was specified as policy group consisting of two policy rules (Table 5.2). Policy actions were specified as simple policy actions. The policy variables "DSCP", "IPPrecedence" and policy values "EF", "AF31", "CRITICAL_PRECEDENCE", "FLASH_PRECEDENCE" were used. Each policy rule consists of two policy actions with a "Do until success" action strategy. This is an excellent example for demonstrating how an administrator can specify device-independent policies. The Policy server will first try to configure DSCP marking, if the device does not support DSCP it will then configure IP Precedence. The administrator does not need to know which device has DSCP capabilities or not. The policy group was applied to incoming traffic at the Ethernet interfaces of the "Main-Campus" and "EGRC" routers. Policy role combinations "Main-Campus&&Network_Egress" and "EGRC&&Network_Egress" were used for this purpose.

Table 5.2: Priority-Based QoS Policy

Policy-Group: Edge-QoS-Policy-Group	**Role combination: Main_Campus&&Network_Egress** **Decision-Strategy: All-Matching**	
if (IPProtocol==UDP && SourcePort==SIP_PORT && Flow_Direction==INCOMING)	**Execution Strategy: Do Until Success**	**Priority = 2**
DSCP = AF31	Action Order = 2	
IP Precedence = FLASH_PRECEDENCE	Action Order = 1	
If (IPProtocol==UDP && SourcePort==RTP_PORT && Flow_Direction==INCOMING)	**Execution Strategy: Do Until Success**	**Priority = 1**
DSCP = EF	Action Order = 2	
IP Precedence = CRITICAL_PRECEDENCE	Action Order = 1	

Table 5.2 shows that SIP (Session Initiation Protocol, a VoIP call control protocol) packets are marked with DSCP code point AF31 or with Critical Precedence (5) and RTP packets are marked with EF DSCP code point or with Flash Precedence (3).

The Policy Server correctly translated the policy into commands that are understood by the Configuration server. The Configuration server successfully configured the network devices by using the policy-map and class-map commands defined in Cisco Modular Quality of Service Command Line Interface [44]. The configuration performed by the Configuration server is shown in Appendix.

Another policy that specifies video packets to be marked with AF41 DSCP or Flash override precedence (4) and video call control packets to be marked with AF32 DSCP or Flash precedence (3) was also successfully tested.

5.2.1.3 Core-Network Policies

These policies are applied to the network devices at the core network. They are used to specify the type of queuing to be used and maximum amount of bandwidth allowed for specific network traffic class). They are enforced to make sure that packets marked with DSCP code points or IP Precedence receive guaranteed bandwidth and other QoS corresponding to their priority. A test case was designed to provision the bandwidth available in the core network. A bandwidth of 64 kbps and 386 kbps was provisioned for VoIP RTP traffic (VoIP packets with G.721 codec) and video traffic respectively. A bandwidth of 16 kbps was provisioned for video and voice call control traffic.

A policy group named “Core-QoS-Policy-Group” with three policy rules was defined. Policy variables “DSCP” and “IPPrecedence” were used to specify policy conditions that match packets belonging to VoIP and video packets. QoS bandwidth and congestion control actions specified in QoS Policy Core Information Model (QPIM [25]) were used to specify the amount of bandwidth, the type of congestion control mechanism and congestion control parameters (e.g. drop threshold level, tail drop queue size). Table 5.3 shows the policy that was specified.

Tail-drop and Random early detect (RED) congestion control mechanisms were specified for VoIP and video traffic respectively. The reason for applying Tail Drop to VoIP packets is that VoIP packets are mostly UDP packets; hence applying a random-detect congestion control algorithm will have no effect. The policy group was applied to outgoing traffic at all serial interfaces of the core network (serial interfaces of NCSU router) and at ingress interfaces (serial interface of "Main-Campus" router and "EGRC" router).

Table 5.3: Core-Network Policy

Policy-Rule: Core-QoS-Policy-Group Decision-Strategy=First-Matching Role combination: Core_Network&&Serial, Main_Campus&&Core_Network_Ingress, EGRC&&Core_Network_Ingress

---

if ((DSCP == EF | IP Precedence == 5) && Flow_Direction==OUTGOING) Execution Strategy = Do-All Priority = 3

QoSBandwidthAction Action Order=1

qpForwardingPriority=1 (Strict Priority / Pre-emptive Forwarding)

qpBandwidthUnits = 0 (kbps)

qpMaxBandwidth = 64

QoSCongestionControlAction Action Order = 2

qpQueueSizeUnits= 0 (Packets)

qpQueueSize =30

qpDropMethod = 1 (Tail Drop)

if ((DSCP==AF41 || IP Precedence==4) && Flow_Direction==OUTGOING) Execution Strategy = Do-All Priority=2

QoSBandwidthAction Action Order=1

qpForwardingPriority=0 (No Pre-emptive Forwarding)

qpBandwidthUnits = 0

qpMaxBandwidth = 386 (386 kbps)

QoSCongestionControlAction Action Order = 2

qpQueueSize Units = 0 (Packets)

qpQueue size = 75

qpDropMethod = 0 (random detect)

qpDropThresholdUnits = 0

qpDropMinThresholdValue = 20

qpDropMaxThresholdValue = 70

if ((DSCP == AF31 | IP Precedence == 3) && Flow_Direction==OUTGOING) Execution Strategy = Do-All Priority = 1

QoSBandwidthAction Action Order=1

qpForwardingPriority=0 (No Pre-emptive Forwarding)

qpBandwidthUnits = 0

qpMaxBandwidth = 16 (16 kbps)

QoSCongestionControlAction Action Order = 2

qpQueueSizeUnits= 0 (Packets)

qpQueueSize =30

qpDropMethod = 1 (Tail Drop)

The Policy Server correctly translated the policy into commands that are understood by the Configuration server. The Configuration server successfully configured the network devices by using the policy-map and class-map commands defined in Cisco Modular Quality of Service Command Line Interface [44]. It is important to note that the policy "Core-QoS-Policy-Group" is applied to two interfaces (serial0 and serial1) of the same network device NCSU router. In this case, the Configuration server creates only one configuration set (class-maps, policy-map, access-lists) and applies them to these interfaces instead of creating two same configuration sets. This reduces the amount of memory consumption in the network device. The configuration server also automatically deletes the configuration (class-maps, policy-map, access-lists) associated with the previous policy (if any) that was assigned to an interface before assigning a new policy to an interface. The configuration performed by the Configuration server is shown in Appendix.

5.2.2 Dynamic Policies

Dynamic policies are only enforced when needed, and are based on changing conditions of the network such as congestion, decrease in the Quality of Service requirements to a flow, detection of a host consuming large bandwidth, etc.

5.2.2.1 Bandwidth Usage Violation / Network Protection Policies

Typically Administrators provision resources in their network. This requires them to know what types of traffic circulates in their network, the QoS requirements, Service-level Agreements (SLA) that need to be satisfied, etc. In a large network provisioning can be a difficult task. Almost everything needs to be provisioned without knowing the dynamic state of network. For example, the network administrator may want to increase the bandwidth allocated to a flow, when the overall network traffic is low, or he/she may want to isolate misbehaving hosts from the network to protect QoS delivered to other users and to reduce the processing at the network devices. The threshold (say bandwidth consumption level) that is used to classify hosts as normal and mis-behaving hosts can also vary depending on the current overall network usage.

This scenario/test case was designed to validate the policy server in the context of Dynamic policies. In this scenario, the network administrator specifies specific policies that identify hosts that are not complaint with QoS SLAs. A user in the Main-Campus network is allowed to use a maximum of 3000 kbps and maximum of 200 flows. The administrator specifies two thresholds. When the bandwidth consumed is between 3000 Kbps and 5000 Kbps or the number of flows is between 200 and 300, a Low Priority Marking (i.e. packets belonging to the user are marked as best effort packets or low priority either by using DSCP or IP Precedence or by not marking the packet explicitly) is specified. When the bandwidth consumed is greater than 5000 Kbps or the number of flows greater than 300, an action that specifies the Policy server to shutdown the switch port to which the user is connected. This policy was also used to perform Pro-active management in the network. In this scenario, it was assumed that if the bandwidth consumption of a user is greater than 5 Mbps either the host is performing a security attack or is affected by such attacks.

A policy named “Bandwidth–Violation-Policy-Group” (Table 5.4) consisting of three policy rules was specified. Two of the policy rules specify policy conditions expressing the bandwidth thresholds and the corresponding policy actions to be taken when the conditions matched. Policy variables “BW_UTILIZATION”, “NUMBER_OF_FLOWS” and policy values “HIGH_LOAD”, “VERY_HIGH_LOAD”, “MAX_NO_OF_FLOWS” and “AVG_NO_OF_FLOWS” were used. The Port

shutdown action was specified as a simple condition using the policy variable "Port_Status" and policy values "UP" and "DOWN". The low priority action was specified as a simple action with policy variables "DSCP" and "IPPrecedence" and policy value "BEST_EFFORT". The third rule (default rule) was used to specify that conforming packets be marked with class 2 DSCP code point (2) or Immediate precedence (2). This rule will be evaluated if and only if the user traffic did not match the first two rules. This order of evaluation was achieved by specifying a First-Matching decision strategy and by assigning priorities to the policy rules.

Table 5.4: Bandwidth Violation/Network Protection Policy

Policy Group: Bandwidth –Violation-Policy-Group Decision Strategy = First Matching Role: Main_Campus
If ((flow_direction==OUTGOING) && **(bw_utilization > VERY_HIGH_LOAD || number_of_flows > MAX_NO_OF_FLOWS))** **Execution Strategy: Do until success Priority =3** Port_Status=DOWN Action Order = 1
If((bw_utilization > HIGH_LOAD Mbps && bw_utilization < VERY_HIGH_LOAD Mbps && flow_direction==OUTGOING) || (number_of_flows > AVG_NO_OF_FLOWS && number_of_flows < MAX_NO_OF_FLOWS && flow_direction== OUTGOING)) **Execution Strategy: Do until Success Priority = 2** DSCP = 0 Action Order = 1 IPPrecedence = 0 Action Order = 2
if (IPProtocol ==TCP || IPProtocol == UDP) Execution Strategy: Do until Success Priority = 1 DSCP = 2 Action Order = 1 IPPrecedence = 2 Action Order = 2

On enabling the policy, the Policy server identified that the policy is a trigger based policy and performed a trigger based policy evaluation (Section 4.3.2). The Trigger manager component of the policy server correctly translated the policy variables and values into Resource Usage Monitor (RUM) parameters and sent a trigger notification request to RUM. The trigger notification was successfully processed by RUM.

After receiving trigger notification (alerts) from RUM, Trigger manager retrieved trigger information (host IP address, amount of bandwidth, number of flows) from RUM using HTTP over SSL. The information (event/system message) was then sent to policy evaluator. After evaluation of the policy, required policy actions were translated to commands that are understood by configuration server. The configuration server successfully configured the network device to either shutdown the port or to mark the traffic appropriately. When the policy was disabled, the trigger manager automatically removed/cancelled the trigger notification request from RUM.

5.2.3 Time-based Policies

5.2.3.1 QoS to Conference calls (VoIP Traffic) during a specific time period

A dynamic QoS Policy may also be time-based. Such policies may specify different levels of QoS at different times. It may also be essential for a network administrator to provide QoS to applications for a specified period of time. A good example is a Conference Call using H.323 video or Voice over IP. For instance, when researchers in different parts of world want to meet over a conference call, during the duration of the call, say an hour, a day, the administrator may wish to provide QoS guarantees. For example, by provisioning the network for sufficient bandwidth and applying appropriate queuing mechanisms QoS guarantees like packet loss less than 1%, 64 Kbps bandwidth, delay less than 150 ms can be provided. However, the admin may not want to provision the network at this level on a permanent basis.

Using the Policy Server, the administrators can dynamically add/enable new policies and remove/disable existing policies from policy groups. A test case was designed to validate the policy server in this context. A policy group named “VoIP-Conference-QoS-Policy-Group“ (Table 5.5) consisting of two rules was specified. The policy rule with lower priority (default policy) was enabled at all times. The default policy specifies that all TCP and UDP packets be serviced using best effort mechanism. A high priority policy rule with policy condition that is used to match packets belonging to VoIP traffic from specific host(s) was defined. Policy action of the high priority rule was specified to mark matching packets with Expedited forwarding (EF) DSCP code point or Critical IP Precedence (5). The high priority rule was enabled only

when required. The time condition of the policy was used to specify the policy validity time interval (i.e. start and end time of the conference call). The Policy management tool can be used to add /modify/delete policy conditions, time conditions, actions of a policy. The user can also add new policies to an existing policy group and can change the priority of the policies in the policy group at any point in time. When the high priority rule is valid, it will override the default rule. This is achieved by specifying a First-Matching decision strategy for the policy group evaluation. The policy was applied to incoming traffic at the Ethernet interface of the Main-Campus network. The Policy role property of the policy was set to "Main-Campus&&Network_Egress".

Table 5.5: Time-Based VoIP Conference Call Policy

Policy Group: VoIP-Conference-QoS-Policy-Group Decision Strategy: First Matching
Role: Main_Campus&&Network_Egress
if ((SourceIPv4Address == RESEARCH1_IPADDR && (SourcePort == RTP_PORT) && (flow_direction="INCOMING"))
Execution Strategy = Do until success Time Condition: 20-June-2002 9:00 Hrs to 20-June-2002 9:04 Hrs Priority = 5
DSCP =EF Action Order = 2
IP Precedence = CRITICAL_PRECEDENCE Action Order = 1
if ((IPProtocol == TCP | IPProtocol == UDP) && (flow_direction=="INCOMING"))
Execution Strategy: Do Until Success Time Condition: All-Times Priority = 1
DSCP =BEST_EFFORT Action Order = 2
IP Precedence = BEST_EFFORT Action Order = 1

IPerf network performance measurement tool was used to simulate voice traffic. As part of the test, seven different flows (three TCP, four UDP flows) were generated. An UDP traffic flow with a bandwidth of 64 kbps and 298 bytes datagram size was used to simulate G.729 voice traffic. Measurements were taken at 30-second intervals. The results of the tests are shown graphically in Figures 5.4, 5.5 and 5.6. The high priority policy rule was valid from 211th second to 510th second (i.e. from 8th to 17th 30 second interval). During the high priority rule's validity time, packets of the voice traffic flow were marked with Expedited

Forwarding (EF) DSCP point at the Ethernet interface of "Main-Campus" router. The Core network provisioning policy

(Section 5.2.1.3) was used to provide QoS to the voice traffic. A class-based strict-priority scheduling mechanism at a bandwidth of 64 kbps was provisioned for the voice traffic. Tables 5.6, 5.7, 5.8 show the values of bandwidth, delay jitter and packet loss before, during and after the valid time of the high priority rule respectively. The voice traffic during policy validity time had a packet loss of 0%, and a delay jitter less than 6.5 ms, and a bandwidth of 64 kbps.

Table 5.6: QoS Parameters of Voice Traffic Before Policy Validity Time Interval

Time	Bandwidth (kbps)	Delay Jitter (ms)	Packet Loss (%)
0 –30	63.3	29.645	1.6
31-60	63.5	34.441	0
61-90	64.1	36.443	0
91-120	64.1	41.207	0
121-150	62.8	37.228	1.7
151-180	64.1	37.126	0
181-210	64	37.086	0

Table 5.7: QoS Parameters of Voice Traffic during Policy Validity Time Interval

Time (secs)	Bandwidth (Kbps)	Delay Jitter (ms)	Packet Loss (%)
211 –240	64.0	8.757	0
241-270	63.9	4.842	0
271-300	64.1	6.461	0
301-330	64.0	5.548	0
331-360	64.0	6.182	0
361-390	63.9	5.331	0
291-420	64.0	6.294	0
421-450	64.1	6.808	0
451-480	64	4.999	0
481-510	63.8	4.206	0

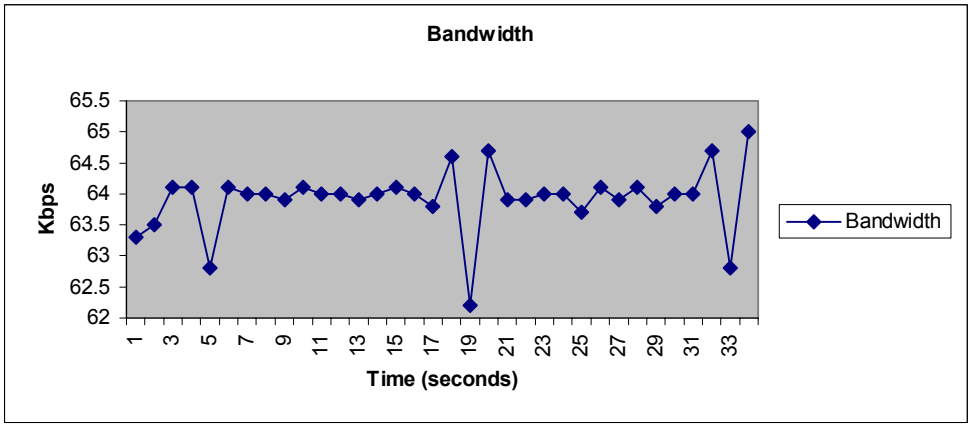


Figure 5.4: Bandwidth vs. Time graph of Voice Traffic

Figure 5.4 shows the change in bandwidth provided to the voice packets of the conference call when the high priority policy rule was enabled at 9:00 Hrs. Since the bandwidth consumed by voice packets was not larger, there was no dramatic change in bandwidth before and after policy deployment. But the bandwidth provided to the voice packets during policy deployment remained constant at 64 kbps, it did not vary as it was before policy deployment.

Table 5.8: QoS Parameters of Voice Traffic after Policy Validity Time Interval

Time (secs)	Bandwidth (Kbps)	Delay Jitter (ms)	Packet Loss (%)	Time (secs)	Bandwidth (Kbps)	Delay Jitter (ms)	Packet Loss (%)
511-540	64.6	18.793	1.1	751-780	64.1	35.736	0
541-570	62.2	16.399	0	781-810	63.9	37.884	0
571-360	64.7	36.61	0	811-840	64.1	39.092	0
601-630	63.9	37.512	0	841-870	63.8	38.28	0
631-660	63.9	36.217	0.082	871-900	64	38.265	0
661-690	64.0	35.999	0	901-930	64	38.483	0
691-720	64.0	37.999	0	931-960	64.7	16.717	0.83
721-750	63.7	35.927	0.24	961-990	62.8	38.21	0

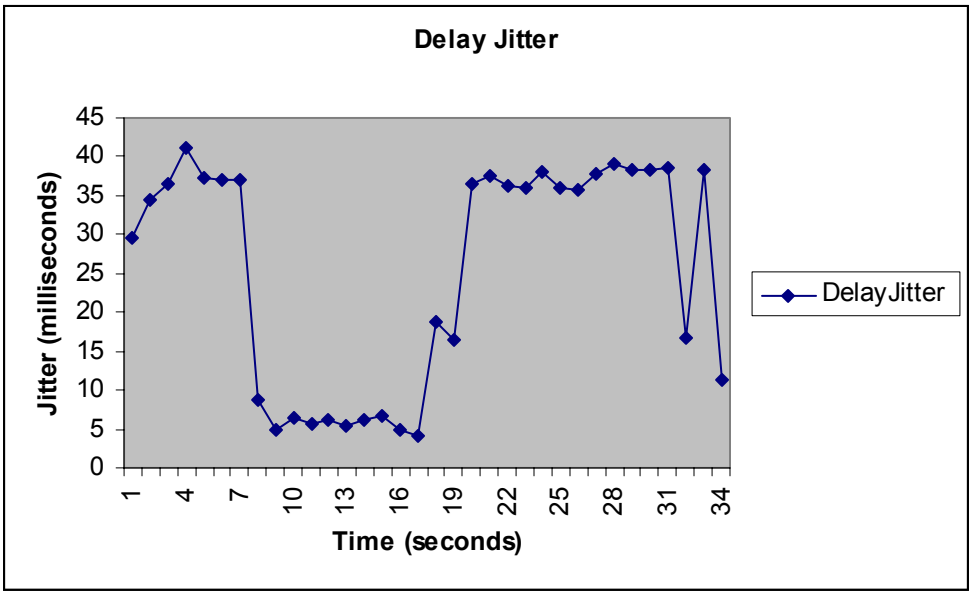


Figure 5.5: Delay Jitter vs. Time graph of Voice Traffic

Figure 5.5 shows the change in delay jitter of voice packets generated as part of this test-case. Before the deployment of the high priority policy rule, voice packets experienced delay jitter ranging from 29 ms to 38ms. In the 8th measurement time interval (interval at which the high priority policy rule was deployed), we can see a steep decrease in delay jitter to 8.757 ms. The delay jitter was in the range of 4-6.5 ms during the policy's validity time. Such small delay jitter was achieved by using strict priority scheduling mechanisms in the network devices along the path to the destination (EGRC network). The delay jitter increased back to 37-39 ms range after the policy validity timeout.

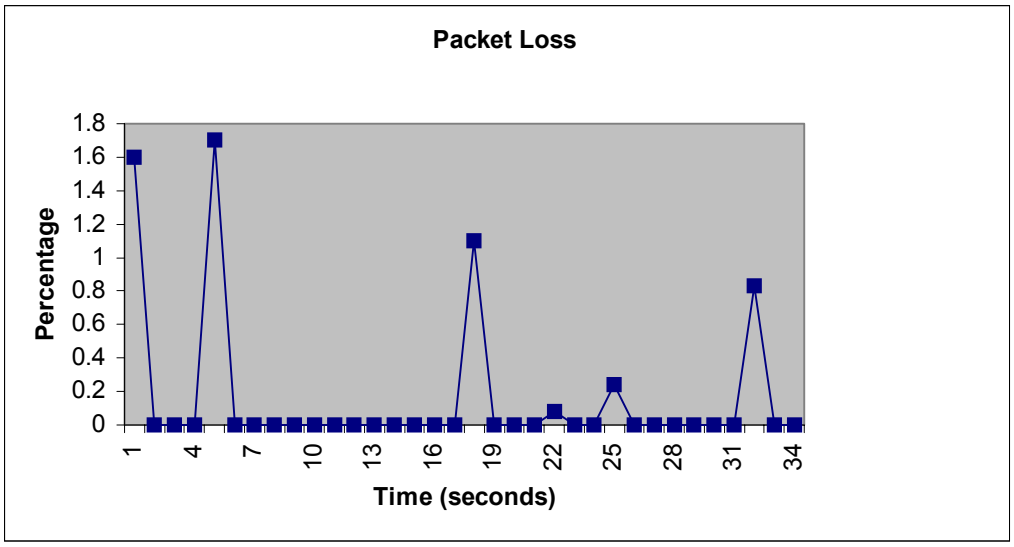


Figure 5.6: Packet loss vs. Time graph of Voice Traffic

Figure 5.6 shows the percentage of voice packet losses. There were few packet losses around 1.7% at certain time intervals. For a good user experience and perfect quality (Table 1.1), the packet loss of voice should be zero. In the above graph, we can observe that in time intervals during which the policy is deployed (8th to 17th time interval), there were no packet losses. Packets continued to get lost after policy's validity timeout.

5.2.3.2 Guaranteed Bandwidth To Video Traffic

Some applications may require QoS guarantees at periodic intervals of time. For example, a network administrator may be required to provide guaranteed QoS to a Class that uses Video over IP services, every three days in a week for a specified number of months. A test case was designed to validate the policy

server in this scenario. In this test case, a network administrator provides guaranteed bandwidth of 386 kbps to a graduate class video traffic on every Monday and Wednesday during summer semester (May to August) from 8:30 PM to 10:10 PM.

A policy group "Video-Policy-Group" (Table 5.9) consisting of two policy rules was specified. The high priority rule contains policy condition that is used to match the packets belonging to the video traffic. The policy action specifies that the matching packets be marked with AF41 Diffserv code point or Flash override precedence (4). The validity time of the high priority rule was specified using "validMonthsMask", "validWeekDaysMask", "validDaysMask", and "validTimeOfADay" properties of the time condition. A default policy that marks all packets with Best-Effort code point was also specified. A reusable time condition "All-Times" was used to specify that the default policy is valid at all times.

Table 5.9: Video Service Policy

Policy Group: Video-Policy-Group Decision Strategy = First Matching Role: EGRC&&Network_Egress
If (SourceIPv4Address==CSC791BIPAddress && SourcePort==POLYCOM-VIDEO_PORT
&&Flow_Direction==INCOMING)
Execution Strategy = Do until Success Priority = 2
Time Condition Properties
Overall Validity Time = 15-May-2002 8:00 Hrs to 09-August-2002 18:00 Hrs
Valid Months = All
Valid Week Days = Monday and Wednesday
Time of A Day = 20:30 Hrs to 22:10 Hrs
DSCP =AF41 Action Order = 2
IP Precedence = 4 Action Order = 1
If (SourcePort == REAL-TIME-VIDEO-PORT && Flow_Direction==INCOMING)
Execution Strategy: Do Until Success Time Condition: THISANDPRIOR Priority = 1
DSCP =0 Action Order = 2
IP Precedence = 0 Action Order = 1

IPerf network performance measurement tool was used to simulate video traffic. As part of the test, four different flows (one TCP, three UDP flows) were generated. An UDP traffic flow with a bandwidth of 360 kbps and 1470 bytes datagram size was used to simulate video traffic. The high priority rule was automatically enabled at every start time and disabled at every end time of its validity time interval. The Policy server performed reconfigurations at the network devices at the corresponding start and end times. During the validity time interval, video packets specified by the high priority rule were appropriately marked and received guaranteed QoS (the core network was statically provisioned to provide the required bandwidth and to apply congestion control mechanisms). All non-matching packets and video packets sent during non-class hours were serviced using best-effort mechanisms.

Table 5.10: QoS Parameters of Video Traffic Before Policy Validity Time Interval

Time (Hours)	Bandwidth (Kbps)	Delay Jitter (ms)	Packet Loss (%)
19:00 –19:10	338	109.394	4.7
19:10 –19:20	298	64.186	17
19:20 –19:30	285	57.91	17
19:30 –19:40	288	77.638	20
19:40 –19:50	293	50.898	19
19:50-20:00	287	80.034	20
20:00-20:10	292	98.229	20
20:10-20:20	292	80.308	18
20:20-20:30	289	97.946	21

Table 5.11: QoS Parameters of Video Traffic during Policy Validity Time Interval

Time (Hours)	Bandwidth (Kbps)	Delay Jitter (ms)	Packet Loss (%)
20:30-20:40	354	71.742	5.5
20:40-20:50	360	23.437	0
20:50-21:00	360	20.958	0
21:00-21:10	360	22.979	0
21:10-21:20	360	22.429	0
21:20-21:30	360	22.435	0
21:30-21:40	360	22.198	0
21:40-21:50	360	24.78	0
21:50-22:00	360	22.716	0

Figure 5.7, 5.8, 5.9 graphically illustrate the impact of the high priority policy rule deployment. A class-based weighted fair queuing mechanism was provisioned to provide a bandwidth of 386 kbps to video traffic. Tables 5.10, 5.11, 5.12 show the values of bandwidth, delay jitter and packet loss before, during and after the valid time of the high priority rule respectively. The video traffic during policy validity time had a packet loss of 0%, and a delay jitter less than 25 ms, and a bandwidth of 360 Kbps. (Though we provisioned for 386 Kbps, traffic was generated only at the rate of 360 Kbps).

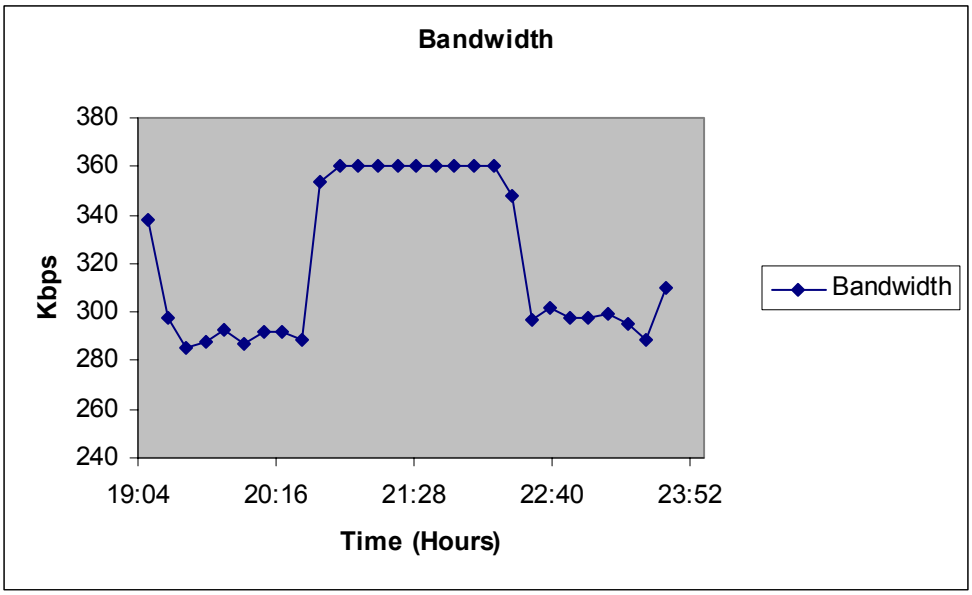


Figure 5.7: Bandwidth vs. Time graph of Video Traffic

Figure 5.7 shows the change in bandwidth provided to the video packets of the graduate class when the high priority policy rule is enabled at 20:30 Hrs. Before 20:30 hrs, bandwidth was in the range of 285 – 300 Kbps. In the time interval 20:30 Hrs - 20:40, we can see the sharp increase in bandwidth provided to the video traffic. This is because the policy server had configured the Main-Campus router to mark all specified incoming video packets at the Ethernet interface with AF41 DSCP or Flash override precedence (4). These marked packets were then forwarded using class based weighted fair queuing mechanisms along the path to their destination (EGRC network) thereby ensuring required bandwidth.

Table 5.12: QoS Parameters of Video Traffic after Policy Validity Time Interval

Time (Hours)	Bandwidth (Kbps)	Delay Jitter (ms)	Packet Loss (%)
22:10-22:20	348	85.51	3.5
22:20:22:30	297	108.987	18
22:30:22:40	302	64.217	16
22:40:22:50	298	68.742	16
22:50:23:00	298	51.691	17
23:00-23:10	299	44.556	18
23:10-23:20	295	109.198	17
23:20-23:30	289	54.444	20
23:30-23:40	310	56.444	14

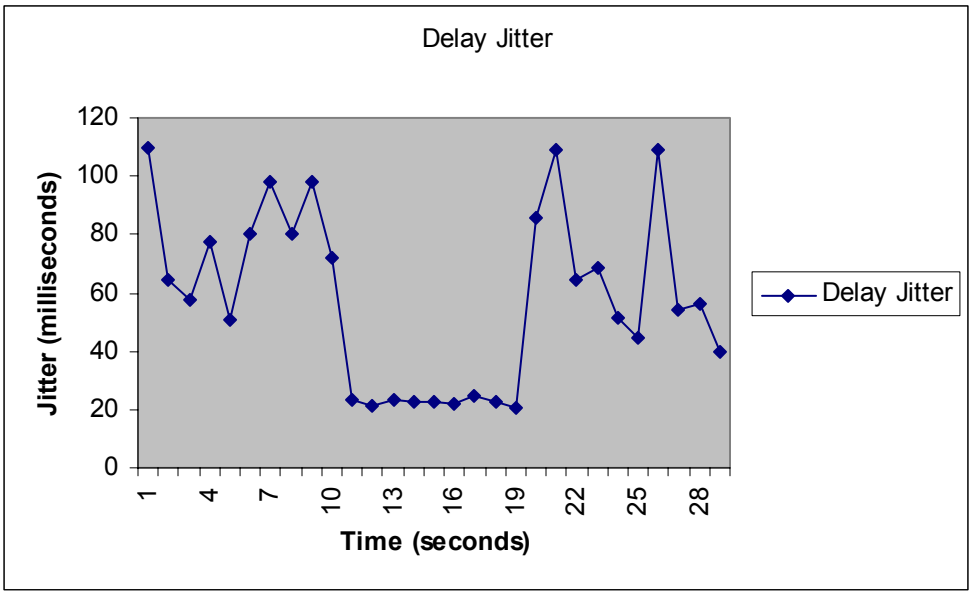


Figure 5.8: Delay Jitter vs. Time graph of Video Traffic

Figure 5.8 shows the change in delay jitter of video packets received at the destination (EGRC network). Before the deployment of the high priority policy rule, video packets experienced delay jitter ranging from 50 ms to 110ms. After the deployment of the high priority rule, delay jitter of video packets was reduced to 22-25ms. It is worth noting that after policy deployment, the delay jitter experienced by video packets is greater than that of voice packets (< 6.5 ms). This is because voice packets were forwarded using strict priority scheduling mechanism thereby resulting in very small delay and jitter. When compared to voice applications, video applications are more tolerant to delay.

Figure 5.9 shows the percentage of video packet losses. Packet loss was in the range of 15-21 % before policy deployment. In the time interval 20:30 – 20:40 Hrs, packet loss was greatly reduced to 5.5% and subsequently to 0% in the next intervals. This is because the policy server had deployed the high priority policy at 20:30 Hrs and hence the video packets received appropriate QoS. We can also see that the packet loss increases back to 15-21% range after the policy's validity timeout.

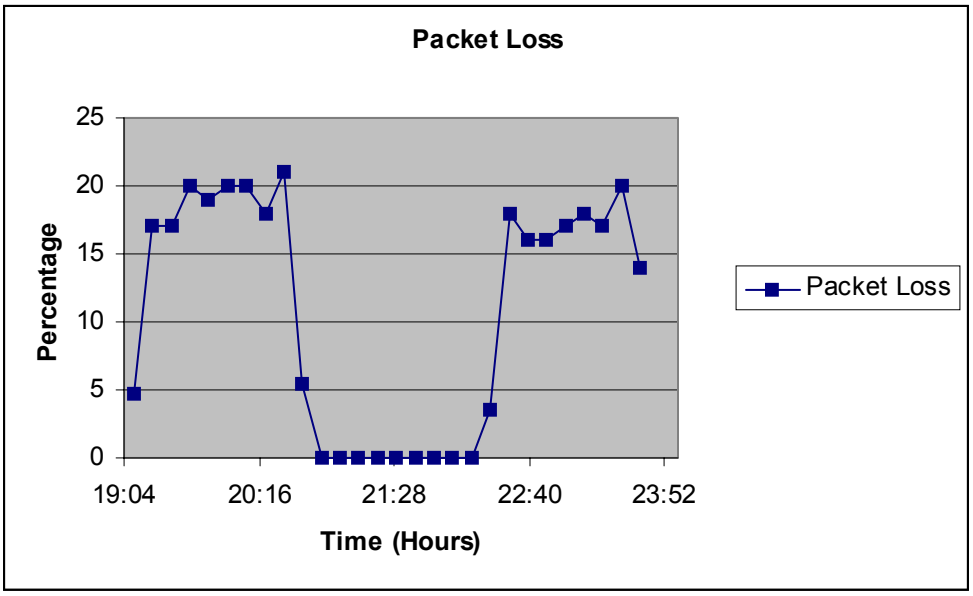


Figure 5.9: Packet Loss vs. Time graph of Video Traffic

This chapter explained scenarios for deployment of static, dynamic and time based policies and showed how to specify policies to protect and guarantee QoS to end user applications. It also illustrated the level of granularity and flexibility provided to network administrator by the Policy Server and Policy Management Tool. The next chapter presents the conclusions of this thesis and discusses future work in Policy Server implementation.

6. CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

A traffic-aware and standards based Policy Server and Policy Management Tool was implemented and validated. Real world policies such as policies providing QoS to individual and aggregated voice, video traffic flows were deployed. Time based QoS policies that would provide QoS to a packet flow only during specified times were used to provide QoS to a simulated video traffic of a graduate class and conference calls. The current implementation supports access-list policies and Differentiated Services QoS policies. A new Role based Policy Conflict Detection algorithm was developed. The algorithm was used to detect intra-specific policy conflicts among policy rules, nested policy rules and policy groups.

Pro-active network management was performed in two ways,

a) By specifying static access list policies. The IP address of the violating host and/or application port number was specified as policy variable in the policy condition. Access to the network was denied by specifying access-list simple policy action.

b) By specifying dynamic measurement-based policies. The threshold levels (such as maximum bandwidth, maximum number of flows) that identify violating hosts were specified as policy conditions. Policy actions such as marking traffic to a lower priority, port shutdown were specified to isolate or reduce the effect of violating hosts in the network, thereby protecting QoS provided to other non-violating hosts in the network.

The performance of the policy server was good. Good performance was achieved by faster retrieval of policy information using optimized LDAP searches and efficient handling of triggers, timer events and user events. The Policy Server is interoperable with other Policy Servers that use the IETF Policy Core Information Model [21,22] and can be extended to build a distributed system of Policy Servers to enforce QoS policies across multiple domains.

In the current implementation, network devices are configured using command line interface. This requires new translation procedures to be implemented when new network devices are added to the network. COPS [24] protocol that provides standardized communication interface to network devices can be used to eliminate this overhead.

The Policy Server is intended for research and educational purposes and can be used in production after performing some more reliability tests. The implementation and validation performed as part of this work, proved that Policy based Networking is an effective solution for providing and protecting QoS to mission critical applications.

6.2 FUTURE WORK

6.2.1 Using Rule Matching Statistics in Policy Evaluation

The Policy evaluator component of the Policy Server can be enhanced to keep statistics of rule matches. By using this functionality, different actions can be specified depending on the nth number of rule match. Such policies will be useful for signature based security intrusion detection. Security attacks like port scans (extending over a short/long period of time) can also be detected. For example, let's assume that we have a signature "if the number of flows of a host is greater than say N for a time period t, then the host is involved in performing port scan ". In order to detect port scans of this signature, a policy can be specified with a condition "Number_of_Flows > N && Number_of_Rule_Match == M ", where M is equal to time period t / sampling interval of the network monitor. The policy server will set appropriate trigger at the network monitor and will receive triggers corresponding to violations on each sampling interval. An appropriate policy action like port shutdown may then be performed when the conditions of the rule match. The time interval between two consecutive rule matches can also be tracked which can be used to reset the "Number_of_Rule_Match" variable, for example, if the time interval is greater than say T, reset "Number_of_Rule_Match" variable to zero.

6.2.2 Fail-Safe Mechanisms

The policy server provides deterministic behavior by evaluating policies according to the specified decision strategy and priority. But it is possible for a network administrator to enable incorrect policies by mistake or intentionally. The deployment of incorrect policies will result in misconfigurations at network devices, which may result in complete network shutdown or system collapse. This problem is different from policy conflict detection. For example, let's consider a pro-active management policy. Let's assume that the policy administrator has specified a policy action of setting "High Priority" to traffic from violating hosts by mistake, such a policy when deployed may further increase the impact of the violating hosts on other non-violating hosts. Also it may result in large number of network triggers, thereby increasing processing at both the network monitor and policy server which may eventually result in exhaustion of memory and CPU resources thereby resulting in a system collapse. Fail-Safe mechanisms must be developed to detect such policy incorrectness and used before policy deployment in order to avoid such system collapse.

6.2.3 Optimization of Policy Rule Evaluation

There is a need to optimize policy evaluation either at the network device level or at the policy server level. For example, if a network administrator specifies an access-list policy group with say two hundred policy rules, the performance of the network device may be (severely) impacted depending on resources available to it (e.g., every packet has to be processed against one or more policy rules and the device may run out of CPU power or memory). Optimization algorithms can be developed to reduce the number of rules to be evaluated, thereby improving performance.

6.2.4 Adaptive End-to-End QoS

Policy Server can be extended to evaluate policies that specify conditions involving QoS values of two or more flows (e.g. bandwidth, delay, jitter) i.e. evaluations can be based on relative values rather than absolute values. For example, the delay of one flow can be compared with the delay of another flow. Such policies can be used to provide Adaptive End-to-End QoS.

6.2.5 Satisfaction of Complex Service Level Agreements

The Policy server's ability to interact with network monitors and control network devices can be used to experiment and deploy new and complex service level agreements. For example let's consider a service level agreement that is used to provide Distributed Quality of Service to different customers in a multi service IP network. Typically the customer traffic enters the service provider network at a single ingress point. QoS for such a customer is provided by using traffic classification, adaptive marking, schedulers, and policers at the ingress in accordance to service level agreement established between the customer and service provider. The need for a new type of Service Level Agreement namely one that allows the customer traffic to enter the service provider network at any ingress point is currently being emphasized especially in campus networks (e.g. a core network being shared by number of universities). In this scenario, a centralized system is required to keep track of total bandwidth currently used by the customer traffic entering at any ingress point. There is also a need to fairly distribute available link bandwidth across flows belonging to different customers entering at the same ingress. Policy Server can be used as a tool to satisfy this complex SLA. The Policy Server can obtain the information about overall bandwidth consumption of a customer entering at N ingress points, by interacting with network monitor(s). Policies can be used to specify the amount of bandwidth allocated to a customer and actions to be taken when the bandwidth consumption is more as per SLA and type of QoS mechanisms to be used in order to provide the level of QoS agreed as per the SLA.

7 LIST OF REFERENCES

(1) International Organization for Standardization, "Packet based multimedia communications systems", ITU-T Recommendation H.323, http://www.packetizer.com/iptel/h323/

(2) International Organization for Standardization, "MPEG-1", ISO/IEC/JTC1/SC29/WG11 N MPEG June 1996, http://mpeg.telecomitalialab.com/standards/mpeg-1/mpeg-1.htm

(3) International Organization for Standardization, "MPEG-2", ISO/IEC/JTC1/SC29/WG11 N MPEG 00, October 2000, http://mpeg.telecomitalialab.com/standards/mpeg-2/mpeg-2.htm

(4) International Organization for Standardization, "MPEG-4 Overview", ISO/ITEC JTC1/SC29/WG11 N4668, March 2002, http://mpeg.telecomitalialab.com/standards/mpeg-4/mpeg-4.htm

(5) Dinesh Verma, *Supporting Service Level Agreements on IP Networks*, (Macmillan 1999)

(6) Levitin, D.J., Mathews, M.V., and MacLean, "The perception of cross-modal simultaneity", *Proc. of International Journal of Computing Anticipatory Systems*, Belgium, 1999. http://www.cs.ubc.ca/~maclean/publics/simultaneity0101.pdf

(7) M.R.Lyu, Y.S.Moon, W.K. Kan and M.A.Vouk, "Web-Based Education Techniques: Workflow, Collaboration, and Quality of Service", *Proc. 1998 Conference on Quality in teaching and Learning in Higher Education, Hong Kong*, December 10-12 1998, Page(s): 245-252

(8) P. Fasano, QoS for IP Telephony (Presentation at ISIT'99, Vancouver, June 10 1999)

(9) H.Schulzrinne, S.Casner, R.Frederick, V.Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996, http://www.ietf.org/rfc/rfc1889.txt

(10) A. Benyassine, E. Shlomot and H. Su, "ITU-T Recommendation G.729 Annex B: A Silence Compression Scheme for Use with G.729 Optimized for V.70 Digital Simultaneous Voice And Data Applications," *IEEE Communication Magazine*, Sept. 1997, Pages: 64–72.

(11) Cisco Systems Inc., "Voice over IP – Per Call Bandwidth Consumption", http://www.cisco.com/warp/public/788/pkt-voice-general/bwidth_consume.html

(12) S.Blake, D.Black, M.Carlson, E.Davis, Z.Wang, W.Weiss, "An Architecture for Differentiated Services," RFC 2475, December 1998. http://www.ietf.org/rfc/rfc2475.txt

(13) K. Nichols, S.Blake, F.Baker, D.Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," RFC 2474, December 1998. http://www.ietf.org/rfc/rfc2474.txt

(14) S.Shenker, C.Partridge, R.Guberin, "Specification of Guaranteed Quality of Service," IETF RFC 2212, September 1997 http://www.ietf.org/rfc/rfc2212.txt

(15) J.Wroclawski, "Specification of the Controlled-Load Network Element Service", IETF RFC 2211, September 1997 http://www.ietf.org/rfc/rfc2211.txt

(16) R.Braden, L.Zhang, S.Berson, S.Herzog, S.Jamin, "Resource Reservation Protocol (RSVP)", RFC 2205, September 1997, http://www.faqs.org/rfcs/rfc2205.html

(17) Haddad, Reda Nassif, *SLA To Controls Mapping in Differentiated Services*, MS Thesis, North Carolina State University, 2000

(18) Nicodemos C. Damianou, *A Policy Framework for Management of Distributed Systems*, PhD Dissertation, Imperial College of Science, Technology and Medicine, February 2002

(19) Gary N. Stone, Geoffrey G.xie, "Network Policy Languages: A survey and a New Approach," *IEEE Network*, Volume: 15, Issue: 1, Jan.-Feb.2001 Page(s): 10 -21

(20) Randeep Bhatia, Jorge Lobo, Madhur Kohli, "Policy Evaluation for Network Management," INFOCOM 2000, *Proc. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*. IEEE, Page(s) 1107-1116 Vol.3

(21) B.Moore, E.Elleson, J.Strassner, "Policy Core Information Model Version 1 Specification," RFC 3060, February 2001 http://www.ietf.org/rfc/rfc3060.txt

(22) B.Moore, L.Rafalow, Y.Ramberg, Y.Snir, A.Westerinen, R.Chadha, M.Brunner, R.Cohen, J.Strassner, "Policy Core Information Model Extensions," IETF, Work in Progress, draft-ietf-policy-pcim-ext-08.txt, May 2002, http://www.ietf.org/internet-drafts/draft-ietf-policy-pcim-ext-08.txt

(23) Zhi Fu, S.Felix Wu, He Huang, Kung Loh, Fengmin Gong, Illa Baldine, Chong Xu, "IPSec/VPN Security Policy: Correctness, Conflict Detection and Resolution", *International Workshop on Policies for Distributed Systems and Networks*, POLICY 2001 Bristol, UK, January 29-31, 2001.

(24) D.Durham, Ed. J.Boyle, R.Cohen, S.Herzog, R.Rajan, A.Sastry, "The COPS (Common Open Policy Service) Protocol," RFC 2748, January 2000, http://www.ietf.org/rfc/rfc2748.txt

(25) Y.Snir, Y.Ramberg, J.Strassner, R.Cohen, B.Moore, "Policy QoS Information Model," Work in Progress, draft-ietf-policy-qos-information-model-04.txt, November 2001,http://www.ietf.org/internet-drafts/draft-ietf-policy-qos-info-model-04.txt

(26) Dulay.N, Lupu.E, Sloman.M, Damianou.N, "A Policy Deployment model for the Ponder language," *Proc. IEEE/IFIP International Symposium on Integrated Network Management,* (IM'2001), Seattle, May 2001, IEEE Press, Page(s): 529-543

(27) Leonidas Lymberopoulos, Emil Lupu, Morris Sloman, "An adaptive Policy Based Management Framework for Differentiated Services Networks", *Proc. Third International Workshop on Policies for Distributed Systems and Networks*, June 2002 Page(s): 147-158

(28) Petri Aukia, Murali Lodialam, Pramod V.N.koppal, T.V.Lakshman, Helena Sarin and Bernhard Suter, Bell Laboratories, Lucent Technologies, "RATES: A Server for MPLS Traffic Engineering", *IEEE Network*, Volume: 14 Issue: 2, March-April 2000, Page(s): 34 –41

(29) W.Yeong, T.Howes, S.Kille, "Lightweight Directory Access Protocol," RFC 1777, March 1995

(30) Zao. J, Sanchez, L. Condell, M. Lynn, C. Fredette, M. Helinek, P. Krishnan, P. Jackson, A. Mankins, D. Shepard, M. Kent, "Domain Based Internet Security Policy Management", *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings*, Volume: 1,1999 Page(s): 41 -53 vol.1

(31) Distributed Management Task Force Inc, "Core Information Model," Version 2.5, http://www.dmtf.org/standards/published_documents.php

(32) Distributed Management Task Force Inc, "DMTF LDAP Schema for the CIM V2.5 Core Information Model," April 2002, http://www.dmtf.org/standards/published_documents.php

(33) Distributed Management Task Force Inc, "DMTF LDAP Schema for the CIM V2.6 Core Information Model," Work in Progress, http://www.dmtf.org/standards/published_documents.php

(34) J.Strassner, E.Ellesson, B.Moore, R.Moats, "Policy Core LDAP Schema," Work in Progress, draft-ietf-policy-core-schema-13.txt, November 2001, http://www.ietf.org/internet-drafts/draft-ietf-policy-core-schema-14.txt

(35) B.Moore, D.Durham, J.Strassner, A.Westerinen, W.Weiss, J.Halpern, "Information model for Describing Network Service QoS Data path Mechanisms, " Work in Progress", draft-ietf-policy-

qos-device-info-model-06.txt, November 2001, http://www.ietf.org/internet-drafts/draft-ietf-policy-qos-device-info-model-06.txt

(36) Atsushi Iwata, Norihito Fujita, "A Hierarchical Multilayer QoS Routing System with Dynamic SLA Management," *Selected Areas in Communications, IEEE Journal*, Volume: 18 Issue: 12, Dec. 2000, Page(s): 2603-2616

(37) Akio Moridera and Kazuo Murano, Yukou Mochida, Fujitsu Limited, "The Network paradigm of the 21st Century and Its Key Technologies," *IEEE Communications Magazine*, Volume: 38, Issue: 11 Nov. 2000, Page(s): 94-98

(38) Raju Rajan, Angelia Chiu and Seyham Civanlar, "A Policy Based Approach for QoS-On-Demand over the Internet," *Eighth International Workshop on Quality of Service, IWQOS 2000*, Page(s): 167-169

(39) Ashish Mehra, Dinesh Verma and Renu Tewari, IBM T.J.Watson Research Center, "Policy-Based Diffserv on Internet Servers: The AIX Approach," *IEEE Internet Computing*, Volume: 4 Issue: 5, Sept-Oct 2000, Page(s): 75-80

(40) Takao Ogura, Kenichi Fukuda, Kohei Iseda and Masato Okuda, Fujitsu Laboratories Ltd, "A Policy Server for an Access Network," *IEEE International Conference on Communications, ICC 2001*.Volume: 8, 2001 Page(s): 2404 -2409

(41) Yasusi Kanada, Central Research Laboratory, Hitachi Ltd, Japan, "A Representation of Network Node QoS Control Policies Using Rule-based building blocks ", *International workshop on Quality of Service, 2000. IWQOS 2000.* Pages: 161-163.

(42) Shepard, S.J., "Policy-based networks: hype and hope," *IT Professional*, Volume: 2 Issue: 1, Jan.-Feb. 2000, Page(s): 12-16

(43) Goff, Brian David, *Distributed Resource Monitoring Tool and its use in Security and Quality of Service Evaluation*, MS Thesis, North Carolina State University, March 2002

(44) Cisco Systems Inc., "Modular Quality of Service Command Line Interface", http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/120newft/120limit/120xe/120xe5/mqc/mcli.htm

(45) Cisco Systems Inc., “Cisco IOS Quality of Service Solutions Configuration Guide Release 12.2” http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122cgcr/fqos_c/index.htm

APPENDIX

BNF Grammar for IETF Policy Core Information Model

The BNF Grammar for the IETF PCIM classes is given below. In the current implementation, policies are specified using the graphical user interface of Policy Management Tool incompliance with this grammar. The following grammar is also used to validate policy values and properties of PCIM classes.

```
pcimPolicy          ::=  pcimRoles    pcimDecisionStrategy   (pcimPolicyRule | pcimPolicyGroup)

pcimPolicyGroup  ::= pcimGroupName     pcimPriority     *(pcimPolicyRule | pcimPolicyGroup)

pcimPolicyRule   :: = pcimRuleName
                      pcimRuleEnabled
                      pcimRuleConditionListType
                      pcimRuleMandatory
                      pcimPriority
                      pcimSequencedActions
                      pcimExecutionStrategy
                      pcimConditionAssociation
                      pcimActionAssociation
                     *(pcimPolicyRule)

pcimCompoundCondition ::= compoundConditionListType
                            *(pcimConditionAssociation)

pcimCompoundAction  ::=  pcimActionOrder
                          *(pcimActionAssociation)

pcimConditionAssociation :: = pcimGroupNumber
                               pcimConditionNegated
```

(pcimCondition | pcimCompoundCondition)

pcimActionAssociation ::= pcimActionOrder

(pcimAction | pcimCompoundAction)

pcimCondition :: = pcimConditionName

(pcimSimplePolicyCondition | pcimCompoundCondition)

pcimAction ::= pcimActionName

(pcimSimplePolicyAction | pcimVendorAction | pcimCompoundAction |

qpimQoSPolicyBandwidthAction | qpimQoSPolicyCongestionControlAction |

qpimQoSPolicyShapeAction | qpimQoSPolicyPoliceAction|

qpimQoSPolicyDiscardAction)

pcimSimplePolicyCondition ::= pcimVariable operator pcimValue

pcimSimplePolicyAction :: = pcimVariable pcimValue

pcimVendorAction ::= pcimActionData pcimActionString

qpimQoSPolicyBandwidthAction ::= qpForwardingPriority

qpBandwidthUnits

qpMinBandwidth

qpMaxBandwidth

qpMaxDelay

qpMaxJitter

qpFairness

qpimQoSPolicyCongestionControlAction ::= qpQueueSizeUnits

qpQueueSize

qpDropMethod

qpDropThresholdUnits

qpDropMinThresholdValue

qpDropMaxThresholdValue

qpimQoSPolicyPoliceAction ::= qpAdmissionScope

qpimQoSPolicyConformActionAssociation

qpimQoSPolicyExceedActionAssociation

qpimQoSPolicyViolateActionAssociation

qpimQoSPolicyShapeAction ::= qpAdmissionScope

qpimQoSPolicyTrfcProfAssociation

qpimQoSPolicyConformActionAssociation ::= qpimQoSPolicyPoliceAction

pcimAction

qpimQoSPolicyExceedActionAssociation ::= qpimQoSPolicyPoliceAction

pcimAction

qpimQoSPolicyViolateActionAssociation ::= qpimQoSPolicyPoliceAction

pcimAction

qpimQoSPolicyTrfcProfAssociation ::= qpimQoSPolicyTrfcProf

(qpimQoSPolicyPoliceAction | qpimQoSPolicyShapeAction)

qpimQoSPolicyTrfcProf ::= qpTBRate

qpTBNormalRate

qpTBExcessRate

policyVariable :: = policyVariableName

policyVariableType

policyVariableType ::= (pcimPolicySourceIPv4Variable | pcimPolicySourceIPv6Variable
pcimPolicyDestinationIPv4Variable | pcimPolicyDestinationIPv6Variable
pcimPolicySourcePortVariable | pcimPolicyDestinationPortVariable
pcimPolicyIPProtocolVariable | pcimPolicyIPVersionVariable
pcimPolicyIPToSVariable | pcimPolicyDSCPVariable
pcimPolicyFlowIdVariable | pcimPolicySourceMACVariable
pcimPolicyDestinationMACVariable | pcimPolicyVLANVariable
pcimPolicyCoSVariable | pcimPolicyEthertypeVariable
pcimPolicyFlowDirectionVariable | pcimPolicyBandwidthVariable
pcimPolicyNumberOfFlowsVariable | pcimPolicySwitchPortSpeedVariable
pcimPolicySwitchPortDuplexModeVariable | pcimPolicySwitchPortStatusVariable
pcimPolicyAccessTypeVariable | pcimPolicyIPPrecedenceVariable)

pcimValue ::= pcimValueName
(pcimPolicyIPv4AddrValue | pcimPolicyMACAddrValue |
pcimPolicyStringValue | pcimPolicyBitStringValue |
pcimPolicyIntegerValue | pcimPolicyBooleanValue)

pcimPolicyIPv4AddrValue ::= IPv4AddrList

pcimPolicyMACAddrValue ::= MACAddrList

pcimPolicyStringValue ::= StringList

pcimPolicyBitStringValue ::=BitStringList

pcimPolicyIntegerValue ::= IntegerList

pcimPolicyBooleanValue ::= BooleanValue

```
IPv4AddrList            ::= 1*(IPv4address | IPv4prefix | IPv4range | IPv4maskedaddress | Hostname)
IPv4address             ::= 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
IPv4prefix              ::= IPv4address "/" 1*2DIGIT
IPv4range               ::= IPv4address"-"IPv4address
IPv4maskedaddress       ::= IPv4address","IPv4address

MacAddrList             ::= 1*(MacAddress | MacmaskedAddress)
MACaddress              ::= 1*4HEXDIG ":" 1*4HEXDIG ":" 1*4HEXDIG
MACmaskedaddress        ::= MACaddress","MACaddress

StringList              ::= 1*(ALPHA)
BitStringList           ::= 1*(bitString | maskedString)
IntegerList             ::= 1*(integer | integerrange)
integer                 ::= [-] 1*DIGIT
integerrange            ::= integer".."integer

bitString               ::= 1*binary-digit
maskedBitString         ::= bitString","bitString
pcimActionOrder         ::= nonzero (1*(DIGIT))
pcimRuleEnabled         ::= (%x30 | %x31 | %x32)
                                ;0 – Administratively Enabled, 1 – Disabled, 2  - Enabled for Debug
pcimExecutionStrategy   ::= (%x30 | %x31 | %x32)
                                ;0 – Do until success, 1 – Do All , 2  - Do until Failure
pcimSequencedActions    ::= (%x30 | %x31 | %x32)
                                ; 1 -mandatory, 2 - recommended, 3 – don't Care
pcimGroupName           ::= (1*(ALPHA))
pcimRuleName            ::= (1*(ALPHA))
pcimPriority            ::= nonzero(1*(DIGIT))
```

```
pcimDecisionStrategy       ::= (%x31  | %x32)
                                    ;1 – First Matching, 2 – All Matching
pcimRuleConditionListType ::=  (%x31  | %x32)
                                    ;1 – DNF, 2 – CNF
pcimGroupNumber            ::= nonzero (1 *(DIGIT))
pcimConditionNegated       ::= TRUE | FALSE
operator                   ::=  (“>” | ”<” | “==” | ”!=”)
pcimRoles                  ::= roleName [&&roleName]*
qpForwardingPriority       ::=  nonzero (1*(DIGIT))
 qpBandwidthUnits          ::= (%x30 | %x31)
                                    ;0 – Kbps, 1 - Percentage
qpMinBandwidth             ::= nonzero (1*(DIGIT))
qpMaxBandwidth             ::= nonzero (1*(DIGIT))
qpMaxDelay                 ::= nonzero (1*(DIGIT))
qpMaxJitter                 ::= nonzero (1*(DIGIT))
 qpFairness                ::= (TRUE / FALSE)
qpQueueSizeUnits           ::= (%x30 | %x31)
                                    ;0 – Packets, 1- Milliseconds
qpQueueSize                ::= nonzero (1 *(DIGIT))
qpDropMethod               ::= (%x30 | %x31 | %x32)
                                    ;0 – Random Detect, 1 = Tail Drop, 2 = Head Drop
 qpDropThresholdUnits      ::= (%x30 | %x31 | %x32)
                                          ;0 – Packets, 1 – Percentage, 2 -Milliseconds
qpDropMinThresholdValue          ::= nonzero (1 *(DIGIT))
qpDropMaxThresholdValue          ::= nonzero (1 *(DIGIT))

roleName                         ::=1*(ALPHA)
```

nonzero ::= %x31-39

; 1-9

binary-digit ::= "0" / "1"

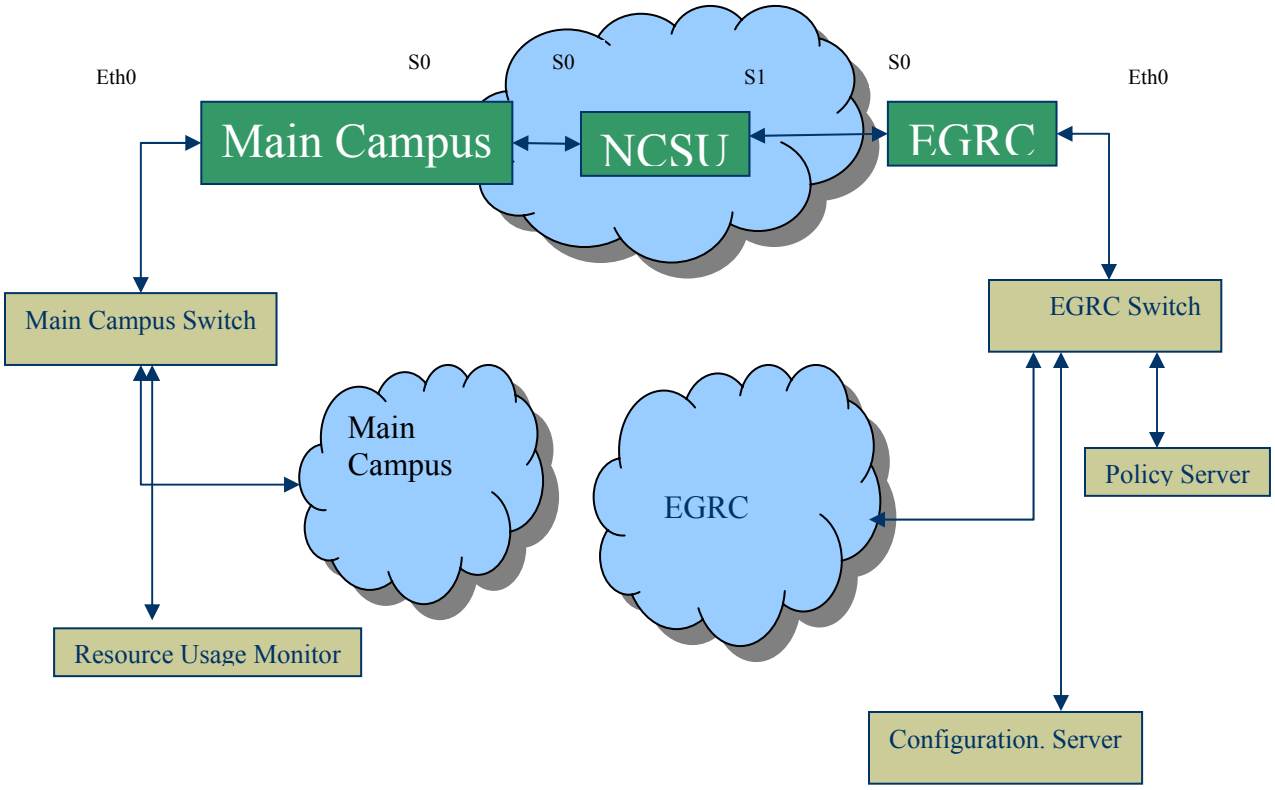
HEXDIG ::=DIGIT / “A” / “B” / “C” / “D” / “E” /”F”

DIGIT ::= %x30 –39 ;0-9

ALPHA ::= %x41-5A / %x61-7A

;A-Z /a-z

Policy Test Bed Documentation



Network Interfaces	Roles
Ethernet0@Main-Campus-Router	Ethernet, Main_Campus, Main-Campus&&Network_Egress
Serial0@Main-Campus-Router	Serial, Main_Campus, Main_Campus&&Core_Network_Ingress
Serial0@NCSU-Router	Serial, NCSU, Core_Network&&Serial
Serial1@NCSU-Router	Serial, NCSU, Core_Network&&Serial
Vlan1@Main-Campus-Switch	
FastEthernet0/1@Main-Campus-Switch	FastEthernet, Main_Campus
FastEthernet0/2@Main-Campus-Switch	FastEthernet, Main_Campus
FastEthernet0/3@Main-Campus-Switch	FastEthernet, Main_Campus
FastEthernet0/4@Main-Campus-Switch	FastEthernet, Main_Campus
FastEthernet0/4@Main-Campus-Switch	FastEthernet, Main_Campus
Serial0@EGRC-Router	Serial, EGRC, EGRC&&Core_Network_Ingress
Ethernet0@EGRC-Router	Ethernet, EGRC, EGRC&&Network_Egress

Network Elements	Roles
Main-Campus-Router	Main_Campus
Main-Campus-Switch	Main_Campus
NCSU-Router	NCSU
EGRC-Router	EGRC
EGRC-Campus-Switch	EGRC

Reusable Policy Variables	Variable Type
DSCP	PcimPolicyIPDSCPVariable
IPPrecedence	PcimPolicyIPPrecedenceVariable
SOURCE_IPv4ADDRESS	PcimPolicySourceIPv4Variable
SOURCE_PORT	PcimPolicySourcePortVariable
IP_Protocol	PcimPolicyIPProtocolVariable
BANDWIDTH	PcimPolicyBandwidthVariable
NUMBER_0F_FLOWS	PcimPolicyNumberOfFlowsVariable
ACCESS_TYPE	PcimPolicyAccessTypeVariable
FLOW_DIRECTION	PcimPolicyFlowDirectionVariable

Reusable Policy Value Name	Policy Value
HTTP_PORT	80
FTP_PORT	21
TELNET_PORT	23
KAZZA_PORT	1214
NAPSTER_PORT	5555
EDONKEY_PORT	4661
KAZAA_VIOLATING_IPHOST	IPv4maskedAddress=152.1.4.11,255.255.255.0
NAPSTER_VIOLATING_IPHOST	IPv4maskedAddress=152.1.4.12,255.255.255.0
EDONKEY_VIOLATING_IPHOST	IPv4maskedAddress=152.1.4.14,255.255.255.0

RTP_PORT	5004
NORMAL_LOAD	10000
SIP_PORT	5060
VIDEO_STREAM_PORT	7007
HIGH_LOAD	20000
VERY_HIGH_LOAD	30000
TCP_PROTOCOL	6
UDP_PROTOCOL	17
PORT_ON	TRUE
PORT_OFF	FALSE
MAX_NO_OF_FLOWS	300
AVG_NO_OF_FLOWS	200
EXPEDITED_FORWARDING_PRIORITY	46
CLASS3_LOW_DROP_PRIORITY, AF31	26
CLASS4_LOW_DROP_PRIORITY, AF41	34
FLASH_PRECEDENCE	3
FLASH_OVERRIDE_PRECEDENCE	4
BEST_EFFORT	0
CRITICAL_PRECEDENCE	5
CSC579IPADDRESS	IPv4maskedAddress=152.1.3.15,255.255.255.0
PERMIT_ACCESS	1
DENY_ACCESS	0
INCOMING	1
OUTGOING	0

Policy Action Name	**Policy Action Information**
Permit-Access-Action	Set ACCESS_TYPE = PERMIT_ACCESS
Deny-Access-Action	Set ACCESS_TYPE = DENY_ACCESS
VOIP-DSCP-Mark-Action	Set DSCP = EXPEDITED_FORWARDING_PRIORITY
VOIP-PREC-Mark-Action	Set IPPrecedence =CRITICAL_PRECEDENCE
VOIP-CONTROL-DSCP-Mark-Action	Set DSCP = CLASS3_LOW_DROP_PRIORITY
VOIP-CONTROL-PREC-Mark-Action	SET IPPrecedence=FLASH_PRECEDENCE
VOIP_BANDWIDTH_ACTION	QoSPolicyBandwidthAction qpForwardingPriority=1 qpBandwidthUnits=0 qpMaxBandwidth = 64
VOIP_VIDEO_CONTROL_BANDWIDTH_ACTION	QoSPolicyBandwidthAction qpForwardingPriority=0 qpBandwidthUnits=0 qpMaxBandwidth = 16
VOIP_VIDEO_CONTROL_CC_ACTION	QoSPolicyCongestionControlAction qpQueueSizeUnits=0 qpQueueSize=20 qpDropMethod=1
VIDE0_BANDWIDTH_ACTION	QoSPolicyBandwidthAction qpForwardingPriority=0 qpBandwidthUnits=0 qpMaxBandwidth = 386

VIDEO_RANDOM_DETECT_CC_ACTION	QoSPolicyCongestionControlAction qpQueueSizeUnits=0 qpQueueSize=75 qpDropMethod=0 qpDropThresholdUnits=0 qpDropMinThresholdValue=20 qpDropMaxThresholdValue=70
BEST-EFFORT-DSCP-Mark-Action	Set DSCP=BEST_EFFORT
BEST-EFFORT-PREC-Mark-Action	Set IPPrecedence=BEST_EFFORT
Port_Shutdown_Action	Set PORT_STATUS=PORT_OFF

Policy Group	**Policy Rules**
Access-Policy-Group **(Role Combination**= Main_Campus&&Network_Egress)	HTTP_ACCESS_RULE FTP_ACCESS_RULE TELNET_ACCESS_RULE KAZAA_ACCESS_RULE NAPSTER_ACCESS_RULE EDONKEY_ACCESS_RULE DEFAULT_ACCESS_RULE
Edge-QoS-Policy-Group **(Role combination:** Main_Campus&&Network_Egress)	VOIP_RTP_MARK_RULE VOIP_SIP_MARK_RULE
Core-QoS-Policy-Group (**Role combination**: Core_Network&&Serial Main_Campus&&Core_Network_Ingress EGRC_Campus&&Core_Network_Ingress)	VOIP_BANDWIDTH_RULE VOIP_CONTROL_CC_RULE VIDEO_CC_RULE
Bandwidth-Violation-Policy-Group (**Role**: Main_Campus)	BANDWIDTH_VIOLATION_RULE BANDWIDTH_EXCEED_RULE DEFAULT_BANDWIDTH_RULE
VoIP-Conference-QoS-Policy-Group (**Role:** Main_Campus&&Network_Egress)	VOIP_CONFERENCE_MARK_RULE DEFAULT_MARK_RULE
Video-Service-Policy-Group (**Role:** Main_Campus&&Network_Egress)	DEFAULT_VIDEO_SERVICE_MARK_RULE

Sample Network Device Configuration Outputs

Note: No traffic was generated at the time of taking the configuration snapshot. Hence you will not see any statistics in the configuration output.

1 Policy Group: Access-Policy-Group

In this example, both the IP address of the host and application port was specified.

Configuration at Serial0 interface of Main_Campus Router

Extended IP access list 101
permit tcp any eq www any
permit tcp any eq ftp any
permit tcp any eq telnet any
deny tcp host 152.1.4.11 eq 1214 any
deny tcp host 152.1.4.12 eq 5555 any
deny tcp host 152.1.4.14 eq 4661 any
permit udp any any
permit tcp any any

Configuration at serial0 interface after removal of KAZAA_ACCESS_RULE

Extended IP access list 102
permit tcp any eq www any
permit tcp any eq ftp any
permit tcp any eq telnet any
deny tcp host 152.1.4.12 eq 5555 any
deny tcp host 152.1.4.14 eq 4661 any
permit udp any any
permit tcp any any

2 Policy Group: Edge-QoS-Policy-Group

Configuration at Ethernet0 interface of Main_Campus Router

mainCampus#sh policy-map int eth0
Ethernet0

Service-policy input: Ethernet0_IN_Policy1028676384166

Class-map: class_1028676384166 (match-any)
0 packets, 0 bytes
5 minute offered rate 0 bps, drop rate 0 bps
Match: access-group 135
0 packets, 0 bytes
5 minute rate 0 bps
police:
8000 bps, 8000 limit, 8000 extended limit
conformed 0 packets, 0 bytes; action: **set-dscp-transmit 46**
exceeded 0 packets, 0 bytes; action: set-dscp-transmit 46
violated 0 packets, 0 bytes; action: set-dscp-transmit 46
conformed 0 bps, exceed 0 bps violate 0 bps

Class-map: class_1028676404515 (match-any)
0 packets, 0 bytes
5 minute offered rate 0 bps, drop rate 0 bps
Match: access-group 136
0 packets, 0 bytes
5 minute rate 0 bps
police:
8000 bps, 8000 limit, 8000 extended limit

conformed 0 packets, 0 bytes; action: **set-dscp-transmit 26**
exceeded 0 packets, 0 bytes; action: set-dscp-transmit 26
violated 0 packets, 0 bytes; action: set-dscp-transmit 26
conformed 0 bps, exceed 0 bps violate 0 bps

Class-map: class-default (match-any)
1 packets, 60 bytes
5 minute offered rate 0 bps, drop rate 0 bps
Match: any
mainCampus#sh access-list 135
Extended IP access list 135
permit udp any eq 5004 any
mainCampus#sh access-list 136
Extended IP access list 136
permit udp any eq 5060 any
mainCampus#

3 Policy-Group: Core-QoS-Policy-Group

Configuration at Serial0 and Serial1 interface of NCSU router
This is an example in which the same policy is applied to multiple interfaces.

ncsu#sh policy-map int serial0
Serial0

Service-policy output: Serial0Serial1_OUT_Policy1028019396336

Class-map: class_1028019396346 (match-any)
0 packets, 0 bytes
5 minute offered rate 0 bps, drop rate 0 bps
Match: ip dscp 46
0 packets, 0 bytes
5 minute rate 0 bps
Match: ip precedence 5
0 packets, 0 bytes
5 minute rate 0 bps
Weighted Fair Queueing
Strict Priority
Output Queue: Conversation 264
Bandwidth 64 (kbps) Burst 48000 (Bytes)
(pkts matched/bytes matched) 0/0
(total drops/bytes drops) 0/0

Class-map: class_1028019399150 (match-any)
0 packets, 0 bytes
5 minute offered rate 0 bps, drop rate 0 bps
Match: ip precedence 3
0 packets, 0 bytes
5 minute rate 0 bps
Match: ip dscp 26
0 packets, 0 bytes
5 minute rate 0 bps
Weighted Fair Queueing
Output Queue: Conversation 265
Bandwidth 16 (kbps) Max Threshold 20 (packets)

(pkts matched/bytes matched) 0/0
(depth/total drops/no-buffer drops) 0/0/0

Class-map: class_1028019402715 (match-any)
0 packets, 0 bytes
5 minute offered rate 0 bps, drop rate 0 bps
Match: ip dscp 34
0 packets, 0 bytes
5 minute rate 0 bps
Match: ip precedence 4
0 packets, 0 bytes
5 minute rate 0 bps
Weighted Fair Queueing
Output Queue: Conversation 266
Bandwidth 386 (kbps)
(pkts matched/bytes matched) 0/0
(depth/total drops/no-buffer drops) 0/0/0
exponential weight: 9
mean queue depth: 0

ed	Transmitted pkts/bytes	Random drop pkts/bytes	Tail drop pkts/bytes	Minimum thresh	Maximum thresh	Mark prob
af11	0/0	0/0	0/0	32	40	1/10
af12	0/0	0/0	0/0	28	40	1/10
af13	0/0	0/0	0/0	24	40	1/10
af21	0/0	0/0	0/0	32	40	1/10
af22	0/0	0/0	0/0	28	40	1/10
af23	0/0	0/0	0/0	24	40	1/10
af31	0/0	0/0	0/0	32	40	1/10
af32	0/0	0/0	0/0	28	40	1/10
af33	0/0	0/0	0/0	24	40	1/10
af41	**0/0**	**0/0**	**0/0**	**20**	**70**	**1/10**
af42	0/0	0/0	0/0	28	40	1/10
af43	0/0	0/0	0/0	24	40	1/10
cs1	0/0	0/0	0/0	22	40	1/10
cs2	0/0	0/0	0/0	24	40	1/10
cs3	0/0	0/0	0/0	26	40	1/10
cs4	0/0	0/0	0/0	28	40	1/10
cs5	0/0	0/0	0/0	30	40	1/10
cs6	0/0	0/0	0/0	32	40	1/10
cs7	0/0	0/0	0/0	34	40	1/10
ef	0/0	0/0	0/0	36	40	1/10
4	**0/0**	**0/0**	**0/0**	**20**	**70**	**1/10**
rsvp	0/0	0/0	0/0	36	40	1/10
default	0/0	0/0	0/0	20	40	1/10

Class-map: class-default (match-any)
91 packets, 4468 bytes
5 minute offered rate 0 bps, drop rate 0 bps
Match: any
ncsu#