ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/3953941

Policy Based QoS Management System for the IntServ/DiffServ Based Internet

Conference Paper · February 2002

DOI: 10.1109/POLICY.2002.1011303 · Source: IEEE Xplore

CITATIONS
46

READS
84

4 authors, including:

Appan Ponnappan
SemOpSys
1 PUBLICATION **46** CITATIONS

SEE PROFILE

Radhakrishna Pillai
Indian Institute of Management, Kozhikode
46 PUBLICATIONS **156** CITATIONS

SEE PROFILE

All in-text references underlined in blue are linked to publications on ResearchGate, letting you access and read them immediately.

Available from: Appan Ponnappan
Retrieved on: 18 September 2016

SAMSUNG 1049

A Policy Based QoS Management System for the IntServ/DiffServ Based Internet

Appan Ponnappan, Lingjia Yang, Radhakrishna Pillai.R,
Kent Ridge Digital Labs, 21 Heng Mui Keng Terrace, Singapore – 119613.
e-mail: {appan, lingjia, pillai }@krdl.org.sg
Peter Braun
Siemens AG, PG U SE D4, Hofmannstr.51, D-81359 Munich, Germany.
e-mail: peter-m.braun@icn.siemens.de

Abstract

Policy based management is being adopted widely for different domains like QoS, security and VPNs in order to allow new services and facilitate network management. Also, the interface to the network device and the information models required for specifying policies are either standardized or being standardized in IETF and DMTF. This paper reports the design, implementation, and performance evaluation of a policy based QoS management system for the IntServ/DiffServ based internet which is based on COPS for interfacing with the network device and on LDAP directory for storing policies. The design is based on distributed components and CORBA is used as a middleware for component interaction. The Diffserv policies are installed based on role combination assigned to the network device interfaces and policy caching is used to improve the performance. The feasibility and performance of managing and deploying IntServ and Diffserv policies for QoS management is demonstrated using Linux-based routers. The preliminary performance measurements show that the policy server response time is around 90ms for 25 COPS-RSVP requests per second. COPS-PR processing takes more time compared to COPS-RSVP requests. It is found that the directory access could become a bottleneck in scaling the performance of the policy server and it can be improved substantially by employing appropriate policy caching mechanisms.

1. Introduction

With the increasing complexity, heterogeneity and size of computer networks, their management is also getting more complicated, error-prone and costly. Also, it is important to achieve business objective-controlled usage of network resources. These are the main drivers for an automated and consistent management of these networks. Policy Based Network Management (PBNM) is an over-arching technology for an automated management of networks. Policies encode the high-level goals and requirements of management.

Policy based management is being adopted widely for different domains like Quality of Service (QoS), security and Virtual Private Networks (VPNs). Also, the interface to the network device and the information models required for specifying policies are either standardized or being standardized in IETF and Distributed Management Task Force (DMTF).

Typical usage of policies include:

- Device configuration – queueing mechanisms, drop strategies, access filter lists, etc.
- Traffic classification/marking (e.g. DiffServ Marking)
- Resource reservation and admission control based on user identity, application type, etc (e.g. with RSVP).
- Service Level Agreement (SLA) between two adjacent domains

This paper reports the design, implementation, and performance evaluation of a policy based QoS management system for the IntServ/DiffServ based internet, developed in the QoS/PBN Project[1]. The design of the policy management system is based on the IETF/DMTF policy framework and supports DiffServ and IntServ policies. Common Open Policy Service (COPS) protocol is used for interfacing with network devices and Lightweight Directory Access Protocol (LDAP) [22] for storing policies in a directory. The system has a distributed, scalable architecture. CORBA is used as a middleware for component interaction. The Diffserv policies are installed based on role combination assigned to the network device interfaces and policy caching is used to improve the performance. The feasibility and performance of managing and deploying IntServ and Diffserv policies for QoS management for Linux-based routers is demonstrated.

[1] The QoS/PBN project is a joint R&D project between the Siemens ICM and the Kent Ridge Digital Labs, Singapore.

The paper is organised as follows: an overview of Internet QoS and policy based networking is given in Section 2. Section 3 describes the policy management system and the performance results are reported in Section 4. Conclusions are drawn in Section 5.

2. Overview of Internet QoS and Policy Based Networking

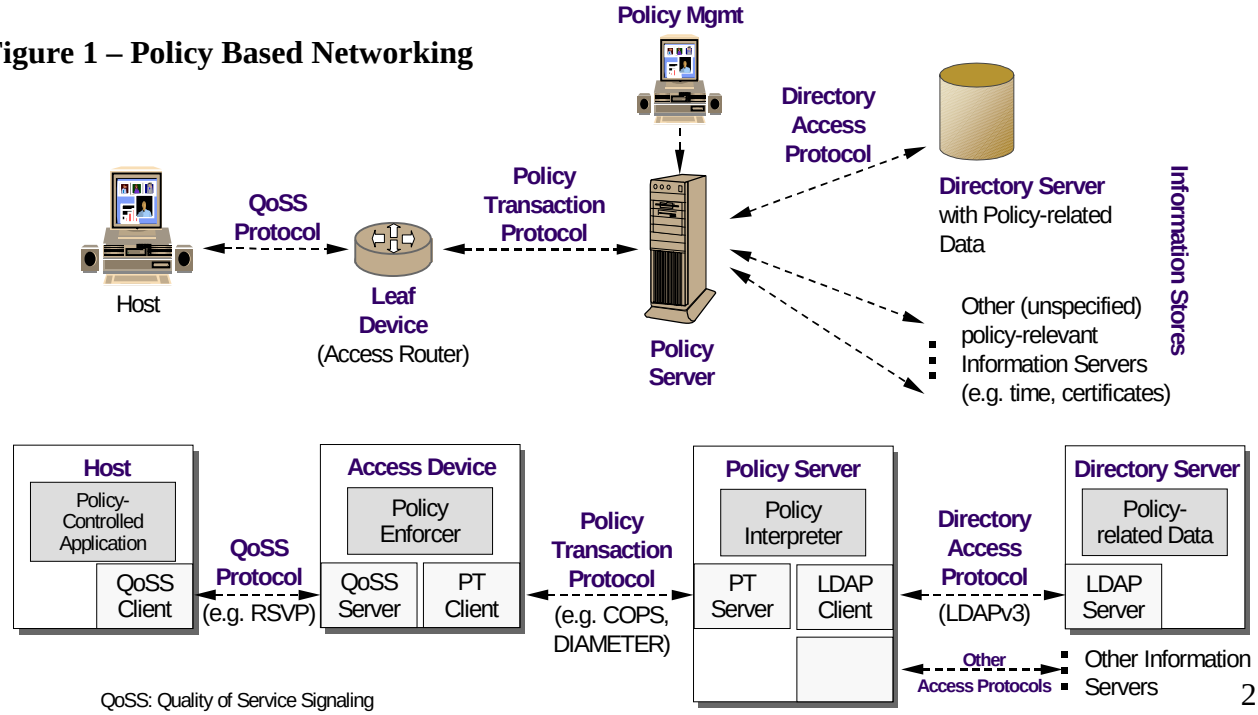
2.1 Internet QoS

Standard Internet Protocol (IP)-based networks provide "best effort" data delivery by default. As more hosts are connected, network service demands eventually exceed capacity, but service is not denied. Instead it degrades gracefully. Although the resulting variability in delivery delays (jitter) and packet loss do not adversely affect typical Internet applications such as e-mail, file transfer and Web applications, real-time applications cannot adapt to inconsistent service levels. The goal of QoS is to provide some level of predictability and control beyond the current IP "best-effort" service. There were a few models considered to provide QoS to IP networks; the Integrated Services (IntServ) and Differentiated Services (DiffServ) models standardized by IETF are accepted by most. The IntServ model includes two sorts of service targeted towards real-time traffic: guaranteed and predictive service. It integrates these services with controlled link sharing, and it is designed to work well with multicast as well as unicast. With Integrated Services, resources (e.g., bandwidth) are explicitly managed in order to meet application requirements. This implies that 'resource reservation' and 'admission control' are key building blocks. As the unit of resource management is the application data flow, per flow resource reservation management is needed in the network. ReSource reserVation Protocol (RSVP) is used as signaling protocol for explicit end-to-end resource reservations per application data flow. DiffServ have emerged in the Internet arena as a Class of Service approach to providing better than best effort QoS. Differentiated Services aim for a relatively simple, lightweight mechanism that does not depend entirely on explicit per-flow resource reservation in contrast to Integrated Services which use a more stringent and complex approach to QoS. With Differentiated Services there is no need to signal the parameters describing the desired type of service or for making resource reservations explicitly. Instead the parameters are programmed into routers, and a class of service is selected by subscription to a Service Level Agreement (SLA) or by marking IP-packets appropriately.
Within the QoS/PBN project, end-to-end QoS is provided by supporting flow based admission control at the border of the network using RSVP and DiffServ forwarding in the core.

2.2 Policy Based Networking

The architecture of a typical policy based network is shown in Figure 1. A policy is a rule that instructs a network node on how to manage requests for network resources. It is essentially a mechanism for encoding business objectives concerning the proper use of scarce resources. For example, policies define (on a per user/group/application/time-of-day basis) what resources a

Figure 1 – Policy Based Networking



given resource consumer can use in the context of a given scenario. The policy management system (or policy server) is also referred to as Policy Decision Point (PDP) or policy interpreter. It weighs the resource request caught by the trigger against a corresponding set of policy rules. As a response to a policy request, the PDP formulates a policy lease. The policy lease is a set of instructions for the policy requester how to react to the network resource request. The policy lease is then transported to a Policy Enforcement Point (PEP) using a policy transaction protocol such as Common Open Policy Service (COPS). The PEP is a network device where the policy lease is realised. Enforcement of policy decisions is carried out by the specific hardware/software features residing in the device such as packet filtering, bandwidth reservation, traffic prioritisation, multiple forwarding queues, etc. Actual policy enforcement requires detailed, low-level hardware/software know-how of network devices such as routers, switches. The policy management client, also referred to as the policy editor (PE) provides a high-level user interface for operator input, translates this input into the proper schema for storage in the directory server and pushes it out to the directory for storage. In policy based networks, a number of different back-end data stores (directory server, time server, certificate server.) provide replicated data throughout the network that serves as input for policy decisions by the PDP. Each policy server serves one network administrative domain. A policy server can communicate with the adjacent domain policy servers for proper establishment of the device configuration requirements in the adjacent domains.

There are two main models used in PBN for policy management, outsourcing and provisioning. With outsourcing model, when a PEP has to make a decision regarding an event, it outsources the decision-making to the PDP. In provisioning model, The PDP typically predicts future configuration needs, and proactively pre-provisions for them ahead of time. Rather than responding to PEP events, the PDP prepares and "pushes" configuration information to the PEP, as a result of an external non-PEP event, such as change of applicable policy, time of day, expiration of account quota, or a result of third party (non PEP) signaling. The provisioning mode is most commonly used for controlling network policy for non-signaled protocols, such as DiffServ, or configuring devices (such as VPNs and VoIP). The policy rules are authored using an editor in the policy management console and stored in the directory server for immediate or later enforcement.

Functionally, the PDP takes policy information that has been entered into the PBN management system and processes it, together with network resource information, to make policy decisions, and sends directives to the network devices. The policy data used by the PDP can either be obtained in real-time as it gets entered at the management console, or from the policy repository on need basis. In the case of an outsourcing policy model, the PDP receives policy requests from a network device, and decides whether or not to grant the request. For a QoS scenario this is typically an admission control decision regarding an RSVP request for resource reservation. Based on the business-level policies found in the repository, i.e. whether the requesting user/application has privilege to reserve the resources, the PDP either allows the RSVP message to enter the network, or rejects it. In the policy provisioning case, new policy is entered at the management console, and the policy rules are distributed in real-time to the PDP. The PDP decides whether the policy should be installed based on various criteria, such as whether any of the devices it controls should enforce the policy, whether time constraints apply at the given time, etc. If all of the criteria are satisfied, the PDP packages the policy rules as configuration commands that the device(s) can interpret, and sends the directives to the device(s). The PDP also handles feedback from the network. All decisions sent out by the PDP are accompanied by an acknowledgment to ensure they were properly installed, or to detect an error with any installation.

In both outsourcing and provisioning policy management models, PEPs receive policy decisions and enforce them at the packet level as data passes through the devices. In the outsourcing case, enforcement typically consists of allowing an RSVP signaled packet to pass through the network or not. With policy provisioning, enforcing a policy decision usually means classifying data packets as they enter the network, and processing them according to the policy rules found in the decision; i.e., identifying packets that originate from a particular subnet, and marking only those packets with a high priority precedence, enforces the policy that traffic from users on that subnet should be treated as mission-critical. The PEP also provides feedback to the PDP regarding the decisions installed at the PEP, i.e., if an error occurs while trying to install the policy, or if a detected failure of some sort has an effect on a previously installed policy decision.

The COPS[16] policy protocol is used to communicate policy information between PDPs and PEPs. This facilitates the exchange of dynamic policy rule information between a PDP and its associated PEPs. There is a built-in concept of client-types in COPS, which leaves room for adding a second layer of client-specific directive and expansions. COPS specifies functionality for both outsourcing (e.g., COPS-RSVP) and provisioning (COPS-PR) models of policy management.

The policy server described in the following section is deployed in a network testbed consisting of a core network based on DiffServ and multiple wired/wireless access networks based on IntServ with RSVP signalling. A single policy server serves the entire testbed. For every RSVP flow, only one IntServ router in each access network contacts the policy server. This is to minimise the amount of signalling at the policy server. The following open source implementation of different protocols/components are used in this work: COPS implementation from the Telia research [18], RSVP implementation from the Darmstadt University [19], and OpenLDAP [20].

2.3 Related Literature

Extensive research work has been carried out on the area of PBNM over the last decade, especially during the past few years. The need of PBNM, depicts the network paradigm of the 21st century, the existing products, standards, the options and the challenges we have ahead are illustrated in [23,24,25]. References [26, 27] explore the area of harmonic coexistence of policies and traditional, possibly hierarchical, management systems and a generic framework has been presented to show how a system for IP DiffServ management can be built using such a framework. An algorithm that evaluates policy rules specified using Policy Description Language is presented in [28]. Reference [29] proposes a new language, Path-Based Policy Language, which offers improvements to current network policy languages. Reference [30] describes techniques for accurately translating from global policy rules to actual per-device configurations. It also shows how these techniques were used in the implementation of Cisco Secure Policy Manager. Reference [31] presents an implementation of policy-based architecture on IBM's AIX operating system that provides DiffServ service to both QoS aware and – unaware applications.

One of the areas that little research has been carried out is on demonstrating the feasibility of managing different types of policies and understanding the performance of PBNM. This paper describes a policy management system for IntServ and DiffServ, demonstrates the feasibility of managing different types of policies, and addresses the scalability issues through preliminary measurements.

3. The Policy Management Architecture

Figure 2 illustrates the policy server (shown as PS in the diagram) and the external entities that exchange information with the policy server. The policy server is responsible for interpreting higher-level policies and translating them into device specific commands for realizing those policies. For allocating resources on inter-domain links and for implementing SLAs, the policy server (especially the bandwidth broker component) has to communicate with the policy server in the provider domain. The policy server is mainly responsible for the following:

- retrieval of relevant policies created by the network administrator through the policy console after resolving any conflicts with existing policies;
- translating the policies relevant for each of the PEP into the corresponding Policy Information Base (PIB) commands;
- arriving at policies decisions from relevant policies for RSVP policy decision requests and maintaining those decision states;
- taking appropriate actions – either deletion of existing RSVP decision states or modification of installed traffic control parameters in the PEP – for any modifications to currently installed policies.

There can be one policy server for each administrative domain. For example, policy server (adjacent domain) refers to the policy server in a DiffServ domain (backbone network) which is providing QoS based data transport service to the IntServ domain. In this case, the IntServ domain is the user of the DiffServ domain. The Policy server in the user domain has to have Service Level Agreements (SLAs) with the policy server in the provider domain. The policy server in the user domain has to communicate with the AAA server in cases where the user authentication, authorization and access control is done by an external AAA server.
CORBA is used as the middleware for the different components to communicate with each other. Policy Editor (PE) is the entity responsible for creating, modifying or deleting policy rules or entries in LDAP server. LDAP protocol provides access to directories supporting the X.500 models, while not incurring the resource requirements of the X.500 Directory Access Protocol (DAP). It is specifically targeted at management applications and browser applications that provide read/write interactive access to directories. It does not have the mechanism for notifying policy consumers of changes in the LDAP server. Therefore, it is the responsibility of the policy editor to indicate the changes in the LDAP server using an internal event messaging service.

Policy Editor uses the interface provided by the Policy Repository Adapter for creating/modifying/deleting a policy rule and notify the changes to the policy server.

The user has to assign priority to a new policy rule if it is in conflict with any other existing rule.

Policy repository adapter acts as a gateway to the policy repository. It encapsulates the specifics of representing policies in the policy repository and hence isolates the clients of the policy repository which are policy editor and the policy server. It implements the QoS policy schema as defined in references [2] and [4]. It can also implement a policy consistency checker functionality since LDAP does not provide this. The Policy Consistency Checker (PCC) maintains the referential integrity and also detects conflicts in policy rule actions which satisfy the same condition. It is more logical to have this entity centralized since the conflict resolution has to be done on policies which belong to the same group and which has been edited by other policy servers. This functionality has not yet been implemented and it is left to the policy

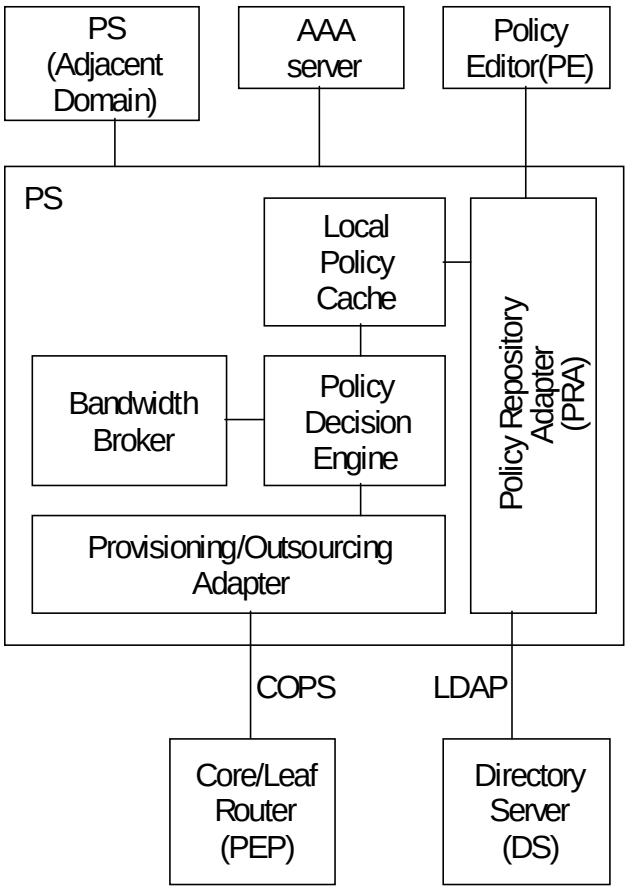


Figure 2: Policy Management System Architecture

administrator to resolve possible conflicts between policies by assigning proper priorities.

AAA Server is an external access server which could provide the user and the service requirement information to the policy server after checking whether the user is a genuine user and whether the user is authorized to use the service requested. The functionality of AAA server is required for end-hosts which can not do RSVP signaling and which is typically the scenario for a mobile end-host. Communication with the external provider domain policy server is required for implementing SLAs.

Policy repository adapter is queried only when the policy rules necessary for processing a request is not available in cache. Local policy cache replacement algorithm takes care of maintaining the full set of policy rules corresponding to a particular policy decision request. Policy decision engine processes decision requests from the provisioning/outsourcing adapter by retrieving the relevant policy rules for each request. In case of provisioning request, it converts them to the device configuration information and each policy rule is mapped to one COPS DEC message and sent to the PEP. In case of outsourcing requests, the decision is evaluated after querying the bandwidth broker for admission control and also the AAA server for authentication and authorization and the positive/negative decision is sent in one COPS DEC message to the PEP.
The bandwidth broker maintains the bandwidth usage information for the links connecting the border router to the adjacent domain. It provides the admission control function for new flows entering the domain.

3.1 COPS Related Policy Server Modules

The Outsourcing Adapter is responsible for handling outsourcing requests from the RSVP daemon in the PEP. It interprets the RSVP objects sent by the PEP. After decoding the RSVP objects received in the RSVP policy decision trigger message (either PATH or RESV message), it delegates the decision making responsibility to the policy decision engine with the received information. It conveys back the decision taken by the PDP, with any overridden reservation parameters, in the COPS-RSVP format required by the PEP.
This is implemented based on:

1. RFC 2749 for encoding/decoding the COPS-RSVP messages.
2. RFC 2750 for encoding/decoding POLICY-DATA RSVP object
3. RFC 2751 for encoding/decoding the policy pre-emption priority POLICY-DATA object
4. IETF draft[21] for encoding/decoding the user identity information

The PEP is responsible for caching the policy decision for the flow with the other flow information so that it can

avoid sending policy decision requests to the policy server for each of the RSVP refresh messages, only when there is no change in the currently reserved resources for that particular flow.

The Provisioning adapter implements policy provisioning mode as defined in the COPS extension for policy provisioning in IETF draft [12] and the DiffServ Policy Information Base (PIB) as defined in [9]. A PIB describes the data structures that are exchanged between the PEP and the PDP in a configuration request. The structure is specified by so-called Provisioning Classes (PRCs). The architecture of the Provisioning Adapter is flexible enough to support other configuration models for devices (e.g. snmp, telnet) if COPS PR is not supported by a device.

The open source implementation of COPS is based on an older version of this draft[10] and hence some changes have been made to this code:

1. Incorporation of marker table in both the PEP and the PDP library code;
2. Work-around for using ifIndex, which is an integer to represent roles.

3.2 Policy Schema and Example Policies

The PBNM architecture assumes that multiple policy systems may need to interoperate within a single domain, and share the same policy information. A central repository can be used to store, distribute, and coordinate policy information among such systems. LDAP Protocol is the industry choice for interoperable standard policy storage.

A multi-organizational effort involving the DMTF, the IETF, and others, has been defining both an information model and the LDAP schemas to represent standard policy information. The policy information is represented using policy rules. Each policy rule consists of a 'conditions' part and 'actions' part. The 'conditions' part specifies the conditions under which the actions have to be executed. The 'conditions' part is made up of one or more simple conditions combined by logical operators and each simple condition contains a variable name and a value related by a relational operator. In our implementation, the classes required for defining the schema are either reused or extended from the IETF policy drafts [1,2,3,4,5].

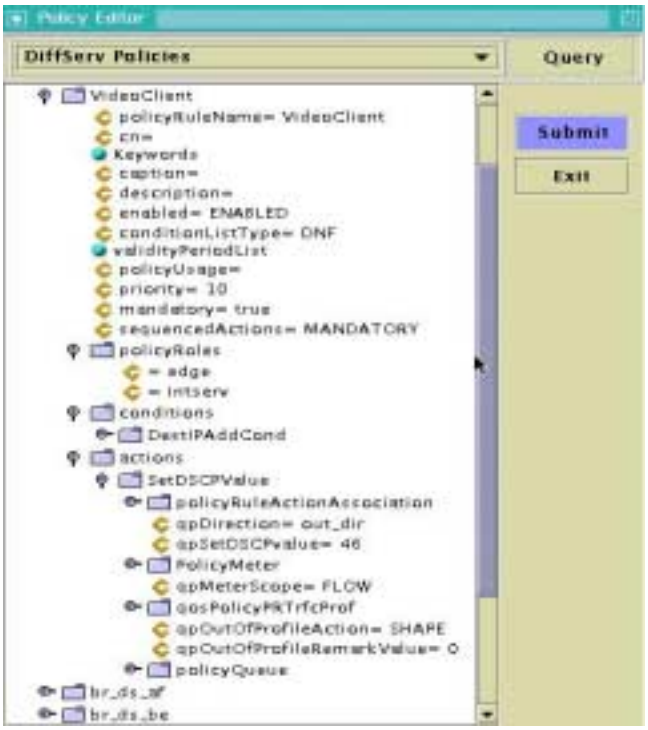


Figure 3: Policy Editor and a Policy Rule example

Figure 3 shows the Policy Editor developed in the system, and an example of a policy rule. Examples of other policies used for the test-bed are:

1. Allow/disallow a particular user depending on application and requested BW
2. Video server load balancing policies depending on the current load on the video server.
3. Maximum allowable bandwidth for each DiffServ code point.
4. Traffic conditioning to be applied for each of the DiffServ code point.
5. DiffServ code point to be used for a particular flow identified using a 5-tuple filter. This can be used for applications which does not support RSVP.
6. The discard policy that is to be adopted for non-conformant flows.
7. Create a marker with traffic specification for a particular DSCP code-point in the ingress link, if role of the link is "DiffServ-edge".

The LDAP directory entries have to be stored in a tree structure called the Directory Information Tree (DIT). The main design goals for a particular grouping of policies in the repository are:

1. Easier location of policies that are specific to a particular policy consumer.

2. Retrieval of policies in an efficient manner using very few LDAP calls as possible.

The LDAP classes that are used for representing QoS policies in the LDAP server are taken from the policy core schema[2] and QoS LDAP schema[4]. The static instances of the DIT are created once, at startup of the policy repository adapter. The policy repository adapter checks for these container classes before creating new policy rules. All the IntServ policies are grouped under a single qosNamedPolicyDomain. This is used as the root Distinguished Name (DN) for retrieving the IntServ related policies for an RSVP request. Similarly, all the DiffServ related policies are grouped under another qosNamedPolicyDomain instance. This is used as the root DN (with the role combination received from the PEP as the search criteria) for retrieving the DiffServ policies for the required role combinations. All the reusable policy conditions and actions are stored under the reusable qosPolicyRepository.

The concept of role is central to the design of entire policy framework. A role defines a particular function of an entity or component that can be used to identify particular behaviour associated with that entity or component. The idea behind roles is to identify the key functions (or roles) played by the network elements or interfaces in the network and assign policies to roles rather than directly to specific network element. The network elements have to report the roles at startup time (COPS-PR has a provision for this). The policy framework is then responsible for configuring each of the resources associated with a role in such a way that it behaves according to the policies specified for that role. More than one role can be assigned to an interface, called a role combination. In this case, it is the responsibility of the policy server to expand a single role combination into a list of role combinations and apply policies specific to each of the role combination [1]. Also, roles can simplify conflict detection. If a certain set of roles are mutually exclusive, then there is no need to do conflict detection between policies which uses these roles separately.

Roles are used in DiffServ for identifying the policies for a device. In Intserv, we use userId and application ID as conditions for the policy rule.

3.3 Component Interaction Scenarios

When the RSVP daemon receives the PATH or the RESV message for resource reservation, it sends a solicited decision request to the PS. The PS retrieves the relevant policy rules for the request using the user ID and/or the application ID sent in the request. Depending on the outcome of the evaluation of the policy rules, it sends back a decision message to either accept or deny the RSVP flow (See figure 4).

When the PEP starts up, it reports the physical interfaces and the roles that are assigned to those interfaces via the COPS REQ message (as shown in figure 5). The PDP retrieves the policy rules for each of the different role combinations that are reported by the PEP. The PRA retrieves all the rules for the requests role combination and sends it to the PDP as part of single result. PDP then converts them to device specific configuration information in the process of evaluating the rules. In our case, the device specific configuration information are the PRovisioning Instances (PRI) of the PRCs of the Diffserv PIB. Then PDP installs each of the rules that applies for the reported role combinations as separate COPS DEC message, as shown in Figure 5.

4. Experimental Results

4.1 Test Scenarios

Experiments were carried out to understand the performance of the policy management system. The response time of the policy server were measured at various loads. In order to facilitate policy decision requests at higher rates, test clients were used to generate COPS-PR and COPS-RSVP decision requests. Since the focus on the policy server response time, a dummy traffic control daemon is used for installing the policy rules at the PEP.

The RSVP test client emulates an RSVP signalling daemon in the PEP. It sends COPS REQ messages at the start of each session, at a given *session generation rate* according to a Poisson distribution. It also sends COPS DRQ message at the end of an RSVP session with mean *session hold-time*. This is fixed as 50 ms for all tests. Using this, the time elapsed between sending COPS REQ or DRQ message and getting back the COPS DEC, is measured. Two IntServ policy rules with different user names are stored in the repository. The experiments were conducted with policy rule caching enabled in the policy server. Sample result is given for the case when policy caching is disabled. The experiments are performed by varying the request rates.

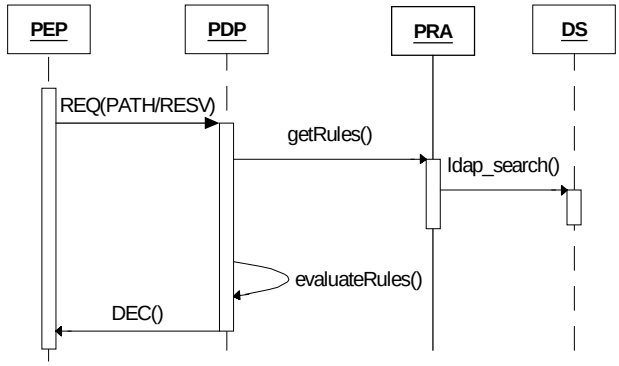


Figure 4: COPS-RSVP Outsourcing Request

The COPS-PEP test client issues COPS-PR requests with constant delay between successive requests. The response time is measured as the time elapsed between sending COPS REQ and getting back the DEC(1) message (as shown in Figure 5).

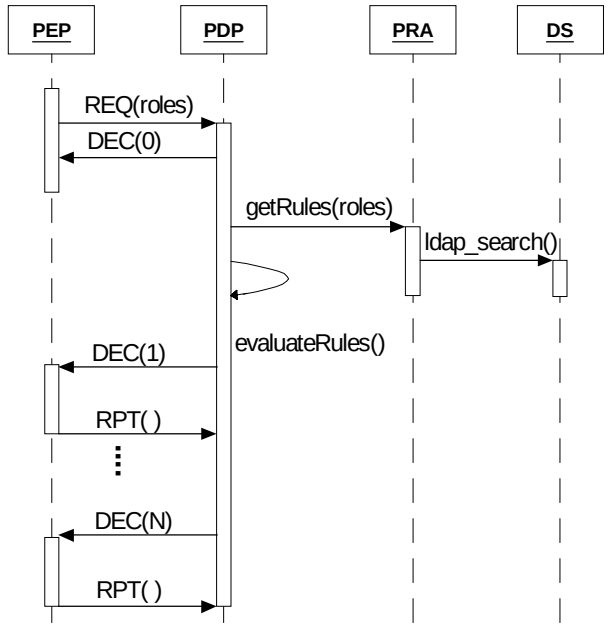


Figure 5: COPS Provisioning Scenario

The experiments were first conducted with policy rule caching in the policy server and then repeated by disabling the policy rule caching. The experiments are performed by varying the request rates.

4.2 Performance Results and Discussions

In our tests, the components of the policy server were running on a Intel Pentium III PC running at 800 MHz and the clients were running on separate machines.

Figure 6 shows the test results for COPS-PR requests. The number of policy rules processed for each client request is 9 and the number of clients used was 5. Because of the limitation of the open source COPS implementation in supporting multi-threaded clients, we were able to send the requests only sequentially from the clients which is the reason for the rates being determined by the response time and the constant time delay provided between two requests at the client.

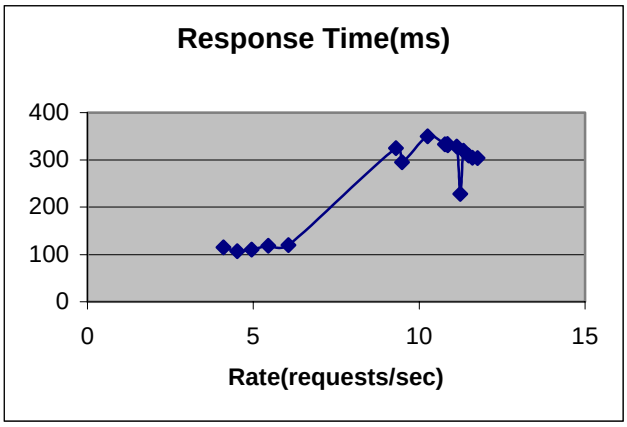


Figure 6: COPS-PR Request/Response Measurement

The following are the observations from the COPS-PR measurements in Figure 6:

1. At low load, the response time is around 100 ms. This shows that the policy server is able to process around 10 requests/second when each requests requires processing of 9 policy rules.
2. When the experiments were repeated with the requests requiring processing of one policy rule, the response time was around 60 ms. This shows the response time is an increasing function of the number of policy rules to be processed, which needs further experiments and analysis.
3. In the current implementation, the COPS-PR requests from different clients are received by multiple threads in the PDP but their decisions are sent from a single thread. This means that the policy server performance could be improved by using multiple threads for sending decisions.
4. The response time for the no caching case was around 1.535 seconds for 2.42 requests per second, which is around 15 times the response time with caching.

5. The response time triples to 300 ms when the request rate is doubled from 5 to 10. This needs further investigation.

Figure 7 shows the average response time against the request rate for COPS-RSVP requests, experienced by 3 clients and with caching. The request rate shown is 4 times the *session generation rate* because of two requests for PATH and RESV states together and two more requests for deleting them. The response time is the average for the 4 requests from each session. Among the 4 requests, the processing time for PATH message is much more than for the other requests and it is the only one which invokes policy rule processing. For each session, only one policy rule is processed.

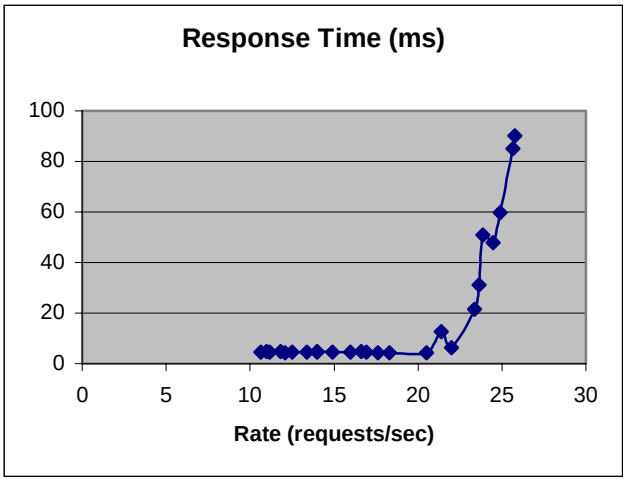


Figure 7: COPS-RSVP Request/Response Measurement

The following are the observations from the measurements:

1. At very low load, the response time is around 4ms. It increases sharply at around 20 requests/second and it could be due to the *session hold time* being fixed at 50 ms. However this needs to be established by further investigation and measurements.
2. The response time shown in figure 7 could be more than the actual response time, for some of the data points. This is because the COPS-RSVP requests from the test clients has to pass two ORB interfaces: one at the PEP which is single-threaded and the other one at the PDP which is multi-threaded, and there could be additional delaying in queueing at the PEP ORB.
3. Figure 7 shows that the policy server response time is around 90ms for a request rate of around 25 COPS-RSVP requests per second totally from 3 clients. Response time at higher rates needs to be investigated.

The difference in response times between the COPS-PR and COPS-RSVP requests for processing one policy rule is around 55 ms. One of the reasons for this is that COPS-PR request processing requires more interactions (5 interactions for the example rule used in tests) between the policy decision engine and the provisioning adapter while COPS-RSVP involves only one interaction. This can be optimized to improve the performance.

The average time taken to retrieve a policy rule from the repository adapter is around 11ms.

The number of rules in the directory has minimal impact on the openLDAP query time since indexing of policy rules is used.

5. Conclusion

The design, implementation and preliminary performance results of a policy server for QoS management of IntServ/Diffserv networks, is reported. The preliminary performance measurements show that the policy server is able to handle around 250 COPS-RSVP requests per second. COPS-PR processing takes more time compared to COPS-RSVP requests. It is found that the directory access could become a bottleneck in scaling the performance of the policy server and it can be improved substantially by employing appropriate policy caching mechanisms. We plan to do more rigorous study of the scalability of the policy server, extension to other domains such as security, inter-domain policy management, and policy consistency checking which is definitely required when more policy rules are defined and added to the policy repository.

6. Acknowledgements

The authors acknowledge the contributions of the QoS/PBN project team members – Jayachandran Maniyeri, Zhang Zhishou, Tian Tsong NG and Thavarajah Arunasala Iyer from KRDL and Dorothea Lampe, Anton Steinmaier and Steffen Thiele from Siemens ICM.

7. References

[1] IETF Policy WG Draft – Policy Core Information Model – version 1 specification, draft-ietf-policy-core-info-model-07.txt

[2] IETF Policy WG Draft – Policy Framework LDAP Core Schema, draft-ietf-policy-core-schema-07.txt

[3] IETF Policy WG Draft – Policy Framework QoS Information Model, draft-ietf-policy-qos-info-model-01.txt(expired April, 2000)

[4] IETF Policy WG Draft – QoS Policy Schema, draft-ietf-policy-qos-schema-01.txt

[5] IETF Policy WG Draft – Information Model for describing Network Device QoS Mechanisms, draft-ietf-policy-qos-device-info-model-01.txt

[6] IETF Policy WG Draft – Domain Per Hop Behavior Set Specification, draft-ronc-domain-phb-set-specification-00.txt

[7] IETF Policy WG Draft – LDAP schema for Domain Per Hop Behavior Set, draft-ronc-domain-phb-set-ldap-rep-00.txt

[8] IETF SnmpConf WG Draft – Policy Based Management MIB, draft-ietf-snmpconf-pm-01.txt

[9] IETF DiffServ WG Draft – DiffServ QoS Policy Information Base, draft-ietf- DiffServ-pib-01.txt

[10] IETF RAP WG Draft – Old version of DiffServ PIB, draft-mfine-cops-pib-02.txt

[11] IETF DiffServ WG Draft – Management Information Base for the DiffServ Architecture, draft-ietf- DiffServ-mib-03.txt

[12] IETF RAP WG Draft – Framework Policy Information Base, draft-ietf-rap-framework-pib-01.txt

[13] IETF RAP WG Draft – COPS for Provisioning, draft-ietf-rap-pr-03.txt

[14] IETF RAP WG Draft – Structure of Policy Provisioning Information, draft-ietf-rap-sppi-01.txt

[15] IETF DiffServ WG Draft – Informal Management Model for DiffServ Routers, draft-ietf- DiffServ-model-04.txt

[16] RFC 2748 – Common Open Policy Service(COPS) protocol

[17] RFC 2749 – RSVP Usage of COPS protocol

[18] http://extwww.lulea.trab.se/cops/

[19] http://www.kom.e-technik.tu-darmstadt.de/rsvp/

[20] http://www.openldap.org/

[21] IETF Draft draft-ietf-rap-rsvp-newidentity-00.txt

[22] Lightweight Directory Access Protocol (v3) - http://www.ietf.org/rfc/rfc2251.txt

[23] Shepard, S.J., "Policy-based networks: hype and hope", IT Professional , Volume 2 Issue 1 , Jan.-Feb. 2000, pp. 12 – 16.

[24] Wang Changkun, "Policy-based network management" Proceedings on International Conference on Communication Technology, WCC - ICCT 2000, Volume 1, pp. 101 –105.

[25] Moridera, A., Murano, K., Mochida, Y., "The network paradigm of the 21st century and its key technologies", IEEE Communications Magazine , Volume 38 Issue 11, Nov. 2000, pp. 94 –98.

[26] Flegkas, P., Trimintzios, P., Pavlou, G., Andrikopoulos, I., Cavalcanti, C. F., "On Policy-based Extensible Hierarchical Network Management in QoS-enabled IP Networks", Proceeding of workshop on policies for distributed Systems and Networks. (Policy 2001), Spinger-verlang LNCS-1995, Bristol, UK, January 2001.

[27] Flegkas, P., Trimintzios, P., Andrikopoulos, I., and Pavlou, G., "Considerations on Policy-based Network Management", 2000 London Communication Symposium.

[28] Bhatia, R.; Lobo, J.; Kohli, M., "Policy evaluation for network management", Proceedings of IEEE INFOCOM 2000, Volume 3, pp. 1107 –1116.

[29] Stone, G.N., Lundy, B. Xie, G.G., "Network policy languages: a survey and a new approach", IEEE Network, Volume 15 Issue 1, Jan.-Feb. 2001, pp. 10 –21.

[30] Hinrichs, S., "Policy-based management: bridging the gap", Proceedings of 15th Annual Computer Security Applications Conference, 1999 (ACSAC '99), pp. 209 – 218

[31] Mehra, A., Verma, D., and Tewari, R., "Policy-based Diffserv on Internet servers: the AIX approach", IEEE Internet Computing , Volume 4, Issue 5, Sept.-Oct. 2000, pp. 75 –80.