UPM: Unified Policy-Based Network Management[1]

K.L. Eddie Law, Achint Saxena
The Edward S. Rogers Sr. Dept of Elec. & Comp. Engineering, University of Toronto, Canada.

ABSTRACT

Besides providing network management to the Internet, it has become essential to offer different Quality of Service (QoS) to users. Policy-based management provides control on network routers to achieve this goal. The Internet Engineering Task Force (IETF) has proposed a two-tier architecture whose implementation is based on the Common Open Policy Service (COPS) protocol and Lightweight Directory Access Protocol (LDAP). However, there are several limitations to this design such as scalability and cross-vendor hardware compatibility. To address these issues, we present a functionally enhanced multi-tier policy management architecture design in this paper. Several extensions are introduced thereby adding flexibility and scalability. In particular, an intermediate entity between the policy server and policy rule database called the Policy Enforcement Agent (PEA) is introduced. By keeping internal data in a common format, using a standard protocol, and by interpreting and translating request and decision messages from multi-vendor hardware, this agent allows a dynamic Unified Information Model throughout the architecture. We have tailor-made this unique information system to save policy rules in the directory server and allow executions of policy rules with dynamic addition of new equipment during run-time.

Keywords: Common Open Policy Service (COPS) protocol, Lightweight Directory Access Protocol (LDAP), network management, Quality of Service (QoS), resource reservation.

1. INTRODUCTION

Internet provides "best-effort" datagram service, which has limited efficiency for large volumes of varying kinds of application traffic. Performance related parameters, such as bandwidth and delay, in the Internet are currently not used for connection management. Managing the Internet and providing different Quality of Service (QoS) classes [1, 2] to users becomes essential, hence allowing a variety of applications, for example, Voice over Internet Protocol (VoIP) and real-time data transmission, to operate harmoniously in the Internet.

To manage different QoS operating schemes, policy-based network management has been proposed [3, 4] where different levels of service are provided according to assigned policies or rules. These policy rules define admission control and traffic shaping mechanisms. Its variables may include bandwidth, jitter, delay, duration and connection volume limitations. The resource allocation protocol and the policy framework working groups in the Internet Engineering Task Force (IETF) are responsible for the standardization process.

The framework model [3] from IETF is a two-tier scheme that consists of policy manager (PM) or policy server (PS), and network routers. Two entities are defined, Policy Decision Point (PDP) and Policy Enforcement Point (PEP). Policy Enforcement Point (PEP) enforces policy rules into a network domain, and Policy Decision Point (PDP) makes decisions on policy requests received from PEP (see Figure 1a) by fetching policy rules that are stored in a centralized location using a Lightweight Directory Access Protocol (LDAP) [5] database. Usually the PEP is located inside an Edge Router (ER) device, while the PDP is located on a Policy Server (PS) machine. There can be several PDPs in a network domain, one for each kind of PEP device.

The Common Open Policy Service (COPS) Protocol [6] has been developed by the IETF in conjunction with this two-tier model, for policy communications between the PEPs and PDPs. COPS is a lightweight, and flexible, client-server based protocol. At the PEP, COPS is used for outsourcing requests. Its extension, the COPS-PR [7], provides bulk DiffServ [1] type enforcements. Though the COPS protocol formats are standard, at present there is no standard for

[1] The UPM project is supported by Nortel Networks at Belleville. The prototype is developed at the Client/Server Lab at the University of Toronto that is established by the Nortel Institute for Telecommunications (NIT) [17].

structures within messages such as request or decisions. This may cause problems with inter-vendor COPS compatibility.

There are several benefits to the two-tier model, in that it is simple and moderately extensible. It is well adapted to the Resource reSerVation Protocol (RSVP) [8] paradigm. However it does have several limitations. The most conspicuous limitation is the scalability issue in heterogeneous networks, in terms of router manufacturers and models. Moreover, each router may have different implementations of COPS, ranging from something as minor as different version numbers to something crucial such as different message formats. The current two-tier scheme assumes that the whole network uses a standard COPS protocol and message format.

Another consequence of this is that legacy routing equipment will not be supported. This is of major importance since it is a hurdle towards the deployment of QoS in current networks. To offset startup costs any QoS system should be able to integrate well with legacy equipment. Moreover, a single PEP could be connected to multiple PDPs, each with a different client type [6]. If a single PDP fails then this may cause continuity problems at the PEP. A backup PDP may exist. However, the PEP will not automatically be aware of the location of the backup PS. Though a list of backup PDPs may be manually configured into PEP at registration time, it may be problematic if all of them are down or unable to accept any more connections, or if the list was outdated.

Further investigation of the two-tier scheme leads us to conclude that it is not very scalable at all. There is no load sharing/balancing mechanism at PDPs, which can lead to problems in large networks with variable points of congestion. PEP blindly keeps on trying to connect to its list of PDPs, waiting for a timeout between each try.

All these issues must be addressed before QoS and policy-based management become widely adapted in the Internet. In this paper we present enhancements to the two-tiered policy-based management paradigm suggested by the IETF. In section 2, we present the new design, which we call the Unified Policy-based Management (UPM) model. In section 3, we discuss the functionality and operation of components in the UPM model. In section 4, we discuss issues related to the implementation of different components in UPM. In section 5, we discuss the design works on the UPM prototype. In section 6, we present a summary and discuss future enhancements.

2. UNIFIED POLICY-BASED MANAGEMENT (UPM) MODEL

2.1 Architecture of The UPM

The UPM is a multi-tier policy management system that is similar to the agent technology or middleware design. In a policy domain (PD), apart from the directory server, there are three tiers in this system architecture (as shown in Figure 1b). The top-tier is where the policy managers or the policy servers (PMs/PSs) are located, and they perform the policy decisions. The middle-tier layer coordinates the communications and creates transparency between network routers and the policy servers. The entity located in this middle-tier system is known as the Policy Enforcement Agent (PEA). If the edge routers do not understand the policy decisions, the PEA is responsible for policy rules translation and executions. The lowest-tier consists of different network routers, the edge routers (ERs), and the boundary routers (BRs), also possibly covering core routers (CRs).

UPM works on both outsourcing events and one-way decision-making. As shown in Figure 1b, the policy server obtains a service request through either the COPS request from the edge routers or from the system administrators with domain level decision. Policy rules are obtained from the directory system and PS will finalize a to-be-applied policy rule and deliver this decision to the PEA at the middle-tier.

Each PEA at the middle-tier enforces all policy rules from the policy servers. Its objective is to provide guaranteed QoS flows at ERs for end-hosts within its policy domain (PD) and BRs from other PDs. The primary functions of the PEA include: (1) relaying protocol messages between PS and ERs/BRs; (2) translating different policy rules; (3) distributing different policy sessions to PSs; (4) ensuring guaranteed QoS traffic at edge/boundary routers (5) informing ERs/BRs about other PEAs around, if required.

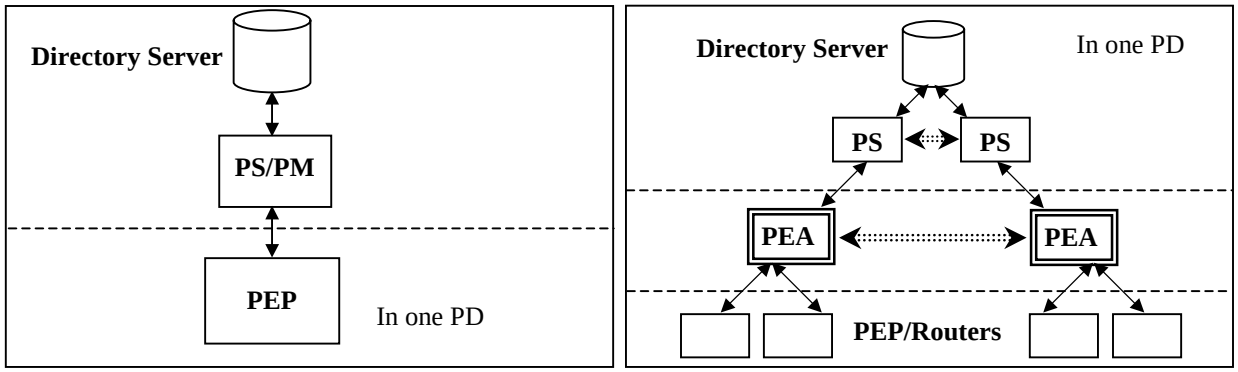

Figure 1 (a) Two-tier scheme. (b) UPM.

One enhancement in UPM is that it provides load balancing and load sharing mechanisms at the middle and upper tiers. When there are numerous routers to control, there may be multiple PEAs in a single PD. The system provides inter-PS communications, inter-PEA communications and PS-PEA communications. Readers may consider the works at PEA may introduce unnecessary delay and workload to PEA. The design of this proposed UPM furthermore provides an expedited message passing mechanism that will be enabled when the PEA notes both routers and PSs can fully understand the information content. In this scenario, the information is passed between the routers and policy servers without the intervention of the middle-tier agent. This UPM design enables a concept of unified information model (UIM) [15] between the policy servers and the middle-tier entities. This design unifies the system design at both the policy server and the middle-tier entity. Therefore, we can add more PSs and PEAs at any time if required without disrupting any existing policy controlled services. Hence this system is named the Unified Policy-based Management (UPM) system.

2.2 Achievements of the UPM Design

There are some important design achievements we have made with the UPM model:

- The load balancing and sharing mechanisms enhance system scalability.
- The system reduces the points of failure at both the middle and upper tiers, and hence enhancing system reliability.
- All communication above the middle tier is via a common protocol (COPS), common message formats; and enables the design of the unified information model.
- PEA can control any routers with different control interfaces through the UIM with proper translation.

3. UPM COMPONENTS

A UPM implementation consists of some components that exist in the two-tier framework design. Moreover, some of these components have enhanced or simplified functions. Note that the PEPs are usually located inside a router, while PDPs are located in a policy server (PS). In providing with the unified information model, COPS is the only policy protocol used to communicate between a PEA and a PS. In this system, all PSs use the same version of COPS and all the messages passed around have the same structure and format. However each PS may not support the same client types in providing extensible feature of the system, this is discussed later. At the current design, PSs communicate with a centralized directory server/database. All PEP/routers communicate via the PEA only. This allows PEA to monitor traffic loading in the networks and on the PSs. Figure 2 shows a conceptual implementation of the UPM.

3.1 Network Routers and Policy Enforcement Point (PEP)

A PEP is normally located inside a router (usually an Edge Router, ER). Differing from the two-tier model, we allow non-COPS compliant PEPs to exist in UPM. All network routers connect via a PEA to the policy servers. The PEA handles device configuration. It is able to monitor the following network routers:

- Native COPS PEP – one that uses the same version of COPS and policy rule content as the upper tier; this PEP still connects to the PEA for traffic monitoring, but the connection is bypassed straightly to a relevant PS.
- COPS PEP – one which uses the COPS standard protocol, but has different version numbers and message formats. This will be the most common case when routers of other vendors try to connect with the PEA.
- Non-COPS router – one that does not have COPS for its policy requests. Most legacy routers will fall under this category. The PEA can set up routers that do support some form of QoS, but cannot pull policies. Most of these routers can be configured by a Command Line Interface (CLI), usually via telnet, and can be monitored via Simple Network Management Protocol (SNMP) [9].

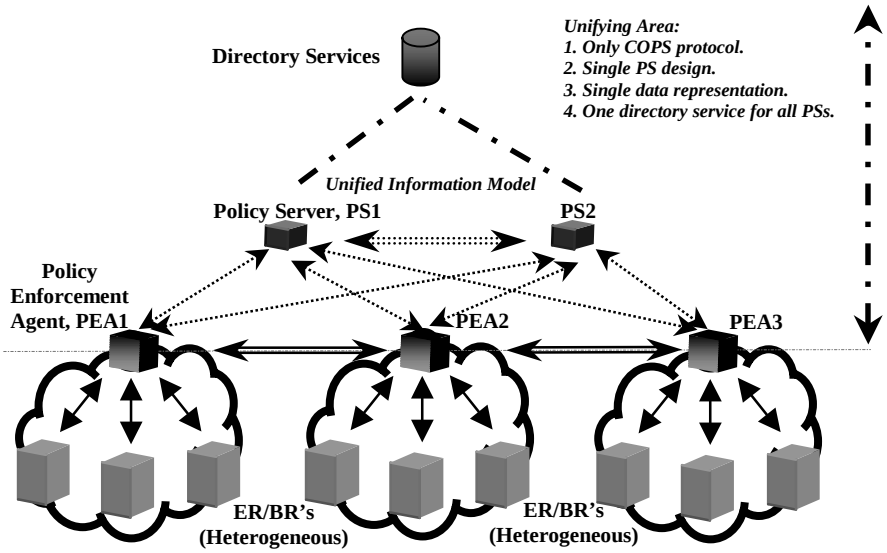

Figure 2: A UPM based implementation

3.2 Policy Enforcement Agent (PEA)

The PEA is at the heart of this multi-tier scheme. It is multi-functional and provides system flexibility and scalability. A PEA is more than just a proxy for service requests; it resembles a gateway to the PDPs. It provides features such as:

- A central and transparent point of connection for all routers, and seamless integration with multiple policy servers. All routers only connect to the PEA, which distributes the loads to PDPs.
- A translation point between routers and policy server, it translates different types of QoS and COPS requests into a version of COPS understood by the PDPs. Translation is done on a module basis, with the modules being dynamically loaded into the PEA.
- Dynamic load balancing/sharing for connections to and from routers in the policy domain.
- A dynamic TCP-based switching mechanism for COPS messages when translation is not required, to reduce latency and also for load balancing.
- A user interface is designed to manage and monitor the network.
- There may be multiple PEAs in a policy domain, and they are able to discover and communicate with each other through the inter-PEA process.

3.3 Policy Decision Point (PDP) at The Policy Server (PS)

Policy server makes all the policy decisions for PEP devices connected to it. To policy server in UPM, PEA works as the PEP. For the PEA can provide proper rule translation, communication version control, and it can have rudimentary QoS functionality, for example, of legacy equipment. Therefore, the design of PS is simplified. This is due to the fact that PEA is able to translate any non-PS compliant messages. It further allows us to increase the number of policy servers at ease. There may have multiple PSs and multiple PEAs in a single policy domain.

3.4 Directory Server

Currently, the directory server is used as a policy rules repository [3] as it was suggested at the IETF. All PDPs within a domain can access this database to obtain the policy rules through the Lightweight Directory Access Protocol (LDAP). LDAP schema for a policy repository is in draft stage (IETF Policy Working Group of [10]). LDAP server is used to provide directory services in the system prototype.

4. RELATED ISSUES

4.1 Design Issues on PEA

The PEA is the most important and complex component of the UPM model. It is not a simple proxy, but rather it is a multifunctional component with a primary goal of providing seamless integration of routers with multiple PSs. To summarize, a router only passes service request to a PEA, which then distributes the request to a PDP. In the following, we discuss the functionality and various issues on the PEA.

4.1.1 Translation

One of the functions of the PEA is to provide translation to the native COPS version and format. Several design methodologies exist. One way of approaching this issue is to divide the problem into two sets, this is shown in Figure 3.

Non-COPS routers – For 'push' only routers, manual configuration will be required. A user interface can be used to monitor and user-based input or a client program can take place of the 'pull' mechanism for the router. The result can then be pushed onto the router using whatever mechanism is available, such as Command Line Interface (CLI), telnet, etc. For 'pull' capable routers, using some other, possibly proprietary, policy protocol a basic store, convert and forward mechanism can be used. The forwarding mechanism consists of opening a new connection or using a pre-existing connection to a PS, to send and receive requests and decision messages.

COPS compliant routers/PEPs – Different vendors may have different implementations of COPS, even if they are standard compliant it is likely that the request and decision message structures will be different for each of them, and to that in the UPM's PSs. In these cases translation will be required. Translation can be divided into two paradigms: a) For non-standard/RFC implementation of COPS the same mechanism as for non-COPS routers can be utilized. b) For COPS standard compliant routers, individual COPS messages can be converted and forwarded as necessary. This operation is more like a transparent TCP proxy, and should be considerably faster than storing and generating new cops messages due to fewer processing overheads. However, a security issue is involved in this situation, in case Keyed-Hashing for Message Authentication (HMAC) [12] is in use. In this case a new Integrity object [6] will have to be issued, and for this the key for the PS needs to be known by the PEA. If this is not acceptable then a local PS will be required which can handle incoming messages and pass on the request and decision messages only. Load balancing can be achieved at this point by forwarding requests to the appropriate PS.

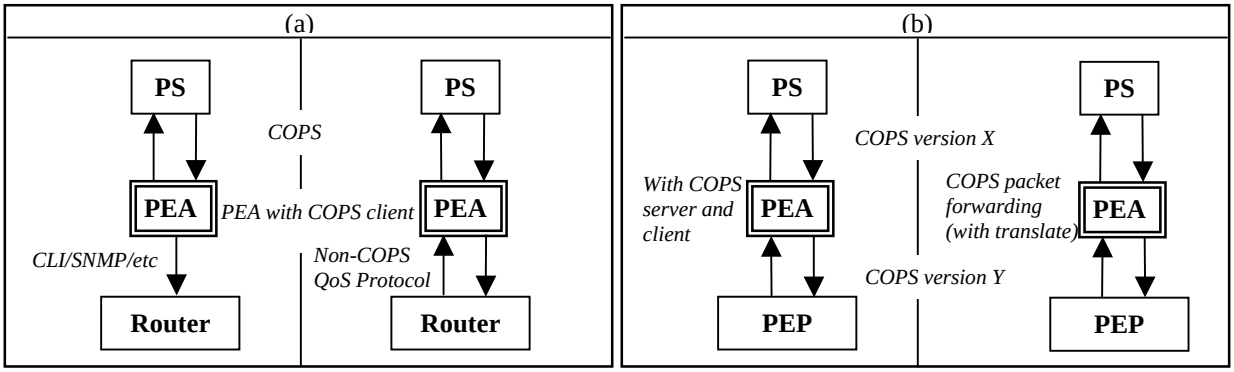

Figure 3: Translating mechanisms (a) Non-COPS router (b) COPS compliant PEP

4.1.2 Translation module Repository

For flexibility, the translation routines in the PEA are kept in modules that can be dynamically loaded. Since there can be multiple PEAs, it would be expedient that the modules be kept in a repository. The repository can be locally or remotely located. Licensing issues may necessitate keeping modules at a fixed location. In this case only certain PEAs may have translation modules, we call these local, static or non-shareable modules. This issue is solved by inter-PEA communication, discussed in section 4.3.4. For real-world implementations, remote Java is particularly useful in this aspect. Remote Method Invocation (RMI) [13] can be used to dynamically load modules from a repository. We do not consider the use of Common Object Request Broker Architecture (CORBA) [14], which is in general a slow process on network communications.

4.1.3 TCP-bypass

TCP-level switching or bypass is designed at the PEA, and it is one of the more important functions of the PEA. If routers and PS can communicate using the same COPS message format and information content, then it does not need any intermediary translation. To reduce network overhead and latency in such cases, the TCP-bypass mechanism can simply forward any incoming connections or requests to one of the PSs known to the PEA. The concept is illustrated in Figure 4.

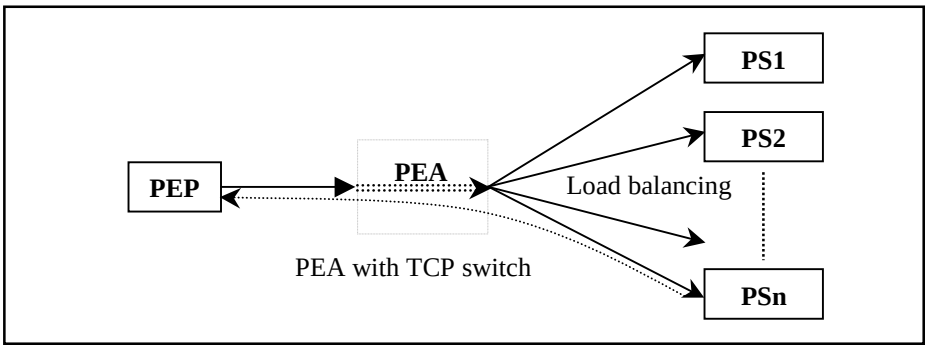

Figure 4. TCP-bypass

The TCP-bypass operates like a transparent proxy but is subtly different. Incoming TCP connections are allowed to pass down through the PEA, or are forwarded without processing, according to the first COPS data packet they transmit. Using the information in the COPS header, the version and implementation signature may be deduced. If this matches the native COPS format and information content, then a new connection is opened to a PS and the TCP connection information is stored/cached. The TCP-switching can be done either at TCP-level, which is TCP connection-based, or at the COPS level, or both. This allows for a "transparent" connection from the PEPs, where persistent connections are set up and maintained. For COPS request bypassing, careful implementation of mechanisms is required to mirror and handle COPS connection establishment messages to the PSs, and their corresponding replies from the PS and unsolicited PS messages. The Client Handle [6] and other information in the COPS Header can uniquely identify the COPS requests.

Load balancing can be placed at this point. A table of PSs known to the PEA can be kept. By monitoring PS related variables such as CPU utilization, memory usage, link utilization, ping times, and link capacity/congestion, a weight can be assigned to the PS. New requests to the PEA can then be serviced in a Weighted Round Robin (WRR) fashion.

Another important feature is the forwarding of connections based on client-type [6]. This can co-exist fairly well along with load-balancing. When a PS registers with the PEA it should notify the PEA about the client-types the PS can support. This information can be kept in the PS table as previously mentioned, and based on this information load-balancing to new connections can be restricted to the PSs which support the incoming client-type.

Placing of this mechanism in an actual implementation must also be considered carefully. For example, in a Linux based machine, to reduce latency the TCP-bypass should be done at kernel level. Switching in user space can be convenient, however it worsens latency and introduces other overheads. A useful side effect of TCP-bypass is that it does not allow direct, i.e. un-forwarded access to any PS. As a result, all unauthorized machines can easily be shut out. This may help to increase the security of the PS machine/device.

4.1.4 Multi-PEA and Inter-PEA communications

To allow system scalability and reliability, multiple PEAs are necessary. After PEA reaches a certain threshold based on open connections, CPU usage, network congestion, and other factors, it may decide to issue COPS redirect message to any new requests. Additionally, in a large network it may be beneficial to connect to PEA that is closer to the routers, rather than one far away. Having multiple PEAs allows the PEAs to be module specific, or area specific. There may be a higher concentration of a certain kind/brand of routers in one area, so PEA could handle those brand/models of routers in that area. Moreover, this may simplify licensing issues, if the router licensee decides to charge per PEA use of the translation modules if in case the vendor develops these modules.

To support system scalability, it is necessary to share data between PEAs, thus we shall design implement inter-PEA communication in the UPM. Data shared may include a PS list/table with associated states, monitoring and utilization information, module information – particularly whether any local non-shareable modules are available. To implement this, many mechanisms are possible. Service Location Protocol (SLP) [11] is one possible approach to dynamically locate other PEAs in a network. Once the locations are known, unicast connections can be established between PEAs in a closed linked-list or ring fashion. Data can then be passed around and updated in regular intervals. Another preferred approach is to extend the COPS extensible message to passing system loading and location information.

4.1.5 Miscellaneous issues

Due to its centralized position in UPM, the PEA is an ideal point at which network monitoring can take place. All connections passing through can be monitored. This information can be supplied to the PSs to help them make decisions. Various network monitoring protocols such as SNMP can be integrated into the PEA.

4.2 Design Issues on PS/PDP

Multiple PSs may exist in a policy domain; therefore, it may be beneficial to have inter-PS communication for exchange of client-type information, etc. While inter-PEA communications negates the need for bulk inter-PS data sharing, an allowance must be made for this capability. In certain circumstances it may be very useful, such as when a PS is purposely brought down for maintenance, its stateless information could be transferred to another PS. The mechanism may also be used with update notification as discussed in section 4.3.4. A similar mechanism to inter-PEA communication could be used, however fewer and non-regular updates should be required.

4.3 Related Design Issues on Directory Services

4.3.1 Search Algorithms

There are some issues regarding the LDAP directory service. The 'critical path' in the whole process of policy-based network management is the searching, filtering and matching of policy rules with requests. To provide optimum speed it is necessary to improve the searching algorithm and also provide it with a matching search database structure. One straightforward way of doing this is to minimize the information kept in the database to a bare minimum and easily parse-able format. Caching of requests at the edge router and in the PS can provide an added performance boost.

4.3.2 Conflict Detection and Resolution (CD&R)

Conflict detection and resolution (CD&R) is an important function when new policy rule is being added to the database. The CD&R should inform both the user and system administrator if new policy rule will be accepted. If there are

conflicts with existing policy rules, it should indicate the existence of policy rule conflict and propose certain resolution, if possible.

4.3.3 Service Level Agreement (SLA) Negotiation

A Service Level Agreement (SLA) is needed to populate the policy rules database. An SLA consists of QoS agreements between two PDs, and is enforced at the edge routers. Agreement parameters can include bandwidth, time, jitter, delay and access restrictions. Creation of an SLA is usually a high level administrative task, and can be achieved via a PS, or preferably a PEA. One approach is to create an SLA client-type for PSs that can be used to modify the LDAP database, rather than directly modifying the database. Such a mechanism must include CD&R at addition time.

4.3.4 Update Notification

A minor drawback of LDAPv3 is that it does not notify clients if the database has been modified [5]. The only unsolicited notification issued by the server is a notice of disconnection. To avoid this problem, all SLA negotiations must take place via the PEA, even if CD&R and other processing actually take place at a PS. The PEA can then notify all the other PEAs in a cascading effect, and the PS can similarly notify all the other PSs of a change.

4.3.5 Replicated Directory Services

LDAP Replication [16] services can be used to enhance the UPM implementation. There are several issues and approaches that can be taken to LDAP replication. One of these is to have one master LDAP database/server, and several replicated servers. The reasoning is that the demand on the LDAP database can be quite high, and since they are not updated that frequently - it is safe and useful to have distributed databases. Only the master database can be updated, and whenever it is all the slave databases will automatically be updated.

5. PROTOTYPE IMPLEMENTATION

5.1 Setup of Current Prototype

A prototype based on the UPM is being developed. Figure 5 illustrates the eventual setup of the prototype. The testbed currently has one Extreme Networks BlackDiamond 6808 layer-3 router [18]. Currently, the testbed also has ten Linux routers, and fifty personal computers to serve as end host systems. Implementation work is being done on Intel Pentium III PCs running Linux with Kernel 2.4, and it is built using both the Java and C programming languages, where Java [19] is used to construct the user interface at the PEA.

At the current stage, all entities are operating properly with some fundamental functionality. Some simple policy rules are created for the Extreme Networks' BlackDiamond 6808 layer 3 router. We have setup one directory server, one PEA. More works are required to work on load sharing, load balancing mechanisms and the TCP-bypass operations.

5.2 Developed Components for UPM Prototype

5.2.1 Directory Server

As mentioned in section 4, LDAP has been used for directory services. IETF defines policy rules for provisioning in the form of Policy Information Base (PIB) [20], which works well with COPS-PR [7], however for experimental purposes we use a simplified version of the LDAP Core Schema [10]. The LDAP server used is an open source implementation from OpenLDAP [21]. It provides a clean and complete C based API for LDAP, as well as several backend database options. The LDBM [21] backend was chosen. The searching and matching operations for OpenLDAP are fairly efficient, especially with caching enabled. Entry of the policy rules and schema setup was done in LDAP Data Interchange Format (LDIF) ASCII format. OpenLDAP provides tools to compile this into LDBM format. The database can also be viewed and modified dynamically with open source tools such as LDAP Browser [22].

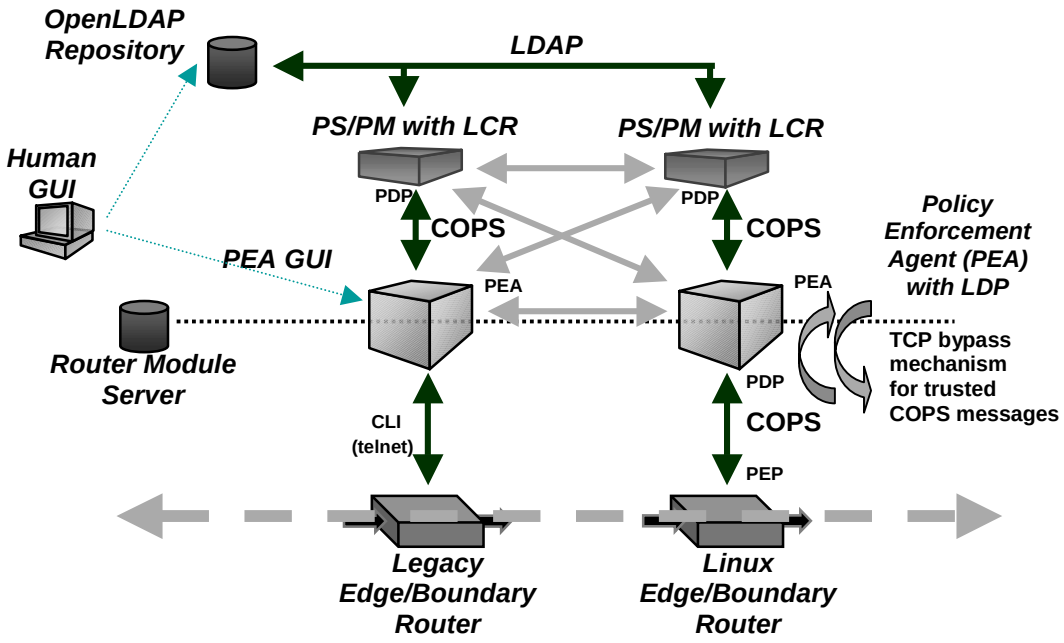

Figure 5. Testbed at The University of Toronto

5.2.2 PS/PDP

The policy manager or policy server program is written in the C programming language. Whenever a COPS request is received, the policy server examines the COPS content and generates an LDAP request message to the OpenLDAP server. All the associated policy rules will be delivered from the LDAP server to the PS. The 'filter, search, and matching' component in the PS is the 'critical path' in this policy based management setup. More efficient search algorithms need to be developed and deployed. Currently only a simple search routine is present, which tries to minimize search time for ranges of addresses. Caching the search results at LDAP level is implemented, however caching at the PS level needs to be implemented. Simplified SLA client type support, with very rudimentary CD&R, has been implemented.

5.2.3 PEA

The PEA is implemented in both Java and C programming languages with Java Native Interface (JNI) [13]. Typically, there are several modules implemented using Java, they are the local router database, the graphical user interface, loadable module server, several translation modules, scheduler for policy enforcement and the policy enforcer. For the modules implemented in C, they are the telnet module, the SNMP (Simple Network Management Protocol) module and the COPS module.

The functions of the modules are outlined as follows: a) *Routers and Switches Database* – is a text based database that stores information such as vendor, model number, IP address, connection method (SNMP, CLI, COPS). b) *Graphical User Interface (GUI)* - is implemented using SWING in Java to provide a consistent look and feel in different operating systems. It allows a user to add, modify, and delete information stored in the routers and switches database; allows the user to manually control network connections such as the scheduling of QOS to be applied on connections; and allows a user to modify various settings in PEA such as the address of module server, vendors list, and connection timeout. c) *Module Server* - since new routers can be brought into the network at anytime, the PEA has to be able to “learn” how to control the new routers. This is done by providing each router with a java module that defines the communication method. As a result, when a new router comes in, the running PEA can be extended by loading the new java module and

will be able to control the new router. The module server is responsible for transferring new java module to the PEA. It simplifies the process of updating new modules in PEA. Multiple PEAs can access this Module Server. d) *Policy Enforcement Scheduler* - schedules QoS to be applied and dispatch them to the policy enforcer according to the start time and end time of the QoS. e) *Policy Enforcer* - currently supports controlling of the routers that only support CLI. Thus when the policy enforcer receives a dispatch command from the policy scheduler, it tries to telnet to the router and send the appropriate CLI commands to modify the QOS for the connections. Once the telnet connection is established, subsequent CLI command to that router when be sent directly without reestablishing the connection. As stated before, the required CLI commands will be specified in the java module designed for that particular router. f) *Telnet Module* - The telnet module is responsible for connecting to the router as a telnet client and transferring CLI commands to the router for QOS control. Policy enforcer uses the telnet module to apply QOS commands. g) *SNMP Module* –the PEA uses this module to communicate with the routers using SNMP commands. If the router supports SNMP [9], the policy enforcer could use the SNMP module in place of the telnet module. SNMP has the advantage of being a connectionless protocol. h) *COPS Module* - this module is responsible for the communications between the PEA and the policy server. Every QOS command will be verified with the policy rules stored in the policy server before being added to the scheduler. The policy server does not belong to the PEA.

Most of the mechanisms described above are for non-COPS routers. For testing purposes some translation modules have also been implemented for COPS routers. These modules have a simple translate and relay as described in section 4.1.1. Implementation of the TCP-bypass is discussed in section 5.3.

5.2.4 PEP/routers

A simple Linux based COPS router/PEP, based on a simple request outsourcing model has been implemented. Libipq and iptables from Netfilter [23] are used to seize the first packets on new UDP and TCP connections. The PEP then connects to the PS via the PEA and obtains the QoS rules to apply. The rules are applied via Class Based Queuing (CBQ) previously compiled into the kernel, and the connection is allowed to continue. The PEA has also been set up to control QoS on the Extreme Networks Black Diamond switch. In this scenario, it represents a non-COPS router without any 'pull' mechanism. A user interface is used to query the PS and this is translated into CLI for the switch.

5.3 A case for TCP-Bypass

TCP-bypass is an integral part of the PEA. It does not only provide load balancing, it also allows native COPS messages to be TCP-switched. In Linux this will be done at the kernel level. To support this idea, five experiments and measurements were made at the PEA. A series of COPS messages such as Open, Keep-alive, Request and Decision were generated via the PEPs as mentioned in section 5.2.4 for testing.

The five experiments, based on the mechanisms illustrated in Figure 3, were: a) non-COPS router: Java/JNI based COPS to CLI translation for Extreme Networks BlackDiamond switch. Any overhead is due to the Policy Enforcer Queue Scheduler implementation. Downstream measurements were taken on only one direction, since a user interface was used to initiate the COPS connection. b) COPS version number switch – a simple Java based translation in which the version flag is switched, both upstream and downstream. c) COPS message parsing and switch – again only the version number is switched, however the COPS message is parsed object-by-object and reconstructed before forwarding, both upstream and downstream. d) Java based, no COPS translation – in this case, the translator code functioned exactly as a transparent TCP proxy. e) A C-based user space TCP forwarder/proxy, to compare performance. The times are measured when the packet arrives and leaves the kernel/system, over at least 500 messages/packets and multiple readings. The average one-way COPS to Translation Format times were noted over all COPS messages. In case no translation is required, the transition time within the system is noted. The results are summarized in Table 1.

From these results, it is obvious that avoiding translation can reduce the transition times/latency by avoiding processing overheads. A TCP-bypass implemented at the kernel level should reduce latency by a significant factor as compared to the C and Java TCP forwarders. This is because of the extra overhead TCP data incurs while passing through the different levels from kernel space to user space. It is safe to conclude that the performance can only get better in kernel space. The time saved will allow for many more transactions per second.

Translation Method (from COPS version 2)	**Average Time (seconds)**
COPSv2 → Telnet	0.0493
COPSv2 → COPSversion1, parsing	0.000771
COPSv2 → COPSversion1, flag switch	0.000495
COPSv2 → No translation (Java)	0.000199
COPSv2 → No translation (C)	0.000119

Table 1: Translation times

6. CONCLUSIONS

A multi-tier unified policy-based network management scheme, the UPM, has been proposed and discussed. This scheme shall provide some solutions to the problems facing the policy-based QoS industry today. Furthermore, inter-domain policy rule and SLA negotiation will be investigated and finalized. Optimized search algorithms for the directory server are required to be investigated in future.

7. REFERENCES

1. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Service", IETF RFC 2475, December 1998.
2. R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an Overview", IETF RFC 1633, June 1994.
3. R. Yavatkar, D. Pendarakis, and R. Guerin, "A Framework for Policy Based Admission Control", IETF RFC 2753, January 2000.
4. H. Mahon, Y. Bernet, S. Herzog, and J. Schnizlein, "Requirements for a Policy Management System", work in progress, November 2000.
5. Wahl, M., Howes, T. and S. Kille, "Lightweight Directory Access Protocol (v3)", IETF RFC 2251, December 1997.
6. J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Raja, and A. Sastry, "The COPS (Common Open Policy Service) Protocol", IETF RFC 2748, January 2000.
7. K. Chan, D. Durham, S. Gai, S. Herzog, K. McCloghrie, F. Reichmeyer, J. Seligson, A. Smith, and R. Yavatkar, "COPS Usage for Policy Provisioning," IETF RFC 3084, March 2001.
8. R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP) - Functional Specification", IETF RFC 2205, September 1997.
9. J. Case, M. Fedor, M. Schoffstall, and J. Davin, "Simple Network Management Protocol", IETF RFC 1157, May 1990.
10. J. Strassner, A. Westerinen, E. Ellesson, R. Moore, and R. Moats, "Policy Framework LDAP Core Schema", work in progress, May 2001.
11. E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol , Version 2", IETF RFC 2608, June 1999.
12. H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", IETF RFC 2104, February 1997.
13. Java, http://java.sun.com.
14. Common Object Request Broker Architecture (CORBA), http://www.corba.org.
15. K.L.E. Law, "XML on LDAP Network Database," IEEE CCECE 2000, Halifax, Canada, 2000.
16. J. Merrells, E. Reed, and U. Srinivasan, "LDAP Replication Architecture", work in progress, March 2, 2000.
17. Nortel Institute for Telecommunications at the University of Toronto, http://www.nit.utoronto.ca.
18. Extreme Networks BlackDiamond 6808 Switch, http://www.extermenetworks.com.
19. Jbuilder Foundation for Linux, http://www.borland.com.

20. M. Fine, K. McCloghrie, J. Seligson, K. Chan, S. Hahn, A. Smith, and F. Reichmeyer, "Framework Policy Information Base", work in progress, November 2000.
21. OpenLDAP, http://www.openldap.org.
22. LDAP Browser/Editor, http://www-unix.mcs.anl.gov/~gawor/ldap.
23. The Netfilter Project: Packet Mangling for Linux 2.4, http://www.samba.org/netfilter.