THE META-POLICY INFORMATION BASE

by

Andreas Polyrakis

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

0-612-58776-2

Abstract

THE META-POLICY INFORMATION BASE

Andreas Polyrakis

M.Sc. thesis, 2001

Graduate Department of Computer Science – University of Toronto

The recent considerable growth of computer networks has revealed significant scalability and efficiency limitations in the traditional management techniques. Policy-Based Networking (PBN) has emerged as a promising paradigm for Network Management. The Common Open Policy Service (COPS) and its extension for policy provisioning (COPS-PR) are currently being developed as the protocols to implement PBN.

COPS-PR has received significant attention and seems efficient for several Management areas. However, the rigidity of its policy-enforcing mechanisms constrains the intelligence that can be pushed towards the managed devices. This work attempts to relax this limitation by using meta-policies, rules that enforce the appropriate policies on the devices. Meta-policies are stored and processed by the devices, independently of their semantics, making in this way the model more efficient, scalable, distributed and robust. The additional functionality is implemented through a new Policy Information Base (PIB) that we have defined, the Meta-Policy PIB.

Acknowledgments

First of all, I need to express my gratefulness to the Department of Computer Science of the University of Toronto for accepting me into the graduate program and giving me the opportunity to pursue a graduate degree here. Also, I would like to thank the department for their financial support, without which I would be unable to complete my studies.

However, most of all, I would like to thank my supervisor, Prof. Raouf Boutaba from the University of Waterloo, for his guidance and support. His ideas and comments inspired me, and his experience and insight in the area of Network Management has been great assets for my work. He gave me the chance to deal with very interesting topics and live experiences that I will never forget. I feel really lucky to have had Raouf as my supervisor.

Apart from Raouf, I would like to thank the rest of his team of graduate students for their comments and feedback, as well as their encouragement and support. Especially, I would like to point out Youssef Iraqi, whose comments influenced significantly my work and Salima Omari, for her valuable help.

I am deeply indebted to Prof. Ken Sevcik, for several reasons. Ken was assigned as my official supervisor from the beginning of my graduate studies, and he took care of all my administrative issues. However, Ken was also the second reader of this thesis, and his comments were important, both in the content and the structure of the thesis. Ken is a brilliant person, which managed to amaze me from the very first time we met.

Two more persons that I need to thank are Prof. Irene Katzela and Fotios Harmantzis, both from the ECE department. I would like to thank Irene for her guidance at the beginning of my graduate studies, for her valuable advice and for several useful conversations throughout the entire period. Fotios is one of the most interesting persons I met in Toronto. His perception of life, his academic experience and his analytic reasoning made his advice on several aspects, academic or not, very valuable and helpful.

Of course, I cannot forget the big community of the Greek graduate CS students. They welcomed me warmly when I arrived in Toronto; they became my friends and roommates; they showed me around the city; they gave me hints and tips about the life in Canada and UofT; they helped me out with all these tiny problems that may look huge if nobody is there to assist you; and, of course, by being computer science graduate students, their advice was significant for any issue of academic or scientific nature.

Another person I need to thank is Verena. I discussed with her several of the issues that came up, and her ideas and comments influenced significantly the outcome of this thesis. However, the main reason that makes me feel indebted to Verena is her moral support and her presence in my life, which gave me the courage to tackle any difficulty and made my life pleasant and beautiful.

Last but not least, I need to express my gratitude to my family and friends for supporting, encouraging and standing by me during all this period. Although I would not like to thank individuals for the fear of forgetting someone, I would like to mention that without their physical or mental support, carrying out this work would not be feasible.

Toronto, March 2001
Andreas Polyrakis

Table of Contents

Chapter 1. Introduction

1.1. Purpose and Goals

This thesis was conducted as a part of a greater research framework that investigates issues towards self-configurable networks. In order to achieve this goal, we believe that two conditions must be met. First, the level of abstraction in Network Administration needs to be raised, so that a higher degree of automation can be allowed. Second, intelligence needs to be pushed towards the managed devices. These two properties, the intelligence of the managed devices in combination with a high degree of automation, will allow the existence of "smart" devices that configure themselves by getting or generating such configuration data that will allow them to adapt to the network state and needs at each specific moment. Our research has two dimensions that address these two conditions, respectively.

Policy-Based Networking (PBN) is a modern trend in Network Management within the first dimension: It raises the abstraction of Network Management by using high-level policies, from which configuration data for the network devices are automatically generated and distributed to the network elements. However, PBN fails to address sufficiently the second dimension: PNB is not a highly centralized model, since it uses special policy servers, which can be distributed within the network. Nevertheless, very little functionality is actually pushed inside the managed devices, which depend on the constant presence of the policy servers to operate properly.

We believe that PBN is a very promising management technique that will affect significantly the future of Network Management. The purpose of this thesis is to enhance the PBN by developing it in the second dimension, too, thus allowing the existence of self-configurable network elements. More details on our goals, as well as the motivation, is presented at the next chapters, along with the necessary background information.

1.2. Dependencies

This work defines a COPS-PR PIB, using the SPPI (Structure of Policy Provisioning Information) specification. At the time this work was conducted, COPS and COPS-PR were RFCs*, and SPPI was an internet-draft*; hence they may be modified before they reach their final form. Future versions of COPS and COPS-PR are not expected to modify the core of the protocols or the PIB functionality and semantics, on which this work is based. However, modifications of the SPPI specification, which is used to define the classes of the PIB, may make the output of this work syntactically out of date. Nevertheless, the revision of the PIB proposed here to make it consistent with the newer SPPI versions should be an easy task.

* In IETF (Internet Engineering Task Force), each new specification is published as an internet-draft. These drafts are widely available, have no formal status, are subject to removal at any time and evolve according to the comments and feedback that they received from the Internet community. If an internet-draft receives significant attention, becomes relatively stable and mature and is globally approved, it evolves into an RFC (Request for Comments). The RFC is an official document that describes the specification in a complete and well-understood way, and is approved by the majority of the Internet community. As with internet-drafts, RFCs do evolve, however the modifications are usually moderate. When the RFC has reached a state where no more modifications are considered necessary, it may evolve into an internet-standard.

1.3. Organization of This Document

The structure of this document is as follows:

- This chapter, **Chapter 1**, briefly presents the goal of our work and describes the structure of this document.
- **Chapter 2** discusses Network Management and the modern trends that seem likely to affect it in the near future.
- **Chapter 3** presents Policy-Based Networking, COPS and COPS-PR. A small example demonstrates how COPS-PR works.
- **Chapter 4** presents the motivation of our work and introduces the concept of meta-policies. The example of the previous chapter is used in order to demonstrate the use of meta-policies. Finally, meta-policies are formally defined.
- **Chapter 5** justifies our decision to use a PIB to implement meta-policies, presents and analyzes the requirements and discusses the design details of the PIB.
- **Chapter 6** defines the PIB. The PIB classes are described and how the data stored into these classes control the behavior of the device is defined.
- **Chapter 8** concludes this thesis by outlining the work in progress and presenting our future research goals, which mainly concentrate on further meta-policing enhancements. We also describe how other management techniques (especially Active Management) can be used to increase the power and efficiency of our work.

1.4. Terminology – Glossary

This document follows the terminology adopted by IETF and other standardization organizations, as outlined in [1]. The most commonly used terms are summarized here:

PBN	- ***Policy-Based Networking:*** A management technique based on high-level policies.
PDP	- ***Policy Decision Point:*** The Policy Server that distributes policing decisions to the PEPs, according to the high level policies.
PEP	- ***Policy Enforcement Point:*** The consumer of the policies. It enforces the policing data received from the PDP to the managed device.
COPS	- ***Common Object Policy Service:*** The protocol that is currently being developed by IETF, in order to implement PBN.
COPS-PR	- ***COPS for Policy Provisioning:*** An extension of COPS, targeting policy provisioning.
PIB	- ***Policy Information Base:*** A special tree structure maintained by the PEP, similar to a Management Information Base (MIB), where all policing data for this PEP is stored. The content of the PIB determines the behavior of the device.
PRC	- ***Provisioning Class:*** A class that defines the format and the semantics of a piece of policing information inside the PIB.
PRI	- ***Provisioning Instance:*** A specific instance of a PRC.
PRID	- ***Provisioning Instance Identifier:*** An identifier that uniquely identifies a PRI inside a PIB.

Chapter 2. Network Management

2.1. Definition

Network Management relates to planning, deploying, operating, monitoring and controlling the network in order to ensure that it is always running undisturbed and efficiently, while its resources are best utilized. Network Management starts with the design and deployment of the network; however, after this initial phase, it is mainly associated with maintenance tasks that collect and analyze data from the various network elements. These data can reveal abnormal or emergency situations as soon as – or even before – they occur. Also, these data allow the administrators to monitor the usage of the network resources, and according to it, fine-tune the network parameters and plan future upgrades.

2.2. The FCAPS Framework

Network Management may be divided into several functional areas. ISO has distinguished and standardized five major ones: ***Fault, Configuration, Accounting, Performance and Security Management***; this standardization is known as the ***FCAPS framework*** [2], [3], [4]:

- **Fault Management** deals with detecting, isolating, fixing and recording errors that occur inside the network.

- **Configuration Management** has to do with maintaining accurate information on the configuration of the network (hardware and software) and controlling parameters that relate to its normal operation.
- **Accounting Management** relates to user management and administration, as well as to accounting and billing for the use of the resources and services.
- **Performance Management** attempts to maximize the network performance. It is strongly related to QoS provisioning and factors like resource utilization, delay, jitter and packet loss.
- **Security Management** deals with ensuring security and safety in the network.

Although this work concentrates explicitly on Configuration Management, it covers implicitly all five management areas, since all of them relate somehow to the appropriate configuration of the network devices.

2.3. Traditional Network Management – SNMP

The management of the network devices, such as routers and switches, has always been a hard task [4]. Initially, the configuration was done through the Command Line Interfaces of the devices; in most cases, the administrator was required to configure each of the devices independently, even when these were configured to operate similarly. However, this soon appeared to be inefficient: while the networks started growing considerably both in size (number of managed nodes) and in complexity (different types of devices, number of configuration parameters), the need for automation became apparent.

For several years, the ***Simple Network Management Protocol (SNMP)*** gave a satisfactory solution to the problem. SNMP is based on special databases, called ***Management Information Bases (MIBs),*** maintained by each network device. MIBs provided a standard interface to manage objects on the devices, in a less device-dependent way. This raised the level of abstraction and allowed devices to be handled in a more unified way. In this way, SNMP allowed the administrators to manage the network remotely and to automate various management tasks.

However, SNMP (versions 1 and 2) was designed mainly for monitoring purposes and, although it managed to give a satisfactory solution to the problem for a while, now it seems to suffer from significant scalability and efficiency problems [4]: SNMP is a highly centralized protocol. In fairly large networks, too many resources may be consumed just to report normal network operation, while the detection of erroneous events and the reaction to them may be too slow. Besides, although SNMP managed to raise the level of abstraction in Network Management, the operations are still device-dependent. The growth of the modern networks demands a further increase in the level of abstraction, as well as decentralization of the management centers. These issues are examined by standardization organizations (such as IETF), which guide the future of Network Management.

2.4. Standardization Organizations - The Role of IETF

The ***Internet Engineering Task Force (IETF)*** [5] is "the protocol engineering and development arm of the Internet". Established in 1986, it is "a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet".

IETF hosts various working groups that cover different areas (e.g., routing, transport, security, etc.). These groups identify problems in the corresponding areas and address them by developing standard protocols.

IETF is closely related to other Internet organizations, such as the Internet Engineering Steering Group (IESG), the Internet Architecture Board, (IAB), the Internet Assigned Numbers Authority (IANA) and Internet Society (ISOC).

IETF plays a crucial role in the evolution of Network Management, since several of its working groups are related to it. For instance, IETF is the organization that has standardized the SNMP protocol. IETF attempts now to address the issues of SNMP through its next version (SNMP v.3). However, there are serious doubts whether SNMP will eventually manage to overcome its limitations and become the dominant protocol for Configuration Management again. This is why IETF also attempts to develop alternative management techniques that may replace or complement the existing ones. The role of IETF and its relation to Network Management will be further discussed later in this document.

2.5. The Future of Network Management

Traditional management techniques are not sufficient to cover the needs of modern Network Management. The need to be replaced, updated or augmented with new ones is evident. Several promising techniques attempt to address the existing issues in various ways. These techniques are presented in this section.

2.5.1. SNMPv3

As mentioned before, ***SNMPv3*** [6] is currently being developed by IETF, in order to resolve several issues of SNMPv2. In general, the new version attempts to unify the two different versions of SNMPv2 (versions 2u and 2*) [6]. Also, it attempts to include administrative and security functionality in the protocol. However, SNMPv3 does not seem to address adequately the scalability issues of SNMP. Nevertheless, due to its simplicity and the wide acceptance and use, it is expected to play a significant role, at least for monitoring, in the near future.

2.5.2. Active Management

Active Management is an attempt to take advantage of the properties of Active Networks in order to enhance the current management techniques, or create new ones.

Active Networks [4], [7], [8], [9] is a relatively new concept that emerged from the broad DARPA community in 1994-95. Architecturally, they can be divided into the Discrete (or programmable) and the Integrated (or capsule) approach [4], [7], [8], [10]; however discussing their difference is out of the scope of this document. In Active Networks, programs can be "injected" into the active devices (such as routers or switches) and affect their behavior and the way they handle data, even on per-application or per-user basis. Active routing and switching devices can be programmed to perform complex tasks and computations according to the content of the packets, which may even be altered as they flow inside the network. The term "active" is justified in two ways [8]: First, active devices perform customized operations on the data flowing through them. Second, authorized users/applications can "inject" their own programs into the nodes, affecting the way their data is manipulated. Due to these properties, open node architecture is achieved, where custom protocols and services can be easily deployed.

The radical changes that Active Networks introduce give to computer networks a flavor of distributed systems, and can be beneficial for a wide range of applications and tools [4], [8], [10]: Firewalls and proxies; nomadic routers; multimedia, real-time applications; multi-path routing; these are just the beginning of a long list. Of course, Network Management techniques can also be enhanced by exploiting the properties of Active Networks. We have already discussed extensively the impact of Active Networks on Network Management [4]. Here, we will just cite the results of the discussions conducted there.

First of all, Active Networks enable the distribution of the management applications and tools [4]. Mobile Agents, programs that travel inside the network and perform several tasks on behalf of the application that generated them, can be used for this purpose. Monitoring centers can be distributed in the network, moving the decision taking closer to the managed devices, and making the monitoring and reactions more prompt and precise. MIBs can be augmented with customizable variables, and alerts can be initiated by the devices. Management can become more direct and customizable, and the network can be managed during abnormal situations, such as high congestion or network partition. Several deficiencies of SNMP can be overcome.

However, apart from the general advantage of management distribution, Active Networks have positive impact on each specific FCAPS area, as well. Fine-tuned monitoring and fast reactions make Fault Management more effective and the network remains manageable during situations in which errors are present. Flexible and robust protocols can be easily deployed, and backup mechanisms can be configured. Configuration management is also significantly enhanced. Mobile agents can be used for inventory and software management. Resources can be partitioned and Virtual Local Area Networks (VLANs) and Virtual Private Networks (VPNs) can be created easily. Accounting Management becomes more accurate, since the users are billed according to the real use of the resources, and new types of Service Level Agreements (SLAs) can be defined. Performance can be increased due to new Quality-of-Service (QoS) protocols that use resources wisely, better traffic policing and shaping mechanisms, multi-routing protocols and application-specific handling of the traffic. Security Management can be enhanced by using special mobile agents that inspect and safeguard the network (e.g., by blocking Denial of Service attacks or by tracing back attackers with fake IP). Also, access to the network resources can be controlled more strictly and precisely. Active

Networks give a new dimension to Network Management by enhancing the existing methods and techniques and allowing the development of novel, radical ones. More details can be found in our previous work [4].

Active management has motivated and influenced significantly our work, although it does not relate to it directly. Our work is placed in context with it in the last chapter of this thesis.

2.5.3. Directory-Enabled Networking

Another promising trend in Network Management is ***Directory-Enabled Networking*** [11], [12]. Directory-Enabled Networking is based on ***Directories***, special purpose databases, storing configuration data for network devices and applications. The devices (or applications) connect to the Directory, query it, retrieve the appropriate configuration parameters and install them. This model allows a high degree of automation in the process of configuration management, and makes the concept of "plug-and-play" networks seem more feasible and realistic. Note that the concept of Directories is not something new: Directory services, such as DHCP, DNS, authentication, or user directories, can be found on current networks. However, Directory-Enabled Networking attempts to integrate all these different directories (which may represented the same or similar data, but not necessarily in the same format) into a single one that will unify and hold all such information, and make management easier and more consistent. Work on this area is mainly coordinated by the Directory Enabled Networks/Desktop Management Task Force (DEN/DMTF) [13].

Architecturally, Directory servers resemble DataBase Management Systems (DBMSs). The main difference is that the configuration data seldom change; hence directories are optimized for rapid responses to high-volume lookups; but their performance in updates is much poorer. Many other features found in DBMS systems, such as triggers, cascading deletes or transaction rollbacks are also of less importance. A matter of a great importance, however, is consistency and load balancing between several servers that implement a single Directory – because Directories are physically distributed, but logically centralized systems.

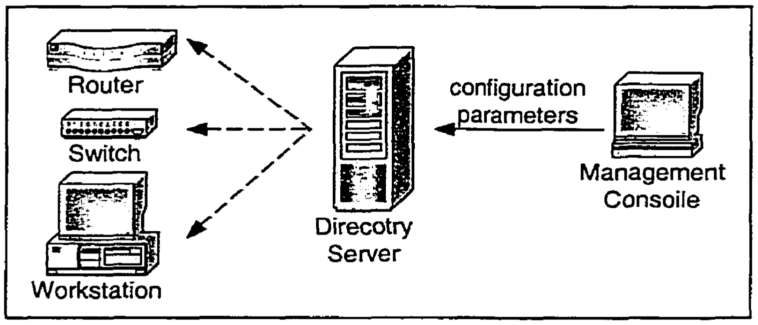


Figure 2.1: *Directory-Enabled Networking*

Directory-Enabled Networking has one significant deficiency: Directories are not efficient for non-static data. However, in Network Management, dynamic data (such as resource usage, statistical information or network events) may be necessary for some aspects of the configuration of the devices. Directories cannot handle such data efficiently, so other mechanisms are required in order to augment the functionality of the Directories. However, Directories handle the issue of static configuration data pretty well, and they are expected to play a significant role in the evolution of Network Management in the future.

2.5.4. Policy-Based Networking

Finally, another promising technique for Network Management is ***Policy-Based Networking***. The central concept in Policy-Based Networking is ***policies***, i.e., rules that determine the behavior of the network nodes. The key idea is that the administrator edits high-level policies that determine goals (rather than procedures). These policies are processed by special servers, which, bind them with the current network state, transform them into dynamic configuration data and send them to the network devices, determining in this way their behavior. The advantage of this model is that (i) the high level of abstraction in editing the policies simplifies the administration of large and complex networks, (ii) automation ensures the integrity and consistency in the behavior of the devices across the entire network, and (iii) the dynamic binding of policies at the policy servers allows new types of policies to be introduced more easily.

Policy-Based Networking will be discussed in detail in the next chapter.

2.6. The Big Picture - Our Contribution

Although all these technologies sound promising and address important issues of Network Management, none of them seems to be sufficient to handle all of them. For this reason, it is considered highly unlikely that one of these techniques will manage to dominate the others.

A more realistic scenario is that these will need to be combined and integrated, in order to efficiently manage present and future networks: Devices and services can be automatically configured through directories; the network behavior can be controlled through policy-based networking; SNMPv3 can be used to perform monitoring tasks in a secure fashion and active management and mobile agents can be used to enhance all previous techniques by making them more distributed and efficient.

Our work is, in general, focused on how PBN can become more decentralized and distributed, and how the other discussed techniques can be used to further improve its performance and efficiency. PBN raises the level of abstraction of Network Management and distributes it into the network to a certain degree. However, the intelligence is still concentrated at the level of the policy servers, which makes the devices depend on them. Our goal is to push intelligence towards the devices and make them more independent.

Also, we would like to allow the devices to exploit and integrate the other management techniques in order to enhance Policy-Based Management. Actually, we believe that Active Networks will give the devices the resources and the ability to perform complex tasks that can be exploited in a Policy-Based environment. This capability will allow them to implement some (and possibly, all) of the PDP functionality and become more independent and self-controlled.

Chapter 3.
Policy-Based Networking

3.1. Overview

Policy-Based Networking (PBN) has emerged as a promising paradigm for network operation and management [14], [15]. It is based on high-level control/management policies [16], [17], [18], i.e. rules that describe the desired behavior of the network, in a way as independent as possible of the network devices and topology. The key concept in PBN is that by describing "what" the network is supposed to do, rather than "how" (which happens with the traditional management techniques), the network details are hidden from the administrators. This makes the network easier to control, increases its flexibility, and ensures a consistent behavior across it.

PBN distinguishes two basic entities: the ***Policy Enforcement Points (PEPs)*** and the ***Policy Decision Points (PDPs)*** [1], [19]. The PEPs typically reside on the managed devices and control them according to directions that they receive from the PDPs. The PDPs process the high-level, abstract policies, along with other data such as network state information, and take policy decisions in the form of configuration data for the PEPs. In this way, the high-level policies that the administrator sets are enforced within the network devices. PBN is illustrated in Figure 3.1.

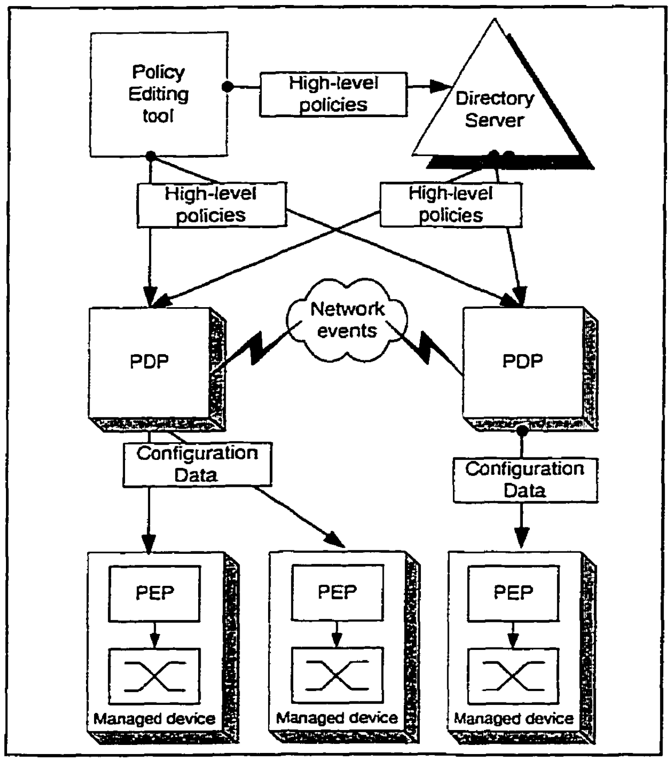


***Figure 3.1:** Policy-Based Networking*

3.2. Policies in PBN

As mentioned before, the basic concept in PBN is the management/control policies that describe the desired behavior of the network elements. The concept of policies is not something innovative; nevertheless, what is new in PBN is that the policies express goals rather than procedures.

In traditional Network Management, the administrators set some goals, and then create procedural policies that implement these goals. For instance if the administrator wants to give high priority to the manager subnet, he/she creates a policy similar to the following:

If ((SourceIP matches 10.10.1.0/24) or (DestinationIP matches 10.10.1.0/24))
then {remark with DSCP=6}

This policy has hardcoded the facts that (i) the manager subnet is 10.10.1.0/24, and (ii) high-priority is achieved by setting the packet's DSCP* to 6.

However, in the PBN approach, the administrator sets as a policy the goal itself:

If ((SourceIP matches Manager Subnet) or (DestinationIP matches Manager Subnet))
then {give high priority}

Of course, in this case, it is implied that the administrator somehow provides additional information that allows this policy to be interpreted (such as which is the "Manager Subnet" or what "high priority" means). However, this information is not hardcoded into the policies themselves. Hence, if for example the manager subnet is expanded to include 10.10.2.0/24, the administrator will only need to declare this fact. All policies related to this subnet will still be valid, since they do not contain information directly related to the network topology or the devices.

* ***DSCP (Differentiated Services Code Point):*** In Differentiated Services, the packets receive different treatment by the switching devices, according to the TOS field of the IP header (also named DS byte in Differentiated Services terminology). Six of its bits are used as a Differentiated Services Code Point (DSCP) in order to categorize each IP packet to one of the DiffServ classes (the other two bits are not used by DiffServ).

3.3. PBN Components

3.3.1. Management Console

Policies are edited using special management tools [20]. These tools provide interfaces that allow the network managers to edit the policies in a high-level, abstract way (Figure 3.2). Syntax, semantics and basic conflict checking are performed on these policies, which are then distributed, either directly or through the use of a directory, to the PDPs.

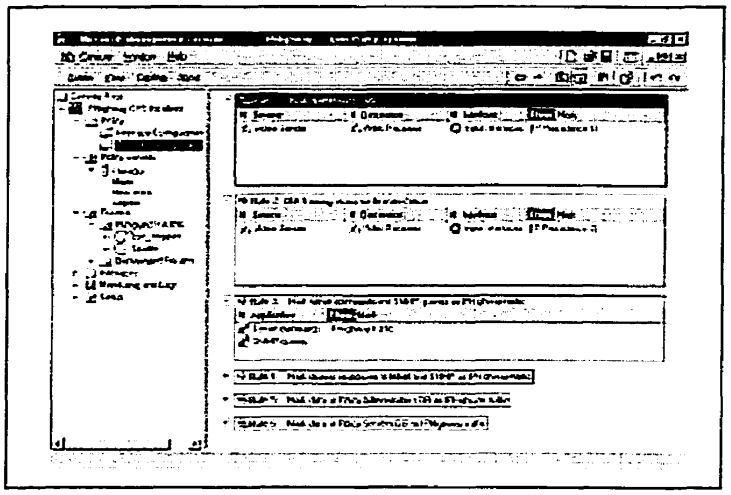


Figure 3.2: Policy editing tool (from [20])

3.3.2. The Policy Decision Points (PDPs)

The Policy Decision Points are responsible for mapping the abstract, high-level policies into low-level, device-specific configuration data [19]. Functionally, the PDP takes policy information entered from the management system, and process them along with other data, such as network state information. The PDP combines the policies with this information and produces the appropriate configuration data for the PEPs that it controls. The configuration data for each PEP is generated according to the capabilities and limitations of the device that this PEP controls.

It is important to emphasize that PDPs do not simply distribute policies to the PEPs. The role of a PDP is (i) to combine the high-level policies with the network state in order to determine the desired behavior of every device at that specific moment, and (ii) to generate the appropriate low-level configuration data for each device (in a supported format and according to its capabilities/limitations) that enforces this behavior. This implies that if the network state or policies change, the PDP may need to readjust the behavior of the devices, by sending updated configuration data.

3.3.3. The Policy Enforcement Points (PEPs)

The Policy Enforcement Points are the policy consumers [19]. Their role is to enforce the configuration data that they receive from the PDPs. The PEPs always obey the commands they receive from the PDPs.

3.4. The Outsourcing and The Provisioning Models

PBN is based on a client-server model of interaction between PEPs and PDPs. Two modes of operation are distinguished: the outsourcing and the provisioning [19], [21].

In the outsourcing model, the PEP receives a signaled event that needs to be treated according to some policy criteria. If the PEP cannot treat this event according to the already installed configuration data, it issues a request to the (appropriate) PDP, notifying it for the event occurrence. The PDP replies to the PEP by sending the data that must be installed in order to handle this event. This model is known as the "pull" model since the PEP "pulls" configuration data from the PDP, or as "reactive" model, because the PDP reacts to the PEP requests.

On the other hand, in the provisioning model, when the PEP connects to the PDP, the latter sends to the former all the applicable policies. These policies are stored in the PEP, and all incoming events are served according to them. This model of operation is known as the "push" model since the PDP "pushes" policies to the PEPs, or as "proactive" model because the PDP sends in advance the appropriate policies to the PEPs.

In both cases, the PDP is aware of the policies enforced by the PEP, and it may decide to update them by installing, deleting or replacing them, whenever it decides that they no longer reflect the desired behavior.

3.5. Why Not Directories?

Policy-Based Networking and Directory-Enabled Networks may seem to have several similarities: Both attempt the raise of the level of abstraction and the automation in the configuration of the network devices. Besides, functionally, the PDPs are similar to Directories, since they both provide the appropriate configuration data to the network devices. However, significant differences do exist.

Directories are simple databases that supply the devices with responses to the queries the latter submit. Directories cannot use the data that they store in order to generate other data. The processing of the data that they produce is restricted to simple database-style operations.

A PDP, on the other hand, does not simply distribute configuration data. The most significant and difficult task of the PDP is to generate these data from the high-level policies, according to the current network state.

To sum up, the difference between Directories and PDPs is that the nature of data that they distribute is different. Directories are efficient for static configuration data, which usually provide the basic configuration for the devices. Such data may include the IP and the subnet mask of the device, the DNS servers or the default PDP that controls this device. The PDPs, on the other hand, provide policies in the form of dynamic configuration data, which are produced by the PDP according to the current network state and may be updated at any time.

Directory-Enabled Networking and Policy Based Networking are two technologies that attempt to address different kinds of problems, and can be considered as complementary to each other. They can coexist in the same network in order to maximize its performance. We have already seen an example of such a cooperation in Figure 3.1, where directories are used in order to supply the high-level policies to the PDPs of the network.

3.6. Benefits of PBN

By using policies that describe goals instead of procedures, the policies are separated from the network details. This approach has several advantages over the traditional management techniques; the most important of them are [12]:

- **High degree of abstraction:** The policies are written in a high-level, abstract way, as independent as possible from the network topology, protocols, services and applications. The administrators can easily determine the behavior of the network by reading the policies, even if they were not their authors, or a long time passes. The behavior of the network is more likely to reflect the goals of the administrators, since the policies now express exactly these goals, rather than procedures that attempt to describe them. Changes in the topology of the network, its protocols, services or applications do not affect the policies, since the goals remain the same – the modifications are automatically integrated and the same policies remain applicable in the modified network.

- **Automation - Consistency:** The PBN model implies high level of automation. This automation ensures consistency in the device behavior across the network, and simplifies significantly the process of configuring the devices.
- **Dynamic policies:** In PBN, the policies are separated from the network details. This binding only takes place on the policy servers, and it is a dynamic procedure. When the network state changes, the policies are updated to reflect these changes. This allows new types of policies to be defined, and gives extra flexibility to the network managers. An example of a dynamic binding of a policy is the following: Suppose that the policy "Give high priority to engineers" has been set. Whenever an engineer logs on to a workstation, the PDP is informed of this fact and generates such configuration data for the network devices that will give high priority to the specific workstation. Such policies are very hard to implement with traditional management techniques.

3.7. PBN Protocols: COPS and Its Extensions

IETF attempts to standardize the communication between PDPs and PEPs through the ***Common Open Policy Service (COPS)*** [21] protocol and its extensions. COPS is being developed by the ***Resource Allocation Protocol (RAP)*** [22] working group. Although RAP purpose is to "establish a scalable policy control model for RSVP" [22], COPS has received significant attention from other research groups, within and outside IETF, and applications based on it have already emerged [12], [20], [23], [24], [25].

3.7.1. COPS

The policy protocol is designed to communicate self-identifying policy-related information, exchanged between the PDP and the PEP. In COPS, each PEP may have one or more clients of different client-types; different client-types exist for the different policing areas (security, QoS, admission control, accounting, etc). By supporting the appropriate clients-types, the PEP provides a way to control the various management aspects of the device.

In COPS, when a PEP boots, it connects to the PDP and its clients identify themselves by reporting their capabilities and limitations. Note that a PEP may have clients that each connects to a different PDP. In the outsourcing mode, if the PEP receives an event that it does not know how to treat, it issues a request to the PDP, asking for configuration data for this event. In the provisioning model, the clients register their capabilities to the PDP, and the PDP sends the appropriate policies (in a pre-agreed format) that the PEPs should enforce. In both cases, the PDP may update the configuration data of the PEPs. COPS also describes synchronization procedures between the PDP and the PEP, and it defines how the PEP should react if the connection to the PDP is lost. Furthermore, COPS defines mechanisms that secure and ensure the integrity of the exchanged messages.

COPS does not define the format or semantics of the exchanged configuration data; it just provides the means to exchange such data. The definition of the format and semantics of the exchanged data has to be defined per client-type in additional documents (typically developed by IETF).

3.7.1.1. COPS Message Format

All COPS messages consist of a common header and a number of objects. The header of the message (8 octets – Figure 3.3) identifies the type of the exchanged message. Ten types of messages exist [21]:

1. Request (REQ)
2. Decision (DEC)
3. Report State (RPT)
4. Delete Request State (DRQ)
5. Synchronize State Req (SSQ)
6. Client-Open (OPN)
7. Client-Accept (CAT)
8. Client-Close (CC)
9. Keep-Alive (KA)
10. Synchronize Complete (SSC)

A detailed description of COPS is out of the scope of this document. However, we would like to mention that the PEP initially sends a Request message (REQ), where it reports its capabilities and limitations and asks for configuration data. PDP decisions, solicited or not, are encapsulated within Decision messages (DEC). Report messages (RPT) are used to report the success or failure of installing the PDP decisions, and to report the usage of the policies (e.g., for accounting purposes). For more details, the reader may refer to "The COPS Protocol" from IETF [21].

At the time this document is written, COPS is an RFC, hence modifications may take place in the future.

0	1	2	3
Version| Flags	Op Code	Client-type	
Message Length			

...(COPS objects follow)...

Figure 3.3: *COPS Header Format*

3.7.2. COPS Extensions (client-types)

The IETF RAP working group has also defined already some client-types for COPS. These client-types are considered as extensions of the base COPS protocol, since they define details for the format and semantics of the configuration data that is exchanged between the PDPs and the PEPs. The most important extensions at this time are the COPS usage for RSVP [26] and COPS usage for Policy Provisioning, or COPS-PR [27]. Here, we are particularly interested in the latter, which will be described in the next paragraphs.

3.8. COPS-PR

RAP has developed ***COPS for Policy Provisioning (COPS-PR)*** [27] as an extension (or, client-type) of COPS. COPS-PR was initially biased towards DiffServ policy provisioning [28]. However, it appears to be suitable for several other management areas (accounting [29], IP filtering [30], [27], security [31], etc. [32], [33], [34]).

As its name implies, COPS-PR operates only in a provisioning style, where the PDP downloads all the relevant policies in its PEPs, and the latter serve all incoming events according to these policies. In COPS-PR, the clients connect to the appropriate PDP (different PDPs may control different clients in a single PEP), report their capabilities and limitations, and request the initial policies to be downloaded to them. The PDP processes the request of each client and, according to the global policies and network state, generates and downloads the appropriate configuration data. If the network state or the policies change afterwards, the PDP may decide to update these configuration data, in order to keep the behavior of the managed device consistent.

3.8.1. The Policy Information Base (PIB)

In COPS-PR, each client has to maintain a special database, called ***Policy Information Base (PIB)*** [35], where it stores all the received configuration data. The PIB is a structure similar to a MIB, and can be described as a conceptual tree namespace, where the branches represent structures of data, or Provisioning Classes (PRCs), and the leaves represent instances of these classes, called Provisioning Instances (PRIs). PIBs are defined by COPS-PR only as abstract structures; the details of each PIB (PRCs and their semantics) are specified in separate standard documents (such as internet-drafts or vendor private documents). Different PIBs are defined in order to cover the various management areas (Differentiated Services, accounting, security etc). PIBs are defined in a high abstraction level; in this way they hide the details of the underlying hardware and provide to the PDP a unified way to control the behavior of the devices, over a specific management area, across the entire network.

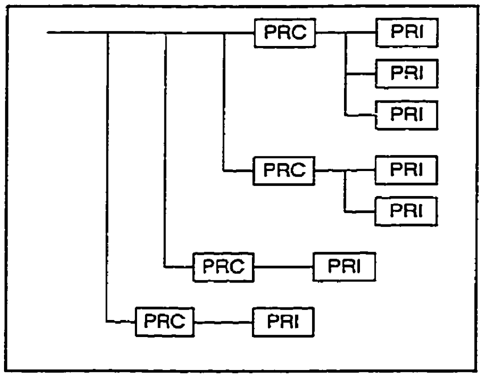


Figure 3.4: *PIB structure*

PRIs are identified within the PIB through a PRI identifier (PRID). The PDP can install or update PRIs by sending an install decision specifying the appropriate PRIDs and their values, or remove PRIs with a remove decision containing the PRIDs of the PRIs to be removed. Policies are formed as a set of PRIs in the PIB; by adding or removing PRIs, the PDP can implement the desired policies, which will be enforced at the device.

It is important to highlight that the policies that each PIB can implement are predefined (in the standard documents that define this PIB). In order to control a device, the PDP has to map the high-level network policies and the network state into policies that can be implemented in the PIB of the PEP.

The Framework Policy Information Base [35] defines a PIB with classes that are common to all PIBs. This PIB should be implemented by all COPS-PR clients.

3.8.2. The Structure of Policy Provisioning Information

PIBs are defined using the ***Structure of Policy Provisioning Information (SPPI)*** specification [36]. Since PIBs resemble MIBs, SPPI is based on the SMI (Structure of Management Information) [38]. Although describing the SPPI in full is out of the scope of this document, we will attempt to give an overview of the most important characteristics that will be used later on.

PIBs are constructed as a tree of PRCs, with PRIs as leaves. The entire tree is under a single, root PRC, with a specific identifier (PRID), usually assigned by IANA (for public PIBs) or the vendor (for private PIBs).

Two types of PRCs exist. The first type is PRCs that group other PRCs. Such PRCs are represented as intermediate nodes in the tree, without having any leaves directly attached to them. Each PRC is described as a table with defined columns-attributes. Each attribute has a specific semantic and type. Each row of the table is a PRI of the specific class. Hence, by defining the column of the table, the attributes and the semantics of the PRIs are defined.

It is important to distinguish the definition of the PIB from its actual data. The definition of the PIB includes the definition of the classes (tables) and their organization into groups. On the other hand, the PRIs are the instances of these classes, and comprise the actual data that is placed in the PIB.

3.8.3. COPS-PR Example

We shall use a small filtering PIB in order to show how COPS-PR works. The network of our example is the network of a small company (Figure 3.5), with the following topology:

- LAN address range: X.Y.0.0/16
- Subnets X.Y.1.0/24 (public), X.Y.2.0/24 (administrators), X.Y.3.0/24 (employees)
- A central router A that routes the LAN and Internet traffic, and serves as the Internet gateway.

Suppose that the following high-level abstract access rules have been set:

#1. Internal LAN traffic is always allowed

#2. The administrator can always access the Internet, whenever and from wherever he/she is logged in.

#3. During overall congestion, traffic between the employee domain and the Internet is denied.

#4. Internet can be accessed only during working hours (Monday to Friday, 9:00-17:00)

(Rule #1 has the highest priority, rule #4 the lowest)

Also, suppose that the term "overall congestion" is evaluated according to whether router A is congested, i.e., based on the load of its interfaces.

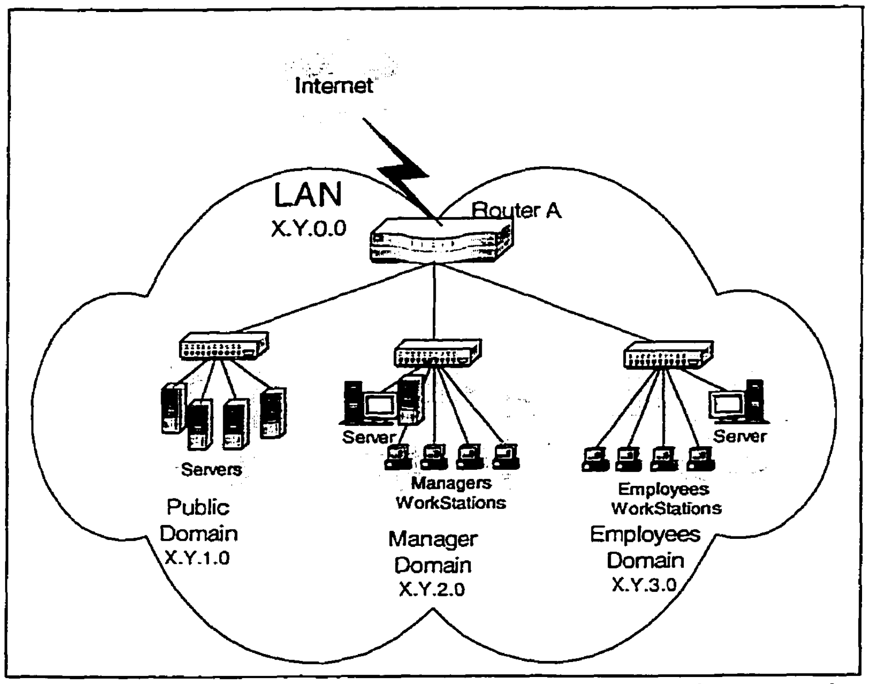


Figure 3.5: *The topology of the company example network*

Suppose that the (PEP of the) routers of the network support a PIB with a single PRC. PRIs of this PIB describe source/destination criteria that allow access to IP traffic within the network. Each PRI in this PIB is a stand-alone policy of the form:

If ((Source matches Srcaddr/Srcmask) and (Destination matches Destaddr/Destmask)) then allow

Traffic that matches at least one PRI in the PIB is allowed. Traffic that does not match any criteria (policies in the PIB) is, by default, denied.

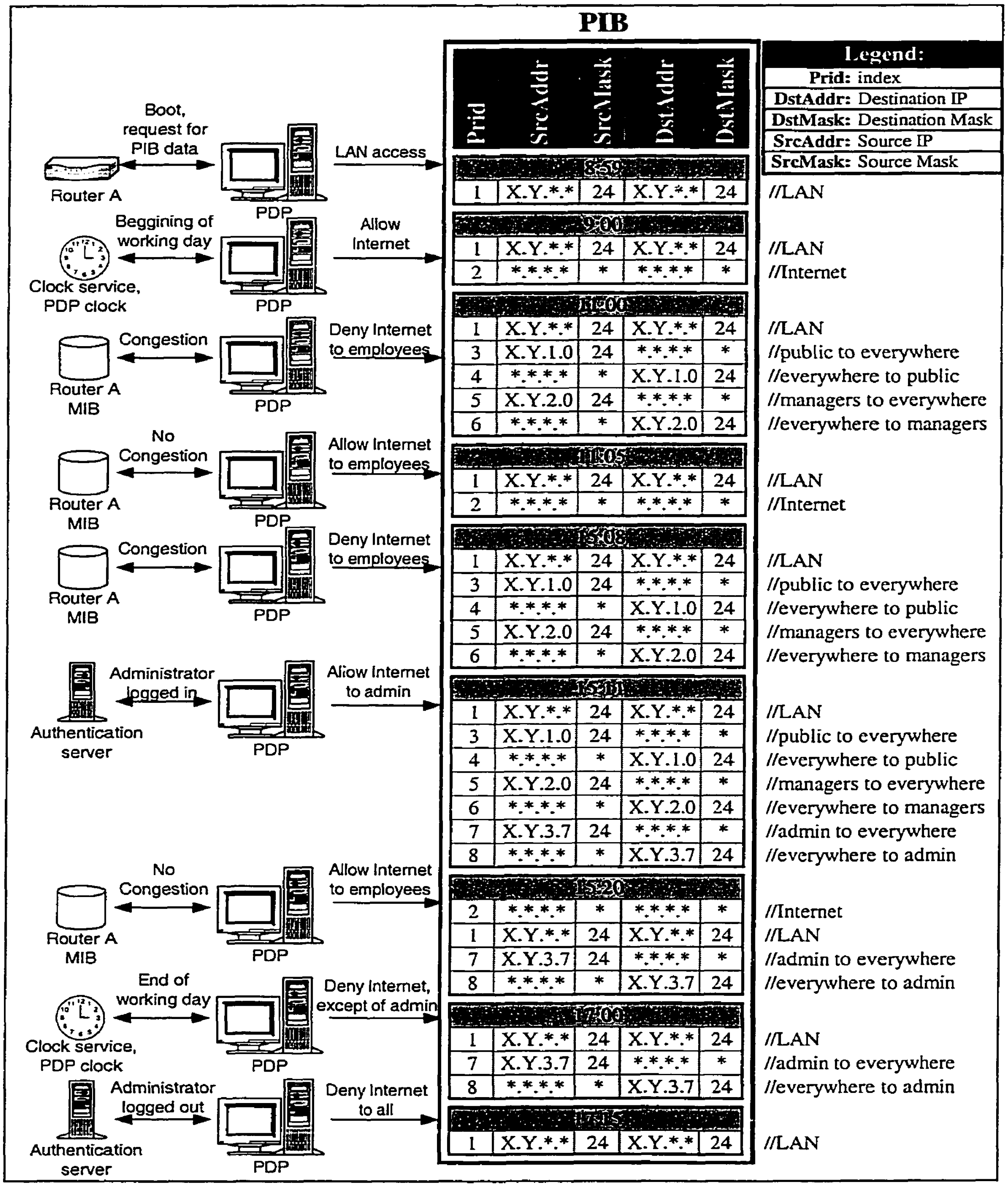
Suppose now that the following events take place:

08:59: *No administrator logged on*	***15:11:*** *administrator logs on at X.Y.3.7*
09:00: *start of working day*	***15:20:*** *no congestion*
11:00: *congestion detected*	***17:00:*** *end of working day*
11:05: *no congestion*	***17:15:*** *administrator logs out*
15:08: *congestion detected*	

Figure 3.6 demonstrates snapshots of the PIB of Router A during the day: When the router boots, the PDP sends a policy that allows all LAN traffic (PRID #1), which implements policy #1. When the PDP detects the beginning of the working day (09:00), policy #4 becomes applicable, and a PRI that allows traffic to/from the Internet is added into the PIB (PRI #1 is now redundant; the PDP may decide to keep it or not; however this does not affect significantly our analysis). When congestion is detected (11:00), the PDP attempts to install policy #3. This policy is in conflict with the already installed policy #4; however policy #3 has higher priority, and hence the employee subnet is banned from Internet traffic. After a while (11:05), the network is no longer congested, and the PIB is restored to its previous state. When the network becomes congested again (15:08), the PIB has to be updated once more, as before. When the administrator logs on at the guest subnet, however (15:11), traffic to/from the Internet to his/her IP is allowed. Note that policy #2 is in conflict with policy #3, which bans traffic to the employee subnet, however the former wins since it has a higher priority. When the network becomes decongested (15:20), policy #3 is uninstalled, and policy #4 is installed again. At the end of the working day (17:00), policy #4 is also uninstalled, and finally, when the administrator logs out, policy #2 is uninstalled as well, denying all Internet access.

3.9. Conclusion

This section introduced Policy-Based Networking and outlined the COPS Protocol and its extension for policy provisioning (COPS-PR). A simple example demonstrated how a PDP controls a COPS-PR PEP (and consequently the behavior of the device) by modifying the configuration data stored in its PIB. Despite its simplicity, this example is sufficient to reveal some shortcomings of COPS-PR. The next chapter presents these shortcomings, discusses how they motivated our work, and presents the concept of meta-policies, which is our proposal to overcome these deficiencies.



Legend:	
Prid:	index
DstAddr:	Destination IP
DstMask:	Destination Mask
SrcAddr:	Source IP
SrcMask:	Source Mask

Prid	SrcAddr	SrcMask	DstAddr	DstMask	
8:59					
1	X.Y.*.*	24	X.Y.*.*	24	//LAN
9:00					
1	X.Y.*.*	24	X.Y.*.*	24	//LAN
2	*.*.*.*	*	*.*.*.*	*	//Internet
11:00					
1	X.Y.*.*	24	X.Y.*.*	24	//LAN
3	X.Y.1.0	24	*.*.*.*	*	//public to everywhere
4	*.*.*.*	*	X.Y.1.0	24	//everywhere to public
5	X.Y.2.0	24	*.*.*.*	*	//managers to everywhere
6	*.*.*.*	*	X.Y.2.0	24	//everywhere to managers
11:05					
1	X.Y.*.*	24	X.Y.*.*	24	//LAN
2	*.*.*.*	*	*.*.*.*	*	//Internet
15:08					
1	X.Y.*.*	24	X.Y.*.*	24	//LAN
3	X.Y.1.0	24	*.*.*.*	*	//public to everywhere
4	*.*.*.*	*	X.Y.1.0	24	//everywhere to public
5	X.Y.2.0	24	*.*.*.*	*	//managers to everywhere
6	*.*.*.*	*	X.Y.2.0	24	//everywhere to managers
15:10					
1	X.Y.*.*	24	X.Y.*.*	24	//LAN
3	X.Y.1.0	24	*.*.*.*	*	//public to everywhere
4	*.*.*.*	*	X.Y.1.0	24	//everywhere to public
5	X.Y.2.0	24	*.*.*.*	*	//managers to everywhere
6	*.*.*.*	*	X.Y.2.0	24	//everywhere to managers
7	X.Y.3.7	24	*.*.*.*	*	//admin to everywhere
8	*.*.*.*	*	X.Y.3.7	24	//everywhere to admin
15:20					
2	*.*.*.*	*	*.*.*.*	*	//Internet
1	X.Y.*.*	24	X.Y.*.*	24	//LAN
7	X.Y.3.7	24	*.*.*.*	*	//admin to everywhere
8	*.*.*.*	*	X.Y.3.7	24	//everywhere to admin
17:00					
1	X.Y.*.*	24	X.Y.*.*	24	//LAN
7	X.Y.3.7	24	*.*.*.*	*	//admin to everywhere
8	*.*.*.*	*	X.Y.3.7	24	//everywhere to admin
17:15					
1	X.Y.*.*	24	X.Y.*.*	24	//LAN

Figure 3.6: Instances of the PIB of router A

Chapter 4. Meta-Policies in COPS-PR

4.1. COPS-PR Shortcomings

The previous example demonstrates how the COPS-PR protocol is used in order to communicate policing information between a PDP and a PEP, and how a PIB is used by the latter in order to store this information. However, this example also reveals some shortcomings of this model.

In COPS-PR, the high-level policies are reflected into the PIBs of the devices. PRIs are installed in or removed from the PIB according to the current (network) state. When various events take place, the state changes and the PIB is modified. Of course, the occurrence of the same event more than once may lead to different PIB contents. (For example, the end of congestion at 11:05 and 15:20 results in different PIB instances.) The occurrence of the same events does not even imply that the PDP will send exactly the same commands to the PEP. However, there is a certain correlation between the network events and the PIB contents, which this model fails to take into consideration.

This shortcoming of COPS-PR has a great impact on its efficiency and performance. In several cases the PDP has to send the same (or similar) commands, when the same event occurs. In the previous example, for instance, while the network alternates between the states "congested" and "not congested", the PDP needs to install and remove the PRIs that deny Internet access to the employee domain. In a more complex example, a big set of PRIs might need to be updated. The PEP needs to be directed about how to treat an event, even if this event has occurred several times in the past. Hence, more PDP resources (to regenerate the policies each time) and more bandwidth (to send them) are consumed, than necessary.

A second limitation lies in the rigidity of the PIBs. PIBs are predefined structures, and the high-level policies cannot directly map into them. The PDPs need to dynamically project the high-level policies into policies that can be represented in the PIB. All policies that do not precisely map to a supported policy type need to be processed at the PDP level. In the previous example, the policy "During overall congestion, traffic between the employee domain and the Internet is denied" cannot fit into the PIB, and has to be processed by the PDP. The latter, depending on the overall network state, produces the PRIs that are in conformance with the initial policy, for the given congestion status. Then, the PEP implements the policies that these PRIs describe. In this case, the high-level policy has to be processed partially by the PDP, and partially by the PEP. Obviously, the involvement of the PDP in cases like this is usually neither efficient nor desired. For the previous policy, for example, the PDP needs to query the MIB of router A in order to determine if there is congestion; then send the appropriate policies back to the router's PIB. Obviously, this policy could be entirely processed at the PEP-level, since congestion could be evaluated locally by the PEP. Similarly, for the policy "The Internet can be accessed only during working hours", the PDP is necessary in order to determine the condition "working hours", since this condition cannot be stored in the PIB of the router. However, supposing that there is a clock service that broadcasts the date and time over the network, this policy could also be evaluated entirely at the PEP-level. The rigidity of the PIBs, though, does not allow any other kind of policies to be evaluated by the PEP apart from these supported by the PIB, making in this way the presence of the PDP necessary, even in cases where this could be avoided. This is a significant drawback, since it makes the model very vulnerable to PDP errors or malfunctions and to network error situations, such as network congestion or network failures.

4.2. Motivation

The previously discussed limitations motivated our work: The intelligence of the COPS-PR model seems to be concentrated at the PDP level. PDP decisions always download policies into the PEP, even when the same events reoccur. The PIB is a rigid structure that allows only limited types of policies to be pushed into the PEP. The PEP depends on the PDP presence, even in cases where this is not absolutely necessary.

This work attempts to extend the policy functionality of the PIB, so that the PEP will be able to take more decisions simply by examining events. Initially, the PDP downloads the applicable policies and directs the PEP how to react on certain events. Apart from that, the role of the PDP is downgraded mainly to communicating such events to the PEP, rather than modifying the configuration data. Also, the PEP can be programmed to monitor some of these events by itself and initiate the appropriate actions.

Assuming this extended functionality, the PDP is able to control the PEP mainly by communicating events, rather than policies. Also, the PEP is able to take certain policing decisions by itself. In this way, intelligence is pushed towards the PEP. From a different point of view, this work pushes some of the PDP functionality inside the PEP.

In order to achieve the described functionality, we use ***meta-policies***, a concept which is defined and discussed in this chapter.

4.3. The Concept of Meta-Policies

In the example of the previous chapter, there was the policy:

During overall congestion, traffic between the employee domain and the Internet is denied.

Suppose that this is the only policy of a small network, consisting of two routers, A and B, where router A is the central router of the network, and B a router of a sub-domain. Also, suppose that these routers have a small filtering PIB like the one examined before, and that the condition "overall congestion" is indicated through some MIB variables of router A.

Whenever congestion is detected, the PDP sends to the PEPs of the routers some configuration data that install some PRIs and update their behavior. Since we have only one policy for this network, each router receives the same commands each time that congestion is detected. Let us call these data DataA and DataB. These PRIs are uninstalled when congestion ends.

Suppose now that the PDP could send to the two routers the following commands, which we shall call ***meta-policies***:

Router A:	Router B:
• *If (Congestion) then {DataA}*	• *If (Congestion) then {DataB}*

Finally, suppose that the PDP somehow directs the PEP of router A on how to evaluate the parameter "Congestion" from the appropriate variables of its MIB and informs the PEP of router B that the value of "Congestion" will be sent to it, each time that it changes. In this case, we can observe the following:

- The PDP only needs to send the meta-policies once. Then the PEPs have all the necessary information to react according to current network state, as long as they are informed about it somehow.
- Router A can evaluate the two meta-policies locally and independently of the PDP. This means that the PDP does not need to process the original policy for router A any more. Also, the PEP will operate according to the administrative goals even in cases of high congestion (that would delay the PDP from querying the MIB of router A and update its PIB), or even while the PDP is down or unreachable.

- Router B still needs to be guided by the PDP. However, the PDP does not need to send policy commands in the form of configuration data (DataB) anymore; it must send only the value of the variable "Congestion". In this way, the PDP load is decreased, less bandwidth is consumed, and the PDP Decision message is less likely to get lost or corrupted (since it is significantly smaller).

Although the case of a network with more than a single policy complicates the situation, based on the previous discussion, we can observe that in general, each high-level policy requires some specific PRIs to exist (or not exist) in the PIB of each device, depending on the network state. Each network event makes applicable some policies that were not applicable before and vice-versa. This means that we can associate combinations of events with PRIs that need to exist in the PIB.

Meta-policies attempt to take advantage of exactly this observation. They associate combinations of network events with PRIs that need to be installed. The event combination comprises the ***condition*** of the meta-policy; the modifications of the PIB that these events trigger are its ***actions***. Meta-policies are generated by the PDP and they are sent to the PEP. The PEP processes these meta-policies and updates its PIB. The decision that the PEP takes is the same that the PDP would take, for the same network events. Of course, in order to do so, the PEP must be aware, somehow, of all the relevant network events. The PDP could be used for this purpose and inform the PEPs about network events that need a global (or at least a relatively "large") network view to be evaluated. In this case, the PEP still depends on the PDP, but less network and PDP resources are consumed. However, the PEP can be informed of network events from other sources, as well: For instance, the PEP may use the MIB of the device where it resides to evaluate local events. A network service or server (like a clock or a notification service) can also be used. Even more, mobile agents can be used to collect and provide notification of such events. The latter implies some degree of programmability and openness at the architecture of the PEP; however, such features are becoming available more and more in modern devices.

An important issue that needs to be addressed is conflicts. Valid meta-policies may be conflicting under certain circumstances. Besides, meta-policies may conflict with PRIs directly installed into the PIB by the PDP. As in COPS-PR, the PDP must resolve these conflicts before sending any commands to the PEP. Conflicts between meta-policies can also be resolved at the PEP level, as long as these policies are associated with priorities., provided by the PDP.

Finally, note that the mapping between meta-policies and high-level policies is not necessarily one to one. Some high-level policies may not be applicable for a device; some may be combined into a single meta-policy; and some others may need to be split into more that one. Besides, the PDP may still decide not to produce a meta-policy for a high-level policy, and implement it by directly installing and uninstalling PRIs into the PIB.

4.4. Formal Definition

We define a meta-policy as a rule of the form:

if (condition) then {actions}

where *"condition"* is a logical expression, e.g., "(C>80%) and (D=true)",
and *"actions"* is a set of PIB commands that install PRIs into the PIB.

Since the actions encode a specific policy, this rule is a rule on how policies are enforced; this is why it is called a "meta-policy".

Each meta-policy is generated for a specific PEP, according to its capabilities, limitations and the device on which it resides; hence it is meaningful only for this PEP.

Meta-Policies are generated by the PDP and consumed by the PEP. The PEP evaluates the condition of each meta-policy, and when it evaluates true, it enforces the actions. The key idea in meta-policies is that the PEP can store and process these meta-policies without knowing their complete semantics: The condition is treated as a logical expression; the actions, pre-generated by the PDP, just denote PRIs that must be installed, something that can be performed by the PEP without understanding policies they implement. In this way, the PEP can process any meta-policy, independently of its complexity and its meaning.

Also, each meta-policy must be assigned a priority. This priority is used by the PEP in order to resolve any conflict between two meta-policies that may need to be activated at the same time, but have conflicting actions.

Both the condition and the actions may contain parameters (such as "Congestion" or "WorkTime"); the values for these parameters are either sent by the PDP or evaluated by the PEP, according to directions provided by the PDP. The parameters that a meta-policy uses must be installed by the PDP prior to installing the meta-policy.

4.4.1. Parameters

The parameters are used in meta-policy conditions in order to determine when a meta-policy must be activated. Moreover, they are used by meta-policy actions in order to dynamically bind the network state within policies. For instance, in the previous example we could have a meta-policy "*if (AdminLogged) then {install (7, AdminIP, 24, *.*.*.*, 24,), install (8, *.*.*.*,24, AdminIP, 24) }*", which installs the PRIs that give to the administrator access to the entire network. This meta-policy contains two parameters: AdminLogged and AdminIP.

When installing a parameter, the PDP must also specify an evaluation method for it. For instance, the PEP can be directed to get a value for a parameter from the MIB of the device. Or, the PDP could provide the value for this parameter. However, other methods are also possible, depending on the capabilities of the device, such as to download and execute a script, use mobile agents, or get the desired information from some server or service (e.g., clock service).

4.5. Example

Consider the company example that we studied before. We shall examine how it is affected by meta-policies.

First of all, the policy #1, "*Internal LAN traffic is always allowed*", must always be enforced. Hence, the PDP directly enforces this policy by installing the PRI #1 into the PIB (Figure 4.1), when the router boots.

In addition, the PDP downloads to the PEP the following meta-policies:

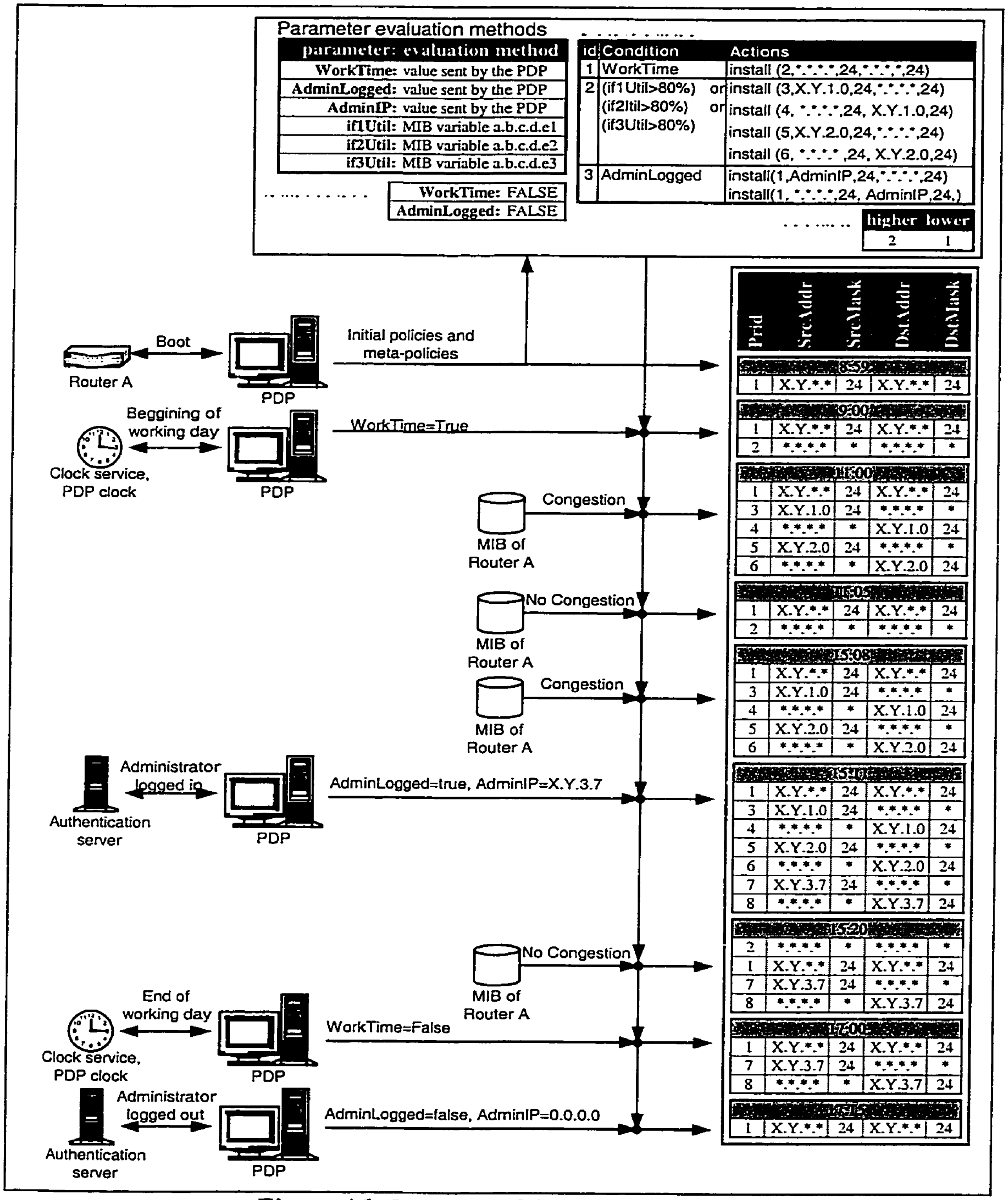
- *if (WorkTime)* ***then*** *{install (2,*.*.*.*,24,*.*.*,*,24)}*
- *if ((if1Util>80%) or (if2Util>80%) or (if3Util>80%))* ***then*** *{*
  install (3,X.Y.1.0,24,.*.*.*,24), install (4, *.*.*.*,24, X.Y.1.0,24)*
  install (5,X.Y.2.0,24,.*.*.*,24), install (6, *.*.*.* ,24, X.Y.2.0,24)*
  }
- *if (AdminLogged)* ***then***
  {install(1,AdminIP,24,.*.*.*,24), install(1, *.*.*.*,24, AdminIP,24,)}*

and informs the PEP that the two first meta-policies are conflicting, and the second one has higher priority.

Since the meta-policies contain parameters, the PDP also has to inform the PEP of the evaluation method for these parameters. In our example, the PDP sends the values of the parameters "WorkTime", "AdminLogged" and "AdminIP", and it directs the PEP to evaluate by itself the parameters "if1Util", "if2Util", "if3Util" through the appropriate MIB variables that denote the usage of the router's interfaces (Figure 4.1).

The PEP monitors the parameters, and when their values change, it re-evaluates the affected conditions. While the condition of a meta-policy is met, the corresponding PRIs are installed in the PIB. In this way, the PIB always contains the appropriate PRIs that implement the desired behavior.

Meta-policies allow the PDP to download initially the applicable policies and meta-policies and then, control the PEP mainly by reporting network events. Moreover, some of these events can be monitored by the PEP itself, without the involvement of the PDP. Note that such events do not have to be local; the PEP can be programmed (e.g. by downloading and executing some scripts, or through mobile agents) to monitor such events through another server or service: for instance, the parameter "WorkTime" could have been monitored by the PEP through a network clock service, without the involvement of the PDP.



parameter: evaluation method
WorkTime: value sent by the PDP
AdminLogged: value sent by the PDP
AdminIP: value sent by the PDP
if1Util: MIB variable a.b.c.d.e1
if2Util: MIB variable a.b.c.d.e2
if3Util: MIB variable a.b.c.d.e3

WorkTime: FALSE	
AdminLogged: FALSE	

id	Condition	Actions
1	WorkTime	install (2,*.*.*.*,24,*.*.*,*,24)
2	(if1Util>80%) or (if2Itil>80%) or (if3Util>80%)	install (3,X.Y.1.0,24,*.*.*.*,24) install (4, *.*.*.*,24, X.Y.1.0,24) install (5,X.Y.2.0,24,*.*.*.*,24) install (6, *.*.*.* ,24, X.Y.2.0,24)
3	AdminLogged	install(1,AdminIP,24,*.*.*.*,24) install(1, *.*.*.*,24, AdminIP,24,)

higher	lower
2	1

Prid	SrcAddr	SrcMask	DstAddr	DstMask
8:59				
1	X.Y.*.*	24	X.Y.*.*	24
9:00				
1	X.Y.*.*	24	X.Y.*.*	24
2	*.*.*.*	*	*.*.*.*	*
11:00				
1	X.Y.*.*	24	X.Y.*.*	24
3	X.Y.1.0	24	*.*.*.*	*
4	*.*.*.*	*	X.Y.1.0	24
5	X.Y.2.0	24	*.*.*.*	*
6	*.*.*.*	*	X.Y.2.0	24
11:05				
1	X.Y.*.*	24	X.Y.*.*	24
2	*.*.*.*	*	*.*.*.*	*
15:08				
1	X.Y.*.*	24	X.Y.*.*	24
3	X.Y.1.0	24	*.*.*.*	*
4	*.*.*.*	*	X.Y.1.0	24
5	X.Y.2.0	24	*.*.*.*	*
6	*.*.*.*	*	X.Y.2.0	24
15:10				
1	X.Y.*.*	24	X.Y.*.*	24
3	X.Y.1.0	24	*.*.*.*	*
4	*.*.*.*	*	X.Y.1.0	24
5	X.Y.2.0	24	*.*.*.*	*
6	*.*.*.*	*	X.Y.2.0	24
7	X.Y.3.7	24	*.*.*.*	*
8	*.*.*.*	*	X.Y.3.7	24
15:20				
2	*.*.*.*	*	*.*.*.*	*
1	X.Y.*.*	24	X.Y.*.*	24
7	X.Y.3.7	24	*.*.*.*	*
8	*.*.*.*	*	X.Y.3.7	24
17:00				
1	X.Y.*.*	24	X.Y.*.*	24
7	X.Y.3.7	24	*.*.*.*	*
8	*.*.*.*	*	X.Y.3.7	24
17:15				
1	X.Y.*.*	24	X.Y.*.*	24

Figure 4.1: *Instances of the PIB of router A*

Chapter 5.

Requirements and Design

The previous section introduced the concept of meta-policies and demonstrated how these can be used to extend the functionality of the PEP. This section analyses the requirements, justifies our choice to use a PIB to implement the additional functionality and discusses design issues of the PIB.

5.1. Early Requirements

5.1.1. General Requirements

The central concept in our work is meta-policies, i.e., rules of the form *"if (conditions) then {actions}"*.

Each condition is a Boolean expression, comprised of a number of simpler conditions. Ultimately, all conditions are decomposed into primitive logical expressions, such as arithmetic comparisons (X+Y>10), Boolean expressions (Congestion==True) network expressions (IP matches X.Y.Z.W), etc.

The actions install PRIs into the PIB. Each action identifies a single target PRI and the value that must be installed into it.

Both conditions and actions may be parametric; hence a way to communicate, store and process parameters is also necessary. Each parameter has a type, which denotes what kind of information it stores (integer, IP address, octet string, etc). Also, each parameter has a way to be evaluated. Several evaluation methods may exist. We distinguish two basic evaluation methods: First, a parameter can get its value from the MIB or PIB of the device. Second, the value can be sent by the PDP initially, and then be updated (by the PDP) whenever it changes. However, other evaluation methods may also exist, depending on the capabilities of the device. For instance, an active/programmable device may download and execute code that will evaluate this parameter. Although it is practically impossible to support any possible evaluation method, it is desirable that the basic methods that we define can be extended with other methods (standard or vendor-specific).

5.1.2. Why a PIB?

The proposed enhancements require meta-policing information to be exchanged between the PDP and the PEP, and be stored and processed by the latter. Hence, a crucial question that must be tackled in the early design phase is what protocols and data structures will be used. We decided to use COPS-PR to communicate such data and define a PIB to store them at the PEP (as opposed to defining another protocol and/or storage structure, or extending the existing ones). This decision was based on a number of reasons:

- Meta-policies need to be sent to the PEP in a provisioning style, and COPS-PR is a protocol defined for policy provisioning.
- Our work is in line with the work conducted in IETF. No new protocols need to be developed, and the proposed PIB can easily be adapted by the Internet community (researchers and vendors). Even legacy devices can support the proposed PIB (e.g., with software updates).

- By using a PIB to store meta-policies, meta-policing data are treated as any PIB data. Consequently, meta-policies on meta-policies could also be defined (this is discussed at the last chapter, as future work).
- Finally, by using COPS-PR and PIBs, the design and the implementation is simplified: the definition of a PIB is much simpler that defining a new protocol. Meta-policy exchange and storage is already handled by the protocol and does not need to be addressed by us. The implementation is based on existing, tested tools. The reuse of knowledge and code makes the design, implementation and testing safer and easier, and minimizes the chance for errors.

In general, although the choice of using a PIB and COPS-PR introduces some further requirements, it does not prevent or hinder us from meeting any of our goals.

5.1.3. COPS-PR/PIB Requirements

Our decision to define a PIB and use COPS-PR to implement our proposal implies that the SPPI specification must be used to define the PIB. SPPI [36] demands all data to be placed in tabular format (each table is a PRC, and the rows of the tables the PRIs). SPPI also demands strong typing of the attributes of the PRIs. However, the SPPI is very flexible in defining new types; this feature is exploited in order to overcome the previous restriction.

5.2. Analysis

5.2.1. Communication and storage

By choosing to define a PIB and use COPS-PR, all communication and storing is addressed by the protocol itself: When the PEP connects to the PDP, it reports its meta-policing capabilities and limitations. According to these capabilities and limitations, the PDP downloads all the appropriate meta-policies. These meta-policies are stored in the PIB and remain there, until they are updated by the PDP.

5.2.2. Meta-Policing Data

5.2.2.1. Meta-policies

Meta-policies consist of a condition and a set of actions. Since valid meta-policies may conflict under certain circumstances, the PDP must be able to declare potentially conflicting meta-policies and denote priorities between them. Also, the status of the meta-policies (i.e., whether they are active, whether they suppress a meta-policy with lower priority or whether they are suppressed) may need to be reported to the PDP.

5.2.2.2. Conditions

Each meta-policy must contain exactly one condition. As mentioned before, the condition is decomposed into one or more primitive expressions that need to be evaluated. Each of these primitives must contain at least one parameter (otherwise, a simpler condition without them exists, since that primitive expression always evaluates either true or false). We distinguish two categories of primitives: Boolean and generic expressions.

Boolean expressions are a subset of the generic expressions, but due to their simplicity and commonality, they are treated separately. Such primitives are evaluated according to the value of a Boolean parameter. For instance, in the expression *((X>Y) && (!Congestion) && (WorkTime))*, *Congestion* and *WorkTime* are such primitives.

Generic expressions contain all the other logical expressions that cannot be decomposed into simpler Boolean primitives. Examples of such primitives are "X>Y", *"IP matches X.Y.Z.W" or "8:00am < time < 5:00pm).* Each PEP can only support specific types of such expressions (e.g., arithmetic), which are reported along with the other PEP capabilities to the PDP. The PDP can only send to the PEP expressions that are supported by the latter.

An important issue is that such expressions need to be standardized in order to be transmitted and stored in the PIB. However, different types of expressions require different operators (e.g., arithmetic expressions need operators like "+","-",">", while network conditions need operators such as "matches" and "subnet"). Besides, the set of types of such expressions is infinite, since any kind of expressions may be valid: the expression "colorl darker that color2" is a valid expression (although probably totally useless for network management). The point is that all types of possible expressions, cannot be predicted in advance, but they need to be standardized. Of course, we could choose to standardize only a few types of expressions that are most commonly used, but this would restrict the applicability of our work.

The solution given to this problem was to define an open, generic mechanism to handle such expressions. The details of this generic mechanism can be defined per expression type (arithmetic, IP expressions, etc). Common expression types have already been defined by us, but these types can easily be extended to include other ones, as well.

More specifically, all expressions are encoded using XML. XML uses tags that give semantics to the data of the XML document. However, the semantics of these tags are defined in separate documents, called Document Type Definitions (DTDs). These DTDs specify the details of the generic mechanism, per expression type. Each PEP reports to the PDP the DTDs that it supports, through an identifier, which uniquely identifies these DTDs (which is the URL where these are published; this is the standard method adapted by the XML standard). By reporting an XML DTD, the PEP declares that it can interpret any XML document (that encodes an expression) written according to this DTD. For example, if a DTD defines tags for numerical operations (+,-,*,/,div) and comparisons (>,=,>=,<,<=,=) then the PEP should be able to understand any arithmetic expression that uses these operators. The PDP, according to the expression that it wants to encode, chooses the most appropriate DTD, encodes the condition and transmits it.

By using XML DTDs we manage to:

- Standardize the exchange of general expressions
- Accomplish a uniform way of storing them into the PIB
- Leave the PIB open to any type of expressions
- Allow each PEP to implement only the functionality that it needs, or that is appropriate, according to its resources.

Note that the PDP is always able to find a way to send an expression, even if this is not optimal: Even if the appropriate DTD is not supported, the expression may be transformed to a supported one. In the worst case, the entire expression is represented as a Boolean parameter, and the PDP sends the value for this parameter.

5.2.2.3. Actions

Each meta-policy is associated with a set of actions. Normally, this set should contain at least one action (a meta-policy without actions is useless; however this situation may exist while the PDP temporarily deactivates or updates a meta-policy). Each action is a binding of a PRID pointing to a single PRI, and the value that must be installed into it. This value can be either static or dynamically evaluated through a parameter.

5.2.2.4. Parameters

Each parameter may be used by the conditions or actions of one or more meta-policies. Parameters that are not referenced by any meta-policy may also exist (although this situation should only be temporal).

Each parameter must be associated with an evaluation method. At least two evaluation methods must be available: Through the MIB or PIB of the device, or through the PDP. However, the vendor should be able to extend these methods. Parameters that are evaluated through the MIB need to be associated with the frequency that the MIB must be polled, to update the value. Obviously, the MIB OID has to be provided, as well. Parameters that are evaluated by the PDP must maintain the last value sent by the PDP.

Chapter 6. The Meta-Policy Information Base

This section defines the Meta-Policy PIB classes and discusses the operation of the PEP and the PDP. The full version of the document that defines the PIB is presented as Appendix A.

6.1. PIB Definition

According to the previous analysis, we defined the classes (tables) that comprise the Meta-Policy PIB. The PIB is defined according to the IETF specifications (i.e., using SPPI).

The PIB is divided into five groups:

- ***The Capabilities Group*** contains the Provisioning classes (PRCs) that store the capabilities and limitations of the PEP (as far as the meta-policy PIB is concerned). The PRIs of these classes are reported to the PDP when the PEP connects.
- ***The Base Meta-Policy Group*** contains the classes that form the meta-policies, define their relative priority in case of conflicts, and report their status.
- ***The Condition Group*** provides classes for forming the conditions of the meta-policies.
- ***The Action Group*** includes the PRCs that define the actions of the meta-policies.
- ***The Parameter Group*** contains the PRCs where the parameters and their evaluation methods are stored.

6.1.1. The Capabilities Group

This group contains a single class, the xmlDTD class. This contains the XML DTDs that the PEP supports, for encoding expressions. Each row of the xmlDTDTable consists of an identifier and the DTD URL. The rows of this table are reported to the PDP in the REQ message.

6.1.2. The Base Meta-Policy Group

This group contains three classes: the metaPolicy, the metaPolicyStatus and the metaPolicyPriority classes.

The metaPolicy class is the PRC where meta-policies are constructed. Each instance of this class represents exactly one meta-policy. The meta-policy comprises an identifier, a name, a reference to a condition (in the condition class, described later) and an action tag. The action tag identifies a group of actions from the action class (also described later), which must be executed when the meta-policy is activated.

The metaPolicyStatus class is a PRC that AUGMENTS the previous class (AUGMENTS is an SPPI term that means that there is a one to one correspondence between the instances of these classes). Each PRI of this PRC reports whether the corresponding meta-policy is active, and whether it is suppressed by another meta-policy with higher priority or it suppresses a meta-policy with lower priority. This class is used to report to the PDP the meta-policy status. However, its PRIs can also be used as PIB parameters for other meta-policies, so as to construct conditions that are based on whether installed meta-policies are active, inactive or suppressed.

Finally, the metaPolicyPriority class reports conflicting meta-policies and direct the PEP how to resolve these conflicts. Each PRI identifies two meta-policies, and defines which one has the higher priority. PRIs with two simultaneously active meta-policies must not exist in this table.

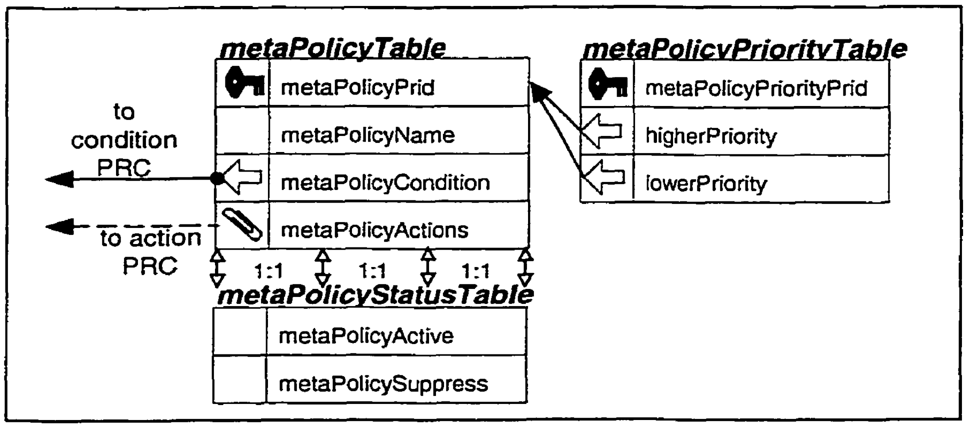


Figure 6.1: *The Base Meta-Policy Group classes*

6.1.3. The Condition Group

This group contains four classes: the condition, the complexCondition, the booleanCondition and the generalCondition classes.

The condition class is the base PRC of this group. Each PRI represents a logical expression and consists of an identifier and an attribute that indicates whether the result of the evaluation should be logically inverted. PRIs of this table must always be associated with PRIs of another class that EXTENDS the base one. (EXTENDS is also an SPPI term, that means that the PRI of this PRC can only exist as extensions of a PRI in the base PRC. These PRIs are referenced through the identifier of the base PRI, and if the latter is uninstalled, the former is uninstalled as well).

Some (but not all) of the rows of this table are used in order to represent conditions of meta-policies. As explained before, a condition may comprise several simpler conditions, which are also stored as PRIs in this table.

In order to break down a condition into simpler ones, the complexCondition class is used. This class EXTENDS the base condition class. Each PRI consists of two references to the condition class, and an operator. The references reference two logical conditions, and the operator defines a logical operation between these two conditions. In this way, the PRI in this table defines a more complex condition. Obviously, the PDP must not install rows that reference themselves, directly or interectly.

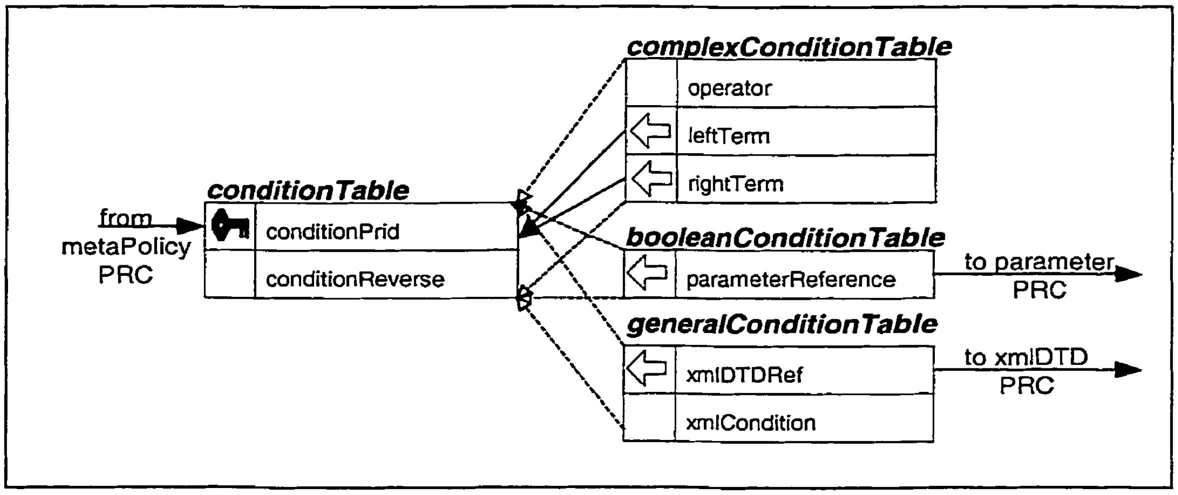


Figure 6.2: The Condition Group classes

The booleanCondition class also EXTENDS the base table. Each PRI contains a reference to a parameter, which must be of type "TrueValue" (i.e., Boolean). The value of the condition is evaluated according to the value of this parameter.

Finally, the generalCondition class is used to allow conditions to be evaluated through more complex expressions. Each row consists of a reference to the xmlDTD class and a string, which encodes an expression in XML. The reference to the xmlDTD class defines the XML DTD that must be used in order to interpret this expression. The expression encoded must be a logical expression, i.e., it must evaluate to either true or false.

6.1.4. The Actions Group

This group consists of three classes: the action, the actionValue and the actionParametricValue classes.

The action class is the base PRC for storing meta-policy actions. Each PRI contains a tag-reference attribute, which is used to group the actions of a single meta-policy. Each PRI of this class specifies a target PRID that specifies the PRI that must be installed.

The value that must be installed at the target PRID is determined either in the actionValue class or the actionParametricValue class. Both classes EXTEND the base one and provide the value that must be installed for the specific target PRID. The former provides a value, encoded according to BER (Basic Encoding Rules [37]), while the latter specifies a parameter, from which the value is evaluated.

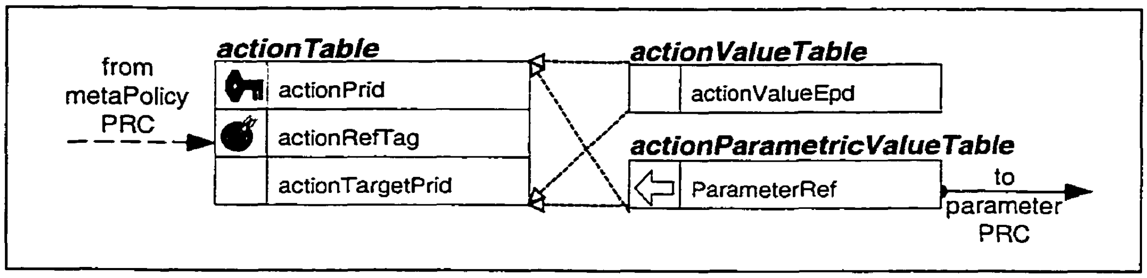


Figure 6.3: *The Actions Group classes*

6.1.5. The Parameter Group

This group contains three tables: the parameter, the mibPibParameter and the pdpParameter classes.

The parameter class is the base class for representing parameters. Each PRI consists of an identifier, a name and an attribute that denotes the type of the parameter. Each PRI must be associated with a PRI of a class that EXTENDS this one.

The mibPibParameter class is such a PRC. It defines a MIB or PIB identifier from which the parameter gets its value. Of course, this identifier must point to an existing variable. Each row also defines the frequency with which this value will be updated. Note that the MIB and PIB identifiers have a different name space, i.e., their prefixes are different; hence, the identifier itself includes the information whether this is a MIB or PIB reference.

The pdpParameter class also extends the base parameter class. Each PRI contains a single attribute that encodes, in BER, the value of the parameter. The PDP updates this PRC whenever this is necessary (usually when the value changes).

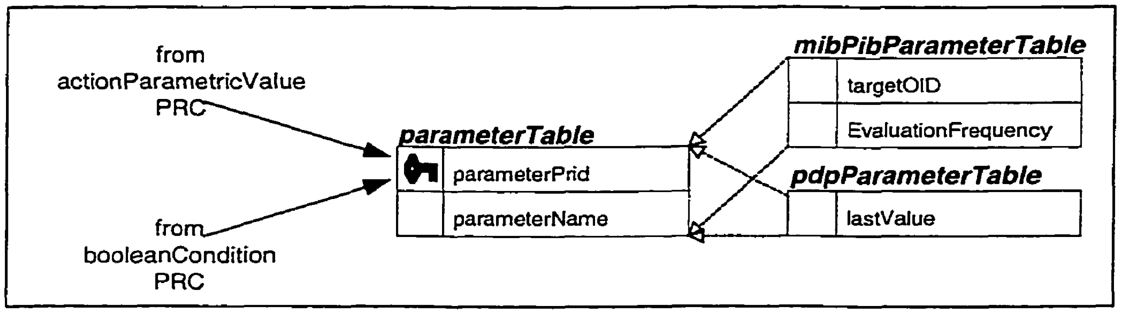


Figure 6.4: *The Meta-Policy Group classes*

6.1.6. Overview of the Entire PIB

Figure 6.5 demonstrates the entire Meta-Policy PIB. The groups are demonstrated as grayed boxes, containing the Provisioning Classes. The Provisioning represented as tables of their attributes. The figure illustrates the Instance Identifiers, the References to Instance Identifiers, the Group Tags, and the Group Tag References, as well as the "augmented" and "extended" classes.

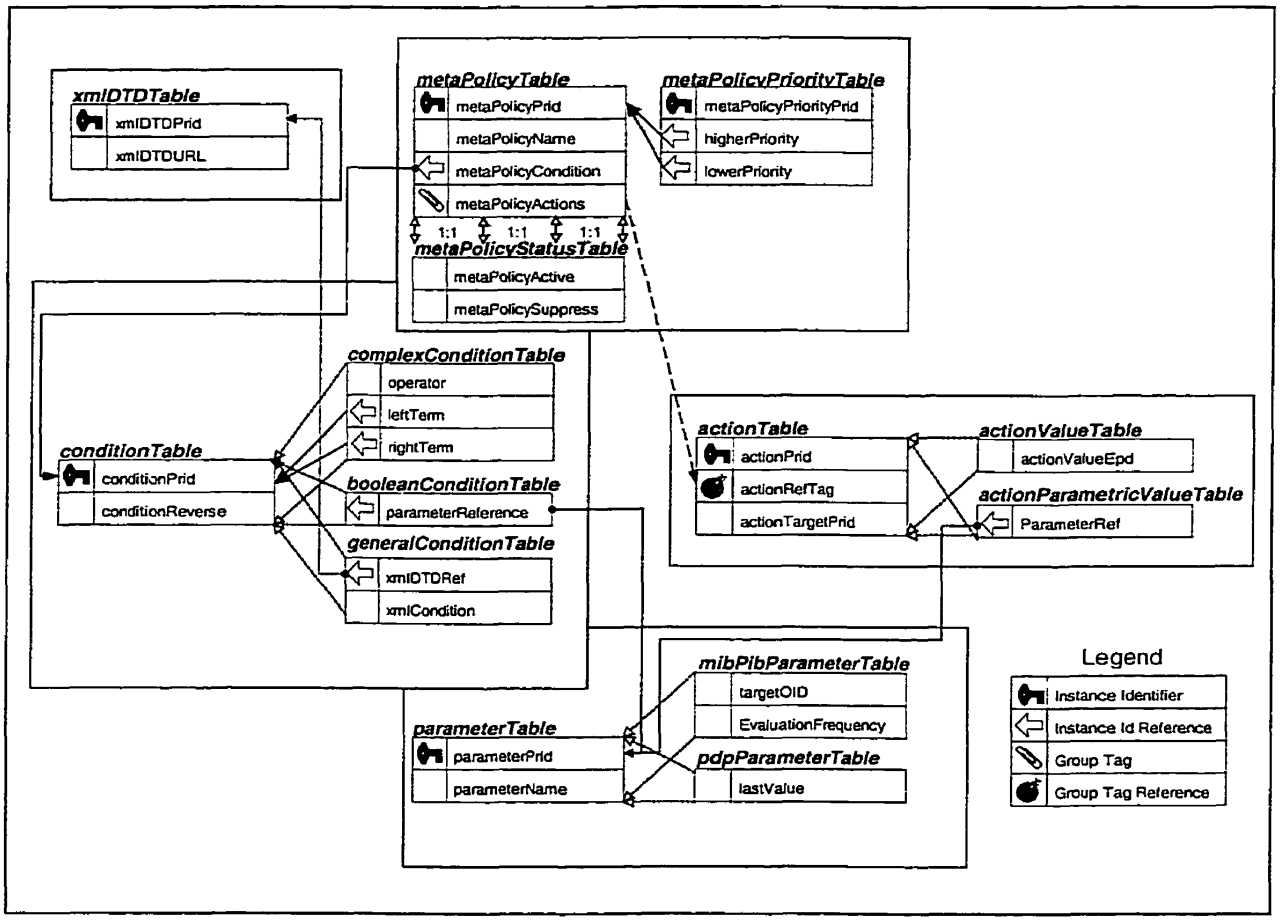


Figure 6.5: The Meta-Policy PIB

6.2. Communication & Storage

The communication and the storage of meta-policies are performed as defined in the COPS-PR protocol. A brief outline will be presented here; for more details, the reader may refer to the "COPS Usage for Policy Provisioning" from IETF [27].

6.2.1.1. Requests

As described by the protocol, when the PEP opens a connection to the PDP, it sends a configuration request (REQ) message, asking for all the applicable policing data. This REQ message also reports the capabilities and limitations of the PEP. According to the SPPI [36], this is performed by sending all the PRIs that are defined with the PIB-ACCESS clause set to "notify" or "install-notify". Thus, a PEP that implements the Meta-Policy PIB must include in this message the PRIs of the xmlDTD class, which report the capabilities of the PEP to interpret XML-encoded expressions.

6.2.1.2. Decisions

The PDP sends solicited decision (DEC) messages as replies to REQ messages, or unsolicited messages, whenever the policing data into the PIB needs to get updated. Meta-policing data is handled as any other kind of PIB data, hence the format of DEC messages and the way these are installed into the PIB are exactly as defined in the COPS protocol. As defined by the SPPI, the PIB can only install/modify PRIs with the PIB-ACCESS clause set to "install" or "install-notify".

Notice, however, that meta-policy data may now report network events to the PEP, since the PDP may send values for parameters that represent such events (e.g., the PDP may report congestion by setting the value of a parameter in the PIB of the PEP).

6.2.1.3. Reports

According to COPS-PR, the PEP reports the success or failure of the DEC message with a report message (RPT). DEC's that update the meta-policy classes are treated as any other DEC messages, hence the PEP must issue reports on whether the PRIs were installed/removed successfully. Note that this has nothing to do with whether the actions of the meta-policies are actually enforced successfully. These messages report whether the meta-poLicy itself was successfully installed/uninstalled, i.e., if the operation is valid according to the meta-policy PIB specification.

RPT messages are also sent unsolicited to report accounting related information. The reported PRIs have the PIB-ACCESS clause set to "report", hence the PRIs of the activeMetaPolicy class are reported to the PDP.

Finally, unsolicited RPT messages can report PEP errors that are not related to a specific DEC message. Such RPT messages can be triggered by badly behaving meta-policies, (e.g., that attempt to install invalid or conflicting PRIs). Although the content of the meta-policies should be checked when the meta-policy is installed, this check cannot detect all possible errors (this should be done by the PDP before sending the meta-policies), hence such situations may arise. Such errors are resolved according to the COPS-PR protocol.

6.3. PEP Operation

This section describes in general the behavior of the PEP and discusses how the data of the meta-policy PIB should be interpreted by it.

6.3.1.1. Installation of meta-policing data

When meta-policing data are to be installed into the PIB, the PEP needs to perform some basic tests to ensure that these data conform to the rules set in the PIB definition. Such tests include:

- ***Integrity:*** The installed PRIs contain the appropriate number and type of attributes.
- ***Consistency:*** The PRIs must not form illegal or invalid meta-policies in the PIB. For instance, references to non-existing PRIs are in several cases illegal: the installation of a condition cannot be performed, unless the parameters that it contains have been installed already, or they are installed in the same DEC message. A meta-policy cannot be declared as conflicting with itself. A condition cannot consist of two simpler conditions, one of which is the initial condition itself. The PEP should check for such situations before modifying the PIB.
- ***Conflicts:*** Whenever two meta-policies may be conflicting, the PDP should direct the PEP how to resolve the conflict through the metaPolicyPriority class. The PEP should check for conflicts that are not reported in this class. Also, the PEP should check for conflicts between the actions of meta-policies and PRIs directly installed by the PDP.

Whenever the PEP detects an erroneous situation like this, the entire DEC message must be rejected, and an RPT message indicating the cause of the error must be sent to the PDP (as defined in COPS-PR).

6.3.1.2. Parameters

The PIB defines two types of evaluation methods for the parameters: The values are either sent by the PDP, or they are evaluated from the MIB/PIB of the device. However, apart from these two integrated methods, new methods may be added in the future by defining classes that extend the parameter table.

Independently of the way a parameter is evaluated, the parameter triggers the reevaluation of the logical expressions in which it is contained. Also, if the parameter is used in the actions of an active meta-policy, whenever its value is modified, the related PRIs must be updated.

6.3.1.3. Conditions

As mentioned before, the condition of each meta-policy is decomposed into primitive logical expressions. Each logical expression contains a number of parameters, which must exist in the PIB before the logical expression is installed. When a logical expression is installed, it is evaluated according to the current values of its parameters. The overall condition is evaluated according to the evaluation of these logical expressions.

The triggering of the re-evaluation of the logical expressions was discussed in the previous section. Whenever the result of a logical expression is modified (i.e., it becomes true from false or vice versa), the condition that contains this expression needs to be reevaluated. The reevaluation of a condition may trigger the reevaluation of other, more complex conditions, containing this condition. For instance, for the condition (A&&(B||(C&&D))), assuming that A, B, C, D are primitive expressions, if D becomes true after being false, then the condition (C&&D) will be reevaluated. If its value changes to true, then the condition (B||(C&&D)) will be reevaluated. If its value also changes, the whole condition needs to be reevaluated. This procedure implies that the previous state of each condition is temporarily stored by the PEP, so that this comparison can be performed.

Also, the PEP may decide not to reevaluate a condition, if this is not considered necessary. For instance, in the previous case, if A is false, the values of B, C and D cannot influence the value of the entire condition, which will be false. However, if A becomes true, the reevaluation of the rest of the condition must be triggered by the PEP.

6.3.1.4. Actions

When the condition of a meta-policy evaluates true, if the meta-policy is not reported to be conflicting with another one with higher priority, the meta-policy is activated. The meta-policy stays active while its condition is met, and no other meta-policy with higher priority is activated.

When a meta-policy is activated, its actions are executed, installing the appropriate PRIs into the PIB. The actions of a PIB cannot modify existing PRIs, because this would be considered as a conflict with the meta-policy or the direct PDP command that installed these PRIs. However, a meta-policy may update its own PRIs (i.e, the PRIs that the meta-policy has installed into the PIB), if the values of these PRIs are parametric, and the values of these parameters change.

When a meta-policy is deactivated, the PRIs installed by this meta-policy are removed from the PIB. Since neither any other meta-policy nor the PDP could modify these PRIs while the meta-policy was active, the removal of the PRIs leaves the PIB consistent. When a meta-policy is deactivated, any meta-policies suppressed by this one may be activated (if their condition is still met and they are not suppressed by any other meta-policy).

6.3.1.5. Conflicts

Normally, all conflicting meta-policies are reported by the PDP in the appropriate class of the PIB (metaPolicyPriority class). When the condition of a meta-policy evaluates true, the PEP has to check the PRIs of this class in order to ensure that no meta-policy with higher priority is active. If no higher-priority meta-policy exists, the meta-policy is activated, else it remains inactive and it is denoted at the metaPolicyStatus class as suppressed. Before the meta-policy is activated, the PEP must deactivate any other meta-policies that conflict with this one and have lower priority (and declare them as suppressed, in the metaPolicyStatus class).

Note that the priorities declared by the PDP are relative priorities (in the form: meta-policy A has higher priority than meta-policy B). Also, note that if a meta-policy A has higher priority than B, and B has higher than C, then A does not necessarily conflict with C, hence both A and C may be active (e.g., as in the example presented in Chapter 4). However, if A and C conflict with each other, A can only have higher priority than C (otherwise, we are lead into a deadlock situation, where the conflict between A, B, C cannot be resolved).

Although all conflicting meta-policies should be reported by the PDP, the PEP should check for conflict both when the meta-policies are installed and executed (since run-time conflicts may also occur). Besides, meta-policies may conflict with PRIs directly installed by the PDP (although such situations should also be prevented at the PDP level). However, if such an abnormal situation occurs, the PEP should either refuse to execute the PDP decision that causes this conflict (installation conflicts) and issue the appropriate solicited failure report message, or refuse to execute the meta-policy that causes the conflict (run-time conflicts) and report the event with an unsolicited failure report message.

6.4. Backwards Compatibility

The proposed PIB does not create any backwards compatibility issues, when PDPs that support the proposed PIB are required to cooperate with PEPs that do not, and vice versa.

If a PEP that does not implement the meta-policy PIB connects to a PDP that supports it, then in the request message of the former no meta-policy classes will be reported. Hence, the PDP is not allowed to send meta-policing data, and it should assume that it must control the PEP in the traditional way, i.e., by directly installing and removing PRIs into its PIB.

If a PEP that supports the meta-policy PIB connects to a PDP that does not, then the PDP will not recognize the meta-policy classes, reported by the PEP in the request message. In this case, as defined by COPS-PR [27], the PDP will not send any configuration data for these classes, and it will control the PEP just by sending commands that directly install or remove PRIs to the rest of its PIB. Hence, the PEP will receive no meta-policies and it will operate as if it did not implement the extra functionality.

Chapter 7. Conclusions and Future Work

The previous chapters introduced the concept of meta-policies, demonstrated their usage and defined the Meta-Policy Information Base. This section presents our work in progress, discusses some interesting research issues related to this work, and concludes this thesis.

7.1. Conclusions

This thesis introduced the concept of COPS-PR meta-policies and proposed, presented and defined a Policy Information Base (PIB) that attempts to push some of the COPS-PR PDP functionality and intelligence towards the PEPs, by using such meta-policies.

This document discussed the current situation in network management and outlined the modern trends and techniques. It introduced Policy-Based Networking, COPS, COPS-PR and PIBs, and demonstrated how these work. Based on these, the motivation of our work was presented, the concept of meta-policies was introduced, and an example of how the latter can enhance the current techniques was demonstrated. The requirements for the proposed PIB were presented and analyzed, and the PIB and the PEP operation were defined. Finally, our work in progress and our future research goals were outlined.

7.2. Work in Progress

7.2.1. Implementation and Testing

The definition of a PIB consists of the definition of the Provisioning Classes (PRCs) and the definition of the operation of the PEP, i.e., how the latter interprets the data stored in the PIB. Nevertheless, it does not require defining the behavior of the PDP, which is allowed to use the provided functionality in any desired fashion. The additional functionality obviously improves the existing techniques in some aspects, since the Meta-Policies allow the PEP to operate correctly in cases where a PEP with no meta-policy support would fail (see example in section 4.5). From this point of view, the work presented in this thesis is complete.

However, we are currently implementing a PEP with meta-policy capabilities, in order to test and compare our proposal with PEPs without meta-policies.

As far as implementation is concerned, we are currently building a PEP that implements the meta-policy PIB. As already discussed and as explained later in more detail, since our work is strongly related to active networks, we implement a PEP that resides on an active router. More specifically, for our experiments, we have at our disposal two Nortel Accelar 1100-B routers (formerly named Passport 1100-B). These routers can download and execute code within the Oplet Runtime Environment (ORE) [39]. ORE is an environment where java classes can be executed. Java classes that allow configuring the parameters of the routers are provided by the vendor. We are currently implementing a PEP that implements the proposed PIB in this environment.

However, implementing the meta-policy PIB is meaningless, unless another PIB exists on the same PEP, which will be controlled by meta-policies. Unfortunately, the PIBs proposed in IETF are currently in a very premature stage, and no freely available implementation or code of any of them exists. Hence, we have also designed a simple PIB for the purposes of our experiments, and we are currently implementing it. This PIB will be executed within ORE as well, and it is written in Java.

Moreover, COPS-PR is also in a premature stage. No suitable free implementation for the protocol exists right now. IPHighway and Intel have independently developed two open-source COPS SDKs [23], [24], which provide the client-side interface for the basic COPS functionality; Vovida recently released free source that implements the COPS stack, with COPS-PR support [25]. However, in all cases the code is written in C/C++. Hence, for our experiments we also encode in Java the classes that implement a simplified version of COPS-PR. (We have not implemented some COPS messages used for integrity, security, etc, that do not affect our work.)

Finally, in order to test and compare the proposed PIB, we will also need a PDP to control this PIB. Currently, no free PDP implementation exists. Intel unofficially claims to have implemented a PDP that can be purchased, but no official presentation of this PDP has been done yet. IPHighway has developed a "COPS Proxy" which seems to be functionally similar to a PDP, although with reduced functionality and intelligence. However, for our experiments we need a PDP that (i) supports COPS-PR, and (ii) can be extended to support the proposed PIB. These implementations meet none of these requirements adequately. Thus, we will probably need to write a simple PDP, as well, that will control the proposed PIB.

After implementing the PIB, the PEP and the PDP, we plan to test the proposed PIB with regard to its performance and efficiency, and compare it with the existing techniques (COPS-PR without meta-policies). Unfortunately, the results of the test are subjective, and depend to the chosen set of policies. An objective comparison could only be performed in a real environment, with real policies, PDP and PIBs. However, such an option is, for the time being, not possible, since COPS-PR has not yet been extensively used in real environments. Thus, our testing and comparison will be carried out with a synthetic set of policies, and it will mainly concentrate on resource usage at the PEP and PDP, as well as the network usage. Also, we will demonstrate scenarios where meta-policies allow the PEP to operate correctly (such as PDP failures). Finally, our experiments will test how active networks and directories can be used to enhance meta-policies.

7.2.2. Contribution to IETF

The contribution of this work is the definition of the meta-policy PIB. We have already published early versions of this work and we are currently submitting new ones. In particular, we are in the process of submitting an Internet-Draft at IETF. Any publication at IETF is reviewed and criticized by any interested individual or party of the Internet community, and either evolves to an Internet-Standard, or it is rejected. By exposing our work directly to IETF, we allow the Internet Community to criticize it and adopt all or parts of it. Also, the feedback and comments that will be acquired through this process will allow us to estimate the interest of other academic and industrial research groups in it, and will affect our future work. The proposed PIB, as presented in Appendix A, will soon be submitted and published under the RAP working group of IETF.

7.3. Future Research

7.3.1. Meta-Policy Hierarchies

The goal of this thesis was to enhance the functionality of the PEPs with meta-policies that manipulate the PIB data, according to the high-level network policies. However, the meta-policies themselves are also PIB data, hence meta-policies that control meta-policies can also exist. As a matter of fact, the PIB defined here allows an unlimited number of levels of meta-policies to be stored in the PIB and control its content.

By constructing hierarchies of meta-policies, more functionality and intelligence can be pushed towards the PEP. For instance, 2nd level meta-policies can be used to group 1st level meta-policies for different generic network states: Different meta-policies can be applicable when the network operates normally, when it is under maintenance, when it is under attack or when it is congested. A second example is the use of meta-policies that self-generate a set of similar meta-policies that control the PEP: A meta-policy could create meta-policies, each of which grants to a user of a group specific privileges.

While allowing hierarchies of meta-policies significantly increases the intelligence of the PEP, this functionality was not one of the goals that drove the design of the proposed PIB. One of our future goals is to examine how this affects this work. More specifically, we are interested in examining what types of meta-policies might be beneficial, and whether these can be implemented in the proposed PIB. We know already that the proposed PIB can support some hierarchies of meta-policies. However, we need to examine whether the provided functionality is sufficient and whether it can be implemented to efficiently support any desired type of meta-policy hierarchies. If not, we would like to investigate the required modifications.

7.3.2. Meta-Policies and Active Networks

As stated several times throughout this document, our work was inspired by Active Networks. Although the proposed PIB does not explicitly demand an active (or programmable) environment, the whole concept of downloading intelligence into the network devices and distributing functionality into them assumes that the network elements have the ability and the resources to perform advanced operations and tasks.

Legacy devices could implement this PIB; nevertheless, such devices usually have limited resources and capabilities, hence only a small number of complex meta-policies could be efficiently handled by them. On the other hand, an active device has the capabilities and the resources to perform complex computations and tasks, and thus, to process and enforce meta-policies efficiently.

However, the most important property of Active Networks, as far as our work is concerned, is the ability to extend the defined PIB. First, the XML DTDs that a device supports can be easily enhanced to support newer DTDs, by downloading modules that process these DTDs and by declaring these DTDs in the xmlDTD PRC. Second, and most important, the parameter evaluation methods can be extended, as well. This is a very important property, since the extension of the evaluation methods allows the PEP to monitor and enforce more meta-policies by itself, independently of the PDP. The extension of the evaluation methods can be performed centrally (to ensure automation and consistency), according to the network topology and services. For instance, active code that queries a directory or an authentication server can be made available and be used by some PEPs in order to provide a value to a PIB parameter. Supposing that the PDP is aware of the existence of a library with such code, as well as of which devices can download and use this code, the PDP can command the PEPs of these devices to use this code to compute some values of the meta-policy parameters. Notice that the code in this library may either be provided by the vendors and be network-independent, or be written or customized by the administrators and be network-specific. The only requirements are that the PEPs will be able to be directed to download this code, and the PDP will be aware of how this code can be used.

The previous discussion makes it obvious that Active Networks significantly affect our work. As future work, we would like to investigate how such Active Network properties can be best exploited.

7.3.3. Meta-Policies and Directories

Another interesting research topic is how our work can be enhanced by using Directories. As mentioned already, we envision PEPs that can download modules or code in order to extend their abilities. Such modules and code could be stored in a Directory Server. Besides, some types of policing information that changes infrequently, used in order to compute PIB parameters, could be stored in Directories as well, and be fetched directly by the PEPs. We intend to investigate in more detail how Directories can influence our work.

7.3.4. Moving the PDPs to the Network Elements

Another interesting observation is that, by using meta-policies, a great degree of functionality can be pushed towards the PEP. Actually, the main difference between a PDP and a PIB loaded with meta-policies is that the latter cannot translate the high-level policies into low-level PIB commands (meta-policies or normal policies). However, future devices with more resources and capabilities could host an extra module that implements this functionality. In this case, the entire PDP could be hosted on the network element. This topic is currently considered very promising, and it is also included in our future research goals.

References*

[1]. A. Westerinen, J. Schnizlein,J. Strassner, Mark Scherling,Bob Quinn,Jay Perry, Shai Herzog, An-Ni Huynh, Mark Carlson, Steve Waldbusser;
"Terminology";
IETF, Internet-Draft, draft-ietf-policy-terminology-02.txt, November 2000 (http://www.ietf.org/internet-drafts/draft-ietf-policy-terminology-02.txt)

[2]. Heinz-Gerd Hegering, Sebastian Abeck and Bernhard Neumair;
"Integrated Management of Networked Systems";
Morgan Kaufman, 1999.

[3]. **"FCAPS Overview";**
http://www.fore.com/products/fv/fv_fcaps_wp.html [September 2000]

[4]. R. Boutaba, Andreas Polyrakis;
"Projecting FCAPS to Active Networks";
accepted at IEEE EntNet 2001; Atlanta, GA, USA; 4-6 June, 2001

[5]. **"Internet Engineering Task Force";**
http://www.ietf.org/

[6]. **"SNMP Version 3 (snmpv3)";**
http://www.ietf.org/html.charters/snmpv3-charter.html

[7]. David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden;

* All URLs were valid on the 15th of April, 2001, unless otherwise stated. All IETF Internet-Drafts and RFCs were available at IETF web site [5] on the 15th of April, 2001, unless otherwise stated.

“A Survey of Active Network Research”;
IEEE Communications Magazine, Vol. 35, No. 1, January 1997, pp.80-86

[8]. Konstantinos Psounis;
“Active Networks: Applications, Security, Safety, and Architectures”;
IEEE Communications Surveys, Vol. 2, No. 1, First Quarter 1999
(http://www.comsoc.org/pubs/surveys/1q99issue/psounis.html)

[9]. Jonathan M. Smith, Kenneth L. Calvert, Sandra L. Murphy, Hilarie K. Orman, Larry L. Peterson;
“Activating networks: a progress report”;
IEEE Computer, Vol. 32, No 4, April 1999, pp.32-41

[10]. D. L. Tennenhouse and D. J. Wetherall;
“Towards an Active Network Architecture”;
Computer Communication Review, Vol. 26, No. 2, April 1996.

[11]. **“Directory-Enabled Networks”;**
3COM, White paper, July 1998.

[12]. **“Policy-Powered Networking and the Role of Directories”;**
3COM, White paper, July 1998.

[13]. **“DMTF Home page”;**
http://www.dmtf.org/

[14]. Susan J. Shepard;
“Policy-based networks: hype and hope“;
IT Professional, Vol. 2, No. 1, January-February 2000, pp.12 -16

[15]. **“Introduction to Policy-based Networking and Quality of Service”;**
IPHighway, White paper, January 2000

[16]. R. Boutaba, K. El-Guemhioui, P. Dini;
"An Outlook on Intranet Management";

IEEE Communications Magazine, Special issue on Intranet Services and Communication Management, October 1997, pp.92-97

[17]. R. Boutaba, S. Znaty,
"An Architectural Approach for Integrated Networks and Systems Management";
ACM-SIGCOM Computer Communication Review, Vol. 25, No 5, October 1995, pp. 13-39

[18]. M. Sloman;
"Policy Driven Management For Distributed Systems";
Plenum Press Journal of Network and Systems Management, Vol. 2, No. 4, December 1994, pp. 333-360

[19]. **"Policy Standards and IETF Terminology";**
IPHighway, White paper, January 2000.

[20]. **"Policy Based Networking Products, Design and Architecture";**
IPHighway, White paper, January 2000.

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

[22]. **"Resource Allocation Protocol (rap)";**
http://www.ietf.org/html.charters/rap-charter.html

[23]. **"Intel COPS client Software Development Kit";**
http://www.intel.com/ial/cops/

[24]. **"IPHighway – COPS open source";**
http://www.iphighway.com/opensource1.htm

[25]. **"COPS Download Page";**
http://www.vovida.org/protocols/downloads/cops/

[26]. S. Herzog, Ed., J. Boyle, R. Cohen, D. Durham, R. Rajan, A. Sastry;
"COPS usage for RSVP";
IETF, RFC 2749, January 2000
(http://www.ietf.org/rfc/rfc2749.txt)

[27]. K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, A. Smith;
"COPS Usage for Policy Provisioning";
IETF, RFC 3084, March 2001
(http://www.ietf.org/rfc/rfc3084.txt)

[28]. M. Fine, K. McCloghrie, J. Seligson, K. Chan, S. Hahn, A. Smith, F. Reichmeyer;
"Differentiated Services Quality of Service Policy Information Base";
IETF, Internet-Draft, draft-ietf-diffserv-pib-03.txt, March 2001
(http://www.ietf.org/internet-drafts/draft-ietf-diffserv-pib-03.txt)

[29]. D. Rawlins, A. Kulkarni, K. Ho Chan, D. Dutt,
"Framework of COPS-PR Policy Information Base for Accounting Usage";
IETF, Internet-Draft, draft-ietf-rap-acct-fr-pib-01.txt, July 2000
(http://www.ietf.org/internet-drafts/draft-ietf-rap-acct-fr-pib-01.txt)

[30]. J. Ottensmeyer, M. Bokaemper, K. Roeber;
"A Filtering Policy Information Base (PIB) for Edge Router Filtering Services and Provisioning via COPS-PR";
IETF, Internet-Draft, draft-otty-cops-pr-filter-pib-00.txt, November 2000
(http://www.ietf.org/internet-drafts/draft-otty-cops-pr-filter-pib-00.txt)

[31]. M. Li, D. Arneson, A. Doria, J. Jason, C. Wang;
"IPSec Policy Information Base";
IETF, Internet-Draft, draft-ietf-ipsp-ipsecpib-02.txt, March 2001
(http://www.ietf.org/internet-drafts/draft-ietf-ipsp-ipsecpib-02.txt)

[32]. T. Anderson, D. Putzolu, A. Doria, J. Yong, J. Sydir, B. Srinivasan;
"Multiple Virtual Router Partitioning Policy Information Base";

Internet Draft draft-anderson-mvr-pib-00.txt, July 2000
(http://www.ietf.org/internet-drafts/draft-anderson-mvr-pib-00.txt) [discontinued]

[33]. T. Anderson, A. Doria, J. Yong, S. Crosby;
"IP Forwarding PIB";
IETF, Internet-Draft, draft-khosravi-ip-fwd-pib-00.txt, July 2000
(http://www.ietf.org/internet-drafts/draft-khosravi-ip-fwd-pib-00.txt) [discontinued]

[34]. Harsha Hegde, Brad Stone;
"Load Balancing Policy Information Base";
IETF, Internet-Draft, draft-hegde-load-balancing-pib-00.txt, February 2001
(http://www.ietf.org/internet-drafts/ draft-hegde-load-balancing-pib-00.txt)

[35]. M. Fine, K. McCloghrie, J. Seligson, K. Chan; S. Hahn, R. Sahita, A. Smith, F. Reichmeyer;
"Framework Policy Information Base",
IETF, Internet-Draft, draft-ietf-rap-frameworkpib-04.txt, November 2000
(http://www.ietf.org/internet-drafts/draft-ietf-rap-frameworkpib-04.txt)

[36]. K. McCloghrie, M. Fine, J. Seligson, K. Chan, S. Hahn, R. Sahita, A. Smith, F. Reichmeyer;
"Structure of Policy Provisioning Information (SPPI)";
IETF, Internet-Draft, draft-ietf-rap-frameworkpib-06.txt, February 2001
(http://www.ietf.org/internet-drafts/ draft-ietf-rap-frameworkpib-06.txt)

[37]. **"Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)";**
Information processing systems - Open Systems Interconnection, International Organization for Standardization, International Standard 8825, December, 1987

[38]. K. McCloghrie, D. Perkins, J. Schoenwaelder, J. Case, M. Rose and S. Waldbusser;
"Structure of Management Information Version 2 (SMIv2)";
IETF, RFC 2578, April 1999
(http://www.ietf.org/rfc/rfc2578.txt)

[39]. **"Oplet Runtime Environment"**

http://www.openetlab.org/ore.htm

[40]. R. Boutaba, Andreas Polyrakis;

"Towards Extensible Policy Enforcement Points";

IEEE Workshop on Policies for Distributed Systems and Networks, Bristol, U.K., 29-31 January 2001, pp. 247-261

[41]. R. Boutaba, Andreas Polyrakis;

"COPS-PR with Meta-Policy Support";

IETF, Internet-Draft, draft-boutaba-copsprmp-00.txt, April 2001 (http://www.ietf.org/internet-drafts/draft-boutaba-copsprmp-00.txt)

Appendices

Appendix A. The Meta-Policy PIB

```
INTERNET-DRAFT                                     Andreas Polyrakis
Resource Allocation Working Group (RAP)            University of Toronto
Intended Category: Standards Track                 Raouf Boutaba
Expires: October, 2001                             University of Waterloo


                 The Meta-Policy Information Base (M-PIB)

                        draft-ietf-rap-mpib-00.txt


Status of this Memo

   This document is an Internet-Draft and is in full conformance with
   all provisions of Section 10 of RFC2026.  Internet-Drafts are
   working documents of the Internet Engineering Task Force (IETF), its
   areas, and its working groups.  Note that other groups may also
   distribute working documents as Internet-Drafts.

   Internet-Drafts are draft documents valid for a maximum of six
   months and may be updated, replaced, or obsoleted by other documents
   at any time.  It is inappropriate to use Internet-Drafts as
   reference material or to cite them other than as "work in progress".

   The list of current Internet-Drafts can be accessed at
   http://www.ietf.org/ietf/1id-abstracts.txt

   The list of Internet-Draft Shadow Directories can be accessed at
   http://www.ietf.org/shadow.html.













   R.Boutaba, A.Polyrakis  Internet-Draft, expires Oct.2001    [page 1]
```

Abstract

This document introduces the concept of COPS-PR meta-policies, and defines the Meta-Policy Information Base.

The meta-policy PIB does not introduce a new policing area. On the contrary, it defines some provisioning classes that can be used by all other PIBs, in order to add meta-policing functionality into them. The meta-policy PIB, like every PIB, stores policing information that controls some policing mechanisms of the device. However, unlike other PIBs, the policing mechanism controlled by the meta-policy PRCs is the PIB itself. The data maintained by these PRCs implement policies that control other policies, this is why they are called meta-policies.

Meta-policies is an attempt to push intelligence towards the COPS-PR PEPs and overcome the rigidity of their PIBs. Through meta-policies, more policing information and functionality can be pushed towards the PEP, less interaction with the PDP is necessary, and less network and PDP resources are consumed. The PEP is more independent and it can bear longer PDP absences.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC-2119].

Glossary - Terminology

This document follows the terminology of [P-TERM]. However, the most commonly used terms are cited again here:

PDP	Policy Decision Point. See [RAP-FRM].
PEP	Policy Enforcement Point. See [RAP-FRM].
PRC	Provisioning Class. See [COPS-PR].
PRI	Provisioning Instance. See [COPS-PR].
PIB	Policy Information Base. See [COPS-PR].
PRID	Provisioning Instance Identifier. Uniquely identifies an instance of a PRC. See [COPS-PR].

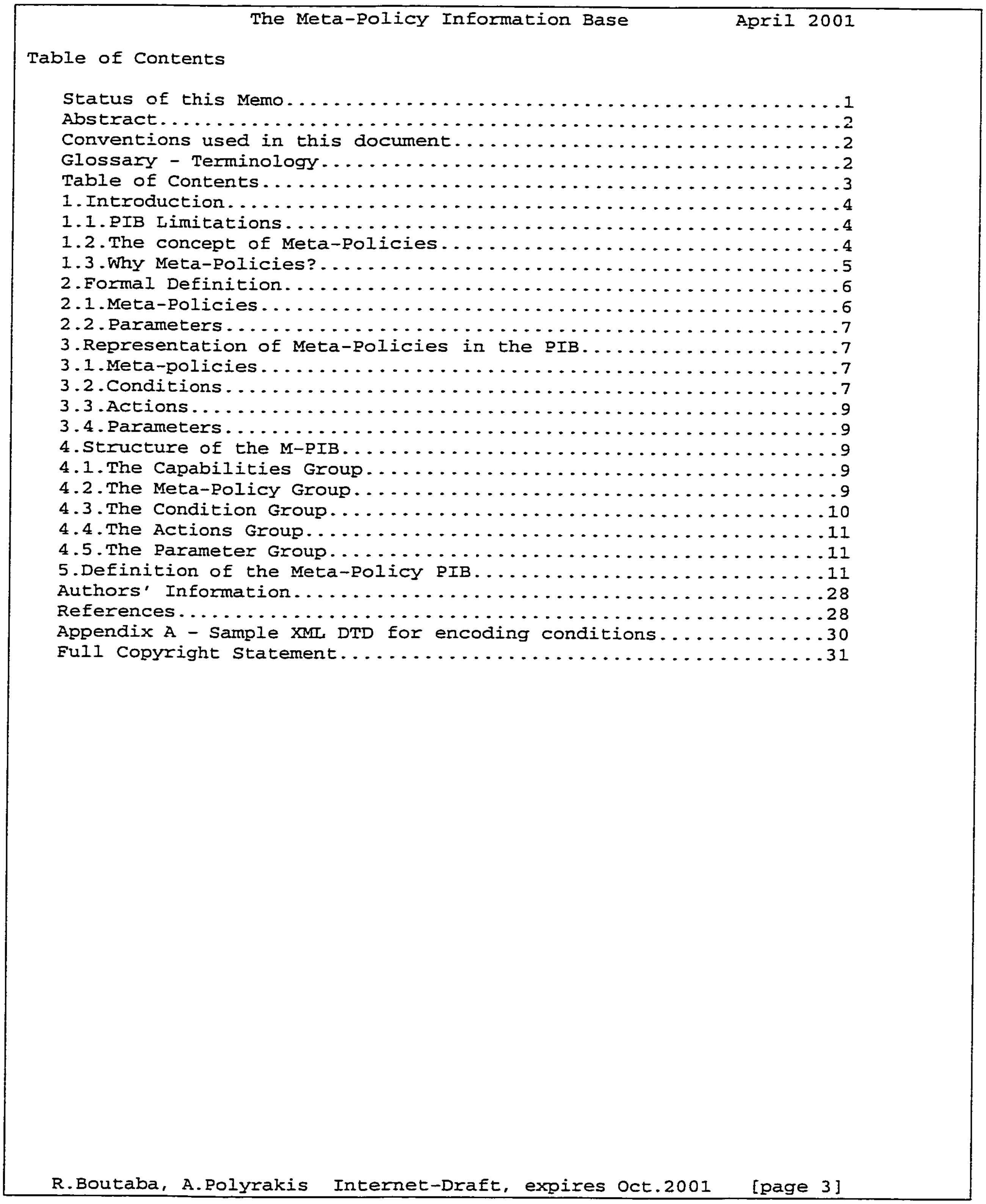
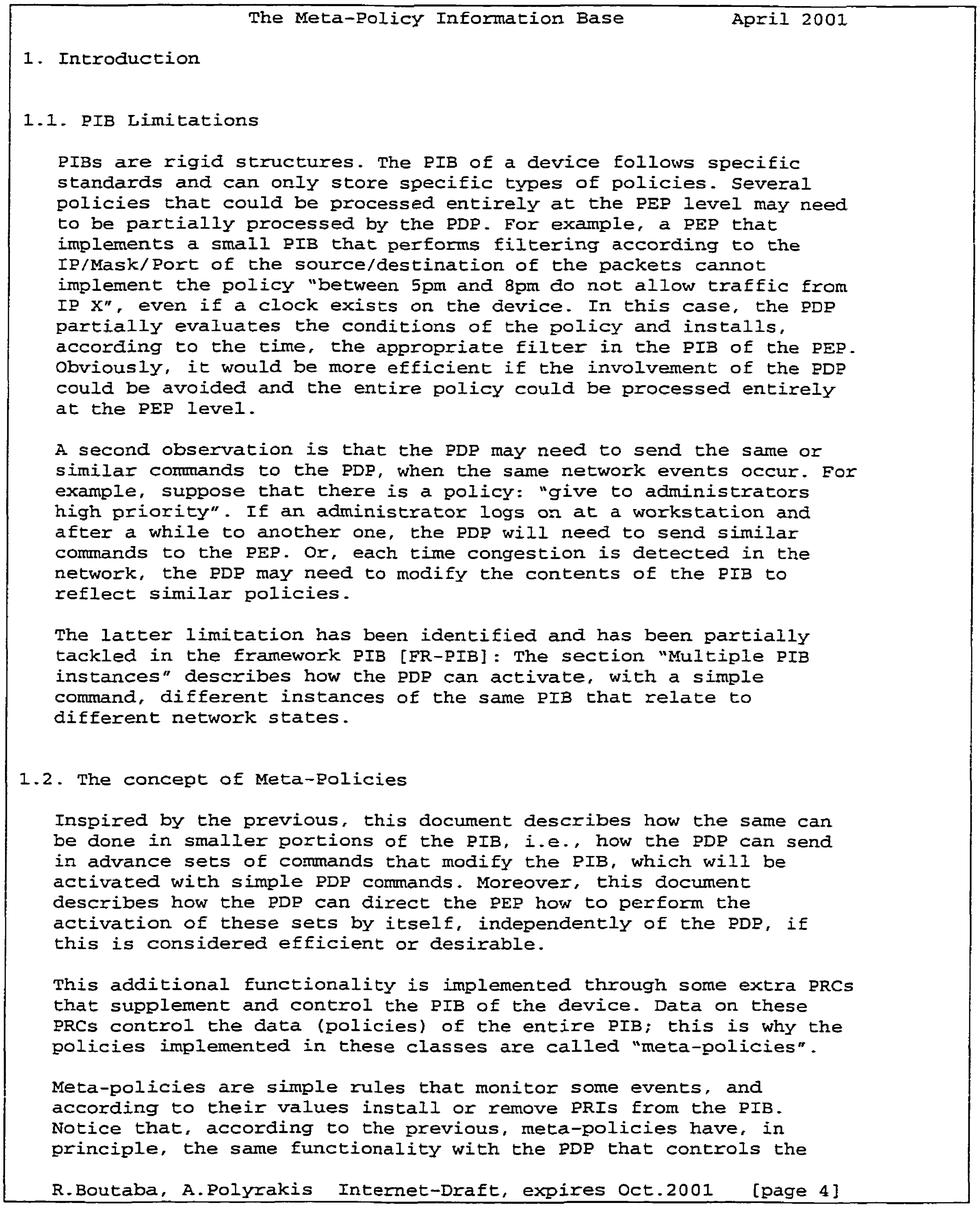


Table of Contents



1. Introduction

1.1. PIB Limitations

PIBs are rigid structures. The PIB of a device follows specific standards and can only store specific types of policies. Several policies that could be processed entirely at the PEP level may need to be partially processed by the PDP. For example, a PEP that implements a small PIB that performs filtering according to the IP/Mask/Port of the source/destination of the packets cannot implement the policy "between 5pm and 8pm do not allow traffic from IP X", even if a clock exists on the device. In this case, the PDP partially evaluates the conditions of the policy and installs, according to the time, the appropriate filter in the PIB of the PEP. Obviously, it would be more efficient if the involvement of the PDP could be avoided and the entire policy could be processed entirely at the PEP level.

A second observation is that the PDP may need to send the same or similar commands to the PDP, when the same network events occur. For example, suppose that there is a policy: "give to administrators high priority". If an administrator logs on at a workstation and after a while to another one, the PDP will need to send similar commands to the PEP. Or, each time congestion is detected in the network, the PDP may need to modify the contents of the PIB to reflect similar policies.

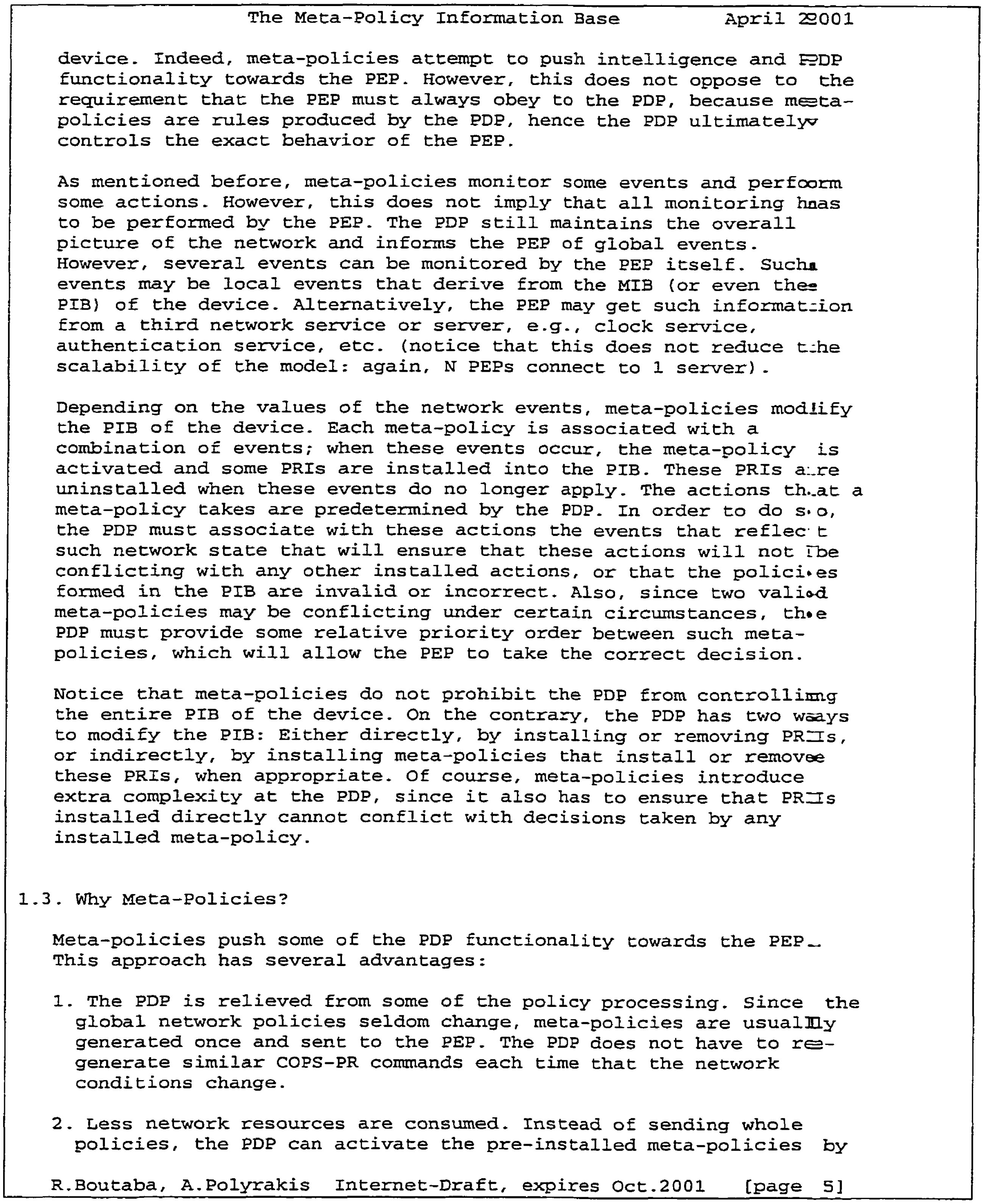
The latter limitation has been identified and has been partially tackled in the framework PIB [FR-PIB]: The section "Multiple PIB instances" describes how the PDP can activate, with a simple command, different instances of the same PIB that relate to different network states.

1.2. The concept of Meta-Policies

Inspired by the previous, this document describes how the same can be done in smaller portions of the PIB, i.e., how the PDP can send in advance sets of commands that modify the PIB, which will be activated with simple PDP commands. Moreover, this document describes how the PDP can direct the PEP how to perform the activation of these sets by itself, independently of the PDP, if this is considered efficient or desirable.

This additional functionality is implemented through some extra PRCs that supplement and control the PIB of the device. Data on these PRCs control the data (policies) of the entire PIB; this is why the policies implemented in these classes are called "meta-policies".

Meta-policies are simple rules that monitor some events, and according to their values install or remove PRIs from the PIB. Notice that, according to the previous, meta-policies have, in principle, the same functionality with the PDP that controls the



device. Indeed, meta-policies attempt to push intelligence and PDP functionality towards the PEP. However, this does not oppose to the requirement that the PEP must always obey to the PDP, because meta-policies are rules produced by the PDP, hence the PDP ultimately controls the exact behavior of the PEP.

As mentioned before, meta-policies monitor some events and perform some actions. However, this does not imply that all monitoring has to be performed by the PEP. The PDP still maintains the overall picture of the network and informs the PEP of global events. However, several events can be monitored by the PEP itself. Such events may be local events that derive from the MIB (or even the PIB) of the device. Alternatively, the PEP may get such information from a third network service or server, e.g., clock service, authentication service, etc. (notice that this does not reduce the scalability of the model: again, N PEPs connect to 1 server).

Depending on the values of the network events, meta-policies modify the PIB of the device. Each meta-policy is associated with a combination of events; when these events occur, the meta-policy is activated and some PRIs are installed into the PIB. These PRIs are uninstalled when these events do no longer apply. The actions that a meta-policy takes are predetermined by the PDP. In order to do so, the PDP must associate with these actions the events that reflect such network state that will ensure that these actions will not be conflicting with any other installed actions, or that the policies formed in the PIB are invalid or incorrect. Also, since two valid meta-policies may be conflicting under certain circumstances, the PDP must provide some relative priority order between such meta-policies, which will allow the PEP to take the correct decision.

Notice that meta-policies do not prohibit the PDP from controlling the entire PIB of the device. On the contrary, the PDP has two ways to modify the PIB: Either directly, by installing or removing PRIs, or indirectly, by installing meta-policies that install or remove these PRIs, when appropriate. Of course, meta-policies introduce extra complexity at the PDP, since it also has to ensure that PRIs installed directly cannot conflict with decisions taken by any installed meta-policy.

1.3. Why Meta-Policies?

Meta-policies push some of the PDP functionality towards the PEP. This approach has several advantages:

1. The PDP is relieved from some of the policy processing. Since the global network policies seldom change, meta-policies are usually generated once and sent to the PEP. The PDP does not have to re-generate similar COPS-PR commands each time that the network conditions change.

2. Less network resources are consumed. Instead of sending whole policies, the PDP can activate the pre-installed meta-policies by

communicating network events. Also, the PEPs can be programmed to monitor local events, which means that these events do not need to be communicated to the PDP, and then back to the PEP.

3. The PEPs become more independent, since they are able to take more decisions, according to various network events. Thus, they can operate correctly during larger PDP absences, and they are less affected by situations such as congestion, high network delays, packet loss and PDP overload.

4. The fact that the behavior of the PEPs can be controlled with smaller messages (network events instead of whole policies) makes the model more robust in erroneous network situations, such as congestion and high packet loss.

In general, meta-policies contribute towards the scalability, distribution, robustness and fault-tolerance of the COPS-PR model.

Note that meta-policies allow the PDP to push towards the PEP as little intelligence as a few simple meta-policies or as much as integrating almost the entire PDP functionality into it.

2. Formal Definition

2.1. Meta-Policies

We define a meta-policy as a rule of the form:

if (condition) then {actions}

where "condition" is a logical expression,
e.g., "(C>80%) and (D=true)",
and "actions" is a set of commands that install PRIs into the PIB.

Since the actions encode a specific policy, this rule is a rule on how policies are enforced; this is why it is called "meta-policy".

Each meta-policy is generated for a specific PEP, according to its capabilities, limitations and the device on which it resides. The PEP evaluates the condition of each meta-policy, and when it evaluates true, it enforces the actions. When it becomes false, the PRIs are uninstalled. The key idea in meta-policies is that the PEP can store and process these meta-policies without knowing their exact semantics: The condition is treated as a logical expression; the actions, pre-generated by the PDP, just denote PRIs that must be installed, and this can be done by the PEP without knowing the policy they really implement. In this way, the PEP can process any meta-policy, independently of its complexity and its meaning.

Both the condition and the actions may contain parameters (such as "Congestion" or "Time"); the values for these parameters are either sent by the PDP or evaluated by the PEP, according to directions provided by the former.

2.2. Parameters

The parameters are used in meta-policy conditions in order to determine when a meta-policy must be activated. Moreover, they are used by meta-policy actions in order to dynamically bind the network state within policies. For example, the meta-policy "if (AdminLogged) then {give high priority to AdminIP}", contains the parameters AdminLogged and AdminIP.

When installing a parameter, the PDP must specify an evaluation method for this parameter. For instance, the PEP can be directed to get a value for a parameter from the MIB or the PIB of the device. Alternatively, the PDP could provide the value for this parameter. However, other methods are also possible, depending on the capabilities of the device, such as to download and execute a script, use mobile agents, or get the desired information from some server or service.

3. Representation of Meta-Policies in the PIB

3.1. Meta-policies

Each meta-policy is comprised of two parts: The "condition" and the "actions". The "condition" is a logical expression that may be divided into simpler conditions. The "actions" is a group of PIB commands that install or remove PRIs. A meta-policy MUST always be associated with a condition, and it is expected to be associated with one or more actions (meta-policies without actions should normally occur only as the result of temporal deactivation of its actions).

Since meta-policies may be conflicting, the relative priority between potentially conflicting meta-policies MUST be declared in the PIB.

3.2. Conditions

The "condition" of a meta-policy is a logical expression that determines when the meta-policy must be activated. Each meta-policy must contain exactly one condition. The condition may consist other simpler conditions; and these conditions may similarly be comprised of even simpler conditions, etc. In this way, the condition is eventually decomposed in primitives that are logical expressions (i.e., they evaluate true or false), but cannot be further decomposed (i.e., the expression (X>Y).

This document distinguishes two types of such primitives: Booleans and generic logical expressions. Booleans are a subset of the generic expressions, but due to their simplicity and commonality, they are treated separately. Such primitives are evaluated according to the value of a Boolean parameter. For instance, the condition (X>Y) && (!Congestion) && (WorkTime) is decomposed into three primitives: "X>Y", "Congestion" and "WorkTime". From these three primitives, only the two are Booleans. Booleans MUST be supported by all meta-policy PIBs.

Generic expressions contain all the other logical expressions that cannot be decomposed into simpler primitives. Examples of such primitives are "X>Y", "IP matches X.Y.Z.W" or "8:00am < time < 5:00pm). Each PEP can only support specific types of such expressions (e.g., arithmetic), which are reported along with the other PEP capabilities to the PDP. The PDP can only sent to the PEP expressions that are supported by the latter.

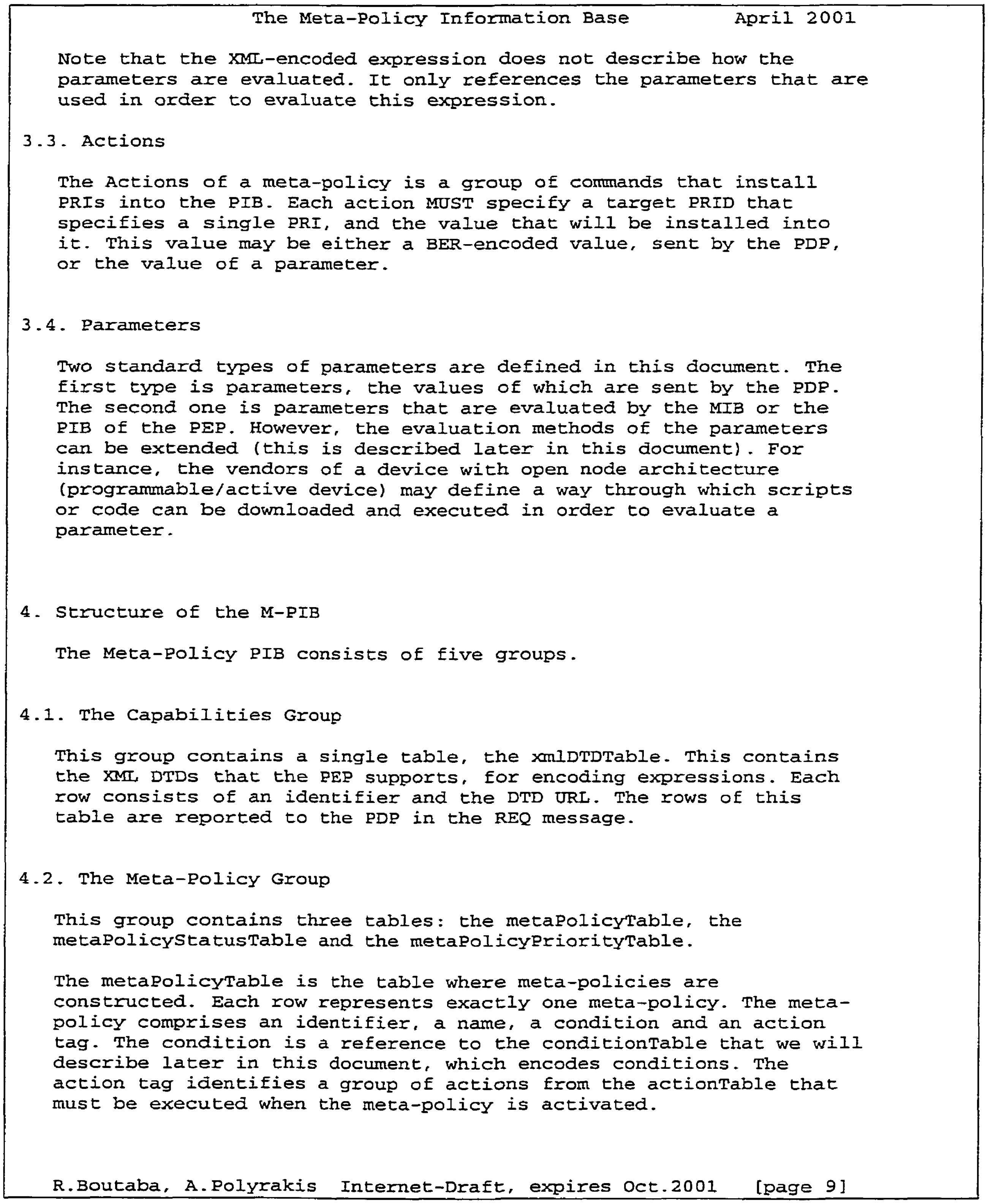
In order to encode and communicate such generic conditions, XML is used. The PEP supports some XML Document Type Definitions (DTDs), which describe the semantics of XML tags that can be used to described such an expression. For instance, a simple DTD that defines XML tags for encoding arithmetic expressions is presented in Appendix A*. The PDP encodes the condition (if this is feasible) according to one of these DTDs, and sends it to the PEP, notifying it which DTD it chose. The PEP MUST be able to interpret any kind of expressions encoded according to the DTDs that it supports (with the exception of some limitations like the size of the XML document, etc, that it reports to the PDP in the REQ message). In this way, complex expressions can be communicated from the PDP to the PEP and be evaluated by the latter. Notice that each atom conditions should be parametric (it does not make sense to use constant conditions), the DTDs MUST provide a way to reference to the parameters that are installed in the PEP, through their identifiers.

For example, suppose that the PDP needs to send to the PEP the expression A "A+B>7". The PEP has reported that it supports the DTD of Appendix A. In this case, the expression will be sent as:

```
<ar_cond comp='GT'>
      <expr>
         <expr><par>1</par><arop op='+'></arop><par>2</par></expr>
      </expr>
      <expr>
         <num>7</num>
      </expr>
</ar_cond>
```

(Parameters "A" and "B" are mapped to the Parameter IDs 1 and 2, respectively)

* Appendix A of this specification, not of the entire document



Note that the XML-encoded expression does not describe how the parameters are evaluated. It only references the parameters that are used in order to evaluate this expression.

3.3. Actions

The Actions of a meta-policy is a group of commands that install PRIs into the PIB. Each action MUST specify a target PRID that specifies a single PRI, and the value that will be installed into it. This value may be either a BER-encoded value, sent by the PDP, or the value of a parameter.

3.4. Parameters

Two standard types of parameters are defined in this document. The first type is parameters, the values of which are sent by the PDP. The second one is parameters that are evaluated by the MIB or the PIB of the PEP. However, the evaluation methods of the parameters can be extended (this is described later in this document). For instance, the vendors of a device with open node architecture (programmable/active device) may define a way through which scripts or code can be downloaded and executed in order to evaluate a parameter.

4. Structure of the M-PIB

The Meta-Policy PIB consists of five groups.

4.1. The Capabilities Group

This group contains a single table, the xmlDTDTable. This contains the XML DTDs that the PEP supports, for encoding expressions. Each row consists of an identifier and the DTD URL. The rows of this table are reported to the PDP in the REQ message.

4.2. The Meta-Policy Group

This group contains three tables: the metaPolicyTable, the metaPolicyStatusTable and the metaPolicyPriorityTable.

The metaPolicyTable is the table where meta-policies are constructed. Each row represents exactly one meta-policy. The meta-policy comprises an identifier, a name, a condition and an action tag. The condition is a reference to the conditionTable that we will describe later in this document, which encodes conditions. The action tag identifies a group of actions from the actionTable that must be executed when the meta-policy is activated.

The metaPolicyStatusTable is a table that AUGMENTS the previous table (this means that there is a 1-1 correspondence between the rows of these tables). Each row of this table reports whether the corresponding meta-policy is active, and whether it suppresses or it is suppressed by anoter meta-policy with higher priority. This table is used to report to the PDP the meta-policy status. This class, unlike the metaPolicy class, is only used to report the status of the meta-policies to the PDP and it cannot be modified by it.

Finally, the metaPolicyPriorityTable is used by the PDP in order to report to the PEP conflicting meta-policies, and direct it how to resolve the conflict. Each row identifies two meta-policies, and defines which one has the higher priority. Rows with two active meta-policies MUST NOT exist in this table.

4.3. The Condition Group

This group contains four tables: the conditionTable, the complexConditionTable, the booleanConditionTable and the generalConditionTable.

The conditionTable is the base table of this group. Each row represents a logical expression. It consists of an identifier and an attribute that defines whether the condition should be logically reversed (i.e., whether its negation must be computed, instead). Rows of this table MUST always be associated with rows of an other table that extends the base one.

Some (but not all) of the rows of this table are used in order to represent conditions of meta-policy. Other rows, though, can be used to break down a complex condition to simpler ones.

In order to achieve that, the complexConditionTable is used. This table EXTENDS the base conditionTable. Each row consists of two references to the conditionTable, and an operator. The references reference two other logical conditions, and the operator defines a logical operation between these two conditions. In this way, the row in this table forms a more complex condition. Obviously, the PDP must not install rows that reference themselves, either explicitly or implicitly.

The booleanConditionTable is a table that also extends the base table. Each row contains a reference to a parameter, which must be of type "TrueValue". The value of the condition is evaluated according to the value of this parameter.

Finally, the generalConditionTable is used to allow conditions to be evaluated through more complex expressions. Each row consists of a reference to the xmlDTDtable and a string, which encodes in XML an expression. The reference to the xmlDTDtable defines the XML DTD that must be used in order to interpret this expression. The expression encoded MUST be a logical expression, i.e., it MUST evaluate either true or false.

4.4. The Actions Group

This group consists of three tables: the actionTable, the actionValueTable and the actionParametricValueTable.

The actionTable is the base table for storing meta-policy actions. Each row contains a tag-reference attribute that groups the actions of a single meta-policy. Each row specifies the PRID of the PRI to be installed.

The value of the PRI is determined either in the actionValueTable or the actionParametricValueTable. Both tables EXTEND the base one and provide the value that must be installed for the specific target PRID. The former provides a BER-encoded value, while the latter specifies a parameter, from where the value is evaluated.

4.5. The Parameter Group

This group contains three tables: the parameterTable, the mibPibParameterTable and the PDPParameterTable.

The parameterTable is the base table for representing conditions. Each row constist of an identifier, a name and an attribute that denotes the type of the parameter. Each row in this table must be associated with a row of a table that EXTENDS this one.

The mibPibParameterTable is such a table. It defines a MIB or PIB identifier from where the parameter gets its value. Of course, this identifier must point to an existing variable. Each row also defines the frequency that this value will be updated.

The pdpParameterTable also extends the base parameterTable. Each row of this table contains a single attribute that encodes, in BER, a single value. The PDP sends the values for this row.

5. Definition of the Meta-Policy PIB

```
META-POLICY-PIB PIB-DEFINITIONS ::= BEGIN

IMPORTS
     Unsigned32, timeticks,
     MODULE-IDENTITY, OBJECT-TYPE,
     InstanceId, ReferenceId
          FROM COPS-PR-SPPI
     TEXTUAL-CONVENTION
```

```
                    The Meta-Policy Information Base         April 2001

          FROM SNMPv2-TC;

metaPolicyPib  MODULE-IDENTITY
     SUBJECT-CATEGORY { all }
     LAST-UPDATED "200104010000"
     ORGANIZATION "IETF"
     CONTACT-INFO " Andreas Polyrakis
                    Dept. of Computer Science,
                    University of Toronto,
                    10 King's College Road,
                    Toronto, Ontario,M5S 3G4, Canada.
                    e-mail: apolyr@cs.toronto.edu
                    Phone: ++1 (416) 978-4837
                    Fax: ++1 (416) 978 1931

                    Raouf Boutaba
                    Dept. of Computer Science,
                    University of Waterloo,
                    200 University Avenue West,
                    Waterloo, Ontario N2L 3G1, Canada
                    e-mail: rboutaba@bbcr.uwaterloo.ca
                    Phone: ++1 (519) 888 4567 ext.4820
                    Fax: ++1 (519) 885 1208"
     DESCRIPTION
          "The meta-policy PIB module. It contains the classes
          that are necessary for the provisioning of meta-policy
          related information. This module is applicable,
          but not mandatory, to all subject-categories"

     ::= { tbd }
-- The root OID for PRCs in the Meta-Policy PIB


---
--- Textual Conventions
---
BERValue ::= TEXTUAL-CONVENTION
     STATUS      current
     DESCRIPTION
          "A sequence of octets that encodes a value using BER.
          The suppoted BER types are:
          Type                 | BER identifier
          ---------------------|----------------
          INTEGER              | 02
          BIT STRING           | 03
          OCTET STRING         | 04
          NULL                 | 05
          OBJECT IDENTIFIER    | 06
          IP ADDRESS           | 40

          By using this type, the PEP can store values for different
types
          of parameters in the same class (PRC)."
"

R.Boutaba, A.Polyrakis  Internet-Draft, expires Oct.2001  [page 12]
```

```
      SYNTAX OCTET STRING (SIZE (0..16))

  XMLString ::= TEXTUAL-CONVENTION
         STATUS      current
         DESCRIPTION
            "A string that contains a logical expression encoded using
 XML.
            The semantics of the XML tags are defined in special DTDs,
 which
            the PEP has denoted that it supports to the PDP."
         SYNTAX OCTET STRING (SIZE (0..1024))
 ---
 --- End of Textual Conventions
 ---

 -----------------------------------------------
 -----------------------------------------------


 -- Meta-Policy Capabilities Group
 metaPolicyCapabilitiesClasses
      OBJECT IDENTIFIER ::= { metaPolicyPib 1 }


 ---
 --- Meta-Policy Capabilities Table
 ---
 xmlDTDTable OBJECT-TYPE
      SYNTAX SEQUENCE OF xmlDTDEntry
      PIB-ACCESS notify
      STATUS current
      DESCRIPTION
           "Each instance of this class specifies a PRC that
           identifies an XML DTD supported by the PEP for encoding
           logical expressions. If this class has no instances,
           then the PEP supports only expressions that are formed
           with boolean predicates and operators, and in this case
           the PDP MUST not attempt to install any XML-encoded
           expressions in the generalConditionTable."
      ::= { metaPolicyCapabilitiesClasses 1 }

 xmlDTDEntry OBJECT-TYPE
      SYNTAX MetaPolicyCapabilitiesEntry
      STATUS current
      DESCRIPTION
           "An instance of the xmlDTDTable class that determines an
           XML DTD that can be used to encode a logical expression"
      INDEX { metaPolicyPrid }
      ::= { metaPolicyTable 1 }

 XmlDTDEntry ::=
      SEQUENCE {
           xmlDTDPrid           InstanceId,
           xmlDTDURL          SnmpAdminString
```

```
     }

xmlDTDPrid OBJECT-TYPE
     SYNTAX InstanceId
     STATUS current
     DESCRIPTION
          "An arbitrary integer that uniquely identifies an
          instance of the xmlDTD class."
     ::= { xmlDTDEntry 1 }

xmlDTDURL OBJECT-TYPE
     SYNTAX SnmpAdminString
     STATUS current
     DESCRIPTION
          "The XML DTD URL. A string that indicates the URL of an
          XML DTD that can be used for encoding expressions.
          These DTDs can be defined either by standardization
          organizations, such as IETF, or be vendor specific.

          When the PDP receives a URL that uniquely identifies
          such a DTD, it knows that it may encode expressions
          according to this DTD that the PEP will be able to
          evaluate."
     ::= { xmlDTDEntry 2 }

--End of xmlDTDTable

-----------------------------------------------
-----------------------------------------------

-- Base Meta-Policy Group
metaPolicyClasses
     OBJECT IDENTIFIER ::= { metaPolicyPib 2 }


---
--- Meta-Policy Table
---
metaPolicyTable OBJECT-TYPE
     SYNTAX SEQUENCE OF metaPolicyEntry
     PIB-ACCESS INSTALL
     STATUS current
     DESCRIPTION
          "Each instance of this class specifies a PRC that
          represents a meta-policy. Each meta-policy, apart
          from a unique identifier and an optional name, it
          constists of a condition and a group of actions"
     ::= { metaPolicyClasses 1 }

metaPolicyEntry OBJECT-TYPE
     SYNTAX MetaPolicyEntry
     STATUS current
     DESCRIPTION
          "An instance of the metaPolicy Class that represents
```

```
            a meta-policy."
        INDEX { metaPolicyPrid }
        ::= { metaPolicyTable 1 }

   MetaPolicyEntry ::=
        SEQUENCE {
             metaPolicyPrid              InstanceId,
             metaPolicyName              SnmpAdminString,
             metaPolicyCondition         ReferenceId,
             metaPolicyActions           TagId
        }

   metaPolicyPrid OBJECT-TYPE
        SYNTAX InstanceId
        STATUS current
        DESCRIPTION
             "An arbitrary integer that uniquely identifies an
             instance of the metaPolicy class."
        ::= { metaPolicyEntry 1 }

   metaPolicyName OBJECT-TYPE
        SYNTAX SnmpAdminString
        STATUS current
        DESCRIPTION
             "A display string that represents the name of the
             meta-policy. It is reccomented that different
             meta-policies have different names. However, similar
             meta-policies may have the same name.
             Also, an empty string can be used as a name."
        ::= { metaPolicyEntry 2 }

   metaPolicyCondition OBJECT-TYPE
        SYNTAX ReferenceId
        PIB-REFERENCES contitionTable
        STATUS current
        DESCRIPTION
             "This attribute associates the specific meta-policy with
             a condition in the condition Class. The condition MUST
             exist when the meta-policy is installed. The meta-policy
             MUST always be assosiated with one condition (which means
             that the attribute can never be null/invalid."
        ::= { metaPolicyEntry 3 }

   metaPolicyActions OBJECT-TYPE
        SYNTAX TagId
        PIB-REFERENCES actionsTable
        STATUS current
        DESCRIPTION
             "A tag that maps this instance (meta-policy) to a group
             of actions in the actions Class. Although the tag should
             map to at least one action, there might be cases where a
             meta-policy is associated to no actions. However such
             cases should be avoided and only be temporal."
        ::= { metaPolicyEntry 4 }
```

```
--End of metaPolicyTable

--
-- Meta-Policy Status Table
--
metaPolicyStatusTable OBJECT-TYPE
     SYNTAX SEQUENCE OF metaPolicyStatusEntry
     PIB-ACCESS REPORT-ONLY
     STATUS current
     DESCRIPTION
          "This class augments the metaPolicy class.
          Each instance of this class defines a PRC that is used
          in order to report to the PDP the status of the
          meta-policies.

          Also, information form this table can be used as a
          parameter to another meta-policy, as an alternative
          way to ensure that two priorities cannot be
          activated at the same time."
     ::= { metaPolicyClasses 2 }

metaPolicyStatusEntry OBJECT-TYPE
     SYNTAX MetaPolicyStatusEntry
     STATUS current
     DESCRIPTION
          "An instance of the metaPolicyStatus class that reports
          the status of the corresponding meta-policy in the
          metaPolicy class."
     AUGMENTS { metaPolicyEntry }
     ::= { metaPolicyStatusTable 1 }

metaPolicyStatusEntry ::=
     SEQUENCE {
          metaPolicyActive     TruthValue,
          metaPolicySuppressed TruthValue
     }

metaPolicyActive OBJECT-TYPE
     SYNTAX TruthValue
     STATUS current
     DESCRIPTION
          "True while the meta-policy is active"
     ::= { metaPolicyStatusEntry 1 }

metaPolicySuppress OBJECT-TYPE
     SYNTAX TruthValue
     STATUS current
     DESCRIPTION
          "If this meta-policy is prevented from being active by
          an other meta-policy (but its conditions are met), this
          attribute is set to true.

          If this meta-policy prevents another meta-policy from
          being active, then this attribute is true.
          In other words:
          Active | Suppr.|
          --------------------------------
           true  | true  | meta-policy active,
                         | it suppresses another one
           true  | false | meta-policy active,
                         | does not suppress another one
           false | true  | meta-policy inactive
                         | because it is suppressed by another one
           false | false | meta-policy inactive because
                         | the conditions are not met
          "
     ::= { metaPolicyStatusEntry 2 }
--End of metaPolicyStatusTable


---
--- Meta-Policy Priority Table
---
metaPolicyPriorityTable OBJECT-TYPE
     SYNTAX SEQUENCE OF metaPolicyPriorityEntry
     PIB-ACCESS INSTALL
     STATUS current
     DESCRIPTION
          "This table reports conflicting meta-policies.
          When a meta-policy needs to be activated, the PEP
          MUST check if it is conflicting with another meta-policy,
          which is already active or needs to be activated at the
          same time. If so, the one that is referenced in the
          higherPriority attribute is activated and the other one
          is deactivated or remains deactivated. Similarly, when a
          meta-policy is deactivated, the PEP must check if a
          lower-priority meta-policy must now be activated."
     ::= { metaPolicyClasses 3 }

metaPolicyPriorityEntry OBJECT-TYPE
     SYNTAX MetaPolicyPriorityEntry
     STATUS current
     DESCRIPTION
          "An instance of the metaPolicyPriority Class that
          identifies the relative priority between two
          meta-policies."
     INDEX { metaPolicyPrid }
     ::= { metaPolicyPriorityTable 1 }

MetaPolicyPriorityEntry ::=
     SEQUENCE {
          metaPolicyPriorityPrid   InstanceId,
          higherPriority           ReferenceId,
          lowerPriority            ReferenceId,
     }

metaPolicyPriorityPrid OBJECT-TYPE
```

```
      SYNTAX InstanceId
      STATUS current
      DESCRIPTION
           "An arbitrary integer  that uniquely identifies an
           instance of the metaPolicyPriority class."
      ::= { metaPolicyPriorityEntry 1 }

higherPriority OBJECT-TYPE
      SYNTAX ReferenceId
      PIB-REFERENCES metaPolicyTable
      STATUS current
      DESCRIPTION
           "This attribute references to the meta-policy that
           has higher priority than the one referenced by the
           lowerPriority attribute"
      ::= { metaPolicyPriorityEntry 2 }

lowerPriority OBJECT-TYPE
      SYNTAX ReferenceId
      PIB-REFERENCES metaPolicyTable
      STATUS current
      DESCRIPTION
           "This attribute references to the meta-policy that
           has lower priority than the one referenced by the
           higherPriority attribute"
      ::= { metaPolicyPriorityEntry 3 }

--End of metaPolicyPriorityTable
-----------------------------------------------
----------------------------------------------


-- Condition Group
conditionClasses
      OBJECT IDENTIFIER ::= { metaPolicyPib 3 }


--
-- Condition Table
--
conditionTable OBJECT-TYPE
      SYNTAX SEQUENCE OF conditionEntry
      PIB-ACCESS INSTALL
      STATUS current
      DESCRIPTION
           "Each instance of this PRC represents a boolean
           expression. The conditionss of the meta-policies are
           instances of this class. However, if the condition of
           a meta-policy contains more than one predicate, the
           predicates are also instances of this PRC.

           For instance, Suppose that we want to encode a condition
           A, which is evaluated as ( B OR C ), where B and C some
           other boolean expressions.
```

```
          In this case, A, B and C are instances of this PRC.

          All instances of this PRC MUST be extended by an instance
          of one of the rest PRCs of this group, in order to denote
          if this condition should be evaluated based on simpler
          conditions, if it is a boolean operand or an other
          logical expression."
      ::= { conditionClasses 1 }

conditionEntry OBJECT-TYPE
      SYNTAX ConditionEntry
      STATUS current
      DESCRIPTION
          "An instance of the condition Class that defines a
          boolean condition"
      INDEX { conditionIndex }
      ::= { conditionTable 1 }

ConditionEntry ::=
      SEQUENCE {
          conditionPrid       InstanceId,
          conditionReverse    Truevalue
      }

conditionPrid OBJECT-TYPE
      SYNTAX InstanceId
      STATUS current
      DESCRIPTION
          "An arbitrary integer that uniquely identifies an
          instance of the condition class."
      ::= { conditionEntry 1 }

conditionReverse OBJECT-TYPE
      SYNTAX Truevalue
      STATUS current
      DESCRIPTION
          "if true, the negation of the logical expression
           is evaluated, instead."
      ::= { conditionEntry 2 }
-- END OF conditionTable

--
-- Complex Condition Table
--
complexConditionTable OBJECT-TYPE
      SYNTAX SEQUENCE OF complexConditionEntry
      PIB-ACCESS INSTALL
      STATUS current
      DESCRIPTION
          "Each instance of this PRC represents a complex
          condition. It consists of two simplier conditions,
          and a logical operator that determines how the two
          terms are assosiated to compose the more
          complicated condition"
      ::= { conditionClasses 2 }

  complexConditionEntry OBJECT-TYPE
        SYNTAX ComplexConditionEntry
        STATUS current
        DESCRIPTION
              "An instance of the complexCondition class that breaks a
              complex condition into two simpler ones."
        EXTENDS { conditionTable }
        ::= { complexConditionTable }

  ComplexConditionEntry ::=
        SEQUENCE {
              operator       Unsigned32,
              leftTerm       ReferenceId,
              rightTerm      ReferenceId
        }

  operator OBJECT-TYPE
        SYNTAX Unsigned32 {
              AND (0),
              OR (1)
              }
        STATUS current
        DESCRIPTION
              "The logical operator in the complex condition"
        ::= { complexConditionEntry 1 }

  leftTerm OBJECT-TYPE
        SYNTAX ReferenceId
        PIB-REFERENCES conditionTable
        STATUS current
        DESCRIPTION
              "A reference to the first simpler condition."
        ::= { complexConditionEntry 2 }

  rightTerm OBJECT-TYPE
        SYNTAX ReferenceId
        PIB-REFERENCES conditionTable
        STATUS current
        DESCRIPTION
              "A reference to the second simpler condition."
        ::= { complexConditionEntry 3 }
  -- END OF complexConditionTable

  --
  -- Boolean Condition Expression Table
  --
  booleanConditionTable OBJECT-TYPE
        SYNTAX SEQUENCE OF booleanConditionEntry
        PIB-ACCESS INSTALL
        STATUS mandatory
        DESCRIPTION
              "Each instance of this class extends the condition class
```

```
          and represents a boolean parameter from which the
          condition is evaluated."
     ::= { metaPolicyPibClasses 2 }

booleanConditionEntry OBJECT-TYPE
     SYNTAX BooleanConditionEntry
     STATUS mandatory
     DESCRIPTION
          "An instance of the booleanCondition class that defines
          the boolean parameter that gives values to the
          corresponding condition."
     EXTENDS { conditionTable }
     ::= { booleanConditionTable 1 }

BooleanConditionEntry ::=
     SEQUENCE {
          parameterReference  ReferenceId
     }

parameterReference OBJECT-TYPE
     SYNTAX ReferenceId
     PIB-REFERENCES parameterTable
     STATUS current
     DESCRIPTION
          "A reference to a parameter from where the condition is
          evaluated. This condition MUST be of type boolean
          (Truthvalue)."
     ::= { booleanConditionEntry 1 }
-- End of booleanConditionTable

--
-- Generic Condition Table
--
genericConditionTable OBJECT-TYPE
     SYNTAX SEQUENCE OF genericConditionEntry
     PIB-ACCESS INSTALL
     STATUS current
     DESCRIPTION
          "Each instance of this class extends the condition class
          and assosiates the corresponding condition with a complex
          logical expression, from where the condition is
          evaluated."
     ::= { conditionClasses 2 }

genericConditionEntry OBJECT-TYPE
     SYNTAX GenericConditionEntry
     STATUS current
     DESCRIPTION
          "An instance of the generalCondition class that defines
          the logical expression for the corresponding condition
          of the condition class."
     EXTENDS { generalConditionTable }
     ::= { conditionNumericalExpressionTable }
```

```
                 The Meta-Policy Information Base          April 2001

  GenericConditionEntry ::=
       SEQUENCE {
          xmlDTDRef   ReferenceId,
            xmlCondition   XMLString
       }

  xmlDTDRef OBJECT-TYPE
       SYNTAX ReferenceId
       PIB-REFERENCES xmlDTDTable
       STATUS current
       DESCRIPTION
            "A reference to the xmlDTD class that determies which
            of the XML DTDs that this PEP supports is used in
            order to encode the expression."
       ::= { genericConditionEntry 1 }

  xmlCondition OBJECT-TYPE
       SYNTAX XMLString
       STATUS mandatory
       DESCRIPTION
            "The XML-encoded expression."
       ::={ genericConditionEntry 2 }
  -- End of genericConditionTable

  ----------------------------------------------
  ----------------------------------------------


  -- Actions Group
  actionClasses
       OBJECT IDENTIFIER ::= { metaPolicyPib 4 }


  --
  -- Actions Table
  --
  actionTable OBJECT-TYPE
       SYNTAX SEQUENCE of actionEntry
       PIB-ACCESS INSTALL
       STATUS current
       DESCRIPTION
            "Each instance of this class stores an action of
            a meta-policy."
       ::= { actionClasses 1 }

  actionEntry OBJECT-TYPE
       SYNTAX ActionEntry
       STATUS current
       DESCRIPTION
            "An instance of the action class that stores an action
            of a meta-policy."
       INDEX { actionPrid }
       ::= { actionTable 1 }


  R.Boutaba, A.Polyrakis  Internet-Draft, expires Oct.2001  [page 22]
```

```
ActionEntry ::=
     SEQUENCE {
          actionPrid          InstanceId,
          actionRefTag        TagReferenceId,
          actionTargetPrid    Prid
     }

actionPrid OBJECT-TYPE
     SYNTAX InstanceId
     STATUS current
     DESCRIPTION
          "An arbitrary integer that uniquely identifies an
          instance of the action class."
     ::= { actionEntry 1 }

actionRefTag OBJECT-TYPE
     SYNTAX TagReferenceId
     PIB-TAG metaPolicyActions
     STATUS current
     DESCRIPTION
          "An attribute that defines a Tag Group of actions.
          All actions with the same tag are grouped as the actions
          of a single meta-policy."
     ::={ actionEntry 2 }

actionTargetPrid OBJECT-TYPE
     SYNTAX Prid
     STATUS current
     DESCRIPTION
          "The PRID of the PRI to be installed/updated.
          The PRID must point to a single PRI."
     ::={ actionEntry 3 }
-- END OF actionsTable

--
-- Action Value table
--
actionValueTable OBJECT-TYPE
     SYNTAX SEQUENCE OF actionValueEntry
     PIB-ACCESS INSTALL
     STATUS current
     DESCRIPTION
          "Each instance of this class extends the corresponding
          instance of the action class. It provides the BER-encoded
value
          that will be installed at the corresponding PRI."
     ::= { actionClasses 2 }

actionValueEntry OBJECT-TYPE
     SYNTAX ActionsValueEntry
     STATUS current
     DESCRIPTION
          "An insance of the actionValue class. It provides
          the value (encoded with BER) that will be installed at
          the PRI denoted by the corresponding instance of the
          action class."
     EXTENDS { actionEntry }
     ::= { actionValueTable 1 }

ActionValueEntry ::=
     SEQUENCE {
          ActionValueEpd      BERValue
     }

actionValueEpd OBJECT-TYPE
     SYNTAX BERValue
     STATUS current
     DESCRIPTION
          "This attribute contains the BER-encoded value of the
          PRI to be installed/updated."
     ::={ actionValueEntry 1 }
-- END OF actionValueTable

--
-- Action Parametric Value Table
--
actionParametricValueTable OBJECT-TYPE
     SYNTAX SEQUENCE OF actionParametricValueEntry
     PIB-ACCESS INSTALL
     STATUS current
     DESCRIPTION
          "Each instance of this class that extends the
           corresponding instance of the action class. It provides
with the
           parametric value that will be installed at the
corresponding PRI."
     ::= { actionClasses 3 }

actionParametricValueEntry OBJECT-TYPE
     SYNTAX ActionParametricValueEntry
     STATUS current
     DESCRIPTION
          "An insance of the actionValue class. It provides with
          the parametric value that will be installed at the PRI
           denoted by the corresponding instance of the action
           class."
     EXTENDS { actionEntry }
     ::= { actionParametricValueTable 1 }

ActionParametricValueEntry ::=
     SEQUENCE {
          ParameterRef       ReferenceId
     }

ParameterRef OBJECT-TYPE
     SYNTAX ReferenceId
     PIB-REFERENCES parameterTable
     STATUS current
```

```
                    The Meta-Policy Information Base         April 2001

          DESCRIPTION
               "A reference to a the parameter, from where the value
               of the installed PRI should be obtained. Whenever the
               value of the parameter changes, the installed PRI
               MUST be updated."
          ::={ actionParametricValueEntry 1 }
     -- END OF actionParametricValueTable

     ------------------------------------------------
     ------------------------------------------------


     -- Parameter Group
     parameterClasses
          OBJECT IDENTIFIER : := { metaPolicyPib 5 }


     -
     - Parameter Table
     -
     parameterTable OBJECT-TYPE
          SYNTAX SEQUENCE OF parameterEntry
          PIB-ACCESS INSTALL
          STATUS current
          DESCRIPTION
               "Each instance of this class defines a parameter
               that has been installed on the PEP. This class
               MUST be extended by a class that defines how
               the value of the parameter will be evaluated."
          ::= { parameterClasses 1}

     parameterEntry OBJECT-TYPE
          SYNTAX ParameterEntry
          STATUS current
          DESCRIPTION
               "An instance of the parameter class that installs
               a parameter into the PEP."
          INDEX { parameterPrid }
          ::= { parameterTable 1 }

     ParameterEntry ::=
          SEQUENCE {
               parameterPrid        InstanceId,
               parameterName        SNMPAdminString,
               parameterType        Unsigned32
          }

     parameterPrid OBJECT-TYPE
          SYNTAX InstanceId
          STATUS current
          DESCRIPTION
               "An arbitrary integer that uniquely identifies an
               instance of the parameter class."
          ::= { parameterEntry 1 }

     R.Boutaba, A.Polyrakis  Internet-Draft, expires Oct.2001  [page 25]
```

```
parameterNameOBJECT-TYPE
     SYNTAX SNMPAdminString
     STATUS current
     DESCRIPTION
          "A string that represents the name of the parameter.
          It is reccomented that different parameter have different
          names. However, similar parameter may have the same name.
          Also, an empty string can be used as a name."
     ::= { parameterEntry 2 }

parameterType
     SYNTAX Unsigned32 {
          INTEGER (02)
          BIT STRING (03)
          OCTET STRING (04)
          NULL (05)
          OBJECT IDENTIFIER (06)
          IP ADDRESS (40)
          }
     STATUS current
     DESCRIPTION
          "The BER type of the parameter.
          The suppoted BER types are:
          Type                  | BER identifier
          ----------------------|----------------
          INTEGER               | 02
          BIT STRING            | 03
          OCTET STRING          | 04
          NULL                  | 05
          OBJECT IDENTIFIER     | 06
          IP ADDRESS            | 40"
     ::= { parameterEntry 3 }
-- END OF parameterTable

--
-- MIBPIB Parameter Table
--
mibPibParameterTable OBJECT-TYPE
     SYNTAX SEQUENCE OF mibPibParameterEntry
     PIB-ACCESS INSTALL
     STATUS current
     DESCRIPTION
          "This class extends the parameter class.
          Each instance of this class assosiates to the
          corresponding parameter a MIB or PIB variable, from
          where the parameter is evaluated"
     ::= { parameterClasses 2 }

mibPibParameterEntry OBJECT-TYPE
     SYNTAX MibPibParameterEntry
     STATUS current
     DESCRIPTION
          "An instance of the mibPibParameter class that provides
          the identifier of the MIB/PIB variable from where the
          corresponding parameter is evaluated."
     EXTENDS { parameterEntry }
     ::= { mibPibParameterTable 1 }

MibPibParameterEntry ::=
     SEQUENCE {
          targetOID        OBJECT-IDENTIFIER,
          EvaluationFrequency timeticks
          }

targetOID OBJECT-TYPE
     SYNTAX OBJECT-IDENTIFIER
     PIB-ACCESS INSTALL
     STATUS current
     DESCRIPTION
          "The object identifier of the MIB/PIB variable.
          The MIB/PIB variable MUST exist in the MIB/PIB of the
          device. Also, the type of the target variable MUST be
          compatible with the type of the corresponding PRI of the
          parameter Class."
     ::={ mibPibParameterEntry 1 }

EvaluationFrequency OBJECT-TYPE
     SYNTAX timeticks
     STATUS current
     DESCRIPTION
          "The frequency of updating the parameter in milliseconds"
     ::={ mibPibParameterEntry 2 }
-- END of mibPibParameterTable


--
-- PDP Parameter Table
--
pdpParameterTable OBJECT-TYPE
     SYNTAX SEQUENCE OF pdpParameterEntry
     PIB-ACCESS INSTALL
     STATUS current
     DESCRIPTION
          "This class extends the parameter class. Each instance
          of this class contains the value of the corresponding
          paramter. This value is send by the PDP and updated
          whenever necessary."
     ::= { parameterClasses 3 }

pdpParameterEntry OBJECT-TYPE
     SYNTAX PdpParameterEntry
     STATUS current
     DESCRIPTION
          "An instance of the pdpParameter class that stores the
          value, sent by the PDP, for the corresponding parameter."
     INDEX { parameterIndex }
     ::= { pdpParametersTable 1 }
```

```
   PdpParameterEntry ::=
        SEQUENCE {
             lastValue          BERValue
        }

   lastValue OBJECT-TYPE
        SYNTAX BERValue
        STATUS current
        DESCRIPTION
             "The latest value of the parameter, encoded with BER.
             The BER-encoded value must be of the same type as the
             corresponding PRI of the parameter class."
        ::={ pdpParameterEntry 1 }
   -- END OF pdpParameterTable


   END
```

Authors' Information

Andreas Polyrakis
Dept. of Computer Science,
University of Toronto,
10 King's College Road,
Toronto, Ontario,M5S 3G4, Canada.
Phone: ++1 (416) 978-4837
Fax: ++1 (416) 978 1931

Raouf Boutaba
Dept. of Computer Science,
University of Waterloo,
200 University Avenue West,
Waterloo, Ontario N2L 3G1, Canada
e-mail: rboutaba@bbcr.uwaterloo.ca
Phone: ++1 (519) 888 4567 ext.4820
Fax: ++1 (519) 885 1208

References

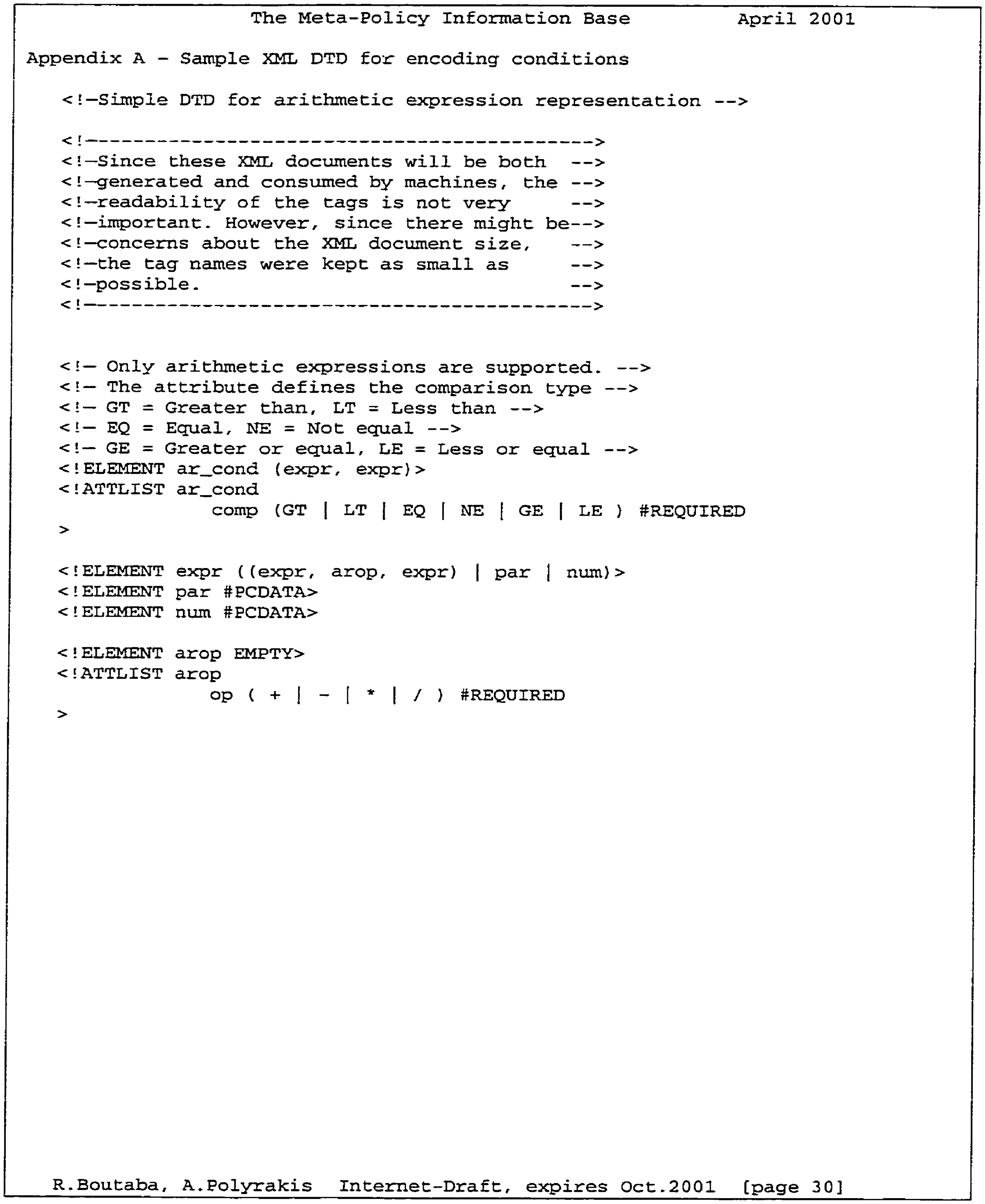
[P-TERM] A. Westerinen, J. Schnizlein, J. Strassner, Mark Scherling, Bob Quinn, Jay Perry, Shai Herzog, An-Ni Huynh, Mark Carlson, "Policy Terminology", Internet draft, draft-ietf-policy-terminology-00.txt, July 2000

[RAP-FRM] R. Yavatkar, D. Pendarakis, "A Framework for Policy-based Admission Control", IETF RFC 2753, January 2000.

[COPS] Boyle, J., Cohen, R., Durham, D., Herzog, S., Raja, R., Sastry, A., "The COPS (Common Open Policy Service) Protocol", IETF RFC 2748, Proposed Standard, January 2000.

[COPS-PR] K. Chan, D. Durham, S. Gai, S. Herzog, K. McCloghrie, F. Reichmeyer, J. Seligson, A. Smith, R. Yavatkar, "COPS Usage for Policy Provisioning," draft-ietf-rap-pr-05.txt, October 30, 2000.

[SPPI] K. McCloghrie, et.al., "Structure of Policy Provisioning Information," draft-ietf-rap-sppi-05.txt, February 2001.



The Meta-Policy Information Base   April 2001

Appendix A - Sample XML DTD for encoding conditions

```
   <!--Simple DTD for arithmetic expression representation -->

   <!------------------------------------------->
   <!--Since these XML documents will be both  -->
   <!--generated and consumed by machines, the -->
   <!--readability of the tags is not very     -->
   <!--important. However, since there might be-->
   <!--concerns about the XML document size,   -->
   <!--the tag names were kept as small as     -->
   <!--possible.                               -->
   <!------------------------------------------->


   <!-- Only arithmetic expressions are supported. -->
   <!-- The attribute defines the comparison type -->
   <!-- GT = Greater than, LT = Less than -->
   <!-- EQ = Equal, NE = Not equal -->
   <!-- GE = Greater or equal, LE = Less or equal -->
   <!ELEMENT ar_cond (expr, expr)>
   <!ATTLIST ar_cond
                comp (GT | LT | EQ | NE | GE | LE ) #REQUIRED
   >

   <!ELEMENT expr ((expr, arop, expr) | par | num)>
   <!ELEMENT par #PCDATA>
   <!ELEMENT num #PCDATA>

   <!ELEMENT arop EMPTY>
   <!ATTLIST arop
                op ( + | - | * | / ) #REQUIRED
   >
```

R.Boutaba, A.Polyrakis   Internet-Draft, expires Oct.2001   [page 30]

Full Copyright Statement

Appendix B. Related Publications (Abstracts)

TOWARDS EXTENSIBLE POLICY ENFORCEMENT POINTS [40]

IEEE Workshop on Policies for Distributed Systems and Networks; Bristol, U.K.; 29-31 January, 2001; pp. 247-261

Raouf Boutaba
University of Waterloo
Dept. of Computer Science
rboutaba@bbcr.uwaterloo.ca

Andreas Polyrakis
University of Toronto
Dept. of Computer Science
apolyr@cs.toronto.edu

Abstract

For several years, Configuration Management has been conducted mainly through command line or SNMP. However, while computer networks started growing bigger in size and complexity, it became apparent that these approaches suffer from significant scalability and efficiency limitations. Policy-Based Networking (PBN) seems to be a promising alternative for Configuration Management, and has already received significant attention. This approach involves the processing of the network policies by special servers (PDPs) that send the appropriate configuration data to the Policy Enforcement Points (PEPs) that reside on the managed entities. COPS and its extension for policy provisioning, COPS-PR, are currently being developed by IETF to implement PBN. In COPS-PR, the PDP installs to the PEP policies that the latter should enforce. However, the types of policies that the PEP can understand are limited and hardwired to it by the manufacturer. In this paper, we propose an architecture that attempts to raise such limitations and push the decision taking from the policy servers to the managed devices.

PROJECTING FCAPS TO ACTIVE NETWORKS [4]

accepted in IEEE EntNet 2001; Atlanta, GA, USA; 4-6 June, 2001

Raouf Boutaba
University of Waterloo
Dept. of Computer Science
rboutaba@bbcr.uwaterloo.ca

Andreas Polyrakis
University of Toronto
Dept. of Computer Science
apolyr@cs.toronto.edu

Abstract

Active Networks is one of the most promising and discussed trends in the area of Computer Networks. It allows us to program the network nodes to perform advanced operations and computations, and thus, control their behavior. These properties change considerably the scenery in the area of computer networks and, consequently, affect Network Management. Indeed, Active Networks do not only open the way to enhance current management techniques and improve their efficiency, but they also create perspectives to deploy novel ones. This paper attempts to present the impact of Active Networks upon the current Network Management techniques. In order to achieve this, Network Management is examined through the five areas of the FCAPS framework; for each one, the limitations of the current applications and tools are presented, and how these can be overcome by exploiting Active Network properties is discussed. The contribution of this paper is to gather and classify the various ideas found in the literature in this area, combine then and propose some new ones

COPS-PR WITH META-POLICY SUPPORT [41]

Published as an independent submission at IETF, April 2001

Raouf Boutaba
University of Waterloo
Dept. of Computer Science
rboutaba@bbcr.uwaterloo.ca

Andreas Polyrakis
University of Toronto
Dept. of Computer Science
apolyr@cs.toronto.edu

Abstract

In COPS-PR, the (clients of the) PEPs use special structures, called Policy Information Bases (PIBs) that store the policies that are sent by the PDPs. PIBs are well-defined structures that are not meant to be modified to adapt to the needs of each network. This makes COPS-PR PEPs rigid and inflexible. This document describes an extension of the COPS-PR protocol that allows the PEPs to store meta-policies that control the content of their PIBs. The set of meta-policies that the PEPs can store is not predefined and customized policies can be supported. The use of meta-policies pushes intelligence towards the PEPs and makes them more self-dependent. In this way, the model becomes more distributed, scalable and fault-tolerant, while the bandwidth consumption and the (real-time) processing load of the PDP are reduced.