Towards Extensible Policy Enforcement Points*

Raouf Boutaba[1], Andreas Polyrakis[2]

[1] Dept. of Computer Science, University of Waterloo,
200 University Avenue West, Ontario, N2L 3G1, Canada
rboutaba@bbcr.uwaterloo.ca

[2] Dept. of Computer Science, University of Toronto
10 King's College Road, Toronto, Ontario, M5S 3G4, Canada
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.

1. Introduction

Configuring network devices, such as routers and switches, has always been a hard task. In most cases, the administrator was required to configure each of the devices independently, even when these were configured to operate similarly. Initially, the configuration was done through the Command Line Interfaces of the devices; later, other technologies attempted to automate this process. The most widely used one was SNMP, which, although designed for monitoring purposes, gave a satisfactory solution to the problem. However, as the networks start growing considerably both in size (number of managed nodes) and in complexity (different types of devices, number of configuration parameters), significant scalability and efficiency problems appeared in the existing configuration methods. IETF attempts now to address such issues through the next version of SNMP (SNMP v.3). However, there are serious doubts whether SNMP will eventually manage to overcome its limitations and

* This research work is supported by research grants from Nortel Networks and the Natural Sciences and Engineering Research Council of Canada.

become the dominant protocol for Configuration Management again. On the other hand, an alternative to SNMP that has already received considerable attention is Policy-Based Networking (PBN) [1], [2], [3].

The basic concepts in PBN are the control/management policies [4], i.e. the rules that govern the network behavior. The administrator edits the policies in a management tool that performs syntax, semantics and basic conflict checking. These policies are then distributed, either directly or through the use of a directory, to special policy servers, called Policy Decision Points (PDPs) [5], [6]. The PDPs process these policies, along with other data such as network state information, and take policy decisions regarding what policies should be enforced and how this will happen. These policies are sent as configuration data to the appropriate Policy Enforcement Points (PEPs), which reside on the managed devices and are responsible for installing and enforcing them. Typically, PDPs run on dedicated servers and PEPs on the managed entities. Usually each device is controlled by one PEP, and a single PDP may control several PEPs. However, different configuration schemata may exist [7]. PBN is demonstrated in Figure 1 [3].

PBN is based on a client-server model of interaction between PEPs and PDPs. It distinguishes two modes of operation: the outsourcing and the provisioning [6]. In the outsourcing model, when the PEP receives a network event that it does not know how to treat, it issues a request to the (appropriate) PDP, notifying it for the event occurrence. The PDP replies to the PEP by sending configuration data that must be installed in order to respond to the event. On the other hand, in the provisioning model, as soon as the PEP connects to the PDP, the latter sends to the former the policies that must be enforced. Of course, these policies are in a format that the PEP can understand. The policies are stored in the PEP, and all incoming events are served according to them. In both cases, the PDP is aware of the policies enforced by the PEP, and if it may decide to update them by installing, deleting or replacing them, whenever it decides that they no longer reflect the desired behavior.

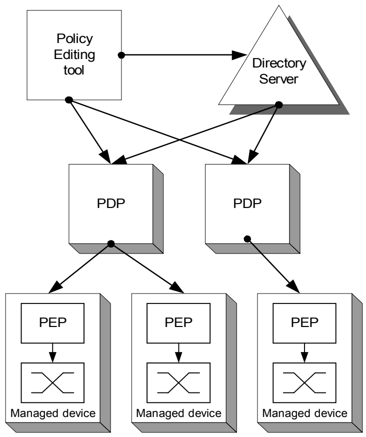


Figure 1. The Policy-Based Networking basic schema

2. COPS and COPS-PR

IETF has proposed the Common Open Policy Service (COPS) [8] as a protocol to implement the described architecture. COPS has recently received considerable attention, and applications based on it have already emerged [8], [9], [10].

The COPS protocol relies on a TCP connection for the exchange of data between PEPs and PDPs. Each PEP may support a number of client-types for the different policy areas (security, QoS, admission control, accounting, etc); those client-types are the actual enforcers of the policies. The clients connect to the appropriate PDP (they are redirected there by the default PDP) and register information about themselves and the device that they serve. Note that a PEP may have clients that each connects to a different PDP. In the outsourcing mode, the clients report to the PDP the events that they do not know how to handle, and the PDP sends them configuration data that define how to handle those events. The PDP may decide to update this data at any time, in a provisioning style. 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. 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. However, COPS does not define the content of the messages; this has to be defined for each client-type separately.

COPS-PR [11] describes the use of COPS for support of policy provisioning. COPS-PR clients have a well-defined Policy Information Base (PIB) ([11], [12]), a structure similar to a MIB, where they store all the policies that are sent by the PDP. Each client reports the capabilities of its PIB to the PDP when it connects to it. The PDP has to maintain the PIB of the PEP updated, according to the current network state and policies. The PEP has to enforce the policies that exist in its PIB. Whenever there is a change to the policies or the state of the network, the PDP has to check whether the PIBs of its PEPs need to be updated with new policies. COPS-PR assumes that the various clients of a PEP serve non-overlapping policing areas. Finally, COPS-PR does not define details for any PIB; this has to be done in separate documents for each management area.

Further description of COPS and COPS-PR is out of the scope of this paper; however the reader may refer to [8], [11] for more details.

3. The Motivation

In COPS-PR, the PDP installs policies in the PIBs of its clients in an attempt to make the network behave in accordance with the policies that the administrator has defined. These policies can be represented in the form:

$$\textit{If } [(c_1)\ o_1\ (c_2)\ o_2\ (c_3)\ \ldots] \textit{ then } \{(a_1) \textit{ and } (a_2) \textit{ and } \ldots\}$$

Where c_i are conditions, o_i are logical operators, a_i, b_i are actions.

Each network element that needs to enforce policies has a PEP. The PEP may have more than one client of different client-types, since each client-type is designated for different policing areas. Each client connects to a PDP, which controls the information of its PIB and, consequently, the device. Throughout this paper, we examine the interaction between a PDP and a single client. Thus, in many cases when we refer to a PEP, we may actually mean the client of the PEP. However, this should be obvious from the context.

Given a specific client of a PEP of a network element, only some of the defined policies may apply to it. However, even if a policy applies to a client, some of its actions may refer to other devices, and some others may be too generic, since they determine the behavior of several devices. Hence, for that specific client, the policy may be considered equivalent to a policy p, translated for the client:

> *If* $[(c_1)\ o_1\ (c_2)\ o_2\ (c_3)\ ...]$ *then* $\{(a_1')$ *and* (a_2') *and* $...\}$

This policy has to be sent by the PDP to the PEP and be enforced to the device. However, the policies that the PEP can store and understand are bound both by the definition of the PIB that allows only predefined types of conditions and actions to be stored, and by the capability of the enforcing mechanism to interpret them (i.e., even if the PIB was able to store any kind of policy, the PEP might not be able to interpret, evaluate and enforce it). Therefore, the PDP needs to transform the policies, from the general form, to a form that the PEP can understand. In order to do so, it must evaluate all the conditions c_i that cannot be stored in the PIB (and, hence, be evaluated by the PEP), as well as any possible parameters in the actions of the policy.

In other words, the PEP can store specific and limited types of policies p_i in its PIB. The PDP has to convert the set of policies to match one of the supported policy types, p_i. Since some policies do not directly fold into any of those types, the PDP has to evaluate the conditions and action parameters that cannot be fitted in the PIB, and according to their values, transform and send the policy according to one of the supported policy formats.

The following example demonstrates the previous. Suppose that the PIB of a router supports only policies (filters) of the form:

> *"if (packet IP matches X) then allow packet"*
> *"if (packet IP matches Y) then deny packet"*

However, the administrator has set a policy

> *"Do not allow access from network A between 5pm and 8am",*

or equivalently,

> *"if ((packet IP matches A) and (time between 8am and 5pm)) then allow"*
> *"if ((packet IP matches A) and (time between 5pm and 8am)) then deny"*

In this case, the PDP has to monitor the time, and depending on its value, send either the policy p_1:"if (packet IP matches with A) then allow" or the policy p_2:"if (packet IP matches with A) then deny". In the general case, the initial policy may also be parametric; in this case, specific values must be given to the parameters, prior to being send to the PEP.

We observe that the role of the PDP is to monitor the conditions that the PEP cannot understand and send the appropriate policies to the PEP according to the values of these conditions. Besides, the PDP may have to embed to the policies that it sends the values of some parameters of the original policy. Nevertheless, if the PEP (i) could store the initial policy p and (ii) could be aware of the values of all relevant conditions and parameters (hereinafter, both will be called "parameters"), it would be able to take the same policing decisions by itself. Moreover, some of these parameters might be possible and efficient to be evaluated by the PEP, supposing that it could be directed how to do so. For example, if the PEP resides on the central router of a network, it may evaluate the condition "general congestion to the network", as long as it is directed to look to the MIB of the router for the appropriate values.

The proposed architecture attempts to extend COPS-PR PEPs so as to become able to (i) store policies of non-predefined types (ii) be notified for the value of all the parameters that they cannot evaluate and (iii) be programmed to self-evaluate some of those parameters, whenever this is considered efficient.

4. The Architecture

The proposed architecture, demonstrated in Figure 2, attempts to address the goals described previously. Three basic decisions drove the design:

- For interoperability reasons, the PDP and PEP should adhere to COPS (COPS-PR may not be sufficient). New client-type(s) might need to be defined.
- The internal PEP (iPEP), for the same reason, should be functionally similar to a COPS-PR PEP. In this way, existing implementations and work on COPS-PR client-types could be reused.
- The internal PDP (iPDP) should be a generic module that can cooperate with any client-type of any iPEP. The iPDP should be able to store policies independently of their contents and the iPEP that those policies are directed to.

4.1. General Overview

In our architecture, the PEP comprises two components: An internal PEP (iPEP) and an internal PDP (iPDP). The iPEP is functionally similar to a COPS-PR PEP: it may contain one or more clients, each having a PIB that controls the device. The iPDP controls the contents of the PIB(s) of the iPEP. For simplicity reasons, we shall consider the case of an iPEP with a single client. As in COPS, the communication flow is:

$$\text{PDP} \leftrightarrow \text{PEP},$$

which is equivalent in our architecture to:

$$\text{PDP} \leftrightarrow \text{iPDP} \leftrightarrow \text{iPEP}.$$

The iPDP intervenes between the PDP and the iPEP. It controls the iPEP in a way similar to COPS-PR, i.e., by controlling the contents of its PIB. The PDP controls the behavior of the iPDP by controlling the contents of its Policy Base and by supplying it with values for the relevant policy parameters that the iPDP cannot evaluate. In this way, the PDP can indirectly control the contents of the PIB in the iPEPs, and consequently, the managed device. In more detail, when the PEP connects to the PDP, the latter sends to the former all the initial, relevant configuration data and parameters. This data is actually handled by the iPDP, which stores them in its Policy Base (PB) and process them. The Policy Base is a structure for storing policies (do not confuse it with a PIB, which is a standard, well-defined Policy structure, for specific types of policies. PBs are discussed later in this paper). At any time, the policies in the PB and the parameters maintained by the iPDP (either evaluated by

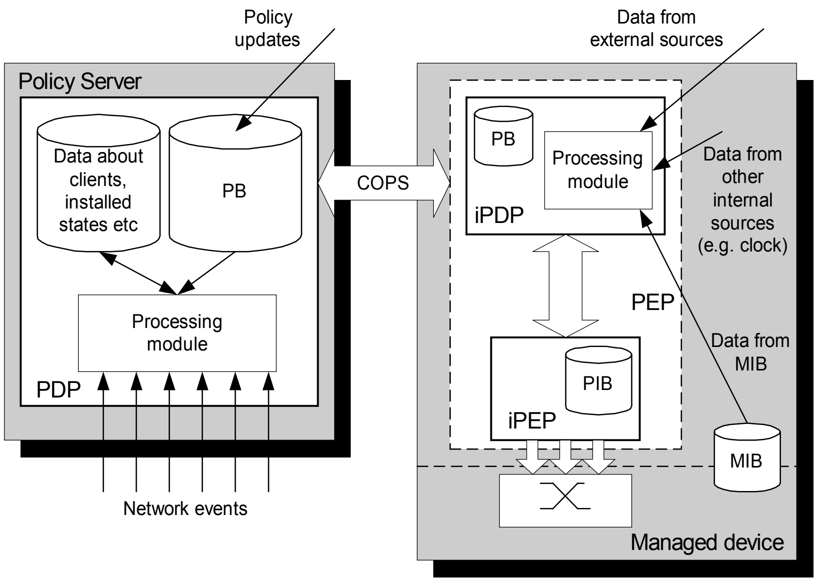


Figure 2. The architecture

itself or sent by the PDP) can be considered as a projection of the current global policies and network status on the specific PEP. Whenever a change in the global policies or the network status is detected by the PDP, it must decide whether their projection on the PEP is affected, and if it is, it should update the data of the iPDP (PB and/or parameters). The iPDP, always having a consistent view of the network policies and status, may send the appropriate commands to the iPEP and manipulate its PIB so as to achieve the desired behavior. Of course, the PDP must be aware of the capabilities of the PEPs that it controls, i.e., their iPDP and iPEP capabilities and type.

In this architecture, the PDP controls the PEP mainly by updating its policy parameters, rather that the policies themselves (policies are sent initially and seldom change). In this way, less data is exchanged through the network. However, the greatest advantage is that the PEP can be programmed to evaluate any parameter by itself. This is not efficient in all cases for scalability reasons (for example, it would be inefficient to program all PEPs to poll the entire network in order to evaluate the condition: "global congestion"). However, there are several cases that such an evaluation is efficient and desired. For example, in some cases, the evaluation of certain conditions may derive from the MIB of the managed device. By letting the iPDP evaluate those parameters, the PEP can take more decisions independently of the PDP. This is a very important property for fault-tolerant and self-configurable networks.

More details on the functions of each of the proposed entities follow, after a simple example.

4.2. A Simple Example

In order to demonstrate a simple example, we consider a router that connects four networks (Figure 3). The router has a very simple, imaginary PIB (a real one would unnecessarily complicate the example). This PIB has a table with five columns: The id of the policy, the router interface to which the policy applies, the IP address of the incoming packets, the action (1=allow, 0=deny) and the priority of the rule. In order to enforce the policies, the IP address of each incoming packet is compared with the IP addresses in the PIB. If the IP matches, then the device takes the decision that the policy specifies (allow/deny). The policies are examined according to their priority. Packets that do not match any authorized IP address are denied.

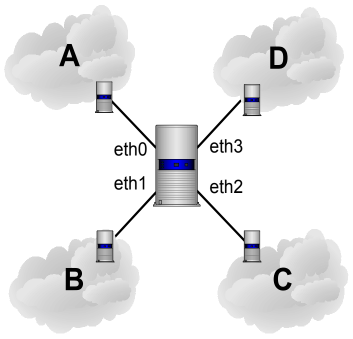


Figure 3. Topology of the network

The policies that govern the router's behavior are:

1. *During maintenance in a network, no access to it is allowed.*
2. *Subnets can exchange data only between office hours (8am to 5pm).*
3. *If the link to A is more that 80% utilized, traffic from A to B and C and vice versa is not allowed.*

(In case of conflicts between policies, the order determines the priority)

Suppose that the following events take place:

3.50: The router boots. Its interface to A is less than 80% utilized.
4.00: The interface to A becomes more than 80% utilized.
4.15: Maintenance starts at network B.
4.20: The interface to A becomes less than 80% utilized.
4.45: Maintenance ends at network B.
5:00: End of working hours.

Figure 4.(a) demonstrates the interaction that would take place in COPS-PR. The arrows indicate exchange of data. We observe that in all cases, the PDP updates the policies in the PIB of the device. Each time, the number of the removed/installed policies equals to the missing/added PIB rows (columns in the figure), respectively.

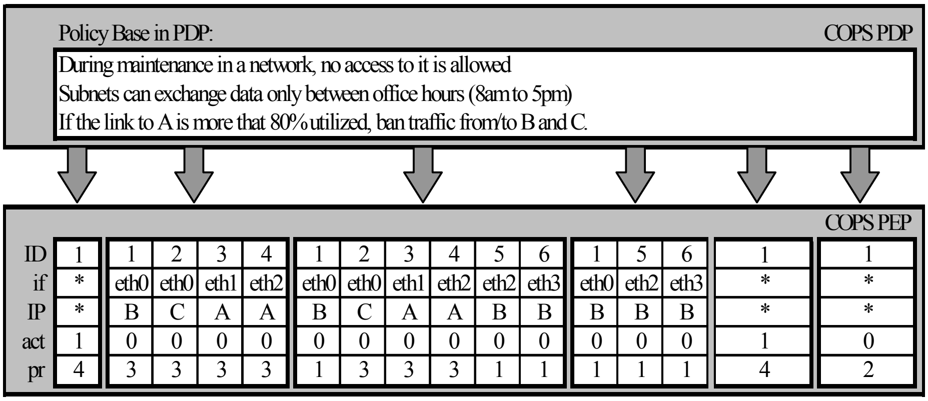
Let us now examine the same example in our architecture, demonstrated in Figure 4.(b). Suppose that the device is equipped with a clock. Also, suppose that the iPDP can be programmed to check the utilization of its interface eth0 through the MIB of the device. Hence, the PDP can assume that it does not need to monitor and evaluate those two parameters. The PDP translates the set of the original policies to the form shown in Figure 4.(b), and only checks if any network is under maintenance (the way that this happens is not relevant), The values that are necessary for the evaluation of the policies of the iPDP ($M, $S, $E1, $E2, $E3) are send from the PDP to the PEP, whenever necessary. The arrows again indicate data exchange. Notice that apart from the initialization cost of the PB of the iPDP, the amount of data exchanged between the PDP and the PEP was significantly reduced. Also, notice that in several cases, the iPDP is able to take policing decisions without interaction with the PDP.

4.3. The Entities

This section discusses in more detail the functionality of the proposed architectural entities. Implementation issues are discussed in section 5.

The Policy Decision Point (PDP): Although some new client-type(s) may need to be defined, the PDP is a COPS PDP. Hence, it has to provide the PEP with the appropriate configuration data that ensures that it operates as desired. This data is the set of the applicable policies and the value of parameters that the PEP cannot evaluate.

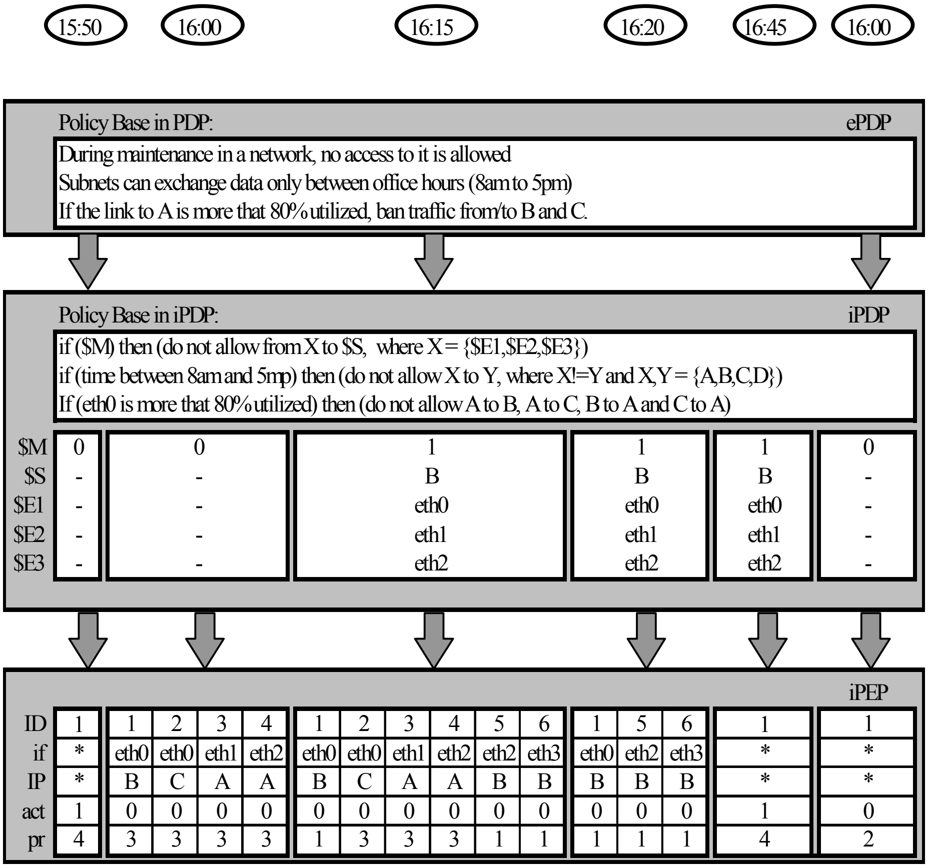
As defined in COPS, the PEP has to send information about its capabilities and the device that it serves as soon as it first connects. In this way, the PDP can send policies and parameters in a format that the PEP can understand. Moreover, the PDP is aware of which of the conditions and parameters cannot be evaluated by the PEP (neither by the iPDP nor by the iPEP). Hence, when the PEP connects, the PDP has to perform the following tasks: (i) Find all the relevant policies (ii) determine which parameters need to be monitored and evaluated on behalf of the PEP (iii) register these parameters for monitoring (iv) transform the policies in the appropriate form that the PEP can understand and (v) send the policies and the initial values of the parameters.



During maintenance in a network, no access to it is allowed
Subnets can exchange data only between office hours (8am to 5pm)
If the link to A is more that 80% utilized, ban traffic from/to B and C.

ID	1	1	2	3	4	1	2	3	4	5	6	1	5	6	1	1
if	*	eth0	eth0	eth1	eth2	eth0	eth0	eth1	eth2	eth2	eth3	eth0	eth2	eth3	*	*
IP	*	B	C	A	A	B	C	A	A	B	B	B	B	B	*	*
act	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
pr	4	3	3	3	3	1	3	3	3	1	1	1	1	1	4	2

(a) *PDP-PEP interaction in COPS-PR*



During maintenance in a network, no access to it is allowed
Subnets can exchange data only between office hours (8am to 5pm)
If the link to A is more that 80% utilized, ban traffic from/to B and C.

if ($M) then (do not allow from X to $S, where X= {$E1,$E2,$E3})
if (time between 8am and 5mp) then (do not allow X to Y, where X!=Y and X,Y = {A,B,C,D})
If (eth0 is more that 80% utilized) then (do not allow A to B, A to C, B to A and C to A)

$M	0	0	1	1	1	0
$S	-	-	B	B	B	-
$E1	-	-	eth0	eth0	eth0	-
$E2	-	-	eth1	eth1	eth1	-
$E3	-	-	eth2	eth2	eth2	-

ID	1	1	2	3	4	1	2	3	4	5	6	1	5	6	1	1
if	*	eth0	eth0	eth1	eth2	eth0	eth0	eth1	eth2	eth2	eth3	eth0	eth2	eth3	*	*
IP	*	B	C	A	A	B	C	A	A	B	B	B	B	B	*	*
act	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
pr	4	3	3	3	3	1	3	3	3	1	1	1	1	1	4	2

(b) *ePDP-iPDP and iPDP-iPEP interaction in the proposed architecture, for the same example*

Figure 4. A simple example

After the initialization, the PDP has to monitor the parameters registered in step (iii) of the initialization, and notify the PEP for changes in their values (As described later, in certain cases the PDP may decide not to monitor and/or update the value of a parameter, if it knows that this will not affect the decisions taken by the iPDP.) Finally, whenever there is a change to the (global) policy repository, the PDP may need to inform the PEP by deleting, updating or installing new policies into it.

The Policy Enforcement Point (PEP): The PEP is a COPS PEP. As described in the previous paragraph, the PEP reports its capabilities to the PDP when it connects to it, and enforces the decisions sent to it. It is comprised of two modules, the iPDP and the iPEP. The iPDP-iPEP relation in the PEP is similar to the client-hardware relation in a COPS PEP: The iPDP has some configuration data (policies and parameters) sent to it by the PDP, and this configuration data is used to control the iPEP.

The internal Policy Decision Point (iPDP): The internal Policy Decision point acts similarly to a COPS-PR PDP for the iPEP that it controls. Although this architecture describes a 1:1:1 relation between iPDPs, iPEP and clients (i.e., each iPEP has one client, and it connects to one iPDP that serves only this client), we believe that it can easily be extended to support a scheme where iPEPs with multiple clients connect to the same iPDP.

The iPDP communicates with the iPEP in a way similar to COPS-PR: the iPDP maintains information about the contents of the PIB of the iPEP, and it decides which policies should be installed or updated into it.

The iPDP maintains information that reflects the view of the network state and policies projected on the specific PEP, at the given time. The policies in this PB may be parametric, i.e., the conditions and actions may be associated with some parameters. For each parameter, information is registered on whether its value is obtained by the PDP or calculated locally. In the second case, the way that this value is computed should also be registered (e.g., MIB variable, clock, SNMP command, other process, etc.).

Finally, the iPDP should check the policies for conflicts prior to installing them to the iPEP, since there may be valid policies that conflict to each other under certain circumstances.

The internal Policy Enforcement Point (iPEP): The internal Policy Enforcement Point (iPEP) has similar functionality to a COPS-PR PEP. The main difference is that the iPEP is controlled by the iPDP, instead of by an (external) PDP, hence it does not need to implement the full COPS-PR protocol. However, the commands that it receives, and its reaction to them are similar to COPS-PR.

5. Implementation Issues

In the previous section we presented the design of our architecture, mainly in the form of requirements in the functionality of the entities. Although we are still in the process of designing the architecture, several implementation issues have been taken into consideration. These are discussed in this part.

5.1. iPEP Implementation

The iPEP is similar to a COPS-PR PEP. The basic difference is that they do not need to implement the communication details of the COPS-PR protocol (TCP connection, encryption, integrity, time-outs, redirection to other PDPs, etc.). However, the cores of an iPEP and a COPS-PR PEP (client) are exactly the same. They implement the same PIB and they are controlled with similar commands; also they produce similar errors and warnings. Hence, similar client-types are expected to exist.

5.2. Parameter Optimization

The following are some ideas that may reduce the number of exchanged messages and resource consumption, both in the PDPs and the PEPs.

1. Obviously, when more than two policies in the PB of the iPDP depend on the same condition or parameter, the PDP only needs to send the value of this parameter once, and this value is shared by all the policies that depend on it.

2. Suppose that there is a policy in the PDP

$$\textit{If } [(c_1) \textit{ and } (c_2) \textit{ and } (c_3) \textit{ and } (c_4) \textit{ and } (c_5)] \textit{ then } \{(a_1) \textit{ and } (a_2) \textit{ and } \ldots\}$$

and the PDP knows that the PEP can evaluate c_3, c_4 and c_5. Also suppose that no other policy depends on c_1 and c_2. In this case, the PEP can replace these two conditions with $c = (c_1 \text{ and } c_2)$ and send one parameter instead of two.

3. Suppose that the PDP has a policy

$$p_1\textit{: If } [(c_1) \textit{ and } (c_2)] \textit{ then } \ldots$$

Where c_1 and c_2 are conditions evaluated by the PDP. Also, suppose that c_2 is not used by any other policy of the iPDP. While c_1 has value “false”, the PDP may decide not to send (or even monitor) the value of c_2, because this would not affect the decision of the iPDP. However, the PDP should send the values of c_2 while c1 is “true”. In this way, unnecessary exchange of data in the network is prevented.

4. Suppose that the PDP has a policy

If $[(c_1)$ *and* $(c_2)]$ *then ...*

Where c_1 can be evaluated by the iPEP but c_2 cannot. In most policies, the conditions that cannot be evaluated locally, represent global, slow-changing conditions, while the ones that are evaluated locally, usually change frequently. However, there may be cases in which this rule does not apply. If c_2 changes frequently but c_1 does not, the PDP must frequently inform the PEP of the value of c_2, even when c_1 is false (the PDP is not aware of the value of c_1). For such policies, the PDP may decide to monitor c_1 as well. In this way, when c_1 is false, the PDP will not send values for c_2, and unnecessary communication between the PDP and the PEP will be avoided.

5.3. Representation of Policies in the iPDP

Suppose that a policy in the PDP

If $[(c_1)\ o_1\ (c_2)\ o_2\ (c_3)\ ...]$ *then* $\{(a_1')$ *and* (a_2') *and ...*$\}$

is equivalent to

If $[(c_1)\ o_1\ (c_2)\ o_2\ (c_3)\ ...]$ *then* p_1

where p_1 is a policy that can be understood by the iPEP (i.e., they can be stored in its PIB). PIBs are structures similar to MIBs. Each PIB is a tree of Policy Rule Classes (PRC's) [11], [12]. The leaves of the tree are instances of those classes, and they are called Policy Rule Instances (PRIs). PRCs (and PRIs) can be described by (standardized) identifiers, similar to MIB identifiers. Policies in the PIB are represented as a set of PRIs. Hence, a policy p_1 could be described as a set of PRIs (identifiers and values) that have to be installed or removed.

The PDP has to maintain three tables: The first table stores the policy id and the policy itself, in the form of a table of PRIs that have to be installed or removed. In this way, p_1 can be stored in this table. The PRIs may be parametric. The second table stores the values of all parameters of all policies in the PIB (either these send by the PDP or evaluated by the iPDP). The third table has a column that stores a list of the conditions that the iPDP should evaluate, and a column with a list of the policy ids that should be installed if the conditions evaluate “true”. Of course, the iPDP has to perform other actions as well, such as maintaining information for the status of its iPEPs, like the PDPs keep similar information for their PEPs.

For the example presented in section 4, the policies would be stored as shown in Figure 5.:

ID	policy
1	ID=N/A If=$E1 IP=$S act=0 PRI=1
2	ID=N/A If=$E2 IP=$S act=0 PRI=1
3	ID=N/A If=$E3 IP=$S act=0 PRI=1
4	ID=N/A If=* IP=* act=0 PRI=2
5	ID=N/A If=eth0 IP=B act=0 PRI=3
6	ID=N/A If=eth0 IP=C act=0 PRI=3
7	ID=N/A If=eth1 IP=A act=0 PRI=3
8	ID=N/A If=eth2 IP=A act=0 PRI=3

Par.	Value
$C	16:23:58.03
$U	83%
$M	TRUE
$S	B
$E1	eth0
$E2	eth2
$E3	eth3

if	then
($M=true)	1,2,3
(8:00<$C<17:00)	4
($U>80%)	5,6,7,8

Figure 5. An instance of the tables of the iPDP

In this way, the policies are represented in a unified way, independent of the PIB of the served iPEP. This allows standardizing the way that policies are sent by the PDP. Also, due to this property, the iPDP can be an entity independent of the vendor and the iPEP client that it serves. However, before sending the policies to the iPEP, a client-dependent (but not vendor dependent) module should perform conflict checking, since the policies in the PB may be conflicting under certain circumstances.

5.4. Communication between PDP and PEP

Since the policies and the parameters are rows of two well-defined tables in the iPDP, the communication between the PDP and the PEP could be COPS. Of course, a new client-type would need to be defined, which would standardize the exchange of those objects. Notice, also, that the exchange of information is performed in a provisioning mode, similar to COPS-PR. We are examining whether an extension of COPS-PR would be suitable for such exchange of data.

In COPS, when a client connects to a PDP, it sends information about the capabilities of its clients and the managed device. This information is used by the PDP in order to map the network state and policies to the specific device. In our architecture, the PEP must send the capabilities of both its iPDP and iPEPs. Hence, when it connects to the PDP, it has to declare that it contains an "internal PDP" and report its capabilities, and then, send to the PDP the capabilities of its (client-types of its) iPEPs, in a COPS-PR style.

6. Conclusions and Future Work

This paper presented an architecture that attempts to extend COPS-PR PEPs in order to be able to take more policy decisions locally. For this purpose, those PEPs are slightly modified and wrapped into an extended PEP, which contains those PEPs (as iPEPs), as well as an internal PDP (iPDP). The iPDP is used in order to push some of the intelligence of the external PDP down to the device.

We claim that this architecture has several advantages, compared to COPS-PR. The most obvious one is that the network utilization is reduced. However, the most important property is that more decisions can be taken by the managed device itself. This property is an important feature that contributes significantly towards fault-tolerant and self-configurable networks. Furthermore, since the device is the converging point of the decisions of different PDPs, by moving the intelligence into the managed device, conflicts can be detected and resolved more efficiently.

As a future work, our first objective is to implement the proposed architecture and evaluate its efficiency in comparison to COPS-PR. Some longer-term objectives have also been set. One of them is to examine the case of multiple clients in each iPEP, which may need to connect (through the iPDP) to different PDPs. Besides, we want to examine how the iPDP can be optimized, especially in terms of storage of policies. Obviously the table representation is not the optimal solution since a significant amount of information may be redundant. Finally, an interesting issue is to examine how mobile agents can be used for the evaluation of (non-local) parameters by the iPDP.

References

1. Shepard, S.J.; "Policy-based networks: hype and hope"; IT Professional, Volume: 2 1, Jan.-Feb. 2000, Page(s): 12 –16
2. "Introduction to Policy-based Networking and Quality of Service"; IPHIGHWAY, White paper, January 2000
3. Hugh Mahon; Yoram Bernet; Shai Herzog; "Requirements for a Policy Managed System"; IETF; Internet draft draft-ietf-policy-req-01.txt, October 1999
4. M. Sloman, "Policy Driven Management For Distributed Systems", Plenum Press Journal of Network and Systems Management, vol 2, no. 4, Dec. 1994, pp. 333-360
5. A. Westerinen; J. Schnizlein; J. Strassner; Mark Scherling; Bob Quinn; Jay Perry; Shai Herzog; An-Ni Huynh; Mark Carlson; "Policy Terminology"; IETF, Internet Draft draft-ietf-policy-terminology-00.txt, July 2000
6. "Policy Standards and IETF Terminology"; IPHighway, White paper, January 2000.
7. R. Yavatkar; D. Pendarakis; R. Guerin; "A Framework for Policy-based Admission Control", IETF, RFC 2753, January 2000
8. D. Durham, Ed.; J. Boyle; R. Cohen; S. Herzog; R. Rajan; A. Sastry; "The COPS (Common Open Policy Service) Protocol"; IETF, RFC 2748, January 2000
9. "Policy-based Networking Products, Design and Architecture"; IPHighway, White paper, January 2000.
10. "Policy-Powered Networking and the Role of Directories"; 3COM, White paper, July 1998.

11. Kwok Ho Chan; David Durham; Silvano Gai; Shai Herzog; Keith McCloghrie; Francis Reichmeyer; John Seligson; Andrew Smith; Raj Yavatkar; "COPS Usage for Policy Provisioning"; IETF, Internet Draft draft-ietf-rap-pr-03.txt, July 2000
12. M. Fine; K. McCloghrie; J. Seligson; K. Chan; S. Hahn; R. Sahita; A. Smith; Francis Reichmeyer; "Framework Policy Information Base", IETF, Internet Draft draft-ietf-rap-frameworkpib-01.txt, July 2000

URLs:

- IETF home page:
  http://www.ietf.org
- Resource Allocation Protocol (rap) Charter:
  http://www.ietf.org/html.charters/rap-charter.html
- Policy Framework (policy) Charter:
  http://www.ietf.org/html.charters/policy-charter.html
- IETF Rap Working group mailing list:
  http://majordomo.iphighway.com/