DESIGN AND IMPLEMENTATION OF A POLICY-BASED RESOURCE MANAGEMENT ARCHITECTURE

Paris Flegkas, Panos Trimintzios, George Pavlou, Antonio Liotta
Centre for Communication Systems Research, School of Electronics, Computing and Mathematics, University of Surrey, Guildford, Surrey GU2 7XH, UK, {P.Flegkas, P.Trimintzios, G.Pavlou, A.Liotta}@eim.surrey.ac.uk}

Abstract: Policy-based Management can guide the behavior of a network or distributed system through high-level declarative directives that are dynamically introduced, checked for.consistency, refined and evaluated, resulting typically in a series of low-level actions. We actually view policies as a means of extending the functionality of management systems dynamically, in conjunction with pre-existing "hard-wired" management logic. In this paper, we first discuss the policy management aspects of a resource management architecture for IP Differentiated Services networks and we focus on the functionality of the network dimensioning component. We then present a detailed description of the design and implementation of the components of the policy management sub-system needed to be deployed in order to make our system policy-driven. Finally, we present examples of network dimensioning policies describing their transformation from their definition by the operator until their enforcement.

Key words: Policy-based Management, IP Differentiated Services, Network Dimensioning, Resource Management

1. INTRODUCTION

For years the Internet networking community has been struggling to develop ways to manage networks. Initial attempts brought mechanisms and protocols that focused on managing and configuring individual networking devices i.e. the Simple Network Management Protocol (SNMP). This model worked well in early deployments of IP management systems for local and metropolitan area networks but now, with the evolution of Quality of Service (QoS) models such as the Differentiated Services (DiffServ) framework, the complexity and overhead of operating and administrating networks increases enormously. As such, it is very difficult to build management systems that can cope with the growing network size, complexity and multi-service operation requirements. There is also a need to be able to program management systems and network components to adapt to emerging requirements and subsequently to be able to dynamically change the behavior of the whole system to support modified or additional functionality. The emerging Policy-based Network Management paradigm claims to be a solution to these requirements.

Policy-based Management has been the subject of extensive research over the last decade [1]. Policies are seen as a way to guide the behavior of a network or distributed system through high-level, declarative directives. The IETF has been investigating policies as a means for managing IP-based multi-service networks, focusing more on the specification of protocols (e.g. COPS) and the object-oriented information models for representing policies. Inconsistencies in policy-based systems are quite likely since management logic is dynamically being added, changed or removed without the rigid analysis, design, implementation, testing and deployment cycle of "hard-wired" long-term logic. Conflict detection and resolution is required in order to avoid or recover from such inconsistencies.

In the next section, we discuss the policy management aspects of a resource management system for IP Differentiated Services networks; we then focus on the functionality of the dimensioning component in section 3 and in section 4, we present a detailed description of the design and implementation of the components of the policy management sub-system needed to be deployed in order to make our system policy-driven. Finally, we present examples of network dimensioning policies, describing their transformation from their definition by the operator until their enforcement.

2. SYSTEM ARCHITECTURE

We have designed a system for supporting QoS in IP DiffServ Networks in the context of the European collaborative research project TEQUILA (Traffic Engineering for QUality of service in the Internet at LArge scale). This architecture can be seen as a detailed decomposition of the concept of an extended Bandwidth Broker (BB) realized as a hierarchical, logically and physically distributed system. A detailed description can be found in [2]. A classification of the policies applied to this system was presented in [3]. In Figure 1 we present only the resource management part of the architecture together with the components of the policy management sub-system i.e. Policy Management Tool, Policy Repository and Policy Consumer needed to make the system extensible through policies.

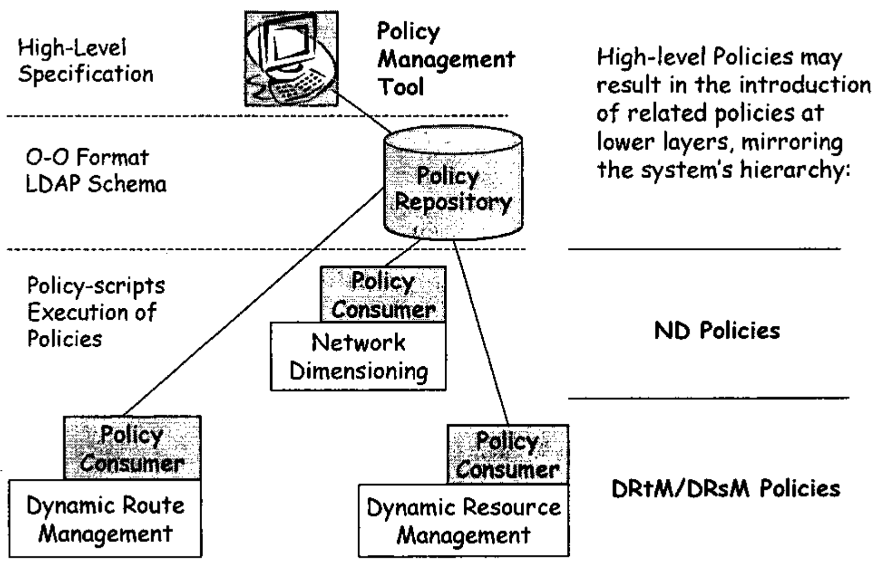


Figure 1. Policy-driven Resource Management System

The Network Dimensioning (ND) component is responsible for mapping traffic requirements to the physical network resources and for providing Network Dimensioning directives in order to accommodate the predicted traffic demands. We describe the functionality and algorithms of Network Dimensioning in more detail in Section 3. The lower level of the system intends to manage the resources allocated by Network Dimensioning during the system operation in real-time, in order to react to statistical traffic fluctuations and special arising conditions. This part is realized by the Dynamic Route (DRtM) and Dynamic Resource Management (DRsM), which both monitor the network resources and act to medium to short term fluctuations. DRtM operates at the edge nodes and is responsible for managing the routing processes in the network. It mainly influences the parameters based on which the selection of one of the established MPLS Labeled Switched Paths (LSPs) is effected at an edge node with the purpose of load balancing. An instance of DRsM operates at each router and aims to ensure that link capacity is appropriately distributed among the PHBs in that link.

A single Policy Management Tool exists for providing a policy creation environment to the administrator where policies are defined in a high-level declarative language and after validation and static conflict detection tests, they are translated into object-oriented (O-O) representation (information objects) and stored in a repository. The Policy Repository is a logically centralized component but may be physically distributed since the technology for implementing this component is the LDAP (Lightweight Directory Access Protocol) Directory. After the policies are stored, activation information may be passed to the responsible Policy Consumer in order to retrieve and enforce them. The Policy Consumer can be seen as a collocated Policy Decision Point (PDP) and Policy Enforcement Point (PEP) with regards to the IETF Policy Framework [4]. A detailed description of the design and implementation of these components is presented in Section 4.

In Figure 1 the representation of the policies at every level of the framework is also depicted showing that every policy is going through two stages of translation/refinement in its life-cycle in order to be enforced: first from the high-level specification to an object-oriented format (LDAP objects) and second from the LDAP objects to a script that is interpreted on the fly, complementing this way conceptually the management intelligence of the above layer in the hierarchy. For example, a policy enforced on the DRsM component is actually enhanced management logic that conceptually belongs to the ND layer of our system. Although policies may be introduced at every layer of our system, higher-level policies may possibly result in the introduction of related policies at lower levels, forming a policy hierarchy mirroring the management system's hierarchy. This means that a policy applied to a hierarchical system might pass through another stage of translation/refinement that will generate the policies that are enforced in the lower levels of the system. It is questionable if the automation of this process is feasible without human intervention. A more detailed discussion on policy-based hierarchical management systems can be found in [5].

In the next section, we present the algorithm by which the Network Dimensioning component calculates the configuration of the network in order for the reader to understand the examples of policies enforced on this component presented in the following sections.

3. NETWORK DIMENSIONING ALGORITHM

ND performs the provisioning activities of the system. It is responsible for the long to medium term configuration of the network resources. By configuration we mean the definition of LSPs as well as the anticipated loading for each PHB on all interfaces, which are subsequently being translated by DRsM into the appropriate scheduling parameters (e.g. priority, weight, rate limits) of the underlying PHB implementation. ND does not provide absolute values but they are in the form of ranges, constituting directives for the function of the PHBs, while for LSPs they are in the form of multiple paths to enable multi-path load balancing. The exact PHB configuration values and the load distribution on the multiple paths are determined by DRsM and DRtM respectively, based on the state of the network, but should always adhere to ND directives.

ND runs periodically, first requesting the predictions for the expected traffic per Ordered Aggregate [6] (OA) in order to be able to compute the provisioning directives. The dimensioning period is in the time scale of a week while the forecasting period is in the time scale of hours. The latter is a period in which we have considerably different predictions as a result of the time schedule of the subscribed Service Level Specifications (SLSs). For example, ND might run every Sunday evening and provide multiple configurations i.e. one for each period of each day of the week (morning, evening, night).

The objectives are both traffic and resource-oriented. The former relate to the obligation towards customers, through the SLSs. These obligations induce a number of restrictions about the treatment of traffic. The resource-oriented objectives are related to the network operation, more specifically they are results of the high-level business policy that dictates the network should be used in an optimal manner. The basic Network Dimensioning functionality is summarized in Table 1.

Table 1. Network Dimensioning Algorithm Overview

Input:
Network topology, link properties (capacity, propagation delay, PHBs)
Pre-processing:
Request traffic forecast, i.e. the potential traffic trunks (TT)
Obtain statistics for the performance of each PHB at each link
Determine the maximum allowable hop count K per TT according to the PHB statistics
Optimisation phase:
Start with an initial allocation (e.g. using the shortest path for each TT)
Iteratively improve the solution such that for each TT find a set of paths:
The minimum bandwidth requirements of the TT are met
The hop-count constraints K is met (delay/ loss requirements are met)
The overall cost function is minimized
Post-processing:
Allocate any extra capacity to the resulted paths of each OA according to resource allocation policies
Sum the path requirements per link per OA, give minimum (optimisation phase) and maximum (post-processing phase) allocation directives to DRsM
Give the appropriate paths calculated in the optimisation phase to DRtM
Store the configuration into the Network Repository

The network is modeled as a directed graph $G = (V,E)$, where V is a set of nodes and E a set of links. With each link $l \in E$ we associate the following parameters: the link physical capacity C_l, the link propagation delay d_l^{prop}, the set of the physical queues K, i.e. Ordered Aggregates (OAs), supported by the link. For each OA, $k \in K$ we associate a bound d_l^k (deterministic or probabilistic depending on the OA) on the maximum delay incurred by traffic entering link l and belonging to the $k \in K$, and a loss probability p_l^k of the same traffic.

The basic traffic model of ND is the traffic trunk (TT). A traffic trunk is an aggregation of a set of traffic flows characterized by similar edge-to-edge performance requirements [7]. Also, each traffic trunk is associated with one ingress node and one egress node, and is unidirectional. The set of all traffic trunks is denoted by T.

The *primary objective* of such an allocation is to ensure that the requirements of each traffic trunk are met as long as the traffic carried by each trunk is at its specified minimum bandwidth. However, with the possible exception of heavily loaded conditions, there will generally be multiple feasible solutions. The design objectives are further refined to incorporate other requirements such as: a) avoid overloading parts of the network while other parts are under loaded, b) provide overall low network load (cost).

The last two requirements do not lead to the same optimization objective. In any case, in order to make the last two requirements more concrete, the notion of "load" has to be quantified. In general, the load (or cost) on a given link is an increasing function of the amount of traffic the link carries. This function may refer to link utilization or may express an average delay, or loss probability on the link. Let x_l^k denote the capacity demand for OA $k \in K$ satisfied by link l and $u_l^k = x_l^k / C_l$ the link utilisation. Then the link cost induced by the load on OA $k \in K$ is a convex function, $f_l^k(u_l^k)$, increasing in u_l^k. The total cost per link is defined as $F_l(\overline{u}_l) = \sum_{k \in K} f_l^k(u_l^k)$, where $\overline{u}_l = \{u_l^k\}_{k \in K}$ is the vector of demands for all OAs of link l. The total cost per link is an approximate function, e.g. $f_l^k(u_l^k) = a_l^k u_l^k$.

We provide an objective that compromises between the two a) and b), that is avoid overloading parts of the network and minimize overall network cost:

$$\text{minimize } \sum_{l \in E} (F_l(\overline{u}_l))^n = \sum_{l \in E} \left(\sum_{k \in K} f_l^k(u_l^k) \right)^n, \quad n \geq 1 \quad (1)$$

When $n = 1$, the objective (1) reduces to objective b), while when $n \to \infty$ it reduces to a).

Each traffic trunk is associated with an end-to-end delay and loss probability constraint of the traffic belonging to the trunk. Hence, the trunk routes must be designed so that these two constraints are satisfied. Both the constraints above are constraints on additive path costs under specific link costs. However the problem of finding routes satisfying these constraints is, in general, NP-complete [8]. Given that this is only part of the problem we need to address, the problem in its generality is rather complex.

Usually, loss probabilities and delay for the same PHB on different nodes are of similar order. We simplify the optimization problem by transforming the delay and loss requirements into constraints for the maximum hop count for each traffic trunk (TT). This transformation is possible by keeping statistics for the delay and loss rate of the PHBs per link, and by using the maximum, average or n-th quantile in order to derive the maximum hop count constraint.

For each traffic trunk $t \in T$ we denote as R_t the set of (explicit) routes defined to serve this trunk. For each $r_t \in R_t$ we denote as b_{r_t} the capacity we have assigned to this explicit route. We seek to minimize (1), such that the hop-count constraints are met and the bandwidth of the explicit routes per traffic trunk should be equal to the trunks' capacity requirements.

This is a network flow problem and considering the non-linear formulation, for the solution we use the general gradient projection method [9]. This is an iterative method, where we start from an initial feasible solution, and at each step we find the minimum first derivative of the cost function path and we shift part of the flow from the other paths to the new path, so that we improve our objective function (1). If the path flow becomes negative, the path flow simply becomes zero. This method is based on the classic unconstraint non-linear optimization theory, and the general point is that we try to decrease the cost function through incremental changes in the path flows. A more detailed description of the algorithm is presented in [10].

4. DESIGN AND IMPLEMENTATION OF THE POLICY COMPONENTS

In the following sections, we present and describe the design and implementation of the policy sub-system components. First in the Policy Management Tool section, the description of policy definition language as well as the capabilities of the graphical interface are presented; we then focus on the Policy Repository where the most important issue is how one models policies in an O-O format and then on the Policy Consumer where policies are translated to scripts and executed on the fly. Examples of policy rules are also presented demonstrating the different way of representation of the rules at every stage of their life cycle i.e. from high-level directives to LDAP objects and finally to interpreted scripts realising the new management logic added through these policies.

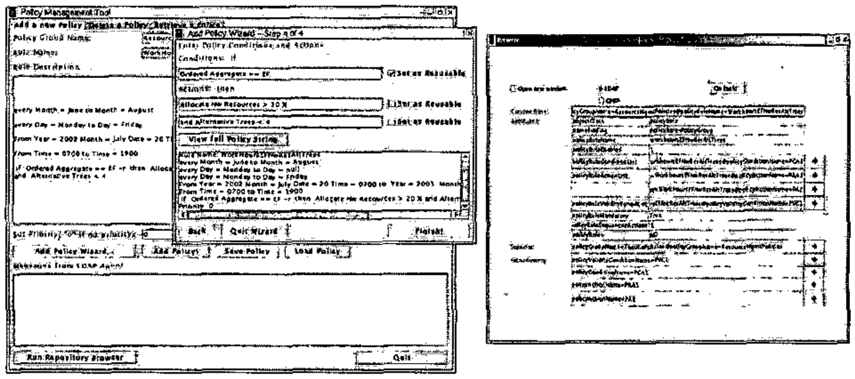
4.1 Policy Management Tool

A high-level definition language has been designed and implemented that provides to the administrator the ability to add, retrieve and update policies in the Policy Repository. The administrator enters a high-level specification of the policy, which is then passed to a translation function that maps this format to entries in an LDAP Directory realizing the Policy Repository (see next section) through LDAP *add* operations, according to an LDAP schema of our information model; the latter has been produced following the guidelines described in [11]. The format of a policy rule specification is shown below:

[Policy ID] [Group ID] [time period condition] **[if** {condition [and] [or]}] **then** {action [and]}

The first two fields define the name of the policy rule and the group that this policy belongs to so that the generated LDAP entries should be placed under the correct policy group entry. The time period condition field specifies the period that the policy rule is valid and supports a range of calendar dates, masks of days, months as well as range of times. The following {if then} clause represents the actual policy rule where the condition and action fields are based on the information model described earlier in this section. Compound Policy Conditions are also supported both in the Disjunctive Normal Form (DNF) (an ORed set of ANDed conditions) and in the Conjunctive Normal Form (CNF) (an ANDed set of ORed conditions) as well as Compound Policy Actions representing a sequence of actions to be applied. Our implementation also caters for the notion of rule-specific and reusable conditions and actions in a way that every time a new policy rule is added, it first checks if its conditions and actions are already stored in the repository as reusable entries. If such entries exist, an entry is added with a DN pointer to the reusable entry under the policy rule object while if not they are treated as rule-specific, placing the condition entry below the policy rule entry.

A compiler has been implemented in order to parse and translate the policy rules specified with the above syntax using SableCC [12]. This is an object-oriented framework that generates compilers by building a strictly-typed abstract syntax tree that matches the grammar of the language, automatically generates tree-walker classes and enables the implementation of the actions on the nodes of the tree using inheritance. Translation classes have been implemented that map the entered policy rules to LDAP *add* operations, according to an LDAP schema of our information model (see next section). Notifications to the corresponding Policy Consumers are also sent every time a policy rule is successfully added to the repository. The role attribute of every policy rule is used to distinguish which policy consumer to notify since this attribute represents the properties of the PEPs that each PDP manages according to Policy Core Information Model (PCIM)[13].



every Month = June to Month = August
every Day = Monday to Day = Friday
From Year = 2002 Month = July Date = 20 T
From Time = 0700 to Time = 1900
If Ordered Aggregate == EF -> then Alloca
and Alternative Trees < 4

Ordered Aggregate == EF
Allocate HW Resources > 20 %
and Alternative Trees < 4

Rule Name: WorkHours
every Month = June to Month = August
every Day = Monday to Day = null
every Day = Monday to Day = Friday
From Year = 2002 Month = July Date = 20 Time = 0700 to Year = 2003 Month
From Time = 0700 to Time = 1900
If Ordered Aggregate == EF -> then Allocate HW Resources > 20 % and Alter
Priority 0

*Figure 2.*a) Policy Management Tool Screenshot b) LDAP Browser Instance

A Graphical User Interface (GUI) has been implemented in order for the administrator to manage policies that are entered in the system (Fig. 2). It provides capabilities to add new policies, retrieve or delete existing policies. The addition of the policies can either be done directly by writing the whole policy rule in the format described previously or follow a wizard that guides the administrator through the process of creating and entering a new policy to the system step by step. A directory browser has also been implemented and integrated with the tool that enables the operator to browse the information stored in a tree-structured repository like the LDAP Directory. In the browser's design, the idea of a current entry was adopted for displaying the contents of a tree. Each browser instance has one current entry, which is a selected tree entry. Its distinguished name (DN), its attributes' types and values, its superior entry's distinguished name and its subordinate entries' relative distinguished names (RDNs) are displayed. The current entry's position in the tree can be identified by the current entry's distinguished name. The user may select any of the current entry's subordinate entries or its superior entry and make it the browser's current entry. This way, one can move up and down the hierarchy of the Directory Information Tree (DIT) that is accessed. If an attribute's entry is a pointer to another entry, the user can make it the current entry, in which case they still keep the option to go back. In addition, the user is allowed to fill in the DN of the entry to be displayed in the Current Entry field.

In order to demonstrate the results of the enforcement of policies we used a 10-node (nodes 0-9) 36-link random topology and a traffic load of 70 % of the total throughput of the network. Our first example (P1) concerns a policy rule that wants to create an explicit LSP following the nodes 4, 9, 7, 6 with the bandwidth of the TT being 2 Mbps that is associated with this LSP. The policy rule is entered with the following syntax:

```
    If OA==EF and Ingress==4 and Egress==6 then Setup LSP 4-9-7-6 2
Mbps (P1)
```

The second example (P2) of a policy rule concerns the effect of the cost function exponent in the capacity allocation of the network. As we mentioned earlier by increasing the cost function exponent, the optimisation objective that avoids overloading parts of the network is favoured. So, if the administrator would like to keep the load of every link below a certain point then he/she should enter the following policy rule in our system using again our policy notation:

```
    If maxLinkLoad > 80% then Increase Exponent by 1         (P2)
```

4.2 Policy Repository

An object-oriented information model has been designed for representing the network dimensioning policies, based on the IETF Policy Core Information Model (PCIM) and its extensions (PCIMe) specified in [13] and [14] respectively. One of the major objectives of such information models is to bridge the gap between the human policy administrator who enters the policies and the actual enforcement commands executed at the component in order to realize the business goal of the administrator. Another goal is to facilitate interoperability among different systems so that policy consumers that belong to different systems understand the same semantics of policy and they have a mutual knowledge of how policies are stored in the policy repository despite the fact that each policy consumer might interpret it differently. IETF has described a QoS Policy Information Model [15], representing QoS policies that result in configuring network elements to enforce the policies, while our information model describes policies that are applied at a higher level (Network Management Level). Some of these policies may possibly be refined into lower-level policies mirroring our architecture's hierarchy and finally result into policies configuring the Network Elements.

In our information model that represents Network Dimensioning policies that can be enforced in our system, most of the conditions are modeled by using the class SimplePolicyCondition with instances of the variable and value classes (IF <variable> matches <value>). Some of the actions are modeled by defining classes derived from the PolicyAction abstract class while others are modeled by using the class SimplePolicyAction with the appropriate aggregations, using instances of the variable and value classes ("SET <variable> TO <value>"). For example, the maximum number of alternative trees that the ND algorithm should calculate for every TT is represented by a pair of maxAltTree variable and IntegerValue classes as well as the definition of the constant used in the link cost function. In [3] the policyAction class hierarchy is described in more detail where the DimensioningPeriodAction class models the policy action that sets the period that ND is calculating a new configuration, while the NwBwAllocationAction and LinkBwAllocationAction classes represent the actions that indicate the amount of bandwidth to be allocated to every OA (depending on the policy condition) at a network wide level and in every link respectively. The SpareBwTreatmentAction and OverBwTreatment Action classes represent the policy actions that drive the post-processing stage of ND as explained in the previous section. The SetLSPAction class models the setup of an LSP that is defined by policy and the HopCountDerivationAction class represents the action that influences the way the derivation of the delay and loss requirements to an upper bound of number of hops is done.

In Figure 3 the policy rule P1 presented as an example in the previous section is depicted modeled according to PCIM and PCIMe. As it can be seen, it comprises a compound policy condition which represents a combination of 3 simple policy conditions in Disjunctive Normal Form (ORs of ANDs) each of them belonging to the same Group (GroupNumber =1) and a Policy Action represented by the SetLSPAction class derived form the PolicyAction abstract class defined in PCIM. The first simple policy condition uses an OA variable which takes integer values from 1 to 4 (EF is 1, AF1x is 2, etc), the second and third simple policy conditions use an Ingress node and Egress node variables which take integer values form 0 to 9 for our network topology (the egress node simple policy condition is not shown in Figure 3 for illustrative purposes). The aggregations used in order to define in order to define this rule are also depicted as defined in PCIMe.

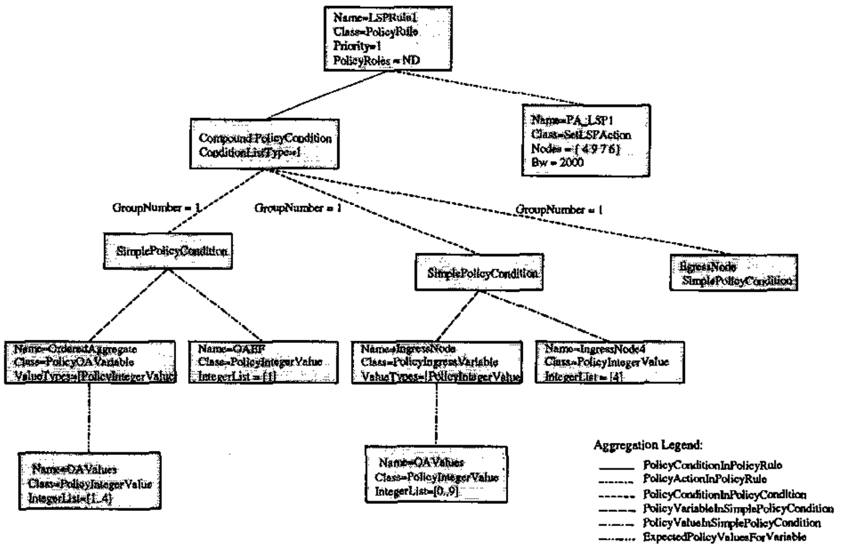


Figure 3. Policy Rule P1 according toPCIM/PCIMe

Using the same methodology, policy rule P2 is modelled according to PCIM/PCIMe using a Simple Policy Condition and ExponentAction class derived from the Policy Action abstract class. These classes are mapped to structural and auxiliary classes as defined in [11] in order to be stored to an LDAP Directory, which realizes our Policy Repository.

Since our system described in Section 2 is a large scale distributed system, it is valid to consider CORBA as the technology to support the remote interactions between the components. This was the key motivation for mapping the LDAP functionality to CORBA realizing the Policy Repository as an LDAP Directory offering a CORBA IDL interface, identical to the LDAP specifications [16], to the rest of the components. The following Code 1 caption shows the part of the specification of an LDAP server in Interface Definition Language (IDL) providing a LDAP search operation to every LDAP client in our system. Note that the CORBA implementation of the LDAP search operation returns the result in a single message while the LDAP protocol returns the multiple matching entries in a series of messages, one for each entry. The results are terminated with a result message, which contains an overall result for the search operation.

```
//...
typedef string LDAPDN_t;
enum Scope_t {
    sc_baseObject,
    sc_singleLevel,
    sc_wholeSubtree
  };
typedef string Filter_t; // filter for this implementation
struct SearchResultEntry_t_struct {
    LDAPDN_t          objectName;
    AttributeList_t   attributes;
  };
typedef SearchResultEntry_t_struct SearchResultEntry_t;
typedef sequence<SearchResultEntry_t> SearchResultEntryList_t;
interface LDAPServer {
  void Search (
    in LDAPDN_t                   baseObject,
    in Scope_t                    scope,
    in Filter_t                   filter,
    in AttributeDescriptionList_t attributeTypes,
    out SearchResultEntryList_t   searchResultList
  ) raises (noSuchObject, invalidDNSyntax, invalidFilterSyntax, generalError);
//...
}; // interface LDAPServer
```

Code 1. LDAP Search operation in IDL

4.3 Policy Consumer

Policy Consumers may be considered as the most critical components of the policy management framework since they are responsible for enforcing the policies on the fly while the system is running. The key aspect of policies apart from their high-level declarative nature is that they can also be seen as a vehicle for "late binding" functionality to management systems, allowing for their graceful evolution as requirements change. So, a policy capable system should provide the flexibility to add, change or remove management intelligence while according to traditional management models, management logic is of static nature, parameterized only through Managed Objects (MOs) attributes and actions. In order to achieve such functionality, a policy is eventually translated to a script "evaluated" by an interpreter with actions resulting in management operations. In [3] a detailed design and decomposition of the policy consumer was presented where every policy consumer comprises a Repository Client which retrieves all the LDAP objects associated with a policy rule, a script generator which is responsible for creating the script that implements the policy, and a policy interpreter which provides the "glue" between the policy consumer and the policy-based component and interprets the script, which includes functions that perform management operations. In the following paragraphs, we present how the policy rule examples P1 and P2 are translated and enforced by the Policy Consumer.

After the P1 rule is correctly translated and stored in the repository, the Policy Management Tool notifies the Policy Consumer associated with ND that a new policy rule is added in the repository, which then goes and retrieves all the associated objects with this policy rule. From the policy objects the consumer generates code that is interpreted and executed on the fly representing the logic added in our system by the new policy rule. In our implementation, we have chosen TCL as the scripting language due to the ease with which it interfaces with C, since the ND component is implemented in C. The pseudo code of how the above policy is realised by the Policy Consumer is shown in caption Code 2.

```
$TT_{OA}$: the set of TTs belonging to OA
For each $tt_i \in TT_{OA}$ we get the following:
  $v_{ingress}, v_{egress}$ : ingress, egress nodes
  $b(tt_i)$: bandwidth requirement of $tt_i$
for each $tt_i \in TT_{EF}$ do
  if (($v_{ingress}$ == 4) and ($v_{egress}$ == 6))
    add_LSP ("4-9-7-6", 2000)
    $b(tt_i) = b(tt_i) - 2000$
  Else
    Policy not executed – TT not found
```

Code 2. Pseudocode produced for enforcing (P1)

As it can be seen from the above pseudo-code, it first searches for a TT in the traffic matrix that matches the criteria specified in the conditions of the policy rule regarding the OA, the ingress and egress node. If a TT is found then it executes the action that creates an LSP with the parameters specified and subtracts the bandwidth requirement of the new LSP from the TT in the traffic matrix file so that the ND algorithm will run for the remaining resources. Note that if the administrator had in mind a particular customer for this LSP then this policy should be refined into a lower level policy enforced on the DRtM component, mapping the address of this customer onto the LSP.

The same procedure explained in the previous example is followed again and the policy consumer enforces this policy by generating a script, which is shown in Caption Code 3.

```
maxLinkLoad: the maximum link load utilisation
after the end of the optimisation algorithm
n: the cost function exponent (initially = 1)
Optimisation_algorithm n
while (maxLinkLoad > 80 )
   n = n+1
   optimisation_algorithm n
```

Code 3. Pseudocode produced for enforcing (P2)

As it can be observed from Figure 4, the enforcement of the policy rule caused the optimization algorithm to run for 4 times until the maximum link load utilisation at the final step drops below 80%. The exponent value that achieved the policy objective was $n = 4$. There might be cases that rules like the one above will cause infinite recursion when the algorithm cannot drop the maximum link load below a certain threshold, so a maximum number of iterations should be defined to avoid these kind of problems.

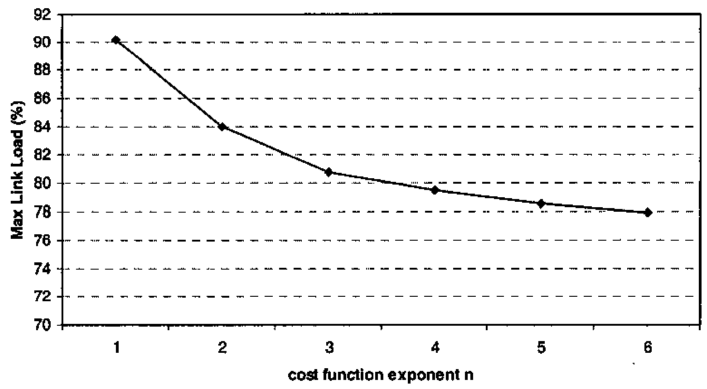


Figure 4. Effect of the cost function exponent on the maximum link load utilization

For the purpose of demonstrating the effects of the enforcement of policies in our system we implemented a TE-GUI shown in Fig. 5. It depicts the topology of the network that the ND component is calculating a new configuration. The GUI draws the links of the topology with different colours according to load utilisation and all the LSPs for every OA created. It has also the capabilities to display overall statistics for the load distribution for every link per OA as well as statistics for every step of the ND algorithm i.e. average link utilisation, link load standard deviation, max link load, running time etc. In the following figure, two snapshots of the TE-GUI are depicted one before and one after the enforcement of the above policies. As it can be seen, the enforcement of the policies caused the link load to fall under 80 % (before the enforcement of policies the link 5->6 was loaded over 90%) as well as the LSP created by the P1 is also drawn.

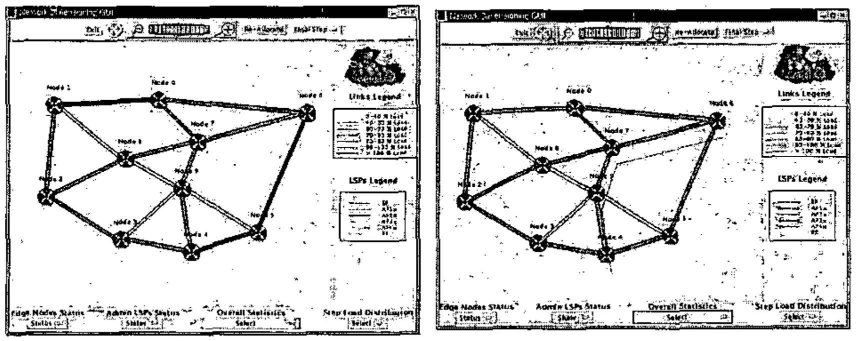


Figure 5. TE-GUI snapshots (a) before and (b) after the enforcement of policies P1 and P2

5. CONCLUSIONS

While most of the work on policies has focused in specifying rules for configuring network elements, our work addresses issues for defining higher level (network-wide) policies that apply to a hierarchical distributed management system. We view policies as a means for enhancing or modifying the functionality of policy influenced components reflecting high-level business decisions. When designing a policy-based system, it is very important to identify the parameters that are influenced by policies resulting in driving the behavior of a system to realize the administrator's business goals. This decision should take into account the inconsistencies caused by the coexistence of policies with "hard-wired" functionality.

In this paper, we presented a policy-driven resource management system and described the components of such a system, focusing on Network Dimensioning. We then presented a detailed description of the design and implementation of the components of the policy sub-system needed to be deployed in order to make our system policy-driven and finally, examples of network dimensioning policies are presented describing their transformation from their definition by the operator until their enforcement.

As a continuation of the work described in this paper, we will be focusing on defining policies for the rest of the components of the TE system and explore the issue of the refinement of policies entered at the Network Dimensioning to lower level policies that apply to Dynamic Resource and Route Management components forming a policy hierarchy. Also we intend to look at the specification of conflict detection and resolution mechanisms specific to our problem domain.

REFERENCES

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

[2] P. Trimintzios et al.. A Management and Control Architecture for Providing IP Differentiated Services in MPLS-based Networks, IEEE Communications, special issue in IP Operations and Management, Vol. 39, No. 5, pp. 80-88, IEEE, May 2001.

[3] P. Flegkas, P. Trimintzios, G. Pavlou. A Policy-based Quality of Service Management Architecture for IP DiffServ Networks, IEEE Network Magazine, special issue on Policy Based Networking, vol. 16, no. 2, pp. 50-56, March/April 2002.

[4] R. Yavatkar, D. Pendarakis, R. Guerin. A Framework for Policy Based Admission Control, Informational RFC 2753, January 2000.

[5] P. Flegkas et al.. On Policy-based Extensible Hierarchical Network Management in QoS-enabled IP Networks, Proceedings of the IEEE Workshop on Policies for Distributed Systems and Networks (Policy '01), Bristol, UK, M. Sloman, J. Lobo, E. Lupu, eds., pp. 230-246, Springer, January 2001.

[6] D. Grossman. New Terminology and Clarifications for DiffServ, IETF IETF Informational RFC 3260, April, 2002.

[7] T. Li, and Y. Rekhter. Provider Architecture for Differentiated Services and Traffic Engineering (PASTE) IETF Informational RFC-2430, October 1998.

[8] Z. Wang, and J. Crowcroft. Quality of Service Routing for Supporting Multimedia Applications, IEEE Journal of Selected Areas in Communications, vol. 14, no. 7, pp. 1228-1234, September 1996.

[9] D. Bertsekas. Nonlinear Programming, (2nd ed.) Athena Scientific, 1999.

[10] P. Trimintzios et al.. Quality of Service Provisioning through Traffic Engineering with Applicability to IP-based Production Networks, to appear in Computer Communications, special issue on Performance Evaluation of IP Networks and Services, Elsevier Science Publishers, Vol. 26, No. 8, 2003.

[11] J. Strassner et al.. Policy Core LDAP Schema, draft-ietf-policy-core-schema-14.txt, January 2002.

[12] E. Gagnon. SableCC, An Object-Oriented Compiler Framework, Master of Science, School of Computer Science, McGill University, Montreal.

[13] B. Moore et al.. Policy Core Information Model – Version 1 Specification, IETF RFC-3060, February 2001.

[14] B. Moore et al.. Policy Core Information Model Extensions, draft-ietf-policy-pcim-ext-08.txt, May 2002.

[15] Y. Snir et al.. Policy QoS Information Model, draft-ietf-policy-qos-info-model-04.txt, November 2001.

[16] M. Wahl, T. Howes, S. Kille. Lightweight Directory Access Protocol (v3), IETF RFC-2251, Decemeber 1997.