A Policy-Based Quality of Service Management System for IP DiffServ Networks

Paris Flegkas, Panos Trimintzios, and George Pavlou, University of Surrey

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 preexisting hard-wired management logic. In this article we first discuss the policy management aspects of architecture for managing quality of service in IP DiffServ networks as presented in [1], and focus on the functionality of the dimensioning and resource management aspects. We then present an analysis of the policies that can influence the dimensioning behavior as well as the inconsistencies that may be caused by the introduction of such policies. Finally, we describe the design and implementation of the generic Policy Consumer component and present the current implementation status.

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, such as 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 is increasing enormously. As such, it is very difficult to build management systems that can cope with growing network size, complexity, and multiservice operation requirements. There is also a need to be able to program management systems and network components to adapt to emerging requirements and subsequently 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 [2]. Policies are seen as a way to guide the behavior of a network or distributed system through high-level declarative directives. The Internet Engineering Task Force (IETF) has been investigating policies as a means for managing IP-based multiservice networks, focusing more on the specification of protocols (e.g., COPS) and object-oriented information models for representing policies. Inconsistencies in policy-based systems are quite likely since management logic is dynamically added, changed, or removed without the rigid analysis, design, implementation, testing, and deployment cycle of hard-wired long-term logic. Conflict detection and resolution are required in order to avoid or recover from such inconsistencies.

In this article we first discuss the policy management aspects of an architecture for managing IP DiffServ networks as presented in [1]; we then focus on the functionality of the dimensioning and resource management parts of the architecture, and present an analysis of the policies that can influence the dimensioning behavior as well as the inconsistencies that may be caused by the introduction of such policies. We finally explore further the design and implementation of the Policy Consumer component and describe our current implementation.

A Policy-Based Quality of Service Management Architecture

In order to support end-to-end quality of service based on service level subscriptions (SLSs), besides the data plane functionality of DiffServ per-hop behavior (PHB) [3] and additional explicit path functionality such as multiprotocol label switching (MPLS), a need for management plane functionality has also been identified. In order to achieve such functionality, the notion of a bandwidth broker (BB) was introduced by [4] as the entity required to perform management and control plane operations on a DiffServ network. The main responsibilities of BBs are to perform admission control, manage network resources and configure leaf and edge devices of the network being managed. In addition to that, BBs can be configured with organizational policies, keep track of the current allocation of marked traffic and interpret new requests to mark traffic in the light of policies and current allocation. A BB may be considered a type of policy manager since it performs a subset of policy management functionality. Ideally, BBs and policy managers should work together to provide an integrated policy services environment [5].

A policy-based functional architecture for supporting QoS

0890-8044/02/$17.00 © 2002 IEEE

in IP DiffServ networks has been designed 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 BB realized as a hierarchical logically and physically distributed system. We present below a brief description of all the aspects of the architecture; a more detailed analysis of the policy aspects is presented in the rest of this section. A detailed description can be found in [1].

Figure 1 shows the basic components of the architecture that are influenced by policies together with the specific policy management components.

Starting from the SLS management part on the left of the figure, the SLS subscription (SLS-S) component includes processes for customer registration and long-term SLS admission control. Customer subscription concerns the service level agreement, containing prices, terms, and conditions, as well as the technical parameters also known as the SLS [6]. SLS-S performs also aids static admission control in the sense that it knows whether a requested long-term SLS can be supported or not given the current network configuration and granted resources. SLS invocation (SLS-I) is the functional block that includes the process of dynamically dealing with a flow; it is part of control plane functionality. It performs measurement-based dynamic admission control by receiving input from the SLS-S (e.g., for authentication purposes) and has a limited view of current spare resources. This admission control takes place at the edges of the network, and finally SLS-I delegates the necessary rules to the traffic conditioner.

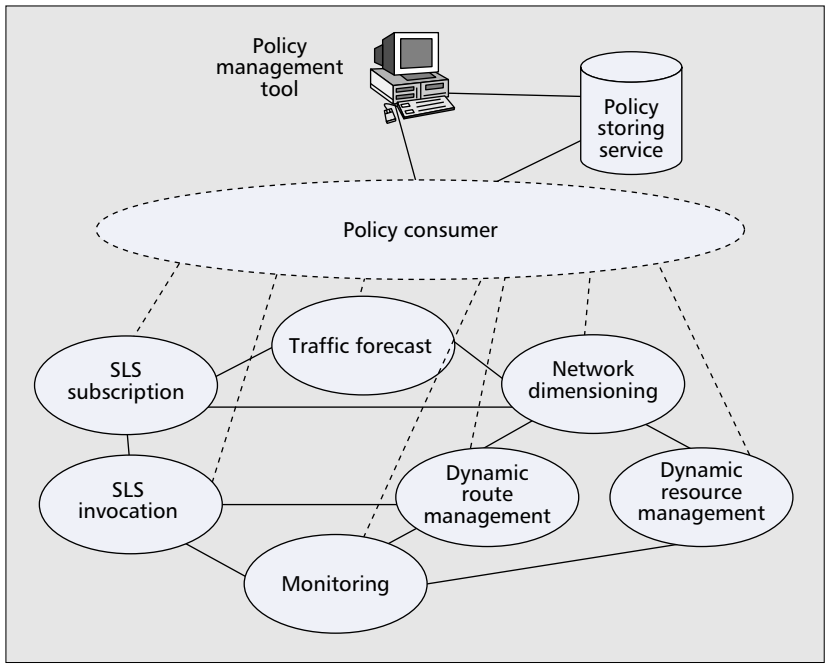


Figure 1. *Policy-based QoS management architecture.*

The traffic engineering aspects of the architecture are on the right of the figure. The network dimensioning (ND) component is responsible for mapping traffic requirements to the physical network resources and providing network provisioning directives in order to accommodate the predicted traffic demands. The lower level of the traffic engineering part intends to manage the resources allocated by network dimensioning during system operation in real time in order to react to statistical traffic fluctuations and special conditions that arise. This part is realized by the dynamic route (DRtM) and dynamic resource management (DRsM). 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 label switched paths (LSPs) is affected 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. It does so by managing the buffer and scheduling parameters according to the guidelines provided by ND. Traffic forecast (TF) generates a traffic estimation matrix based on the currently subscribed SLSs and historical data from measurements. This is in fact the "glue" between the SLS management customer-oriented framework and the traffic engineering resource-oriented framework of the architecture. Thus, the provisioning of the network is effectively achieved by taking into account both the long-term SLSs in a time-dependent manner (ND) and the dynamic network state (DRtM, DRsM).

Policy management includes components such as the policy management tool (PolMT), policy storing service (PolSS), and policy consumers (PolCs or PCs) or policy decision points (PDPs). A single PolMT exists for providing a policy creation environment to the administrator where policies are defined in a high-level declarative language; after validation and static conflict detection tests, they are translated into object-oriented representation (information objects) and stored in the repository (PolSS). PolSS is a logically centralized component but may be physically distributed since the technology for implementing this component is the Lightweight Directory Access Protocol (LDAP) directory [7]. After the policies are stored, activation information may be passed to the responsible PC/PDP in order to retrieve and enforce them.

The methodology for applying policies to a hierarchically distributed system like our architecture is described in detail in [8]. Although Fig. 1 shows a single PolC/PDP on top of all the functional blocks for illustrative purposes, our model assumes many instances of policy consumers. Every instance of the PC is attached to the component that is influenced by the policies this PC is responsible to enforce, complementing in this way the static management intelligence of the above layer of 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 model. Policies may be introduced at every layer of our system, but higher-level policies may possibly result in the introduction of related policies at lower levels, mirroring the system hierarchy.

The components in the architecture can be categorized logically by to which part they belong: SLS components on the left side and TE components on the right (separated horizontally) and hierarchically depending on which layer of the hierarchy they belong (separated vertically). The traffic forecast component has an SLS-aware part that belongs to the SLS management part, and an SLS-unaware part that belongs to the TE part. Components located in the lower level (i.e., SLS-I, DRtM, and DRsM) are dynamic and measurement-based, while those that reside in the upper level that is, ND and SLS-S are static, time-based, and offline.

The same classification can be applied to policies enforced in our system, based on the location of the component to which they relate. So SLS management policies are those enforced to the SLS-S and SLS-I component, and are related mostly to directives and constraints regarding admission control decisions, while TE policies are those enforced to the ND, DRsM, and DRtM components regarding allocation of

Input:
Network topology, link properties (capacity, propagation delay, supported PHBs)

Preprocessing:
- Request traffic forecast per PSC, i.e. traffic trunks (TT) (bandwidth, end-to-end delay, end-to-end loss probability requirements)
- Obtain statistics for the performance of each PHB at each link
- Determine the maximum allowable hop count K per TT according to the above statistics.

Optimization phase:
For each TT find a set of paths (or trees for the hose model) for which:
- The bandwidth requirements of the TT are met
- The delay and loss requirements are met (by using the hop count constraint as an upper bound)
- The overall cost function is minimized

Post-processing:
- Allocate any extra capacity to the paths up to the maximum link bandwidth according to policy
- Sum all the path requirements per link per PSC and configure the PHBs
- Configure the appropriate label switched paths calculated in the optimization phase (note that trees are supported through multiple paths)

Table 1. *Main steps of the ND functionality.*

resources and routing processes. Categorization of policies can also be done according to the layer of the hierarchy the policy-influenced component belongs. Policies that apply to the upper layer of the hierarchy that is, SLS-S and ND are more time dependent and the trigger for their execution depends mostly on the state of the component they apply to. On the other hand, policies enforced to the lower level of the hierarchy (i.e., SLS-I, DRtM, and DRsM) are more measurement-based, meaning they are triggered not only by the component they influence but also by certain conditions and events in the network. Such network state dependent triggers are registered by the correspondent policy consumers to a monitoring component (Fig. 1). The latter notifies them when an event has occurred so that the relevant policy consumer is triggered to enforce the policy actions.

In the rest of the article we concentrate in the functionality of the network dimensioning component and explore issues of making this component policy-influenced by defining ND-specific policies.

Network Dimensioning

Network dimensioning (ND) performs the provisioning activities of the management system. It is responsible for the long- to medium-term configuration of network resources. By configuration we mean the setup of LSPs as well as the parameters (e.g., priority, weight, bandwidth) required for the operation of PHBs on every link. The values provided by ND are not absolute but are in the form of a range, constituting directives for the function of the PHBs, while for LSPs they are in the form of multiple paths for the pipe model or multiple trees for the hose model [1],[1] in order to enable multipath load balancing. The exact configuration values and the chosen path among the multiple paths to be used are determined by DRsM and DRtM, respectively, based on the current state of the network.

ND runs periodically, by first requesting the predictions for the expected traffic per PHB scheduling class (PSC)[2] in order to be able to compute the provisioning directives. The dimensioning period is a week, while the forecasting period is in the timescale of hours. The latter is a period in which we have considerably different predictions as a result of the time schedule of the subscribed SLSs. For example, ND might run every Sunday evening and provide multiple configurations, one for each period of the day (morning, evening, night). So, effectively the provisioning cycle is on the same timescale as the forecasting period. The functionality of ND is summarized in [1], but we also provide a description here in order to be able to explain the relevant policies later.

Its goals are to optimally distribute the projected traffic over the network resources by minimizing the overall cost and at the same time not overloading parts of the network while others are underloaded. In general, this problem can be formulated as a network flow optimization problem [9]. We defined the cost of each link as the sum of linear functions $f_h(x_{l,h})$ per PHB, where $x_{l,h}$ is the load on the link l from PHB h. The total cost should be the sum of f_h for all PHBs over all links; this is the objective function to be minimized.

Another important function of ND is to handle the QoS requirements of the expected traffic in terms of delay and loss requirements. In our implementation of ND functionality, we simplify our optimization problem by transforming the delay and loss requirements into constraints for the maximum hop count for each traffic trunk (TT); the latter is an abstract representation of traffic with specific characteristics (e.g., ingress/egress, class), in fact an aggregation of traffic flows of the same class. 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 nth quantile in order to derive the maximum hop count constraint. We envisage that by using the maximum we are too conservative (appropriate for EF traffic), while by using an average we possibly underestimate the QoS requirements; for example, for AF traffic we may use the 80th percentile. The accuracy of the statistics is determined by the period used to obtain them; methods like smoothing and exponential weighted moving average over long periods must be used.

Finally, we have to configure the network by setting up MPLS LSPs that will support pipe or hose traffic trunks and by provisioning the PHBs for each link according to the sum of the requirements of the paths passing through that link.

The ND functionality is depicted in Table 1 as a set of steps.

Policies for Network Dimensioning

Management standardization efforts in the IP community have traditionally been concerned with management at the network element level, known as element or device management and realized in a centralized fashion. This approach is also reflected in the work undertaken in the area of policy-based management by the IETF: policies are specified for controlling edge nodes, classifying traffic flows, and enforcing decisions related to admission control without addressing the problem of network-wide end-to-end resource management.

[1] *In [6] we used the terms pipe, hose, and funnel models for defining SLSs. The current focus of our work is on supporting the pipe (one ingress and one egress) and hose (one ingress and many egresses) models.*

[2] *A PSC is a PHB group for which a common constraint is that ordering of at least those packets belonging to the same microflow must be preserved. The set of PHBs that are applied to this set of behavior aggregates constitutes a PHB scheduling class. The terms PSC and PHB will be used interchangeably in the rest of this article.*

In the architecture shown in Fig. 1, ND, besides providing long-term guidelines for sharing the network resources, can also be policy influenced, so its behavior can be modified dynamically at runtime, reflecting high-level business objectives. The critical issue for designing a policy-capable resource management component is to specify the parameters influenced by the enforcement of a policy that will result in different allocation of resources in terms of business decisions. These policies, which are in fact management logic, are not hard-wired in the component but downloaded on the fly while the system is operating. However, this may cause inconsistencies since policies have not been tested to coexist with the rest of the system functionality without conflicts.

The ND block is triggered by time and not network state events and as result policies that are enforced on this component are not triggered from events that occur within the network. There are only static dimensioning policies considered ahead of time, and the policy influenced dimensioning function runs for the remaining resources. So two categories of policies are identified for such a static offline resource management component. The first category concerns policies that result in providing initial values to variables, which are essential for the functionality of ND and do not depend on any state but just reflect decisions of the policy administrator. An example of this kind of policies is the definition of the period that ND calculates a new configuration. The second category of ND policies concerns those that depend on the input from the traffic forecast module concerning the predicted volume of traffic the produced configuration should satisfy. Such policies are those whose execution is based on the type of traffic and on the resulting configuration of the network, for example, policies that are enforced if the traffic is EF and those enforced if there is spare physical capacity after the configuration is produced.

We discuss below the directives/constraints specified as policies that should be taken into account when the dimensioning component is calculating a new configuration and the corresponding inconsistencies potentially caused by the dynamic enforcement of such policies.

Since dimensioning runs in a periodic fashion, the policy administrator should specify this period; consequently, ND will ask from the traffic forecast the predicted volume of traffic for this period. The priority of this policy should be specified in order not to cause any inconsistencies when redimensioning is triggered by notifications sent from the dynamic control parts of the system. For example, when DRtM and DRsM are unable to perform an adaptation of the network with the current configuration, and the SLS subscription component has rejected a certain amount of SLSs. The administrator should have the option to force ND either to ignore or not these alarms by prioritizing these events according to business decisions.

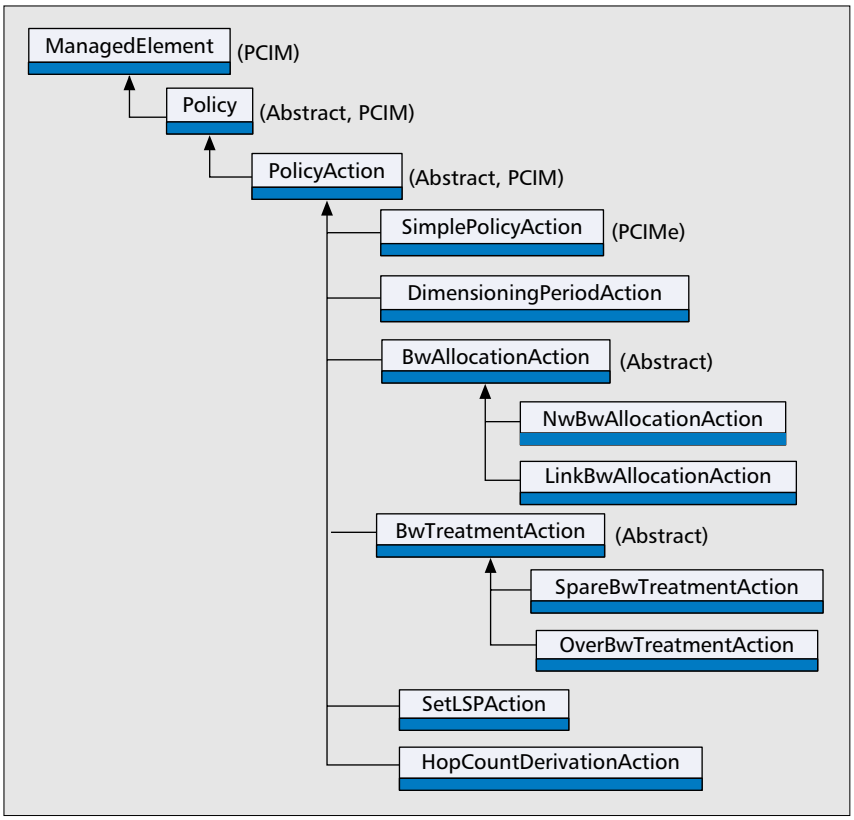
The policy administrator should be able to specify the amount of network resources (giving a minimum, maximum, or range) that should be allocated to each traffic type (i.e., expedited forwarding, assured forwarding, and best effort). This will cause dimensioning to take into account this policy when calculating the new configuration for this PSC together with the information produced by the traffic forecast component of the architecture. More specifically, ND should allocate resources in a way that does not violate the policy, and then calculate the configuration taking into account the remaining resources. This again might result in a conflicting situation where both traffic forecast requirements and policy cannot be satisfied. This means that by enforcing this policy, the SLS requirements for this dimensioning period will not be satisfied, so a conflict detection mechanism should cater for notifying the administrator; the latter should decide whether to proceed with the enforcement of this policy despite the consequences of failing to satisfy the SLS requirements or not. Another policy similar to the previous one can influence the allocation of resources to each PSC in every link belonging to the managed domain in order to meet the requirements specified by this policy, while the previous policy rule caters for overall networkwide allocation. A more flexible option should be for the policy administrator to indicate how the resources should be shared in specific (critical) links.

After the dimensioning algorithm finishes, ND enters a post-processing stage where it will try to assign the residual physical capacity to the various traffic classes. In the case of the MPLS-based approach, allocation of resources can be done up to the point where it can be accommodated over the recently calculated paths (LSPs) for every PSC. This distribution of spare capacity is left to be defined by policies that indicate whether it should be done proportionally to the way resources are already allocated or explicitly for every traffic class. A similar policy to the previous one would be to specify the way the capacity allocated to each PSC should be reduced because the link capacity is not enough to satisfy the predicted traffic requirements. Both these policies might conflict with resource allocation policies explained in the previous paragraph since this increase or reduction of allocated resources to PSCs might result in a situation where the amount of network/link resources specified in the previous policy is violated.

Policies that allow the administrator for a particular reason to explicitly specify an LSP a traffic trunk (TT) should follow can also be defined. Of course, this should override the algorithm's decision about the creation of the LSP for this TT, and it should continue to run for the rest of the entries in the traffic matrix. Inconsistencies might arise since this LSP is not the "best" path to satisfy the objectives of the optimization algorithm, but reflects only a business decision of the administrator. Another constraint related to the creation of trees is the maximum number of alternative trees ND defines for every traffic trunk for the purpose of load balancing.

Another constraint policies may add to the ND component is that of the number of hops of the routes. The administrator should have the flexibility to specify the maximum number of hops routes are permitted to have. This number may vary depending on the QoS class to which the traffic belongs. This has an impact on the loss probability and delay constraint of TTs because the dimensioning algorithm actually translates these requirements on an upper bound on the number of hops per route. Thus, this policy should override the default value that ND calculates. Moreover, how this translation is done can also be influenced by policy rules. For example, the safest approach to satisfy the TT requirement would be to assume that every link and node belonging to the route induces a delay equal to the maximum delay caused by a link and node along the route. So this policy rule will allow the administrator to decide if the maximum, average, or minimum delay, or loss induced by a node or link along the route should be used to derive the hop count constraint according to the PSC to which the TT belongs.

The approximate cost function used by ND is linear to the bandwidth allocated to a PHB, that is, $f_{l,h}(x_{l,h}) = \text{al,h}\, x_{l,h}$, where $x_{l,h}$ is the bandwidth allocated to PHB h on link l and $a_{l,h}$ is a constant. The value of this constant belongs to a range (e.g., 0.5–1.5) depending on the PHB. This policy provides the flexibility to the policy administrator to specify this constant depending on cost (i.e., importance) of a particular PHB. A more flexible approach would be to allow the administrator to specify with the policy the cost function to be used.



■ Figure 2. *Class inheritance hierarchy.*

Policy Information Model and Language

An object-oriented information model has been designed to represent the network dimensioning policies described above, based on the IETF Policy Core Information Model (PCIM) and its extensions specified in [10, 11], 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 PCs belonging to different systems understand the same semantics of policy and have a mutual knowledge of how policies are stored in the policy repository despite the fact that each PC might interpret it differently. The IETF has described a QoS Policy Information Model [12], 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 in policies configuring the network elements.

Figure 2 depicts a part of the inheritance hierarchy of our information model representing the policies discussed in the previous section, and also indicates its relationships to PCIM and PCIMe. Note that some of the actions are not directly modeled. Instead, they are modeled by using the class SimplePolicyAction with the appropriate associations, using instances of the variable and value classes ("SET <variable> TO <value>").

For example, the maximum number of alternative trees 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. Conditions are also modeled by using the class SimplePolicyCondition with instances of the variable and value classes (IF <variable> matches <value>). For example, when the condition of the policy is based on the PSC to which the traffic belongs, it is modeled by a simple condition where PSC is an implicit variable and the corresponding value is an integer (EF is 1, AF11 is 2, etc.). 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 PSC (depending on the policy condition) at a networkwide level and in every link, respectively. The SpareBwTreatmentAction and OverBwTreatmentAction 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.

A high-level definition language has also been designed and implemented to provide the administrator the ability to add, retrieve, and update policies in the Policy Storing Service. 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 PolSS through LDAP *add* operations, according to an LDAP schema of our information model, which was produced following the guidelines described in [7]. 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 to which this policy belongs, so the generated LDAP entries should be placed under the correct policy group entry. The time period condition field specifies the period the policy rule is valid and supports a range of calendar dates, masks of days and months, as well as a 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 in both the Disjunctive Normal Form (DNF) (an ORed set of ANDed conditions) and 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 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; if not, they are treated as rule-specific, placing the condition entry below the policy rule entry.

An example of one of the policies described in this section in our proprietary policy language is given below:

PolicyRule1: **From** Time=0900 **to** Time=1800
If (PSC == EF) **and** (spareBw == True)
then allocateSpareBw>30%

This policy forces ND to allocate at least 30 percent of the spare bandwidth to traffic that belongs to EF PSC if there is spare physical capacity after the configuration of the network is produced. A time period condition is also added that makes this policy valid only between 09:00–18:00. The time period condition is mapped to an instance of ptpConditionAuxClass[3] with the attribute ptpConditionTimeOfDayMask set to T090000/T180000, which is attached to an instance of a policyRuleValidityAssociation[3] structural class. The overall condition of the policy rule is considered of DNF type with the simple policy conditions belonging to the same group. The corresponding policyVariable and policyValue auxiliary classes are attached to policyRuleConditionAssociation[3] structural classes so that each can be retrieved with a single LDAP search operation. Finally, the action of the policy rule is mapped to an instance of the spareBwTreatmentAction (Fig. 2) auxiliary class, which is attached to a policyRuleActionAssociation[3] structural class, with the attributes bwUnits set to 1 (%) and minBw set to 30.

We present below a description of the design and implementation of the generic PC as well as the overall implementation of our system.

Design and Implementation

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. In the following figure a decomposition of the Policy Consumer component is depicted and its interactions with the other components of the architecture. A similar approach was presented in [13] for implementing a policy manager-agent.

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. Thus, 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. This approach has also been followed by the management by delegation (MbD) paradigm [14] and the IETF Script management information base (MIB) [15] specifying an implementation architecture. PCs can be attached to the component they are associated with, having local access to the MOs, or enforce the policies remotely. The former approach is depicted in Fig. 3.

As shown in Fig. 3, the PC component is decomposed in three parts. The first part, the *repository client*, provides access to the Policy Storing Service, and is responsible for downloading the associated objects stored in PolSS that make up the policy rules this specific PC should enforce in order to influence the behavior of the component to which it is attached. The repository client has knowledge of how the policy information is stored in the PolSS. After a new policy is stored, it receives a notification from the policy management tool that a new policy for this consumer is stored in the repository by passing the appropriate information (i.e., the distinguished name of the "root" policy object); it then goes and retrieves all the necessary entries from the PolSS. This part is the same for every instance of the PC in our architecture.

The second part of the PC is the *script generator*, which is responsible for creating the script that implements the policy. It contains logic, specific to the component to which the PC is attached, that automates the process of generating a script from the higher-level representation of policies as they are stored in the PolSS. The *policy interpreter* provides the "glue" between the PC and the policy-based component, and interprets a language, which includes functions that perform management operations. Two kinds of functions are identified: those that provide access to local MOs and those that provide access to MOs of remote components. Since a policy rule is defined as a set of actions when certain conditions occur, the purpose of accessing MOs is twofold. First, evaluating the condition of a rule can be done by either polling attributes of the MOs or following an event-driven paradigm where the MOs send a notification that triggers the execution of the policy action. These MOs can be either local or remote, depending on whether the condition of the policy is based on the state of the component to which this PC is attached or on information that can be available from a remote component. Second, policy actions are implemented by executing scripts that result in setting attributes or invoking operations available at the object boundary of the local MOs.

An important consideration regarding the implementation of a PC is related to the choice of the relevant scripting language. Since in our architecture most of the components are implemented in C++, the first important requirement for the

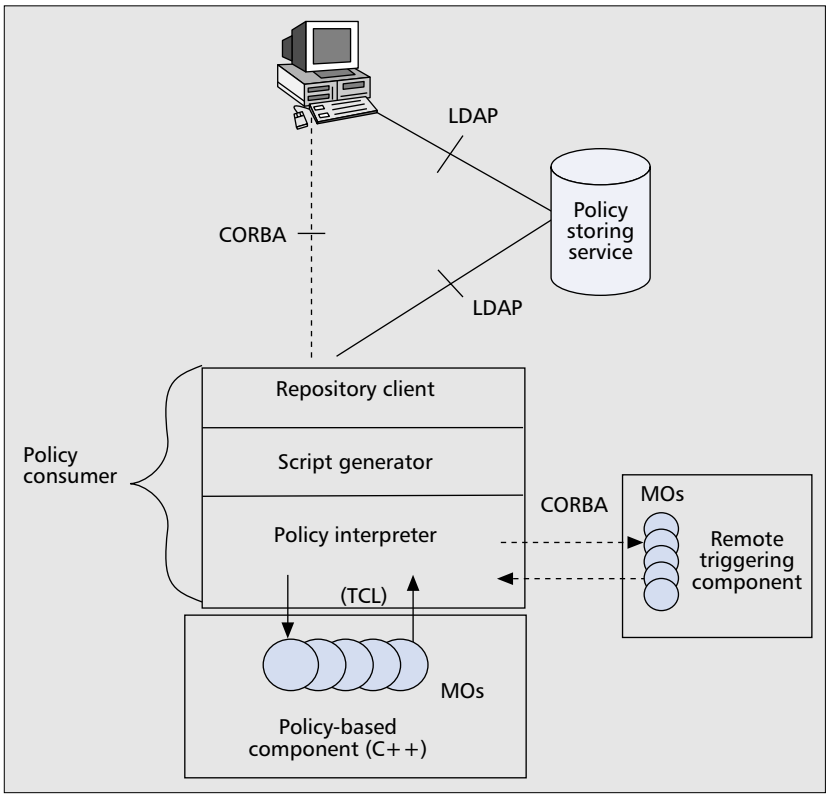


Figure 3. *Decomposition of a policy consumer.*

[3] *These classes are all defined in the policy core LDAP schema [7].*

scripting language is to be extensible and able to interface easily with C++. Tcl was chosen for this purpose due to the ease with which it interfaces to C++. The interface between Tcl and the C++ environment of the component is in two directions: from Tcl to C++, for accessing local or remote managed objects, and from C++ to Tcl for sending notifications and starting the execution of the policy scripts. In order to support remote invocations on MOs, a Common Object Request Broker Architecture (CORBA) environment is used.

The current implementation of our system includes only one instance of the policy consumer attached to the ND component. In addition to the policy notation presented in the previous section, a directory browser has also been implemented, providing the ability to browse through the stored entries and evaluate that all the policies entered in the format described above are correctly translated and represented as LDAP entries in the directory. Since the system described in an earlier section 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 Storing Service as an LDAP directory offering a CORBA IDL interface, identical to the LDAP specifications [16], to the rest of the components.

It is our intention to use this system on top of a simulated network using the ns simulator as well as on a Linux-based DiffServ-capable testbed with the ND component producing corresponding configuration commands reflecting the policies entered using our policy notation.

Conclusions and Future Work

While most of the work on policies has focused on specifying rules for configuring network elements, our work addresses issues for defining higher-level (networkwide) 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 article we describe the salient characteristics of policy-based management in a hierarchical system for IP DiffServ management, presenting an initial categorization of policies depending on the "location" of the components they influence. We then present the way the network dimension component allocates the resources of the network and describe the parameters influenced by policies focusing on conflicts that may appear. We also briefly present an object-oriented information model for representing these policies by extending the IETF core information model and describe the format of a policy rule in our policy notation. Finally, design and implementation issues of the policy consumer are discussed and the status of our implementation is briefly presented.

As a continuation of the work described in this article, we will be concentrating on defining policies for the rest of the components of our architecture (Fig. 1) and study how they can coexist with the rest of the static functionality of the system. The refinement of policies entered at ND to lower-level policies that apply to dynamic resource and route management components will also be explored. Moreover, we will be focusing on the specification of conflict detection and resolution mechanisms specific to our problem domain. We intend to share our experiences with the overall policy-based QoS management system in future papers.

References

[1] P. Trimintzios *et al.*, "A Management and Control Architecture for Providing IP Differentiated Services in MPLS-based Networks," *IEEE Commun. Mag.*, Special Issue in IP Operations and Management, vol. 39, no. 5, IEEE, May 2001, pp. 80–88.

[2] M. Sloman, "Policy Driven Management For Distributed Systems," *J. Net. and Sys. Mgmt.*, vol. 2, no. 4, Dec. 1994, pp. 333–60.

[3] S. Blake *et al.*, "An Architecture for Differentiated Services," IETF RFC-2475, Dec. 1998.

[4] K. Nichols, V. Jacobson and L. Zhang, "A Two Bit Differentiated Services Architecture for the Internet," IETF RFC 2638, July 1999.

[5] R. Neilson *et al.*, "A Discussion of Bandwidth Broker Requirements for Internet2 Qbone Deployment," Internet2 Qbone Advisory Council, v 0.7, http://www.merit.edu/working.groups/i2-qbone-bb/doc/BB_Req7.pdf, Aug. 1999.

[6] D. Goderis *et al.*, "Service Level Specification Semantics and Parameters," draft-tequila-sls-01.txt, June 2001.

[7] J. Strassner *et al.*, "Policy Core LDAP Schema," draft-ietf-policy-core-schema-13.txt, Nov. 2001.

[8] P. Flegkas *et al.*, "On Policy-based Extensible Hierarchical Network Management in QoS-enabled IP Networks," *Proc. IEEE Wksp. Policies Dist. Sys. Net.*, Bristol, U.K., M. Sloman, J. Lobo, and E. Lupu, Eds., Jan. 2001, pp. 230–46.

[9] R. Ahuja, T. Magnanti, and J. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall, 1993.

[10] B. Moore *et al.*, "Policy Core Information Model — Version 1 Specification," IETF RFC-3060, Feb. 2001.

[11] B. Moore *et al.*, "Policy Core Information Model Extensions," draft-ietf-policy-pcim-ext-06.txt, Nov. 2001.

[12] Y. Snir *et al.*, "Policy QoS Information Model," draft-ietf-policy-qos-info-model-04.txt, Nov. 2001.

[13] D. Marriott and M. Sloman, "Implementation of a Management Agent for Interpreting Obligation Policy," *Proc. IFIP/IEEE Int'l. Wksp. Dist. Sys.: Ops. and Mgmt.*, L' Aquila, Italy, Oct. 1996.

[14] Y. Yemini, G. Goldszmidt, and S. Yemini, "Network Management by Delegation," *Proc. 2nd Int'l. Symp. Int. Net. Man.*, Washington, DC, Apr. 1991.

[15] D. Levi and J. Schoenwaelder, "Definitions of Managed Objects for the Delegation of Management Scripts," IETF RFC-2592, May 1999.

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

Biographies

PARIS FLEGKAS (P.Flegkas@eim.surrey.ac.uk) received a Diploma in electrical and computer engineering from Aristotle University, Thessaloniki, Greece, and an M.Sc. in telematics (communications and software) from the University of Surrey, U.K., in 1998 and 1999, respectively. Currently he is studying for his Ph.D., while he is a research fellow in the Networks Research Group at the Center for Communication Systems Research (CCSR) of the University of Surrey, working in European Union and United Kingdom funded research projects. His research interests are in the areas of policy-based management, traffic engineering, IP QoS, and network and service management.

PANOS TRIMINTZIOS (P.Trimintzios@eim.surrey.ac.uk) received a B.Sc. in computer science and an M.Sc. in computer networks, both from the University of Crete, Greece, in 1996 and 1998, respectively. From 1995 to 1998 he was a research associate at ICS-FORTH, Greece, working on research projects involving high-speed network management and charging network and user services. Currently he is a research rellow at CCSR, University of Surrey, where he is also finishing his Ph.D. His main research interests include traffic engineering, QoS provisioning, network performance control, policy-based networking, network monitoring, and service offering.

GEORGE PAVLOU (G.Pavlou@eim.surrey.ac.uk) is professor of communication and information systems at CCSR, School of Electronics and Computing, University of Surrey, where he leads the activities of the Networks Research Group. He received a Diploma in electrical engineering from the National Technical University of Athens, Greece, and M.Sc. and Ph.D. degrees in computer science from University College London. His research interests include network planning and dimensioning, traffic engineering and management, policy-based networking, programmable and active networks, multimedia service control, and technologies for object-oriented distributed systems. He has contributed to standardization activities in ISO, ITU-T, TMF, OMG, and IETF. He was technical program co-chair of IEEE/IFIP Integrated Management 2001.