Journal of Network and Systems Management, Vol. 11, No. 3, September 2003 (© 2003)

An Adaptive Policy-Based Framework for Network Services Management

Leonidas Lymberopoulos,[1,2] Emil Lupu,[1] and Morris Sloman[1]

This paper presents a framework for specifying policies for the management of network services. Although policy-based management has been the subject of considerable research, proposed solutions are often restricted to condition-action rules, where conditions are matched against incoming traffic flows. This results in static policy configurations where manual intervention is required to cater for configuration changes and to enable policy deployment. The framework presented in this paper supports automated policy deployment and flexible event triggers to permit dynamic policy configuration. While current research focuses mostly on rules for low-level device configuration, significant challenges remain to be addressed in order to:a) provide policy specification and adaptation across different abstraction layers; and, b) provide tools and services for the engineering of policy-driven systems. In particular, this paper focuses on solutions for dynamic adaptation of policies in response to changes within the managed environment. Policy adaptation includes both dynamically changing policy parameters and reconfiguring the policy objects. Access control for network services is also discussed.

KEY WORDS: Policy-Based Management; management of differentiated services; service management; adaptive management; policy adaptation.

1. INTRODUCTION

Network services are developing from best-effort packet-forwarding services to services that provide Quality-of-Service (QoS) guarantees to the user. Two approaches have been proposed for providing QoS to services within IP networks. *Integrated Services* (IntServ) [1] uses the Resource ReSerVation Protocol (RSVP) [2] to provide per-flow QoS support by dynamically reserving resources on RSVP-enabled routers. *Differentiated Services* (DiffServ) is a much simpler alternative to IntServ/RSVP. The QoS information is encoded in the Type of Service (ToS) byte in the IP header to identify different classes of service.

[1]Department of Computing, Imperial College London, London, United Kingdom.
[2]To whom correspondence should be addressed at 180 Queen's Gate, SW7 2BZ, London, United Kingdom. E-mail: *llymber@doc.ic.ac.uk*

1064-7570/03/0900-0277/0

Service Level Agreements (SLAs) are established between a service provider and its customers to formally define the expectations and obligations that exist in their business relationship. SLAs can also be defined between multiple peer service providers who cooperate to provide an overall service that spans multiple administrative domains.

Many current approaches to specifying Service Level Agreements, particularly for network services, concentrate on specifying quality-of-service parameters such as delay, throughput, error rates and availability. The specification of the service is essentially static in that it often assumes a single type of service is provided at all times; but many clients require services which vary according to date or time. In addition, 'fallback' classes of services should be provided under failure conditions when the main class of service cannot be provided — service adaptation may take place either as a result of failures within the network or to accommodate changes in client application requirements. For example, a collaborative design application may switch from an audio phase to a phase needing video services, so the client application must be able to trigger changes to the underlying communication service.

A service provider may provide a sophisticated set of services, which are offered to a client organization consisting of many different users. Not all users within a client organization may need access to all the offered services. Authorization should be part of the SLA management system to specify which users are permitted to access particular services or functions within the services. This information is also likely to change during the lifetime of the SLA as new services or service functions are offered, or the set of client users changes.

The *Ponder* language developed at Imperial College provides a framework for specifying both authorization policies—the conditions under which users can perform actions on resources—and, obligation policies—event triggered condition action rules. It is a declarative object-oriented language with support for policy structuring to cater for policy specification in complex systems. In this paper we discuss some of the issues that arise in using Ponder for service management, and then focus on how our policy-based management framework can be used to provide dynamic management of services in Differentiated Services (DiffServ) networks. The fundamental objective of policy-based management is to allow flexible and adaptive management where the policies define the adaptation choices or strategy which can be modified without recoding or even shutting down the system. In this paper we describe the use of policies for adaptation at the service layer to select and modify policies at the network layer.

The rest of the paper is organized as follows: in Section 2 we outline the requirements for a policy-based system for service management. Section 3 briefly presents the Ponder language, and Section 4 analyzes the use of policy adaptation and gives an enforcement architecture for an adaptive policy system. Section 5 presents how our adaptive policy framework can apply in a Differentiated Services

environment, followed by a description of how it has been implemented in a network simulator in Section 6, as well as some results from the simulation. In Section 7 we present and compare our approach with related work, and the final section discusses conclusions and directions for future work.

2. SERVICE MANAGEMENT ISSUES

Consider a typical network of a large enterprise which consists of several local area networks (LANs) interconnected with a wide area network (WAN) through one or more access routers. The IT department of the enterprise is responsible for operating the network so as to satisfy the SLA established in the enterprise. Following the policy based management approach, the administrator will deploy network policy rules and the management system will automatically distribute the rules to the network devices. The enforcement of the policy rules will provide the network service QoS guarantees to the applications using the service. For example, if the established SLA in the enterprise states that "*A video application between clients in Site A and a video server in Site B should receive Gold Service*" and Differentiated Services architecture [3] is deployed in the network then the administrator should deploy a policy rule that instructs the network to forward the packets that belong to the video application according to the Expedited Per Hop Behavior [4].

A more sophisticated approach towards the automatic deployment of SLAs is a management system that can automatically derive network policy information from service specific information. In this approach, the technical part of the SLA is specified as a set of Service Level Specifications (SLSs). A SLS is a set of parameters (throughput, delay, jitter etc.) and their values which together define the service offered to a traffic stream by a QoS-enabled network. It includes specific values or bounds for the traffic stream QoS metrics (e.g., round-trip delay, throughput, packet loss probability, etc.). The management system will perform a mapping function from the SLSs within an SLA, in order to derive network policy information, as shown in Fig. 1. In general, refining an abstract high level service specification to implementable policies is extremely difficult and cannot be automated, but the SLS considered here is a comparatively simple set of parameters which can be used to query a database in order to automate the mapping.

An interesting variation of this could be the deployment of a mapping function responsible not only for deriving the parameters of a network policy from the SLS parameters, but also for selecting which network policy will be used for the application described in the SLS. For example, if "*Gold Network Service*" is defined with specific low values on the upper bounds of round-trip delay and packet loss, then a network policy, which can guarantee these specific bounds should be chosen for the video application.

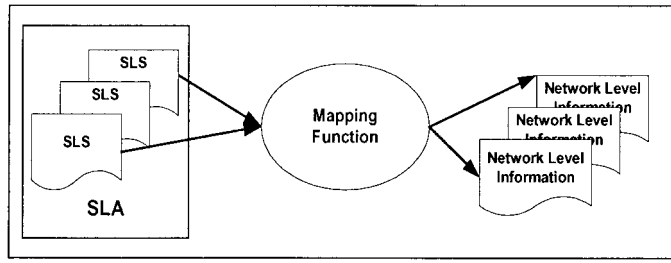

Fig. 1. SLS to network-level policy mapping.

In addition to mapping from SLS to network policy information, a management system should also support dynamic service management in order to react to changes that require modification of the existing network configuration. Figure 2 outlines the typical cases where the management system should change the existing network configuration—these include:

- A new user or an application request changes to the provided QoS. In the video application example, clients in site A may request more network

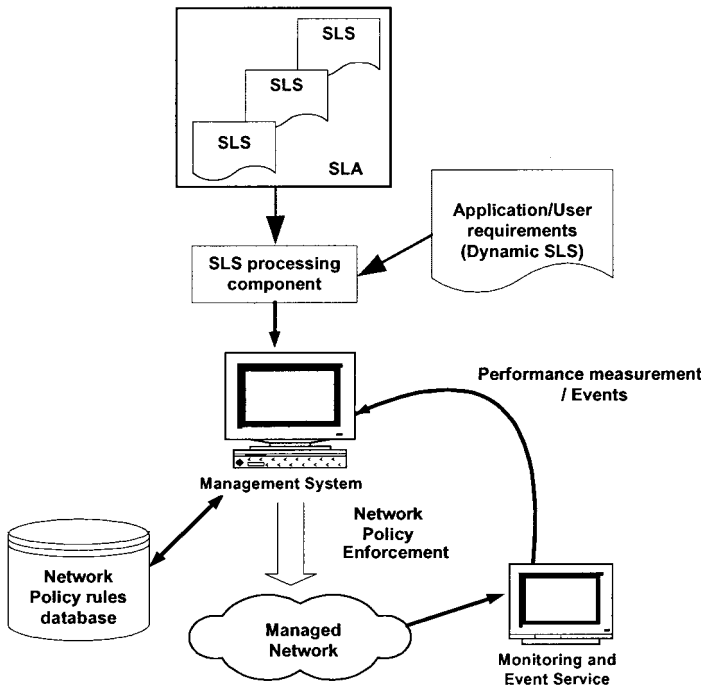

Fig. 2. Service management with a policy-based management system.

resources for a running session, in order to receive better video quality from the video server located at Site B. Adaptive applications, which tailor their behavior according to the available network resources, can change their QoS requirements at run-time. This implies that network policy attributes must be changed at run-time to support the new user or application requirements.

- Performance measurements from a monitoring service may indicate performance degradation, thus requiring changes in the service network configuration or even the selection of a new service to cater for the client application. This in turn may require attribute changes in the deployed network policy rules or even the selection of a different network policy to cater for the application. For example, if a deployed network policy that handles the video application packets can no longer guarantee low packet loss due to high congestion, then a different network policy rule which can guarantee low packet loss should be chosen for the video application.
- Events indicating network failures or time events may trigger changes. For example, a network policy deployed only within a specific path of routers in the managed network may not be suitable for the video application when the routing path inside the managed domain changes. In this case, a new network policy, which can be applied to the new path, must be automatically configured and distributed in order to handle the video application packets.

In addition, it is necessary to specify who is authorized to access specific services or management functions. A certain group of users should be able to access either specific services or functions within the provided service. For example, the administrator may want "Gold" service to be accessible only to users in Sites A and C, but not to users in Site D of the enterprise. On the other hand, only users with administrative privileges in Sites A and B should be given the ability to change parameters of the service, such as the bandwidth allocated to the service. This information can also change dynamically as new services are being offered or the set of client users changes.

We propose an adaptive policy-based framework to cover the wide range of requirements identified earlier for the management of services. In this, policy is specified with Ponder [5], a declarative, object-oriented language for specifying security and management policies for distributed systems. Policy adaptation is specified and enforced by other policies, specified in the same Ponder policy notation.

3. THE PONDER POLICY LANGUAGE

Ponder is an object-oriented, declarative language for specifying management and security policies. This paper focuses on the use of *obligation policies*,

```
inst oblig policyName "{"
on              event-specification ;
subject   [<type>] domain-Scope-Expression ;
[ target  [<type>] domain-Scope-Expression ;]
do              obligation-action-list ;
```

Fig. 3. Obligation Policy Syntax.

which specify the actions that managers must perform when certain events occur, and provide the ability to respond to changing circumstances. Obligations are event-triggered condition-action rules, which explicitly identify the *subjects* (i.e., managers or configuration agents) that are responsible for performing the management actions on *target* objects. Both subject and target objects are specified in terms of *domains*, which are a means of grouping objects to which policies apply [6]. Events can be internal, e.g., a timer event, or external events, which are collected and distributed by a monitoring service. Composite events can be specified using the event composition operators that the language supports. The syntax of obligation policies is shown in Fig. 3.

Actions can be operations defined in the management interface of the target object or internal operation of the management agent. In the latter case, the target element of a policy is optional. The concurrency operators "->" and "||" separate the policy actions in the obligation action list and respectively specify whether actions should be executed sequentially or in parallel. The optional catch-clause specifies an exception that is executed if the execution of the policy actions fails for some reason. This syntax is used for declaring a policy instance. The language provides reuse by supporting definition of policy types, which can be instantiated for each specific environment. Figure 4 shows the syntax for declaring obligation policy types and instantiations.

Policies are automatically deployed into the relevant Policy Management Agents (PMA) specified by the subject of the policy. The PMA interprets and enforces the obligation policies on a domain of target devices. In the current Ponder prototype implementation [7], an obligation policy enforcement object is implemented as a Java program downloaded to a PMA. The PMA registers with

```
type oblig policyType "(" formalParameters ")" "{"{ obligation-policy-parts } "}"
inst oblig policyName = policyType "(" actualParameters
```

Fig. 4. Obligation Types and Instantiations.

the event service to receive the relevant events, which will trigger the policies it holds. Events may pass parameters to the PMA.

We have given a very brief overview of Ponder. More details on authorization policies, event composition, composite policies and constraints can be found in (see Damianou *et al.* [5]) and a discussion on conflict detection and resolution in (see Lupu and Sloman [8]).

4. POLICY ADAPTATION WITHIN THE PONDER FRAMEWORK

When applying policies to network elements, the policy actions are those provided by the management interface of the managed element. Thus, the "level of abstraction" of the policies is determined by the available implementation. However, as discussed in Section 2, service management may require adaptation of existing network policies to cater for changes within the managed network. Thus, policies themselves need to be managed and adapted. In this paper, we identify different adaptation requirements and show how policy adaptation can itself be specified and enforced by other policies, specified in the same Ponder policy notation.

We use the term "*Policy Adaptation*" to describe the ability of the policy-based management system to modify network behavior in one of the following ways:

- Adaptation by dynamically changing the parameters of a QoS policy to specify new attribute values for the run-time configuration of managed objects.
- Adaptation by selecting and enabling/disabling a policy from a set of pre-defined QoS policies at run-time. The parameters of the selected network QoS policy are set at run-time.
- Adaptation by learning which are the most suitable policy configuration strategies from the system behavior. This can be used to select policies or even generate new ones when needed.

In this paper, we will focus only on the first two categories of policy adaptation as adaptation by learning still requires considerable further work.

4.1. Run-Time Modification of Policy Parameters

In the general case, the specification of a network-level QoS policy used for dynamically adapting the managed devices' configuration follows the format shown in Fig. 5.

In Fig. 5, an event triggers the execution of the policy in one or more subjects, i.e., Network-level Policy Management Agents (PMAs). Using the **EventParameters**, the PMA calculates the required policy **ActionParameters**,

```
inst oblig   NetworkQoSPolicy {
subject      NetworkLevelPMA;
target       targetSet = TargetDomainofDevices;
on           Event(EventParameters[]);
do           ActionParameters[] = CalculateActionParameters(EventParameters[]) ->
             targetSet.executeAction (ActionParameters[]); }
```

Fig. 5. Generic format for network QoS policy.

by calling the internal method **CalculateActionParameters**, then it invokes the relevant policy actions on the target objects in the **TargetDomainofDevices** with the new policy parameters.

Policies provide a flexible means for providing this type of adaptation rather than components written in a procedural language. New adaptation strategies can be incorporated into the management system by adding new policies which react to different events using the existing policy actions or by replacing existing policies with new versions, which either implement new actions on the managed objects or new actions on the Policy Management Agents. With programmable networks, new actions may be added, via a management interface in network elements, so the policies can be updated to access this new functionality. The code that implements new actions or new calculation methods within the Policy Management Agent's engine can be loaded at run-time either through the administration console or by the new policies themselves. If the functionality of the PMA were implemented using a traditional programming language, it would be necessary to recompile the code and replace the agent which would require stopping the system while this was being done.

4.2. Adaptation by Dynamically Selecting and Enabling Policies from a Set of Policies

In this approach, higher-level control policies are triggered by reconfiguration events and determine which lower-level network policy must be enabled/disabled to adapt the configuration of the managed system. As we discussed in the previous section, the advantage of using policies rather than a procedural language for selecting and enabling the appropriate network-level policies is that modifying or adding new management strategy at this level can be achieved by replacing the control policy or adding new ones. Furthermore, the same Ponder deployment framework can be used to distribute both high-level control policies and network QoS policies [7]. In this paper we will focus on service management policies as

```
inst oblig   GenericServicePolicy {
subject      ServicePMA;
on           AdaptationRequest (params[]);
do           QoSpolicy = selectPolicy (params[])->
             QoSPolicy.enable() ->QoSpolicy'sParams [] =  calculate (QoSPolicy, params[]) ->
             EventService.GenerateEvent( QoSPolicy'sObligationEvent, QoSpolicy'sParameters []);}
```

Fig. 6. Specification of a generic service management policy.

an example of these higher-level policies. In the general case, a service management policy is specified with the template obligation rule **GenericServicePolicy**, presented in Fig. 6.

In the policy shown in Fig. 6, an **AdaptationRequest** event triggers the selection of policies for configuring network elements such as routers. The **ServicePMA** first selects the most appropriate network-level policy to actually implement the configuration, using the **selectPolicy** method; the policy is enabled; parameters related to the specific policy are calculated; and, finally an event is sent via the event service to pass parameters and trigger the network-level policy. The network-level policy will be interpreted by one or more Network-level PMAs which will actually configure network devices. Note that the advantage of triggering the network-level policy via the event service is that there may be multiple agents managing subsets of the network devices. These agents will all receive the event and configure their respective devices.

4.3. Enforcement Architecture

In the general case, the management functionality of the generic Policy Management Agent ServicePMA is specified with the obligation rule **GenericServicePolicy**, presented in Fig. 6.

The enforcement architecture is presented in Fig. 7.

1. The **ServicePMA** receives the event **AdaptationRequest** from the event service. As discussed in Section 2, the adaptation event could be from a new application requiring changes to QoS, performance measurements coming from a monitoring service that require changes in the service network configuration, events indicating network failures or time events, etc.
2. The **ServicePMA** requests the current policy database from the policy service.

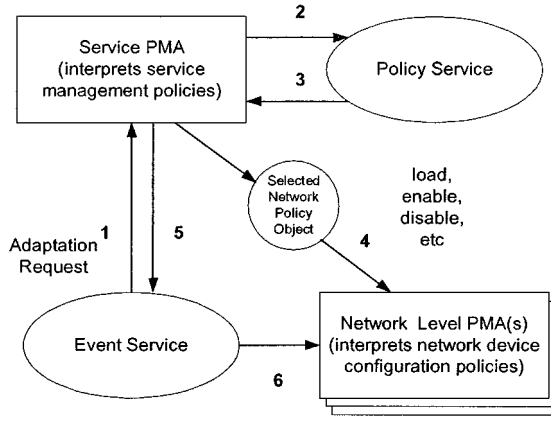

Fig. 7. Enforcement architecture for policy adaptation.

3. The policy database is received and the **ServicePMA** then invokes a selection algorithm to choose a suitable network policy from the set of implemented network policies.
4. The **enable()** method is called on the selected network policy object, which in turn calls the **enable()** method on the relevant Network-level PMAs. Enabling the policy means that policy enforcement objects within the PMAs register the obligation event with the event service, as described by Dulay [7]. At this point, the selected policy is activated in the Network-level PMAs, but it needs to be triggered by the obligation event to perform the policy actions. Furthermore, an "old" policy can be unloaded or disabled from the corresponding PMAs.
5. The obligation event is generated with the network policy calculated parameters to trigger the policy.
6. The event service disseminates the obligation event to all Network-level PMAs that are registered to receive the specific event.

5. SERVICE MANAGEMENT OVER DIFFERENTIATED SERVICES NETWORKS

In our approach, adaptation is enforced by higher-level policies. This section presents a usage scenario, where network policy that provides Per Domain Behavior in a Differentiated Services environment is adapted by service management policies. Service management policies are enforced by Policy Management Agents at the service-level. The latter are responsible for the management of services that run within the managed DiffServ network. Section 5.3 presents how

authorization policy, specified in the Ponder notation can be used to control access to the services provided in the DiffServ network.

5.1. Network-level Per Domain Behavior Policies

The IETF DiffServ working group Nichols and Capenter [9] has proposed the term Per Domain Behavior (PDB) to describe the behavior experienced by a particular set of packets as they cross a DiffServ domain. A PDB is characterized by specific metrics that quantify the treatment a set of packets with a particular DSCP (or set of DSCPs) will receive as it crosses a DiffServ domain. A PDB specifies a forwarding path treatment for a specific aggregate. A PDB is implemented with one or more Per Hop Behaviors (PHBs). A PHB describes the forwarding behavior of a DiffServ node applied to a particular DiffServ behavior aggregate. PHBs are implemented in nodes by means of some buffer management and packet scheduling mechanisms. Each PDB has measurable attributes that can be used to describe what happens to its packets as they enter and cross the DiffServ domain. In our framework, each PDB is implemented as a network-level policy rule. Each rule guarantees the PDB attributes to the corresponding traffic aggregate. Table I presents examples of QoS guarantees that PDB policies can offer to their associated traffic aggregates.

In our framework, PDB policies are specified as Ponder obligation rules. The actual implementation of the PDB policy, i.e., the implementation of the PHB (or the set of PHBs) that will guarantee the QoS characteristics to the corresponding traffic aggregate, is hidden from the customer. The customer (human or automated agent) is offered the externally observable PDB QoS attributes. An example of a PDB Ponder policy rule is given here.

Example 1. Policy rule for providing a specific PDB

```
instoblig  /Policies/PDBPolicy1
subject    /PMAs/DiffServAgent;
target     r = /DiffServDomainA/Routers/CoreRouters;
on         PDB1_ConfigRequest(DS, max_input_rate, min_output_rate);
do         /* DS: The Diffserv codepoint for EF: 101110.PDB1 is
           implemented with the EF PHB*/
           r.applyEFPHB(DS, max_input_rate, min_output_rate);
when       max_input_rate <= min_output_rate; /* Property that EF
           traffic must satisfy */
```

In this example, we assume that the core routers within the DiffServ domain implement the EF PHB, so the PDB policy will configure them accordingly. A possible implementation described by Jacobson et al. in [4] of the **applyEFPHB** action can use the Weighted Round Robin scheduling algorithm to schedule the packets at the egress interface of each core router. The weight for the EF traffic

Table I. PDB Policies and Their QoS Characteristics

PDB identifier	Enforcement network policy	Assured bandwidth (Mbps)	Delay (ms)	Jitter (ms)	Loss (%)	Enforcement routers path	Time when valid	
PDB1	/Policies/Policy1	10	≤20	≤3	≤1	⟨ rl, . . . , rN⟩	Every day	
PDB2	/Policies/Policy2	20	≤10	≤1	≤0.1	⟨ rl, . . . , rM⟩	Working hours	
	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

class is calculated from the parameter **min_output_rate** as:

weight = **min_output_rate/total_output_rate**,

where **total_output_rate** is the egress interface's bandwidth. Note that the policy rule in this example will configure the core routers within the DiffServ domain to implement the EF PHB on the corresponding traffic aggregate. An additional policy is needed to provide the necessary configuration to the edge routers of the DiffServ domain. When the network-level PMA **DiffServAgent** receives the event **PDB1_ConfigRequest**, it invokes the **applyEFPHB** action on all routers in the target domain. This way, all core routers within the target domain will guarantee a minimum output rate (throughput) to the EF-marked packets, when these packets do not exceed the configured maximum input rate at the ingress router interface. An alternative implementation of EF could use priority scheduling at the egress interface of the target core routers. More details on the specification of network-level DiffServ policies can be found (see Lymberopoulos et al. in [10]) which also describes the generic enforcement architecture within the Ponder deployment model.

Our current implementation extends the Ponder toolkit [11] with the functionality to enforce DiffServ policies. Policies in the Ponder toolkit are Java RMI objects. The DiffServ specific policy actions (e.g., **applyEFPHB**) are methods within the policy object that the network-level Policy Management Agents invoke when triggered by the configuration request event. Policy actions are constructed using the DiffServ element classes that the DiffServ implementation [12] provides.

5.2. Service Management Policies

The Per Hop Domain Behavior (PDB) policies which we introduced in the previous section are enforced by DiffServ enabled Network-level PMAs (Fig. 7). These configure the QoS mechanisms of the managed devices within the DiffServ network. However, as we have already discussed in Section 2, network service management requires additional functionality. The required functionality that enables dynamic service management is provided in our framework by Service Level Policy Management Agents at the service level. In the following, we will provide examples of service management policies for dynamic service management.

5.2.1. SLS to PDB Mapping Policy

SLS to PDB mapping can be performed by the **ServiceManagementAgent** when the administrator triggers the policy rule **SLSMappingPolicy** by means of an SLS request.

Example 2. SLS to PDB mapping policy

```
instoblig   SLSMappingPolicy {
subject     ServiceManagementAgent;
on          SLS_Request (SLS_parameters);
do          pdb_policy = select_using_algorithmA(SLS_parameters)->
            pdb_policy.enable()->
            pdb_policy_parameters = calculate (pbd_policy, SLS_parameters)->
            EventService.GenerateEvent (pdb_policyObligationEvent,
            pdb_policy_parameters;
```

In Example 2, a new SLS request from an application with specific parameters will trigger the **SLSMappingPolicy**. The **ServiceManagementAgent** uses the SLS parameters to select a suitable PDB policy from the policy database. The PDB policy is enabled and triggered to configure network devices. Note that we use the Tequila project SLS approach for DiffServ SLS parameter specification [13]. A "parser" component within the Ponder management toolkit is used to translate the Tequila external SLS specification to pairs of ⟨parameter, value⟩. These pairs are stored in the structure SLS_parameters and are conveyed to the agent with the obligation event **SLS_Request**. Upon the receipt of the **SLS_Request**, the **ServiceManagementAgent** will select the appropriate PDB policy for this specific request. An example of a selection algorithm is outlined in [14], where the PDB is selected according to the triple ⟨delay, loss, throuhput⟩.

5.2.2. Policy to Handle Service Performance Degradation

A number of different adaptation strategies could be used for handling service run-time performance degradation, notified by the monitoring service, as indicated in Fig. 2. There may be a need to dynamically change these strategies by replacing a policy within the **ServiceManagementAgent** with a new version or by enabling/disabling different versions of the policy. Policies provide a more flexible means of implementing this type of service-level adaptation than scripts or special purpose code. Events indicating high delay or high packet loss could trigger policies in the **ServiceManagementAgent**. In the following examples, we indicate adaptation strategies, which could be implemented by Ponder policies for the management of a network service such as the video client application described in Section 2, but we do not define the actual policies. In all the examples, we assume that the video application receives the EF network service and that the EF PHB is implemented as presented in Example 1.

The monitoring system detects that the EF service end-to-end delay exceeds a threshold so it generates a **HighDelay** event received by the **ServiceManagementAgent**. Corrective actions which may be performed include: a) Increase the minimum departure rate of the EF traffic at the egress of every core router to guarantee that the service packets (especially large ones) will remain in the output

queue for less time before being transmitted to the next hop; and, b) Notify the client application to choose a different state, which requires less bandwith and hence decreases the incoming traffic rate at the ingress interface. This way, the EF aggregate will experience less delay.

The action for a **HighPacketLoss** event would be to increase the maximum arrival rate of the incoming EF traffic at the ingress interfaces of both the edge and the core routers that the EF traffic traverses. This will reduce the number of packets being dropped by the policer at the ingress interface. Alternatively, as packet loss is proportional to the aggregation degree, the number of EF microflows can be reduced, in order to reduce packet loss in the remaining EF traffic.

5.2.3. Policy to Support Changes in Routing or Link Failures

A PDB is usually associated with a path of routers within the DiffServ domian (e.g., when using DiffServ over MPLS). When a link fails or routing changes for a specific class of traffic, the corresponding PDB may not be guaranteed by the routers in the new path. A new PDB must be selected for this class of traffic that satisfies the network service QoS requirements and that can be served by the new path. Which PDB is selected and how, is a decision that can be formulated as a service management policy. The following example provides a policy that implements this functionality.

Example 3. Policy for configuring DiffServ upon link failures or routing changes

```
instoblig  RoutingChangePolicy {
subject    ServiceManagementAgent;
on         routeChanged (newPath);
do         pdb = select_using_algorithmA(SLS_params, newPath) ->
           pdb.enable() -> pdb_params [] = calculate(SLS_params) ->
           EventService.GenerateEvent (pdbObligationEvent, pdb_params[]);
```

This policy instructs the **ServiceManagementAgent** to find a suitable PDB for the service with SLS parameters (**SLS_parameters[]**) when the path of routers that serves the service packets has changed. Information about the new path is conveyed to the **ServiceManagementAgent** with the **routeChanged** event. This event could be triggered by a component which is responsible for detecting and reporting routing changes. The new PDB is selected by the selection algorithm A. As in Example 1, the selection of the PDB can be adapted by replacing the existing policy with one using a different selection algorithm.

5.2.4. Policy to Reflect Changes in Application or User Requirements

The user/application may request different QoS guarantees at run-time by updating SLS parameters. As a consequence, network policy attributes must be

changed to support the new user/application requirements. A policy example, which enables the **ServiceManagementAgent** to provide this type of service adaptation is given later.

Example 4. Policy for re-configuring DiffServ when SLS parameters change at run-time

```
instoblig   SLSRenegotiationPolicy {
subject     ServiceManagementAgent;
on          SLS_Request (new_SLS_parameters[], service_id);
do          pdb_policy = policyService.lookup (service_id) ->
            new_pdb_policy_parameters[] = calculate (pbd_policy,
            new_SLS_parameters[]) ->
            EventService.GenerateEvent(pdb_policyObligationEvent,
            new_pdb_policy_parameter[]);}
```

In this policy example, the event SLS_Request carries both the new SLS parameters that the application/user requires and a unique identifier of the client application that requires its SLS renegotiation (this identifier could be the Flow Description parameter [13] of the Tequila SLS). The PDB policy reference that is responsible for this specific service is obtained via a lookup() operation on the Policy service, assuming that a table containing the service identifiers and their PDBs is updated when the initial request for SLS to PDB mapping has been successful. Alternatively, the PDB that will guarantee the new service requirements could be selected at run-time among the set of implemented PDBs, as in the SLSMappingPolicy in the example 1.

5.3. Service Authorization Policies

As we have discussed in Section 2 of this paper, authorization should be part of the service management system to specify which users are able to access particular services or functions within the services. Consider a scenario where users request network services for their applications through Service Access Points (SAPs). Access control agents should be implemented at each SAP to interpret authorization policies and control requests related to the service. In Example 5, the policy rule **GoldServiceAccessControlPolicy** allows only users from sites A and C to perform the action of allocating “Gold Service” to their client applications. “Gold Service” will be allocated to the client application only if the requested bandwidth is less than 100 Mbps.

Example 5. Policy for controlling users' access to a particular network service

instauth+ GoldServiceAccessControlPolicy{
subject /Users/SiteA_users + /Users/SiteC_users;
target ServiceAccessPoint_Agent;
action allocateGoldService (client_application, bandwidth);
when bandwidth < 100;

It is possible to permit selected customer administrators to access the SAP to set service parameters such as changing the bandwidth allocated to the "Gold Service". This can be implemented using the following policy:

Example 6. Policy for controlling access to management function within a network service

instauth+ GoldService_BandwidthControlPolicy {
subject /Users/SiteA_users/Admins + /Users/SiteB_users/Admins;
target ServiceAccessPoint_Agent;
action allocateBandwidthToGoldService(bandwidth);}

6. IMPLEMENTATION OF A PROTOTYPE ADAPTIVE MANAGEMENT SYSTEM AND EVALUATION

Our enforcement architecture has been implemented using the Ponder toolkit [11]. A Service Policy Management Agent implements one or more selection algorithms which query a simple policy database to choose a suitable policy. Using a graphical tool, the administrator selects which network QoS policies should be included in the policy database and edits the QoS attributes (see Table I) of each selected policy. New policies can be included in the policy description database at run-time or existing policies can be removed. The administrator can also load new selection algorithms in the Service PMA. This allows new adaptation strategies to be implemented within the management system at run-time by adding new service management policies which use the new selection algorithms.

The applicability of our approach to the adaptive management of Differentiated Services networks has been tested on simulated DiffServ networks, using the JavaSim [15] network simulator, which offers DiffServ functionality. We have extended the JavaSim simulator with new components, which allow the communication between the simulator and the Policy Management Agents, which are implemented using the Ponder toolkit. These components and their interaction with the Ponder Policy Management Agents are presented in Fig. 8.

Monitoring Components measure the performance characteristics such as throughput, packet loss and end-to-end delay for DiffServ traffic classes during a simulation. These measurements are used to display performance graphs and by *Event Components* to generate performance degradation events when traffic

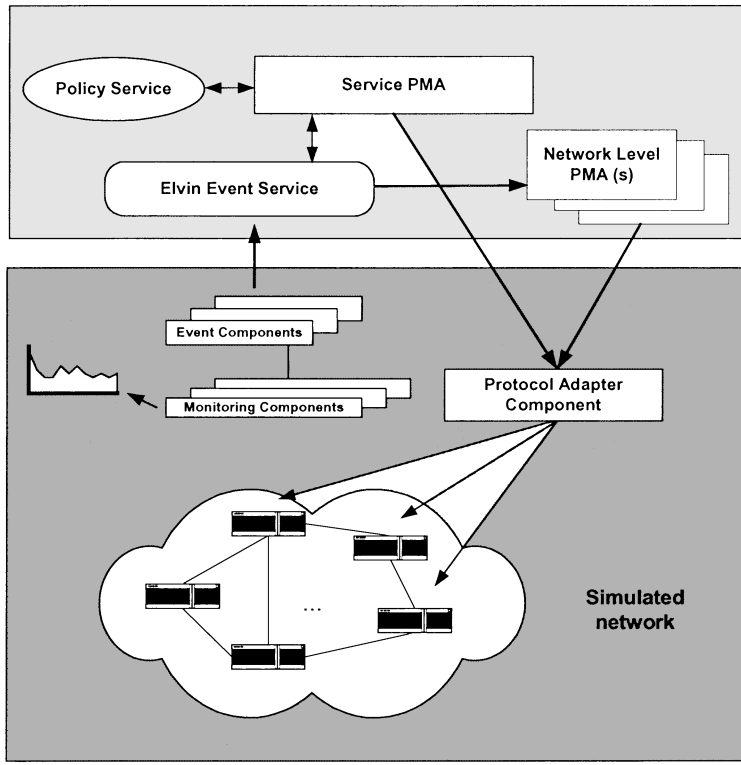

Fig. 8. Prototype implementation of an adaptive policy based management system.

measurements exceed specific thresholds. For example, an event indicating packet loss will be created when the packet loss of a specific class of traffic exceeds a configured threshold.

In addition, *Event Components* generate events indicating network failures or new user/application QoS requirements. For example, when a host inside the simulated network requests certain QoS for for a new traffic flow, a QoS request event will be generated. All events are dispatched to the Service PMA, where they trigger policies, through the Elvin Event Service [16], a publish/subscribing messaging system, which we are currently using within the Ponder Toolkit. The policies either select lower-level network policies to enable or enforce corrective actions on the managed devices. In this implementation, corrective actions are invoked on the target devices directly by the Service PMA, although the same actions could be initiated by Network-level PMAs after a Service PMA request.

The Policy Management Agents and the JavaSim simulator run on different hosts. Policy actions are communicated to the simulated nodes through the *Protocol*

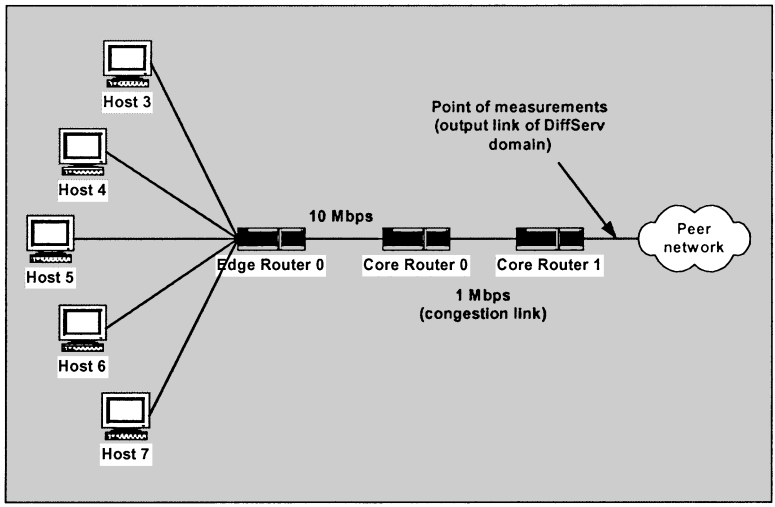

Fig. 9. Simulated DiffServ network.

Adapter Component, which acts as an interface to the simulator. Policy Management Agents send TCP messages, using our proprietary text-based protocol to the *Protocol Adapter Component* which translates the messages into commands sent to the management interface of the target simulation nodes. Note that our policy based management system abstracts the protocols used for communication between a manager agent and the managed device. A *Protocol Adapter Component* could easily be implemented to translate management actions into standard policy protocols such as COPS or SNMP which would be supported by real network elements.

We present results of an experiment based on a simulated typical DiffServ network, with the topology indicated in Fig. 9. We implemented three PDB policies within this network, the *GreenPolicy PDB* with the EF PHB, the *YellowPolicy PDB* with the AF11 PHB and the *RedPolicy PDB* with the BE PHB. We used priority scheduling at the egress interface of each core router. Using a graphical tool, the administrator selects which network policies should be included in the QoS policy database, and edits the QoS attributes for each PDB. The tool interfaces with a domain browser to select policies from the directory server, where policy objects are stored. The values assigned to QoS attributes represent the attributes' upper thresholds, where -1 indicates there is no upper threshold.

In this experiment, we show how our adaptive management system automatically performs the SLS to PDB mapping for new SLS requests, implemented with the policy *SLSMappingPolicy* from Example 2. The event *SLSRequest* parameters are the address of the node issuing the SLS request and the requested values of

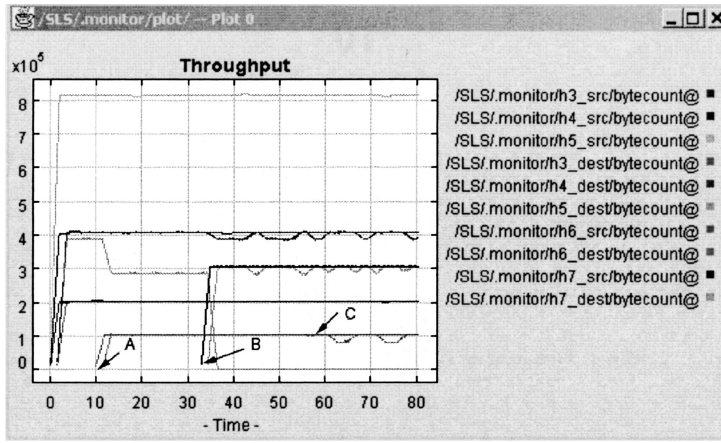

Fig. 10. Throughput of Individual traffic flows.

delay, packet loss and jitter with the selection algorithm choosing the closest PDB according to the triple ⟨delay, packetLoss, jitter⟩. Figure 10 indicates the throughput of each individual flow that traverses the DiffServ network, while Figure 11 displays the throughput of the DiffServ aggregates. At first, Host 3 sends 200Kbps of EF traffic, host 4 sends 400Kbps of AF11 traffic and host 5 sends 800 Kbps of BE traffic. At point A in Figs 10 and 11, host 6 issues a SLS request for a 100 Kbps service which guarantees delay <80 ms, packet loss <0.3% and jitter < 6ms. This SLS request is handled by our management system to adapt configuration of the

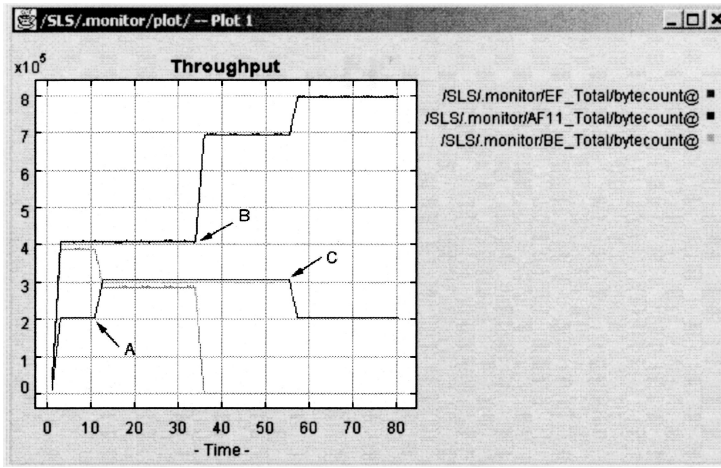

Fig. 11. Throughput of DiffServ aggregates.

DiffServ network. As we can observe, the ServicePMA chooses the *GreenService* for this SLS request and triggers the *GreenPolicy*. As a result of the enforcement of the *GreenPolicy*, the DiffServ network adapts configuration to guarantee the EF PHB to the traffic originating from host 6.

At point B, host 7 issues a SLS request for a service of 300 Kbps with delay $<$120.0 ms, packet loss $<$2% and jitter $<$10 ms. This time the ServicePMA chooses the *YellowService* and host 7 receives the AF11 PHB. We can observe that after this point the BE class suffers from starvation, since the total throughput of EF and AF11 traffic is 1000 Kbps, which is the maximum capacity of the core network. Later, at point C, the host 6 request an alternativel service of 100 Kbps with delay $<$110 ms, packet loss $<$3% and jitter $<$15 ms. The ServicePMA reacts to this request by choosing the AF11 service for the user application.

We have conducted several other experiments within simulated DiffServ networks. These demonstrate how service management policies cater for the dynamic management of network services. However, due to space limitations, we will not provide in this paper the results from these experiments.

7. RELATED WORK

Various frameworks have been proposed for providing service management in QoS enabled networks. Many of them propose a Service Level Specification to configuration mapping function in their architecture. Other research groups are working on policy specification and enforcement. Our work aims at bringing together these areas, by showing how to use the flexibility of a policy based management framework for dynamic service management.

The IETF Policy working group is defining a framework for managing QoS within networks [17]. They do not have a language for specifying policies but are using the X.500 directory schema. IETF policies are of the form *if* ⟨*set of conditions*⟩ *then do* ⟨*a set of actions*⟩. Directories are used for storing policies but not for grouping subjects and targets. They do not have the concepts of subject and target that can be used to determine to which components a policy applies, so the mapping of policies to components has to be done by other means (i.e., interface roles). Furthermore, they do not support policy rules that can be dynamically triggered by events to reconfigure the managed system according to changing circumstances. The policy work in the IETF seems to be focused only in the network layer and they have not considered the interaction between application and network policy.

A number of vendors are marketing policy toolkits for defining policies for DiffServ enabled networks, e.g., [18, 19]. Most of these are similar to the IETF ideas. None of them supports a language but they do have graphical editors that allow the administrator to define individual policies and then explicitly identify the enforcement components to which the policies must be loaded. None of these tools

appear to have considered the automation of the policy lifecycle and how to adapt the configuration of network elements when conditions change. New configuration need to be imposed manually by the administrator through the management console.

A policy-based management system is proposed by Verma et al. [20] for managing Service Level Agreements within DiffServ networks. They use a tabular specification where a policy table contains entries, which map traffic aggregates into classes of service. The list of PHBs that different devices support is obtained by a resource discovery mechanism. Thus, rather than providing a policy-based management system for managing the characteristics of DiffServ devices, the proposed system only maps application flows into predefined and already implemented PHBs. Moreover, this system can only communicate policies to the enforcement devices during the initial configuration process, initiated by the administrator and cannot adapt to run-time changes in the managed environment. In addition, the scope of this approach is specifically aimed at a management system for a DiffServ network, whereas our work is applicable to a wide range of management areas.

A SLS to DiffServ configuration mapping framework is proposed by Prieto and Brunner [14]. The management system consists of two parts. The first performs both the SLS to PDB mapping process and an admission control process. The mapping module uses an N-dimensional space (e.g., delay, packet loss, and throughput) to classify an input SLS into an available intradomain service, which is offered by an implemented PDB within the DiffServ network. The second is the policy-based control part which controls the SLS mapping and the admission control processes. Network policy is used as the device configuration mechanism but they not have any concrete proposals for the policy part of the framework. Furthermore, the SLS to PDB mapping process is only initiated by the user; no actions are undertaken by the management system to dynamically select a new PDB when network conditions change.

Keller *et al.* [21] proposes a contract-based architecture for application-level service management. Contracts are used for defining, deploying, monitoring and enforcing SLAs in a dynamic e-Business environment. A generic object-oriented model describes the various sections of a contract between a client and a service provider. Contracts are managed by a Contract Management System, whose main functional components are: a measurement, a violation detection and a management component. The measurement component is responsible for collecting data relevant to service QoS parameters. The violation detection component retrieves data from the measurement component and evaluates if the guarantees defined in the contract are met. In case of a QoS violation, a notification is sent to the management component which initiates corrective measures. The advantage of our proposed framework for network-level service management is the flexibility to implement dynamically new management strategies within the service management system.

A Customer Service Management (CSM) architecture [22] allows delegation of the service management task from the service provider to the customer. Customers can adjust SLS parameters through a parameter setting function block within the CSM module which implements a SLS mapping function, to derive device configuration from SLS information. Our framework can provide this functionality, by allowing users to trigger the execution of management actions within the Service Management Agent.

The framework proposed by Yoshihara *et al.* [23] adapts policy parameters on monitoring the network. A management script includes policies, expressed in the IETF representation, and also specifies how the policy life cycle should be managed. The script notifies the management system about QoS threshold violations. In this work, a prototype implementation is provided for Differentiated Services, where policy parameters, such as the peak rate of a traffic profile, its peak burst size and the associated DSCPs, are changed dynamically to adapt to system behavior. The framework we propose for the adaptive management of DiffServ can specify, in a uniform way, all the necessary information required for enforcement and adaptation of policies using obligation rules. In addition to providing adaptation by changing policy parameters, we can also select new policies to be enabled upon events other than just QoS violation events.

An architecture for the management of a network offering active services is presented by Marshall *et al.* [24]. A bacterial algorithm forms the basis for the adaptation performed by autonomous controllers which are programmed (like a bacterium) to autonomously replicate policies that improve its performance and de-activate policies that degrade performance. This way, "useful" policies spread and "poor" policies die out. A policy is evaluated though a fitness (revenue-cost) function. In this work, each policy is related to one active service; policies control the deployment of services (proxylets) in their active services environment. Marshall and Roadknight [25] presents an example of this type of adaptation for providing QoS differentiation of active services, where the queue length of network servers (DPSs) is adapted to provide either short delay or low loss to service(s), depending on the users QoS requirements. Example of these requirements (policies or service genes) can be: "Accept request for service A if DPS <80% busy" of "Accept request for service C if queue length <20". In our framework, policies are used in a more generic sense, describing the actions that management agents must undertake when receiving different types of requests. We provide adaptation, in a more systematic way, by adapting the policy based management system itself, either by changing attributes of policies or by removing and adding new policies.

A QoS Architecture transport system for a multicast, multimedia networking environment offers a QoS configurable API at the transport layer, which enables applications to have control over QoS [26]. QoS is specified at the API in terms of a flow specification, which includes parameters such as delay, throughput, jitter etc. and a QoS policy. The QoS policy enables users to advise the infrastructure

on how to deal with the flow when resource availability changes. A distributed QoS adapter interprets the policy and is responsible for informing applications when resources become available. A QoS adaptation protocol is implemented for the communication between QoS adapters. Our framework can provide this functionality, but also it may apply adaptive behavior in other circumstances, as we presented through the examples in Section 4.

A lot of work on QoS adaptation has also been carried out in the Distributed Systems area, which we will not present due to space limitations. Most of this work provides adaptation by hard coded QoS management and monitoring in middleware systems for supporting multimedia applications.

8. CONCLUSIONS AND FUTURE WORK

In this paper we have presented an adaptive policy-based framework for network services management. Our approach provides the administrator the flexibility to define network, service management and service authorization policy using the Ponder high-level language.

Network policy rules configure the QoS mechanisms of the devices within the target domain. Network rules specified within our framework are dynamically triggered by events, in order to change the configuration of the managed objects under certain circumstances. This dynamic configuration of policy forms the basis of the adaptive management our framework can provide.

We presented how policy adaptation in our framework is enforced by higher-level Ponder-policies. Adaptation is provided in one of the following ways: a) by dynamically changing the parameters of a QoS policy to specify new attribute values for the run-time configuration of managed objects; and, b) by selecting and enabling/disabling a policy from a set of pre-defined QoS policies at run-time. The parameters of the selected network QoS policy are calculated and set at run-time.

Service management policies are a specific case of higher-level management policies which adapt the underlying network policy. We presented examples that demonstrate how service management policies cater for the dynamic management of services in a Differentiated Services network.

One issue that needs to be addressed here is the problems that may arise using an adaptive management system rather than manual intervention to adapt network configuration. Possible problems are that the management system may oscillate or fail to deliver a feasible network configuration. For the first problem, we give the network administrator the ability to configure the *Event Components* of the adaptive management system, to filter out frequent events of the same type and to use thresholds to ensure that small changes in performance do not generate events to trigger configuration changes. Badly programmed policies could possibly introduce instability, as is the case with any incorrect program. In order to deliver a feasible network configuration, we are currently working on the design

and implementation of a validation process inside our adaptive management framework. This process will ensure that the policy is consistent with the functional or resource constraints within the target environment.

Finally, in order to protect network services from unauthorized usage, we provide the administrator the ability to specify with service authorization policies which users are able to access particular services or functions within the services.

An important issue that needs to be addressed is to enhance the functionality of the Service Management system to initiate corrective actions which are not pre-defined. Currently, its task is to adapt the set of underlying network policies upon pre-defined conditions. However, corrective measures should be undertaken to remedy any causes of violations in the delivery of the service to the client application. This will require the management system to carry out problem determination tasks and to perform root cause analysis in order to initiate the corrective actions when violations are detected. We also intend to experiment with Linux based routers as well as commercial routers or switches to evaluate the performance implications of executing policies on routers. Future work also includes the application of our approach to the management of MPLS networks.

9. ACKNOWLEDGMENTS

We gratefully acknowledge the support of the EPSRC for research grant GR/R31409/01 (PolyNet) as well as Cisco Systems for support on the Polyander project.

REFERENCES

1. R. Braden, D. Clark, and D. Shenker, Integrated Services in the Internet Architecture: an Overview, RFC 1633, June 1994.
2. R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, ReSerVation Protocol (RSVP) Version 1 Functional Specification, RFC 2205, September 1997.
3. M. Carlson, W. Weiss, S. Blake, Z. Wang, D. Black, and E. Davies, An Architecture for Differentiated Services, RFC 2475, December 1998.
4. V. Jacobson, K. Nichols, and K. Poduri, An Expedited Forwarding PHB, RFC2598, September 1999.
5. N. Damianou, N. Dulay, E. Lupu, and M. Sloman, The Ponder Policy Specification Language, Proc. Policy 2001, *International Workshop on Policies for Distributed Systems and Networks*, Bristol, United Kingdom, January 29–31 2001.
6. M. Sloman, and K. Twidle, Domains: A framework for structuring management policy, Chap. 16 *Networks and Distributed Systems Management*, Sloman, pp. 433–453, 1994.
7. N. Dulay, E. Lupu, M. Sloman, and N. Damianou, A Policy Deployment Model for the Ponder Language, *IEEE/IFIP International Symposium on Intergrated Network Management*, Seattle, May 14–18 pp. 529–544, 2001.
8. E. Lupu and M. Sloman, Conflicts in policy-based distributed systems management, *IEEE Transactions on Software Engineering, Special Issue on Inconsistency Management*, Vol. 25, No. 6, pp. 852–869, December 1999.

9. K. Nichols and B. Carpenter, Definition of Differentiated Services Per Domain Behaviors and Rules for their Specification, RFC 3086, April 2001.
10. L. Lymberopoulos, E. Lupu and M. Sloman, An An adaptive Policy Based Management Framework for Differentiated Services Networks. *IEEE Third International Workshop on Policies for Distributed Systems and Networks*, Monterey, California June 5–7 2002.
11. N. Damianou, N. Dulay, E. Lupu, M. Sloman and T. Tonouchi, Tools for domain-based policy management of distributed systems. *Eighth Network Operations and Management Symposium*, Florence, Italy, April 15–19 2002.
12. P. Martinez, M. Brunner, J. Quittek, F. Strauß, J. Schönwälder, S. Mertens, and T. Klie, Using the script MIB for policy-based configuration management, *Eighth Network Operations and Management Symposium*, Florence, Italy, April 15–19 2002.
13. D. Goderis, Y. T'joens, C. Jacquenet, G. Memenios, G. R. Egan, D. Griffin, P. Georgatsos, L. Georgiadis, and P. V. Heuven, Service Level Specification Semantics, Parameters and negotiation requirements, *draft-tequila-sls-01.txt*, June 2001.
14. A. Prieto and M. Brunner, SLS to DiffServ configuration mappings, *12th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Nancy, France, October 15–17 2001.
15. DRCL JavaSim, *http://www.javasim.org*
16. Elvin: Content Based Messaging, *http://elvin.dstc.edu.au/*
17. Y. Snir, Y. Ramberg, J. Strassner and R. Cohen, Policy Framework QoS Information Model, Internet Draft, *draft-ietf-policy-qos-info-model-03.txt*, April 2001.
18. Cisco COPS QoS Policy Manager product documentation, *http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/qos/qpm2_1/index.htm*
19. Allot Communications NetPolicy Policy Based Management System product documentation, *http://www.allot.com/html/products_netpolicy.shtm*
20. D. Verma, M. Beigi, and R. Jennings, Policy Based SLA Management in Enterprise Networks. *International Workshop on Policies for Distributed Systems and Networks*, Bristol, United Kingdom January 29–31 2001. Springer-Verlag LNCS, pp. 137–152, 1995
21. A. Keller, G. Kar, H. Ludwig, A. Dan, and J. Hellerstein, Managing Dynamic Services: A Contract-based Approach to a Conceptual Architecture. *Eighth Network Operations and Management Symposium*, Florence, Italy, April 15–19 2002.
22. R. Sprenkels, A. Pras, B. Beinjum, and L. de Goede, A Customer Service Management Architecture for the Internet. *11th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Texas, December 4–6 2000.
23. K. Yoshihara, M. Isomura, and H. Horiuchi, Distributed Policy-based Management Enabling Policy Adaptation on Monitoring using Active Network Technology. *12th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Nancy, France October 15–17 2001.
24. I. Marshall, H. Gharib, H. Hardwicke, and C. Roadknight A novel architecture for active service management. *IEEE/IFIP International Symposium on Intergrated Network Management*, Seattle, pp. 795–810, May 2001.
25. I.W. Marshall and C.M.Roadknight, Provision of quality-of-service for active services, *Computer Networks*, Vol. 36, No. 1, June 2001.
26. A.T. Campbell, A Quality of Service Architecture, Ph.D. Thesis, Lancaster University, United Kingdom January 1996.

Leonidas Lymberopoulos graduated in July 2000 from the Department of Electrical Engineering and Computer Science of National Technical University of Athens. Since September 2000, he has been

a Research Associate in the DSE Group in the Department of Computing, Imperial College London. His research interests are in the area of Policy-Based Networking.

Emil Lupu is a lecturer in the Department of Computing, Imperial College London. He has over 9 years experience in policy-driven network and systems management and serves on the program committee of the IFIP/IEEE Integrated Management and Network Operations and Management symposia, and on the steering committee of the Policy for Networks and Distributed Systems conference.

Morris Sloman leads the DSE Group in the Department of Computing Imperial College London. He is a member of the editorial board of the *Journal of Network and Systems Management*, the steering committee for the IFIP/IEEE Integrated Management Symposium, and the Policy for Networks and Distributed Systems conferences.