An Adaptive Policy Based Framework for Network Services Management

Leonidas Lymberopoulos, Emil Lupu and Morris Sloman
Imperial College, Department of Computing, 180 Queen's Gate, SW7 2BZ, London, UK
{llymber, e.c.lupu, mss}@doc.ic.ac.uk

Abstract

This paper presents a framework for specifying policies for the management of network services. Although policy-based management has been the subject of intensifying research efforts, 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. Whilst 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 policy 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.

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.

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. However, 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 should take place either resulting from failures within the network or possibly adaptations to the changes in service requirements relating to the client application. The latter implies that the client application must be able to trigger changes to the service within the service provider.

A service provider may provide a sophisticated set of services, which are offered to a client organisation consisting of many different users. Not all users within a client organisation may need access to all the offered services. Authorization should be part of the SLA management system to specify which users are able to access particular services or functions within the services. This information is also dynamic in that it is 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 of 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 rest of the paper is organized as follows: in section 2 we outline the requirements of a policy-based system for service management. Section 3 briefly presents the Ponder language and section 4 analyses the use of policy adaptation and gives an outline implementation of an adaptive policy system. Section 5 presents how our adaptive policy framework can apply in a Differentiated Services environment. In section 6 we present and compare our approach with related work and we outline conclusions and directions for future work in section 7.

2. Service Management Issues

Consider a typical network of a large enterprise. Such a network 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's QoS guarantees to the applications, which are 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 perhaps 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 which can

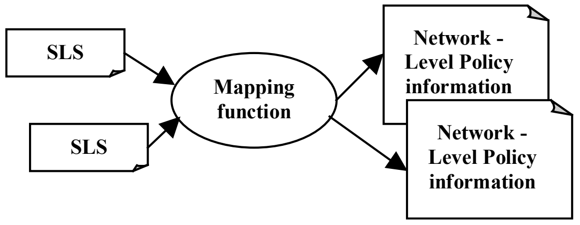
SLS
SLS
Mapping function
Network - Level Policy information
Network - Level Policy information

Figure 1: SLS to network-level policy mapping

automatically derive network policy information from service specific information. In this approach, the technical part of the SLA is formally specified as a set of Service Level Specifications (SLSs). A SLS is a set of parameters 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's QoS metrics (e.g. round-trip delay, throughput, packet loss probability, etc.). The management system will perform a mapping function from the SLSs that are specific to a negotiated SLA, in order to derive network policy information, as shown in Figure1.

An interesting variation of the above, 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.

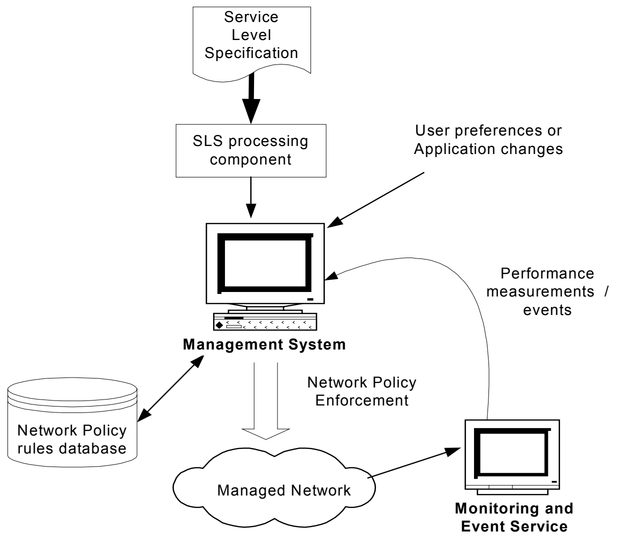
Service Level Specification
SLS processing component
User preferences or Application changes
Performance measurements / events
Management System
Network Policy Enforcement
Network Policy rules database
Managed Network
Monitoring and Event Service

Figure 2: Service management with a policy-based management system

However, 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. Typical cases where the management system should change the existing network configuration are outlined in Figure 2. These can be:

- New user or application requirements requiring changes to QoS. In the video application example, clients in site A may request more network resources for a running session, in order to receive better video quality from the video server located at Site B. Moreover, the application itself can change its QoS requirements at run-time. An example is the case of adaptive applications, which tailor their behavior according to the available network resources. This in turn means that network policies attributes must be changed at run-time to support the new user's/application's requirements.

- Performance measurements coming from a monitoring service may indicate performance degradation and thus may require changes in the service's 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's 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's packets.

In addition to the above, it is necessary to specify who is authorised 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 above for the management of services. In this, policy is specified with Ponder [5], a declarative, object-oriented language, developed at Imperial College 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,* 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 Figure 3.

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

Figure 3 Obligation Policy Syntax

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. Concurrency operators specify whether actions should be executed sequentially or in parallel and are used to separate actions in an obligation policy. The optional catch-clause specifies an exception that is executed if the execution of the policy actions fails for some reason. The above syntax is used for the 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.

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

Figure 4 Obligation 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 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 authorisation policies, event composition, composite policies and constraints can be found in [5] and a discussion on conflict detection and resolution in [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's 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 follows the format shown in Figure 5.

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

Figure 5 Generic format for network QoS policy

In this type of network-level QoS policy adaptation, the parameters of the policy action(s) are dynamically calculated from the event attributes. Thus, a re-configuration of the network devices can be changed dynamically by triggering the policy with a new event containing the new values.

4.2 Adaptation by dynamically selecting and enabling policies from a set of policies

In this approach, higher-level control policies receive events, which require system adaptation and decide which lower-level network policy must be enabled/disabled to adapt the configuration of the managed system. The advantage of using policies rather than a procedural language for selecting and enabling the appropriate network-level policies is that modifying the management strategy at this level can be achieved by dynamically changing the control policy. Furthermore, the same Ponder deployment framework can be used to distribute both high-level control policies and network DiffServ policies [7].

In the general case, a control policy is specified with the template obligation rule GenericControlPolicy, presented in Figure 6.

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

Figure 6 Specification of a generic control policy

4.3 Enforcement architecture

In the general case, the management functionality of the generic Policy Management Agent ControlPMA is specified with the obligation rule GenericControlPolicy, presented in Figure 6.

The ControlPMA must be able to:

a) Select, using a suitable algorithm, the most appropriate lower-level policy to actually implement the configuration adaptation, when the event AdaptationRequest occurs.
b) Calculate the selected policy's specific parameters.
c) Enable and trigger the selected policy with the derived parameters.

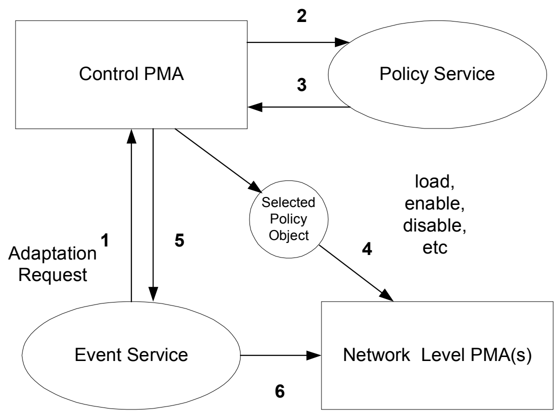

Figure 7 Enforcement architecture for policy adaptation

The enforcement architecture is presented in Figure 7.

1. The ControlPMA receives the event AdaptationRequest from the event service.
2. The ControlPMA invokes a selection algorithm to choose a suitable policy from the policy description database in the policy service.
3. The policy service replies with the selected policy object.
4. The enable() method is called on the selected policy object, which in turn calls the enable() method on the relevant PMAs. Enabling the policy means that policy enforcement objects within the PMAs register the obligation event with the event service, as described in [7]. At this point, the selected policy is activated on its PMAs. In addition, an "old" policy can be unloaded or disabled from the corresponding PMA's.
5. An event is generated with the policy's calculated parameters to trigger the policy.
6. The obligation event is sent by the event service to the registered Policy Enforcement Objects.

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

5.1 Per Domain Behavior policies

The IETF DiffServ working group has proposed in [9] the term Per Hop 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 a PHB or a set of PHB's.

Each PDB has measurable attributes that can be used to describe what happens to its packets as they enter and cross the DS 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 1 presents examples of QoS guarantees that PDB policies can offer to their associated traffic aggregates.

In our framework, PDB policies are specified as

Table 1. 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	<r1,…, rN>	Every day
PDB2	/Policies/Policy2	20	≤ 10	< 1	≤ 0.1	<r1,…, rM>	Working hours
…	…	…	…	…	…	…	

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 (a customer can be either human or an automated agent) is offered the externally observable PDB's QoS attributes. An example of a PDB Ponder policy rule is given below.

Example 1 Policy rule for providing a specific PDB

```
inst oblig   /Policies/PDBPolicy1 {
subject      /DiffServAgents/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, the PDB policy is implemented with the EF PHB. Upon the request PDB1_ConfigRequest, the network-level PMA DiffServAgent will invoke the applyEFPHB action to all routers that belong in the target domain. This way, all core routers within the target domain will guarantee low delay and low loss to the EF-marked packets. In addition, a minimum output rate (throughput) is guaranteed to the EF-marked packets, when these packets do not exceed the configured maximum input rate at the ingress router's interface. More details on the specification of network-level DiffServ policies can be found 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. In this implementation, element classes represent DiffServ functional elements (e.g. classifiers, filters, meters, droppers, etc). These classes have been modelled based on the DiffServ MIB [13] data model. A Java component is used to translate the DiffServ device-independent element classes to Linux "tc" [14] traffic control commands. This translation is done in the network-level PMA's engine. After the translation process, the network PMA opens a telnet session and downloads the "tc" commands to the policy's target Linux routers.

We also intend to provide an implementation using the SNMP as the management protocol. In this, DiffServ functional elements within the PMA are also modelled according to the DiffServ MIB, but SNMP will be used to communicate the functional elements to the SNMP agent at the target device. The latter will have the capability of translating the DiffServ MIB objects to the corresponding kernel configuration, using "netlink" sockets as the communication mechanism between the agent (user) and the kernel.

5.2 Service Management policies

Per Hop Domain Behavior policies are enforced by DiffServ enabled Network-level PMAs (Figure 7). This configures the QoS mechanisms of the managed devices within the DiffServ network. However, as we have already discussed in section 2, network services management requires additional functionality.

The required functionality that enables dynamic service management is provided in our framework by Policy Management Agents at the service level. A PMA responsible for performing service management tasks (ServiceManagementAgent) is a specific case of a control PMA in our generic policy adaptation architecture, which was explained in section 4.3. In the following, we will provide examples of service management policies for dynamic service management.

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

```
inst oblig   SLSMappingPolicy {
subject      ServiceManagementAgent;
on           SLS_Request  (SLS_parameters[]);
do           pdb_policy =
select_using_algorithmA(SLS_parameters[])->
pdb_policy_parameters[] =
    calculate ( pbd_policy, SLS_parameters[])->
pdb_policy.enable()->
    EventService.GenerateEvent (pdb_policy'sObligationEvent,
                                pdb_policy_parameters[]);}
```

The obligation rule SLSMappingPolicy instructs the ServiceManagementAgent to perform the SLS to PDB mapping function upon a SLS request. The SLS request conveys the SLS parameters to the agent. These SLS parameters and their semantics can be described in a formal model for SLS specification. An example for

Differentiated Services could be the Tequila's project [15] SLS specification.

[16] lists and presents the semantics of a set of basic SLS parameters when Differentiated Services is used as the underlying QoS mechanism. A "parser" component within the Ponder management toolkit is used to translate the Tequila's 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 service request. An example of a selection algorithm is outlined in [17], where the PDB is selected according to the triple <delay, loss, throughput>.

The advantage of implementing the mapping functionality in the ServiceManagementAgent's engine, is that different mapping strategies can be loaded into the agent at run-time and selected by different policy rules. If the mapping function is written in a procedural language, the implementation of a new mapping algorithm would require the recompilation of this component to include the new algorithm. In our framework, new mapping strategies can be loaded as new actions to the ServiceManagementAgent. A new policy, with action a new mapping function (e.g. selection_algorithmB) will instruct the agent to perform the new mapping function.

Policy to handle service's performance degradation

A number of different adaptation strategies could be adopted for handling service's performance degradation (notified by the monitoring service, see Figure 2) at run-time. 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, high jitter 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 the network service that the video client application described in section 2 receives, but omit the actual policies. In all the examples, we assume that the video application receives the EF network service.

- The monitoring system detects that the EF service's packet delays exceed a threshold so it generates a HighDelay event received by the ServiceManagement Agent. Corrective actions which may be performed include: a) Increase the minimum departure rate of the EF traffic to guarantee that the service's packets (especially large ones) will remain in the output queue for less time before being transmitted to the next hop. b) Notify the client application to choose a different state, which requires less bandwidth and hence decrease the incoming traffic rate at the ingress interface. This way, the EF aggregate will experience less delay.
- Jitter is not reduced by increasing the EF service rate, when the EF aggregate is constructed from a single microflow. On the contrary, when the EF aggregation degree increases, jitter increases rapidly with the number of microflows and with the EF load. Thus, there are two possible corrective actions for a HighJitter event: a) Decrease the number of microflows, by degrading other EF traffic to receive a lower service. b) Reduce the EF load, by reducing resources assigned to the client application.
- The action for a HighPacketLoss event would be to increase the maximum arrival rate of the incoming EF traffic. 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.

Policy to support changes in routing or link failures

A PDB is usually associated with a path of routers within the DiffServ domain (e.g. when using DiffServ over MPLS.). When a link fails or routing changes for a specific flow, the corresponding PDB may not be appropriate for the routers in the new path, or it may no longer be suitable. A new PDB must be selected, that satisfies the service's QoS expectations and that can be applicable to the new path. This can be implemented using the following policy inside the ServiceManagementAgent:

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

```
inst oblig   PolicyUponRoutingChangesOrLinkFailures   {
   subject   ServiceManagementAgent;
   on    routeIsChanged (newPath);
   do    pdb = select_using_algorithmA(SLS_params[],
                                        newPath) ->
         /* A PDB suitable for the new path must be selected
         to cater for the service */
         pdb.enable() ->
         pdb_params[] = calculate(SLS_params[]) ->
         EventService.GenerateEvent (
                   pdb'sObligationEvent, 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 will serve the service's packets has changed. As in example 1, different algorithms can be used for the selection of the new PDB.

Policy to reflect changes in application or user requirements

The user/application may have the ability to demand different QoS guarantees at run-time. This means that the user or the application itself may change SLS parameters at run-time by. As a consequence, network policies' attributes must be changed to support the new user's/application's requirements. Below follows a policy example, which enables the ServiceManagementAgent to provide this type of service adaptation.

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

```
inst oblig      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_policy'sObligationEvent, new_pdb_policy_parameters[]);}
```

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 [15] 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 succesful. 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 Authorisation policies

As we have discussed in section 2 of this paper, authorisation 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 authorisation policies and control requests related to the service.

In the following example, 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

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

It 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

```
inst auth+ GoldService_BandwidthControlPolicy   {
subject   /Users/SiteA_users/Admins +
          /Users/SiteB_users/Admins;
target    ServiceAccessPoint_Agent;
action    allocateBandwidthToGoldService(  bandwidth );
```

6. 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 [18] is defining a framework for managing QoS within networks, [19]. 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., [20], [21]. 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 configurations need to be imposed manually by the administrator through the management console.

In [22], a policy-based management system is proposed for managing Service Level Agreements within DiffServ networks. They use a tabular specification (described in detail in [23]) 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 configuration process, initiated by the administrator. Configuration can not dynamically be changed at run-time to reflect 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 in [17]. In their architecture, 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 intra-domain service, which is offered by an implemented PDB within the DiffServ network. The second is the policy-based control part. This controls the SLS mapping and the admission control processes. Network policy is used as the device configuration mechanism. However, this work does 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.

[24] 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's 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. The latter, upon reception of a violation notification, initiates corrective measures to remedy the causes of the violation. 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 is proposed in [25]. This allows delegation of the service management task from the service provider to the customer. A CSM module is the basic block of the proposed management system. Customers can adjust SLS parameters through a parameter setting function block within the CSM module. A SLS mapping function is implemented within the CSM module, 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 in [26] 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 behaviour. 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. Furthermore, 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.

The system proposed in [27] for the management of QoS in Multi-Protocol Label Switching (MPLS) networks, also follows the IETF Policy working group approach. They have extended the Common Information Model (CIM) policy model with MPLS specific classes. This system has the same limitations as the IETF framework. In [28], IETF's Policy Core Information Model (PCIM) is extended, to provide support for goal specification. Service-level goals can be specified to enforce QoS on a per-user, per-application basis. Monitored data is used to evaluate whether the specified goals are satisfied. These service-level goals can be expressed in our framework as higher-level obligation policy rules.

[29] presents an architecture for the management of a network offering active services. In their architecture, a bacterial algorithm forms the basis for the adaptation performed by autonomous controllers. These controllers 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. [30] 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.

[31] presents a policy-driven framework for QoS management of multimedia applications. They specify policy at the application layer using the Ponder language, although they rely on violation of constraints to trigger policy rules instead of events. Their QoS policy only provides the QoSHostManager component with a notification message; the corrective actions which are enforced upon QoS violation are described in other types of rules. No formal specification is discussed for these rules, although they could be specified with Ponder's obligation policies as well. Furthermore, they use the term "adaptation" to refer only to the actions, which are taken when a QoS violation occurs. We can support the type of adaptation provided in [31], but we consider our approach to be more general than theirs.

[32] presents a QoS Architecture transport system for a multicast, multimedia networking environment. It offers a QoS configurable API at the transport layer, which enables applications to have control over QoS. 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, e.g. [33, 34]. Most of this work provides adaptation by hard coded QoS management and monitoring in middleware systems for supporting multimedia applications.

7. 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 authorisation policy. Policy is specified in the Ponder policy language.

Network policy rules configure the QoS mechanisms of the devices within the target domain. We presented how Per Hop Domain Behavior policies are specified and deployed automatically to configure Linux DiffServ routers. For the enforcement part, we are currently using the DiffServ modules of the implementation described in [12]. We intend to provide an additional enforcement implementation using the SNMP as the management protocol. 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. An enforcement architecture for our ideas on policy adaptation was also presented.

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.

Finally, in order to protect network services from unauthorised usage, we provide the administrator the ability to specify with service authorisation 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.

References

[1] Braden, R., Clark, D. & Shenker, D., Integrated Services in the Internet Architecture: an Overview, RFC 1633, June 1994.

[2] Braden, R., Zhang, L., Berson, S., Herzog, S. & Jamin, S., ReSerVation Protocol (RSVP) Version 1 Functional Specification, RFC 2205, September 1997.

[3] Carlson, M., Weiss, W., Blake, S., Wang, Z., Black, D. & Davies, E., An Architecture for Differentiated Services, RFC 2475, December1998.

[4] Jacobson, V., Nichols, K. & Poduri, K., An Expedited Forwarding PHB, RFC2598, September 1999.

[5] Damianou, N., Dulay, N., Lupu, E. & Sloman, M. The Ponder Policy Specification Language. Proc. Policy 2001: International Workshop on Policies for Distributed Systems and Networks, Bristol, UK, 29-31 Jan. 2001, Springer-Verlag LNCS 1995, pp. 18-39.

[6] Sloman, M. & Twidle, K., Domains: A framework for Structuring Management Policy. Chapter 16 in Networks and Distributed Systems Management (Sloman, 1994ed), 1994a: pp. 433-453.

[7] Dulay, N., Lupu, E., Sloman, M. & Damianou, N. A Policy Deployment Model for the Ponder Language. Proc. IM 2001: 2001 IEEE/IFIP International Symposium on Intergrated Network Management, Seattle, USA, 14-18 May 2001, pp. 529-544.

[8] Lupu, E. & Sloman, M., Conflicts in Policy-Based Distributed Systems Management. IEEE Transactions on Software Engineering, Special Issue on Inconsistency Management, 25(6):852-869, Nov./Dec. 1999.

[9] K. Nichols, K. & Carpenter, B., Definition of Differentiated Services Per Domain Behaviors and Rules for their Specification, RFC 3086, April 2001

[10] Lymberopoulos, L., Lupu, E. & Sloman, M. An An Adaptive Policy Based Management Framework for Differentiated Services Networks. To appear in Proc. Policy 2002: IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, Monterey, CA, USA, 5-7 Jun. 2002.

[11] Damianou, N., Dulay, N., Lupu, E., Sloman, M., Tonouchi, T. Tools for Domain-based Policy Management of Distributed Systems. Proc. NOMS 2002: 8th Network Operations and Management Symposium, Florence, Italy, 15-19 Apr. 2002.

[12] Martinez, M. et al. Using the Script MIB for Policy-based Configuration Management. Proc. NOMS 2002: 8th Network Operations and Management Symposium, Florence, Italy, 15-19 Apr. 2002.

[13] Baker, F., Smith, A. & Chan, K., Differentiated Services MIB, Internet Draft, draft-ietf-diffserv-mib-09.txt, March 2001.

[14] Linux Advanced Routing & Traffic Control, http://lartc.org.

[15] Tequila project, http://www.ist-tequila.org.

[16] Goderis, D. et al. Service Level Specification Semantics, Parameters and negotiation requirements, draft-tequila-sls-01.txt, June 2001.

[17] Prieto, A. & Brunner, M. SLS to DiffServ configuration mappings, Proc. DSOM 2001: 12th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, Nancy, France, 15-17 Oct. 2001.

[18] Internet Engineering Task Force, Policy Working Group, http://www.ietf.org/html.charters/policy-charter.html

[19] Snir, Y., Ramberg, Y., Strassner, J. & Cohen, R., Policy Framework QoS Information Model, Internet Draft, draft-ietf-policy-qos-info-model-03.txt, April 2001.

[20] Cisco COPS QoS Policy Manager product documentation, htttp://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/qos/qpm2_1/index.htm

[21] Allot Communications NetPolicy Policy Based Management System product documentation, http://www.allot.com/html/products_netpolicy.shtm

[22] Verma, D., Beigi, M. & Jennings, R. Policy Based SLA Management in Enterprise Networks. Proc. Policy 2001: International Workshop on Policies for Distributed Systems and Networks, Bristol, UK, 29-31 Jan. 2001, Springer-Verlag LNCS 1995, pp. 137-152.

[23] Verma, D. (2001). Policy-Based Networking, Architecture and Algorithms. New Riders Publishing.

[24] Keller, A., Kar, G., Ludwig, H., Dan, A. & Hellerstein, J. Managing Dynamic Services: A Contract-based Approach to a Conceptual Architecture. Proc. NOMS 2002: 8th Network Operations and Management Symposium, Florence, Italy, 15-19 Apr. 2002.

[25] Sprenkels, R. Pras A., et al. A Customer Service Management Architecture for the Internet. Proc. DSOM 2000: 11th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, Texas, USA, 4-6 Dec. 2000.

[26] Yoshihara. K., Isomura M. & Horiuchi, H. Distributed Policy-based Management Enabling Policy Adaptation on Monitoring using Active Network Technology. Proc. DSOM 2001: 12th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, Nancy, France, 15-17 Oct. 2001.

[27] Brunner, M. & Quittek, J. MPLS Management using Policies. Proc. IM 2001: 2001 IEEE/IFIP International Symposium on Intergrated Network Management, Seattle, USA, 14-18 May 2001, pp. 515-528.

[28] Bearden, M., Garg, S. & Lee, W. Integrating Goal Specification in Policy-Based Management Proc. Policy 2001: International Workshop on Policies for Distributed Systems and Networks, Bristol, UK, 29-31 Jan. 2001, Springer-Verlag LNCS 1995, pp. 153-170.

[29] Marshall, I., Gharib, H., Hardwicke, H. &.Roadknight C. A novel architecture for active service management. Proc. IM 2001: 2001 IEEE/IFIP International Symposium on Intergrated Network Management, Seattle, USA, May 2001, pp. 795-810.

[30] I.W.Marshall and C.M.Roadknight "Provision of quality of service for active services" Computer Networks, Vol. 36, No. 1, June 2001.

[31] Lutfiyya, H., Molenkamp, G., Katchabaw, M. & Bauer, M. Issues in Managing Soft QoS Requirements in Distributed Systems Using a Policy-Based Framework. Proc. Policy 2001: International Workshop on Policies for Distributed Systems and Networks, Bristol, UK, 29-31 Jan. 2001, Springer-Verlag LNCS 1995, pp. 185-201.

[32] Campbell, A.T., "A Quality of Service Architecture", PhD Thesis, Lancaster University , UK, January 1996.

[33] Gordon, G. et al. Adaptive Middleware for Mobile Multimedia Applications. Proc. NOSSDAV '97: Network and Operating System Support for Digital Audio and Video , St Louis, USA 1997.

[34] Wang, N. et al. “Adaptive and Reflective Middleware for QoS-Enabled CCM Applications”, Distributed Systems Online (www.computer.org/dsonline)