An Adaptive Policy Based Management Framework for Differentiated Services Networks

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 Differentiated Services (DiffServ) networks. 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 DiffServ networks according to changes in requirements. Policy adaptation includes both dynamically changing policy parameters and reconfiguring the policy objects.

1. Introduction

Network applications commonly require more bandwidth than networks can provide, leading to congestion and variable performance in terms of throughput, latency variations (jitter), propagation delay, etc. Traditional IP networks provide a "best-effort" service which treats all applications equally when competing for network resources, irrespective of how critical they are. To address this issue, research and industrial efforts have focused on the development of Quality of Service (QoS) enabled networks which provide mechanisms to allow network applications to request and receive predictable performance levels.

Two approaches have been proposed for providing QoS in 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. Each flow is identified by the destination IP address, the transport protocol and the port number used by the application. This approach has significant scalability problems as routers must maintain a lot of information about the application flows and their reservations and must process a large number of messages for each reserved flow.

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. Only edge routers in the DiffServ architecture need to perform the classification of traffic flows into classes of service. The core routers queue and schedule packets according to the value of the ToS field. This ensures that DiffServ has significantly better scalability characteristics and is therefore becoming more popular with network providers.

The IETF DiffServ working group [3] has defined an architecture for DiffServ (RFC 2475) and standard Per Hop Behaviors (RFCs 2474, 2597 and 2598). In addition, a model for representing DiffServ routers (DSMODEL) is also proposed [4]. Ongoing work includes the definition of a DiffServ Management Information Base (MIB) and Policy Information Base (PIB) to provide standards for managing DiffServ mechanisms using existing management protocols, such as the Simple Network Management Protocol (SNMP) and the Common Policy Service Protocol (COPS). Furthermore, various research groups are investigating the automation and the simplification of the management process using rules that can dynamically change the behaviour of a DiffServ-enabled network. This approach is known as *policy-based management*. We have identified the following requirements for policy-based management of DiffServ networks:

- The management system must have the flexibility and necessary abstractions to manage a variety of device types, with different capabilities and limitations, from different vendors. The system architecture should be sufficiently flexible to allow adding new device types with minimal updates and recording of existing components.

- To cater for large-scale networks, the management system must be able to apply policy rules on sets of devices rather than individual ones. When adding a new device to a set, relevant policies should be automatically deployed and enforced on it.
- The management system must be able to adapt to events such as failures or QoS violations within the system. In addition to adapting the behaviour of managed devices, the management system should also adapt its own behaviour, if necessary.
- Managed entities can be either applications or network elements. Thus, there is the need to specify policy at both the application and the network level. Policies at one level may need to exchange information, notifications or trigger the execution of policy actions in another level.

This paper presents a flexible, expressive and extensible framework to cover the wide range of requirements identified above for the management of Differentiated Services. Ponder [5], a declarative, object-oriented language for specifying security and management policies in distributed systems, is used as the policy specification language.

The rest of the paper is organized as follows: section 2 explains the IETF DiffServ architecture and device representation mode. Section 3, presents our approach for the management of the functional components of DiffServ-enabled devices and illustrates the enforcement architecture for the proposed policy-driven system. Section 4 analyses the use of policy adaptation and gives an outline implementation of an adaptive policy system for DiffServ. In section 5 we briefly present and compare our approach with related work and we outline conclusions and directions for future work in section 6.

2. DiffServ Architecture

According to the DiffServ Architecture [6], traffic entering a network is classified and possibly conditioned at the network boundary, and assigned to different behaviour aggregates identified by DiffServ CodePoints (DSCPs). A DSCP is encoded in the most significant 6 bits of the ToS byte contained in the IP header of both Ipv4 and Ipv6. In the core network, routers forward incoming packets to the next hop according to the Per-Hop Behaviour (PHB) associated with the packet's DSCP. The IETF DiffServ working group has defined a number of standard PHBs – Class-Selector, Assured Forwarding (AF), Expedited Forwarding (EF), and Default [7], [8], [9]. A DiffServ-enabled node typically uses several components [4], to implement a PHB.

The DSMODEL [4] represents the various functional building blocks defined in the DiffServ Architecture to implement the PHB. It includes abstract definitions for Traffic Classification Elements, Metering Functions, Actions of Marking, Absolute Dropping, Counting and Multiplexing, and Queuing elements. The latter include capabilities for algorithmic dropping and scheduling. Combinations of the above functional datapath elements form higher-level blocks known as Traffic Conditioning Blocks (TCBs). Figure 1 shows a TCB comprising six components from the DiffServ architecture.

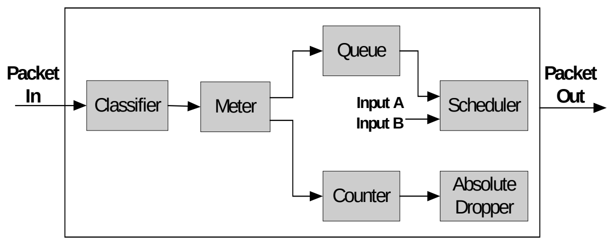

Figure 1 Example of a Traffic Conditioning Block

According to [4], a DiffServ-enabled device consists of a interconnected set of TCBs, a routing component and a configuration and management module as shown in Figure 2.

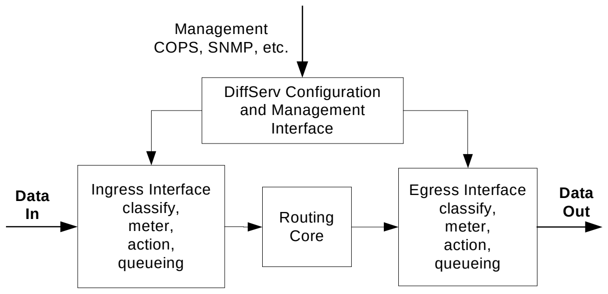
Management
COPS, SNMP, etc.
DiffServ Configuration and Management Interface
Data In
Ingress Interface classify, meter, action, queueing
Routing Core
Egress Interface classify, meter, action, queueing
Data Out

Figure 2 Model of a DiffServ-enabled device

The management of the individual functional elements and TCB's in a DiffServ device, is realized through the Configuration and Management Interface via one or more management protocols, such as SNMP or COPS, or through other configuration tools such as serial terminal or telnet consoles. Our work aims at providing a Policy-based DiffServ Management system using policy rules that govern the behaviour of each node within a DiffServ network. Our policy system communicates with each managed node through the configuration and management interface that the node provides.

3. Using Ponder for Management of DiffServ Networks

Ponder [5] is an object-oriented, declarative language developed at Imperial College 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 [10]. 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 [12], an obligation policy enforcement object is implemented as a Java program downloaded to a PMA. The PMA registers with the monitoring service to receive the relevant events which will trigger the policies it holds. Policy actions are executed on the target managed QoS-enabled devices when the triggering event is received by the PMA. Events may pass parameters to the PMA and the policy actions may also have parameters, which typically correspond to the QoS configuration parameters (managed object attributes) of a target device.

We have given a very brief overview of Ponder. More details on event composition, composite policies and constraints can be found in [5] and a discussion on conflict detection and resolution in [13]. In the following sections we describe how Ponder policies can be used to manage a DiffServ enabled device.

3.1. Policy Rules for Configuring DiffServ Functional Elements

Although in this paper we use the abstractions of DiffServ provided in [4], the implementation will use the more detailed formal model proposed by the IETF Policy working group for describing the QoS datapath elements for DiffServ [11].

In our framework, functional elements of a DiffServ device are represented as objects, which belong to the following classes (derived from [4]): Class Classifier, Class Meter, Class Marker, Class AbsoluteDropper, Class Counter, Class Multiplexer, Class Queue, Class AlgorithmicDropper, and Class Scheduler. These fundamental classes can be extended (as shown in [11]), to define more specific functional elements. For example, the Class Meter may have five subclasses: Average Rate Meter, Exponential Weighted Moving Average (EWMA) Meter, Two-Parameter Token Bucket Meter, Multi-Stage Token Bucket Meter, and Null Meter. Furthermore, each fundamental class contains a field that points to the next functional element along the datapath, as explained in section 2.

This approach provides the administrator the flexibility to define generic policy types, which can apply to any type of DiffServ device, and can be instantiated with device specific parameters. For example, the following policy type can be written to insert a Meter (i.e., any subclass of MeterT) on a router.

Example 1 Policy type that configures any Meter type in any DiffServ device

```
type oblig    insertMeter (target router, MeterT meter) {
  subject     DiffServManager;
  on          addMeterRequest();
  do          router.addMeter(meter);}
```

The parameter meter used in this policy type is an object of class MeterT. The Policy Management Agent can detect the actual subclass, which may correspond to one of those identified above, and if necessary, take specific actions for this particular subclass. The same insertMeter type can be instantiated multiple times in order to create specific rules, which apply to specific devices within the DiffServ domain. For example, the administrator may want to apply the AverageRateMeterA on the edge routerA and the TwoParameterTokenBucketMeterB on the core routerB. Two policy instances should be created for that purpose, as shown in Example 2.

Example 2 Instantiation of the insertMeter policy type for different devices

```
inst  Configuration1 =
      insertMeter(/Routers/EdgeRouters/RouterA,
                  AverageRateMeterA);
inst  Configuration2 =
      insertMeter(/Routers/CoreRouters/RouterB,
                  TwoParameterTokenBucketMeterB);
```

3.2. Policy Rules for the Management of a Set of Devices

Since subjects and targets are explicitly specified in the Ponder policies in terms of domains, the same policy can be easily applied to a large set of devices. For example, an administrator may want to impose the same classification rules on all edge routers in a domain, in order to provide a common set of DiffServ codepoints. The solution provided by our approach is a policy rule whose scope is not a particular device, but all devices within the defined domain.

Example 3 Policy that adds the same classifier entry to all devices within DomainA of edge routers

```
inst oblig    insertClassifierToDomainA {
subject       DiffServManager;
target        r = /Routers/EdgeRouters/DomainA;
on            AddClassifierEntryRequest(classifierEntry);
do            r.addEntryToClassifier(classifierEntry);}
```

When an AddClassifierEntryRequest event occurs (this event is usually triggered through the management console), this rule will insert the same entry in the classifier elements of all routers that belong to the domain /Routers/EdgeRouters/DomainA. Moreover, if a new device is later added to that domain, the corresponding Policy Management Agent DiffServManager will be notified and will apply the policy to the newly added device automatically.

3.3. Policy Rules for Implementing DiffServ Per Hop Behaviours

Per-Hop Behaviour specifies the treatment packets should receive on a DiffServ node, as they are traversing a sequence of functional datapath elements. The implementation of both standard and vendor specific PHBs can be provided by using more complex policy rules, whose specification can be easily altered to provide varying PHBs.

In Example 4, we present a policy which will configure a set of routers within the DS domain to implement the required EF PHB. In this example, traffic above the configured maximum input rate is counted and discarded. Figure 1 shows the functional elements to be configured on each device to provide the requested PHB.

Example 4 Policy rule for providing EF PHB.

```
inst oblig     EFConfigurationPolicy {
subject   DiffServManager;
target    r = /DomainA/Routers/CoreRouters;
on EFConfigRequest(DS,max_input_rate,min_output_rate);
do          /* DS: The Diffserv codepoint for EF: 101110 */
      r.addEntryToClassifier(DS)->
      meter = newAverageRateMeter(max_input_rate)->
      r.addMeter(meter)->r.addQueue(queue)->
      schedulerEntry=newSchEntry(WRR,min_output_rate)->
      r.addEntryToScheduler(schedulerEntry)->
      r.addCounter(counter)-> r.addDropper(dropper);
when      max_input_rate <= min_output_rate;
              /* Property that EF traffic must satisfy */ }

       This example assumes that a Weighted Round
       Robin scheduling algorithm is used. The weight for
       this class of traffic must be calculated from the
       parameter min_output_rate as : weight =
       min_output_rate / total_output_rate, where
       total_output_rate is the egress interface's
       bandwitdh.
```

The EFConfigurationPolicy is triggered by the administrator's request EFConfigRequest and will configure all routers within the target domain.

Note that it is possible to define more complex policy actions as scripts within the PMA, using a suitable scripting language, which may include conditional statements, atomic transactions, etc. For example, all the

actions in Example 4 could be combined in a single script action:
applyEFPHB(DS, max_input_rate, min_output_rate).

In our framework, a policy rule is always active, unless it has explicitly been disabled by the administrator. PMAs can receive events (other that configuration requests) relating to changes in the system and then enforce the relevant policies. This means that the policy rule EFConfigurationPolicy can also be triggered in other circumstances. For example, when the monitoring service detects a high drop rate of the EF traffic class, it can raise an EFConfigRequest with a higher maximum input rate value. This dynamic triggering of policies is one of the mechanisms used for adaptive management in our framework, as explained in section 4.

3.4. Enforcement Architecture

The enforcement of rules that can be defined in our framework require the execution of low-level actions, e.g., addMeter(meter). These provide a uniform way to represent management and configuration actions and abstract the protocols used for communication between a management agent and the managed device. The proposed architecture is shown in Figure 5.

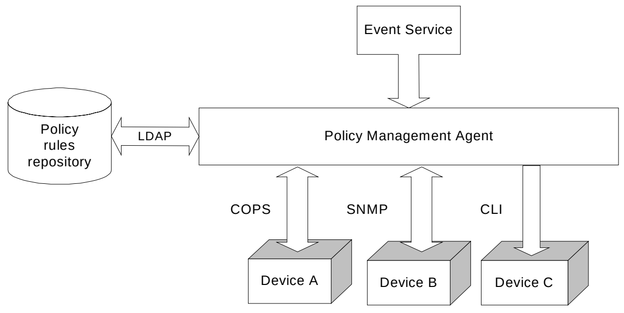

Figure 5 Enforcement Architecture

Multiple PMAs can co-exist within the system. The management responsibility can be divided amongst them according to several criteria: each PMA is responsible for the enforcement of a different set of policies or responsible for managing a different set of devices. The management actions specified in the policies may need to be implemented by lower-level device specific commands which are downloaded to the device using:

- Command Line Interface (CLI): the policy actions are translated into the corresponding CLI commands, which are downloaded to the device using a script that performs a telnet session between the PMA and the device.
- SNMP: the SMIv2 MIB for Differentiated Services [14] supports the abstractions defined in [4]. Thus, once the functional elements are created by the PMA, SNMP can be used to configure the corresponding entries into the device's MIB.
- COPS-PR: this is a protocol for policy provisioning [15], which can be used with the Differentiated Services PIB [16], which also follows the abstractions provided by [4]. The PMA creates Provisioning Instances (PRI's) and downloads the policy information to the device using COPS-PR.

The architecture presented in this paper can use any of these low-level protocols for communicating the policy information between the PMA and the managed devices. In the future, it is possible that more sophisticated network devices will be capable of downloading and directly interpreting Ponder policies from a repository. That is, the network device will incorporate a PMA and receive policy triggering events.

4. Policy Adaptation

When applying policies to DiffServ elements, the policy actions are those provided by the management interface of the datapath elements determining the PHB. Thus, the "level of abstraction" of the policies is determined by the available implementation. However, changes in the system such as failures or QoS violations may require adaptation of existing policies to new circumstances. 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 following format:

```
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. For example, in the EFConfigurationPolic*y* rule from Example 4, which provides the EF PHB, the max_input_rate and min_output_rate configuration parameters are given by the EFConfigRequest event. Thus, a re-configuration of the datapath elements can be changed dynamically by triggering the policy with a new event containing the new values. A re-configuration may occur in either of the following cases:

- upon the admission of a new EF traffic flow. The new portion of EF traffic will be aggregated with the existing EF traffic via DiffServ's classification and marking processes. The monitoring system detects the new traffic flow and notifies the QoS management PMA. It calculates new values for maximum arrival and the minimum departure rates as the sum of the existing allocation for EF, plus the requested bandwidth for the admitted new flow. The PMA then triggers the EFConfigRequest event with the new parameters and the event triggers the re-evaluation of the EFConfigurationPolicy rule.
- to provide time-based bandwidth variation for the EF traffic class. Typically, gold customers in an ISP require high network usage during peak business hours. In this case, the rule EFConfigurationPolicy of the Example 4 will be triggered by a time event, as shown in Example 5.

Example 5 Policies for time-based bandwidth variation

```
type oblig  EFConfigurationPolicy (String timeOfDay,
            float max_input_rate, float min_output_rate) {
subject     DiffServManager;
target      r = /Routers/CoreRouters/RelevantRouters;
on          timer.At(timeOfDay);
do          r.applyEFPHB (DS, max_input_rate,
                          min_output_rate);

when        time.dayOfWeek() != "sat" ||
            time.dayOfWeek() !="sun"; }

inst oblig timePolicy1 = EFConfigurationPolicy(
                              "08:00:00",20,20);
inst oblig timePolicy2 = EFConfigurationPolicy(
                              "17:00:00",10,10);
```

In the above, min_output_rate = max_input_rate, but in practice, min_output_rate = k * max_input_rate, where k>=1.03. The rule timePolicy1 sets the bandwidth for the EF traffic at 20Mbps every weekday at 8am. The rule timePolicy2 sets the EF's allocated bandwidth at 10MBps again every weekday (during the weekend, timePolicy1 or timePolicy2 do not have to be re-enforced)

- to support adaptive applications, which adapt their behaviour under changing network conditions, by increasing/decreasing compression ratio, changing coding algorithm etc. In our framework network QoS policy parameters can thus be adapted to the application's requirements.

Consider an adaptive multimedia application. The QoS metrics for a typical video flow would be the spatial resolution (number of pixels/frame) and the temporal resolution (number of frames/sec). Using a mapping technique (for example as defined in [17]), these application-level QoS metrics are translated into network-level QoS metrics such as network bandwidth, which the network must guarantee. However, when the available bandwidth increases, the application may want to transmit a better quality of video. Assume the application receives notification of newly available bandwidth newBW from the network, calculates new requirements for adjusted application behaviour via a call calculate(newBw) and notifies the network of its new requirements by generating an EFConfigRequest event with requiredBW as a parameter, (where requiredBw ≤ newBw). We assume that the application's packets are EF treated. This can be realised using an obligation policy at the application layer as shown in Example 6. The low-level EFConfigurationPolicy in the NetworkLevelPMA is triggered by the EFConfigRequest event with requiredBw as a parameter and resets the relevant network devices.

Example 6 Policy for providing adaptation to the video application

```
inst oblig    AdaptationPolicyForVideoApplication {
subject       ApplicationLevelQoSManager;
on       AvailableBandwidth (newBw);
do       requiredBw = calculate (newBw) ->
         EventService.GenerateEvent(EFConfigRequest,
                                    requiredBw); }
```

A number of different adaptation strategies could be adopted for the video application, and so there may be a need to dynamically change these strategies by replacing AdaptationPolicyForVideoApplication with a new version or by enabling/disabling different versions of the policy. Policies provide a more flexible means of implementing this type of application-level adaptation than scripts or special purpose code. Note that other events indicating high delay, high jitter or high packet loss could trigger policies in the ApplicationLevelQoS manager. In the following examples, we indicate adaptation strategies which could be implemented by Ponder policies similar to Example 6, but omit the actual policies.

- The monitoring system detects that video packet delays exceed a threshold so it generates a HighDelay event received by the application manager. Corrective actions which may be performed include: a) Increase the minimum departure rate of the EF traffic to guarantee that the video packets (especially large ones) will remain in the output queue for less time before being transmitted to the next hop. This is verified from the measurements provided in [18]. b) Choose a different application state in which the video quality is decreased but which requires less bandwidth and hence less delay.
- Measurements provided by [18] and from simulations in RFC 2598 show that 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 a lower PHB. b) Reduce the EF load, by choosing a lower video quality.
- 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.

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 DiffServ 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 [12].

In the general case, a control policy is specified with the template obligation rule GenericControlPolicy, presented in Figure 6. In the following sub-section, we will present specific examples of control policies, which follow this template.

```
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

4.2.1. Usage scenarios. In this section, we will consider a set of Per-Domain-Behavior (PDB) DiffServ policies from which one must be selected and enabled, in response to an event requiring system re-configuration. As defined in [19], a PDB describes the edge-to-edge behaviour across a DiffServ domain. A particular PHB (or a set of PHBs) and traffic conditioning requirements are associated with each PDB. Using PDBs, it is possible to guarantee that a class of service will receive guaranteed QoS characteristics when traversing a DiffServ domain.

In our proposed framework, the QoS guarantees for PDB policies are specified in a policy database, which is part of the Policy service. Table 1 shows an example policy description database.

Typical network-level QoS PDB policies include an EFConfigurationPolicy for implementing EF behaviour, an AFConfigurationPolicy for implementing Assured Forwarding (AF) services, etc. In the following, we will present usage scenarios of our framework.

- *Selection of a PDB policy to provide the required QoS for a new service to be supported by the DiffServ network.* The QoS needs of the service are stated in its associated Service Level Specification (SLS). A detailed description of a SLS can be found in [20]. The Policy Based Management system must select the PDB policy, which most closely matches the SLS. A framework for mapping SLSs to PDBs can be found in [21]. For example, if the SLS states that the application must be guaranteed the lowest possible values of delay and jitter, then the PDB that provides the lowest delay and jitter must be selected from Table 1. This operation will be

Table 1. PDB policies and their QoS characteristics (part of the Policy Service)

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
…	…	…	…	…	…	…	

performed by a Control PMA, using the policy as in Example 7.

Example 7 Policy to configure PDB for a new service

```
inst oblig    ServiceConfigurationPolicy {
subject       PDBSelectorAgent;
on        newServiceRequest (SLS_params[]);
do        pdb = selectPDB (SLS_params[]) ->
/* pdb is the reference to the policy object representing the
PDB policy*/
          pdb.enable()->
          pdb_params[]= calculate(SLS_params[]) ->
          EventService.GenerateEvent (pdb'sObligationEvent,
          pdb_params[]);}
```

• *Selection of a new PDB policy in response to link failures or routing changes.* 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 QoS expectations for the flow and that can be applicable to the new path. This can be implemented using the following control policy:

Example 8 Policy for configuring DiffServ upon link failures or routing changes

```
inst oblig ConfigPolicyUponRoutingChangesOrLinkFailures
    {
    subject   PDBSelectorAgent;
    on    routeIsChanged (newPath);
    do    pdb = selectPDB (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[]); }
```

• *Selecting a new PDB policy in response to performance degradation.* It is possible that a PDB cannot deliver its QoS promises, due to congestion on links or due to a high degree of flow aggregation. This could be a result of over provisioning if too many PDBs try to guarantee QoS characteristics. In this case, a flow or a traffic class will experience performance degradation when being served by a "violated" PDB. A solution is to dynamically select a new PDB that will more closely match the service(s) QoS requirements.

• *Time-based selection of PDBs.* This usage scenario is similar to the one presented in section 4.1. A set of PDBs is chosen taking into account the Time Validity field of Table 1. Time Events activate the relevant policies as shown in section 4.1.

4.2.2. Enforcement architecture. In the previous section, we presented scenarios where a policy is selected and enabled by the higher-level Policy Management Agent PDBSelector. In the general case, this 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.

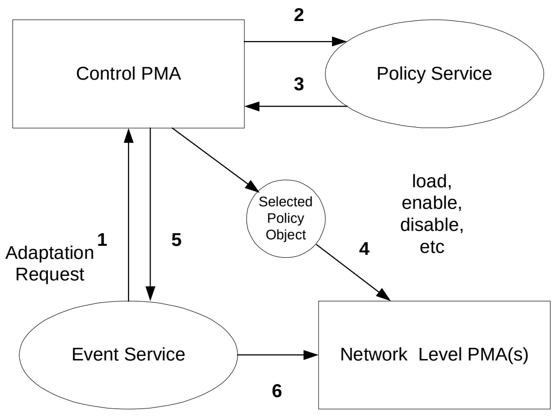

Figure 7 Enforcement architecture for policy adaptation

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 [12]. 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.

The pseudocode for the selectPolicy() method within the ControlPMA is given below:

```
PolicyObjectReference selectPolicy (parameters[]) {
/* This method returns the reference of the policy object. In
the current implementation, it is an RMI reference.*/
selectedPolicy = PolicyService.selectPolicy(parameters[]);
/* selectPolicy is the specific implementation of an
algorithm for policy selection within the Policy Service. The
user could provide his/hers selection algorithms*/
return ( SelectedPolicy); }
```

5. Related Work

Various research groups have presented ideas regarding QoS adaptation in which they have identified the need for specifying policy in order to adapt system configuration in response to network changes, but no acknowledged solutions have emerged. Other research groups are working on policy specification and enforcement. Out work aims at bringing together these areas, by showing how to use the flexibility of a policy based management framework for the adaptive management of DiffServ networks.

The IETF Policy working group [22] is defining a framework for managing QoS within networks, [23]. 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., [24], [25]. 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 [26], a policy-based management system is proposed for managing Service Level Agreements within DiffServ networks. They use a tabular specification (described in detail in [27]) 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. 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.

The framework proposed in [28] 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 [29] 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 [30], 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.

In [20], an architectural framework is proposed for providing QoS in DiffServ networks. Although they identify the need for a policy-driven management plane, no concrete proposals for its architecture have been presented yet.

[31] 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. [32] 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.

[33] 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 which corresponds to the actions in a Ponder obligation policy. Section 4.1 describes how we would support the type of adaptation provided in [33], but we consider our approach to be more general than theirs.

[34] 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. [35, 36]. Most of this work provides adaptation by hard coded QoS management and monitoring in middleware systems for supporting multimedia applications.

6. Conclusions and Future Work

In this paper we have presented a case study of applying the Ponder policy framework to the adaptive management of Differentiated Services networks. The use of parameterised policy types gives applicability to a wide range of DiffServ-enabled devices while the policy system communicates with each managed node through the configuration and management interface that the node provides. Our approach provides the administrator with the flexibility to define rules at both the network and the application level for managing DiffServ networks and network-aware applications. More complex policy rules, such as rules for establishing standard (i.e. AF, EF) or customized Per-Hop Behaviours in a DiffServ domain can be expressed in our framework. They can be communicated to the managed devices by existing protocols, such as SNMP and COPS-PR, or through the CLI. Network policies can be dynamically triggered by events, in order to automatically change the configuration of the managed objects under changing circumstances. The dynamic configuration of policies forms the basis of the adaptive management our framework can provide.

We identified the need to manage and adapt policies themselves and discussed different adaptation requirements. Furthermore, we showed how policy adaptation can itself be enforced by other policies, specified in the same Ponder policy notation. We presented examples of adaptation by dynamically changing the parameters of a network QoS policy to give new values to attributes of the managed objects at run-time and of adaptation by selecting and enabling a policy from a set of pre-defined QoS policies Finally, we proposed an enforcement architecture for our ideas on policy adaptation.

We are currently implementing Policy Management Agents that interpret policy actions into configuration parameters for DiffServ routers, using SNMP for configuring DiffServ MIB's within Linux routers [37]. We have done an implementation for policy enforcement using (CLI–like) commands that the Network Simulator (NS-2) provides for managing DiffServ nodes and have experimented with adapting the DiffServ parameters.

The main issues to resolve are to determine what can be specified and implemented as policies and where scripts may be needed to provide the required adaptability. We intend to experiment with Linux-PC based routers as well as commercial routers or switches and to evaluate the performance implications of executing policies on routers. We are also interested in investigating particular methods or design patterns for building policy aware applications and to extend our framework with other techniques and interfaces for interaction between policy-based applications and policy-enabled networks in order to support dynamic adaptation.

We also intend to use this approach for management of MPLS networks in the future.

Acknowledgments

We gratefully acknowledge the support of the EPSRC for research grant GR/M 86019 (Ponds) as well as BT for support on the Alpine project.

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] Internet Engineering Task Force, Differentiated Services Working Group, http://www.ietf.org/html.charters/diffserv-charter.html

[4] Bernet, Y., Smith, A., Blake, S. & Grossman, D., An Informal Management Model for DiffServ Routers, Internet Draft, draft-ietf-diffserv-model-06.txt, February 2001.

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

[7] Nichols, K., Blake, S., Baker, F. & Black, D., Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, RFC 2474, December 1998.

[8] Heinanen, J., Baker, F., Weiss, W. & Wroclawski, J., Assured Forwarding PHB Group, RFC 2597, September 1999.

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

[10] 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.

[11] Strassner, J., Westerinen, A. & Moore, B., Information Model for Describing Network Device QoS Datapath Mechanisms, Internet Draft, draft-ietf-policy-qos-device-info-model-03.txt, May 2001.

[12] 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.

[13] 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.

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

[15] Chan, K., Durham, D., Gai, S., Herzog, S., McCloghrie, K., Reichmeyer, F., Seligson, J., Smith, A. & Yavatkar, R., COPS Usage for Policy Provisioning, RFC 3084, March 2001.

[16] Fine, M., McCloghrie, K., Seligson, J., Chan, K., Hahn, S., Bell, C., Smith, A. & Reichmeyer, F., Differentiated Services Quality of Service Policy Information Base, Internet Draft, draft-ietf-diffserv-pib-03.txt, March 2001.

[17] T. Yamazaki: "Adaptive QoS Management for Multimedia Applications in Heterogeneous Environments: A Case Study with Video QoS Mediation", IEICE Trans. Commun., November 1999.

[18] Ferrari, T. & Chimento, F. (2000) A Measurement-based Analysis of Expedited Forwarding PHB Mechanisms. Proc. IWQoS 2000, Pittsburgh, PA, June 2000.

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

[20] Trimitzios, P. et al. (2001). An Architectural Framework for Providing QoS in IP Differentiated Services Networks. Proc. IM 2001: 2001 IEEE/IFIP International Symposium on Intergrated Network Management, Seattle, USA, May 2001, pp. 17-34.

[21] Prieto, A. & Brunner, M. (2001) 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.

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

[23] 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.

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

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

[26] 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.

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

[28] Yoshihara. K., Isomura M. & Horiuchi, H.(2001) 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.

[29] 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.

[30] 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.

[31] Marshall, I., Gharib, H., Hardwicke, H. &.Roadknight C. (2001). 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.

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

[33] Lutfiyya, H., Molenkamp, G., Katchabaw, M. & Bauer, M. (2001). 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.

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

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

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

[37] Kim, J.-Y., Hong, J. W.-K., Ryu, S.-H. & Choi, T.-S. (2000). Constructing End-to-End Traffic Flows for Managing Differentiated Services Networks. Proc. DSOM 2000: 11th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, Austin, TX, USA, 4-6 Dec. 2000, Springer-Verlag LNCS 1960, pp. 83-94