To be Presented at 9th IEEE/IFIP Network Operations and Management Symposium
(NOMS 2004) May 2004, Seoul, Korea

PONDER Policy Implementation and Validation in a CIM and Differentiated Services Framework

L. Lymberopoulos, E. Lupu and M. Sloman
Imperial College London, Department of Computing
180 Queen's Gate, SW7 2BZ, London, UK
{llymber, e.c.lupu, m.sloman}@imperial.ac.uk

Abstract

Policies are often used to define management strategies for networks, storage services or applications. Validation determines whether the policy implementation is feasible for the specific environment to which they apply and requires checking that the policy is consistent with the functional or resource constraints within the target environment. For example, do the policies assume functionality or specific operations, which do not exist in target routers, or bandwidth in excess of the capacity of data links? Where possible, static checking should be done prior to policy deployment in order to detect invalid policies at design time, but there are some policies, related to resource allocation, that depend on the current state of the system, and require policy constraints that must be checked dynamically at execution time. In this paper, we show how PONDER policies can be implemented and validated for Differentiated Services (DiffServ) by using CIM as the modelling framework for network resources as this is device independent. We describe a CIM DiffServ-metrics sub-model extension of the CIM Network sub-model which represents DiffServ traffic statistics and a Linux driver which translates CIM classes and variables to Linux traffic control classes and variables respectively.

Keywords

Policy-based management, Policy deployment, Policy validation, DiffServ

1. Motivation

There is considerable interest in the Internet community on policy-based techniques as a means of implementing adaptive QoS (Quality of Service) management, caching, persistence and security to support modern multimedia applications, mobility and ubiquitous computing. Policies are defined as rules governing the choices in behaviour of a system. Authorisation policies define what services or resources a subject (management agent, user or role) can access. Obligation policies are event triggered condition-action rules which can be used to modify the behaviour of routers, firewalls, or even applications in response to events such as failures, system attacks or

changing application requirements. In this paper we focus on router specific policies for reserving network resources, changing queuing strategy, or loading code onto a programmable router. In a real networking scenario, multiple policies will apply to the network elements in order to support the requirements of different applications, different users and cooperating but distinct administrative domains. There are many different network elements with varying capabilities and interfaces, so it is essential to ensure that they have the capability to implement the policy.

Policy validation is required to check that the device to which the policies apply support the required functionality i.e. the policy invokes an operation actually implemented by the device; the device has the required resources needed to satisfy the policy action and that any application specific constraints or restrictions imposed by the existing managed environment will actually be enforced. Validating policies at compile or configuration time prevents the management overhead and the potential problems that arise when actually trying to enforce policies that are not feasible in the given network environment. In this paper, we will discuss how network policy that applies to a DiffServ domain can be validated against the capabilities of the individual DiffServ network elements. Validation of policies that relate to end-to-end flows or service level agreements (SLAs) is not discussed in this paper.

CIM provides a standard device independent information model for the network elements which are potential targets for policies. Policies can be specified with respect to generic CIM target objects and so can be used for many different network elements which then map the CIM variables and actions to those actually implemented by the specific device. CIM could also be used to model the elements which actually interpret policies and perform policy-based management, but there is less emphasis on standardisation of these as they are vendor specific. Eventually vendors may support CIM interfaces to devices and network elements. In this paper we give a mapping from CIM to Linux Traffic Control commands for DiffServ routers.

In section 2 of the paper we outline the PONDER Policy framework and section 3 elaborates on policy validation within the framework. Section 4 describes our implementation of policy enforcement and validation for Linux DiffServ routers, section 5 gives example usage of our implementation with conclusions in section 6.

2. The PONDER Policy Framework

PONDER is an object-oriented, declarative language developed at Imperial College for specifying management and security policies [1]. For example, the following authorisation policy with the name bwalloc permits the Agroup to perform the action of setting up a videoconference with bandwidth of 4 Mb/s and priority of 3 to the BGroup in New York or the Dgroup in Boston between 16.00 and 18.00 daily.

```
inst auth+      bwalloc   {
  subject       Agroup;
  target        BGroupNY + DGroupBoston;
  action        videoconf (bw=4, priority=3);
  when          time.between (1600, 1800); }
```

The following obligation policy type named videoSetUp, takes two parameters – a subject which evaluates the policy and a target on which the action to reserve bandwidth is performed, when an event is received for a videoRequest with the requested bandwidth bw as a parameter. A constraint defines that the reservation will only take place if the allocated bandwidth plus the request is less than a maximum allowed bandwidth. It is assumed that the allocated and maximum bandwidth are variables held within the subject. Two instances of the policy are then created for different gateway subjects and router targets.

```
type oblig      videoSetUp (subject s; target t;) {
  on            videoRequest (bw);
  do            t.bwreserve (bw);
  when          ((s.allocatedbw + bw) < s.maxbw); }

inst USvideoSetUP = videoSetUp (gateways/USgateway, routers/USedgeRouter);
inst UKvideoSetUP = videoSetUp (gateways/UKgateway, routers/UKedgeRouter);
```

PONDER also supports grouping a set of policies into roles related to positions in organisationsRoles can also be used to group policies for a core or edge router [2]. Management structures can be defined as configurations of roles with policies applying to relationships between roles for organisational units such as departments or buildings. Inheritance permits specialisation of existing policy specifications for different environments. PONDER also allows complex actions to be implemented by dynamically loading scripts within the subject policy interpreter. Further details of the PONDER language are described in [1]. It should be noted that PONDER provides a superset of the functionality of the IETF and DMTF Policy Core Information Model (PCIM) and supports event triggered condition action rules, which is an essential requirement for adaptive policy-based management which performs management actions in response to external events.

3. Policy Validation within the PONDER Framework

We can observe that the enforcement of a network-level policy results in the creation and/or configuration of the functional elements which implement the policy within the routers to which the policy applies. For example, in a DiffServ environment, network policies are implemented through the creation, configuration and parameterisation of classifiers, meters, schedulers, queues, etc., that the network elements support. These individual functional elements are described in several information models, such as the CIM Network sub-model [3], and IETF DiffServ router model [4].

In this context, a policy is validated with respect to the target device capabilities to determine whether the target device is able to create the requested new functional element and/or configure functional elements with requested values. In the following, we will provide details of each case of validation and we will outline how policy validation can be implemented within the PONDER framework using CIM as the model for representing the policy target mechanisms and capabilities.

3.1. Policy validation with respect to the ability of the target devices to create DiffServ functional elements

This validation requires checking whether the target device is capable of performing an "add" operation for a DiffServ functional element, i.e.
i) Does the device support the requested DiffServ functional element type?
ii) Does the device have enough resources to create (add) the new functional element?

For the first case, consider the following policy that the administrator specifies for providing traffic conditioning on a set of edge routers.

Example 1: Network policy rule for creating DiffServ functional elements within the target routers

```
inst oblig    /Policies/TrafficConditioningOnEdge {
  subject     /PolicyAgents/NetworkPMA;
  target      t = /Routers/EdgeRouters;
  on          TrafficConditioningRequest ( meter_rate, dscp);
  do          AverageRateMeter avg_meter = new AverageRateMeter
              (meter_rate) ->
              t.addDiffServElement (avg_meter) ->
              DSCPMarker dscp_marker = new DCSPMarker (dscp) ->
              t.addDiffServElement (dscp_marker); }
```

The policy TrafficConditioningOnEdge instructs the target edge routers to create two functional elements: a meter of type AverageRateMeter and a marker of type DSCPMarker. This policy is valid only if the target devices support the meter type AverageRateMeter and the marker type DSCPMarker. In order to perform this type of validation, we must know the different types of functional elements a DiffServ-enabled device supports. A solution to this would be to query the CIM representation of the device which can indicate which types of functional elements a DiffServ-enabled device implements.

In the general case this type of policy validation can be implemented within the PONDER framework by using meta-policies as a means to specify the constraints that network policies must satisfy with respect to the DiffServ mechanisms that the targets support. Meta-policies defined in PONDER specify constraints over a set of policies, with respect to the permitted types of policies or the elements within the policies. These constraints apply to policies within a specific scope and effectively limit the permitted policies in the system. This allows the policies to be checked at specification time, before deployment, which is clearly an advantage over checking at policy execution time. The syntax of a meta-policy is based on the OMG Object Constraint Language (OCL) [5]. The body of the meta-policy specifies the constraint as a series of OCL semicolons separated expressions which can be boolean or navigation expressions. If any of the boolean expressions evaluates to *true*, execution stops and the action following the *raises*-clause is executed. Example 2 shows a meta-policy specifying constraints over a set of network policies with respect to the DiffServ mechanisms supported by policy targets, but meta-policies can be used for specifying validation constraints for any underlying technology.

Example 2: Meta-policy for specifying constraints for DiffServ mechanisms

```
inst meta notSupportedElement raises
   notSupportedElementException(invalid_policies) {
 [invalid_policies] = this.policies ->
  select   (p |   p.action ->
      exists (a | a.name = "addDiffServElement"     and p.target ->
          exists(t | t.notSupports(a.parameter.oclType))));

invalid_policies -> notEmpty; }
```

The body of the meta-policy contains two OCL expressions. The first one selects all policies (p) with the following characteristics: the action set of p contains at least one action named "addDiffServElement", whose parameter type (i.e. the type of the DiffServ element that p wants to add to each of the target devices) is not supported by at least one of the policy's target devices. Note that we use the OCL method oclType to obtain the type of the "addDiffServElement" action parameter. The action notSupports on the target device is a look-up operation on the CIM representation of the device which returns true if the device does not support the specific DiffServ element type, passed as a parameter. The second OCL expression returns true if the variable invalid_policies, which is returned from the first OCL exception is not empty. If the result of the last expression is true, the notSupportedElementException specified in the raises-clause is executed with the invalid_policies set as a parameter. An obligation policy could be triggered by this exception to perform corrective actions to resolve invalid policies. An example would be to install missing DiffServ mechanisms in a programmable router, as shown in example 3. The exception notSupportedElementException from Example 2 triggers the obligation policy PolicyForInvalidDiffServPolicies which installs the missing DiffServ mechanisms in the relevant programmable router.

Example 3: Policy rule for installing the non supported DiffServ mechanisms in the appropriate programmable routers.

```
inst oblig      PolicyForInvalidDiffServPolicies {
  subject       /PolicyAgents/NetworkPMA;
  on            notSupportedElementException (Policy p[])
                /* p is the set of policy objects that the meta policy in Example 2
                has returned as not valid*/
  do
  foreach pol in [p] {
    foreach action in [pol.actions]
      foreach t in [pol.target] {
        if (action.name = "addDiffServElement" and
                        t.notSupports( (DiffServ_Class) action.parameter))
        then   t.installMechanism   ((DiffServ_Class) action.parameter)); }}}
```

The policy rule PolicyForInvalidDiffServPolicies receives the set of invalid network policies with the event notSupportedElementException, which the meta-policy in Example 2 raises. The pseudocode following the do statement could be

implemented as a script policy action which finds the pairs < target_device, non-supported mechanism > and installs the non-supported mechanisms in the appropriate target devices using the method installMechanism.

As mentioned earlier, a policy is not valid when the device does not have the resources to create the requested functional element. As an example, consider a DiffServ device that can only support a limited number of traffic classes. A policy that tries to create a new traffic class within the device will fail if the maximum limit is exceeded.

This case of policy validation can be implemented using our CIM extensions as a means to specify the necessary information about the **capabilities** and the **current state** of the device – the maximum number of traffic classes and the current number of classes respectively. Since the current version of CIM does not include DiffServ-specific capabilities and state information, we are proposing a "DiffServ-metrics" extension to provide this information, which we will present in detail in section 4.

However, unlike the previous case, we cannot decide offline whether the policy is valid by checking the CIM representation of the device, since the decision depends on device current state. This means that the decision must be made at the time the policy is to be enforced, which implies that the conditions under which the policy is valid must be specified as constraints *within the policy specification*, as indicated in the following example. The policy rule PolicyToAddTrafficClass will only add a new traffic class when current number of classes is less than the maximum.

Example 4: Policy rule for creating a new traffic class when the target device supports a limited number of traffic classes

```
inst oblig      /Policies/PolicyToAddTrafficClass {
  subject       /PolicyAgents/NetworkPMA;
  target        t = /Routers/CoreRouters;
  on            addTrafficClassRequest (classOfTrafficParams[]);
  do            t.addTrafficClass (( classOfTrafficParams [] );
      /* The attributes CurrentClassesOfTraffic and MaximumClassesOfTraffic are defined
      in our proposed extension to CIM, see Figure 1 ) */
  when          t.CIM_Get (CurrentClassesOfTraffic) <
                t.CIM_Get (MaximumClassesOfTraffic); }
```

3.2. Policy validation with respect to the permitted values of variables within the DiffServ device

In the second case of policy validation, checking is necessary to ensure that policies can set variable values within the device only if the requested values fall within permitted ranges with respect to the device capabilities. We can distinguish between variables with static bounds and variables with dynamic bounds.

- **Variables with static bounds.** As an example, the length of a DiffServ queue is a variable that has a specific upper bound. A policy rule that attempts to set this variable greater than its upper bound is invalid. We can detect offline if the policy is valid, if we check the (extended) CIM representation of the device, which includes the **capability** information of the device (i.e. the maximum number of traffic classes the device supports).

- **Variables with dynamic bounds.** In this case, the bounds of the variable to be set are not static, but they depend on the state of the device. As an example, the maximum bandwidth that can be allocated to a particular class of traffic cannot exceed the available free bandwidth that the device can offer, which is calculated as: *available_free_bw = total_output_bw – current_allocated_bw (for all traffic classes)*

Assume that CIM extensions can provide the necessary information about the **capabilities** and the **current state** of the device: the total output bandwidth and current allocated bandwidth to all traffic classes respectively. However, we cannot decide offline whether the policy is valid so the validity must be specified as constraints in the policy specification, which is similar to the policy in example 4 so is not shown here.

3.3. Summary

A CIM based model for representing network devices, can provide two types of management information:

- Information about device's capabilities (e.g. type of mechanisms it supports, bounds on resources, bounds on specific objects' attributes)
- Information about the current state of the device (e.g. current resource allocation, current values of specific objects' attributes)

We can use this information to support static policy validation with respect to device capabilities, which can be performed offline using meta-policies as a means to specify the constraints on the permitted DiffServ mechanisms, resources and bounds on object attributes. The required information about the capabilities of the devices can be obtained from CIM models. Dynamic policy validation with respect to current state of the device can only be performed at the time the policy is to be enforced, by means of constraints which are specified as part of the policy rules to define the conditions under which the policy is valid. These constraints can reference information extracted from the CIM device model.

4. Implementation of policy enforcement and policy validation on Linux DiffServ routers

As we have discussed, for the purpose of validating policy against device and network capabilities, it is necessary to extract several parameters such as the number of QoS classes, the bandwidth allocated to each class, or the QoS mechanisms the managed routers support. CIM provides a generic model for representing DiffServ-enabled managed devices. In particular, the CIM Network sub-model (v2.7) defines the class QoSService, which is a subclass of the generic class NetworkService [3]. The QoSService is an aggregation of instances of the ConditioningService class, whose subclasses define the router DiffServ mechanisms represented by the CIM classes ClassifierService, MeterService, MarkerService, DropperService, DropperThreshold-Service, QueuingService and PacketSchedulingService. Each ConditioningService class may have subclasses that correspond to specific DiffServ mechanisms. For

example, the classes AverageRateMeterService, TokenBucketMeterService and EWMAMeterService (Exponential Weighted Moving Average) derive from the generic class MeterService to represent specific DiffServ metering mechanisms.

Our implementation uses the CIM network sub-model for representing the DiffServ elements that a router supports. In addition, since we need statistics related to DiffServ, e.g. number of implemented traffic classes, bandwidth allocated to each class of traffic, we have designed a "DiffServ metrics" sub-model extension to CIM, which provides this information for the management system as shown in Figure 1.

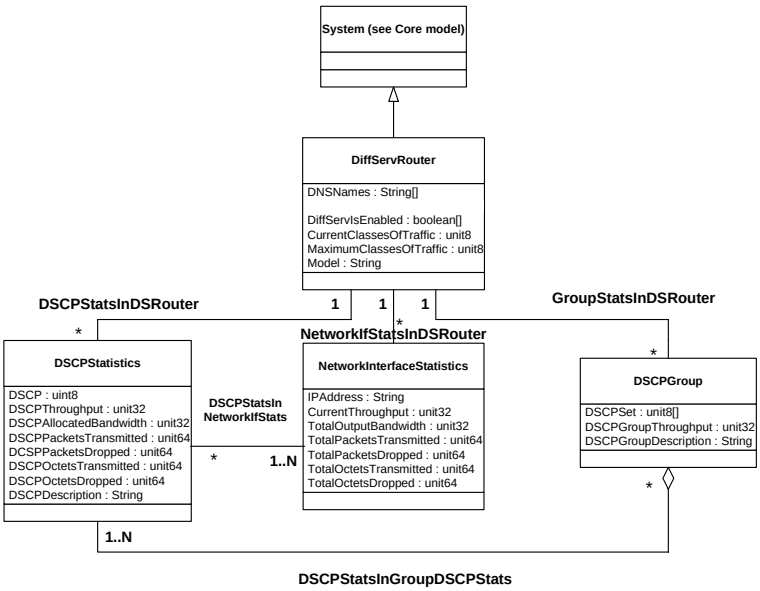

Figure 1: UML Diagram of the DiffServ-metrics CIM sub-model extension

The CIM class DiffServRouter is the abstraction of a DiffServ-enabled router and derives from the System class in the Core Model and focuses on representing a system that is DiffServ-enabled. The classes DSCPStatistics, NetworkInterfaceStatistics and GroupDSCPStatistics derive from the CIM class StatisticalData in the Core Model (this inheritance is not shown in Figure 1). The NetworkInterfaceStatistics class provides traffic statistics for every network interface card that belongs to the router. The DSCPStatistics class caters for statistics per implemented DSCP. A DSCPStatistics instance is associated with one or more NetworkInterfaceStatistics instances, as a particular DSCP may be implemented in more that one network interface on a single router. Finally, the GroupDSCPStatistics class is an aggregation of DSCPStatistics classes and provides statistics for a group of DSCPs which together define a class of service offered to the corresponding traffic aggregates. For example, the "Gold Group" in a router may be constructed from the DSCP offering the Expedited Forwarding per hop behaviour (EF PHB) [6], while the

"Silver Group" may be constructed from the DSCPs offering the Assured Forwarding per hop behaviour (AF PHB) [7]. A DSCPStatistics instance may belong to more than one GroupDSCPStatistics instances.

The DiffServ part of the CIM network sub-model and the DiffServ metrics sub-model have been implemented within the CIM Object Manager (CIMOM) from the WBEM Services project [8]. It provides a Java implementation of a CIMOM with two interfaces as shown in Figure 2. These are:

- The javax.wbem.client interface, used by clients for transferring CIM classes to and from the CIMOM. This communication can be realised either through HTTP (where CIM operations are defined in XML) or through Java RMI.
- The javax.wbem.provider interface, which providers attached to the CIMOM use to communicate with the CIMOM. Providers are implemented as Java classes and each provider is responsible for handling one or more CIM classes.

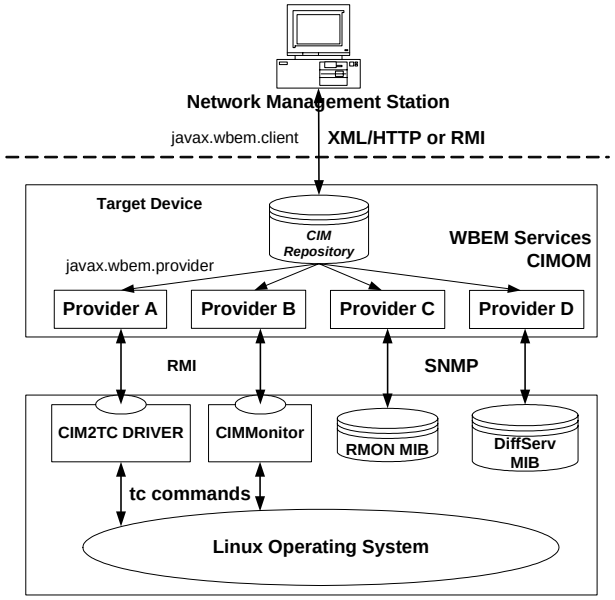

Figure 2: Architecture of Policy Enforcement and Validation System

Our current implementation includes a CIM2TC Linux Driver which is used to configure a Linux router using traffic control commands [9] via CIM classes specified in the CIM network sub-model. This approach is similar to that of [10], in which a driver component is used to translate classes that follow the DiffServ MIB object model [11] to Linux traffic control commands. We have used the "jtc" package from this implementation to represent the traffic control mechanisms of the Linux DiffServ router which is indicated as the LinuxDiffServ.LinuxDriver.tc package in our implementation, as shown in Figure 3. However our implementation overcomes the

limitations of the types of low level mechanisms they can configure as we provide a new algorithm for translating the DiffServ classes of the CIM network sub-model to the correct low-level "tc" classes, as indicated in the pseudocode outline for the CIM2TCDriver in Figure 4.

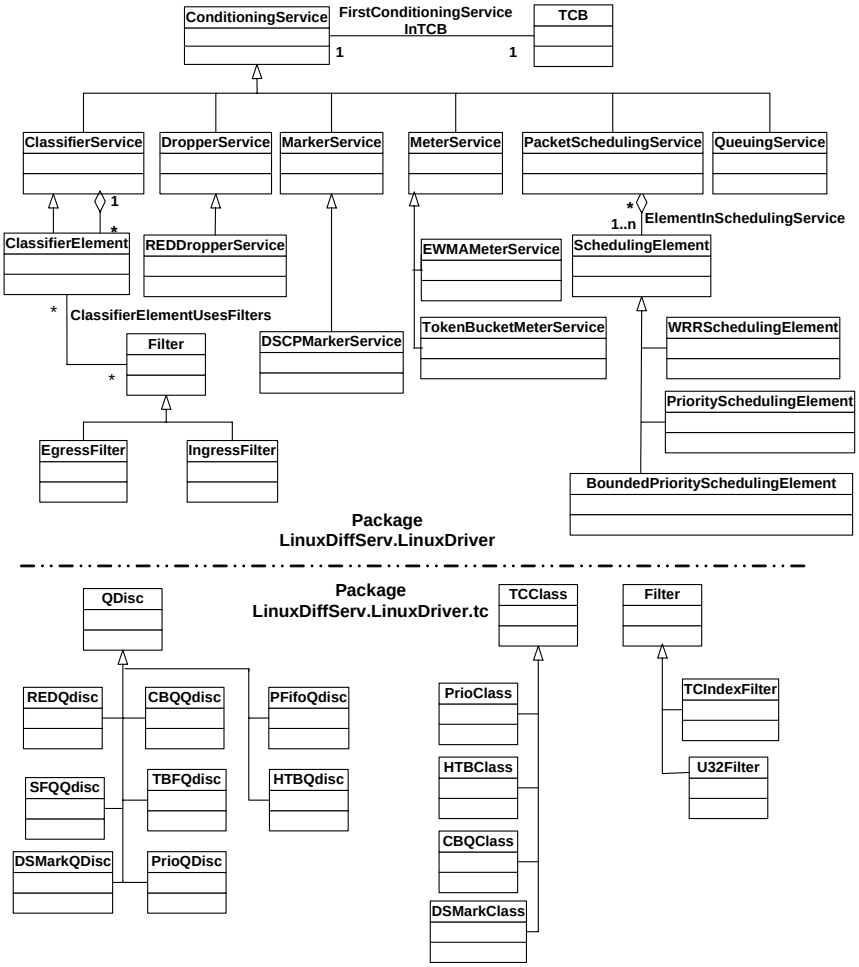

Figure 3: Object Model used by the CIM2TC Driver component

The CIM2TCDriver takes as input a set of instances of the Traffic Control Block (TCB) class. We added to the CIM network sub-model the TCB class to represent higher-level blocks constituted by combinations of ConditioningServices. A TCB object uses the association FirstConditioningServiceInTCB to hold the first ConditioningService; the other ConditioningServices are retrieved using the association NextService already defined in

CIM. Note that in our current implementation, in addition to the CIM classes defined in the CIM network sub-model, we have also used the non-CIM class Filter to represent a generic DiffServ filter within a ClassifierElement, its subclasses IngressFilter and EgressFilter to distinguish among filters at the egress and ingress interfaces and the association ClassifierElementUsesFilters to associate Filter objects to ClassifierElements. The CIM2TCDriver parses each TCB chain using the translation algorithm outlined above in order to derive the set of "tc" classes that correspond to the input TCB. We have also developed a Graphical User Interface for manually configuring Linux routers [12].

```
foreach tcb in tcbs[] {
        if (deviceIsNotConfigured) then outerQdisc = new DSMarkQdisc();
        schedulingElement = tcb.getSchedulingElement();
        filters[] = tcb.getFilters();
        if (schedulingElement is a WRRSchedulingElement) then {
          innerQdisc = new CBQdisc( … );
          DiffServClass = new CBQClass( …);}
        else if (schedulingElement is a PrioritySchedulingElement) then {
                innerQDisc = new PrioQdisc( …);
          DiffServClass = new PrioClass( … );}
        else if (schedulingElement is a BoundedPrioritySchedulingElement) then {..}
        // Implement policing or shaping behaviour according to the position of the
        MeterService in the tcb
        if ( exists MeterService before ClassifierElement ) then create policing filters;
        // Create metering, marking and dropping classes and qdiscs inside the DiffServ class
        foreach element in tcb after ClassifierElement and before SchedulingElement {
          if ( element is a DSCPMarkerService) then add new DSMarkQdisc ( … ) ;
          if ( element is a TokenBucketMeterService) then add new TBFQdisc (…) ;
                if ( element is a REDDropperService) then add new REDQdisc(…); }
        // Create the filters that will direct the packets to the DiffServ class
        createTCFilters (filters[ ]); }
```

Figure 4: Pseudocode for the CIM2TC Driver component

We have implemented a Provider to handle the DiffServ metrics sub-model classes presented in Figure 1 (shown as provider B in Figure 2). This communicates with the CIMMonitor component to get DiffServ variables from the Linux operating system by issuing traffic control statistics commands ("tc –s commands"). For example, in order to obtain the CIM variable CurrentClassesOfTraffic (from CIM class DiffServRouter), Provider B issues a request to the CIMMonitor, which in turn queries the current "tc" configuration of the device to get the current number of implemented DiffServ classes. An alternative for retrieving DiffServ traffic statistics could be to use SNMP to get variables from a DiffServ RMON probe indicated by Provider C in figure 2 which could translate CIM attributes to SNMP get requests for an RMOM MIB. However, there is not yet an implementation of an RMON probe for DiffServ on Linux, so this has not been implemented.

We have implemented Provider A shown in Figure 2 to handle the DiffServ CIM classes that belong to the package Linux.DiffServ.LinuxDriver presented in Figure 3. This provider is used to change the configuration of the Linux router starting from the CIM representation of the DiffServ mechanisms. This is done by communicating the new TCBs to the CIM2TCDriver component. It would be possible to use SNMP to

configure the Linux device, as described in [13] and indicated by Provider D in Figure 2, which could issue SNMP set requests to set MIB variables within the DiffServ MIB. We were unable to find a complete implementation of a DiffServ MIB for Linux, so have not implemented this.

We have provided PONDER Policy Management Agents (PMAs) and Target Policy Objects [14] as CIM clients to allow the PONDER components, which reside in the Network Management System in Figure 2, to add/update/remove CIM classes to configure the Linux DiffServ mechanism and evaluate policy constraints or meta-policies that contain CIM variables. Note that the PONDER system does not communicate directly with the CIM2TCDriver or the CIMMonitor components, but via the CIMOM. This allows the PONDER system to configure and validate policies that apply to other types of CIM-enabled devices as well as Linux routers.

5. Linux Router Configuration Example

In the following, we will show an example of using our prototype implementation based on a testbed of 4 Linux DiffServ routers to enforce the policy validation rule presented in Example 4. The core router in our network setup is the Linux box *Achilles* (146.169.14.74). The policy rule PolicyToAddTrafficClass must be applied to the target devices only if the current number of implemented DiffServ classes is less than the maximum number the devices can support. The administrator uses the Editor tool of the PONDER Toolkit [14] to edit, compile, load and enable the policy rule PolicyToAddTrafficClass. The policy objects are stored in an LDAP directory server.

```
[root@Achilles root]# ./RunDriver.sh
tc class   add   dev eth1 classid 2:5 parent 2:0 cbq bandwidth 100Mbit avpkt 1000 rate 450Kbit
bounded   isolated   weight 45 prio 6
tc qdisc add dev eth1 parent 2:5 handle 3:0 pfifo   limit 5
tc filter   add   dev eth1 parent 2:0 prio 4 protocol ip handle 0x2b tcindex   classid 2:5 pass_on

[root@Achilles root]# DiffServMonitor &
ClassID    Queing    Rate      Sent packets    Sent bytes    Dropped Packets    Bandwidth (bps)
2:1        cbq       150Kbit        109          10682              0               784
2:2        cbq       250Kbit          0              0              0                 0
2:3        cbq       350Kbit          0              0              0                 0
2:4        cbq       5Mbi          2018         384727              0              9088
ClassID    Queing    Rate      Sent packets    Sent bytes    Dropped Packets    Bandwidth (bps)
2:1        cbq       150Kbit        111          10878              0              1568
2:2        cbq       250Kbit          0              0              0                 0
2:3        cbq       350Kbit          0              0              0                 0
2:4        cbq       5Mbi           205         388843              0             32928
2:5        cbq       450Kbit          0              0              0                 0
ClassID    Queing    Rate      Sent packets    Sent bytes    Dropped Packets    Bandwidth (bps)
2:1        cbq       150Kbit        112          10976              0               784
2:2        cbq       250Kbit          0              0              0                 0
2:3        cbq       350Kbit          0              0              0                 0
2:4        cbq       5Mbit         2057         390127              0             10272
2:5        cbq       450Kbit          2            184              0              1472
```

Figure 5: Result of enforcement of the rule PolicyToAddTrafficClass

The policy rule PolicyToAddTrafficClass is evaluated at run-time by the management agent NetworkPMA. If the constraint defined by the when clause evaluates to true, the target router is instructed to perform the addTrafficClass operation. The NetworkPMA receives the obligation event addTrafficClassRequest with parameters indicating the network interface on the target where the new class is to be added, the DSCP associated with the traffic class, the bandwidth and the priority that will be assigned to this class. The policy rule retrieves the relevant CIM variables from the target router and evaluates the policy constraint which evaluates to true for the request addTrafficClassRequest ("eth1", 43, 450, 6). 43 (0x2b) is the new DSCP we want to implement. As the policy rule is valid for the current network state, it will be enforced on the network. However the second time this rule is triggered, the policy constraint evaluates to false since we have reached the maximum number of supported classes and therefore the policy action is not enforced. Figure 5 displays the "tc" commands that the CIM2TCDriver generated for performing the policy action addTrafficClass, when the action is implemented as the TCB presented in Figure 4. It also displays the new DiffServ configuration upon the enforcement of the policy action addTrafficClass.

Further implementation work is needed to integrate the PONDER Editor with the Graphical User Interface so that the administrator is able to graphically specify policy actions that involve CIM classes to add/update/delete TCBs within the managed devices. Finally, we also aim to integrate our policy framework with the CISCO Information Model (CIM-CX) that represents Cisco's specific network mechanisms. This will enable our framework to apply to both Linux and Cisco routers.

6. Conclusions

We have presented an approach for validation of policies with respect the devices to which they apply. Although we presented examples of DiffServ routers, the approach could be applied to validation of security management policies or building management policies etc. A representation of the device capabilities and states such as those in our CIM extensions is essential. Our approach applies to individual network devices within a domain in which the DiffServ based policies apply. The situation becomes much more complex when interactions between the policies related to end-to-end flows or different service level agreements (SLAs) are considered. This requires determining whether the introduction of a new SLA could potentially violate the policies relating to existing SLAs. In the simplest situation this could require determining whether the current network topology has the resources to support the new SLA which can be done by Traffic Engineering systems. However, when the SLAs cater for dynamic allocation of resources based on time or application requests, this becomes more complex to do.

Another issue is that the end-to-end path may not be within the administrative domain of a single service provider. This requires interaction between service providers to exchange policy information, and current state of the network topology. These issues will be addressed in a new project in collaboration with Surrey University.

ACKNOWLEDGMENT

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

References

[1] Damianou, N., N. Dulay, E. Lupu and M. Sloman. The PONDER Policy Specification Language. Workshop on Policies for Distributed Systems and Networks, Bristol, UK, Jan. 2001, Springer-Verlag LNCS 1995, pp. 18-39.

[2] Sloman, M. and E. Lupu. Policy Specification for Programmable Networks. Proc. of First International Working Conference on Active Networks (IWAN'99), Berlin, June 1999, ed. S. Covaci, LNCS, Springer Verlag, Berlin, June 1999, pp. 73-84.

[3] DMTF Common Information Model (CIM) Version 2.7 Specification, http://www.dmtf.org/standards/cim_schema_v27.php

[4] Bernet, Y., Smith, A., Blake, S. & Grossman, D., An Informal Management Model for DiffServ Routers, RFC 3290, May 2002.

[5] OMG 1999B. Object Constraint Language Specification, version 1.3

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

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

[8] WBEM Services Project, http://wbemservices.sourceforge.net

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

[10] Martinez, M. M. Brunner, J. Quittek F. Strauß, J. Schönwälder, S. Mertens, T. Klie, Using the Script MIB for Policy-based Configuration Management. Proc. IEEE/IFIP NOMS 2002: Network Operations and Management Symposium, Florence, Italy, Apr. 2002, pp 187-202.

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

[12] Chinedu Zelu. Graphical User Interface for DiffServ configuration of CIM enabled Linux Routers, MSc. Thesis, Department of Computing, Imperial College London, Sep. 2003.

[13] Kim, J.-Y., Hong, J. W.-K., Ryu, S.-H. & Choi, T.-S. Constructing End-to-End Traffic Flows for Managing Differentiated Services Networks. Proc. 11th IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management, Houston, Dec. 2000, Springer LNCS 1960, pp. 83-94.

[14] Damianou, N., Dulay, N., Lupu, E., Sloman, M., Tonouchi, T. Tools for Domain-based Policy Management of Distributed Systems. Proc. IEEE/IFIP NOMS 2002: Network Operations and Management Symposium, Florence, Italy, Apr. 2002, pp. 203-217.