Quality of Service Policy Management over a DiffServ Network

L. Cruvinel, T. Vazão, F. Mira da Silva

Abstract—This paper proposes a QoS network architecture, which provides a complete top-to-bottom policy definition approach. IETF’s Differentiated Services (DiffServ) architecture and mechanisms are used here as a framework for a Quality of Service implementation. Commands and high level policies (SLA – Service Level Agreement, and TCS – Traffic Conditioning Specifications) are initially defined using XML, then stored in a LDAP repository for further reference, and mapped to a platform-independent Policy Information Base (PIB) for distribution using the COPS protocol. An extensible framework of service classes helps classifying traffic and aggregating flows with similar requirements, providing them the adequate share of network resources. The architecture is scalable and may be adopted in large networks.

Index Terms—QoS, DiffServ, Policy-Based Management, COPS.

I. INTRODUCTION

The need for Quality of Service (QoS) support in the Internet has been motivated by the emergence of new services and applications.

To date, real-time conferencing, voice and other multimedia traffic is routed by IP networks in essentially the same way than any other type of traffic. Treatment given by IP protocol to the bulk of packets is based in a best-effort (BE) service model with no delivery guaranties or priorities. However, bandwidth limitation is not the only responsible for a bad performance in multimedia applications. Other requirements are not compatible with egalitarian treatment for all network packets [1][2][3].

Broadband is critical for getting a good performance in the communications networks nowadays, but its use in exclusive to solve traffic limitations will probably result in worst management problems in the medium to long term. A better approach for this issue is the systematic study of traffic conditions and profiles to optimize resource utilization and avoid wasteness [4].

The Internet Engineering Task Force (IETF) proposes a scalable architecture - Differentiated Services (DiffServ) - where high level policies, the Service Level Agreements (SLAs), are negotiated between the customer and the service provider. The DiffServ model relies on the use of a restricted set of packet handling mechanisms that implement QoS policies, which are applied to incoming traffic according to its associated service class. DiffServ domain’s border routers classify, mark and police traffic, while interior routers apply Per-Hop Behaviors (PHB) to each aggregate as specified by a service class [5].

DiffServ’s architecture does not specify management and access control. This may be provided, for example, by the adoption of Policy-Based Networking (PBN), a model developed by the Resource Allocation Protocol (RAP) group in IETF [6], [7].

Some of the direct or implicit needs of the DiffServ model include:

- Proper syntax and semantics for the definition of high-level policies. Means of manipulation and languages have been proposed for this purpose by the IETF or others [8], [9].
- An interface between the high-level policy definitions and the configuration commands in routers from different manufacturers. IETF provides a Policy Information Base (PIB) for the transportation of policies between the policy manager and the routers where they must be implemented [12]. The transport protocol for PIB tables is Common Open Policy Service (COPS) and the policy provisioning is done with COPS Usage for Policy Provisioning (COPS-PR) [13], [14].

Several architectures have been already proposed and developed, in works that generally implement part of the architecture [15], [16], [17]. However, none of them presents a complete framework, where users (clients or network operators) can specify high level policies, which are directly mapped into the low level policies, most adequate to the QoS support architecture provided by the network. The work described in this paper proposes a complete top-to-bottom architecture and focuses on the architecture design of the policies and related management environment in a DiffServ network, and on its implementation in Linux and Java-supporting machines. Our proposal comprises a set of applications that provide an interface between high-level policies of service contracts and a provisioning manager distributing low level policies to the relevant systems.

This paper is organized as follows: this section presents the motivation and technical challenges behind the work, section II refers and discusses some related work, section III defines the two types of policy definitions used in the
model, section IV discusses the proposed architecture, section V describes the implementation and shows some experimental results, and section VI presents some conclusions and proposals of future work.

II. RELATED WORK

The last decade of the 20th century saw the arising of a new management paradigm, the Policy-Based Management, where the managed system behavior may be dynamically changed for better attending network actual states. The adaptation is made through Policies for the various areas of management, including QoS and SLA provisioning. Distributed, centralized and hierarchical models have been used for implementing Policy-Based Management architecture.

The distributed model originated with the scientific research group of Imperial College offering support for security and access control in distributed systems of telecommunications networks [18]. A main characteristic of this model is the presence of a set of common services to maintain communication, distributed object and distributed processing altogether with management services to support the management applications.

IETF is currently working on a standard centralized framework and related protocols for PBN model [19], [20]. The IETF’s DiffServ model is an architecture for provisioning of QoS in IP domains. A domain is any contiguous set of routers operating with the same policies and behavior definitions. DiffServ is scalable because it does not use end-to-end signalling or flow state storing. It specifies the behavior for flow aggregates inside a particular domain (BA – Behavior Aggregate), but allows inter-domain traffic conditioning agreements [21]. Once the policies are in place, border routers mark selected packets using IP header’s Differentiated Services Codepoint (DSCP) attribute. The main advantage of DiffServ model is the definition of standard service classes, allowing the domain border routers the measurement and aggregation of inter-domain traffic. Inside the domain, the traffic is subject to routing behaviors of the interior nodes based on DSCP. The recommendations of the framework include:

- A management console to create, edit or look at policies.
- The repository to store and reuse policies.
- A Policy Management Tool (PMT), to maintain policies and manage conflicts among them.
- A Policy Decision Point (PDP), to distribute policies to network elements. IETF suggests using Policy Information Base (PIB), a virtual set of tables for policies) and COPS for transportation and handle of policies here.
- Agents in network elements where the policies are to be implemented, known as Policy Enforcement Points (PEP).

Use of a hierarchical model has been proposed in the Tequila Project, where agent-manager hybrid entities exist at different layers of the hierarchy to manage network elements of the next lower layer [22]. High level policies are refined into lower level policies for each layer to reflect the management architecture.

All of these architectures provide a separation between the high level policies and the supporting infrastructure. The IETF model has been chosen for this work as it allows the use of standard, industry tested and accepted, open-source technologies, being specially targeted for adoption in corporate or educational networks.

III. POLICY DEFINITION

The concept of Policy-Based Networking allows definition of QoS policies in two levels: high level policies, related to the business process, and low level policies, related to technology. Policies are initially built in relatively simple terms in an administration station, then are translated to a more technologically adequate form and sent to network elements.

A. High Level Policies Definitions

Textual representation may not be sufficient nor adequate for exchanging information among diverse systems, leading to the need for a consistent language capable of specifying management parameters.

In the case of IETF’s policy framework, as the language is not part of the specification, each implementation must choose one. Extensible Markup Language (XML) choice has proved adequate because the syntax and semantics of data may be verified programmatically, using parser libraries publically available. In the interoperability’s sake, it is recommended that the applications can recognize, validate attributes and apply specific behaviors to the elements of a XML document or information unit, and also store and reuse information in this format [23].

This work uses XML structures to transport policies from the User Interface to the Policy Management Tool and from the PMT to the Policy Decision Point. In fact, all conversation among these modules is done in XML.

A directory is a local or distributed database, optimized for browsing and searching. LDAP initially was an extension of X.500 protocol for IP networks, and requires few resources for accessing directory services using connected-oriented protocols [24].

LDAP directory allows aggregating in a single place information about access, authentication and authorization. Its logical structure is based in the concept of entry, which contains information about some object. For example, an object may be a SLA or a Traffic Conditioning Specification (TCS). Entries have attributes with syntax, type and one or more values, and are organized in a tree structure, the Directory Information Tree (DIT). Each object in the tree is globally identified thru its Distinguished Name (dn).

High level policies coming from UI are segmented in their building and functional blocks before storing them in the repository. This segmenting allows reusability of any condition or action that already exists.

B. Low Level Policies Definitions

A Policy Information Base is a virtual, expandable database that defines provisioning classes to map service requirements to the capabilities and role of devices. It works as an interface between high level policies
(business-oriented and business-like defined) and the policy agent in network elements (which knows how to configure its specific host for a given policy). Basic provisioning classes are defined by the IETF as the Framework Policy Information Base and also include textual conventions common to all provisioning clients. A specific PIB exists for DiffServ clients [25], [26].

The proposed solution aims to define and implement a restrict, but usable, set of interfaces for filter parameterization, built upon a static basic configuration. The available options include the IP header attributes as entries of the PIB table frwkpFilterTable [25]. PIB transport between PDP and PEP is done using COPS protocol as suggested by IETF [27]. The COPS protocol and its extensions for provisioning (COPS-PR) offer support for multiple clients and make the provisioning of specific-areas policies, such as QoS, security, accounting, etc. It uses a client/server model over TCP and defines the mechanisms and conventions used for communication between the provisioning server and its clients [27], [28].

IV. PROPOSED ARCHITECTURE

A. Functional Blocks Definitions

The architecture of the solution is based on IETF’s DiffServ model. Implementation of this model is not necessarily bound to specific technologies, although there are suggestions (e.g. LDAP for the repository, specific queue disciplines for specific classes) as well as related working being done (e.g. policy specification languages).

The functional modules of this architecture, depicted in Figure 1, include a graphical User Interface (UI), the Policy Management Tool (PMT), the Policy Repository (LDAP repository), the Policy Decision Point (PDP) and the Policy Enforcement Point (PEP).

Main principles of the architecture are:

- Initial, static configuration of basic DiffServ mechanisms and service classes.
- Dynamic service allocation.
- Automatic translation of policies from high level definition to low level form.
- Dynamic implementation and removal of policies in agreement with established SLA and available resources.
- One-to-one correspondence between DiffServ classes and Service Classes presented to the user.

A static basic DiffServ configuration in border and interior routers is at the core of the proposed architecture. It implements DiffServ classes BE (Best Effort, the default) AF1 to AF4 (Assured Forwarding with three drop priorities each) and EF (Expedited Forwarding), with behaviors shown in Table I for out-of-profile packets.

This static basic configuration is implemented by an agent running at the routers, the PEP, which, at startup, reads PHB information from a configuration file and runs a script with adequate parameters. 10 Mbps were used as a generic value for maximum rate in all classes, although modifications may done for experimental evaluation. Additional dynamic high level policies are mapped on the flight from TCS built by the user or administrator.

<table>
<thead>
<tr>
<th>Service Class</th>
<th>DiffServ Class</th>
<th>Action for Out-of-Profile Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diamond</td>
<td>EF</td>
<td>Drop*</td>
</tr>
<tr>
<td>Platinum</td>
<td>AF1</td>
<td>Remark (DP1 &amp; DP2), Drop (DP3)*</td>
</tr>
<tr>
<td>Gold</td>
<td>AF2</td>
<td>Remark (DP1 &amp; DP2), Drop (DP3)*</td>
</tr>
<tr>
<td>Silver</td>
<td>AF3</td>
<td>Remark (DP1 &amp; DP2), Drop (DP3)*</td>
</tr>
<tr>
<td>Bronze</td>
<td>AF4</td>
<td>Remark (DP1 &amp; DP2), Drop (DP3)*</td>
</tr>
<tr>
<td>Traditional</td>
<td>BE</td>
<td>Nothing</td>
</tr>
</tbody>
</table>

*Out-of-profile EF packets must be dropped from queue tail, to obey low delay requirement.

If out-of-profile AF packets were remarked to other classes, they could reach destination out of sequence, which is not compatible with Assured Forward Service as defined by IETF.

Whenever defining a new policy, the User Interface should be used. A series of dialog boxes allow the administrator to define SLA and the user or administrator to create TCS. SLA are assigned to a username and define service class, traffic endpoints, maximum bandwidth and time of application. TCS are children of SLA, and must respect the parent’s parameters. There may be many TCS children of one SLA, provided that the sum of their respective bandwidths is equal or less their SLA’s total bandwidth. Only the administrator or the user to which the SLA is assigned to may create children TCS. Only the administrator may remove a SLA.

Communication between UI and PMT and between PMT and PDP is done in XML. This benefits integration and consistency of the architecture’s modules. Translation from XML schema to LDAP schema attributes is almost always one-to-one. Exceptions are conditions and actions, which represent reusable objects and are used as references. PMT is the repository interface. It will track start and end times for the policies, acting in conformance to command the PDP to install or remove a policy in all border routers.

High level policies trigger the creation of low level ones, by defining policy filters, which are sent by the PDP to all relevant PEP agents in response to a command of the PMT. The PDP knows all running routers in the domain (as long as their PEP are reporting). PIB tables are sent between the PEP and the PDP, first for reporting the type and capabilities of the router and, after that, for configuring additional policy filters in the external
interfaces of the border routers.

An essential aspect of the architecture is the difference in basic static configuration for external and internal interfaces of the border router. External interfaces are initially configured only with support for policing filters, without other elements. When in place, dynamic filters select and police flows of traffic. This selection does not mark packets physically, what is done by the internal interfaces which have all DiffServ Classes defined. When the selected packets leave the border router and enter the DiffServ domain, they are marked and are submitted to specific behaviors configured in interior routers.

Figure 2 depicts the process of creation of a top-to-bottom policy, concerning the creation of a new TCS.

First of all, user authentication and SLA identification are required, meaning that a set of XML messages are transferred between the UI and the PMT. Then, the creation of the TCS is requested, by the PMT to the PDP, using an XML message that describes all the important filter parameters, like classification fields and meter parameters. This high level policy is converted into a COPS request, which is send to the PEP and, when it arrives to the PEP, it is translated into the LINUX Traffic Controller scripts, used to implement the filter.

**V. IMPLEMENTATION AND EXPERIMENTAL EVALUATION**

The architecture was developed in Java-supporting machines. User Interface and PMT were written from the scratch. PDP and PEP are based in libraries and example code available from University of Tampere [10].

The implementation was tested functionally and in performance. Functional tests can demonstrate the capabilities of the developed applications. Performance tests help proving the model’s adequacy and implementation advantages in a real network. Tests took place in an isolated network built for this purpose.

All computers had the same hardware and operating system (Linux Red Hat 9, kernel 2.4.20.8) and the arrangement is shown in figure 3. Using the monitoring computer as a host for the PMT, as well as one single machine for the PDP and UI roles, will have little impact on the performance tests, because their main purpose is to measure the DiffServ working on the network, after configuration by the UI, PMT and PDP.

Classes were predefined in the LDAP repository to map the DiffServ model. They are Diamond (EF), Platinum DP1 to DP3 (AF1), Gold DP1 to DP3 (AF2), Silver DP1 to DP3 (AF3), Bronze DP1 to DP3 (AF4).

Each class was given a bandwidth limit of 10 Mbps.

A. **Functional Tests**

These tests were meant to confirm (1) the correct working of the User Interface in showing, creating, modifying and removing policies (SLA & TCS); (2) the effective configuration of border and interior routers. The latter includes the basic configurations applied by PEP application (different for each interface of the border router and different for the interior router) plus the policies (TCS) defined with the User Interface (applied to the external interface of the border router).

After starting PEP application, the external interface of the border router may accept additional marking and policing configuration commands, while the internal interface will have a static basic configuration. The interfaces of the interior router are configured in a similar way to that of the border router’s interior interface.

Setup and elimination of SLA and TCS are done through User Interface, in any Java-enabled machine connected to the network. TCS implementations will create various conditions and actions in the repository, if they are not yet present. Using a packet capture application, the desired behaviors were confirmed to be implemented in both routers. TCS elimination is also done in a consistent way by the applications.

**B. Performance Tests**

Performance tests require more control on the running environment, to avoid unnecessary loads and wrong time measures. All clocks were synchronized with NTP (Network Time Protocol) and the monitoring was done by a separate machine (see figure 3).

The administrator will have worse response times in starting the User Interface, because it loads, from the repository, all SLA and TCS information at once through the PDP application (applied to the external interface of the border router).

Setup and elimination of SLA and TCS are done through User Interface, in any Java-enabled machine connected to the network. TCS implementations will create various conditions and actions in the repository, if they are not yet present. Using a packet capture application, the desired behaviors were confirmed to be implemented in both routers. TCS elimination is also done in a consistent way by the applications.

Ping commands were used to evaluate TCS effects on traffic. Graphic in figure 4 shows the times and losses for the same set of ping commands (with flooding option and
packets of 84 bytes) but configuring bandwidth differently for each test with the use of the developed applications. Further tests showed that shaping and policing were being done almost exclusively at the border router.

VI. CONCLUSION

QoS policy management in a computer network may be implemented altogether with IETF’s DiffServ architecture to easy router configuration and to make it consistent at the same time. High-level policy definitions are managed, stored and reused independently of the technical details of their implementation in the relevant equipments. The proposed solution, even if tested with a limited number of policies and routers, demonstrates the viability and usefulness of the DiffServ and PBN architectures, and the possibility of implementing them with open, well-established technologies.

Future work should include additional development of the applications and libraries for the model to keep updated, mainly with the standards of XML and PIB. New functionalities may be easily added, e.g. getting a real time report of policies in each router thru the User Interface, or using the UI for managing DiffServ classes’ parameters and the platform should be integrated in an educational environment.

ACKNOWLEDGMENT

The authors wish to thank the people at Universidade Autónoma de Lisboa (Lisbon, Portugal) for providing the means used in the development and tests of this implementation.

REFERENCES