An XML-driven Framework for Policy-Based QoS Management

by

Ntanzi Machungo Carrilho

BSc(ENG) Electrical and Computer Engineering

A thesis submitted to the

Department of Electrical Engineering

The University of Cape Town

for the degree of

MSC(ENG) ELECTRICAL ENGINEERING

Supervisor: Neco Ventura

June 2007

Statement of originality

I declare that the work presented in the thesis is, to the best of my knowledge and belief, original and my own work, except as acknowledged in the text, and that the material has not been submitted, either in whole or in part, for a degree at this or any other university.

Ntanzi Machungo Carrilho

Signed by candidate

June 2007

Acknowledgments

I would like to express my sincere gratitude to the following individuals and organisations for their assistance during the course of this project.

Mr. Neco Ventura, for the assistance he provided throughout the execution of this project.

Mr. Vitalis Ozyanyi, Mr. David Waiting, and Miss Gabriela Aragao for their constructive feedback.

Mr. Eric Beda, for the encouragement and support.

My fellow colleagues in the Communication Research Group (CRG), for all the interesting discussions.

Dr. O. S. Ameen and Prof. Nicolas Novitzky, for keeping me healthy during this period.

My parents and my sister, for their unconditional love and support.

Abstract

Policy-Based Network Management (PBNM) is a new management paradigm that simplifies the management and administration of communication networks. Current PBNM solutions have, however, shown a number of shortfalls. Firstly, the systems deal with a wide variety of policy representation forms that pose serious problems related to the manageability of policy frameworks, and introduce interoperability issues between the various elements of a PBNM system. Secondly, the policy provisioning protocol normally used in PBNM systems - the Common Open Policy Service for Policy Provisioning (COPS-PR) - is associated with a number of serious problems related to the flexibility and efficiency of its data model and usability of the protocol itself.

This thesis proposes to solve the aforementioned problems by applying XML and companion standards in the design and implementation of a PBNM system, placing focus on QoS management. XML has powerful modelling capabilities and allows the design of data models of arbitrary complexity, as opposed to the data description language of COPS-PR which is limited in flexibility. XML-based protocols are text-based and therefore human readable. As a consequence such protocols are easier to use. Furthermore, because of the widespread support of text- and XML-based processing tools the development costs and deployment cycles of systems would be reduced.

To test this proposition, the author proposes an XML-driven architecture for Policy-Based QoS Management focusing on the IETF Differentiated Services framework. The architecture employs an XML-based network management protocol - the IETF Network Configuration protocol (NETCONF) - for policy provisioning, replacing COPS-PR. Apart from the traditional PBNM architectural elements, the architecture encompasses an XML-based QoS policy data model divided into three abstraction layers to simplify network management, and two QoS policy translation algorithms.

An evaluation platform was developed to implement the various design ideas and test the performance of the proposed system against a PBNM system based on COPS-PR. The performance results show that the proposed XML-driven PBNM system achieves satisfactory and, in certain areas, better results than the COPS-PR-based system.

Contents

List of Figures

List of Tables

List of Abbreviations

Notation	Description
AF	Assured Forwarding
BER	Basic Encoding Rules
cbs	Committed Burst Size
CIM	Common Information Model
cir	Committed Information Rate
COPS-PR	Common Open Policy Service for Policy Provisioning
DF	Default Forwarding
DLP	Device-Level Policy
DOM	Document Object Model
DSCP	DiffServ Code Point
ebs	Extended Burst Size
EF	Expedited Forwarding
GRED	Generic Random Early Drop
HTB	Hierarchical Token Bucket
LDAP	Lightweight Directory Access Protocol

Notation	Description
MF	Multi-Field Classifier
MIB	Management Information Base
MS	Management Station
NETCONF	Network Configuration Protocol
NLP	Network-Level Policy
NM	Network Management
PBNM	Policy-Based Network Management
pbs	Peak Burst Size
PCIM	Policy Core Information Model
PCIMe	Policy Core Information Model extensions
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PHB	Per-Hop Behaviour
PIB	Policy Information Base
pir	Peak Information Rate
PR	Policy Repository
PRC	Provisioning Class
PRI	Provisioning Instance
qdisc	Queueing Discipline
QPIM	QoS Policy Information Model
RED	Random Early Drop
SAX	Simple API for XML Processing
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SPPI	Structure of Policy Provisioning Information
srTCM	Single Rate Three Colour Marker

Notation	Description
TCB	Traffic Control Block
tcng	Linux Traffic Control - Next Generation
trTCM	Two Rate Three Colour Marker
XInclude	XML Include Language
XML	eXtensible Markup Language
XML-RPC	XML Remote Procedure Call
XPath	XML Path Language
XPointer	XML Pointer Language
XSLT	XML Stylesheet Language Transformation

Chapter 1

Introduction

The Internet started as a research project funded by the Advanced Research Projects Agency (ARPA) in 1969. The ARPAnet, as it was once called, connected only four computers. The first e-mail system was introduced in 1972 and wide deployment of TCP/IP, the Internet protocol suite, began in 1983. Its most popular application, the World Wide Web, was invented in 1990 by Tim Berners-Lee at the European Laboratory for Particle Physics Research (CERN). By 1992 the number of computers connected to the Internet was over a million, and since then the Internet Protocol (IP) became the dominant networking protocol [1]. IP was conceived as a best-effort protocol that provides robust, but unreliable service delivery. In other words it delivers packets to their destination without any service guarantees. This paradigm suited early Internet applications well, such as e-mail and file transfer, that did not have any strict quality of service requirements, and could adapt well to changes in the network load.

The simplicity of the Internet, lack of central administration, and non-proprietary nature, allowed it to grow at unprecedented rates during the past decade. Today the number of users has increased dramatically, and traffic is highly diverse. Applications such as IP telephony, streaming audio, video on demand, video conferencing, and interactive gaming are always used. These applications have different requirements in terms of bandwidth, delay, packet loss and jitter, and require controlled Quality of Service (QoS) levels.

QoS can be described as the differentiated treatment of one or more metrics dependent of the type of application it is being designed for [2]. In other words, the treatment that routing devices apply to traffic streams depends on the characteristics of such

streams. Clearly, the management of QoS is can be extremely complex; this complexity increases when managing devices with different capabilities and commands. One way to ease QoS management is to use policies to control network resources. Such network management philosophy is known as Policy-Based Network Management (PBNM).

PBNM represents a shift in management paradigm; instead of concentrating on each individual device it focuses on the network. PBNM provide mechanisms by which the management and administration process of networks can be largely simplified, automated, and distributed. Today PBNM is viewed by many as the management solution for Next Generation Networks [3, 4]. However, the successful deployment of PBNM solutions depends on a wide range of network protocols and technologies.

The Internet Engineering Task Force (IETF) is working on the standardisation of PBNM technologies such as the Common Open Policy Service protocol for Policy Provisioning (COPS-PR) [5] and its associated data definition language, the Structure of Policy Provisioning Information (SPPI) [6]. The design of COPS-PR addresses some of the shortcomings of the Simple Network Management Protocol (SNMP). Specifically, it can provision more complex configurations, it supports larger message sizes, and provides reliable message delivery.

1.1 Problem Description

The first challenge faced by current PBNM systems is the variety of policy representation forms the system has to deal with. For example, high-level business policies may be specified in XML form and stored in Lightweight Directory Access Protocol (LDAP) directories according to LDAP Schemas. In turn, LDAP directory data may have to be translated to XML or software objects in order to be processed by applications. Lower-level policies may be stored in Management Information Bases (MIB) or Policy Information Bases (PIB) and may have to be encoded using Basic Encoding Rules (BER) to be transferred by COPS-PR. At the other end, received policies may have to be converted again to a parser-friendly format so that they can be processed. This adds unnecessary complexity to the internal structure of PBNM systems and poses serious problems related to the manageability of policy frameworks [7]. Additionally, it also introduces interoperability issues between the various elements of a PBNM system within the same or different management domains.

The second issue currently affecting PBNM solutions is, in fact, the nature of the core protocol and standard used - COPS-PR and SPPI. In particular, COPS-PR was designed to be easily integrated into the Simple Network Management Protocol (SNMP) framework. Therefore it has only minor improvements over SNMP [8] and it is associated with a number issues related to the flexibility and efficiency of its data model and usability of the protocol itself:

- **Management data model:** The data description language employed by COPS-PR (i.e SPPI) does not support the modelling of management information in a hierarchical fashion, using nested data types, which is often desired. For that reason data models used with COPS-PR are not sufficiently flexible and its design is limited to what SPPI can provide. Moreover, the Object Identifier (OID) naming scheme used by COPS-PR is rudimentary, and always contains redundant information. As consequence the protocol do not support easy retrieval and playback of device configurations [9].

- **Protocol usability:** COPS-PR messages are encoded according to BER, a binary encoding scheme. As consequence, messages are not human readable and cannot be consumed by text-processing tools, and the protocol difficult to use, understand, or debug. Moreover, because of the of the limited availability of parsers capable of processing these messages, developers spend more time worrying about the details of the parser instead of working on the application, thus increasing the management application development costs [8].

In fact, RFC3535 [9] provides an overview of a 2002 workshop held by the Internet Architecture Board (IAB) on Network Management. The goal of the workshop was to continue the dialogue between network operators and protocol developers, and to guide the IETF's focus on future work regarding network management. In this document, developers and operators are encouraging the IETF to cut the development of COPS-PR and SPPI modules and spend resources on the development of alternate management technologies:

> "The workshop had rough consensus from the protocol developers that the IETF should not spend resources on COPS-PR development ... they felt that further development of COPS-PR might be a waste of resources as

> they assume that COPS-PR does not really address their requirements ... The workshop had rough consensus from the protocol developers that the IETF should not spend resources on SPPI PIB definitions. The operators had rough consensus that they do not care about SPPI PIBs."

For these reasons, up to this point in time, COPS-PR and SPPI have failed to gain considerable market acceptance [8, 10].

In summary, current PBNM solution are still not feasible particularly because the number of technologies and standards involved add unnecessary complexity to the internal structure of such the systems. Furthermore the core protocol used for QoS provisioning in PBNM solution has serious problems related to flexibility of its data description language and usability of the protocol itself.

1.2 Proposed Solution

During the past years we have experienced an explosion in the use of eXtensible Markup Language (XML) [11] for the representation and exchange of data in Web-based as well as traditional software applications. This, coupled with the maturing of XML companion standards and wide availability of XML processing tools is catalysing research on the application of XML to device and network management [12, 13, 14]. Recently the IETF charted a Working Group with the aim of creating a XML-based Network Configuration protocol (NETCONF) [15]. NETCONF solves most of the issues of SNMP. Specifically, it supports sophisticated data models, it allows atomic transfers, and it offers a programmatic interface.

Although XML seems to be a very promising technology for network management so far it has been applied mostly to device configuration management. However, after reviewing previous work [13, 16, 7] the author comes to the conclusion that XML provides a solution for the problems described in the previous section:

- **Management data model:** Contrary to SPPI, XML has powerful modelling capabilities and allows the design of data models of arbitrary complexity. XML supports seamless representation of hierarchical structures, which can be extremely cumbersome using PIB . Moreover, the naming schemes employed by

XML documents are much more simple, unambiguous and elegant, than OIDs used by SPPI.

- **Protocol usability:** Unlike COPS-PR, NETCONF messages are text-based and therefore human readable. As consequence the protocol is easier to use. Furthermore, because of the widespread support of text- and XML-based processing tools the development costs and deployment cycles of PBNM systems can be reduced by replacing COPS-PR with NETCONF. The widely accepted PBNM architecture proposed by the IETF [17] suggests that any suitable protocol can used for the communication between decision points and enforcement points, thus the use of COPS-PR is not mandatory.

- **Manageability and Interoperability:** XML and companion standards provide an opportunity for unifying PBNM under a common collection of representations and protocols, thus simplifying the internal structure of the system, while increasing flexibility, manageability, and reducing development costs. Additionally, by using XML-based interfaces between components, the PBNM system has improved possibilities for integration with other systems.

1.3 Thesis Objectives

This research aims at using XML technologies in the design and implementation of a PBNM framework in order to improve its usability and manageability, increase its flexibility, and reduce development costs. This study specifically focus on the management of a Differentiated Services network. However, the same principles can be used for the management of other networks or technologies.

This work creates a suitable XML-based policy framework and subdivides it into different abstraction levels, to ease the administration of the network. XML is used throughout the entire policy life cycle, thus improving interoperability and simplicity. Because vendors tend to follow the recent IETF PBNM standards, the author proposes a QoS policy framework based on these standards. Strategies for mapping the IETF policy models to XML-based representations are created. The mapping strategies take into consideration factors such as usability, and flexibility of the resulting XML documents. One major problem faced by XML-based applications is performance.

Therefore when designing QoS policy models careful attention is paid to the impact of the design choices on the performance of the entire system.

This work aims to use NETCONF as a replacement for COPS-PR, as the policy provisioning protocol, and study its impact on usage and performance. Due to the fact that NETCONF was designed specifically for configuration management its communication model does not necessarily suit the needs of a PBNM system. For that reason we have to adapt its communication model to fit into the PBNM framework designed, without modifying the protocol.

The author aims to create an architecture for PBNM comprising of all the standard functional components. The various design principles are tested in a prototype implementation. The performance of the resulting prototype is compared to that of a PBNM prototype using COPS-PR and related technologies. The performance evaluation indicates whether, and under what conditions, the trade-off of clarity, simplicity, and flexibility, for performance, is favourable.

1.4 Scope and Limitations

The principal aim of this study is find ways in which XML can be used to improve interoperability, simplicity, and usability in PBNM systems. Moreover, the author intends to measure of the proposed ideas on the performance of the system. As such, the author does not aim to create a complete management PBNM framework. Furthermore, despite the fact that the principles introduced here can be used in the management of more networks or technologies, the focus of this work is placed solely on Differentiated Services (DiffServ) networks.

Depending on the type of network being managed, several business-level policy models can be created. For example, when managing a enterprise network the focus may be placed on users, accessing applications, running on servers. On the other hand, Internet Service Providers (ISPs) may place the focus on users and billing. For this reason this research work will not attempt to create any business-level policy model. Instead, it will focus on network- and device-level policies.

The PBNM framework created here does not consider cases where the management system provision policies dynamically based on monitoring feedback or external events. Furthermore, the complexity involved in creating generic policy translation algorithms

limits the range of possible ways in which QoS policies can be defined. Nonetheless, the range of QoS policies considered here can be used in the most common differentiated QoS management scenarios.

Finally, the PBNM system proposed here does not incorporate any policy conflict resolution mechanisms. However, this is an import part of PBNM systems, as such it should be addressed in future studies.

1.5 Theses Outline

Chapter 2 motivates the work in this thesis by providing a survey of related research in the fields of XML- and Policy-based Network Management. The chapter starts giving a general description of XML and companion standards, describing the ways in which they can improve general network management applications. It then proceeds by giving an account of research on XML-based network management, focusing on the mapping of data models based on SNMP's Structure of Management Information (SMI) to data models based on XML and on communication protocols based on open web technologies. The second part of Chapter 2 introduces fundamental PBNM principles, placing emphasis on the problems described in Section 1.1. It then proceeds by reviewing research on PBNM with XML technologies, highlighting their strengths and drawbacks.

Chapter 3 presents the design of the proposed policy framework, including the design of the different the policy abstraction layers, the motivations behind the design choices, and the policy translation algorithms used. This chapter also presents the proposed PBNM architecture, including the design principles behind each one of the architectural elements, as well as the interaction between the various architectural components.

In Chapter 4, the author describes the prototype created to evaluate the various principles presented in Chapter 3. It details exactly how the policy framework is implemented, and provides details regarding the implementation of the various architectural components.

Chapter 5 presents the evaluation and performance results. It provides a case study, where the system created is used to manage a small DiffServ network. Furthermore, presents a quantitative performance comparison between the PBNM system created in

this thesis, using NETCONF as the provisioning protocol, and a PBNM system using COPS-PR. Observed parameters are the network traffic created by each scheme, and processing overhead generated at the managed device, including memory consumption and processing delays.

Chapter 6 presents the conclusions drawn from the design, implementation, and evaluation of the proposed XML-driven Policy-based QoS Management framework. It concludes by giving recommendations for future work that should be done in this field.

Chapter 2

Literature Review

2.1 Introduction

This chapter gives an account of the most notable research in the area of XML-based management and XML-driven PBNM. For better understanding, the author describes some of the key XML technologies referenced throughout the thesis as well as basic PBNM principles. The chapter provides the origins and motivations behind the work on this thesis. Section 2.2 gives a brief description of XML and its companion standards focusing on its usage for Network Management (NM). The technologies described are some of the core technologies forming the PBNM framework in this thesis. Section 2.3 presents a review of the most relevant work that proposes to apply XML and its companion standards to NM. . Whereas the previous sections focus on simple network configuration and monitoring, Section 2.5 looks specifically at PBNM. It describes basic PBNM principles and provides a further review of key work applying XML technologies to PBNM.

2.2 XML and Companion Standards

XML is an open general purpose markup language developed by the World Wide Web Consortium (W3C)[1] for documents containing structured information. The design goals for XML include making XML data easily usable over the Internet, supporting

[1]World Wide Web Consortium (W3C): http://www.w3.org

a wide variety of applications, making it simple for programs to process XML data, and making XML documents easy to use and create [11]. The standard has been quickly adopted by industry and today it is a core technology for data definition and interchange across a number of application domains.

The features of XML can be effectively exploited in the area of Network Management. Its primary use is to represent management data as XML documents, which has the following advantages: First, management data can be represented in an open, platform-neutral, vendor-neutral, and language-neutral format [18]. Second, its simple and extensible encoding mechanism allows for the representation of management data of almost arbitrary complexity. Third, XML is self-describing which can simplify the development and management of applications. Fourth, ubiquitous XML processing tools and applications can be used in the development of management applications, thus reducing development costs. Finally, and most importantly, management applications can stand to gain advantage of a number of standards developed by the W3C to complement XML. The following paragraphs describe some of the XML companion standards used in this work, and state how management applications can benefit from them.

2.2.1 Document Type Definitions and XML Schemas

Document Type Definition (DTD) and XML Schema are the most well known modelling languages for XML documents. DTD is the original standard for XML document definition. It defines the vocabulary (element, attributes, etc) and structure of documents. In particular, it specifies the order and quantity of XML elements and the relationship between them. But DTD has many deficiencies [19], thus it does not support complex information models. For that reason another modelling language was created by the W3C. XML Schemas are XML documents therefore they can be processed by standard XML processing tools. XML Schemas provide a comprehensive hierarchy of data-types, support user-defined data-types, support namespaces, and enable schema modularisation and reuse.

XML Schemas provide a more powerful, extensible, and user-friendly modelling mechanisms than SMI or SPPI, thus they can be used for the definition of comprehensive management data models and for validation of documents containing management

data.

2.2.2 DOM and SAX

The Document Object Model (DOM) [20] and the Simple API for XML (SAX) [21] processing are interfaces for accessing and manipulating XML documents. They are the communication path between XML parsers and applications. Parsers using SAX read XML documents sequentially and generate events as they find elements, attributes, or text. This is cumbersome for applications because they are not given an explicit tree that matches the XML document, instead they have to listen to events and determine which element is being processed. On the other hand, DOM is a object-based interface used by parsers to build a tree of objects in memory, containing all the elements of the document being processed. While DOM imposes more memory constraints than SAX, it is a more convenient interface to use because applications manipulate a DOM tree in memory as if they were manipulating the document.

XML-based management applications can benefit from these technologies because not only do they provide a simple and standardised means of accessing and manipulating XML documents containing management information, but also they are platform- and language-independent. Moreover, whereas SNMP or COPS-PR applications have to use specialised tools for data processing, XML-based management applications can benefit from the ubiquity of DOM/SAX parsers; hence simplifying the task of application developers.

2.2.3 XPath, XPointer and XInclude

The XML Path language (XPath) [22] provides a mechanism to address particular parts of an XML document. The syntax of XPath is similar to Unix file paths, starting from the root of the document and listing elements along the way. Elements within an XML document are referenced by specifying the path of the element relative to the element being processed. By using a compact, non-XML syntax XPath expressions can be used inside attributes in XML documents. Therefore, XPath expressions can provide a simplified but powerful addressing mechanism for management data documents used by NM applications.

The XML Pointer (XPointer) [23] specification is partially built on top of the XPath foundation. XPointer is used for the location of XML nodes to be found outside of the referencing document. XPointer is used by the XML Include (XInclude) [24] specification to retrieve and include XML document fragments located at some location other than that of the referencing document (e.g. different file, database or machine). In the NM context XInclude can provide the foundation for the development of data models in a modular fashion by creating complex documents from reusable atomic data items.

2.2.4 XSLT

XML documents only contain information about the structure and semantics of data. In order to view XML documents they need to be converted into a format suitable for presentation. The XML Stylesheet Language Transformation (XSLT) [25] provides a general-purpose translation framework for XML documents. XSLT document are well-formed XML documents that contain rules that guide the translation of XML documents to other formats such as HTML or WML or any other structured textual document format. XSLT works by looking for matching patterns within the original XML document using XPath expressions and applying the desired formatting to the data. Therefore XSLT is useful not only for presentation purposes but also for general document transformation.

In the context of NM, XSLT has two very useful applications. First, it can be used for the presentation of management data to an administrator and for the generation of statistics; and second, it can be used to create other text-based formats. For example, a Management Information Base (MIB) module can be represented in an XML document that is understood by all management devices. But because each managed device may have different configuration languages, each device would then apply a XSLT transformation to the generic XML document representing the MIB. That way different device types can be managed using the same XML document by applying a suitable transformation to it. However this scenario is more attractive in the case of software routers, where one can freely work on the interface to NM applications. This thesis makes use of XSLT transformation for that purpose as well as for the presentation of management data. Chapter 3 provides more details on this.

2.2.5 SOAP

The Simple Object Access Protocol (SOAP) [26] is a widely deployed protocol for the exchange of information over distributed environments. Over the past years SOAP has become one of the most popular message exchange protocol on the Internet partly because it is an integral part of the Web Services infrastructure. The specification allows it to transfer XML-based messages over HTTP or SMTP; however HTTP is the most accepted transport protocol primarily because it is very well established in the Internet infrastructure. The messaging pattern most often used is the Remote Procedure Call (RPC) pattern where a client sends a request and immediately a server sends a response back. In this regards SOAP can be seen as a specialisation of the XML-RPC [27].

The discussion above suggest that SOAP can be used in NM for the interface between a manager and agents, where agents expose an API through which a manager can call operations in a distributed NM environment. On the downside SOAP messages carry more overhead than XML-RPC. For that reason this thesis only uses SOAP for device interfaces at the top architectural layers; Chapter 3 provides more details.

2.3 XML-based Network Management

The previous sections described XML and its companion standards focusing on ways in which they could be applied to NM. This section looks at the most relevant work where these technologies are applied to NM. The section starts by looking at research on the enhancement of SNMP management architecture with XML technologies. It then proceeds by reviewing research focusing on management architectures based solely on XML technologies proposed to replace the traditional SNMP architectures.

2.3.1 Enhancing SNMP with XML

SNMP was devised as a simple means of managing IP networks. For a long time it was the dominant IP NM framework. However, the SNMP framework has many drawbacks; e.g., its poor performance when managing large networks, its poor support for configuration management processes, and a limited and rudimentary set of operations

[12]. Research has shown XML technologies can provide a solution for those shortfalls [28, 13, 29].

Due to the fact that SNMP is still widely used, it is convenient to devise means of reusing the already deployed MIBs and using current XML-based management stations to manage legacy SNMP devices. This section provides an account of the most relevant work on the SNMP-to-XML management data mapping and gateways. It provides a clear evidence of the advantages gained by using XML technologies in NM in general.

Models for Converting SNMP SMI to XML Schema Definitions

In an SNMP framework management information is viewed as a group of managed objects residing in a virtual information store termed Management Information Base (MIB). A MIB consists of a set of related managed objects referred to as MIB modules. MIB modules are defined using the Structure of Management Information (SMI) version 1 (SMIv1) and 2 (SMIv2). SNMP SMI merits from the fact the it is a well established information meta-model [18] with more than 200 MIB modules published by the IETF [30]. As reported in literature, SMI has many drawbacks. Specifically, it has a limited set of data types [18]; it is unable to clearly define models consisting of nested structures; and it has an OID naming scheme which is very impractical [31, 32]. On the other hand XML Schema has powerful modelling capabilities that allow the design of data models of arbitrary complexity. Moreover, XML documents support seamless representation of nested structures and use an XPath naming scheme that is simple, unambiguous and elegant [13, 31]. Thus the first step towards the enhancement of SNMP architectures with XML technologies is the translation of SMI to XML Schema definitions. The following paragraphs describe some of the most commonly used SMI-to-XML mapping schemes.

J. P. Martin-Flatin was amongst the first to propose XML for NM as part of his research on a Web-Based Integrated Management Architecture (WIMA) [33]. He proposed two schemes for mapping of SMI MIB modules to XML format: Model-Level Mapping and Meta Model-Level Mapping. A Model-Level mapping is one in which each MIB module is mapped to a DTD or XML Schema. XML elements are mapped directly from corresponding MIB objects. The following example illustrates this:

```
<Interfaces>
  <ifSpeed type='Gauge32'>100</ifSpeed>
</Interfaces>
```

The advantages of this mapping scheme are that the resulting XML documents are easy to read, easy to parse, and render. Furthermore it gives the developer more validation power because MIB objects are directly mapped to XML elements. On the other hand, each MIB module requires a DTD or XML Schema.

A Meta Model-Level mapping is one in which the DTD or XML Schema gives a generic description of all the MIB modules. In this case, instead of mapping MIB objects to XML element names or attributes, MIB objects are mapped to XML attribute values defined by the DTD or XML Schema. An example of a Meta Model-Level mapping is shown below.

```
<Class name='Interfaces'>
  <Property name='ifSpeed' type='Gauge32'>
    <Value>100</Value>
  </Property>
</Class>
```

The advantages of this scheme are that, as opposed to the former mapping scheme, only one generic DTD or XML Schema is required to describe MIB modules. Unfortunately, the resulting XML documents are difficult to read because they are not intuitive, therefore it is harder to debug and render documents. Furthermore this mapping scheme gives the administrator or developer less validation power so documents are more prone to errors.

These two mapping models are the basis of the most notable SMI to XML mapping proposals found in literature. It should be noted that the same models (or the XML Schemas representing the models) are used for the reverse mapping of XML documents containing management information or messages to SNMP MIB modules or SNMP messages, respectively. The next section presents an account of such research work. The policy data model proposed in this thesis is based on the Model-Level mapping scheme described above. Chapter 3 gives a detailed account of the mapping strategy used for the proposed QoS policy data model.

XML-based Management of SNMP Devices

Yoon et. al. [32] developed an XML-to-SNMP gateway to manage SNMP devices from an XML-based manager, thus taking advantage of the flexibility and expressive power of XML, and reducing time and development costs for management applications. For the mapping of SNMP MIB modules to XML Schema definitions a translation algorithm based on the Meta-Model approach was used. It converts all SMI-based data types to XML Schema definitions and maps each MIB module to an XML Schema definition. The resulting XML documents may represent instance data of a given managed device at a specific point in time. The algorithm has the advantage of retaining all relevant MIB information. On the other hand, because the mapping is driven by SNMP practice, the resulting XML documents have deep hierarchical structures and redundant information.

Strauss and Klie [12, 31] also proposed a variation of the Model-Level mapping scheme described above. In their research they used XML Schemas instead of DTDs to define MIB modules due to the fact that they are more flexible and can represent various types of constraints. Furthermore, because XML Schemas are XML documents they can be processed by XML parsers, thus reducing development periods. Unlike Yoon et. al., they do not make a straightforward mapping of MIB modules to XML Schemas (i.e., deep OID hierarchies are not reflected in XML documents). Instead they represent data in a manner so as to explore the useful features of XML documents, thus making the resulting XML instance documents as convenient for reading and parsing as possible. This is the approach this research, as described in Chapter 3.

The proposals described above use the Meta-Model mapping scheme proposed in [33]. This is primarily due to the fact that although at least one XML Schema has to be generated per MIB module, this approach is more convenient for the reading as well as parsing of XML documents containing management data. Moreover, both proposals use the DOM APIs instead of SAX. This is due to the fact that although SAX is known to perform better in terms of memory consumption [14], DOM provides a more convenient means of manipulating XML documents.

2.3.2 XML-based Management Architectures

XML-to-SNMP gateways are a good way to provide an XML interface to configure and manage deployed devices. However, a more efficient solution is to create new XML engines to run directly on the devices. This section provides an account of the most relevant research in this area.

Ju et. al. [28] presents an architecture for XML-based device management using web servers embedded in the managed devices, for providing management information in XML form. The proposal aims to take advantage of open modern web technologies, such as HTTP and XML, to allow efficient application development and effective management of network devices. Similar to the work described in 2.3.1, XML documents containing management information are transferred as an HTTP payload. However, in this case agents are equipped with an embedded web server that, coupled with a DOM tree and an XPath processor, is capable of providing management information directly in XML form, thus not requiring a SNMP-XML gateway like previous approaches. The management information model used is defined using XML Schema although details concerning the model itself are not provided. This research represents a shift in paradigm in NM due to the fact that an XML-based agent uses a management back-end directly to provide management data and configuration of the device.

Mi-Jung Choi et. al. [14] extended the work carried by Ju et. al. by enhancing the architecture with advanced features. Specifically, a native XML database was used as a central management data repository for long term analysis. Native XML databases are tailored specifically for XML documents. They store the data structured as XML without having to translate it to a relational or object database structure. This is particularly valuable for complex XML structures that would be difficult to map to a more structured database. The database engine queries and processes XML documents directly, thus enhancing productivity. Another module that is added to the manager architecture is an XSL/XSLT processor that generates HTML pages from XML documents on the fly, and provides the user interface.

Hyoun-Mi Choi et. al. [34] carried out a further extension on the work presented by Mi-Jung Choi by enabling devices to communicate via SOAP messages over HTTP. Based on XML and SOAP, an XML-based manager can directly call operations on de-

vices via SOAP RPC. This is a convenient communication approach since management operations can be easily extended. Furthermore, since SOAP is a standard message exchange protocol for the web it opens up possibilities such as re-use of existing standards, ease of software development, and integration with deployed systems. On the other hand SOAP messages carry more overhead than XML/HTTP messages [10].

The work presented above is heavily based on SNMP practice. Specifically, management transactions and data models are based on the SNMP framework. Laurence Menten [13] presents data and transaction models designed specifically to take advantage of the features of XML and related standards. His transaction model is based on operations invoked as RPCs sent in well defined XML documents. Managed devices expose an API through which basic operations (i.e delete, add, replace, and merge) can be executed by managers. Operations use an elegant and unambiguous XPath naming scheme to address management data. The devices communicate through SOAP and HTTP.

A common design challenge faced by XML developers is when to use XML attributes or XML elements. When mapping management data models to XML representations this issue always arises. Laurence Menten [13] proposes to express fields that together make up a unique identifier for a configuration target object as XML attributes. This allows XPath expressions to address configuration targets in a concise way. Furthermore, he maps configuration components that can appear in multiple instances (i.e., "object-like") to XML Elements. And unlike other mapping proposals [31, 32], Laurence Menten maps single valued configuration components to XML attributes on the "object-like" XML elements. This mapping scheme results in XML representations that are easier to read, are more easily searched using XPath, and require less complex XML Schema. Moreover, because less XML elements are used the resulting XML documents generate less traffic. As described in Chapter 3, this thesis employs a similar mapping strategy.

Web-Based Enterprise Management

Web-Based Enterprise Management (WBEM) [35] is a DMTF[2] initiative which provides a set of management and Internet standard technologies developed to unify the

[2]Distributed Management Task Force (DMTF): http://www.dmtf.org

management of distributed environments. WBEM is made up of a management information model - the Common Information Model (CIM); an encoding specification - CIM-XML; and a transport mechanism - CIM over HTTP. CIM is a comprehensive object-oriented model that provides a conceptual view of the entire managed environment. The CIM Schema defines how elements of a managed environment are represented as a collection of objects and relationships among them. The Schema is independent of any platform or storage technology.

CIM-XML provides a specification for the mapping of CIM Schema to an XML definition. Similar to the mapping of SMI to XML presented in 2.3.1, there are two essentially different models for mapping CIM to XML, i.e., Schema Mapping and MetaSchema Mapping [36].

MetaSchema Mapping is one in which the XML schema is used to describe the CIM meta-schema, and both CIM classes and instances are valid XML documents for that schema. CIM element names are mapped to XML attribute or element values, rather than XML element names. When this mapping scheme is employed only one generic XML Schema is required. However, it has limited validation power and the resulting XML documents are not convenient for reading and parsing.

Schema Mapping is one in which the XML Schema is used to describe CIM classes and CIM Instances are mapped to valid XML documents for that schema. XML element names and attributes are taken directly from the corresponding CIM element names and attributes respectively. Although this mapping scheme implies the use of many XML Schemas, it provides a more intuitive representation of CIM in XML. Furthermore, it gives the developer more validation power.

The nature of CIM makes it an attractive technology for the representation of network resources. This thesis makes use of CIM classes to represent QoS policies; the Schema Mapping strategy described above is used to map CIM policy models to XML Schema. Chapter 3 gives a detailed description of the data models used and the employed mapping strategies.

Web Services

Web Services belong to the group of emerging technologies based on XML. Web services are software systems designed to support inter-operable machine-to-machine interaction over a network [37]. A web service has an interface that describes a collection of operations that are accessible via a network through XML messaging. Normally the interface is described using the Web Services Description Language (WSDL). Other systems interact with the web service via SOAP messages conveyed through HTTP.

Recently XML-based management research focused specifically on the use of web services for NM. Some researchers focused on management architectures using web services [38, 39], whereas others studied the performance of web services in a NM environment [40].

Thurm presents a distributed web-based architecture for NM using web services [39]. As described above these web services make use SOAP/XML messages conveyed using HTTP. The author proposes a management architecture consisting of four layers; the user interface layer, the infrastructure services layer, the management services layer, and the network layer. The user interface was designed to support multiple client types (e.g., Java, HTML and WML). The infrastructure services layer provides support functionality such as client adaptation functions, a services registry, and a policy repository. The layer with management services contains the actual management procedures which in turn communicate with devices at the network layer equipped with SNMP agents.

This architecture is robust and benefits from web services in the sense that it can integrate well with other systems and it is location- and client-independent. However the WSDL interfaces used are not standardised and all the "intelligence" of the management system is placed at the management services layer. That is because SNMP only provides primitive "get" and "set" operations.

This thesis uses some of the principles presented by Bernhard Thurm. Specifically, it creates a management architecture consisting of a hierarchy of management entities with the bottom of this hierarchy containing the managed network, the middle consisting of a Policy Decision Point that translates high-level network QoS policies into low-level QoS policies, and the top of the hierarchy consisting of management stations which make use of high-level services provided by the components located at

the middle of the hierarchy. Chapter 3 and 4 gives a comprehensive overview of this subject.

Whereas the work presented above focuses specifically on the design of a management architecture using web services, Pras et. al. conduct an interesting study comparing the performance of SNMP and web services [40]. The the performance metrics used by the authors are bandwidth consumption, system resource usage (CPU time and memory), and network round trip delays. Due to the verbose nature of XML the authors also investigate the effects of compression on the performance of the system.

The authors conclude that for the retrieval of a small number of objects, SNMP is more efficient than web services. In the case where large numbers of objects are retrieved web services become more efficient than SNMP, by using less bandwidth, while incurring comparable round-trip delays. The authors also find that encoding is less expensive than data retrieval, and thus the choice between BER and XML is not the main factor determining performance.

The work by Aiko focuses on web services and SNMP, and achieves encouraging results. Since NETCONF uses the same encoding mechanism as web services (i.e. XML) and COPS-PR uses the same encoding mechanism as SNMP (i.e. BER), this indicates that the approach used in this thesis would get positive performance results. Chapter 5 of this document details the methodology used for the performance study and contains the thesis' performance results.

2.4 Network Configuration Protocol

The previous section presented an account of research on XML-based management architectures. These architecture often make use of proprietary web-based protocols such as HTTP and SOAP to transfer XML documents containing management data. In order for XML-based management architectures to succeed the communication between network entities should be made in a standardised way. The IETF recognised this need and in 2003 it chartered the Network Configuration Working Group (WG)[3] with the aim of developing an XML-based network configuration protocol (NETCONF) [15]. Since then NETCONF has gained widespread acceptance in the operator community [41].

[3]Network Configuration Charter: http://www.ietf.org/html.charters/netconf-charter.html

NETCONF uses an RPC mechanism to expose a formal Application Programming Interface (API). Managers and agents communicate with each other by sending and receiving RPCs encoded in XML. The characteristics of the protocol are as follows:

- A NETCONF session is the logical connection between a network administrator or network configuration application and a network device. A device must support at least one NETCONF session.
- Management information is separated into two classes, i.e., configuration data and state data. Configuration data are writable while state data are read-only and represent network device statistics.
- NETCONF is a connection-oriented protocol which requires a persistent connection between the manager and the agent although it is not bound to any particular transport protocol.
- Its connections must provide authentication, data integrity, and privacy. NETCONF depends on the transport protocol for this capability.

NETCONF can be conceptually separated into 4 layers: content layer, operations layer, RPC layer and transport layer. The transport protocol layer provides a communication path between the client and server. It must be connection oriented requiring a long-lived and persistent connection between the manager and the agent. NETCONF is defined to be transport independent, allowing mappings to be defined for multiple transport protocols as long as they comply with the transport requirements of the protocol. The WG has selected SSH to transport NETCONF messages because it is widely deployed and provides security and confidentiality. However, SOAP over HTTP and (Blocks Extensible Exchange Protocol) BEEP may also be used to transport NETCONF messages.

The RPC layer provides a simple, transport-independent framing mechanism for encoding RPCs. XML messages are defined by using RPC elements *<rpc>* and *<rpc-reply>*. NETCONF peers use these elements to frame requests and responses.

The operations layer defines a small set of base operations invoked as RPC methods with XML-encoded parameters. These operations provide the ability to write, retrieve, copy, edit and delete management information in the agent. NETCONF operations

are executed against configuration datastores, which are complete sets of configuration data.

The content layer presents configuration data and is outside of the scope of the protocol and depends on a particular NETCONF implementation. This thesis creates a DiffServ data model (based on the DiffServ PIB) to be used with NETCONF. The protocol has a locking mechanism that ensures that only one manager updates configuration of a managed device at any given time.

Although NETCONF was designed specifically for network configuration, some research has suggested that it can also be used for PBNM provisioning [10]. This thesis creates a PBNM architecture using NETCONF as the provisioning protocol replacing the Common Open Policy Service for Policy Provisioning (COPS-PR).

The next section describes the IETF PBNM framework including COPS-PR, and it gives an account of research on XML-driven PBNM, which is the focus of this thesis.

2.5 Policy-Based Network Management

The previous sections looked the application of XML technologies to simple network configuration and monitoring. These management architectures do not suite the need of QoS-enabled networks consisting of a wide array of devices that are configured in many different ways. This may be the case of Differentiated Services (DiffServ) networks where the setup and configuration of hundreds of devices can be daunting. For those case it is desirable to have a management system that is largely automated and distributed.

PBNM is a paradigm that allows for distributed and automated management of large networks. Unlike the management architecture described in the previous sections PBNM architectures focus on the network as opposed to the device. This simplifies the configuration and administration of networks. Figure 2.1 depicts the nature of PBNM systems. Network administrators define abstract policies using informal business terms and language and feed them into the PBNM system. The PBNM system translates the abstract policies into technical network configurations and distributes them to the target devices, therefore simplifying the work of the network administrator.

Careful design of XML-driven PBNM architectures can bring about great benefits,

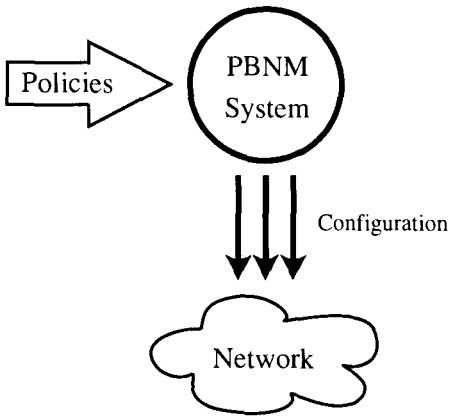

Figure 2.1: PBNM philosophy

as shown by research described in Section 2.5.4. This is in line with the aim of this thesis. In order to improve current PBNM architectures it is important to study them carefully. There are currently a number of efforts concentrating on the development of PBNM standards [2]; however the subsequent sections focus on the IETF PBNM architecture and models, since these models are well established and accepted by the research community as well as industry.

2.5.1 Key concepts

Policy

RFC3198 [42] defines a policy as "a definite goal, course or method of action to guide and determine present and future decisions". Policies are implemented or executed within a particular context. In network terms a "Policy is a set of rules to administer, manage, and control access to network resources". This thesis uses the latter definition. In general a basic rule is formed by a *<condition, action>* pair that defines an *action* to be executed when a *condition* is met. PBNM allows for automated configuration of network devices through the use of policies.

Policy Roles

The concept of *Role* is central to the design of a PBNM framework. A policy role is associated with one or more policy rules. Each managed resource is assigned one or more *roles*, thus the applicability of a policy to a managed resource is specified by assigning a *role* to it. The use of roles facilitates the selection of one or more policies from a larger set of available policies [43]. If a network administrator wants to change network behaviour, he/she simply needs to update a policy associated with a *role*, and the policy framework will perform the necessary policy updates to all managed resources playing that *role*.

As an example, in a DiffServ network, interfaces on edge routers may play an "edgeRouter" role, while interfaces on core routers may play a "coreRouter" role. The network administrator creates policies which implement Per-Hop Behaviours (PHB) suitable for edge and core routers. PHBs can be defined as the treatment applied to a packet when it is transversing a hop (i.e., router). Policies which implement edge PHBs have a "edgeRouter" attribute, whereas policies which implement PHBs suitable for core routers have a "coreRouter" attribute.

2.5.2 Policy Information and Data Models

Policy information models model the concept of policy and make it possible for users to define policies that guide the operation of a network. Normally these information models are platform- and technology-independent. On the other hand, configuration policy data models are used at a lower level of abstraction and normally contain technology-specific concepts. Although there are many proposals of policy information and data models the models proposed by IETF and the Distributed Management Task Force (DMTF) are the most widely used.

The IETF and the DMTF have jointly developed object oriented models for policy representation, namely the Policy Core Information Model (PCIM) [44] and its extensions (PCIMe) [43]. These models are extensions of CIM, described in Section 2.3.2. PCIM and PCIMe are vendor- and device-independent models that define the generic structure of a policy and can be extended to enable developers and administrators to define policies of different types. The Policy QoS Information Model (QPIM) [45] is a further extension of PCIM and PCIMe designed specifically for the representation of

QoS policies. These information models are designed to be independent of any storage technology. This thesis focuses on the mapping of these technologies to XML formats that are self-describing, human- and machine-readable, easy to process, and portable. Chapter 3 describes the strategies developed for the mapping of the models to XML representations.

PCIM and PCIMe may be extended to create business-level policies [46, 16]. On the other hand, QPIM may be used for the modelling of network-level QoS configuration policies focusing on the network as a whole. Network-level QoS configuration policies are normally translated to low-level device-specific policies based on the DiffServ PIB [47] or the Information model for Describing Network Device QoS Datapath Mechanisms (QDDIM) [48].

QDDIM is an effort from the IETF and the DMTF designed to be used with QPIM to model policies to manage and configure QoS mechanisms (i.e., classification, marking, metering, dropping, queueing, and scheduling functionality) of devices. Together, these two documents (QPIM and QDDIM) describe how to write QoS policy rules to configure and manage the QoS mechanisms present in the datapaths of devices [48]. On the other hand, the DiffServ PIB is a data model that describes a policy information base for devices implementing the DiffServ architecture [49]. The provisioning classes defined in the DiffServ PIB provide policy control over resources implementing the DiffServ architecture. These provisioning classes can be used to provide a comprehensive policy controlled mapping of service requirement to device resource capability and usage [47].

various abstraction levels, olicy trsnaltion

2.5.3 PBNM Architectural Framework

Within the IETF a PBNM architecture was initially proposed as part of a framework for providing policy-based admission control decisions in the context of the Integrated Services model. However it was soon recognised that those approaches could also be applied in other areas of NM.

The IETF PBNM architecture is depicted in Figure 2.2. It includes four functional blocks: a Policy Management Station (PMS), a Policy Repository (PR), a Policy Decision Point (PDP), and a Policy Enforcement Point (PEP). The PMS provides an

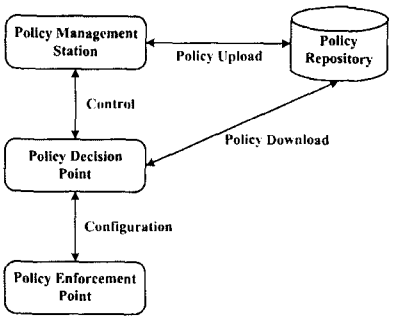

Figure 2.2: IETF PBNM Architecture

interface to create, edit, and store policies in a repository. It is also used to monitor the status of the PBNM environment.

The policy repository stores policies according to an information model. Stored policies can be retrieved by PDPs by means of a repository access protocol. IETF recommends the Lightweight Directory Access Protocol (LDAP) for repository access even though other mechanisms such as HTTP, SNMP or SSH are possible.

The PDP is the central component of the system, it is where policy decisions are made. This entity retrieves policies, interprets them, detects policy conflicts, receives policy requests from PEPs and returns policy decisions to them. Policy decisions are triggered by special events, polling or explicit requests.

PEPs are responsible for enforcing policy decisions made by PDPs. A PEP may or may not host the target elements to which actions are applied to. According to the IETF framework one PDP is responsible for multiple PEPs although they may also be combined and co-located. This framework defines the Common Open Policy Service (COPS) as the protocol for policy exchange between the PDP and PEP. Nonetheless other protocols such as HTTP, FTP or SNMP may be used.

Common Open Policy Service for Policy Provisioning

The Common Open Policy Service is an attempt by the IETF to standardise communication between a policy server and client. COPS-PR is an extension of COPS for policy provisioning. COPS-PR allows for efficient transport of attributes, large atomic transactions of data, and flexible error reporting. It follows a request/response model where the PEP is the client and the PDP is the server. It has a single connection

between the PDP and PEP per area of policy control (e.g., DiffServ), which guarantees that only one server updates a client's policy configuration at any given time. Furthermore, COPS-PR uses reliable TCP transport, and uses a state synchronisation mechanism so that the PDP and PEP only have to exchange differential updates.

Normally the PEP sends a configuration request to the PDP, describing itself, its configurable parameters, and the default configuration. In response, the PDP sends all the relevant configuration policies to that PEP. After receiving the provisioned policies, the PEP maps them into its local QoS mechanisms, and installs them. If conditions change at the PDP or if it receives an external event, it sends the changes in configuration policy to the PEP. The PEP then updates its local configuration appropriately.

COPS-PR carries data that conforms to a Policy Information Base. The PIB can be seen as a conceptual tree where the branches of the tree represent structures of data or Provisioning Classes (PRC), and the leaves represent various instantiations of these classes (PRI). PRIs are unequivocally identified using a Provisioning Instance Identifier (PRID). Usually Object Identifiers (OID) are assigned to PRCs and PRIDs[4]. This Object Identifier naming scheme is rudimentary, and contains redundant information. As a consequence COPS-PR does not support easy retrieval and playback of device configurations [9].

PIBs are defined using SPPI, which is similar to SNMP's SMI. Like SMI, SPPI does not support the modelling of management information in a hierarchical fashion, using nested data types, which is often desired. For that reason data models used with COPS-PR are not sufficiently flexible and its design is limited to what SPPI can provide.

PIB data is transferred by COPS-PR by applying a BER encoding scheme (a binary encoding scheme) and encapsulating it in request (by PEPs) and decision (by PDPs) messages. As consequence the protocol messages are not human readable and cannot be consumed by text-processing tools. This makes it difficult to debug erroneous applications. Moreover, because of the of the limited availability of parsers capable of processing these messages, developers spend more time worrying about the details of the parser instead of working on the application, thus increasing the management

[4]PRCs can be seen as tables and PRIs can be seen as table rows, where PRIDs identify each table row

application development cycles.

The Differentiated Services PIB and the Framework PIB

The DiffServ PIB defines policy provisioning classes used to control resources implementing the DiffServ architecture. The PIB is designed according to the Differentiated Services Informal Management Model [50], which describes Traffic Control Blocks (TCB) consisting of various datapath elements. These datapath elements can be assembled to realise the traffic conditioning and Per-Hop-Behaviours (PHB) defined by the DiffServ architecture.

The DiffServ PIB classes are divided into two groups: *dsPolicyClasses* and *dsCapabilityClasses*. *dsPolicyClasses* are used to define the TCBs that realise the required traffic conditioning and PHBs. On the other hand *dsCapabilityClasses* describe the capabilities and limitations of the interfaces on the device. The DiffServ PIB is usually used along with the Framework PIB. This is a more generic PIB that is common to all policy clients and provides mechanisms to associate policy roles and capabilities to interfaces. Furthermore, it also defines a number of classes for packet classification and marking.

2.5.4 XML-driven PBNM

The previous sections presented a brief review of the IETF PBNM framework. This section proceeds by presenting an account of relevant research focusing of the design and development of XML-driven PBNM architectures. Some of the principles presented in Chapter 3 are based on the ideas described here.

XML-based Policy Modelling

This section gives an account of research on XML-driven PBNM and it places focus on the modelling and representation of network configuration policies using XML technologies.

Beller and Jamhour [16] proposes a framework for defining reusable business-level policies for a DiffServ network using XML for policy representation. High-level policies are designed based on PCIM, following a modular approach to increase re-usability.

That is done by using XPointers to locate reusable policy fragments placed in a reusable policy container. Although this is a good approach, the author feels that using the XInclude standard would yield a better result, since it would automate the parsing and inclusion of re-usable fragments. This is the approach followed this thesis.

Beller and Jamhour design low-level policies according to the DiffServ PIB. These DiffServ policies are also represented in XML. The authors propose to use COPS-PR to provision low-level policies. High-level configuration policies in XML format are translated to DiffServ policies also in XML format, which are converted to binary format in order to be transported by COPS-PR. At the receiver end there is another conversion back to XML in order for the device to process the provisioned policies. This procedure adds unnecessary complexity to the system.

Clemente et. al. [7] propose a similar PBNM architecture focusing on IP Security (IPSec). XML is used for representation of policies during their entire life cycle, therefore simplifying the internal structure of the PBNM system and improving the manageability of the system. PCIM is used to model high-level policies whereas lower-level policies are modelled based on the IPSec PIB. A Model-Level mapping scheme described in Section 2.3 is used for lower-level policy mapping. The authors choose to map PIB attributes to XML elements instead of attributes, resulting in XML documents that are more verbose but at times easier to read. Like Beller et. al., the COPS-PR is used for low-level policy provisioning. However, the low-level policies are transported in XML format resulting in a system that is easier to develop, although the data model has to follow a more strict structure in order to be supported by COPS-PR.

XML-based Policy Provisioning Architectures

Most of the proposals mentioned above only partially solve the problems stated in Chapter 1. This is primarily because although XML is used for policy representation, the message exchange protocol used is COPS-PR; and as described in Section 2.5.3 this protocol has a number of shortfalls. The following paragraphs review research which recognises the potential of using XML-based protocols for PBNM.

Franco et. al. presents a PBNM architecture with two policy provisioning models, based on NETCONF and SOAP [10]. In the first scheme, NETCONF is used to pro-

vision XML-based DiffServ policies automatically derived from the DiffServ PIB using the *libsmi* library [51] [5]. The second provisioning scheme uses SOAP to provision DiffServ policies. However, in this case the DiffServ PIB is translated to C++ objects. These objects are then sent to PEPs using the SOAP framework. The PBNM framework defined in this thesis uses an approach similar to the NETCONF scheme proposed by Franco et. al.. However the DiffServ PIB is not mapped to an XML Schema in a direct fashion; instead the mapping is made in a way that to explores the useful features of XML documents, thus making the resulting XML instance documents as convenient for reading and parsing as possible.

As part of the same work, Franco et. al. [10] compare the performance of the two proposed XML-based provisioning schemes to a COPS-PR scheme. The study shows that, as compared to COPS-PR, both schemes waste more bandwidth. An interesting finding is that if compression is used for XML messages both schemes consume less bandwidth than COPS-PR. The performance evaluation methodology used in this thesis is partially based on the study by Franco el. al..

Torsten Klie and Lars Wolf propose a PBNM architecture based on web service composition [52]. The policy model proposed has two policy abstraction layers representing high-level policies or services and low-level policies or services. High-level policies are a combination of two or more low level services exposed by the different devices located at the device layer. Web service composition is used by a policy engine to create high-level services from low-level services. The authors mention NETCONF as one of the supported protocols at the device layer. Other proposed or proprietary schemes may be supported by the addition of conversion gateways. This work is still in its early stages therefore the proposed design is still very generic.

2.6 Discussion

As demonstrated in Section 2.3 the use of XML in general device or network management brings about many advantages such as improved flexibility, completeness, and robustness of the management infrastructure. Most of the research on XML-based NM

[5] *libsmi* is software library (developed at the Institute of Operating Systems and Computer Networks of the Technical University Carolo-Wilhelmina at Brunswick) capable of translating SMI MIB and SPPI PIB modules to other languages, such as Java, CORBA, UML and C.

has focused on simple device configuration architectures. However these NM architectures are not sufficient to manage large QoS-enabled networks, which require more efficient management architectures.

Policy-Based NM (PBNM) is a management paradigm that provides mechanisms to simplify and automate the management and administration of networks, therefore reducing development and administration costs. However, to develop or deploy current PBNM solutions one has to master a wide array of technologies. The amount of technologies involved adds unnecessary complexity to the internal structure of PBNM systems, and pose serious problems concerning the manageability of policy frameworks [7]. Furthermore, the COPS-PR protocol traditionally used for communication between PBNM entities is associated with a number of issues related to the flexibility and efficiency of its data model and usability of the protocol.

The application of XML technologies to PBNM provides an opportunity for the unification PBNM under a common collection of representations and protocols, thus simplifying the internal structure of the system, while increasing flexibility, manageability, and reducing development costs. At same time by using an XML-based protocol for policy provisioning the problems introduced by the use of COPS-PR are solved. In fact, some researchers have suggested that NETCONF, which is an XML-base network configuration protocol, can successfully replace COPS-PR in PBNM architectures [10].

It is clear that most of the research on XML-based NM has focused on simple device configuration architectures. Regardless of this, those design principles can also be applied in the development of more advanced management architectures such PBNM architectures, as described in the previous section.

The next chapter presents an XML-driven PBNM architecture that uses XML for the modelling and representation of configuration policies throughout the entire policy life-cycle. The modelling of policies is based on the principles presented in previous sections. The proposed PBNM architecture uses NETCONF for policy provisioning as a replacement for COPS-PR.

Chapter 3

Proposed XML-Driven PBNM Framework

3.1 Introduction

The concept of PBNM has emerged as a way to simplify, automate, and distribute the management of large networks. Network administrators simply have to define policies at a high-level of abstraction and the PBNM system performs the complex device configuration tasks. Specifically, the PBNM system translates abstract policies into technical configurations and distributes them to the network devices to which the policies apply. Normally PBNM systems are complex and consist of many different entities and technologies. For that reason the development and management of a PBNM solution is non-trivial.

This thesis aim to simplify the development and administration of PBNM solutions by using XML technologies to model and represent policies throughout the entire policy life-cycle. Moreover XML serves as basis for all the information exchange protocols used. Therefore this work aims to unify PBNM under a common collection of representations and protocols, thus simplifying the internal structure of the system, while increasing flexibility, manageability, and reducing the system development and deployment costs.

The PBNM solution proposed in this thesis consists of a policy framework and PBNM architecture. The policy framework defines the way in which policies at various ab-

straction level are represented, and how the high-level policies are translated into low-level policies. The PBNM architecture consists of a number of logical or physical network entities and standard communications protocols between them.

3.2 Policy Framework

PBNM solutions close the gap between the definition of abstract high-level policies to the actual configuration of network devices. Network administrators are not required to know technical details regarding the network, they simply define policies at a high level of abstraction and the PBNM system translates them into configuration commands. The policy framework defines a policy hierarchy consisting of various abstraction levels. It defines how different policy information models are mapped to XML formats and how policies are translated by the PBNM system. Figure 3.1 depicts the policy framework proposed in this thesis.

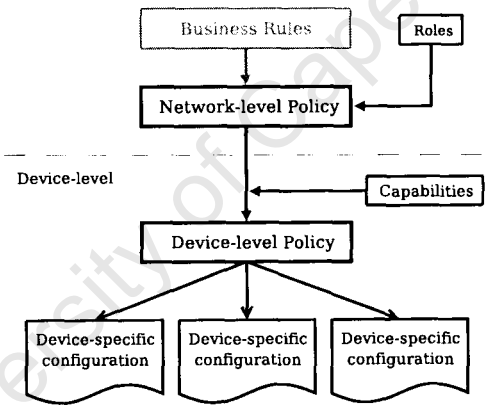

Figure 3.1: Policy framework.

A Business-level policy is defined in terms of a language closer to the business needs of an organisation rather than in terms of the specific technology needed to deploy the policy [46]. The PBNM system converts business-level policies into technical network configuration policies which are ultimately converted into device-specific commands. Therefore, the network adapts automatically to the defined business-level policy, thereby simplifying the job of the network administrator. This thesis addresses

the highlighted portion of Figure 3.1, namely Network-Level Policies (NLP), Device-Level Policies (DLP) and device-specific configuration policies.

NLPs represent generic network operational and configuration information that is independent of any specific device and general QoS mechanisms (such as the type of queue management strategy employed). They are used to express the requirements of the different applications, and determine which applications get preferential treatment when the network is congested. At this level policies control "network" QoS resources instead of controlling individual network element configurations. As opposed to business policies that are expressed in human-friendly language, NLPs are expressed in network-specific terminology.

Positioned at the bottom of the policy continuum are DLPs and device-specific configuration policies. DLPs are generalised QoS configuration policies that are independent of any specific device's implementation technology, but dependent of the capabilities of each device (for example the set of QoS mechanisms supported). This allows for devices made by different vendors or even devices made by the same vendor running different software to be controlled using a single uniform data model.

Ultimately DLPs have to be translated to device-specific configuration policies that depend on the device's implementation technology. For example two DiffServ-enabled software routers implemented using the Click Modular Router and the Linux Traffic Control respectively, support different configuration mechanisms. However the same device-independent policies may be used to control both routers, provided that the appropriate translations are made at the routers.

A typical management scenario would require the execution of the following steps:

1. An abstract QoS policy would be formulated in a language-neutral terminology.
2. The network administrator analyses the network topology and determines the roles of the various device interfaces.
3. He/She then creates a formal representation of the abstract policy following the NLP model.
4. "Roles" are assigned to the NLPs, matching the roles assigned to device interfaces. For example a group of policies that controls resources on a device

interface located at the core of the network should be assigned a "core" role, matching the "core" device interfaces.

5. The PBNM system translates all NLPs to DLPs according to the capabilities of each device.

6. Each device translates the DLPs to suitable device-specific configuration commands needed to enforce the policies.

Representing policies at different levels of abstraction using different representation formats would complicate the management and maintenance of the policy framework. Therefore using a common XML format to represent policies during the entire policy life-cycle eliminates that complexity. Furthermore the use of XML makes the policy framework highly portable and increases the chances for integration with other systems. In this work policies at various abstraction levels are mapped to XML and XML Schemas. Therefore the policy syntax checking functionality is done intrinsically by the XML parser through the validation of the XML policy against its XML schema. The following sections detail the implementation of XML policy at the various abstraction levels.

3.2.1 Network-Level Policies

NLPs are QoS policies modelled based on PCIM/PCIMe and QPIM. As the name suggest these models are all extensions of CIM. PCIM and PCIMe define the generic structure of a policy and can be extended to enable developers and administrators to define policies of different types. QPIM is an extension of PCIM and PCIMe designed specifically for the representation of QoS policies. Some of their strengths are that they allow for hierarchical policy organisation and rule re-usability. The organisation of policies hierarchically reflects the way humans think about complex policy. Moreover, it allows policy to mirror most administrative organisations [45]. Rule re-usability enables complex policy rules to be constructed from multiple simpler rules enhancing the manageability and scalability of the policy framework.

Figure 3.2 depicts a simplified version of the PCIM/PCIMe model used in this work. Structural class *PolicyRule* is central to the model. It aggregates *PolicyActions* and

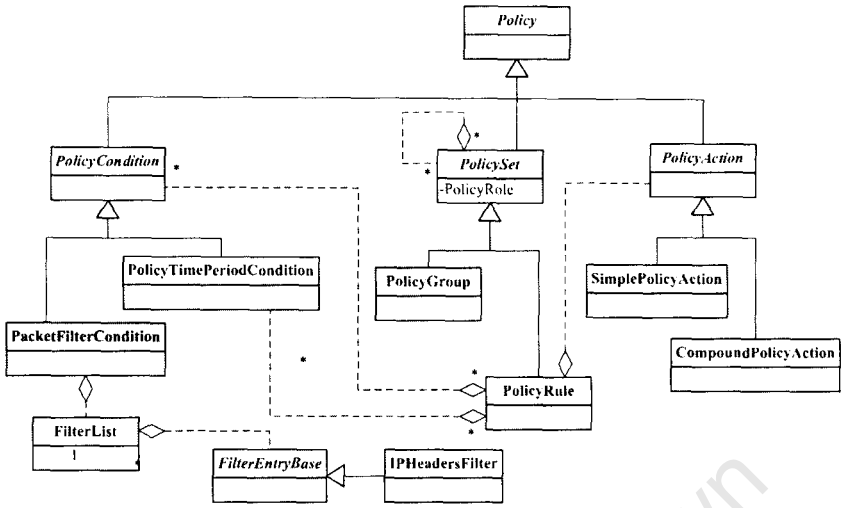

Figure 3.2: Simplified representation of PCIM/PCIMe.

PolicyConditions through the *PolicyActionInPolicyRule* and *PolicyConditionInPolicyRule* aggregations respectively. *PolicyCondition* is used to formulate conditions that must be met in order for *PolicyActions* to be executed. PCIMe defines the class *SimplePolicyCondition* which aggregates *PolicyVariables* and *PolicyValues* used for condition evaluation. However this method seems at times cumbersome since a simple condition evaluation involves many classes. Therefore a class *PacketFilterCondition* (highlighted in Figure 3.2) is added to the model to simplify packet filtering in policy condition. *PacketFilterCondition* aggregates *FilterLists* through the added *FilterListInPacketCondition* aggregation. *FilterList* may aggregate various types of filter classes. This work only uses the *IPHeadersFilter* which contains the most often used properties for filtering IP, UDP, and or TCP headers.

Abstract class *PolicyAction* is the base class of all actions. In this thesis *PolicyActions* specify parameters such as specific Code Points and the Per-Hop-Behaviour (PHB), profile characteristics and treatment of the traffic for those Code Points [42]. QoS Policy actions are defined according to QPIM. Figure 3.3 illustrates a simplified version of QPIM used in this work. Because this study focuses specifically on DiffServ-enabled IP networks, only classes used for DiffServ policy definition are depicted.

Although QPIM and PCIM/PCIMe are demonstrated separately they are in fact combined in a single information model. The main classes of QPIM are classes de-

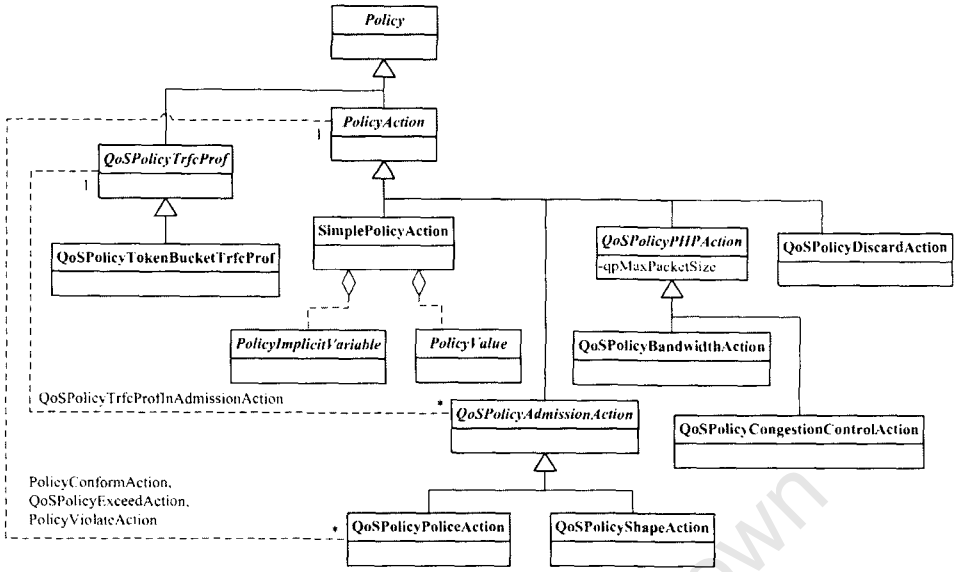

Figure 3.3: Simplified representation of QPIM.

rived from *PolicyAction* modelling predefined actions (*SimplePolicyAction*, *QoSPolicyBandwidthAction*, *QoSPolicyCongestionControlAction*, and *QoSPolicyDiscardAction*), classes modelling traffic profiles (*QoSProlicyTokenBucketTrfcProf*) and classes performing admission decisions.

As mentioned before these models provide for information re-usability and rule nesting. Re-usability is enabled by allowing all structural classes inherited from abstract class Policy to be included in a *ReusablePolicyRepository* using the *ReusablePolicy* association. Support for rule nesting is provided by the aggregation *PolicySetComponent*, depicted in Figure 3.2 by the aggregation line linking *PolicySet* to itself. It allows for *PolicyRules* or *PolicyGroups* to aggregate other rules or group of rules.

Mapping Strategy

PCIM/PCIMe and QPIM are independent of storage technology. Therefore they have to be mapped to a format that depends on the storage technology used. There are many benefits of mapping these models to XML formats. First, XML is known for its ability to represent data of almost arbitrary complexity, easily enabling data nesting and re-usability. Second, XML documents are highly portable. Third, there are many

tools available to process XML. Fourth, XML documents are human-readable and can be processed by text-processing tools.

As described in the previous chapter there are two possible mapping schemes: Schema Mapping and Meta-schema Mapping [53]. In this work we opted for the schema mapping since it provides a more intuitive representation of PCIM/PCIMe and QPIM in XML. Moreover, because XML Schema is used to describe CIM classes, this mapping scheme gives the user more validation power.

The mapping of CIM classes is straight forward. On the other hand the mapping of CIM association is more tricky because of the need to provide for data re-usability. The mapping strategy adopted in this work is as follows.

- CIM structural classes:
  - There is a one-to-one mapping of CIM structural classes to XML elements.
  - CIM class attributes are mapped to the corresponding XML element attributes.
  - Only concrete CIM classes are mapped.
  - Abstract classes are omitted and their attributes are inherited by the closest concrete descendant class.
  - Re-usable classes are placed in a re-usable policy container.
- CIM associations:
  1. Association not involving reusable information:
     - In this case the association class is not represented.
     - Its attributes are mapped to the structural class representing the *Dependent* or *PartComponent* end of the association or aggregation respectively.
     - The structural class in the previous step is mapped to an XML element that has as a parent, the XML element resulting from the mapping of the structural class representing the the *Antecedent* or *GroupComponent* end of the association or aggregation, respectively.
  2. Association involving reusable information:

- The association class is explicitly mapped to an XML element, which has as a parent the XML element resulting from the mapping of the structural class representing the the *Antecedent* or *GroupComponent* end of the association.
- The structural class in the end of the association or aggregation representing the *Dependent* or *PartComponent* is mapped to an XInclude element which points to the reusable XML element. The XInclude element has as a parent the XML element that resulted from the mapping of the association in the previous step.

Support for information re-usability is provided by XInclude elements. The advantage of using XInclude is that it allows for policies to be created in a modular fashion. Furthermore, some native XML databases already come equipped with an XInclude processor that replaces XInclude elements with the XML fragments they point to, making it very easy to deploy solutions that benefit from XInclude. The XML Schema derived from this mapping strategy is used to validate documents before the processing of XInclude elements.

Figure 3.4[1] depicts a fragment of the XML schema that resulted from the mapping of the information models above. The element *PolicySetComponent* was mapped from the aggregation of the same name and it enables re-use of information. As described in the mapping strategy it contains an XInclude element that resulted from the mapping of the structural class in the *PartComponent* end of the *PolicySetComponent* aggregation. The elements *PolicyRule* and *PolicyGroup* represent the mapping of aggregations that do not involve re-usable information. Therefore the structural classes were mapped and the aggregations were omitted.

As Figure 3.4 demonstrates XML allows for the seamless nesting of rules. In this case a *PolicyGroup* may contain further *PolicyGroups* or *PolicyRules*. In this way a policy may be very easily organised hierarchically, constructing complex rules from simple ones. This also applies for *PolicyActions*, as illustrated in Figure 3.5 and *PolicyConditons*. In this case a *CompoundPolicyAction* allows for the atomic execution of a compound action made up of more simple policy actions.

[1] Although the figure represents an XML Schema, XML document terminology is used to describe it in order to ease understanding.

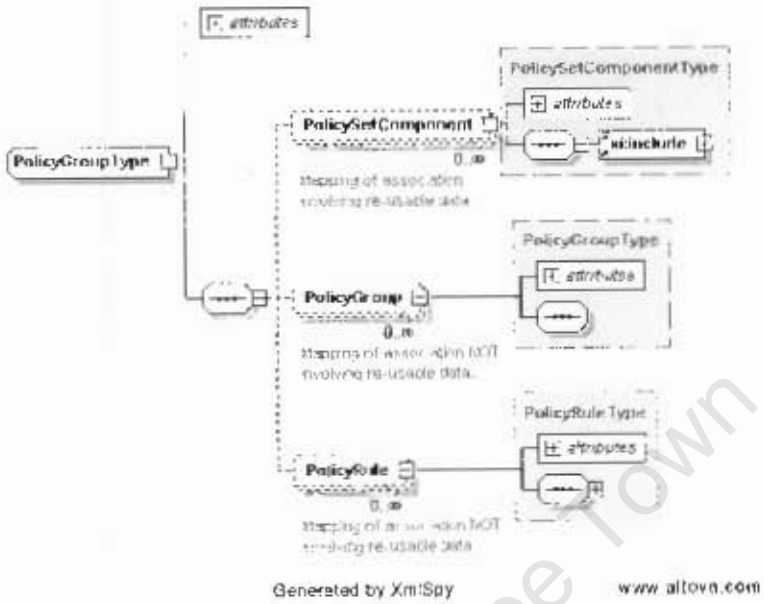

Figure 3.4: Fragment of an XML Schema depicting the mapping of *PolicyGroup*.

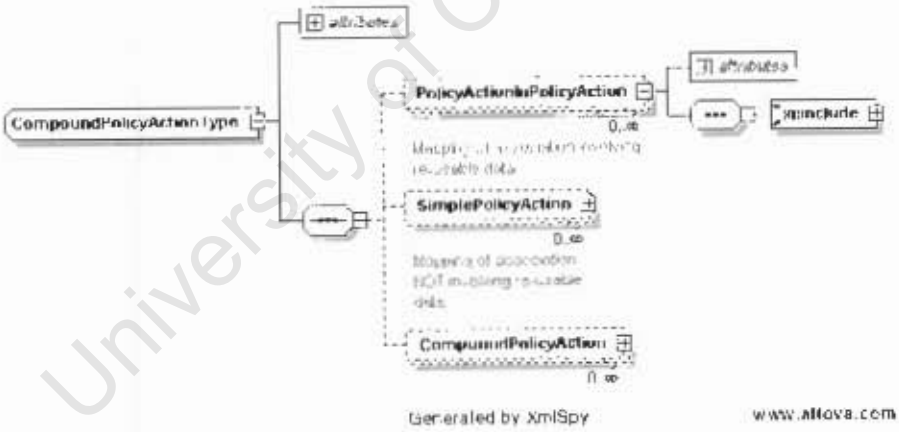

Figure 3.5: Fragment of an XML Schema depicting the mapping of *CompoundPolicyAction*.

Appendix B contains examples that illustrate the usage of an NLP described in this section.

3.2.2 Device-Level Policies

DLPs are specific for a given device. The PBNM system translates NLPs to DLPs based on the capabilities supplied by each device. Whereas NLPs provide a common level of abstraction required to manage the network as a whole, DLPs provide adequate policies for each device based on the device's capabilities. In this work DLPs are modelled based on the DiffServ PIB and Framework PIB[2]. We opted to use the DiffServ PIB, in favour of QDDIM (Information model for Describing Network Device QoS Datapath Mechanisms)[48] because of its simplicity and acceptance.

Mapping Strategy

As described in Chapter 2 there are two possible schemes for mapping SPPI-based PIB modules, such as the DiffServ and Framework PIBs, to XML Schema definitions: the model-level mapping scheme and the metamodel-level mapping scheme. This thesis employs the model-level mapping because it provides a more intuitive representation of the PIB in XML, and it gives more validation power. The mapping strategy is as follows.

- Each PRC is mapped to a generic XML element container and instances of the PRC (i.e. PRIs) are mapped to child XML elements grouped within the PRC container.
- All PRIs have unique names which may be a unique string or the PRID.
- PRI parameters are mapped to XML attributes (as opposed to child elements [7]) to make the resulting XML documents less verbose.
- As an exception, PRI parameters that refer to other PRIs are mapped to child XML elements. This is to facilitate the manipulation of documents and to make the documents more convenient for reading. These elements have an attribute Pointer, which has as value, an XPath expression locating the desired PRI.

[2]Because DLPs are modelled based on the DiffServ PIB the terms "Device-Level Policy" or DLP and "DiffServ Policy" will be used interchangeably from this point onwards.

- Namespaces are used to identify PIB modules. Therefore all elements belonging to a certain PIB modules are defined in the corresponding namespace. Each namespace and PIB module is defined in a separate XML Schema.
- The cases where PRCs are extended with other PRCs are represented by XML element nesting.
- XML documents or portions of XML documents containing configurations have a root element <Pib> which may contain a number of PIB modules. These PIB modules are identified by namespaces.

It should be noted that the hierarchical feature of XML documents (where parent XML elements have child XML elements) is not adopted to represent the relationship between datapath elements. Instead documents make use of XPath expressions to locate upstream datapath elements. In this way datapath elements can be re-used freely without having to carry out complex processing. Moreover, this avoids having XML documents with deep nesting structures.

Figure 3.6 depicts a fragment of the XML Schema that resulted from the mapping scheme described above. All the PRCs descend from *PolicyClasses* and PRIs of the same type are grouped within containers. Notice that each traffic control block (TCB) has only one *DataPathTable*.

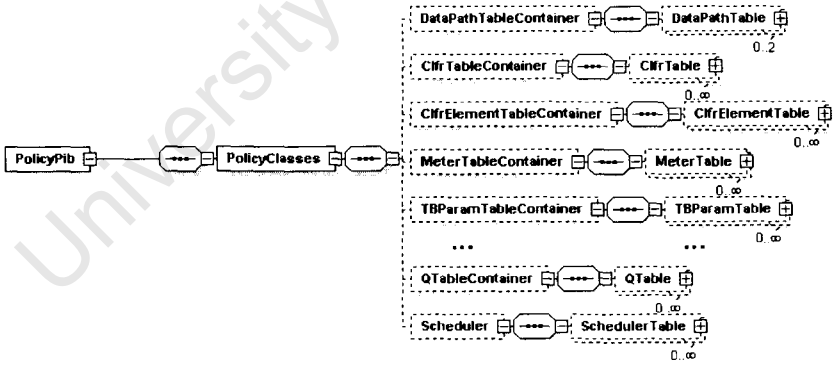

Figure 3.6: Fragment of an XML Schema depicting the mapping of the DiffServ PIB.

Figure 3.7 depicts the mapping of an individual PRI. The PRI has three children elements, each with an attribute *Pointer* that may contain an XPath expression used to

locate other PRIs. The rest of the PRI parameter are mapped to XML attributes (e.g. *MeterPrid*). An extra attribute termed *Name* was added to make the XML documents more convenient for reading. This attribute may be used in XPath expressions to locate other PRIs.

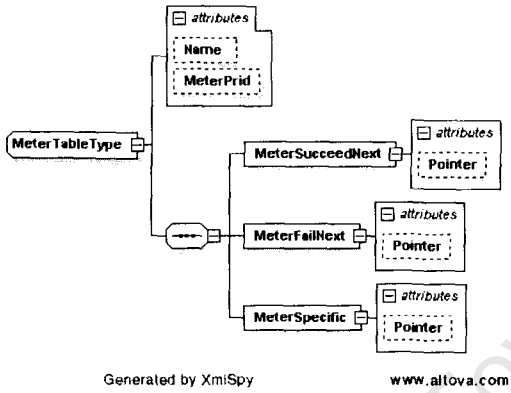

Figure 3.7: Fragment of an XML Schema depicting the mapping of a *MeterTable* PRC.

It is important to note that the mapping strategy above is not intended to be straight forward and generic mapping of PIB to XML Schema. Instead, it is based on the semantics of the DiffServ and Framework PIB and it benefits from the characteristics of XML documents in order to make the resulting XML documents convenient for reading and processing.

Appendix B contains XML samples that illustrate the usage of a DLP described in this section.

3.2.3 Device-specific Configuration Policies

DLPs provide a common configuration model for network devices from different vendors with potentially different hardware architectures, operating systems, and supporting different programming languages. Therefore DLPs always have to be translated to device-specific configuration policies.

The work in this thesis is based on the Differentiated Services functions provided by the traffic control "next generation" (tcng) infrastructure on Linux [54]. *tcng* is a revision of the traditional Linux traffic control (*tc*) designed to overcome its shortcomings

and make it more extensible. Whereas *tc* configuration scripts were cumbersome and cryptic *tcng* provides a user-friendly configuration language in which traffic control systems can be expressed in a intuitive manner.

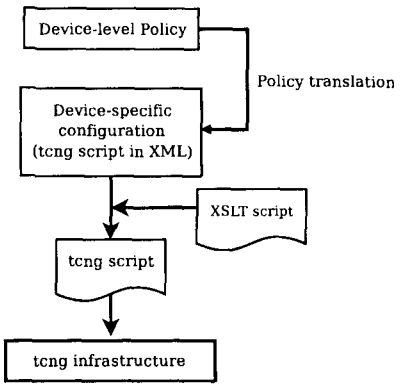

Figure 3.8: XML Schema used for configuring *tcng*.

In order to facilitate the interaction of *tcng* with network management systems, an XML-based configuration model is designed based on the *tcng* language. XML documents following this model are transformed to *tcng* scripts by applying appropriately designed XSLT stylesheets. Whereas the *tcng* language eases the task of a human administrator, XML makes integration of *tcng* with NM applications possible. Figure 3.8 depicts a scenario where a NM application configures traffic control using XML. After the DiffServ DLP is translated into XML device-specific configurations, XSLT stylesheets are used to transform these XML device-specific configurations into *tcng* scripts that can be fed to the *tcng* infrastructure and subsequently to the router.

This case highlights one of the benefits of the policy framework developed in this thesis. Although this work focuses on devices supporting *tcng* the same policy framework could be used to configure devices supporting Command Line Interfaces (CLIs). In that case a different low-level XML-based configuration model would be used. Another alternative would be to translate the DLPs directly to configuration commands (e.g, *tc* scripts), instead of using an intermediate XML-based configuration model. However, this intermediate XML-based configuration model facilitates the integration of NM systems with the devices being managed.

Figure 3.9 depicts an XML Schema used for the intermediate XML configuration documents. As detailed in the next section, this model does not support every single

configuration setup, instead it supports the most widely used configuration setups. However, this model could be easily extended as needs arise. This model makes the following assumptions:

- Traffic control is done at the egress interface. This is because in the Linux traffic control infrastructure packet marking, shaping and scheduling is only supported at the egress.
- Classification and metering is done using class selection paths[3].
- The Hierarchical Token Bucket (HTB) queueing discipline (qdisc) is used for queueing, scheduling, and shaping [55].
- The Generalised Random Early Detection (GRED) qdisc is used to implement (RED) droppers [56].

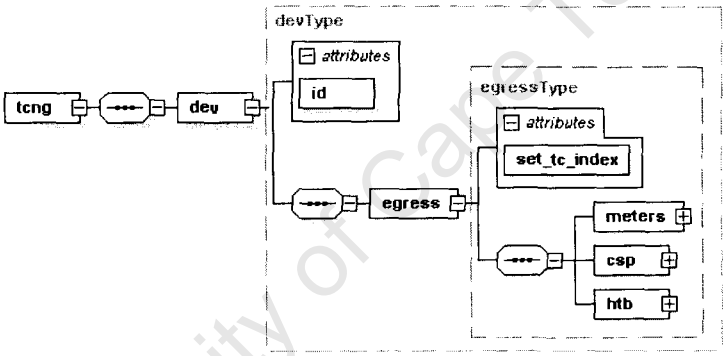

Figure 3.9: XML Schema used for configuring *tcng*.

The root element is *tcng*. It may contain one or more *dev* elements used to determine the interfaces being configured. Element *egress* defines the top-level egress qdisc of a device. Meters are defined inside the *meters* element. The *csp* element sets up class selection paths and is used for classification, metering and marking purposes. As the name suggests, HTB cascades multiple token bucket controlled classes in a hierarchy. These hierarchical trees are seamlessly represented using XML. The *htb* element in Figure 3.9 models the HTB qdisc and it may contain further classes and qdiscs. *htb* is used to control all queueing and scheduling.

[3] "Class selection paths" are *tcng* constructs that make use of the *dsmark* queueing discipline and the *tcindex* filter to classify and meter packets once and use the result several times.

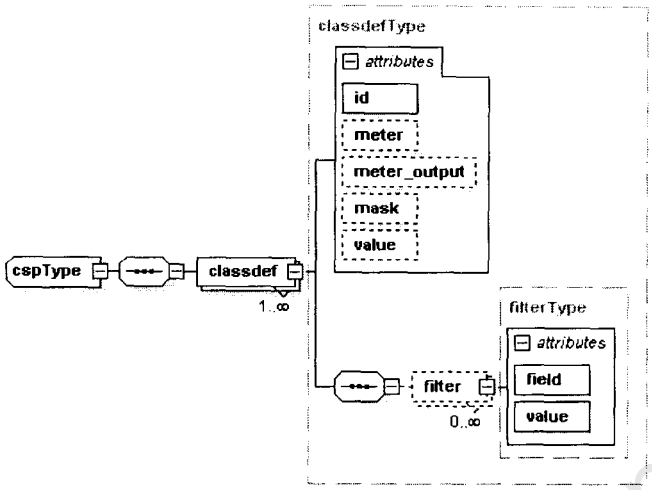

Figure 3.10: XML Schema used for configuring *tcng*.

Figure 3.10 depicts a fragment of the previously mentioned XML Schema corresponding to the definition of the *csp* element. Element *classdef* defines the conditions that must be met in order for a certain class to be selected. These classes are defined in the *htb* section and are uniquely identified by attribute *id*. Attribute *meter* identifies the meter used for a certain class. This meter should have been defined inside the *meter* element depicted in Fig 3.10. As the name suggests *meter_output* selects the meter output after testing the conformance of traffic to a profile defined within element *meter*. Elements *mask* and *value* are used for packet marking. *value* is an hexadecimal number assigned to the entire DS field of a packet. The *filter* elements depicted in Fig 3.10 are used for packet classification. Each element contains an attribute *field* that indicates the packet header field to be used and an attribute *value* that contains the value to test.

Appendix B contains an example that illustrates the usage of the XML-based *tcng* model described in this section.

3.3 Policy Translation Process

The previous section described the XML-based policy framework created in this thesis. As mentioned earlier this policy framework is made up of different abstraction layers.

In a working PBNM system the QoS NLPs have to be translated to QoS DLPs. In turn DLPs have to be translated to low-level, device-specific configuration policies. In the case of this project these low-level policies are specific to the *tcng* infrastructure. The final translation stage corresponds to the XSL transformation of the XML-based *tcng* policies to native tcng scripts which are fed to the tcng infrastructure.

This section gives details concerning the policy translation process at the various abstraction levels. Because there are many possible configuration scenarios, this thesis only focuses on the most widely adopted QoS router configuration scenarios. This is because it is very difficult to develop translation algorithms that are sufficiently generic to accommodate every possible configuration scenario.

Policies at all abstraction layers are designed following the configuration guidelines given in RFC 4594 [57]. Table 3.2 describes the DiffServ traffic classes supported and the mechanisms used to implement them[4]. By focusing on these scenarios the translation process can be greatly simplified.

Table 3.1: DiffServ classes supported.

Traffic Class	DSCP	Edge Conditioning	Queueing	Active Queue Management
Standard	DF	-	Rate	Random Drop Per DSCP
Telephony	EF	*sr+bs*	Priority	-
Multimedia Conferencing	AF41, AF42, AF43	trTCM	Rate	Random Drop Per DSCP
Multimedia Streaming	AF31, AF32, AF33	trTCM	Rate	Random Drop Per DSCP
Low Latency Data	AF21, AF22, AF23	srTCM	Rate	Random Drop Per DSCP
High Throughput Data	AF11, AF12, AF213	trTCM	Rate	Random Drop Per DSCP

An explanation for some of the terms in Table 3.2 is given below:

- sr+bs - Policer with a single rate and burst size;
- trTCM - Two Rate Three Color Marker defined in RFC2698 [58];

[4]Table 3.2 is adapted from RFC 4594.

- trTCM - Two Rate Three Color Marker defined in RFC2697 [59].

The various traffic classes are grouped into six main DiffServ classes [57]: one Default Forwarding (DF) class, one Expedited Forwarding (EF) class, and four independent Assured Forwarding (AF) classes (i.e. AF1, AF2, AF3, and AF4), each with three drop precedence levels. Figure 3.11 illustrates how the 6 classes are implemented using DiffServ datapaths at the edge of the network. At core interfaces the same datapath applies with the exception that core interfaces do not police or mark traffic.

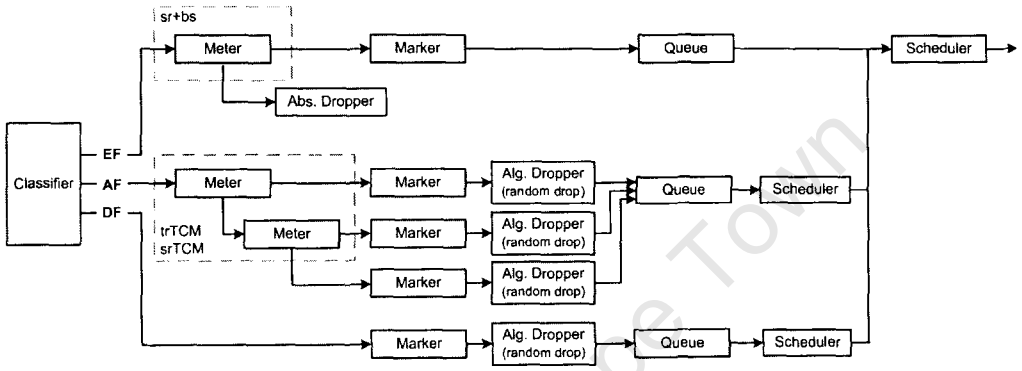

Figure 3.11: Datapaths implementing the DiffServ classes at the edge of the network.

3.3.1 Translation of Network-Level Policies to Device-Level Policies

This section presents the translation process of a QoS NLP following the model presented in Section 3.2.1 to DiffServ Policies (or DLPs) following the model presented in Section 3.2.2. The translation process depends on the capabilities of each device. Due to the fact that it would be very difficult to develop a translation algorithm that is sufficiently generic to accommodate all possible capability combination, the translation algorithm is designed to support a limited number of capabilities. However, the capabilities considered are sufficient to implement the DiffServ service classes presented in the previous section. The following is a list of supported device capabilities:

- Multi-field Classifiers (MF) with parameters: source and destination IP addresses, source and destination port numbers and the DiffServ Code Point (DSCP).

- sr+bs meters with a single rate and burst size.
- Colour-blind srTCM and trTCM meters.
- Algorithmic dropping implemented using "tail drop", "random drop", or "always drop" algorithms [50].
- Schedulers implemented using the *diffServSchedulerPriority*, or *diffServSchedulerWRR* algorithms [50].

In case there is a need for the support of new device capabilities, enhancement of the algorithm would be necessary.

The translation algorithm

The following steps describe how the translation algorithm maps NLP elements derived from QPIM classes to DLP elements derived from the Differentiated Services and the Framework PIBs. Appendix C contains the mapping table that is used by the algorithm.

- QPIM element *PacketFilterCondition* is mapped to an *IpFilterTable* element defined in the Framework PIB namespace (*frwk-pib*).
- QPIM *QosTokenBucketTrfcProf* is mapped to a *TBParamTable* element defined in the DiffServ PIB namespace (*ds-pib*). A single rate policer is represented by a single *QosTokenBucketTrfcProf* element. A two rate policer is represented using two *QosTokenBucketTrfcProf* elements. These elements are mapped to two cascaded DiffServ PIB *MeterTables*. QPIM parameter *TBRate* of the first element *QosTokenBucketTrfcProf* is mapped to DiffServ PIB parameter *TBParamRate* linked to the first *MeterTable* via a *TBParamTable*. In the case a single rate policer, QPIM parameters *TBNormalBurst* and *TBExcessBurst* are mapped to parameter *TBParamBurstSize* of the first and second *MeterTable*, respectively. The same *TBRate* value is used for the both *MeterTables*. In the case of a two rate policer, QPIM parameter *TBNormalBurst* of first and second *QosTokenBucketTrfcProf* element is mapped to parameter *TBParamBurstSize* of the first

and second *MeterTable*, respectively. *TBRate* of the first *QosTokenBucketTrfcProf* element is mapped to *TBParamRate* of the first *MeterTable* and *TBRate* of the second *QosTokenBucketTrfcProf* element is mapped to *TBParamRate* of the second *MeterTable*.

- QPIM element *PolicyDSCPVariable* is mapped to element *DSCPMarkActTable* in the *ds-pib* namespace.
- QPIM element *QoSPolicyCongestionControlAction* is mapped to element *AlgDropTable* in the *ds-pib* namespace.
- QPIM element *QoSPolicyBandwidth* is mapped to a *QTable* element, a *MinRateTable* element, and a *MaxRateTable* element, in the *ds-pib* namespace. The *QTable* is always represented. On the other hand, the *MinRateTable* is only represented in case element *QoSPolicyBandwidth* has attributes *MinRatePriority* and/or *MinRateAbsolute*. Furthermore the *MaxRateTable* is only represented in case element *QoSPolicyBandwidth* has attribute *MaxRateAbsolute*.

The mapping scheme above is used by the translation algorithm described in the following pseudo-code[5].

```
create DataPathTable;
create ClfrTable;
link DataPathTable with ClfrTable;
for each PolicyGroup do:
  for each PolicyRule with the highest Priority do:
    create data structure saved_elements to save incomplete datapath elements;
    for each PacketFilterCondition do:
      create a IpFilterTable;
      create a ClfrElementTable and link it to the IpFilterTable;
      save ClfrElementTableSpecific;
    for each PolicyAction with the lowest ActionOrder:
      if PolicyAction is QoSPolicyPoliceAction:
        execute QoSPolicyPoliceActionFunction;
      if PolicyAction is QoSCongestionControlAction:
        execute QoSCongestionControlActionFunction;
      if PolicyAction is QoSPolicyBandwidthAction:
        execute QoSPolicyBandwidthActionFunction;
```

[5]In the pseudo-code the term "link" refers to the creation of a PRI *Pointer* attribute that contains an XPath expression which refers to another PRI. Moreover the term "create" refers to the mapping of QPIM elements to DiffServ PIB elements according to the above description.

The QoSPolicyPoliceActionFunction used above is described by the following pseudo-code.

```
if (QoSPolicyTrfcProf):
  execute MeterTableFunction;
for each PolicyAction in PolicyConformAction / PolicyExceedAction:
  if (SimplePolicyAction)
    execute SimplePolicyActionFucntion;
  if (QoSPolicyCongestionControl)
    execute QoSPolicyCongestionControlActionFunction;
  if (QoSPolicyDiscardAction):
    create AlgDropTable with parameter "alwaysDrop";
    link MeterFailNext with AlgDropTable;
repeat for PolicyExceedAction;
```

The MeterTableFunction used above is described by the following pseudo-code.

```
create MeterTable;
if another MeterTable exists in this datapath link QMinRate to MinRateTable;
  link saved MeterFailNext to existing MeterTable;
else
  link MeterTable with all saved ClfrElementTableSpecific elements;
create TBParamTable;
link MeterTableSpecific to TBParamTable;
save MeterTableSucceedNext;
save MeterTableFailNext;
```

Function SimplePolicyActionFunction is described by the pseudo-code below. It simply marks packets with a DSCP.

```
create ActionTable;
create DSCPMarkActionTable;
link ActionTableSpecific to DSCPMarkActionTable;
if (MeterTableSucceedNext is saved in saved_elements):
  link ActionTable to saved MeterTableSucceedNext;
else
  link ActionTable to saved MeterTableFailNext;
save ActionTableNext;
```

Function QoSCongestionControlActionFunction is described by the following pseudo-code.

```
create AlgDropTable;
create RandomDropTable;
link AlgDropTableSpecific to RandomDropTable;
if(ActionTableNext is saved in saved_elements)
  link saved ActionTableNext to AlgDropTable;
else
  link all saved ClfrElementNext to AlgDropTable;
save AlgDropTableNext;
```

Function `QoSBandwidthActionFunction` is described by the pseudo-code below.

```
create QTable
create MinRateTable;
create MaxRateTable;
link QMinRate to MinRateTable;
link QMaxRate to MinRateTable;
create SchedulerTable;
link QTableNext to SchedulerTable;
if (SchedulerTable exists in saved_datapath_elements):
  link created SchedulerTable to saved SchedulerTable;
save SchedulerTable;
```

Notice a *SchedulerTable* is not mapped from any specific QPIM element. Instead it is created whenever a *QTable* element is created. In those cases the *QTable* precedes the *SchedulerTable*. Moreover *MinRateTables* and *MaxRateTables* are always linked to a *QTable*, as described previously. An exception occurs when all *QTables* preceding a certain *SchedulerTable* have the same value for parameter *Priority*. In those cases the *MinRateTable* containing the *Priority* is copied to the *SchedulerTable*.

In order to simplify the translation algorithm the following assumption are made about the network level policy:

- *PolicyRules* are grouped into one *PolicyGroup*.
- Each *PolicyRule* is formed by one *CompoundPolicyCondition* and one *CompoundPolicyAction*.
- Only one *QoSPolicyPoliceAction* exists per rule, therefore for a given PHB all *ClfrElementTables* are linked with the first *MeterTable*. This behaviour corresponds to the representation of datapaths with colour-blind meters.
- All *PolicyConditions* are ORed. Moreover parameter *ConditionNegated* is always FALSE.

3.3.2 Translation of Device-Level Policies to Device-specific Configuration Policies

This section presents the translation process of DLPs following the model presented in Section 3.2.2 to device configuration following the model presented in Section 3.2.3.

The translation process follows the generic datapath depicted in Figure 3.12. This generic datapath is derived from the datapath illustrated in Figure 3.11. It is designed to accommodate the 6 DiffServ classes introduced in Section 3.3. Furthermore it supports both edge and core interface configurations.

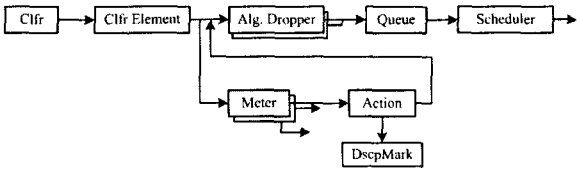

Figure 3.12: Generic DiffServ datapath.

The datapath in Figure 3.12 may be implemented by devices having interfaces with different capabilities so it is difficult to develop an algorithm which caters for all possible capability combinations. Therefore the algorithm assumes that device interfaces support support the following mechanisms:

- Multi-field Classifier (MF) with parameters: source and destination IP addresses, source and destination port numbers and the DSCP. The algorithm assumes that one ClfrTable is used, thus hierarchical classification is not supported.
- sr+bs implemented using one meter with a single rate and burst size. sr+bs is implemented using *tcng*'s SLB meter.
- srTCM implemented using two meters, with both meters having the same rate but different burst sizes.
- trTCM is implemented using two meters having distinct rates and burst sizes.
- All meters are colour blind which means that in the case of a srTCM and a trTCM the only input that is feeding the second meter is non-conforming output of the first meter [50].
- Algorithmic dropping implemented using *tailDrop*, *randomDrop*, or *alwaysDrop* algorithms [50].
- Schedulers implemented using the *diffServSchedulerPriority*, or *diffServSchedulerWRR* algorithms specified in RFC 3289 [50].

The translation algorithm follows the generic datapath depicted in Figure 3.12 and it maps DiffServ datapath elements to *tcng* elements (complying to the XML Scheme presented in Section 3.2.3). The following is a simplified description of the algorithm:

1. Schedulers are mapped to *htb classes*, expect the final scheduler which is mapped to the root *htb* element and a child *class*. Each scheduler, with the exception of the final scheduler, is positioned as a child of the *class* element which has an attribute *id* that is the same as the attribute *Name* of the scheduler following the scheduler being processed. The value of the *id* of the newly added *class* is taken from the *Name* of the scheduler being mapped. *htb* and *class* attributes *rate* and *ceil* are mapped from *dsMinRateAbsolute* and *dsMaxRateAbsolute*, respectively.

2. Queues are mapped to *htb classes* with a child *fifo* element. With the exception of classes with preceding random droppers, in these cases the *fifo* element is left out. Each queue is positioned as a child of the *class* element which has an attribute *id* that is the same as the attribute *Name* of the scheduler following the queue being processed. The *id* of the newly added *class* is mapped from the *Name* of the queue being mapped. *htb* and *class* attributes *rate* and *ceil* are mapped from *dsMinRateAbsolute* and *dsMaxRateAbsolute*, respectively.

3. Random droppers are mapped to *red* or *gred* elements. If only one random dropper precedes a queue the random dropper is mapped to a *red* qdisc element. Its attribute *id* is mapped from attribute *Name* of the queue following the algorithmic dropper being processed. In case more than one random dropper precedes a queue, the group of random droppers would be mapped to *classes* inside a single *gred* qdisc element. The value of attribute *id* of each *class* is taken from the *Name* of the algorithmic dropper being processed. The *red* or *gred* element is positioned as a child of the *class* element which has an attribute *id* that is the same as the attribute *Name* of the queue following the algorithmic dropper being processed.

4. Attributes *min*, *max*, *limit*, and *probability* of the *red* and *gred* qdiscs are taken from parameters *RandomDropMinThreshBytes* (or *RandomDropMinThreshPkts*), *RandomDropMaxThreshBytes* (or *RandomDropMaxThreshPkts*), and *AlgDropQThreshold*, respectively. In cases where more then one random dropper are mapped to

a single *gred* qdisc (AFx setups), its attributes *min*, *max*, and *limit* are taken from the first random dropper processed. Therefore in these cases it is desirable that all random droppers preceding a queue have the same value for parameters *RandomDropMinThreshBytes* (or *RandomDropMinThreshPkts*), *RandomDropMaxThreshBytes* (or *RandomDropMaxThreshPkts*), and *AlgDropQThreshold*.

5. *ActionTables* parametrised by *DSCPMarkActionTables* are mapped to new *classdef* elements. Parameter[6] *DscpMarkActDscp* of a PRI *DSCPMarkActionTable* is mapped to attribute *value* of the newly created XML element *classdef*. The *classdef* element is labelled with an attribute *id* that is the same as the attribute *Name* of the datapath element following marker being processed (this datapath element can be an algorithmic dropper or a queue).

6. Single meters are mapped to *SLB* elements inside the *meters* element. The attribute *id* of newly created SLB element is taken from the *Name* of the meter being mapped. For each newly created *SLB* element an attribute *meter* is added to a *classdef* element with an *id* equal to the *Name* of the first queue or algorithmic dropper following the meter being processed. *SLB* attributes cir (committed information rate) and cbs (committed burst size) are mapped from the parameters *TBParamRate* and *TBParamBurstSize*, respectively.

7. Two cascaded meters are mapped to *trTCM* or *srTCM* elements inside the *meters* element. The attribute *id* of newly created meter element is taken from the *Name* of the DiffServ meter being mapped. For each newly created meter element an attribute *meter* is added to a *classdef* element with an *id* equal to the *Name* of the first queue or algorithmic dropper following the meter being processed. The value of attributes *cir* and *cbs* are taken from the parameters *TBParamRate* and *TBParamBurstSize* of the first *MeterTable*. In the case of a trTCM pir (peak information rate) and pbs (peak burst size) are mapped from the parameters *TBParamRate* and *TBParamBurstSize* of the second *MeterTable*. In the case of an srTCM, ebs (extended burst size) is mapped from parameter *TBParamBurstSize* of the second *MeterTable*.

8. Each *ClfrElementTable* is mapped to a number of *filter* elements inside the *class-*

[6]Parameter or attribute are used interchangeably in this thesis.

def element with *id* equal to the *Name* of the first queue or algorithmic dropper following the *ClfrElementTable* being mapped. For each parameter of the *IpFilter* that parametrises the *ClfrElementTable* a new *tcng filter* element is created.

9. The classification of AFx groups is treated as a special case. These cases are identified when two or three *ClfrElementTables* that classify traffic based on DSCPs precede a set of two cascaded *MeterTables*. In these cases the three leftmost bits of the DSCPs are used to identify the AFx group. An example of one such filter is given below. The filter is selecting all AF4 packets (with DSCP 100xxx), irrespective of their drop precedence.

```
<filter field="ip_dscp >> 3" value="0b100"/>
```

The algorithm described above follows rigorously the datapath of Figure 3.12. In case new datapath setups are required the algorithm would have to be modified to suite the new requirements.

As described before a *tcng* XML configuration[7] obtained with this algorithm would be transformed to *tcng* scripts using XSLT stylesheets. Appendix E contains the XSLT stylesheets developed in this thesis. The following example demonstrates how an XML *gred* element, with three *class* elements representing the three drop precedences of an AFx group, is translated to a *tcng* script fragment.

Table 3.2: Example of a transformation of a *tcng* XML configuration to a *tcng* script.

tcng XML configuration	*tcng* script resulting from an XSLT transformation

`<gred bandwidth="3Mbps" limit="1536kB"`
`min="128kB" max="384kB" burst="218kB`
`avpkt="1000B">`
`  <class id="af41" probability="0.1" />`
`  <class id="af41" probability="0.4" />`
`  <class id="af41" probability="0.6"`
`  default="true" />`
`</gred>`
	`gred(bandwidth="3Mbps", limit="1536kB",`
	`min="128kB", max="384kB", burst="218kB",`
	`avpkt="1000B") {`
	`  $af41 = class(probability="0.1");`
	`  $af41 = class(probability="0.4");`
	`  $af41 = class(probability="0.6",`
	`  default="true");`
	`}`

[7]Although all policies are represented in XML, the term XML is added explicitly in this case to differentiate the *tcng* configuration represented in XML form, from the native *tcng* configuration represented using the *tcng* language.

From the above table one should realise that the *tcng* XML configuration has a structure that closely follows the *tcng* language. This is to simplify the XSLT stylesheets used for the transformations. At this point a question might arise: Why are *tcng* configurations in XML required? This is simply to ease the interaction between the NM system and the *tcng* system. Otherwise one would need to develop a specialised parser to generate the required *tcng* scripts directly from the DiffServ policy. This highlights one of the benefits of using XML: one can focus on the development of the application instead of focusing on the details of the parser.

3.4 Proposed PBNM Architecture

The previous sections described the policy framework developed in this work. This section presents the proposed policy-based QoS management architecture. It describes how the policy framework developed in the previous sections fits into the proposed architecture.

Figure 3.13 shows the proposed architecture. Note that some of the functional elements were originally proposed by the IETF for its PBNM architecture, as described in the previous chapter.

The Policy Repository (PR) stores NLPs and the DiffServ network topology in XML format. The network topology consists of all PDPs being managed, and all PEPs being managed by each one of the PDPs. The PR interfaces with other functional elements using SOAP and XML-RPC interfaces.

The Management Station (MS) is used to insert NLPs into the repository. Furthermore it can activate/deactivate all PDPs and monitor their behaviour. It interfaces with the PR and PDPs using SOAP web services. The author chose to use SOAP interfaces in favour of XML-RPC due to the fact that SOAP is becoming the de-facto XML-based message exchange protocol. Thus it opens up possibilities for integrating the PDP and repository with different management applications.

The PDP is the main component of the architecture. It is responsible for translating (using the translation algorithm described in Section 3.3.1) NLPs to DiffServ policies based on the capabilities of each device. It keeps a local repository containing details of each PEP being managed, including the active DiffServ policies and supported roles and capabilities. The PDP fetches policies in the repository using an XML-RPC

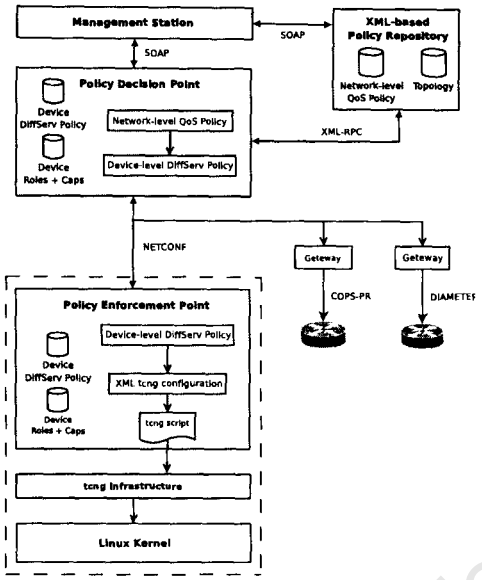

Figure 3.13: Global view of the proposed architecture.

interface. This interface was chosen in favour of a SOAP interface because in this case the PDP and the repository are tightly integrated, thus the issue of interoperability with external systems becomes less relevant. DiffServ policies are sent to PEPs using the NETCONF API exposed. The PDP exposes a number of web services called by the MS to control and monitor its operation.

It must be noted that practical networks have PEPs supporting various protocols such as COPS-PR, DIAMETER or even SNMP. For this reason the author envisages the use of conversion gateways, such as those discussed on Chapter 2, for the management of those devices. These gateways perform the conversion of NETCONF messages to other messages conforming to other management protocols such as COPS-PR or DIAMETER. This scenario is illustrated in Figure 3.13.

In the framework proposed in this thesis, PEPs expose a NETCONF API used by PDPs to execute management operations (i.e., send DiffServ policies and fetch data such as supported roles and capabilities). As mentioned before the Linux *tcng* infrastructure is used for the implementation of DiffServ-enabled IP routers. Therefore PEPs are responsible for configuring such entities, according to the DiffServ policies received from the PDP. PEPs translate DiffServ policies to *tcng* XML configurations conforming to the XML Schema described in Section 3.2.3. Such configurations are

subsequently transformed to *tcng* scripts (using XSLT stylesheets), which are fed to the *tcng* system. The PDP and PEP share the DiffServ policy that is active on the PEP at any given time.

This architecture decouples the PEP from the routing device. This makes it possible for different DiffServ routers to be managed by the same system, by using different translation algorithms at the PEP.

3.4.1 Element Interaction

This sections provides more details with regards to the interaction between the various elements of the proposed architecture. Figure 3.14 shows how the various architecture elements interact.

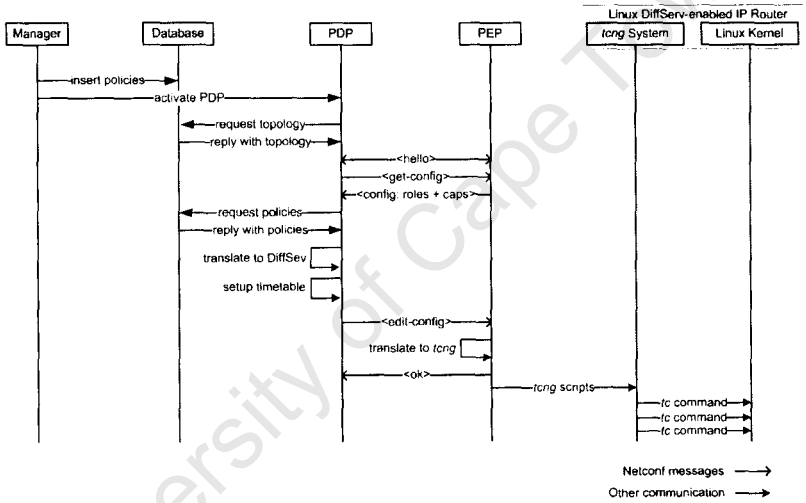

Figure 3.14: Interaction between the various architecture elements.

The network administrator inserts network-level policies in the PR. Some or all of the PDPs would be activated. Upon activation the PDP requests the network topology (i.e the PEPs it is required to manage). It then establishes a NETCONF session with each PEP, and requests the PEP's roles and capabilities using NETCONF's *<get-config>* operation. The PDP uses the supplied roles to fetch the applicable NLPs in the PR. After validating the received NLPs against the XML Schema described in Section 3.2.1, it uses the capabilities supplied by each PEP to translate the NLPs to

DiffServ policies. A time grid is set up and at desired times the DiffServ policies are sent to the corresponding PEPs using a NETCONF's *<edit-config>* operation.

Upon receiving the DiffServ policies, the PEP translates them to *tcng* XML configurations, and transforms the configurations to *tcng* scripts (described using the *tcng* language). These scripts are then fed to the *tcng* system. If in the translation or transformation phases the PEP encounters a problem it reports it to the PDP using a NETCONF's *<rpc-error>* message. Otherwise it sends an *<rpc-reply>* with the *<ok>* element to the PDP; this means that the transaction was successful.

3.4.2 Decision Process at the PDP

The decision process at the PEP does not take into consideration any external events. As such the principal factors affecting the decision process are the roles and capabillities supplied by PEPs and the *PolicyTimePeriodConditions* (TPCs) attached to PolicyRules (i.e., the times at which the rules are active).

When the PDP connects a PEP it requests the roles (given by *RoleComboTable* parameter *RoleComboRoles*) and capabilities (given by *RoleComboTable* parameter *RoleComboCapSetName*) of the interfaces managed by the PEP. The roles are used to select the appropriate NLPs in the PR and the capabilities are used in the policy translation process. For each (*RoleComboRoles*, *RoleComboCapSetName*, *DataPathIfDirection*) combination the PDP generates one DiffServ Traffic Control Block (TCB) with a single *DataPathTable*. This means that for each single interface there can be only two *DataPathTables*. Upon receiving a policy the PEP uses each *DataPathTable*'s (*RoleComboRoles*, *RoleComboCapSetName*) combination to determine the interfaces for which the policy applies.

As far as time schedules are concerned each *PolicyRule* may have a TPC attached to it. Upon selecting the applicable policies on the repository using the *RoleComboRoles* supplied by the PEP, the PDP translates all the *PolicyRules* associated with each policy and saves the resulting DiffServ policies in a local configuration file. The PDP then generates DOM tree representing a time grid with 24 intervals representing each hour of the day. In each grid the PDP places the active DiffServ policies during that interval. This group of policies represents the policy incarnation that is active on the PEP at that moment. Therefore at each hour the PDP verifies the time grid and if

the policy incarnation changes the PDP immediately sends the new incarnation to the PEP.

3.4.3 Enforcement Process at the PEP

In the previous section the author described how interface roles and capabilities are interpreted by the PDP. This section describes how these parameters are used when DiffServ policies are translated to *tcng* configurations.

During the translation process for each *DataPathTable* processed the PEP fetches the (*DataPathIfDirection*, *DapaPathRoles*) combination and uses it to identify the interfaces for which the policy apply. For each one of these combinations, one *dev* element is generated. This element may represent one or more interfaces.

As mentioned before most Linux traffic control functionality is only applicable to the "outbound" side of egress interfaces. Therefore during the translation process *DataPathTable* parameter *DataPathIfDirection* is ignored. For this reason under a given *dev* element only one *egress* qdisc (or element) may be defined.

3.4.4 The PDP-PEP Communication Model

As illustrated in Figure 3.14 the communication model between the PDP and PEP employed in this scheme is different from that employed by schemes that use COPS-PR. COPS-PR sees the PDP as the server and the PEP as the client. The PEP opens the connection with the server. In case of failure the client is responsible for re-establishing the lost connection.

On the other hand NETCONF employs a client/server model where the PDP is the client whereas the PEP is the server. The PDP is configured with a list of PEPs that it manages. It is responsible for initiating communication with all PEPs, requesting roles and capabilities of each device, and sending policy decisions. The PDP has to keep track of all open sessions to PEPs. In case of connection failure the PDP is responsible for re-establishing the lost connection. This characteristic makes this scheme less attractive than the COPS-PR scheme as far as failure recovery is concerned.

3.5 Discussion

In this chapter the design of the proposed XML-driven Policy-Based QoS Management framework has been presented. One component of this framework is the QoS policy data model. This model is divided into three abstraction layers: the NLP layer, the DLP layer, and the layer defining device-specific configuration policies tailored for the Linux *tcng* infrastructure. The NLP layer enables the system to hide its internal intricacies from a network administrator, whereas the DLP layer enables the system to manage QoS in devices made by different vendors, supporting different configuration commands. By changing the policy rules at the NLP layer, a network operator can alter the IP services it offers to its subscribers without having to know the details of its network configuration and mechanisms of the network devices.

QoS Policies at the three abstraction layers are modelled and represented in XML, thus simplifying the internal structure of the system by using the same tools to process QoS policies at the various abstraction layers. Furthermore NLPs are stored in XML format which facilitates the translation of NLPs to DLPs. As exemplified in the different sections the resulting QoS policies are self-describing and easy to use.

As part of the system design two policy translation algorithms were defined: one that translates NLPs to DLPs, and one used to translate DLPs to XML-based configuration commands. To simplify the policy translation process the algorithms only support three DiffServ classes: EF, AF, and DF. However these DiffServ classes can be used in most management scenarios [60, 57].

This chapter concludes by defining a PBNM architecture (including the various architectural elements, the interfaces and interaction between them), and describing how the QoS policy data model is used in the context of this architecture.

The PBNM framework proposed in this thesis should be useful to manage QoS in the network core in enterprise networks and ISPs. In this case a network administrator would use NLPs to define different QoS service levels and assign them to traffic from different users or departments, using port numbers and IP addresses.

Furthermore, the PBNM architecture proposed in this thesis is similar to the Policy Control Architecture adopted by the Third Generation Partnership Project (3GPP) for use with its service delivery platform, the IP Multimedia Subsystem (IMS) [61]. Therefore it is possible to use the system proposed in this thesis to manage QoS in

an IMS network. However small modifications would have to be applied to the PDP. In particular, the PDP would have to be enhanced with a specialised DIAMETER stack to interface with a Proxy-Call Session Control Function (P-CSCF). Furthermore, the translation logic in the PDP would have to consider external factors such as the service information conveyed in the Session Description Protocol (SDP) field of Session Initiation Protocol (SIP) messages received at the P-CSCF; that information would be used by the PDP to select the appropriate NLPs in the PR.

Chapter 4

Implementation of an Evaluation Platform

4.1 Introduction

This chapter presents the implementation of the architecture proposed in Chapter 3. The implementation of the various architectural elements (MS, PR, PDP, PEP, and DiffServ Router), and interfaces between them is given in detail. This evaluation framework serves as a proof of concept for the proposed XML-driven Policy-based QoS Management framework. Furthermore, it is used to evaluate the performance of the proposed system.

Appendix E contains the source code for the software developed in this thesis.

4.2 Network Topology

Fig 4.1 shows the network topology of the evaluation platform. To save resources the MS, PR, and PDP were implemented in one machine. These logical elements communicate via the loopback interface. The following sections contain detailed descriptions of the implementation of each one of the PBNM architectural elements.

.

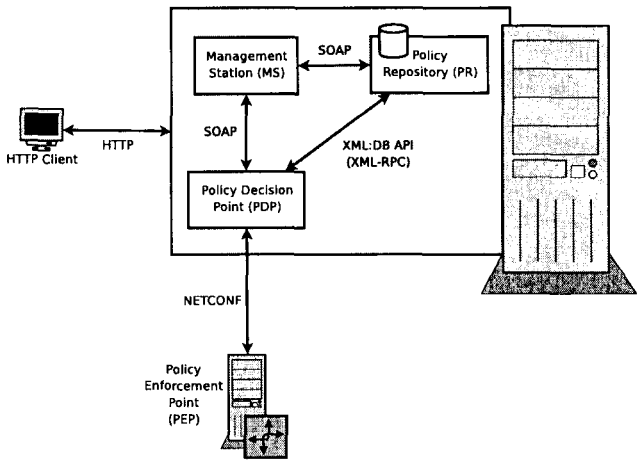

Figure 4.1: Evaluation platform

4.3 The Management Station

As described in the previous chapter the MS is used to retrieve or modify NLPs in the PR, and control the operation of PDPs. The MS is implemented as a PHP5 application running behind an Apache web server. The PHP application consists of a one PHP script and two PHP classes. The PHP script uses the classes to access the PR and control the PDPs. The PHP classes implement two SOAP clients that call operations on the PR and the PDPs. After NLPs are fetched from the PR, the MS dynamically generates HTML pages (i.e., the user interface) by transforming the XML documents using XSLT scripts stored in a local repository.

Figure 4.2 illustrates the architecture of the MS. At this point the MS is only able to query the PR, retrieve NLPs and the network topology, and start, restart, or stop the PDP's engine. By using SOAP it is easy to add operations at the PDP and PR, and thus extend the MS with more functionality. Currently, to insert or modify information in the PR a Java client supplied with the PR may be used.

Figure 4.3 is a snapshot of the MS. It shows how the MS displays the information on the PR. In Figure 4.3 three collections are shown. The user can browse the collections and verify the documents. Figure 4.4 shows a snapshot of the MS, where a *Policy-Group* is displayed in a HTML page. This illustrates one of the uses of XML - XML documents can be easily translated to a user interface by applying appropriate XSLT

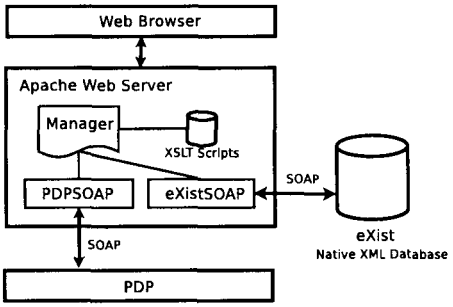

Figure 4.2: Implementation of the Management Station

transformations.

4.4 The Policy Repository

The PR stores NLPs and the network topology (PDPs) as XML documents. The XML Schema used for the NLPs was described in the previous chapter; the following extract illustrates the structure of the XML document in the PR that contains the details of the PDPs being controlled by the MS. The details kept are the IP addresses and port numbers of the SOAP servers at the PDPs, and the IP addresses and port numbers of the NETCONF servers at the PEPs. Furthermore, as depicted by the extract, the network manager or administrator decides which PEPs are managed by which PDPs.

```
<ManagedNetwork>
  <PDP IpAddress="10.128.1.44" PortNum="8080">
    <PEP IpAddress="10.128.1.35" PortNum="5454" />
    <PEP IpAddress="10.128.1.36" PortNum="5454" />
  </PDP>
  ...
</ManagedNetwork>
```

The PR is implemented using eXist, an open source native XML database developed in Java [62]. eXist stores documents in hierarchical collections similar to storing files in a standard file system. Collections are not bound to any XML Schema. eXist is equipped with an XQuery engine that is used to query the database; XQuery can be likened to SQL for relational databases. Moreover, the database support updates through an XQuery extension; it also supports XInclude, XPath, and XPointer.

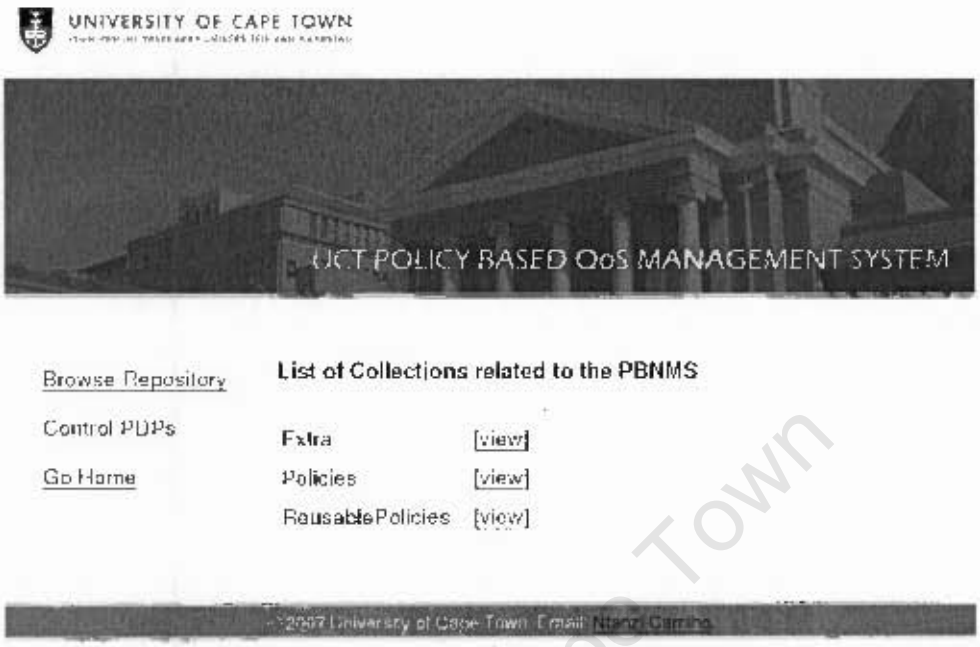

Figure 4.3: Screen-shot of the Management Station

Figure 4.4: Screen-shot of the Management Station. View of a *PolicyGroup*.

For the work in this thesis eXist was deployed as part of a web application served by a web server (Jetty), included in the library. The database interfaces with the MS and PDPs through a standard XML:DB API [63] and SOAP interface. The XML:DB API is used by the PDP, whereas the SOAP interface is used by the MS. Two web services may be accessed via the SOAP interface: one contains methods to query the server and retrieve documents, and the other contains methods for storing and removing documents and collections. At this point only the former service is used by the MS. As mentioned in the previous section, if one desires to update the PR a Java client supplied with eXist may be used. The client interfaces with the PR via using the XML:DB API.

Currently all data are stored under a *PBNMS* collection. This collection contains four other collections:

- *Policies* - used to store stand-alone NLPs and the NLP XML Schema described in Chapter 3.
- *ReusablePolicies* - used to store re-usable policy data such as re-usable actions and conditions.
- *Extra* - used to store other useful documents such as other XML Schemas used.
- *Topology* - used to store the XML document describing managed network topology.

4.5 The Policy Decision Point

The PDP is the "brain" of the PBNM system. It is in charge of the policy decision making. In other words it must fetch PEPs' roles and capabilities; it uses the supplied roles to fetch the applicable NLPs in the PR; it translates the NLPs based on each PEP's interface capabilities; it sets up a provisioning time schedule; and sends the DiffServ policies to the various PEPs at the requested times.

The PDP was developed in Java and it consists of several modules as shown in Fig 4.5. The following sections describe the different modules in Fig 4.5.

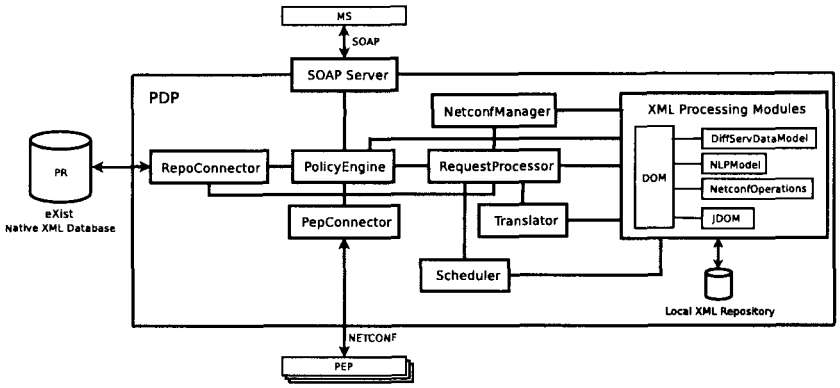

Figure 4.5: Implementation of the PDP

XML Processing Modules

The XML parsing modules are used for parsing and processing of XML documents and serve as the interface between the various modules and the local repository. *DiffServDataModel*, *NLPModel* and *NetconfOperations* are Java packages automatically generated from XML Schemas using XML Data Binding. XML data binding is the mapping of one or more XML data elements to a set of related software objects. It allows developers to programmatically work with XML documents from within software applications via a set of objects. Therefore these three packages serve as an abstraction layer between the PDP application and the XML documents containing DLPs, NLPs, NETCONF operations, thus alleviating the need for having to process those documents using low-level APIs. The three packages consist of a number JavaBeans representing the XML policy data elements, and include methods for serialising XML documents to in-memory DOM trees and back to files, for validation of XML documents, and for general XML processing.

The JDOM library [64] is used in favour of the low-level DOM API for general XML processing in case XML data binding is not used. This is because this library provides a straightforward interface for efficient reading, manipulating and writing of XML documents, and it is optimised for Java programming. Additionally JDOM is used to access the local repository, which is simply a collection of XML documents stored in the file system.

PolicyEngine

This is the main module in the PDP. It implements three methods: *connectAll*, *disconnectAll*, and *restartAll*. *connectAll* establishes connection with all PEPs assigned to the PDP; *disconnectAll* terminates connection with the PEPs; *restartAll* re-establishes all connections, so that the policy decision process is repeated. *restartAll* is mostly used when NLPs are updated on the PR and the PDP needs to re-do all policy decisions. These three methods are exposed to the MS in a web service accessed through the SOAP server included in the PDP.

The SOAP server is implemented using Apache SOAP [65]. Apache SOAP is comprised of a servlet which listens for SOAP requests from the MS, and the corresponding Java classes necessary for translating the requests into calls to the service code on the PDP, and for translating the result of the calls into SOAP responses to be sent to the MS. The servlet runs on Apache Tomcat, an open source application server [66].

Upon activation, the *PolicyEngine* calls an initialisation method that loads the necessary PDP configuration parameters into memory, in a DOM tree. These parameters are used throughout the operation period. These configuration parameters are specified in an XML document (*pdpconfig.xml*) that is located on the *config* directory of the PDP library. This XML document has the following format.

```
<PDPConfig>
  <DLPFile>/etc/pbnms/dlps.xml</DLPFile>
    <PR>
      <Driver>org.exist.xmldb.DatabaseImpl</Driver>
      <URIPrefix>xmldb:exist://localhost:8080/exist/xmlrpc</URI>
      <Collections>
        <Policies>/db/PBNMS/Policies</Policies>
        <Topology>/db/PBNMS/Topology</Topology>
      </Collections>
    </PR>
  <Compression>true</Compression>
</PDPConfig>
```

Element *DLPFile* contains the path to the XML document that contains the translated DLPs. *PolicyRepositories* contain the details of the PR used. *Driver* is the implementation of the database interface that encapsulates the database access logic. Each

database that supports the XML:DB API must provide a database specific driver. *URIPrefix* defines the driver class to use as well as the host address of the database server on the network. *Collections* contain the paths to stand-alone NLPs and the document with the network topology. Finally, element *Compression* defines whether XML message compression is used or not.

PepConnetor

The *PepConnector* serves as the interface to the various PEPs. It has a simple API to connect to the PEP, and to get or send a message. The *PepConnector* is invoked by the *PolicyEngine*. The number of *PepConnectors* created is the same as the number of PEPs being managed.

Due to the verbose nature of XML documents, prior to sending messages, the PDP must first compress them in order to reduce network usage. Therefore the *PepConnector* is equipped with a zlib [67] compression library used to compress messages prior to sending, and de-compress received messages. Chapter 5 provides a discussion on effect of this compression on the performance of the system.

In order to secure NETCONF messages *PepConnector* encrypts them using the Secure Sockets Layer (SSL) protocol.

RepoConnetor

This module is used by the *PolicyEngine* to fetch the network topology and by the *RequestProcessor* to fetch NLPs and from the PR. It interfaces with the PR using the XML:DB API. The *RepoConnector* fetches NLPs using the interface *roles* supplied by the various PEPs. The required NLPs are selected using an XPath expression.

Translator

The *Translator* is used to translate NLPs to DLPs according to the interface capabilities supplied by the PEP. This class is invoked by the *RequestProcessor*. Refer to the discussion on the *RequestProcessor* for more details about the policy decision process.

NetconfManager

This module contains an API used to construct NETCONF messages using the *NetconfOperations* package in the parser layer. The *NetconfManager* is invoked by the *RequestProcessor*. Refer to the discussion below on the *RequestProcessor* for more details.

Scheduler

The *Scheduler* is responsible for creating a time schedule based on the *PolicyTimePeriodConditions* attached to the NLPs. The time grid has 24 intervals representing each hour of the day. In each time slot the *Scheduler* places the active DiffServ policies during that interval; these policies are kept in a DOM tree which is rebuilt every 24 hours. In other words the policy decision process is repeated every 24 hours, although this period can be adjusted.

That group of policies in each time slop represents the policy incarnation that should be active on the PEP at that moment. Therefore at each hour the Scheduler verifies the time grid, and if the policy incarnation changes it informs the *RequestProcessor*, which immediately sends the new incarnation to the PEP.

RequestProcessor

This class is designed as a Java thread and is invoked by the *PolicyEngine*. After the *PolicyEngine* fetches the list of PEPs to manage, it loops through each of them and creates a *PepConnector* that establishes the connection; then the *PolicyEngine* creates a *RequestProcessor* thread and passes it the newly created *PepConnector*. In this way all PEPs are handled in parallel using separate threads, thus speeding up the whole QoS provisioning process and allowing the continuous operation of the *PolicyEngine*.

The *RequestProcessor* contains most of the logic of the PDP. After it receives a *PepConnector* object it fetches the interface roles and capabilities from the PEP; the fetched roles and capabilities are stored temporarily in a DOM tree. The roles are used to select the applicable NLPs in the PR through the *RepoConnector*. When armed with the required NLPs, the *RequestProcessor* calls the *Translator* and passes it the DOM tree containing the saved interface capabilities. The *Translator* applies the

translation algorithm described in Chapter 3 to the NLPs; the translation algorithm uses the interface capabilities stored in the DOM tree. The resulting DLPs are passed back to the *RequestProcessor*. At this point the *RequestProcessor* passes the newly translated DLPs to the *NetconfManager*, which uses it to build a NETCONF *<rpc>* with an *<edit-config>* operation; this operation adds the DLPs to the PEP. Armed with the NETCONF message, the *RequestProcessor* passes it to the *PepConnector*, which sends it to the PEP. Notice that during all this process the *PepConnector* keeps an active connection or session to a PEP.

If the reply from the PEP is positive (i.e., the PEP sends an *<ok>* message back) the *RequestProcessor* uses the parsing modules to save the active DLPs in the local repository. Otherwise the *RequestProcessor* adds an entry to the log in the local repository containing the details of the error. At any given point in time the DLPs saved in the PDP are exactly the same as those saved in the PEP.

Upon completion of the provisioning process the *RequestProcessor* goes to "sleep" mode and it is woken up by the *Scheduler* when a new incarnation becomes active. When this happens the *RequestProcessor* sends this new DLP incarnation to the PEP and goes back to "sleep" mode. Notice that in real management scenarios this operation mode should not be used because it does not allow the PEP to report any unusual events that might occur. Instead the NETCONF notification model should be used. In this model the PDP has to subscribe for notifications at the PEP. Upon subscription the PEP may send asynchronous event messages to the PDP when unusual events occur. This feature was not implemented in the system due to the fact that the NETCONF working group [68] is still actively working on the event notifications document.

4.6 The Policy Enforcement Point

PEPs are responsible for performing the last policy translation, where the actual device configurations (for Linux *tcng*) are derived from DPLs (i.e. DiffServ Policies). In the proposed system PEPs are equipped with NETCONF agents capable of processing NETCONF requests and emitting replies to the PDP. DiffServ Policies are kept in a local repository, which is simply a collection of XML documents.

PEPs are implemented using Yenca, an open source NETCONF implementation [69].

Yenca is a simple NETCONF agent written in C by the LORIA-INRIA Lorraine Madynes Research team, in France [70]. The agent has a modular design allowing users to extend it with modules that are loaded dynamically. The author has extended the Yenca agent with a DiffServ module that supports DiffServ Policies conforming to the XML Schema described in the previous chapter. The module validates policies and translates them to *tcng* scripts, which are fed to the *tcng* system. The *tcng* system translates the *tcng* scripts to *tc* commands and executes them. At this point the desired configurations are created on the kernel and the device would be ready to provide differentiated treatment to IP packets.

The architecture of the PEP is illustrated in Fig 4.6. The *Agent Engine* consists of a module, *netconfd*, and it is responsible for the creation of the agent daemon, and provides the User Interface (UI). The *Network Layer* is in charge of the communication with the PDP; this includes securing the NETCONF sessions using the SSL implementation provided by the OpenSSL library. The *Parser Layer* contains a generic DOM parsing API to facilitate the parsing and processing of XML documents. Furthermore, the *Parser Layer* finds the module[1] to which the message is destined by inspecting the content layer[2] of the message. The following XML extract represents a request from a PDP to replace the configuration datastore at the PEP, for the specified module, with the configuration supplied in-line. The element highlighted identifies the module required to process the request.

```
<rpc message-id="1">
  <copy-config>
    <target>
      <running/>
    </target>
    <source>
      <config>
        <diffservt>
          <Pib>
            ...
            <DataPathTable Name="dpt1" DataPathCapSetName="defCaps"
            DataPathRoles="edgeRouter" DataPathIfDirection="2">
```

[1]Although not depicted in the picture the Yenca agent is also equipped with two modules, for interface and routing configuration.

[2]The term "content layer" refers to the one of the logical layers of the NETCONF protocol, describes in Chapter 2.

```
                <DataPathStart Pointer="clfr1" />
              </DataPathTable>
              ...
            </Pib>
          </diffservt>
        </config>
      </source>
    </copy-config>
  </rpc>
```

The DiffServ module is responsible for translating DiffServ Policies to *tcng* scripts. It serves as the NETCONF agent's interface to the *tcng* infrastructure. This module consists of four modules: *diffservt*, *tcng_interface*, *translator*, and *parser*. *diffservt* executes the operations defined at the operation layer of NETCONF (refer to Chapter 2), and is responsible for saving and retrieving configurations to and from the local repository; it does so through the *Parser Layer*. *tcng_interface*, as the name suggests, is responsible for invoking the *tcng* system after translating the DLP. It uses the *translator* to translate DLPs to *tcng* configurations. It then transforms the *tcng* configurations using the appropriate XSLT scripts stored in the local repository.

parser contains a parsing API created specifically to handle documents containing DiffServ configurations. In other words it understands DiffServ semantics. For example, one might want to discover the first element in a specific datapath. With a generic XML parser it is impossible to do that, because the XML Schema that defines a DiffServ Policy does not use the hierarchical features of XML documents to represent the relationships between elements in the same DiffServ datapath.

4.7 Limitations of the Implementation

This section highlights the limitations of the evaluation platform described in the previous sections as a result of time constraints:

- The GUI used by the MS is still very rudimentary and can only be used to view data in the PR and activate/deactivate operations at the PDPs.
- Due to the complexity of NETCONF's *<edit-config>* operation (it supports flexible subtree and XPath filtering) the PEP only supports the replacement of the

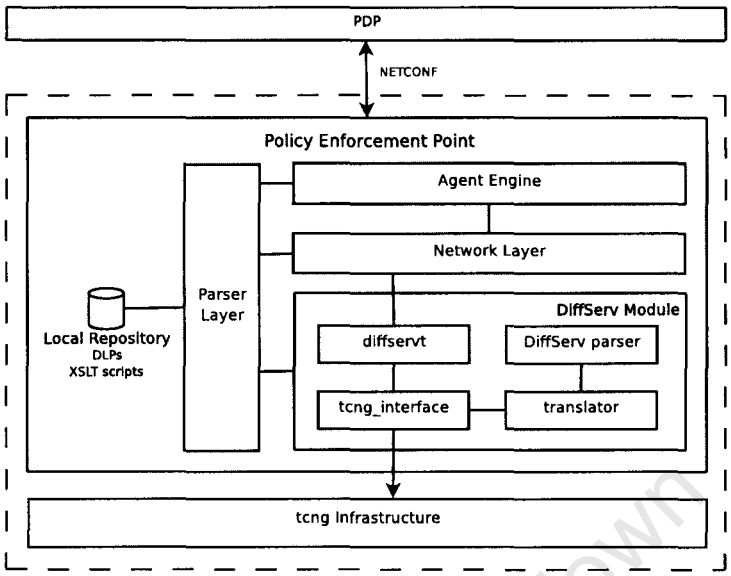

Figure 4.6: Implementation of the PEP

whole configuration. This means that the *<edit-config>* operation used is actually behaving like the *<copy-config>* operation.

- The PDP is not capable of executing incremental updates on the PEP, where only fragments of the configuration are modified. This means that if a policy is updated in the PR the PDP has to send the whole policy to the PEP even if only a small number of configuration parameters are changed.
- The PDP's *Scheduler* module is not fully functional, therefore the PDP updates the PEPs as soon as it translates the NLPs.
- Only one generic XSLT script is used by the PEP to transform the *tcng* XML configuration into a native *tcng* script.

4.8 Discussion

In this chapter the prototype implementation of the framework proposed in Chapter 3 was presented. All the PBNM architectural elements were described in detail and

the issues surrounding their implementation were addressed. The implementation prototype serves to validate the proposed design.

Due to that fact that XML is used throughout the entire policy life-cycle and that NETCONF messages are XML-based, both the PEP and PDP use a similar DOM-based parsers. Furthermore the same parser is used at the PEP and PDP to process protocol messages as well as XML documents containing QoS policies. This speeded up the system's development process.

In the next chapter the tests carried out on the evaluation platform are presented. The evaluations serve to demonstrate the use of the proposed system in a management scenario and to study the performance of the system compared to a PBNM system using COPS-PR and an SPPI-based DiffServ PIB.

Chapter 5

System Validation and Performance Evaluation

5.1 Introduction

As demonstrated in previous chapters there are many practical benefits of using XML for the modelling and representation of policies. However, like any other markup language, XML poses serious concerns with regards to the performance of the system due to its verbosity. It is therefore important to measure the impact the use of XML and companion standards have on the performance of the system. This chapter presents a performance study where the performance of the proposed XML-based system is compared to the performance of a similar PBNM system based on COPS-PR and an SPPI-based DiffServ PIB. The evaluation platform described in the previous chapter is used to run the performance tests. Furthermore, the same platform is used to validate the operation of the proposed system.

5.2 System Validation

This section presents the tests conducted to validate the operation of the system. Figure 5.1 illustrates the layout of the evaluation framework used for the performance tests. This platform, with the exception of the three clients and server, was presented in the previous chapter. All network interface cards support data rates up to 100

Mbps. The PEP is used to route traffic from the server to clients 1, 2, and 3. Network traffic is generated by the server using the Distributed Internet Traffic Generator (D-ITG) [71]; the three clients use the same library but operate as traffic sinks. Prior to packet sending, the traffic generator marks all packets with the required DSCPs. The router in Figure 5.1 represents an edge router and assumes that the packets are already marked with DSCPs.

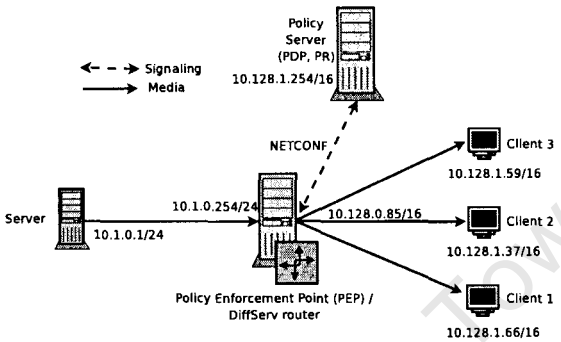

Figure 5.1: Detailed representation of the evaluation platform.

5.2.1 Validation Scenarios

To validate the operation of the system two scenarios were considered. In the first scenario three flows are sent from the server to clients 1 and 2. Client 1 receives UDP traffic representing traffic from IP telephony applications; in this case two UDP flows are sent with packets marked with the EF DSCP. The first EF flow is sent at a rate of 1.75 Mbps, whereas the second EF flow is sent at a rate of 1 Mbps. Client 2 receives best-effort traffic, marked with the DF DSCP. This DF flow is sent at a rate of 9 Mbps. Table 5.1 describes the three flows.

Table 5.1: Description of the traffic flows used for the first validation scenario.

Flow	DSCP	DSCP	Rate
1	DF	TCP	9 Mbps
2	EF	UDP	1.75 Mbps
3	EF	UDP	1 Mbps

Table 5.2 describes the periods at which the various flows are transmitted. The DF

flow (flow 1 on Table 5.2) is present throughout the entire testing period. The first EF flow (flow 2 on Table 5.2) starts at time 60s and stops at time 140s; the second EF flow (flow 3 on Table 5.2) starts at time 100s and also stops at time 140s.

Table 5.2: Description of the times at which the flows are sent in the first validation scenario.

Time (s)	0	...	60	...	100	...	140
Flows	1	1	1, 2	1, 2	1, 2, 3	1, 2, 3	1

In the second scenario an AF4 traffic flow is added from the server to client 3. Furthermore only one EF flow is present. The AF41 flow represents traffic generated from multimedia conferencing applications. Table 5.3 describes the rates used for the second scenario. Best-effort traffic is transmitted at 9 Mbps; EF traffic is sent at a constant rate of 2Mbps; and finally AF41 traffic is transmitted at 5 Mbps.

Table 5.3: Description of the traffic flows used for the second validation scenario.

Flow	Protocol	DSCP	Rate
1	TCP	DF	9 Mbps
2	UDP	EF	2 Mbps
3	TCP	AF41	5 Mbps

Table 5.4 describes the periods at which the flows in the second test scenario are transmitted. The server begins by sending the third flow representing AF41 traffic produced by multimedia conferencing applications; this flow stops at time 80 s. In the mean time, at time 30 s, the first flow representing best-effort traffic starts; this flow is active until the end of the testing period. At time 60s the EF flow starts (flow 2 on Table 5.4) and stops at time 120s; thus from time 60s to 80s the three flows are active.

Table 5.4: Description of the times at which the flows are sent in the second validation scenario.

Time (s)	0	...	30	...	60	...	80	...	120	...	160
Flows	3	3	3, 1	3, 1	3, 1, 2	3, 1, 2	1, 2	1, 2	1	1	1

5.2.2 QoS Configuration

Based on the test scenarios described in the previous section one DiffServ edge router configuration is created. The configuration contains three DiffServ classes required

by IP telephony traffic (EF DSCP), multimedia conferencing traffic (AF4 DSCP with three drop precedence levels), and standard traffic (best-effort traffic - DF DSCP). A single NLP defines the QoS configuration. It contains three *PolicyRules* corresponding to the DiffServ classes.

The NLP specifies that EF traffic is to be policed using a single rate (2 Mbps) and burst size meter (SB). Any exceeding EF traffic is dropped. Because the traffic is inelastic and requires low delays, no Active Queue Management (AQM) is applied and the traffic is given high priority over other traffic. In order to protect other classes from starving, EF traffic is limited to a maximum rate of 2 Mbps.

AF4 traffic is metered using a two-rate Three Color Marker (trTCM) that marks traffic to three drop precedence classes. The trTCM has two rates and two bust sizes. The committed rate is 2 Mbps and the peak rate is 3 Mbps. The committed burst size is 4 KB whereas the peak burst size is 8 KB. The trTCM is defined using two *QoSPolicyTokenBucketTrfcProf* elements, each one of them representing a single (rate, burst size) pair. Because AF4 traffic is elastic, after it is metered, some AQM is applied to it; the RED algorithm is used with three different drop probabilities for each drop precedence class (AF41, AF42, and AF43). 30 % of the available network bandwidth is guaranteed to this class.

Finally, best-effort traffic marked with the DF DSCP is not metered and is assigned the lowest priority. However, 50 % of the bandwidth is guaranteed to this class. In order to emulate congestion, the link rate is set to 10 Mbps, which is less than the sum of the transmitted rates of each flow in both validation scenarios. The next section presents the results of the validation tests.

5.2.3 Validation Results

Figure 5.2 shows the rates at which packets arrive at clients 1 and 2 when the PEP is not configured by the PDP. It can be seen that the rates of all three flows are very unstable. Before the start of the first EF flow, the DF flow is received at approximately 9 Mbps. When the first EF flow starts, the rate of the DF flow decreases to 7 Mbps and the EF flow is received at 2 Mbps. At time 100s, when the second EF flow starts, the rates of both EF flows add to 3 Mbps, while the rate of the DF flow is reduced to approximately 6 Mbps.

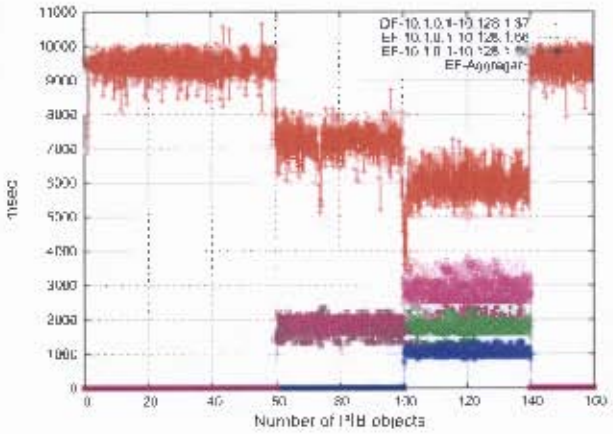

Figure 5.2: Results of the validation scenario 1 without the PBNM system.

Figure 5.3 shows the effect of applying the QoS policies. It is noticeable that the rates of all flows are more stable, in particular the EF flows that require steady rates. From time 0s to 60s DF traffic is received at 9 Mbps, which is close to the total available bandwidth. At time 60s EF packets are transmitted at a bit rate of 1.75 Mbps. Since this traffic is conforming to the defined meter, the rate is maintained, whereas the rate of the DF flow is reduced to approximately 8 Mbps. At time 100s the second EF flow is transmitted at a rate of 1 Mbps. At this stage the total EF rate is 2.75, which is more than the rate specified in the meter (i.e. 2 Mbps); thus some EF packets are dropped and the aggregate EF rate received at client 1 is 2 Mbps, i.e., the maximum allowed rate.

Figure 5.4 shows the rates at which the three flows in validation scenario 2 are received at clients 1, 2, and 3. Note that the DF and AF4 flows are bursty and are sharing bandwidth. In addition, the inelastic EF flow poses the treat of starving the other flows.

Figure 5.5 shows the effect of applying the QoS policy. At time 0s to 40s AF41 traffic is transmitted at 5 Mbps. Since enough bandwidth is available, the flow receives the required bandwidth. However, since the rate of the traffic flow exceeds the peak rate defined in the trTCM, some of the AF41 traffic is remarked to AF42 and AF43. Figure 5.5 clearly shows the effect of remarking the flow. It is clear that more packets are remarked to AF43, the class with the highest drop precedence. At time 30s, packets marked with the DF DSCP are sent at 9 Mbps. At this stage the total transmitted

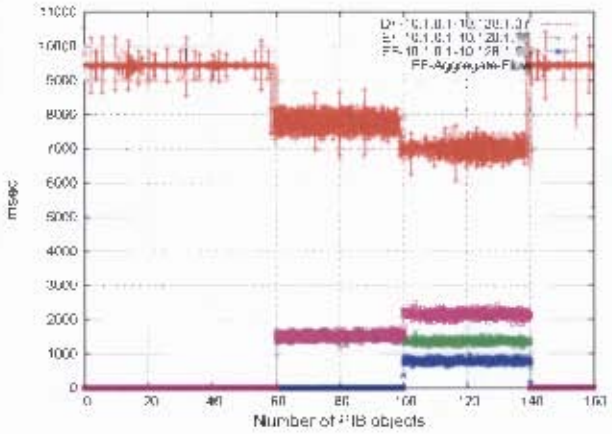

Figure 5.3: Results of the validation scenario 1 with the PBNM system active.

bandwidth is 14 Mbps, which exceeds the available bandwidth. Thus, some DF packets are lost, and the rate of the DF flow is limited and maintained at 5 Mbps, which is the minimum rate specified by the NLP. The AF4x traffic gets the remaining share of the total bandwidth.

At instant 60s the EF flow is sent at a rate of 2 Mbps. Since the link is already congested, the EF flows is accommodated at the expense of the AF4x flows. This is due to the fact that EF traffic has higher priority. Furthermore, the NLP specifies that AF4 traffic gets at least 30 % of bandwidth, and this is what happens - the rate of the aggregate AF4x flows is reduced to almost 3 Mbps. At this point the bandwidth is distributed proportionally according to the minimum bandwidths defined in the NLP. However, it should be noted that the aggregate rate of the AF4x flows is reduced at the expense of AF43 traffic, which has the highest drop precedence value. Therefore, from time 60s, when the EF flow starts, to time 80s, when the AF4x flow stops, almost all AF43 packets are lost, and the rate of the AF43 flow is reduced to very low levels.

In summary, this section validates the operation of the proposed system, while at the same time demonstrating the usefulness of QoS management. The next section presents the performance test carried on the evaluation platform.

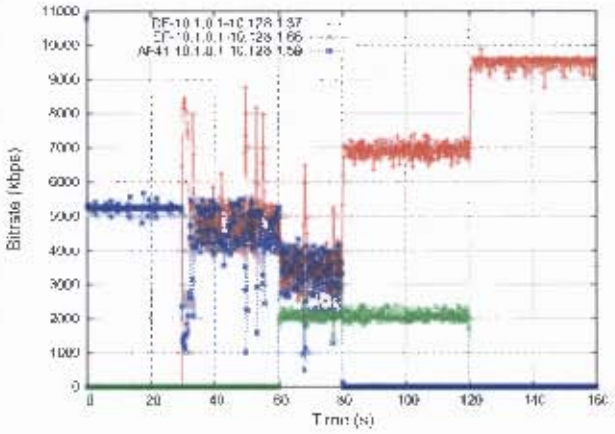

Figure 5.4: Results of the validation scenario 2 without the PBNM system.

5.3 Performance Study

In this section the performance of the proposed system is compared to the performance of a COPS-PR-based PBNM system. A COPS-PR-based PBNM system is used as basis for comparison due to the fact that COPS-PR is the most widely known protocol for QoS policy provisioning. Furthermore COPS-PR messages are usually binary-encoded, following the Basic Encoding Rules (BER), resulting in shorter messages and good usage of network bandwidth and system resources.

For the COPS-PR/DiffServ system the author uses the COPS-PR library developed by the Networks and Protocols Group of the Tampere University of Technology [72]. This implementation was chosen because, like Yenca (i.e., the NETCONF implementation), it is written in C, and it is simple and small, providing a good basis for comparison. The library was used to develop a simple PDP and PEP for the purposes of this study.

The parameters taken into consideration when comparing the performance of both schemes were network usage when the PDP sends decisions to the PEP, and the system resource usage at the PEP when it parses and saves policy decisions, and sends replies to the PDP. Policy decisions are sent from the PDP to the PEP using NETCONF's *<copy-config>* operation in the case of the system proposed in this thesis, and the *DEC* operation in the case of the COPS-PR-based system. Due to the verbose nature of XML, different algorithms are used to compress XML messages before transmission at the PDP and to decompress them for processing at the PEP. The impact on the

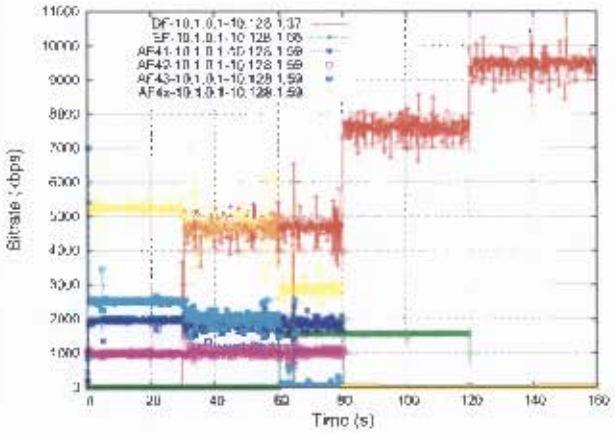

Figure 5.5: Results of the validation scenario 2 with the PBNM system active.

performance of the system is studied. The compression/decompression[1] libraries used in the study are zlib [67], XMill [73], and XMLPPM [74]. zlib is a widely used general-purpose data compression library, whereas XMill and XMLPPM are XML-conscious compression libraries.

One important factor to note is that the XML-based PEP only embeds the zlib compression code. On the other hand XMill and XMLPPM compression are made possible by two external utilities that are executed directly in the Linux shell. This is because the zlib library is mature, and widely spread and supported.

The performance tests were carried in five iterations, ranging from simple to complex. In each iteration, DiffServ policies representing the various IETF-recommended PHBs were provisioned to the two PEPs (the XML-based PEP, and the COPS-PR-based PEP). Table 5.5 gives a summary of the five iterations. The number of objects transmitted is increased from 67 in iteration 1, to 373 in the last iteration.

Table 5.6 provides the specifications of the PDP and PEP. The machine playing the role of PDP runs both systems (i.e., the one proposed in this study, using NETCONF for policy provisioning, and the COPS-PR-based system). The same applies for the PEP.

[1] When the author refers to a compression library it is assumed that the same library is used for compression as well as decompression.

Table 5.5: Provisioned policies.

Iteration	Provisioned Classes	Number of Objects
1	EF	67
2	AF4x	113
3	EF, AF4x	213
4	EF, AF4x, AF3x	293
5	EF, AF4x, AF3x, AF1x	373

Table 5.6: Hardware and software configuration.

	PDP	PEP
Hardware	AMD Sempron 2600+ 705MB RAM	Pentium II, 233 MHz 450 MB RAM
Operating System	Linux, Ubuntu	Linux, Ubuntu

5.3.1 Network Traffic

Figure 5.6 shows the network traffic generated by the two PDPs as they go through the iterations described in Table 5.5. The results show that the proposed XML-based system generates approximately 3 times more traffic than the COPS-PR based system. As discussed before, this was expected due to the verbosity of XML messages.

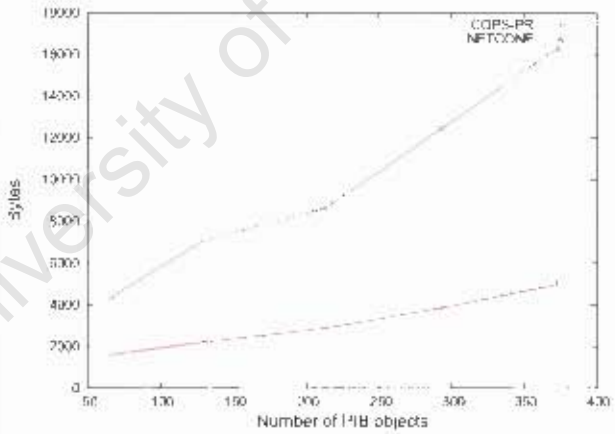

Figure 5.6: Network usage

Figure 5.7 shows the effect of NETCONF message compression on network traffic usage. The results indicate that when more than a certain number of objects is transmitted the NETCONF schemes using compression perform better than COPS-PR. This

number of objects varies according to the compression scheme used. The XMLPPM compression scheme performs better, by a small margin, than the zlib compression scheme. NETCONF with XMLPPM compression outperforms COPS-PR when more than 240 objects are sent, whereas NETCONF with zlib compression performs better than COPS-PR when more than 260 objects are transmitted. XMill compression proved to be the worst compression scheme. Nonetheless it performs better than COPS-PR when more than 310 objects are transmitted. Note that as the size of XML messages increases the compression rates improve.

These results are in-line with the results presented by Pras et. al. [40] and Franco et. al. [10], as discussed in Chapter 2.

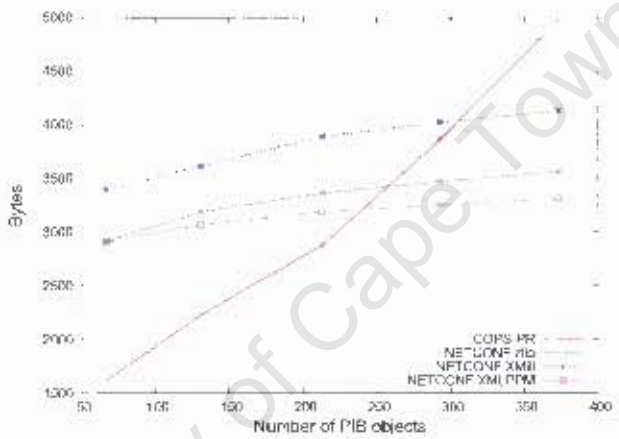

Figure 5.7: Network usage when compression is applied to NETCONF messages

5.3.2 System Resource Usage

To compare the processing delays and memory consumption the software running at the PEP was modified to include measurement code. The *gettimeofday* function was used to measure processing delay; it gets the current time in microseconds. The processing delay is computed by determining the actual time difference between the start and end of an operation or group of operations. To increase the precision of the measurement each measurement was taken 1000 times and the average was determined. To measure the memory consumed, the Valgrind library [75] and the UNIX *top* utility were used.

Please note that only QoS policy decisions are compressed before transmission at the PDP. The replies sent from the PEP to the PDP are not compressed before transmission. This is because all reply messages sent from the PEP are very short in size.

Processing Delays

Figure 5.8 shows the processing delays incurred at the PEP when is parses and saves decisions, and sends replies to the PDP. The measurements show the PEP proposed in this thesis (based on NETCONF) generates almost twice more processing delays. This is explained by the fact that XML messages go through complex parsing and validation before data is extracted. Nonetheless the difference is small, ranging from from 5 to 11 milliseconds.

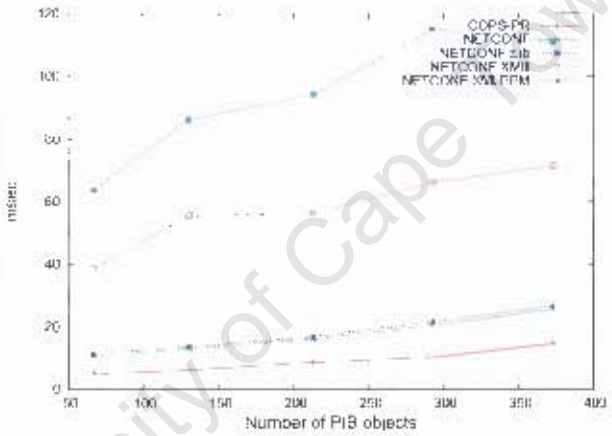

Figure 5.8: Processing delays

Figure 5.8 also shows the effect of decompression of NETCONF messages at the PEP on the processing latencies. zlib decompression performs significantly better than the other two XML-specific compression schemes. The added latencies due to zlib decompression are insignificant, ranging from 0.5 to 1 milliseconds. Zlib is 3 to 4 times faster than XMill, with a difference in processing delays ranging from 27 to 45 milliseconds. XMLPPM gives the worst performance with processing delays 4 to 6 times slower than zlib.

The poor performance of XMill and XMLPPM can be explained by the fact that these schemes generate an extra compression/decompression overhead to explore the

features of XML documents in order to achieve better compression rates .

The results presented in this section differ considerably from results presented by Franco el. al [10] (i.e, Franco et. al. concludes that a NETCONF-based PEP incurs less processing delays than a COPS-PR-based PEP). This is because the two implementations used for comparison studies in this thesis have the same degree of complexity. On the other hand, the two implementations used by Franco et. al. have different complexities. In particular the COPS-PR implementation used by Franco et. al. is a lot more comprehensive than the NETCONF implementation.

Memory Usage

The other parameter considered as part of the system resource usage was the memory consumed by both PEPs. Memory is needed to hold a process's instructions as well as data. Furthermore memory can also be allocated dynamically at run-time. Table 5.7 shows the instructions as well as data memory requirements. The XML-based PEP requires more memory to hold the code and static data. However the difference is insignificant.

Table 5.7: Instructions and data memory.

PEP	Instructions	Data
COPS-PR-based system	12 KB	252 KB
XML-based system	20 KB	242 KB

Figure 5.9 shows that there is a considerable difference in the memory allocated dynamically by both PEPs. The XML-based PEP allocates around 5 times more memory than its COPS-PR counterpart. This can be explained by the fact that the XML-based PEP uses a DOM parser to process XML messages. To access a portion of an XML document the parser loads the DOM tree of the whole document into memory, and this takes up considerable resources. To achieve better performance a parser using the Simple API for XML (SAX) processing should be used. SAX parsers are usually much lighter than DOM parsers since they read XML documents sequentially and generate events as they find elements, attributes, or text. Although this is cumbersome for applications, the memory consumption can be greatly improved.

The results in Figure 5.9 show that when the XML-based PEP decompresses NETCONF messages using zlib only a small extra fraction of memory is allocated, i.e., an

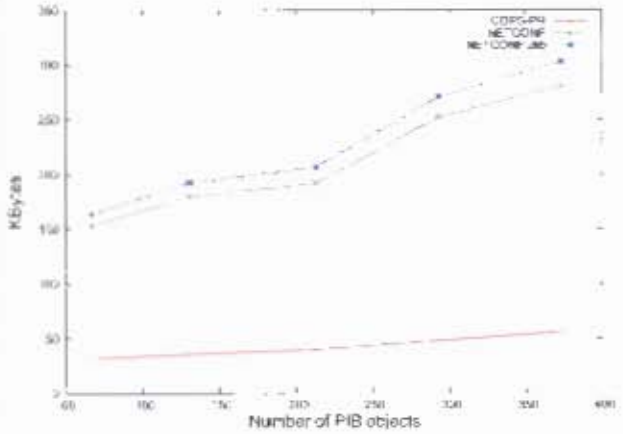

Figure 5.9: Dynamic memory allocations.

additional 10kB to 22kB are allocated. On the other hand, as depicted in Figure 5.10, the other two compression schemes allocate an amount of memory that is prohibitively larger than the zlib scheme. Specifically, XMill requires 10 to 20 times more memory allocation than zlib, whereas XMLPPM gives the worst performance requiring 60 to 90 times more memory allocation.

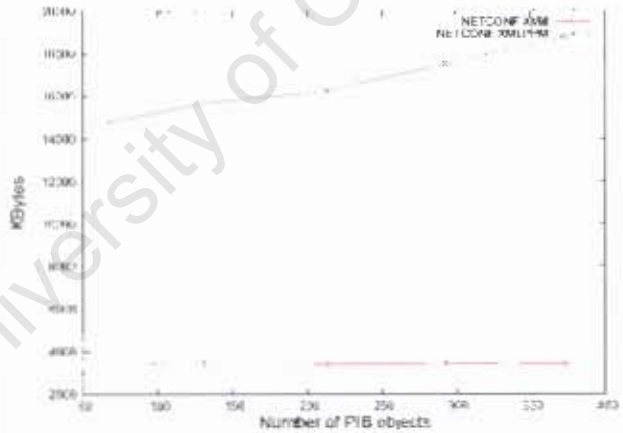

Figure 5.10: Dynamic memory allocations using alternative compression algorithms.

Comparative processing delays at the PEP

The previous section presented a performance comparison between an implementation of the system proposed in this thesis and a COPS-PR-based implementation. In particular it presented the processing delays incurred by the PEPs when they parse and save requests. However, apart from parsing and saving requests the PEP developed in this study also translates, transforms and applies DiffServ policies to the DiffServ router. As such it is important to study the impact of these execution stages on the performance of the PEP. This section focuses solely on the implementation of the system proposed in this thesis and examines the performance of the PEP during various stages of execution. Three execution stages where chosen for this particular study:

- Stage A - the DiffServ policy translation stage.
- Stage B - the *tcng* configuration transformation stage.
- Stage C - the *tcng* policy application stage. This involves the translation of *tcng* to *tc* scripts, and the execution of the *tc* scripts.

From Figure 5.11 one factor stands out; specifically, the results show that more than half of the total processing time of the PEP is spent in stage C, when the tcng scripts are translated and applied to the router. Hence the performance deterioration in terms of processing delays resulting from the use of XML is less important than the performance deterioration resulting from the application of the QoS configurations to the router. Thus a system equipped with a fast encoding/decoding scheme would achieve comparable processing delays, since the penalties resulting from the decoding of XML are much lower than the penalties resulting from the application of the policies to the system.

However, these results should not be overestimated since the most significant performance penalties are related to dynamic memory consumption by the proposed XML-based system, and not the incurred processing delays.

Comparative processing delays at the PDP

A similar study is conducted for the performance of the PDP with regards to the performance delays incurred in various execution stages. In order to use the same

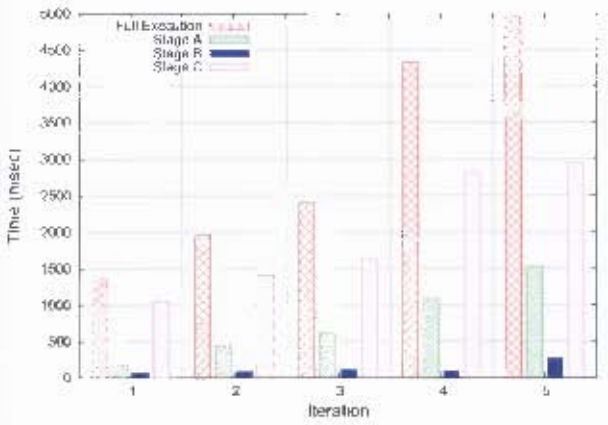

Figure 5.11: Comparative processing delays at the PEP.

iterations described previously, 5 NLPs were created in a way that their translation results in DiffServ policies containing the classes described in Table 5.5. The execution stages considered for this study are as follows:

- Stage A - when the PDP fetches roles and capabilities from a single PEP; this includes encoding of the request and decoding of the response.
- Stage B - when the PDP fetches NLPs from the PR using roles supplied by a single PEP.
- Stage C - when the PDP translates NLPs to DLPs.
- Stage D - when the PDP sends DLPs to a single PEP, receives and decodes the PEP's reply, and saves the translated DLPs.

Figure 5.12 shows the results. It is evident that the PDP spends more time on stage D, when it is waiting for a reply from the PEP. This is expected, since, as demonstrated previously, the PEP takes considerable time to apply the DiffServ policies. The second factor that stands out is that the complexity of the translation algorithm is the least important factor affecting the overall performance of the PDP as far as processing delays are concerned. The third factor that stands out is that when the PEP has to apply the policies to the router it takes up to 6 times the amount of time its takes when it simply has to parse requests for roles and capabilities, and send replies. This stresses

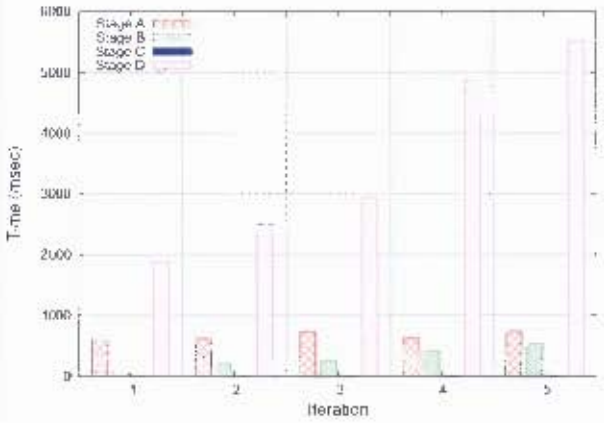

Figure 5.12: Comparative processing delays at the PDP.

the fact that the encoding/decoding and parsing of XML is not the most important factor affecting the performance of the PEP as far as processing delays are concerned. Finally, from Figure 5.12, in iteration 5, the time taken for the PDP to fetch the roles and capabilities from a single PEP (including the encoding of the request and decoding of the response) is close to 25 % more than the time taken by the PDP to fetch NLPs from the PR using roles supplied by the PEP. Thus, in scenarios where the sizes of NLPs are large it would be more feasible to have the PDP and PR co-located to speed up the execution of the system and reduce network traffic. This is, of course, only possible if one PDP is used per management domain.

5.4 Discussion

This chapter presented several tests carried on the evaluation platform to validate the operation of the proposed system, as well as to evaluate its performance. The validation tests carried successfully demonstrated the correct operation of the system, as well as the importance of QoS management in IP networks.

The performance evaluation tests show that the proposed system generates approximately 3 times more network traffic than its COPS-PR counterpart. However when NETCONF messages are compressed, the XML-based system out-performs the COPS-PR-based system when more than 260 objects are sent, while using zlib compression.

The performance results show that the XML-based system incurs more processing delays than the system based on COPS-PR. However the difference is small, ranging form 5 to 11 milliseconds. The added latencies due to zlib decompression are insignificant. The two other XML-conscious compression algorithms perform considerably worst than zlib, incurring 3 to 6 times more latencies than zlib.

The worst performance indicator for the XML-based system is the dynamic memory consumed at the PEP during execution. The results show that the XML-based PEP needs around 5 times more dynamic memory than its COPS-PR counterpart. This is because a DOM parser was used to process XML messages. DOM parsers are known to be heavy-memory consuming due to the fact that they have to load entire XML documents into memory in order to process the information. To reduce memory consumption, a SAX parser should be used in favour of the DOM parser. SAX processes XML documents serially therefore requiring less memory than DOM.

The memory consumption results also show that to decompress XML messages with zlib, only a small fraction of additional memory is allocated (10kB to 22kB). On the other hand, the XMill and XMLPPM decompression schemes require amounts of memory that are prohibitively larger than the zlib scheme. Specifically, XMill requires 10 to 20 times more memory allocations than zlib, whereas XMLPPM requires 60 to 90 times more memory. Therefore, overall, zlib was found to give the best performance.

Finally, when comparing the processing delays incurred by the various execution stages at the PEP and PDP, it was found that more than half of the total processing time at the PEP is spent applying the *tcng* scripts to the DiffServ-enabled IP router. This suggests that the performance deterioration in terms of processing latencies resulting from the use of XML is less significant than the performance deterioration resulting from the execution of the final QoS configurations.

Chapter 6

Conclusions and Recommendations

6.1 Conclusions

The objective of this study is to apply XML technologies in the design and implementation of a Policy-Based QoS Management framework in order to improve its usability and manageability, and reduce development costs. XML has powerful modelling capabilities and allows the design of data models of arbitrary complexity; it enables a seamless representation of nested structures, and provides simple but powerful naming schemes. XML protocol messages are text-based and therefore human readable, hence XML-based protocols are easier to use. Furthermore, because of widespread support for text- and XML-based processing tools the development costs and deployment cycles of applications can be reduced.

As part of this work an XML-based QoS policy data model has been developed. The data model is divided into three abstraction layers - the NLP layer, the DLP layer, and the layer defining device-specific configuration policies. The NLP layer enables the system to hide the network's intricacies from a network administrator, whereas the DLP layer enables the system to manage QoS in devices made by different vendors, supporting different configuration commands. Device-specific configuration policies are tightly related to the DiffServ router's implementation technology. QoS Policies at the three abstraction layers are modelled and represented in XML, thus simplifying the system's internal structure. As part of the system design two policy translation algorithms were defined: one translates NLPs to DLPs, and the other translates DLPs

to device-specific configuration policies.

An evaluation platform was developed to implement the proposed Policy-Based QoS Management framework. This platform was used to validate the proposed ideas and to study the performance of the system, as compared to a PBNM system based on COPS-PR and an SPPI-based DiffServ PIB. Tests carried on the evaluation platform confirm that the proposed system performs correctly and achieves satisfactory performance results.

Based on the experiences taken from the design and implementation of the system the author draws the following conclusions:

- The IETF/DTMF policy information models - PCIM/PCIMe and QPIM - can be easily mapped to XML Schemas. They provide a good basis for the modelling of policies, since they support hierarchical structures and data re-usability. Those features are mapped seamlessly to XML documents.

- The DiffServ PIB and Framework PIB can also be easily mapped to XML representations. The XML representations of those PIB modules are human-readable, self-describing, and easy to use. However, the author feels that these PIB modules contain unnecessary complexity, and since XML provides more powerful data modelling features that SPPI, a totally new DiffServ data model should be designed based on XML, to be used with NETCONF. This data model should fully explore all the important features of XML documents.

- The division of the policy data model into abstraction layers facilitates network administration, since higher-level policies are more intuitive for an administrator. However the author came to the conclusion that the translation of a PCIM/QPIM-based policy to a DiffServ PIB-based policy is very complex, and limits the flexibility of the data model. This emphasises the fact that a new DiffServ data model would be useful since it would probably simplify this translation process.

- As compared to COPS-PR, NETCONF is simple and cost-effective from the perspective of the network administrator, since it is text-based, and its functionality closely mirrors the functionality of the managed device. Furthermore, the fact

that the same tools are used to process protocol messages and policy configurations speeds up application development. However, with regard to device failure recovery, COPS-PR is a more attractive solution than NETCONF for the PDP-PEPs interaction. This is because with NETCONF the PDP has to manage the connections to all PEPs and if any of these connections breaks down the PDP has to re-establish it.

- As far as performance is concerned, the proposed XML-Driven PBNM system generates approximately 3 times more network traffic than the COPS-PR-based system. However, when NETCONF messages are compressed, the network traffic generated by the XML-based PDP is reduced by up to a factor of 5, and the proposed system performs better than the COPS-PR based system when more than 260 management objects are sent. Overall, the zlib compression library achieves the best performance when compared to two other XML-conscious compression utilities.

- The COPS-PR system incurs less processing delays than the proposed XML-based system. However the difference is insignificant, ranging from 0 to 6 milliseconds. On the other hand the proposed system requires around 5 times more dynamic memory allocations than its COPS-PR counterpart. This is primarily because the PEP was implemented using a DOM parser; DOM parsers are known to be memory-consuming, but easy to use from an application developer's perspective.

- The added processing latency and memory consumption due to zlib decompression of NETCONF messages is negligible. Therefore, by using compression/decompression of NETCONF messages the gains in terms of reduced network traffic outweigh the losses in terms of processing overhead at the PEP.

- When the PEP applies DiffServ policies to the router it takes up to 6 times more the amount of time its takes when it simply has to parse the PDP's request, locate and load the requested roles and capabilities, and send the reply. As such, the encoding/decoding and parsing of XML is not the most important factor affecting the performance of the PEP as far as processing delays are concerned. The most important factor is the application/execution of the final QoS configurations on the router.

In summary, the author concludes that by using XML in the design and implementation of the system, its entire structure is simplified, while adding flexibility to the data model, and opening up possibilities for integration with other systems. Although XML raises concerns regarding the performance of systems, due to its verbose nature, the performance results shows that the proposed XML-driven PBNM system achieves satisfactory and, in certain areas, better results then the COPS-PR-based system.

6.2 Recommendations

As a result of the scope, limitations, and conclusions of this study, the following recommendations are made for future work:

- This study did not attempt to develop a business-level QoS policy model. By making use of business policies, the PBNM system ensures that QoS management closely reflects the business goals of the organisation. As such, it is recommended that a suitable business-level policy model is developed in future to simplify network administration.
- The translation algorithms developed in this thesis do not support the creation of NLPs representing hierarchical schedulers with more than 2 levels. It is thus recommended that further investigation is conducted on the development of more flexible policy translation algorithms.
- Although the system proposed in this thesis simplifies and automates QoS management, there is still room for improvement in this regard. One way to achieve this is to use web service composition and orchestration to create high-level services from low-level operations using predefined patterns. This is an emerging field of research and can further improve the automation of PBNM systems, therefore further investigations in this area are encouraged.
- The PBNM system developed in this thesis operates in a static manner, i.e., policies are only reconfigured by the network administrator. However real automation is only possible if the system is able to reconfigure itself. One way to achieve this is to have the system monitor the network and to have policies that control QoS policies. In this way, based on monitoring feedback, QoS policies

would be adjusted by control policies so that the network would automatically adjust to new conditions.

- The PBNM system developed in this study does not take into consideration external events. However, to improve automation it is required that the decision process at the PDP be triggered by external events. One scenario where this would be necessary is when managing QoS in the IP Multimedia Subsystem (IMS) [61]. In this case the PDP has to react to external events such as QoS requests arriving from a Proxy-Call Session Control Function (P-CSCF) entity. The translation logic in the PDP would consider external factors such as the service information conveyed in the SDP field of SIP messages received at the P-CSCF; the information would be used by the PDP to select the appropriate NLPs in the PR. This is an emerging field of research that requires further investigation.

Bibliography

[1] "The Internet Socitety - ISOC." http://www.isoc.org/internet/history/, Nov. 2006.

[2] J. C. Strassner, *Policy-Based Network Management: Solutions for the Next Generation.* Morgan Kaufmann, 2003.

[3] M. Li and K. Sandrasegaran, "Network Management Challenges for Next Generation Networks," in *LCN '05: Proceedings of the The IEEE Conference on Local Computer Networks 30th Anniversary*, (Sydney, Australia), pp. 593–598, IEEE Computer Society, 2005.

[4] P. Goncalves, J. L. Oliveira, and R. L. Aguiar, "A WBEM based Solution for a 4G Network Integrated Management," in *Autonomic and Autonomous Systems and International Conference on Networking and Services, ICAS-ICNS*, pp. 29–29, Oct. 2005.

[5] K. Chan, J. Seligson, D. Durham, S. Gai, K. McCloghrie, S.Herzog, F. Reichmeyer, R. Yavatkar, and A. Smith, "COPS Usage for Policy Provisioning (COPS-PR)." RFC 3084 (Proposed Standard), Mar. 2001.

[6] K. McCloghrie, M. Fine, J. Seligson, K. Chan, S. Hahn, R.Sahita, A. Smith, and F. Reichmeyer, "Structure of Policy Provisioning Information (SPPI)." RFC 3159 (Proposed Standard), Aug. 2001.

[7] F. J. G. Clemente, G. M. Pérez, and A. Gómez-Skarmeta, "An XML-Seamless Policy Based Management Framework," in *Third International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security, MMM-ACNS 2005*, pp. 418–423, Sep. 2005.

[8] J. Schoenwaelder, A. Pras, and J.-P. Martin-Flatin, "On the Future of Internet Management Technologies," *IEEE Communications Magazine*, vol. 41, pp. 90–97, Oct. 2003.

[9] J. Schoenwaelder, "Overview of the 2002 IAB Network Management Workshop." RFC 3535 (Informational), May 2003.

[10] T. F. Franco, W. Q. Lima, G. S. andRafael Corezola Pereira, M. J. B. Almeida, L. R. Tarouco, L. Z. Granville, AndreBeller, E. Jamhour, and M. Fonseca, "Substituting COPS-PR: An Evaluation of NETCONF and SOAP for Policy Provisioning," in *POLICY '06: Proceedings of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'06)*, pp. 195–204, IEEE Computer Society, Jul. 2006.

[11] T. Bray, J. Paoli, and C. M. Sperberg-McQueen (Eds), "Extensible Markup Language (XML) 1.0 (Fourth Edition)." W3C Recommendation, 2000.

[12] F. Strauss and T. Klie, "Towards XML Oriented Internet Management," in *8th IFIP/IEEE International Symposium on Integrated Network Management*, (Colorado Springs), pp. 505–518, Mar. 2003.

[13] L. E. Menten, "Experiences in the Application of XML for Device Management," *Communications Magazine, IEEE*, vol. 42, pp. 92– 100, Jul. 2004.

[14] M.-J. Choi, J. W. Hong, and H.-T. Ju, "XML-Based Network Management for IP Networks," *Electronics and Telecommunications Research Institute journal*, vol. 25, no. 6, pp. 445–463, 2003.

[15] R. Enns, "NETCONF Configuration Protocol." RFC 4741 (Standards Track), Dec. 2006.

[16] A. Beller, E. Jamhour, and M. E. Pellenz, "Defining Reusable Business-Level QoS Policies for DiffServ," in *IFIP/IEEE International Workshop on Distributed Systems Operations and Management, DSOM 2003*, pp. 40–51, Nov. 2004.

[17] R. Yavatkar, D. Pendarakis, and R. Guerin, "A Framework for Policy-based Admission Control." RFC 2753 (Informational), Jan. 2000.

[18] M.-J. Choi, *An Architectural Framework for XML-based Network Management.* PhD thesis, Pohang University of Science and Technology, Department of Electrical and Computer Engineering, December 18 2004.

[19] B. Daum and U. Merten, *System Architecture with XML.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

[20] P. L. Hégaret, S. Byrne, M. Champion, L. Wood, A. L. Hors, G. Nicol, and J. Robie, "Document Object Model (DOM) Level 3 Core Specification," W3C Recommendation, World Wide Web Consortium (W3C), Apr. 2004. http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407.

[21] D. Megginson, "Simple API for XML," May 2000. http://www.megginson.com/SAX/.

[22] D. Chamberlin, M. Kay, J. Robie, M. F. Fernández, J. Siméon, S. Boag, and A. Berglund, "XML Path Language (XPath) 2.0," W3C Recommendation, World Wide Web Consortium (W3C), Jan. 2007. http://www.w3.org/TR/2007/REC-xpath20-20070123/.

[23] E. Maler, J. Marsh, N. Walsh, and P. Grosso, "XPointer Framework," W3C Recommendation, World Wide Web Consortium (W3C), Mar. 2003. http://www.w3.org/TR/2003/REC-xptr-framework-20030325/.

[24] J. Marsh, D. Veillard, and D. Orchard, "XML Inclusions (XInclude) Version 1.0 (Second Edition)," W3C Recommendation, World Wide Web Consortium (W3C), Nov. 2006. http://www.w3.org/TR/2006/REC-xinclude-20061115/.

[25] M. Kay, "XSL Transformations (XSLT) Version 2.0," W3C Recommendation, World Wide Web Consortium (W3C), Jan. 2007. http://www.w3.org/TR/2007/REC-xslt20-20070123/.

[26] Y. Lafon and N. Mitra, "SOAP Version 1.2 Part 0: Primer (Second Edition)," W3C Proposed Edited Recommendation, World Wide Web Consortium (W3C), Dec. 2006. http://www.w3.org/TR/2006/PER-soap12-part0-20061219/.

[27] D. Winer, "XML-RPC Specification," June 1999. http://www.xmlrpc.com/spec.

[28] H.-T. Ju, M.-J. Choi, S. Han, Y. Oh, J.-H. Yoon, H. Lee, and J. W. Hong, "An Embedded Web Server Architecture for XML-Based Network Management," in *IEEE/IFIP Network Operations and Management Symposium (NOMS 2002)*, (Florence, Italy), pp. 5–18, April 2002.

[29] P. A. Shafer and R. Enns, "JUNOScript: An XML-based Network Management API." IETF Network Working Group, Internet-Draft, Aug 2002. http://xml.coverpages.org/draft-shafer-js-xml-api-00.txt.

[30] J. Schonwalder, "Characterization of SNMP MIB modules," in *9th IFI/IEEE Symposium on Integrated Management, 2005. IM'2005.*, pp. 615–628, 2005.

[31] T. Klie and F. Strauss, "Integrating SNMP agents with XML-based management systems," *IEEE Communications Magazine*, vol. 42, pp. 76 – 83, Jul 2004.

[32] J.-H. Yoon, H.-T. Ju, and J. W. Hong, "Development of SNMP-XML translator and gateway for XML-based integrated network management," *International Journal of Network Management*, vol. 13, no. 4, pp. 259–276, 2003.

[33] J. Martin-Flatin, *Web-Based Management of IP Networks and Systems.* PhD thesis, Swiss Federal Institute of Technology, Lausanne (EPFL), Oct 2000.

[34] H.-M. Choi, M.-J. Choi, and J. W. Hong, "Design and Implementation of XML-based Configuration Management System for Distributed Systems," in *IEEE/IFIP Network Operations and Management Symposium (NOMS 2004)*, vol. 1, (Seoul, Korea), pp. 831– 844, Apr 2004.

[35] "Web-Based Enterprise Management." The Distributed Management Task Force, Inc. http://www.dmtf.org/standards/wbem/.

[36] "Specification for the Representation of CIM in XML, Version 2.0." The Distributed Management Task Force, Inc., Dec 2004. http://www.dmtf.org/standards/wbem/DSP201.html.

[37] M. Champion, H. Haas, E. Newcomer, C. Ferris, D. Booth, D. Orchard, and F. McCabe, "Web Services Architecture," W3C Note, World Wide Web Consortium (W3C), Feb. 2004. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[38] S.-M. Yoo, H.-T. Ju, and J. W.-K. Hong, "Web Services Based Configuration Management for IP Network Devices," in *8th International Conference on Management of Multimedia Networks and Services (MMNS 2005), LNCS 3754*, pp. 254–265, Oct 2005.

[39] B. Thurm, "Web Services for Network Management - A Universal Architecture and Its Application to MPLS Networks," in *LCN '02: Proceedings of the 27th Annual IEEE Conference on Local Computer Networks*, (Washington, DC, USA), p. 0463, IEEE Computer Society, 2002.

[40] A. Pras, T. Drevers, R. van de Meent, and D. Quartel, "Comparing the Performance of SNMP and Web Services-Based Management," *IEEE eTNSM (Transactions on Network and Service Management)*, vol. 1, p. 11, December 2004.

[41] S. Subramanium, "XML Based Device Management," White Paper, Wipro Technologies, 2006.

[42] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B.Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, and S. Waldbusser, "Terminology for Policy-Based Management." RFC 3198 (Informational), Nov. 2001.

[43] B. Moore, "Policy Core Information Model (PCIM) Extensions." RFC 3460 (Proposed Standard), Jan. 2003.

[44] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen, "Policy Core Information Model – Version 1 Specification." RFC 3060 (Proposed Standard), Feb. 2001. Updated by RFC 3460.

[45] Y. Snir, Y. Ramberg, J. Strassner, R. Cohen, and B. Moore, "Policy Quality of Service (QoS) Information Model." RFC 3644 (Proposed Standard), Nov. 2003.

[46] D. Verma, "Simplifying Network Administration using Policy-Based Management," *IEEE Network Magazine*, vol. 16, pp. 20–26, Mar/Apr 2002.

[47] K. Chan, R. Sahita, S. Hahn, and K. McCloghrie, "Differentiated Services Quality of Service Policy Information Base." RFC 3317 (Informational), Mar. 2003.

[48] B. Moore, D. Durham, J. Strassner, A. Westerinen, and W. Weiss, "Information Model for Describing Network Device QoS Datapath Mechanisms." RFC 3670 (Proposed Standard), Jan. 2004.

[49] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Service." RFC 2475 (Informational), Dec. 1998. Updated by RFC 3260.

[50] F. Baker, K. Chan, and A. Smith, "Management Information Base for the Differentiated Services Architecture." RFC 3289 (Proposed Standard), May 2002.

[51] F. Strauss and J. Schoenwaelder, "libsmi - a library to access SMI MIB information." http://www.ibr.cs.tu-bs.de/projects/libsmi/, Nov. 2005.

[52] T. Klie and L. Wolf, "Autonomic Policy-based Management using Web Services," in *2nd Conference on Future Networking Technologies*, (Lisbon, Portugal), Dec 2006.

[53] K. S. Youn and C. S. Hong, "A Network Management Architecture Using XML-Based Policy Information Base," in *International Conference on Information Networking 2003, ICOIN 2003*, pp. 876–885, Feb. 2003.

[54] W. Almesberger, "Linux Traffic Control - Next Generation," in *9th International Linux System Technology Conference*, Sep 2002.

[55] M. Devera and D. Cohen, *HTB Linux queuing discipline manual - user guide*, 2002.

[56] L. Balliache, "Differentiated Service on Linux HOWTO." http://www.opalsoft.net/qos/DS.htm, Aug 2003.

[57] J. Babiarz, K. Chan, and F. Baker, "Configuration Guidelines for DiffServ Service Classes." RFC 4594 (Informational), Aug. 2006.

[58] J. Heinanen and R. Guerin, "A Two Rate Three Color Marker." RFC 2698 (Informational), Sept. 1999.

[59] J. Heinanen and R. Guerin, "A Single Rate Three Color Marker." RFC 2697 (Informational), Sept. 1999.

[60] T. Borosa, B. Marsic, and S. Pocuca, "QoS support in IP multimedia subsystem using DiffServ," in *7th International Conference on Telecommunications (ConTEL 2003)*, vol. 2, pp. 669– 672, Jun 2003.

[61] 3GPP TS 23.228, "IP Multimedia Subsystem (IMS); Stage 2," 2007.

[62] W. Meier, "eXist Version 1.0 - Open Source Native XML Database." Website: http://exist.sourceforge.net/, 2006.

[63] J. Borden, L. Martin, and K. Staken, "XML:DB Initiative for XML Databases." Website: http://xmldb-org.sourceforge.net/, 2003.

[64] J. Hunter, "JDOM." Website: http://www.jdom.org/, 2004.

[65] "Apache Web Services Project - Apache SOAP." Website: http://ws.apache.org/soap/, 2003.

[66] "The Apache Software Foundation - Apache Tomcat." Website: http://tomcat.apache.org/, 2007.

[67] M. Adler and J. loup Gailly, "zlib Compression Library." Website: http://www.zlib.net/, 2005.

[68] "NETCONF IETF Working Group." Website, 2006.

[69] J. Bourdelon and R. State, "YENCA Network Management Framework." Website: http://sourceforge.net/projects/yenca/, 2004.

[70] "MADYNES - INRIA-Nancy Universités Research Team." Website: http://madynes.loria.fr/.

[71] "D-ITG, Distributed Internet Traffic Generator." http://www.grid.unina.it/software/ITG/, 2007. Dipartimento di Informatica e Sistemistica, Universita' degli Studi di Napoli "Federico II" (Italy).

[72] "Networks and Protocols Group, Tampere University of Technology." http://www.cs.tut.fi/tlt/npg.html.

[73] H. Liefke and D. Suciu, "XMill: an efficient compressor for XML data," in *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 153–164, ACM Press, 2000.

[74] J. Cheney, "XMLPPM: XML-Conscious PPM." http://xmlppm.sourceforge.net/, Nov 2004.

[75] C. Armour-Brown, J. Fitzhardinge, T. Hughes, N. Nethercote, P. Mackerras, D. Mueller, J. Seward, R. Walsh, and J. Weidendorfer, "Valgrind." http://valgrind.org/, 2006.

[76] Y. Bernet, S. Blake, D. Grossman, and A. Smith, "An Informal Management Model for Diffserv Routers." RFC 3290 (Informational), May 2002.

[77] V. Jacobson, K. Nichols, and K. Poduri, "An Expedited Forwarding PHB." RFC 2598 (Proposed Standard), June 1999. Obsoleted by RFC 3246.

[78] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski, "Assured Forwarding PHB Group." RFC 2597 (Proposed Standard), June 1999. Updated by RFC 3260.

[79] R. Sahita, S. Hahn, K. Chan, and K. McCloghrie, "Framework Policy Information Base." RFC 3318 (Informational), Mar. 2003.

Appendix A

Differentiated Services

The IETF Differentiated Services (DiffServ) architecture was designed to implement service differentiation in the Internet while achieving scalability. Whereas the Integrated Services (IntServ) architecture is designed to provide QoS through resource reservation for individual flows, DiffServ classifies each data packet into a limited number of pre-defined traffic classes. Each router in a DiffServ domain would be configured to differentiate traffic based on intrinsic characteristics of these classes.

DiffServ is based on the following principle: on the edge of the network traffic streams are classified and possibly conditioned (e.g., shaped to a certain maximum data rate), and assigned to different Behaviour Aggregate (BA) groups. Aggregate groups determine the type of Per-Hop-Behaviour (PHB) received by each packet as it traverses the DiffServ network. In the core of the network traffic streams belonging to the same BA group receive the same forwarding treatment (i.e., PHB). The packet handling/treatment is defined in a contract (Service Level Agreement - SLA) between a customer and a service provider.

Core routers in a DiffServ domain perform simpler tasks as compared to edge routers; they simply forward packets based on their classification. To achieve scalability, the packet handling complexity is confined at the edge of the network, where the traffic volumes are reasonably low.

DiffServ utilises the IPv4 Type-Of-Service (TOS) octet in a IP packet header as the DiffServ (DS) field. As shown in Figure A.1, six bits of the DS field represent the DiffServ Code Point (DSCP), allowing 64 classes of services to be defined. The remaining

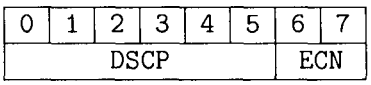

Figure A.1: Differentiated service field.

two bits are used for TCP Explicit Congestion Notification (ECN) and are ignored by DiffServ. Usually edge routers set or re-set (i.e. mark or re-mark) the DSCP whereas core routers merely examine it to determine the PHB to apply to packets.

A.1 Building Blocks

PHBs are implemented by combining a series of simple functional elements in datapaths to form larger Traffic Conditioning Blocks (TCBs). Datapath elements are the building blocks of the Informal Management Model for DiffServ routers developed by the IETF [76]. The model includes definitions for Classifiers, Meters, Actions Elements (for Marking, Absolute Dropping, Counting and Multiplexing), and Queueing Elements, including Schedulers and Algorithmic Droppers.

A.1.1 Traffic Classification Elements

Classifiers select packets based on predefined filtering criteria. They take a single input stream and split it into a number of logically separate output streams using filters to match a portion of the packet's content. BA classifiers split streams according to the DSCP whereas Multi-Field (MF) classifiers use one or more fields of the packet header (e.g. source and destination addresses), including the DSCP.

A.1.2 Meters

A Meter measures and splits traffic into a number of outgoing streams. The temporal properties of traffic streams are measured against predefined traffic profiles. Conforming streams are forwarded to an in-profile output and non-conforming streams to an out-of-profile output.

A.1.3 Action Elements

These elements perform a number of actions on packets. They are usually placed after classifiers and meters. *Markers* set the DSCP according to the result of the classification or metering process. Packets are normally marked by edge nodes although re-marking can be done in the core of the network. A *counter* simply counts the number of processed packets. The result is normally used for accounting purposes. *Absolute droppers* discard packets arriving its input. Discarded packets are usually non-conforming to a traffic profile.

A.1.4 Queueing Elements

Queueing elements are used to store packets, selectively discard them, and/or shape network traffic by delaying the departure of packets. A *queue* is a data structure that keeps packets that are awaiting transmission. A simple queueing system is made up of a queue and a scheduler. *Schedulers* use scheduling algorithms (service disciplines) to gate the departure of packets. They may also be used to shape non-conforming traffic to a predefined profile by delaying packets until they are conforming. Two typical queueing systems are priority and rate-based queueing. In a *priority-based queueing* system, a scheduler empties the queue according to the priorities of packets. *Rate-based queueing* systems use schedulers to empty queues, each at a specified rate. The depth of queues is usually managed using Active Queue Management (AQM) techniques that can employ *algorithmic droppers* to selectively drop packets when predefined queue thresholds are crossed.

A.2 Standard PHBs

PHBs define the treatment a BA receives at each DiffServ router. A number of PHB are standardised. This section describes the three most generally used PHBs.

A.2.1 Expedited Forwarding

The Expedited Forwarding (EF) PHB [77] provides low delay, low loss, and low jitter services. It is generally used for voice or any other traffic type that requires "wire-

like" behaviour. EF traffic not exceeding a predefined rate is given strict delay and jitter assurances. Due to the fact that EF has priority over other traffic, it has to be shaped and policed so that it does not starve the link. The recommended DSCP for EF packets is 101110.

A.2.2 Assured Forwarding

The Assured Forwarding (AF) PHB [78] has four traffic classes. Each traffic class is assured a certain amount of resources (e.g. bandwidth). Within AF classes packets can be assigned to one of three drop precedence levels. When a node is congested packets having higher drop precedence have a higher chance of being dropped. This PHB provides relative differentiation of services, and ensures that users may exceed the subscribed rate provided that excess traffic is not delivered with as high priority as conforming traffic. Table A.1 shows the recommended DSCP for AF classes and their discard priorities.

Table A.1: AF DSCPs and discard priorities.

Discard Priority	AF Class			
	Class 1	Class 2	Class 3	Class 4
Low	AF11 101010	AF21 101010	AF31 101010	AF41 101010
Medium	AF12 101010	AF22 101010	AF32 101010	AF42 101010
High	AF13 101010	AF23 101010	AF33 101010	AF43 101010

A.2.3 Default Forwarding

All traffic not belonging to any other class is place on this group. Default Forwarding (DF) traffic receives best-effort services with bandwidth guarantees, and when possible, controlled delays. The recommended DSCP for DF traffic is 000000.

Appendix B

QoS Policy Demonstration Examples

This appendix contains examples that illustrate how the XML-based QoS Policy model developed in this thesis should be used.

B.1 NLP Usage Demonstration Example

This section contains XML samples that illustrate the usage of a Network-Level Policy (NLP). In order to reduce the size of the code snippets most of the XML attributes are omitted.

The samples below depict a *PolicyRule* that controls QoS resources assigned to traffic generated from IP telephony applications. This example assumes that traffic is pre-marked with the corresponding DSCP at a location close to the user. At the edge node traffic is classified and policed against a pre-defined traffic profile. Exceeding traffic is discarded and conforming traffic is treated according to a pre-defined PHB.

The *PolicyRule* aggregates a *CompoundPolicyAction* that represents services pre-defined by an administrator. The *CompoundPolicyAction* aggregates further *PolicyActions* that are placed in a *ReusablePolicyRepository*. The example below only shows a *QoSPolicyPoliceAction* corresponding to a traffic policer. Because the action is re-usable a *PolicyActionInPolicyAction* aggregation is mapped to an XML element which contains an XInclude element that points to the re-usable *QoSPolicyPolice-*

Action. Notice XInclude first locates the document *QoSPolicyPoliceAction.xml* in a policy repository. XPointer uses an XPath expression to locate the exact re-usable policy fragment within the XML document.

```
<PolicyGroup Name="Gold_Service_Class"
PolicyRoles="edge">
  <PolicyRule Name="EFedge-policyRule">
    <...> ... </...><!-- PolicyConditions -->
    <CompoundPolicyAction Name="EFedge-actions">
      <PolicyActionInPolicyAction>
        <xi:include href="/db/QoSPolicies/ReusablePolicies/
        QoSPolicyPoliceAction.xml#xpointer(
        //QoSPolicyPoliceAction[
        @Name='EFedge-policer'])"/>
      </PolicyActionInPolicyAction>
      <...> ... </...><!-- Other PolicyActions -->
    </CompoundPolicyAction>
  </PolicyRule>
  <...> ... </...><!-- Other PolicyRules -->
</PolicyGroup>
```

The extract below corresponds to the re-usable *QoSPolicyPoliceAction* mentioned previously. The action consists of a token bucket traffic profile for traffic policing, and two re-usable actions - one *SimplePolicyAction* to mark DSCPs for conforming traffic and one *QoSPolicyDiscardAction* to discard excess packets.

```
<ReusablePolicyContainer>
  <...> ... </...><!-- Other re-usable Policy elements -->
  <QoSPolicyPoliceAction Name="EFedge-policer">
    <QoSPolicyTrfcProfInAdmissionAction>
      <xi:include href="/path_to_file.xml#xpointer(//
      QoSPolicyTokenBucketTrfcProf[@Name='EFedge-TB'])"/>
    </QoSPolicyTrfcProfInAdmissionAction>
    <PolicyConformAction>
      <xi:include href="/path_to_file.xml#xpointer(//
      SimplePolicyAction[@Name='EF-dscpMark'])"/>
    </PolicyConformAction>
    <PolicyExceedAction>
      <xi:include href="/path_to_file.xml#xpointer(
```

```
      //QoSPolicyDiscardAction[@Name='discardAction'])"/>
    </PolicyExceedAction>
  </QoSPolicyPoliceAction>
  <...> ... </...><!-- Other re-usable Policy elements -->
<ReusablePolicyContainer>
```

In the extract above a *QoSPolicyTrfcProfInAdmissionAction* association links the *QoSPolicyPoliceAction* with the token bucket profile depicted below. It contains one rate and one burst parameter.

```
<ReusablePolicyContainer>
  <QoSPolicyTokenBucketTrfcProf Name="EFedge-TB"
  qpTBRate="200" qpTBNormalBurst="300"/>
</ReusablePolicyContainer>
```

The re-usable *SimplePolicyAction* for DSCP marking in case the traffic is conforming is depicted below. The value "101110", corresponding to the Expedited Forwarding (EF) class [57], is assigned to a *PolicyDSCPVariable*.

```
<ReusablePolicyContainer>
  <SimplePolicyAction Name="EF-dscpMark">
    <PolicyDSCPVariable/>
    <PolicyBitStringValue BitString="101110"/>
  </SimplePolicyAction>
</ReusablePolicies>
```

Finally the re-usable *QoSPolicyDiscardAction* is shown below. This simply indicates that packets should be dropped.

```
<ReusablePolicyContainer>
  <QoSPolicyDiscardAction Name="discardAction"/>
</ReusablePolicies>
```

The examples above illustrate how XML easily enables the definition of policies in a modular way by using XInclude to locate and include re-usable policy data modules or fragments. This modular design becomes even more attractive considering that some native XML databases already come equipped with XInclude processors that include re-usable data fragments on the fly.

B.2 DLP Usage Demonstration Example

This section contains XML samples that illustrate the usage of a Device-Level Policy (DLP). Figure B.1 depicts a very simplified version of a TCB where traffic belonging to the Expedited Forwarding (EF) class, with DSCP 101110, is policed against a traffic profile and conforming traffic is forwarded in a non-work-conserving scheduler whereas non-conforming traffic is dropped. In the figure the blocks represent PRIs. The arrows depicted are linking PRIs using their PRIDs as reference. The XML fragment below corresponds to the mapping of the highlighted portion of Figure B.1.

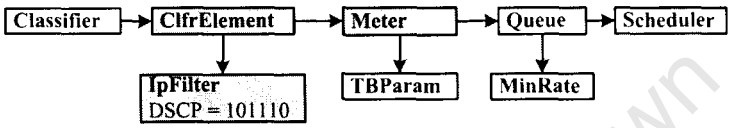

Figure B.1: Simplified example of a DiffServ TCB.

Classification is performed by *ClfrElementTable* with *Prid* 1, based on filter defined by *IpfilterTable* with *Prid* 3. *ClfrElementTable* has a child element *ClfrElementSpecific* that points to the IP filter, which is defined in the *frwk-pib* namespace. This namespace is defined in an XML Schema resulting from the mapping of the Framework PIB. After classification all EF traffic is directed to a meter defined by element *MeterTable* with *Prid* 1. Notice all PRIs are grouped in XML element containers.

```
<frwk-pib:IpFilterTableContainer>
  <frwk-pib:IpFilterTable BaseFilterPrid="3" IpFilterDscp="101110"/>
</frwk-pib:IpFilterTableContainer>
...
<ds-pib:ClfrElementTableContainer>
  <ds-pib:ClfrElementTable ClfrElementPrid="1">
    <ds-pib:ClfrElementSpecific
    Pointer="//frwk-pib:IpFilterTable[@BaseFilterPrid=3]"/>
    <ds-pib:ClfrElementNext
    Pointer="//ds-pib:MeterTable[@MeterPrid=1]"/>
  </ds-pib:ClfrElementTable>
</ds-pib:ClfrElementTableContainer>
<ds-pib:MeterTableContainer>
  <ds-pib:MeterTable MeterPrid="1">
  ...
```

```
    </ds-pib:MeterTable>
  </ds-pib:MeterTableContainer>
```

As this example illustrates, there are many advantages of using XML over binary SPPI for policy representation. XML-based PIB is clear, easy to use and facilitates debugging. The XPath-based naming scheme presented here is more clear and easy to understand than the Object Identifier (OID) naming scheme employed by SPPI-based PIBs. As depicted in the XML fragment above, the generated XML document is easy readable and self-describing. Furthermore, because XML parsers are widely available, developers spend less time worrying about the details of the parser itself and more time working on the application. This certainly reduces the development cycles of PBNM applications.

B.3 Device-specific Configuration Policy Usage Demonstration Example

This section contains an XML fragment that illustrates the usage of the XML-based *tcng* model developed in this thesis. The XML fragment configures the *egress* side of interface eth0 (the default interface on a Linux machine). In this example EF traffic is separated using a filter that matches variable *ip_dscp* (the DSCP field of the an IP packet header) to *value* 101110[1]. Traffic that does not match this filter is classified into group *be* (best-effort). The HTB qdisc is specified using element *htb* with a child element *class*. Inside the HTB qdisc two classes are defined, one with higher priority than the other[2]. Class attribute *id* is used to select the classes inside the HTB qdisc, so in this example the EF traffic is placed in the higher priority class which allows bandwidths up to 256 kbps.

```
<tcng> <dev> <egress set_tc_index="true">
 <csp>
  <classdef id="ef">
   <filter field="ip_dscp" value="0b101110" />
```

[1]RFC 4594 indicates that traffic belonging to the EF PHP group should be marked with DSCP 101110.

[2]In tc a lower integer corresponds to a higher priority.

```
  </classdef>
  <classdef id="be"/>
 </csp>
 <htb id="priority-sched" rate="3Mbps" ceil="3Mbps">
  <class rate="3Mbps" ceil="3Mbps">
   <class id="ef" prio="0" rate="256kbps" ceil="256kbps">
    <fifo/>
   </class>
   <class id="be" prio="1" rate="2.5Mbps" ceil="3Mbps">
    <fifo/>
   </class>
  </class>
 </htb>
</egress> </dev> </tcng>
```

Appendix C

Mapping Tables used by the Translation Algorithms

This appendix presents a brief description of the mapping tables used by the two translation algorithms. The tables are used when translating NLPs to DLPs, as well as when DLPs are translated to the *tcng*-specific configuration model in XML (referred to in Chapter 3 as the device-specific configuration model). To ease understanding the translation table is subdivided according to the nature of the attributes (e.g. attributes used for traffic classification purposes are presented in one table).

Table C.1 shows how the attributes used for traffic classification are mapped. All NLP attributes belong to XML element IPHeadersFilter, whereas DLP elements belong to XML element *IpFilterTable* (derived from the mapping of the Framework PIB [79]). The values in the last column (*tcng* in XML) are string values assigned to the attribute *value* of element *filter*. It was mentioned in Chapter 3 that *filter* elements are used for packet classification. Each element contains an attribute *field* that indicates the packet header field to be used and an attribute *value* that contains the value to test. Furthermore, in the case of TCP ports the four values in the third column are mapped to *tcng* variables *tcp_dport* or *tcp_sport*. In this case the XSLT transformation script generates *tcng* arithmetic operators $>$ and $<$ to specify the minimum ad maximum ports.

Table C.2 depicts the mapping of attributes used for traffic metering purposes. All NLP attributes in the table below belong to XML element *QoSPolicyTokenBucket-*

Table C.1: Translation table used for traffic classification purposes.

NLP	DLP	*tcng* in XML
HdrDestAddress	IpFilterDstAddr	ip_dst
HdrSrcAddress	IpFilterSrcAddr	ip_src
HdrDSCP	IpFilterDscp	ip_tos
HdrProtocolID	IpFilterProtocol	ip_proto
HdrDestPortStart	IpFilterDstL4PortMin	tcp_dport_min
HdrDestPortEnd	IpFilterDstL4PortMax	tcp_dport_max
HdrSrcPortStart	IpFilterSrcL4PortMin	tcp_sport_min
HdrSrcPortEnd	IpFilterSrcL4PortMax	tcp_sport_max

TrfcProf; DLP attributes belong to XML element *TBParamTable*. The XML *tcng*[1] attributes may belong to XML elements *SLB*, *srTCM*, or *trTCM*.

Table C.2: Translation table used for traffic metering purposes.

NLP	DLP	*tcng* in XML
TBRate	TBParamRate	cir, pir
TBNormalBurst	TBParamBurstSize	cbs, pbs
TBExcessBurstb	TBParamBurstSize	ebs

Table C.3 depicts the mapping of attributes used for traffic marking purposes. NLP attribute *BitString* belongs to XML element *PolicyBitStringValue*, DLP attribute *DscpMarkactDscp* belongs to XML element *DscpMarkActTable*, and *tcng* attribute *value* belongs to XML element *classdef*.

Table C.3: Translation table used for traffic marking purposes.

NLP	DLP	*tcng* in XML
BitString	DscpMarkActDscp	value

Table C.4 shows how attributes used for traffic queueing and scheduling are mapped. The NLP attributes in Table C.4 belong to XML elements *QoSPolicyCongestionControlAction* or *QoSPolicyBandwidthAction*. DLP attributes may belong to a number of XML elements: *AlgDropTable*, *RandomDropTable*, *MinRateTable*, or *MaxRateTable*. XML *tcng* attribute *limit* may belong to either elements (i.e. qdiscs) *gred* or *red*, or

[1]The term 'XML *tcng*' refers to the device-specific configuration model described in Chapter 3, which in the case of this thesis is specific to a DiffServ router implemented with *tcng*.

element (i.e. qdisc) *fifo*. That decision is made by the translation algorithm based on the value of DLP attribute *AlgDropType*. If the value of *AlgDropType* is 4[2] (corresponding to the *randomDrop* algorithm [47]) then *limit* will belong to XML element *gred* or *red*; otherwise it will belong to XML element *fifo* (corresponding to a *fifo* qdisc). The rest XML *tcng* attributes belong to XML elements *gred* or *red*.

Table C.4: Translation table used for traffic metering purposes.

NLP	**DLP**	***tcng* in XML**
QueueSize	AlgDropQThreshold	limit
DropMethod	AlgDropType	-
MinThresholdValue	RandomDropMinThreshBytes RandomDropMinThresPkts	min
MaxThresholdValue	RandomDropMaxThreshBytes RandomDropMaxThresPkts	max
ProbabilityDrop	RandomDropProbMax	probability
ForwardingPriority	MinRatePriority	prio
MinBandwidth	MinRateAbsolute	ceil
MaxBandwidth	MaxRateAbsolute	rate

XML *tcng prio* belongs to any element *class* under *htb* (i.e. the classes inside an hierarchical token bucket qdisc). Attributes *rate* and *ceil* belong to XML element *htb* and its child *class* [55].

Note that because units of various attributes may differ, the translation algorithms apply appropriate arithmetic translations when executing the mappings.

[2]For a description of all the values that the various variables may take refer to RFC3317, RFC3318, RFC3644. This is because, as mentioned before, the XML-based QoS Policy Model developed in this thesis is based on the DiffServ PIB, the Framework PIB, and QPIM, respectively.

Appendix D

Evaluation Software

This appendix details the software used to implement the XML-driven Policy-Based QoS Management system developed in this project. This includes existing open source software as well as software developed in this project. The source code is available on the accompanying CD-ROM.

D.1 Developed Software

D.1.1 Management Station

The MS application was programmed in PHP5. The source code is located in folder *pbnms_ms*. This folder should be saved in the document root of the Apache web server. The file *index.php* controls the operation of the MS. This file should be edited with the details of the native XML database (eXist) as well as the the details of the web service on the PDP.

D.1.2 eXist - Native XML Database

eXist [62] is a powerful Java open source native XML database that was used to store NLPs and the network topology as XML documents. The source code can be found in folder *pr*. Installation instructions can be found at *http://exist.sourceforge.net/*.

D.1.3 Policy Decision Point

The PDP is coded in Java. It consists of many different files and packages, located in folder *pdpv015*. The main package is *exp.pdp.uct*. The following files should be edited in order to for the PDP to work properly: *RepoConnector.java*, *ParserUtils.java*, and *PolicyEngine.java*. These files require the full path of a configuration file as well as the details of the Policy Repository.

There are two way of running the PDP: as a stand-alone application or as a web service. When running as a stand-alone application the PDP cannot be controlled or monitored by the MS. This mode of operation is appropriate for testing purposes. To run the stand-alone application the following steps should be taken (the first line compiles the application and the second line calls the main Java class):

```
# ant
# java exp.pdp.uct.PolicyEngine
```

To run as a web service, the PDP must be running behind an application server, preferably *Apache Tomcat 5*. Therefore the folder *pdpv015* must be placed in the *webapps* folder of *tomcat5*. Note that because Tomcat rewrites the classpath every time it starts, the classpath of the PDP must be added to Tomcat manually (e.g. if the reader is using Fedora Core 5/6 edit the file */usr/bin/tomcat5*). When running as a web service the PDP starts operating as the web container (i.e. Tomcat) starts, but its operations can be controlled via the MS.

D.1.4 Policy Enforcement Point

The PEP is coded in C. It is based on the Yenca [69] NETCONF agent. Its source code is located in folder *pep*. To run the PEP execute the following commands:

```
# ./configure
# make
# make install
# netconfd
```

The *netconfd* application controls the operation of the PEP. It can take two parameters

- *v* - run in verbose mode;
- *c* - run in compress/decompress NETCONF messages

The following configuration file must be edited in order for the PEP to run successfully: *repository/pepconfig-r.xml*. This file contains the roles and capabilities of the PEP. The PEP is simply a NETCONF agent. DiffServ support is provided by the *tcng* infrastructure. Therefore in order for the PEP to run successfully *tcng* must be correctly installed on the same machine. The source code and installation instructions for *tcng* are located in the folder *software/tcng*.

D.2 Auxiliary Utilities

D.2.1 Valgrind

Valgrind is a suite of tool for debugging and profiling Linux programs. In this project the author made extensive use of valgrind's *memcheck* tool to profile the PEP application. The following command may be used to run the tool and profile the *netconfd* program mentioned previously:

```
valgrind --tool=memcheck --leak-check=summary netconfd
```

D.2.2 Distributed Internet Traffic Generators

The Distribute Internet Traffic Generator (D-ITG) [71] was used to automatically generate network traffic when running the experiments presented in Chapter 5. D-ITG is capable of generating traffic at network, transport, and application layers. For this project D-ITG was used to generate EF, AFx, and DF traffic in the DiffServ context. D-ITG is accompanied by intuitive and comprehensive documentation.

D.2.3 decode_wireshark.c

This utility was programmed by the author due to the need of decoding Wireshark/Ethereal XML output. The decoded output can then be easily plotted using *gnuplot*. This utility is located in folder *software/utilities*.

Appendix E

Accompanying CDROM

The accompanying CD-ROM is located on the inside back cover of this document. The contents of the CD ROM are as follows:

- A soft copy of this thesis in PDF format.
- The LyX source files used in generating this document.
- The source code of the evaluation platform.
- Relevant publications used during the research of this thesis.