An Infrastructure for End Customer Metering of Networked Services

Marcelo Rita Pias

A dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
of the
University of London.

Department of Computer Science
University College London

15th January 2004
Abstract

The Internet today is driven by business relations between customers and service providers. An increasing number of services offered in this context require guaranteed quality, specified as part of a contract. The challenge is then to guarantee Quality of Service (QoS) in a best effort network, such as the Internet. Existing standards (IETF DiffServ/Intserv) provide means to offer this QoS. How these services ought to be charged is another important aspect, as it gives the provider incentives to recover its costs and make a stream of revenue.

It is argued that before any real deployment of QoS and charging mechanisms takes place, other developments have to occur. Services provided to end-customers need to be measured according to the terms stated in the contract. Such terms specify the acceptable performance level in the service level agreement (SLA), in addition to a tariff used to charge for the service. Thus, a metering system to report and store resource usage information is necessary to (a) dynamically assert whether a service delivered conforms to its SLA, and (b) charge for the service based on the tariff.

Metering in this business scenario, however, poses issues that need to be addressed. Security is an important one, as there will be incentives for customers or providers to cheat. Also, the metering system has to be instructed to measure whatever parameters the service contract relies on, therefore requiring its automatic configuration and management. Finding the limitations of a measurement system in terms of performance is another relevant aspect, so as to help the understanding of its operation in heterogeneous host environments.

The thesis of this work is that an infrastructure tailored for end-customer metering of networked services is necessary to address the above issues. The main contributions of this dissertation can be summarised as follows: (a) The EdgeMeter system is proposed to facilitate the metering of services in a general business context. It supports safe transport and storage of measurement data, as well as flexible configuration of metering points based on SLA/Tariffs automatically distributed by service providers; (b) A modular approach allows extensibility and flexibility of this infrastructure. Measurement modules can be stored and located in the network. A scalable network location framework, Lighthouse, is proposed and evaluated. It is shown that this framework computes more accurate network locations, when compared to other related techniques; (c) A security framework based on a trustworthy third-party component, the Meter Inspection Authority (MIA), is proposed to tackle security issues. MIA endorses the accuracy and safety of metering systems; (d) To assess the performance of metering systems, experimental work is carried out using a prototype implementation of EdgeMeter. It is found that the performance of such systems has strong influence on accuracy. Under certain loads, in which the normal behaviour of the system is degraded (poor performance), the metering service may be inaccurate.
Acknowledgements

I have always considered pursuing a PhD to be a great challenge and writing a PhD dissertation a dream. Without the help of several people, this dream would have never come true.

I would like to thank my supervisor, Steve Wilbur, for his enthusiastic advice and encouragement during my enjoyable time at UCL. Hopefully he will give me some advice in photography, which is another area he is an expert in!

I would like to thank Jon Crowcroft for his guidance and friendship while my second supervisor at UCL. Jon gave me the opportunity to begin this unique experience of pursuing a PhD in the first place. He always had a good word of advice and his knowledge in my research area has been very helpful.

I am grateful to Saleem Bhatti for taking over as my second supervisor. Saleem has been supportive and has given me invaluable guidance throughout the PhD process.

This work was financially supported by a grant from British Telecom Labs. I am grateful to Bob Briscoe for showing me the importance of network metering and charging as a research area, and for the very rewarding six months that I spent in the Distributed System Group at British Telecom Labs, Ipswich. Bob has always provided invaluable feedback on my research work. Jerome Tassel, Mike Rizzo and Kostas Damianakis also helped make my time there enjoyable and challenging.

I am indebted to Steve Wilbur, Jon Crowcroft, Saleem Bhatti, Dimitrios Miras, Kennedy Cheng, Gabriele Corliano and Simon Williams for reading and commenting on drafts of this dissertation. Also, Christophe Diot, Gianluca Iannaccone and Tim Griffin from Intel Research have given me invaluable input on the experimental work chapter. I am very thankful for that.

Thanks also to all my colleagues at UCL, in particular Adam, Socrates, Dimitrios, Tristan and all the colleagues from room 212 for their support and friendship.

Most especially my thanks are due to Karen for everything. She is the world’s greatest wife!
Preface

Except where otherwise stated in the text, this dissertation is the result of my own work. No part of my dissertation has already been, or is being currently submitted for a degree or diploma or any other qualification at any other university.

This dissertation does not exceed a hundred thousand words, including tables, footnotes and bibliography.
# Contents

1 Introduction ................................................................. 15
   1.1 Business Context .................................................. 15
   1.2 The Case for Metering ............................................. 17
   1.3 Issues with Metering ............................................... 19
   1.4 Contributions ....................................................... 20
   1.5 Outline of dissertation ........................................... 21

2 Internet Metering .......................................................... 23
   2.1 Overview ............................................................ 23
   2.2 Common Framework ............................................. 23
      2.2.1 Meter Location .............................................. 25
      2.2.2 Measurement Technique ............................... 26
      2.2.3 Granularity of the measurement data ............ 30
   2.3 Taxonomy of Architectures .................................... 32
      2.3.1 Passive Metering ........................................ 32
      2.3.2 Active Metering Architectures .................... 35
      2.3.3 Hybrid Architectures .................................. 36
   2.4 Summary .......................................................... 36

3 Issues with Internet Metering .......................................... 39
   3.1 The case for measuring the service quality ............. 40
      3.1.1 Research Developments ................................ 41
      3.1.2 IETF Architectures ....................................... 42
      3.1.3 Metering Requirements for QoS mechanisms ...... 44
   3.2 The case for charging for advanced services .......... 45
   3.3 Metering Issues .................................................. 51
      3.3.1 Metering Scenarios ...................................... 51
3.3.2 Flexibility ......................................................... 55
3.3.3 Issue: Control of the Metering System .................. 56
3.3.4 Issue: Security .................................................. 58
3.3.5 Issue: Performance ......................................... 60
3.4 Approaches to Metering ........................................... 60
3.5 Summary ............................................................... 64

4 EdgeMeter System .......................................................... 66
4.1 Introduction ........................................................... 66
4.2 Business Context ...................................................... 68
  4.2.1 Initialisation: Code Distribution .......................... 68
  4.2.2 Operation ....................................................... 69
4.3 Architectural Microscopic View .................................. 72
4.4 Configuration of EdgeMeter ..................................... 73
  4.4.1 Policy Centred Approach to Meter Management .... 73
  4.4.2 Service Related Control .................................... 75
  4.4.3 Policyset ......................................................... 76
  4.4.4 Measurement specification ................................. 76
  4.4.5 Policy Management .......................................... 80
4.5 Module Manager ....................................................... 83
4.6 Network Input ........................................................ 85
4.7 Translation Mechanism: Overview ............................ 87
4.8 Module Discoverer .................................................... 88
4.9 Summary ............................................................... 92

5 EdgeMeter Design Issues .................................................. 93
5.1 Module Manager ....................................................... 93
  5.1.1 Loading a module ........................................... 93
  5.1.2 Session Scheduling ......................................... 94
  5.1.3 Session Memory .............................................. 95
5.2 Module Discoverer (Network Location) ...................... 97
5.3 Lighthouses ............................................................. 99
  5.3.1 Finding Lighthouses ......................................... 100
  5.3.2 Local Basis Coordinates .................................. 101
  5.3.3 Host Coordinates ........................................... 102
Contents

7.4 Summary ................................................................. 156

8 Discussion ...................................................................... 158
8.1 Summary of the Experimental Findings .................. 158
8.2 Customer versus provider measurement .............. 159
8.3 Further experiments ............................................... 160
8.4 Security ................................................................. 161
8.5 System Architecture .................................................. 162
  8.5.1 Flexibility .......................................................... 162
  8.5.2 Implementation .................................................... 163
  8.5.3 Module Discoverer ............................................. 163
  8.5.4 Service Translator ............................................... 164
8.6 Summary ................................................................. 164

9 Conclusions and Future Work .................................. 166
9.1 Summary ................................................................. 166
9.2 Contributions ............................................................. 169
9.3 Current Limitations to the Edge Metering Approach . 170
  9.3.1 Distinguishing Application’s Data Streams ....... 170
  9.3.2 Security Difficulties ............................................. 171
  9.3.3 Monitoring of Global SLAs ............................... 172
  9.3.4 Bursty and Steady Data Streams ..................... 173
9.4 Future Work ................................................................. 173
  9.4.1 Exploring logical processors ......................... 173
  9.4.2 Metering accuracy of flow-based systems ......... 174
  9.4.3 Lighthouses for network proximity exploitation in Peer-to-Peer Systems . 174
  9.4.4 Multidimensional search to approach resource discovery .... 175

10 Glossary of terms ..................................................... 176

A SLA/Tariff Specification language (STS) ................. 183

B EdgePol Language ......................................................... 187
  B.1 Introduction .......................................................... 187
  B.2 Measurement Philosophy ....................................... 188
    B.2.1 Network Field .................................................. 188
Contents

B.2.2 Network Flux: Measure of Network Field ................................................. 190
B.3 Lexical Structures ........................................................................................... 191
  B.3.1 Identifier (< id >) .............................................................................. 191
  B.3.2 Keyword (< keyword >) .............................................................. 192
  B.3.3 Literal (< literal >) .......................................................................... 192
  B.3.4 Operator (< operator >) ................................................................. 192
  B.3.5 Separator (< separator >) ............................................................... 192
  B.3.6 Comment (< comment >) ................................................................. 193
B.4 General EdgePol Structure ............................................................................ 194
  B.4.1 Policyset ............................................................................................. 194
  B.4.2 Policyset Header .................................................................................. 195
  B.4.3 Policyset Body ..................................................................................... 195
  B.4.4 Policyset Footer .................................................................................. 197
  B.4.5 Network Field/Flux Specification ......................................................... 197
  B.4.6 Generalisation of Granularity ............................................................. 200
  B.4.7 Metric .................................................................................................. 201
  B.4.8 Meter Profile ....................................................................................... 203
B.5 Specific EdgePol Constructs ......................................................................... 204
  B.5.1 Expressions (< exp >) ........................................................................ 204
  B.5.2 Statements (< stm >) .......................................................................... 206
  B.5.3 Policy with the ECA (Event-condition-action) Model .......................... 209
  B.5.4 Declaration ........................................................................................... 212
## List of Figures

1.1 Business Model ................................................................. 16

2.1 Simplified model of a monitored communication ................. 24
2.2 Meters deployed inside the network .................................... 26
2.3 Meters deployed at the edges of the network ..................... 27
2.4 Passive measurement ...................................................... 27
2.5 Active measurement system ............................................ 28
2.6 Granularity of the measurement data ................................. 31

3.1 QoS metering ................................................................. 42
3.2 Integrated services over DiffServ networks ....................... 44
3.3 Metering inside the network ............................................ 52
3.4 Metering at the edges of the network ............................... 53
3.5 Dynamic control of metering ............................................ 57

4.1 Initialisation phase ......................................................... 69
4.2 Operation phase ............................................................ 70
4.3 Overview of the architecture .......................................... 72
4.4 Microscopic view of the architecture ............................... 75
4.5 Example of TCP network field and flux ............................ 77
4.6 Example of a network field template ............................... 78
4.7 Example of a network flux template ............................... 79
4.8 Example of an ECA rule ................................................ 80
4.9 Network reader ............................................................ 86
4.10 Overview of the translation mechanism ............................ 87
4.11 Module discovery ........................................................ 89

5.1 Network field and flux hash tables ................................. 96
5.2 Global x multiple local bases ......................................... 99
List of Figures

5.3 Lighthouse overview ......................................................... 100
5.4 2-D example ......................................................................... 101
5.5 Gram-Schmidt process (QR decomposition) .......................... 102
5.6 Accuracy of Lighthouse: calibration ...................................... 106
5.7 Accuracy of Lighthouse: extrapolation ................................. 107
5.8 Service translator .............................................................. 108
5.9 Example of service specification in STS ............................... 109
5.10 Service graph $G$ ............................................................. 111
5.11 $G_1$: Interdependency tariff-SLA graph ............................ 112
5.12 $G_2$: Interdependency network field graph ....................... 113
5.13 $G_3$: Interdependency event graph .................................... 114

6.1 Possible meter installation points ......................................... 119
6.2 Initialisation phase ............................................................ 125
6.3 Initialisation phase (protocol steps) .................................... 127
6.4 Operation phase .............................................................. 128
6.5 Operation phase (protocol steps) ........................................ 130

7.1 Business Context .............................................................. 133
7.2 DiffServ Testbed .............................................................. 137
7.3 Comparison of measured against generated packet rate (absolute error) ........ 141
7.4 Relative error curves ........................................................ 142
7.5 EdgeMeter Implementation: Overview ................................ 146
7.6 EdgeMeter testbed ........................................................... 146
7.7 Comparison of measured against generated packet rate .......... 147
7.8 Comparison of measured against generated for rates of 30k pkt/s ........ 148
7.9 BPF architecture (packet path) at receiver machine .......... 148
7.10 Average CPU consumption for various packet rates ............ 150
7.11 CPU cost for rates of 30k, 33k and 36k pkt/s ..................... 151
7.12 Receive Livelock problem ................................................. 153
7.13 CPU cost per packet copy ............................................... 154

B.1 Network field framework ................................................ 189
B.2 Example of TCP network field and flux ............................ 190
B.3 Example of identifiers ...................................................... 192
List of Figures

B.38 Syntax of the variable declaration ........................................... 213
B.39 Syntax of function declarations ............................................. 213
List of Tables

2.1 Definitions of some terms used in this thesis .......................................................... 25
2.2 Passive and active measurements comparison ....................................................... 29
2.3 Overview of the characteristics of metering systems ............................................. 38

4.1 Examples of metric modules ................................................................................... 83
4.2 Examples of modules for data aggregation and analysis ....................................... 84

5.1 Key Parameters ............................................................................................................105

6.1 Security Requirements ..............................................................................................120
6.2 Meter Inspection Certificate (MIC) format ...............................................................124
6.3 Notation .........................................................................................................................125
6.4 Security protocol: initialisation phase ......................................................................127
6.5 Security protocol: operation phase .............................................................................131

7.1 Mapping to DiffServ classes ......................................................................................136
7.2 Machines specifications .............................................................................................137
7.3 Packet rates generated ..............................................................................................138
7.4 Relative error statistics ..............................................................................................143
7.5 Breakdown of CPU costs ..........................................................................................154

B.1 Keywords ....................................................................................................................192
B.2 Examples of granularity and their specification attributes .....................................201
B.3 Applicable states for an event .................................................................................209
B.4 Source and type of events .........................................................................................210
B.5 Action context ............................................................................................................211
B.6 Action ........................................................................................................................211
Chapter 1

Introduction

1.1 Business Context

The Internet, as originally conceived, offered network services to a restricted yet privileged user community, mostly formed of researchers. In those days, applications with soft requirements in terms of delivery time were dominant.

However, the Internet has dramatically changed in the last decade. New applications have been developed. Commercial service providers have built massive network backbones, and end-users have had the opportunity to join in by subscribing to network access services. The original research network testbed has now evolved into a huge mesh of interconnected hosts. More importantly, these interconnections are driven today by business relations between users and service providers, which have a great impact on the engineering aspects of the Internet. Should the demand arise for profitable new services, providers will be keen to exploit opportunities within this business context.

This thesis explores some of the technical requirements of the market for network access services. To understand this market, a canonical business model is introduced in Figure 1.1. This model presents business roles and interfaces to describe the interactions of customers and service providers in a multiservice network environment. Corliano et al [51] analyses a variation of this model in the market for public mobile access.

Bulk network providers own a massive network backbone infrastructure (routers, optical fibres, radio channels) usually being geographically spread. Services offered include network-based services, such as backbone connectivity to network wholesalers.

Network wholesalers also own a network backbone, but on a minor scale compared to
bulk providers. They deal with network retailers, selling network-based services. In the UK market, the BT Wholesale division, for instance, is a provider that offers network connectivity to network retailers such as AOL and FreeServe.

At the bottom, network retailers sell network-based services directly to customers. The latter can be divided into end-users and corporations. Usual services include network access to the Internet, either via dedicated lines (ADSL/Cable) or dialling access. BT OpenWorld, AOL and NTL are examples of network retailers in the UK.

Application-based services such as video/audio streaming are offered by application-level providers. It is important to note that such providers interact with network providers in order to have network connectivity. The provider to be dealt with will depend on the scale of the application-level provider. The larger the provider, the larger the network provider must be to accommodate the connectivity requirements.

Finally, the role of the network management provider consists of offering enhanced services for network management purposes. The list of services offered include, but is not limited to, network monitoring and planning.

The dashed lines of the figure represent the business interface between providers and customers through a service contract. Such a legal document has clauses for the specification of how the service ought to be charged (tariff), in addition to service level objectives that describe acceptable quality of the service delivered - the service level agreement (SLA). Hence, the service offered is required to be constantly metered so as to maintain these contract clauses.
sections that follow introduce requirements for metering applicable to this business context.

1.2 The Case for Metering

The majority of services offered to customers in this business context will require guaranteed quality, specified as part of the agreed contract. The challenge is then to deploy mechanisms to guarantee Quality of Service (QoS) in a best effort network such as the Internet.

Research and standardisation developments have addressed the issue of deploying a large-scale QoS mechanism for the Internet. A solution to this issue came from two developments in the Internet Engineering Task Force (IETF), the Intserv [21] and more recently the DiffServ framework [18].

The Integrated Services (Intserv) architecture introduced a set of end-to-end services tailored to realtime applications. The architecture uses per-flow 1 traffic conditioning at every network element along the end-to-end path. Network elements are any component of the network that is capable of exercising QoS control over the data flowing through it. These include individual devices such as routers and end-host operating systems.

In contrast, in the DiffServ architecture traffic control is mainly done at the edges of the network. Thus, the core routers are left just with the task of forwarding packets and traffic conditioning on aggregated traffic. DiffServ eliminates the need for per-flow state and per-flow conditioning, the main issue of Intserv, by treating the traffic in classes of aggregated flows.

Merely introducing guaranteed services with either DiffServ or IntServ is not sufficient. How these services ought to be charged for is another important question. Cost recovery is crucial to the service provider business. Charging for the offered service gives the provider incentives to recover its costs and make a stream of revenue. Prominent realtime services with guaranteed quality require further investments mainly in network infrastructure. For instance, the non-supporting router’s operating system should be upgraded to assist a particular QoS mechanism.

Applying the same price for different services does not give the right incentives for the cost recovery [75]. Providers know the relative costs of offering various services at discrete qualities. Therefore, each of these services ought to be charged differently. From an economic

---

1Flow is a distinguishable stream of related packets that results from a single user activity and requires the same QoS. Flow can be specified as a 5-tuple (source IP address, destination IP address, source transport port, destination transport port and protocol ID). An example of a flow specification is (10.0.0.1, 10.0.0.2, 0, 80, UDP).
1.2. The Case for Metering

perspective, price differentiation can lead to a more efficient use of resources [50].

A number of pricing mechanisms have been extensively proposed in the literature. It has been suggested in [64] that flat fee pricing, a standard and constant fee for Internet access commonly practised today, is very inefficient because it requires light users to subsidise heavy users. This claim can be observed today in the provision of broadband services. Providers are beginning to suffer due to heavy traffic imposed by peer-to-peer file sharing, which allows customers to download video and music files, for example. Some providers in the UK have already considered alternative pricing schemes such as usage-based charging. Other potential alternatives include congestion charging [97] and priority pricing [50].

However, before any real deployment of these mechanisms takes place it is recognised that other developments have to occur [10, 70, 26]. While there are many charging mechanisms which potentially fit most of the economic incentives to the efficient utilisation of the network, some form of *underlying metering service* is essential to the deployment of an integrated QoS and charging system, so as to provide:

- Fine-grained usage metering, where a service requires either network or machine resources for its provision. Similarly, a service level agreement (SLA) specifies acceptable thresholds of quality for this service provision as part of the agreed contract. Therefore, a metering system to report and store resource usage information is necessary to dynamically assert whether a service delivered conforms with its SLA. For instance, an IP video service that has an SLA guaranteeing a bit rate of 64 Kbps needs continuous metering of the delivered bandwidth. If the service rate is below the threshold of 64 Kbps, the SLA has been violated.

- Accounting, which is the process of correlating resource usage data with the identity of the user. For a metered service, the data provided by a metering service has to be first associated with user information before it can be used for charging for the QoS provided, for instance. In usage-based charging applied to different quality of service levels, such as the British Telecom (BT) framework [24], accounting is necessary between providers and customers. In addition, congestion charging schemes heavily rely on network feedback information, which can be collected by a metering service, to inform whether or not congestion has arisen [81, 97].

Within this context, metering is vital to the deployment of SLA validation and charging
systems. This naturally prompts the question: what are the issues involved in designing a metering system tailored to this business context?

1.3 Issues with Metering

The nature of the business relations established among providers and customers is determined by the QoS and charging approaches chosen for a given scenario. For instance, a network provider who controls a DiffServ region may sell services to a provider outside this region, which in turn provides services to end customers [70]. Such relationships pose varied requirements for the placement and function of metering systems.

One possible business scenario requires metering to be carried out inside the network, usually between network providers' interfaces, so as to measure their backbone performance. Although this an important research topic that has been addressed in [129], it is outside the scope of this dissertation.

The scenario considered here is one where service providers, namely application-level providers and network retailers, directly interact with end customers (shown at bottom of Figure 1.1). Wherever meters are installed, they are certainly located at the edges of the network. Meters may be installed at the edge of the service provider's domain, which is an arrangement that is inherited from the telecommunication sector and seem to be the standard practise at present. On the other hand, customers may opt to place and run meters in their own equipment.

This flexibility poses issues that need to be addressed. The metering system needs to be instructed to measure whatever parameters the service contract relies on. Such a legal document can be described in a high-level specification language. Automatic configuration of the system seems to be a reasonable controlling approach, for instance, when the dynamic negotiation of SLA parameters takes place between customers and providers [69]. The question is how the metering system will be controlled based solely on this high-level specification of the contract. If the SLA is translated into low level metering system configuration code, how might this translation be performed?

Security is another important aspect in this business context. There will be incentives for customers and even providers to cheat when advanced charging schemes which take into account the correlation between price, resource usage and quality of service are introduced. Such fraud may lead to drastic results if achieved on a large scale. The first goal is to prevent fraud being committed by malicious customers, such as tampering with the meter system. The
second goal is to protect customers from any damage that a meter installed in their systems might cause. Hence, security issues must be cautiously analysed and addressed.

Furthermore, finding the limitations of a measurement system in terms of performance will help the understanding of its operation in heterogeneous host environments. Performance issues may degrade the normal behaviour of a metering system, resulting in errors in the outcome of a measurement session. Understanding metering accuracy and its effects in a business context is very relevant.

1.4 Contributions

Frameworks and tools have been previously developed to measure the performance of networks. These have included research and development activities from within the research community and standardisation bodies, such as the IETF.

Many of these earlier projects have focused on measurements for IP network backbones. Motivated by a lack of a general measurement infrastructure to assist the planning and engineering of a service provider's network backbone, these mechanisms aimed at the collection of bulk statistics of the passing network traffic. Few other projects have concentrated on gathering more detailed information (on a per packet basis) on the network traffic.

However, the new market for Internet access offered to end-customers has introduced requirements other than the ones that motivated these projects. Measurements need to be taken at the edges of the network for billing and checking of the QoS delivered to customers (bottom of Figure 1.1). The issues that arise from this business context had not yet been sufficiently studied:

- Metering at the edges of the network is highly exposed to security threats as customers will eventually be billed for the usage of a service. These issues have been neglected in the majority of these solutions.

- Due to its complexity and dependency on many variables, performance related issues of metering systems have been left on hold in most of these projects.

This thesis addresses these issues. The main contributions can be summarised as follows:

- An infrastructure, named EdgeMeter, is proposed to address the issues associated with end-customer metering in scenarios of charging and SLA maintenance. A translation
mechanism is introduced to tackle the problem of managing a metering system based on high-level code specification, such as electronic tariffs and SLAs.

- A security framework based on a trustworthy third-party component, the Meter Inspection Authority (MIA), is proposed. MIA endorses the accuracy and safety of metering systems. This framework also introduces a set of cryptographic protocols tailored for this business context.

- A modular approach allows extensibility and flexibility of this infrastructure. Measurement modules can be stored and located in the network. A scalable network location framework, Lighthouse, is proposed and evaluated. It is shown that this framework computes more accurate network locations, when compared to other related techniques.

- Findings of experimental work carried out with a prototype implementation of EdgeMeter shows that measurements undertaken at the edges of the network always under-estimate the true traffic above a certain offered load imposed to the system hosting the meter code. Such a point, called the Maximum Error-free Metering Rate (MEFMR), depends on environmental conditions of the system, such as the operating system, network interface card (NIC) model and its device driver implementation. The performance of metering systems has strong influence on accuracy.

1.5 Outline of dissertation

The organisation of the remainder of this dissertation is as follows.

Chapter 2 provides background material relevant to this work. The state of art in Internet metering is given by placing the architectures proposed in a common framework. Comparisons among the proposals are also presented to highlight the advantages and limitations in prior research.

Chapter 3 discusses why, and to what extent, metering of network services is necessary. First, it considers two support cases for metering by analysing a scenario where service providers interact directly with end customers. It is argued that without a tailored metering system, the realisation of charging and SLA validation systems is impractical.

Chapter 4 introduces the EdgeMeter system and its design principles which aim to facilitate the accounting of network services and the measurement of the QoS received by end customers. Chapter 5 details the design issues of the central components of EdgeMeter.
1.5. Outline of dissertation

A framework to address the security issues associated with the metering process is presented in Chapter 6. The requirements for security are compiled and the framework based on a trusted third-party model is introduced. This framework also introduces a set of protocols, tailored for different business contexts, that aim to fulfil the security requirements.

Chapter 7 provides an evaluation of metering performance and accuracy through the use of a prototype implementation of EdgeMeter. Chapter 8 draws broader conclusions from the experimental findings.

Finally, Chapter 9 summarises the main findings of the research and suggests areas for future research arising from this work.
Chapter 2

Internet Metering

2.1 Overview

Internet metering (or measurement) provides us with measures of activity and performance. It is applicable where attributes are required to be quantified for a particular application. In the research literature, there are proposals of measurement infrastructures for the IP network. These approaches deal with the network measurements most commonly used in support of network capacity planning. More recently, service level agreement (SLA) validation, mostly motivated by new user requirements and business opportunities, has gained special attention of the service providers. These needs, in association with billing, have provided an important new use for network measurements.

To conduct network measurements practically we require a flexible and scalable measurement infrastructure which has specific characteristics regarding meter location, and the granularity and type of the measurements. As a result, we present in this chapter the state of art in Internet metering by putting the architectures proposed in a common framework. This will help us in drawing comparisons among the proposals by highlighting the advantages and limitations of this research.

2.2 Common Framework

Table 2.1 gives the definitions of some terms used throughout this thesis. By referring to these definitions, Figure 2.1 shows a simplified model for monitored communication on the Internet that takes place between two hosts, one sending packets and the other receiving them. The network, represented as a box, forwards these packets hop-by-hop until they reach their final destination. The management components shown above the dashed line, the "metering sys-
2.2. Common Framework

In essence, the manager instructs the meter to conduct a set of measurements for a period of time. The results are gathered by the collector and possibly post-processed by the aggregator. It is important to note, however, that limited data aggregation is sometimes part of the meter function.

![Simplified model of a monitored communication](image)

**Figure 2.1: Simplified model of a monitored communication**

The management traffic, introduced in the network by the metering system, can interfere with the normal course of the operational traffic. The extent of the impact will depend basically on the type of the measurement technique in use: active measurement (intrusive) or passive measurement. Generally, the impact is a performance degradation as some of the network resources are used to process and forward management packets.

On the one hand, active measurement schemes probe the network with 'test' packets in order to collect measurements (e.g. end-to-end delay) between a pair of meters. As a result, the greater the volume of test packets sent, the greater the management traffic and interference caused. On the other hand, passive measurements capture the operational traffic for further analysis, for instance, counting the packets observed at a particular point of the network within a given time interval. It is expected, therefore, that in this case the impact on the operational traffic is minimal because passive measurement is non-intrusive, i.e. does not introduce additional packets. However, there is still a resource cost for carrying out passive measurements.

The framework developed in this chapter is composed of three parts. The location of meters is one of them (Figure 2.1). Meters can be placed at the edge of the network, i.e. close to senders and receivers, or inside the network, for instance, in cases when an operator’s network backbone is being monitored. The second part concerns the type of measurement: active and passive. Granularity, i.e. how detailed the resulting measurement data can be, is the third part.
developed within this framework.

<table>
<thead>
<tr>
<th>Operational Traffic</th>
<th>Traffic generated by a sender destined to a receiver element.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management Traffic</td>
<td>Traffic injected in the network by management components (e.g. the metering system).</td>
</tr>
<tr>
<td>Network Traffic</td>
<td>Traffic which is observed at a metering point as the sum of the operational and management traffic.</td>
</tr>
<tr>
<td>Metric</td>
<td>An attribute of an entity that can be expressed qualitatively (what to measure) along with the specification of one possible way of calculating it (how to measure). Throughput, peak rate and packet loss are examples.</td>
</tr>
<tr>
<td>Measure</td>
<td>The numerical value of some attributes of a metric.</td>
</tr>
<tr>
<td>Metering</td>
<td>A process by which a measure is assigned to a metric. Measurement and monitoring are other terms used interchangeably.</td>
</tr>
<tr>
<td>Metering System</td>
<td>A group of elements with the ability to gather and further refine the measures for a set of metrics. This system is formed by meters and extra modules such as managers, collectors and aggregators. They can be attached to the meter or reside externally.</td>
</tr>
<tr>
<td>Meter</td>
<td>Element that provides measures for a set of metrics by passive 'tapping' or active probing of the network. It is sometimes referred to in the literature as a probe or trace.</td>
</tr>
<tr>
<td>Manager</td>
<td>Administers the meter using a configuration script. This process assumes a set of metrics to be measured and how the results will be gathered by collectors.</td>
</tr>
<tr>
<td>Collector</td>
<td>Collects, periodically or on demand, the measurement data exported by one or more meters. The collector can poll the meter (pull mode); or it can register with the meter as a listener of the measurement data (push mode).</td>
</tr>
<tr>
<td>Aggregator</td>
<td>Module that groups measurement records together. It can further classify and refine the data.</td>
</tr>
<tr>
<td>Passive metering</td>
<td>The traffic is captured (e.g. tapping the network) at the metering points with minimal impact to the normal network operation. A large group of statistics (e.g. byte counters) can be calculated.</td>
</tr>
<tr>
<td>Active metering</td>
<td>This technique probes the network by injecting test packets (management traffic). Active metering relies on a pair of meters.</td>
</tr>
<tr>
<td>Measurement data</td>
<td>The resulting data generated by the metering system organised in data units called measurement records.</td>
</tr>
</tbody>
</table>

Table 2.1: Definitions of some terms used in this thesis

2.2.1 Meter Location

Driven by their own motivation and measurement objectives, service providers and customers have preferences for metering points. To understand the behaviour and trends of traffic, network providers design and deploy infrastructures by installing meters within their administrative domains. Information gathered out of these systems can help to plan the expansion of their network. For example, bottleneck links can be identified and upgraded to higher capacity.
In contrast, small scale service providers and customers want to monitor their connections for various reasons; accounting for network usage is one of them.

**Inside the Network**

Network service providers, in particular those who manage the core network, try to detect, diagnose and fix problems with their network backbone. To support fault management, meters are deployed between core routers (Figure 2.2). The resulting information can populate spatial representation of the traffic, for instance, a *traffic matrix* model relating to data volume per source-destination pair. This matrix characterises the 'offered load' on the network [82].

![Figure 2.2: Meters deployed inside the network](image)

However, measuring the activity and performance inside the network is tremendously difficult. Issues of storage and transport of data are among others the main challenges faced by the proposals in the literature. The volume of data collected from the meters in a network backbone can require large storage space [77] and processing power of a meter machine [133].

**Edge of the Network**

The edge of the network, in this context, refers to the boundary of an administrative domain. This includes the edge routers which transport network traffic from/to the domain. Meters in this scenario will be installed at the edges of a provider or customer domain as depicted in Figure 2.3.

When billing and QoS monitoring are part of the measurement process objectives, providers might decide to place meters in the customer premises. One of the motivations is that any testing traffic will experience the same performance as the customer's traffic.

**2.2.2 Measurement Technique**

A fundamental part of our framework is the broad view of techniques to conduct measurements. This section introduces passive and active measurements.
Passive

Passive measurement schemes monitor the performance and behaviour of the network by capturing a set of packets from the network. The operational traffic, captured at the metering points, continues its normal course without any disruption toward the receiver (Figure 2.4). Thus, the low volume of management traffic injected into the network is one of the reasons why passive metering is the customary technique used in many of today’s architectures.

Passive measurements have disadvantages though. Due to the packet capturing process the volume of data produced by the meter can be very high. Fraleigh et al [77] reports that the data collected for 24 hours of packet traces on the Sprint backbone (USA) required around 1.1 Terabytes of disk space. This raises the question of where to take passive measurements: inside the network or at the edges. Certainly, the core network placement introduces considerable challenges, such as how to manage resource consumption.

Furthermore, the meter taps the network irrespective of the sender and receiver in what is called 'promiscuous mode'. In this case, any traffic flow that traverses a particular monitored segment will be copied by the metering system. As a consequence, passive metering raises issues related to data privacy.

One counter-measure to this problem is the use of packet filtering inside the meter, which
selects a subset of packets out of the total traffic seen. By tuning a filter, we can avoid meters capturing inappropriate traffic. Furthermore, meters can be configured to process only a portion of the packet content. For instance, the access to the first $n$ bytes of a packet allows us to capture the headers of interest (e.g. IP, TCP/UDP) and discard the packet payload. Even so, users may still complain about the disclosure of header fields (e.g. IP addresses). Techniques for header anonymization [157] can be used in such cases.

**Active**

This class of technique injects, periodically or on-demand, test packets (management traffic) into the network (Figure 2.5). This is useful to obtain measures for metrics that need the interaction of two or more meters, for instance the one way delay [5]. The clock of each of these meter machines may be synchronised in order to generate accurate timing information.

To discuss clock synchronisation, three definitions are necessary [132]. Clock resolution is the smallest unit by which the clock’s time is updated. In addition, the offset is the difference between the time reported by the clock and the ’true’ time as defined by national standards. Therefore, accuracy is how close the absolute value of the offset is to zero.

In this respect, two meter clocks are synchronised when they both have a similar notion of the time within an accuracy tolerance. The Network Time Protocol (NTP) [112] is used to synchronise machine clocks to an external reference source. The machine to be synchronised exchanges timestamp information with NTP servers that are hierarchically organised. However, NTP synchronised hosts can only provide accuracy in the range of milliseconds.

In some scenarios, the accuracy of the NTP based timestamp is not sufficient. Hence, a hardware supported system may be a suitable choice. For instance, Global Positioning System (GPS) receivers can be used instead of NTP as the timing reference source with accuracy of
around 1 microsecond [130]. One drawback of GPS, however, is the installation cost of a receiver. It is not only the purchase cost that counts but also the need to install an antenna where it has a clear view of the sky [108].

The intrusiveness of active metering leads to other questions. In order to avoid network resource starvation, which can affect end-user applications' behaviour, careful analysis of its use and interactions has to be considered. A wiser approach would evaluate the overhead imposed by the volume of injected traffic to the available network bandwidth.

Nevertheless, active measurements have dramatically increased in recent years. One of the reasons for this trend may be that active measurements make the gathering of accurate end-to-end statistics feasible. To measure the one way delay, for instance, two end points need to interact. By sending packets containing a sequence number and an accurate timestamp, these end-points meters can calculate the propagation delay (i.e. minimum) experienced in one path direction (forward/backward). The Internet Engineering Task Force (IETF) have put forward some standard proposals that specify the measurement procedure of some end-to-end metrics [134, 5, 6].

<table>
<thead>
<tr>
<th>Processing</th>
<th>Passive</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reasonable use on average but it increases proportionally to the number of packets captured.</td>
<td>Low demand as processing is related to the test traffic used. The latter should be kept as low as possible.</td>
</tr>
<tr>
<td>Storage</td>
<td>It depends on the detail level of the measurement data. Considerable disk space is used by collectors/aggregators in packet level monitoring. Similarly, meters require internal memory in flow-based measurements.</td>
<td>Usually it needs more disk space than memory in order to store long-term statistics but much lower than typical passive techniques.</td>
</tr>
<tr>
<td>Network resources</td>
<td>Reporting of measurement data may involve high volume of traffic.</td>
<td>It depends on the volume of management traffic generated. The higher the frequency of test traffic generated, the higher the network resources required.</td>
</tr>
<tr>
<td>Security</td>
<td>Privacy and integrity of the captured data.</td>
<td>Similar general issues of a standard communication between two points in the Internet: privacy, integrity and authentication.</td>
</tr>
<tr>
<td>Meter Location</td>
<td>Meters inside the network introduce challenges that need to be carefully addressed.</td>
<td>They are installed at multiple points. Active techniques involve the instrumentation of a pair of meters.</td>
</tr>
</tbody>
</table>

Table 2.2: Passive and active measurements comparison
2.2. Common Framework

Comparison

Table 2.2 compares passive and active measurements techniques. Passive metering demands machine resources (e.g. CPU, memory and disk) that have an increase consumption with passive measurement of high-speed links at the full line rate (core network).

On the other hand, active systems may consume few machine resources when care has been taken such as controlling the volume of test traffic used to avoid stressing the machine and causing network overload.

2.2.3 Granularity of the measurement data

Measurement data, collected from multiple sources, can be correlated to study aspects of the network usage. Questions such as "what the maximum delay that a packet has experienced while in the network" require different measurement data. Should the first question be answered with some form of aggregated measurement statistics, the second one would certainly need a finer level of detail or a finer granularity.

The AT&T measurement infrastructure described in [31] gathers measurement data at various granularities from their commercial ISP network. For instance, packet headers were collected from a T3 link to investigate the traffic burstiness and other properties in short time scales. In addition, bulk statistics such as packet counters helped to calculate at a macroscopic level the 'traffic matrix' of their network. Thus, different measurement objectives/questions require measurement data at various granularities.

Estan et al [67] briefly compares metering systems able to generate measurement data at different granularities. It is argued that for very detailed measurement data (e.g. packet traces), the measurement records may be accurate but not concise. On the other hand, coarse-grained measurement data such as aggregated packet flow statistics [45] may lose information during the aggregation process. Therefore, they propose a new type of granularity that results from identifying high volume of traffic clusters that can generalise the other aggregates. Because this is ongoing research, little detail has been given on the actual algorithms that identify the traffic clusters. Sommer et al [156] discusses methods to overcome the information lost by the aggregation process in flow-based systems.

Given the relevance to a metering system, granularity is introduced as the third part of our framework. By increasing the granularity of the measurement reports, the overall accuracy is increased likewise. Therefore, finding the balance between these two deserves further attention.
2.2. Common Framework

Figure 2.6: Granularity of the measurement data

**Fine-Grained**

Figure 2.6.a introduces the fine granularity. The metering process receives IP packets (left side of the figure) of operational or management traffic as the input for the measurement. A packet contains the payload and headers (e.g., IP, TCP/UDP). The output measurement records will be composed of information extracted from the headers in addition to supplementary data such as a timestamp, ID and record size (right side of the figure). With this granularity, accurate information can be obtained. Therefore, fine-grained data such as packet-level metering can be obtained by correlating the data received from multiple meters. When some form of aggregation is deemed necessary, it can still be performed on the output data.

**Coarse-grained**

In coarse granularities, the metering process classifies and aggregates the IP packets based on groups of related traffic (Figure 2.6.b). The output is a measurement record with an ID, a field list of statistics and additional meta data. A description of some meta data attributes can be found in [4]. The statistics can be, for instance, counters for a router interface or information about a TCP connection. The list that follows exemplifies some statistics commonly used.

- **Flow statistics.** A set of traffic statistics for related packets observed in a time interval. Flow is defined as a set of packets from A to B, observed at a metering point, which traverse a path using forward arcs (directed graph) throughout the network. It is often identified using a label such as a flow ID created by a combination of flow keys; these keys can be obtained from the packet headers (e.g., source and destination addresses, IPv6
2.3. Taxonomy of Architectures

In this section, we develop a taxonomy of metering systems by following the common framework previously discussed.

2.3.1 Passive Metering

Packet Monitoring

Fraleigh et al [77] described the IPMON architecture for high performance passive monitoring. Meters were installed within the Sprint (USA) IP backbone. GPS receivers were used for clock synchronisation among the metering points. The IPMON experience shows how challenging a system measuring at full rates of high speed links such as OC-3 (155 Mbit/s), OC-12 (622 Mbit/s) and OC-48 (2.48 Gbit/s) can be. To cope with the daily data collected, which was around 1.1 TBytes of packet traces for 24 hours, storage infrastructure along with 16 machines for the data analysis were dedicated to the project.

With the same goals, the Caida CoralReef suite [98, 33] can collect timestamped packet headers from high-speed links. However, the meters are not synchronised with GPS, leading to a lack of measurement accuracy.

These two approaches tackle some of the issues related to passive metering inside the network. In addition, packet level monitoring at the network edges can be deployed with inexpensive standalone PCs that tap the network [163] [165] [71]. The majority of these tools, often freely available, typically use the Berkeley Packet Filter (BPF) architecture [106].
2.3. Taxonomy of Architectures

The BPF design assumes that only copies of packets should be collected. Therefore, it avoids any route disruption in the actual path followed by an arriving packet. It has two main components: the network tap and the packet filter. The network tap copies packets from network device drivers and delivers them to listening applications. In contrast, the packet filter decides whether or not a packet should be accepted for delivery to the listening application. Filters can be configured by small programs written in a simple pseudo-machine language. The advantages of filtering packets are flexibility and performance gains, as most applications of a packet capture system reject far more packets than they accept.

Prior to the current mature stage of these tools, a flexible API capable of configuring the BPF at a reasonable level of abstraction had to be designed. With this goal, the libpcap API [92] was developed and quickly became the standard programming interface for packet capture applications.

Additionally, extensions to the original BPF design [65] [14] showed performance improvements. BPF+ [14] eliminates filter redundancies with global and local optimisation. Because the majority of the Unix [92] and Windows [149] implementations of libpcap rely on the original BPF architecture, it will take some time before we see such improvements implemented widely.

Flow-based Approaches

In flow-based systems, the meter classifies a stream of packets into flows based on a five-tuple set: source/destination IP and port numbers and protocol number. Monk et al [32] proposes a passive system largely based on flow-based measurements. At present, the specification is still a draft document, with little consideration given to security and performance.

Vendors have also introduced flow-based systems to fulfill the service providers demands for a router based measuring system. Cisco NetFlow [44] is perhaps the best known of these products. Technically, NetFlow is a module of the Cisco router operating system (IOS). The cache manager and the export system of NetFlow groups metering data in a UDP datagram that is then sent out to external collector machines. The overhead imposed on the router is one of the disadvantages of this system. Processing power originally allocated to packet forwarding is now shared with metering processes. More recently, new releases of NetFlow support packet sampling and aggregation techniques within the router in response to this weakness [45].

The lack of interoperability between vendor products motivated the IETF to define stan-
2.3. Taxonomy of Architectures

standards for flow-level based systems. The Real Time Flow Measurement (RTFM) architecture [30] was perhaps the first working group to focus on this area. The group defined the Meter-MIB [29], which compiles a list of common flow attributes (e.g. packet counters) in a MIB structure. Further support for new flow attributes [84] were added to better reflect the work of other IETF groups in the area of QoS.

The RTFM architecture proposes three components: (a) meters, which collect packets from the network; (b) readers, that gather measurement data periodically (pull mode) from the meters; and (c) managers, which configure meters through a rule set written in a high level language such as the SRL [27]. The reader polls the meter periodically (pull mode).

The RTFM architecture is powerful since it allows complex flow measurements to be specified with rule sets. However, if flow measurement is to be integrated into management application code, rule set specification is no longer appropriate anymore. Instead, developers of management applications would benefit from a declarative specification, a form that states only what to measure without any concern with how to measure, as opposed to the procedural nature of RTFM rule sets.

Considering this requirement, Quittek et al [142] [141] proposed an extension to the RTFM architecture. They developed a set of high-level declarative interfaces that simplify the configuration process by hiding several details of the architecture. As a result, flow measurement applications developed at British Telecom Labs and NEC Europe could integrate a measurement system component such as a RTFM implementation [28] into their core code.

More recently, the IETF IP Flow Information Export (IPFX) working group [143] is defining a format and selecting a transport protocol for exporting flow data from meter devices [175] [34] [46]. Although the group envisages a universal system that can be deployed anywhere, it seems likely that the RTFM architecture will remain the primary choice for measurements at the edge of the network, whereas the IPFX system will be used for preference in the core network. Unlike RTFM, the IPFX meter pushes the measurement data to the collectors.

Bulk-based Architectures

The SNMP has been important for the network management community. Its simplicity made it a protocol implemented in the majority of routers and management devices in the Internet. The MIB, a tree-based database structure, stores statistics and related information accessible through the SNMP. Three operations are supported by the SNMP version 1: (a) get, which
reads the content of a MIB object; (b) set, which writes a value in a object and (c) get-next that allows the next object of the MIB tree to be read. SNMP version 2 introduced the get-bulk operation, which allows bulk reading of groups of objects.

There are both IETF standardised MIBs and ad-hoc ones developed by vendors. In the area of measurements, MIB-II [107] and RMON [159] are the most notable ones. MIB-II, largely supported by routers, enables an overall understanding of the network. It supports variables that convey state information of the router resources (e.g. CPU and memory load) along with the network local environment. This information is most useful to long-term measurements as the variables will be updated in large time-scales (e.g. minutes to hours).

On the other hand, RMON provides information in short time-scales. It also supports extra variable objects and finer statistics compared to the MIB-II. For instance, it enables the network element to collect a traffic matrix at layer 2 and apply filters to it. RMON's complexity makes its implementation expensive. This explains why it is not often deployed today.

The RMON2 MIB, work started by IETF in 1994, extended the RMON capabilities to allow traffic monitoring above layer 2. The network manager may monitor traffic based on network and transport layers protocols. Key information regarding applications such as e-mail, file transfer and WWW may be constantly gathered. However, RMON2 (like RMON) is expensive to implement in devices such as routers and access servers. Some implementations have used a subset of the full functionality leading to simpler systems.

### 2.3.2 Active Metering Architectures

The IETF has also made progress in the area of active measurements. The IP Performance Metrics (IPPM) working group defined a set of standards that derive measures for one way metrics such as packet delay [5] and loss [6] provided the meters' clocks are synchronised.

Large-scale active measurement infrastructures have been deployed [109, 96, 166, 105]. The Surveyor project [96] aims at collecting accurate end-to-end delay and loss measures. The approach taken was to install meters, e.g. customised PCs synchronised with GPS, at various places around the Internet (currently 42 locations, mostly of them university and government sites). These meters send timestamped messages to each other in order to calculate a set of statistics.

Sometimes, fine-tuning clock synchronisation is not a strong requirement. For instance, the one way delay can be estimated by taking half the round trip time (RTT). This approach is
followed in the Active Measurement Project (AMP) from the National Laboratory for Applied Network Research (NLANR) in the USA [109]. The project measures the RTT, path connectivity and link loss between 130 AMP monitors around the USA. However, the problem of inferring one way delay from the RTT measure is that routing paths are sometimes asymmetric. The test traffic may follow different outbound and return paths, therefore leading to inaccurate measures for one way metrics such as delay [170].

Additionally, single tools that measure metrics with active measurement techniques have been developed. Link capacity related measurement tools [122] [62] [93] are among the most notable ones.

2.3.3 Hybrid Architectures

Choosing one measurement technique, active or passive, as opposed to the other may simplify the system design under certain assumptions but at the expense of flexibility. Often measurements have to be taken with both techniques and a hybrid architecture must be used.

Before a QoS mechanism (such as the DiffServ [18]) is introduced on the Internet, a flexible measurement infrastructure should be developed. In [103] the authors describe a new approach for network monitoring based on a hybrid system. They propose a combination of active and passive measurements between edge routers that surround the core network.

The National Internet Measurement Infrastructure (NIMI) [1, 135] proposed a base system capable of deploying various measurement infrastructures. NIMI offers the flexibility of installing any type of measurement tool developed by different parties. The infrastructure is composed of diversity-administered hosts rather than an infrastructure controlled by a single entity. The architecture is modular, therefore allowing 'third-party' plug-ins to be used. However, one of the main NIMI issues is how to update the measurement modules on a host platform and how to control the resource consumption of individual measurements [133].

2.4 Summary

This chapter introduced the state of art in Internet metering by putting the architectures proposed in a common framework, which is based on three parts. The location of meters is one of them: meters can be placed at the edge of the network, i.e. close to senders and receivers; or inside the network, for instance, in cases where an operator's network backbone is being monitored. The second part of the framework concerns the measurement technique, which can be either
active or passive. Granularity, i.e. how detailed the measurement data can be, is the third part developed within this framework. Table 2.3 draws an outline of the metering systems surveyed in this chapter in respect to these principal aspects of the framework.

Passive metering dominates the techniques employed by the metering architectures. On the other hand, granularity types seem to be balanced, with a number of systems supporting both coarse and fine-grained metering.

However, the majority of these metering systems have been designed for scenarios where metering is crucial but is conducted inside the network. In principle, it rules out measurements being undertaken at the edges of the network, say for instance in the customer domain.
<table>
<thead>
<tr>
<th>Measurement System</th>
<th>Location</th>
<th>Technique</th>
<th>Granularity</th>
<th>Applicability Scenario</th>
<th>Additional Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sprint IPMON [77]</td>
<td>Network</td>
<td>Passive</td>
<td>Fine-grained</td>
<td>Provide detailed backbone traffic data for internal research projects.</td>
<td>Issues of data storage. Security issues were not reported.</td>
</tr>
<tr>
<td>Caida CoralReef [98]</td>
<td>Both</td>
<td>Passive</td>
<td>Both</td>
<td>Network planning tasks</td>
<td>Flexible since it allows components to be plugged into the system. It provides raw traffic data (fine-grained) and flow-based statistics (coarse-grained).</td>
</tr>
<tr>
<td>BPF based tools [163, 71]</td>
<td>Both</td>
<td>Passive</td>
<td>Both</td>
<td>Network analysis and QoS monitoring</td>
<td>Low-cost monitoring tools. It runs on virtually any operating system.</td>
</tr>
<tr>
<td>Caida Measurement Spec.</td>
<td>Network</td>
<td>Passive</td>
<td>Both</td>
<td>Backbone engineering and planning</td>
<td>It lists a set of requirements along with some discussion of the data model to be used (e.g. metrics and data attributes).</td>
</tr>
<tr>
<td>NetFlow [46, 44]</td>
<td>Network</td>
<td>Passive</td>
<td>Coarse-grained</td>
<td>Accounting for Billing and Network Planning</td>
<td>Overhead to the router as flows statistics are exported without filtering.</td>
</tr>
<tr>
<td>Extended RTFM [142]</td>
<td>Edge</td>
<td>Passive</td>
<td>Coarse-grained</td>
<td>QoS monitoring and Billing</td>
<td>Dynamic configuration of the architecture to integrate applications.</td>
</tr>
<tr>
<td>IPFX [143]</td>
<td>Both</td>
<td>Passive</td>
<td>Coarse-grained</td>
<td>Flow information to accounting/billing and network management systems</td>
<td>Addressing security issues is one of the objectives.</td>
</tr>
<tr>
<td>RMON, RMON2 [159]</td>
<td>Network</td>
<td>Passive</td>
<td>Both</td>
<td>Network Planning</td>
<td>Not well deployed due to its complexity and operational overhead.</td>
</tr>
<tr>
<td>AMP [109]</td>
<td>Network</td>
<td>Active</td>
<td>Fine-grained</td>
<td>Monitoring of the connection of the participant sites</td>
<td>Uses RTT to calculate packet loss and delay.</td>
</tr>
<tr>
<td>Surveyor [96]</td>
<td>Network</td>
<td>Active</td>
<td>Fine-grained</td>
<td>Performance of the network</td>
<td>Large scale system/dedicated meters.</td>
</tr>
<tr>
<td>NIMI [135]</td>
<td>Network</td>
<td>Both</td>
<td>Both</td>
<td>Various Scenarios</td>
<td>Modular system that allows diverse types of measurements. The architecture has security mechanisms.</td>
</tr>
</tbody>
</table>

Table 2.3: Overview of the characteristics of metering systems
Chapter 3

Issues with Internet Metering

This chapter discusses why and to what extent metering of network services is necessary. At first, it considers two supporting cases for metering by analysing a scenario where service providers interact directly with end customers. Also, a detailed discussion of related metering issues is presented.

Implementation of mechanisms to provide QoS in the Internet require metering of network resource usage. In the near future, electronic service level agreements (SLAs) will play an important role by specifying acceptable thresholds of quality for the provision of a network service. As such, an infrastructure to report and store resource usage information is needed for the dynamic maintenance of an SLA.

The analysis of the case above led to the conclusion that implementing a QoS mechanism has associated deployment and follow up maintenance costs that must be recovered. One advocated solution to cost recovery is to charge customers for the differentiated quality of service provided. The second supporting case is based on this conclusion; it presents requirements for a metering system in a charging context.

It is argued that these two supporting cases require a flexible measurement platform. This flexibility, however, poses issues that need to be dealt with:

- Any configuration of the metering system should be translated from a network service specification, i.e. the merging of an electronic tariff and service level agreement (SLA) structures. The issue then becomes how the translation ought to be performed.

- Security is an important aspect in metering. There will be incentives for customers and even providers to cheat when some advanced charging scheme, one that takes into account
3. The correlation between price, resource usage and quality of service, is introduced. Such fraud may lead to drastic outcomes if achieved on a large scale. Hence, security issues must be carefully analysed and addressed.

- Finding the limitations of a measurement system in terms of performance will help the understanding of its operation in heterogeneous host environments. Performance issues may degrade the normal behaviour of a metering system. It would be the intention of a long term project to investigate all the subtle performance issues. This thesis, however, is concerned with a subset of those. In particular, we are interested in characterising the performance of the packet capturing process.

The outlined issues will be discussed in greater detail in the second part of this chapter. The first part is devoted to a discussion of the two supporting cases for metering.

3.1 The case for measuring the service quality

The last decade has seen considerable research into mechanisms to provide quality of service (QoS) for the Internet drawing on the lack of service guarantees for realtime applications. Though not yet widespread, some of these realtime applications are already commercially available [123]. How the network copes with their requirements in terms of packet delay, jitter and loss is still an open question when it comes to the real deployment. In practice, none of the proposed QoS mechanisms are widely deployed in network elements at present.

Users of Internet Telephony applications, for instance Net2Phone [121], have experienced the effects of network congestion. In a best effort network, such as the Internet, there is no guarantee that voice sessions can be accommodated without any perceived distortion such as 'a discomforting voice break' due to significant packet loss and severe packet jitter.

The common understanding is that generally the core of the network has sufficient capacity to cope with realtime applications (over-provisioned). The problem lies within the end-user system and its access provider perimeter, the local loop/last mile, where bandwidth is scarce leading to bottlenecks. Sometimes the performance of the terminal system, including the end-user machine and its operating system, accentuates the problem. Improvements in bandwidth have been predicted with the recent large deployment of broadband connections over copper telephone wire [74].

To support the quality guarantees essential to realtime multimedia applications, indepen-
dent QoS architectures were put forward. Such architectures arose from different groups of individuals:

- The research community have proposed several key components to build end-to-end QoS architectures.

- The Standards bodies, perhaps the IETF is the most notable one, have taken a step further by addressing the issues of implementing and deploying a large-scale QoS mechanism.

We are concerned in this thesis with the extent to which metering of resource, in particular network resources demanded by network services, is important to the realisation of such QoS architectures. In the sections that follow, relevant proposals are examined to identify what is required in terms of resource metering.

3.1.1 Research Developments

In [12] the authors elaborated a framework to profile the architectures that support QoS for realtime multimedia applications. Some key characteristics of the surveyed architectures are:

- The management mechanism of a particular architecture relies on the QoS requirements supplied by the users and consequently on the resource availability.

- If the architecture is to match the requested QoS levels, sub-components for traffic conditioning (packet scheduling, policing, shaping and metering) should be available in the architecture.

- The architecture continuously meters the QoS levels for the feedback loop to the maintenance of service level agreements (SLA).

Lazar et al [101] described the XRM (Extended Integrated Reference Model), a QoS architecture validated on an ATM network. XRM defines CORBA interfaces of which the interface for its management plane, termed the N-plane, and the interface for the resource control, e.g. VirtualSensor interface, are the most relevant. Little detail was given on how the QoS metering can be achieved in an IP network but the authors stressed its vital importance to the architecture function. In contrast, the OMEGA architecture [116] made the assumption of fully available metering functions implemented in the network subsystems.

To ensure that the QoS delivered matches that requested, a metering layer is used in another architecture proposed by Campbell et al [36]. The QoS-A architecture characterises a set of
management planes, formed by metering and policing functions, that act pro-actively by taking direct actions or informing the upper layers of the management hierarchy that a problem such as QoS degradation has arisen.

These architectures are similar in how they approach the issue of metering resources, generalised here by Figure 3.1. Input traffic arrives at the ingress interface and traffic conditioning functions based on an SLA are applied to it. In this case, the meter component provides metering services to the traffic conditioners and SLA maintainer.

To support traffic conditioning, it is necessary to have short timescale metering. A traffic shaper, for instance, regulates the inter-packet timing so as to avoid traffic burstiness, whereas the traffic scheduling decides whether or not to forward on these packets based on shaping policies. These decisions very often rely on instantaneous feedback of the metering component.

On the other hand, the SLA maintenance component samples the measurements as it operates on larger timescales. The SLA maintainer compares the QoS levels metered against those requested. Degradation of the QoS might result, so acceptable thresholds are used to decide whether or not the application should be signalled if poor QoS is experienced.

### 3.1.2 IETF Architectures

Providing QoS in the Internet today would probably still be confined to the research community if a standard body had not taken the initiative to develop a standard for interoperability. Two proposals have come from the IETF. This section outlines the DiffServ and Intserv approaches and conclude with a hybrid version of them.

**Intserv:** The Integrated Services (Intserv) [21] architecture introduced a set of end-to-end services tailored to realtime applications. The architecture uses per-flow traffic conditioning at every network element along the end-to-end path. Network elements, components of the network capable of exercising QoS control over the data flowing through them [155], include individual devices such as routers and end-host operating systems.
3.1. The case for measuring the service quality

Intserv assumes that some explicit signalling mechanism requests resource reservation within a network element, so that the end-to-end services can be accommodated. The RSVP [22] protocol has been adopted for this purpose. Packets that belong to the same flow receive the same QoS treatment along the end-to-end path.

Currently, guaranteed service (GS) [154] and controlled load (CL) service [172] are defined. The former offers firm bounds on end-to-end delays. Therefore, it is applicable to real-time applications with strict bandwidth and delay requirements. The latter provides applications with a QoS similar to what they would receive from a best effort network in unloaded conditions.

However, the complexity of the Intserv architecture raises scalability issues. Per-flow state and per-flow processing impose an overhead on the network elements along the end-to-end path. In addition, the lack of widespread support for RSVP has motivated the development of another mechanism that tackles scalability from its inception.

**DiffServ:** Traffic control is mainly done at the edges of the network; thus, the core routers are left with the task of forwarding packets [18]. Diffserv eliminates the need for per-flow state and per-flow conditioning (the main issue of Intserv) by treating the traffic in aggregates of flows.

Packets are aggregated in a small number of classes. Each class is identified by a number known as the DiffServ Code Point (DSCP) and it is associated with a per-hop behaviour (PHB) specifying how packets of this class should be treated in each network element. The DSCP value is inserted into the Type of Service (TOS) field of the IPv4 header. Two PHBs are currently specified. The Expedited Forward (EF) PHB gives low loss, low delay and low jitter to applications by ensuring that the service rate of EF packets on a given output interface exceeds their arrival rate at that interface over long and short time intervals [68].

In contrast, the Assured Forwarding (AF) PHB supports the delivery of packets in four forwarded classes, also identified by independent DSCP codepoints. Each of the AF classes allocates a share of buffer space and bandwidth. The service mechanism is simple. Packets are grouped in the AF classes and subsequently classified by the drop precedence value, therefore determining the relative importance of a packet. Packets in classes with lower drop precedence are prioritised if network congestion ever arises [87].

**Hybrid framework:** Intserv allows end hosts to request resources along end-to-end paths
but suffers from scalability issues. In contrast, DiffServ is characterised by its scalability for large scale networks. Motivated by these observations, Bernet et al [70] envisaged a hybrid framework that puts together the benefits of DiffServ in the context of the Intserv in pursuit of an end-to-end QoS.

Intserv capable nodes and Diffserv regions, where aggregate traffic control is applied, are encompassed in the hybrid framework (Figure 3.2). One of the assumptions is that the routers within the Diffserv region may or may not support the resource reservation requested through the RSVP protocol. From the Intserv viewpoint, the Diffserv region is treated as a virtual link network element.

The RSVP protocol is still used by the end hosts connected to the edge routers (ER) in order to reserve the resources according to their applications QoS requirements. Also, the edge routers interface with the Diffserv border routers (BR). In scenarios where the Diffserv region includes no RSVP aware devices, there will be a mapping function placed in the ER-BR interface. Such a component will translate the QoS requirements of an Intserv service into a Diffserv one which may be a PHB or perhaps a per-domain behaviour (PDB) specification [126].

3.1.3 Metering Requirements for QoS mechanisms

The hybrid approach is likely to be the choice for end-to-end QoS in the Internet. All these QoS architectures have undergone deep scrutiny regarding scalability over the past years which has resulted in commitments of the Industry, in particular router vendors, to the IETF standard proposals. However, before any real deployment of these mechanisms takes place it is recognised that other developments have to occur [10, 70]:

Figure 3.2: Integrated services over DiffServ networks
• Fine-grained resource usage metering. A service requires either network or machine resources for its provision. Also, a service level agreement should specify acceptable thresholds of quality for this service provision. Therefore, a mechanism to report and store resource usage information is necessary to dynamically assert whether a service delivered conforms in full with its SLA. For instance, an IP video service that has an SLA guaranteeing a bit rate of 128 Kbps needs continuous metering of the delivered bandwidth to ensure that the service rate does not drop below the threshold of 128 Kbps.

• Accounting is the process of correlating resource usage data with the identity of the user. In fact, it is an additional part of metering, and sometimes the terms 'metering' and 'accounting' are used interchangeably. The data exported by the meters, marked as M in Figure 3.2, has to be first associated with user information before, for instance, charging for the QoS provided. These meters are likely to be embedded in a router device or in standalone machines.

We will explore the issues that arise from the resource usage metering and accounting in Section 3.3. In the following section, we discuss the measurement requirements of charging mechanisms related to the business model presented in Chapter 1.

3.2 The case for charging for advanced services

In today's best effort Internet, cost recovery is crucial to the service provider business. Charging for the offered service gives incentives to the provider to recover its costs and open a stream of revenue. However, the prominent realtime services with the quality guaranteed requires further investments mainly in network infrastructure. For instance, the router's operating system should be upgraded to support a particular QoS mechanism. The question is how these services ought to be charged.

Applying the same price for different services does not give the right incentives for cost recovery [75]. Providers know the relative costs of providing various services at discrete qualities, therefore, each of these services ought to be charged differently. From an economic perspective the price differentiation can lead to a more efficient use of resources [50].

This section briefly introduces some of the pricing mechanisms proposed in the literature. The intention is not review them in detail but capture the type of measurement support that is needed before one can realise any of these approaches.
Flat fee: this is still the predominant pricing scheme in use today. Users periodically pay a standard and constant fee for the access to the Internet. Flat fee does not require any complex charging infrastructure, nor any network measurement despite the latter being convenient for network management purposes.

Predictability is one of the benefits of the flat fee scheme. On the one hand, users can estimate what cost is due in advance. On the other hand, providers can foresee their monthly income which provides low financial risk and planned expansion of the network capacity.

However, the flat fee scheme does raise a subtle issue. In unmetered access policy, i.e. typically the flat fee, users who value the access service highly are affected by other users who value the service less. For the latter users, the more free bandwidth they have available, the more they demand from the system. Therefore, the whole system’s performance degrades severely—a phenomenon known as the tragedy of the commons [85].

This phenomenon is accentuated by the introduction of broadband services that offer “always on” connectivity. Users now stay connected to the Internet while running bandwidth devouring applications. For instance, peer-to-peer file sharing has increased significantly with broadband services. Consequently, service providers are beginning to suffer due to the heavy traffic imposed by this type of application in their networks.

Some providers have even considered alternative pricing policies to counter what seems to be one of the Internet’s 'killer applications'. In this sense, the flat fee is likely to evolve into a scheme that turns into usage based charging should the user exceed a pre-defined usage limit. In the UK, companies like NTL have taken the lead by introducing 1 GBytes volume limit per day on the user traffic received. Any excess is charged on a usage basis.

What is happening now was to some extent observed in the Internet Demand Experiment (INDEX) [64]. The INDEX was a market and technology experiment to determine how much users value different qualities of service for Internet access. The results suggested that flat fee pricing is very inefficient because it requires light users to subsidise heavy users. This observation was described as a real threat to the provision of broadband services if charged on a flat fee basis as more bandwidth is made available to heavy users. The authors argued that an alternative is for the service provider to offer differentiated service quality with usage based charges that reflect the real cost of resources.

Congestion charging: It is recognised that network congestion is an important issue to
address. When a router queue builds up, packets are regularly delayed or even dropped in severe situations. In fact, there are many ways of dealing with congestion in packet networks. The QoS mechanisms described in Section 3.1 are some of them. More recently, there have been congestion pricing proposals suggesting that the end host application’s behaviour should be controlled by pricing driven rate controllers.

Gibbens and Kelly [81] describe a pricing model where the network can cope with N packets per time slot. Whenever the packet arrival rate in a slot exceeds N, packets are marked to indicate incipient congestion. This scheme is very flexible. A utility function characterises the value a user attaches to a given service. Depending on the user’s willingness-to-pay (WTP), the rate controller algorithm at the end hosts can accommodate different user’s utility functions. If the network congestion gets worse, i.e. the number of marked packets increases, action can be taken.

The action depends on the policy implemented in the rate controller. The user’s application, for example, can back off or perhaps continue its transmission by paying more. In this particular case, one practical issue is how the algorithm finds out the number of congestion marks from the network. To bridge this engineering gap, the Explicit Congestion Notification (ECN) [145] has been used to indicate congestion, which can be evaluated by the number of ECN marked packets received [55] [97]. ECN relies on active queue management installed in routers, e.g. random early detection (RED), to make a decision on whether or not to set a congestion bits in the TCP and IP headers. For congestion charging, however, only the network layer part of ECN seems relevant as the congestion level can be inferred by solely analysing the ECN bits of the IP header.

When the average queue size $x$ of an ECN-capable router is within minimum and maximum thresholds, packets are individually marked according to a probability function $p_a(x)$. An implementation of this function might be that packets arriving at a router when $x$ is close to the maximum threshold are more likely to be marked. However, above the maximum threshold packets will be dropped in any case. On the other hand, in levels of $x$ below the minimum threshold packets will not be marked.

The ECN marking is the network feedback mechanism chosen in the allocated capacity framework by Clark et al [48]. In this pricing scheme, the user’s traffic is tagged as in or out of profile. If the user’s traffic does not comply with the negotiated profile, it is tagged as out of profile or outside the allocated capacity. Out of profile tagged packets have higher drop
3.2. The case for charging for advanced services

probability in the event of congestion.

For instance, a traffic meter installed at the receiver host can check whether a stream of received packets is in profile. If so, the meter will turn off the ECN bits in those packets which have been marked by congested routers. When these packets arrive at the TCP receiver with ECN bits on, it means the traffic is out of profile and the sender should be notified.

**Auctions:** MacKie-Mason and Varian [104] present a model, the 'smart market', that calculates a second price auction for packets arriving at congested routers. Users pay the bid of the highest priority packet that is not admitted in the network as opposed to the price they have bid. As a result, this model is predictable in the sense that users know the maximum price they will end up paying.

However, the cost of implementing the smart market is high. For instance, in order to support the semantics of a bid-value in the IP header, the end host and the router software must be upgraded. Also, it is not clear how the auction takes place. Should it be centralised in one decision point then there is the issue of scalability. Should it be decentralised and simultaneous at each router, then the cost of a packet's transmission may vary at different parts in the network, which can make even worse the problem this model is trying to solve in the first place.

**Priority and Quota pricing:** Cocchi et al [50] introduce a priority-based pricing scheme where users pay less for a lower priority service. Therefore, the service and the price are differentiated.

There are four separate classes of service based on two bits in the IP header: the priority service flag and the priority no-drop flag. Each router along the path gives preference to those packets with the two priority flags on. Hence, in the case of severe congestion, the packets with both priority flags off will have the lowest priority.

Bohn et al [20] discuss another priority-based pricing model. Traffic is also divided in classes such as realtime, non realtime and some variation of these two.

The precedence bits in the TOS field of the IP header conveys the precedence level set by the user’s application according to the type of traffic required. Intelligent scheduling algorithms running in the routers select and forward packets based on the precedence bits.

The provider then imposes quotas on the total volume of customer traffic communicated based on IP precedence levels. Should the customers exceed their quota, they will be penalised. The penalty may be an increase of the customer's charge or even the termination of the service.
3.2. The case for charging for advanced services

**Edge pricing:** Shenker et al [153] point out that computing real congestion requires global knowledge of the network condition along an entire end-to-end path. To achieve this, a complex and high cost metering system has to be globally available. Instead, the authors propose the 'edge pricing' model. Edge pricing advocates that charging has to rely solely on the expected congestion and expected path followed by the user's packets.

Charging for an expected congestion event rather than the actual occurrence of this event requires metering at the edges of the network only. That makes the price calculation possible at the user's access points, and the entire system scales.

On the other hand, though the source of congestion cannot be precisely determined, there are means today to infer that a path between points A and B is congested. By measuring the number of packets that had some bits of the IP header (ECN bits) set by congested routers, the destination point B can infer from this information whether or not congestion has occurred in the network [25].

Briscoe [24] discusses split-edge-pricing, an extension of edge pricing, for multi-service networks. Each provider offers each of their service classes to their neighbours at a separate price for each direction of transmission. Since neighbour domains can locally determine how to apportion the value of information that flows between them, prices only depend on direct neighbours' interaction. The model makes the assumption of usage-based charging for different quality of service levels. In this case, measurements will be required not globally but locally between the providers and customers interfaces. This is one of the aspects that makes split-edge pricing simple and manageable.

**Metering Requirements**

The assumption that measurements will be somehow available for applications to calculate the user's charge [50] or enforce a traffic policy [48] is sound as long as there is a support infrastructure underneath. The majority of the charging approaches discussed in the previous section assume that there is a metering service or a similar building block capable of providing the set of measurements necessary to their deployment.

The type of measurements, meter positioning and granularity will vary depending on the charging policy:

- *Flat rate* pricing does not require complex usage statistics. However, it is clear that for network management a metering infrastructure, perhaps in its simple form, should be
3.2. The case for charging for advanced services

installed to measure the user's traffic. Information such as the usage trends of broadband users and what the preferred applications are can be consolidated for capacity planning and load balancing [129].

The majority of the service providers today use an authentication, authorisation and accounting (AAA) protocol (e.g. Radius) [148, 35]. The traffic is usually metered at the access server which sends data such as the volume transmitted per-user session back to centralised AAA servers. This may explain why providers insist on authenticating users for ADSL connections, even though this type of connection is supposed to be 'dedicated' and should not require any explicit authentication.

- The congestion charging schemes heavily rely on the network feedback information to inform whether or not congestion has arisen. A promising feedback mechanism is to measure the number of ECN marked packets by means of a token bucket (R,B) where R is the conformance rate and B is the bucket size. When the arrival rate of ECN marks exceeds R, meaning there is an indication of congestion, the mechanism reports this to the rate controller of the application. The controller decides the follow-up action, which may be an adjustment in the service rate.

- As far as the priority and quota pricing models are concerned, user's traffic is metered for each class of service utilised. In the quota scheme, the measurements validate the remaining quota the user still has available.

- Finally, in usage-based charging applied to different quality of service levels, e.g. split-edge pricing, bulk measurements will be sufficient in inter-provider interfaces; whereas at the edges of the network fine-grained measurements (per-flow, for instance) may still be required depending on the actual contract [24]. It is important to note, however, that coarse-grained measurements carried in the provider interfaces scale better since the overhead imposed on the meter system is at least manageable.

Thus, we conclude that implementing a QoS mechanism has associated deployment and follow-up maintenance costs to recover. An advocated solution to the cost recovery is to charge customers for the quality of service provided. While there are a plethora of charging mechanisms which fit most of the economic incentives to the efficient utilisation of the network, some form of underlying metering layer is essential to the deployment of an integrated QoS and charging architecture.
3.3 Metering Issues

Lack of scalability for Internet wide usage has been the primary criticism in some of the QoS and charging schemes proposed so far. Assessing the scalability issues is not a straightforward task. For instance, a crude analysis of the Intserv architecture shows that it does not scale well within the core network because the cost of maintaining the soft state information per network element of all the flow reservations is prohibitive. These concerns have driven the design of most of the architectures discussed in Section 3.1 and 3.2. The design principles for scalability influence the architecture of a metering system as follows:

- Within the network it is preferable to adopt *coarse-grained measurements*. Otherwise the overhead in the metering system is high [77] [26].

- Less traffic passes through the edges than the core of the network. The implication of this is that measurement becomes manageable at *finer levels* of detail, e.g. on a per-packet basis, at the edges of the network.

The following section introduces two metering scenarios, one inside the network and one at the edge, that are applicable to some of the charging and QoS mechanisms previously discussed.

3.3.1 Metering Scenarios

From an economic standpoint, the business model presented in Chapter 1 considers business roles in a multi-service network. The nature of the relations established among these roles determines the QoS and charging approaches to be used for a given instance of the business model. For instance, a network provider who controls a Diffserv region may sell services to a provider outside the Diffserv region, which in turn provides services to end customers [70]. Such relationships pose different requirements for meter placement and type of granularity.

**Metering inside the network**

Figure 3.3 shows the first metering scenario. In this case, providers trade network based services at a certain QoS between themselves. Metering will be carried out inside the network, usually between the provider interfaces. The question of whether metering is provider A's (Figure 3.3a) or B's (Figure 3.3b) responsibility will be a decision made in their contract.

Nevertheless, inter-provider metering is not an easy task, as discussed in Section 2.2.1. There will be issues on how to control the resources consumed by the measurement process in
3.3. Metering Issues

The point stressed earlier, very relevant to passive metering, that fine-grained metering inside the network for charging purposes, e.g. per-flow measurement, is to be avoided where possible is emphasised in [26]. Instead, meter resource consumption issues can be minimised with bulk metering. Collecting aggregated statistics only avoids the meter overhead caused by the collection of fine-grained statistics inside the network [77] at the expense of losing some information.

On the other hand, active measurement requires less resources. In this case, it can be carried out without much overhead in the meter machine. Suppose the provider A is interested in gathering one way delay statistics between its interface with the provider B as part of their service level agreement (SLA). Delay is measured in this case with a few test packets only, so there is no heavy overhead on the meters.

Metering inside the network, especially at inter-provider interfaces, is an important research topic but it is outside the scope of this thesis. The scenario considered is the one where service providers, namely application level providers and network retailers, directly interface with end customers. Wherever meters are installed, they certainly sit at the edges of the network introducing two regions of meter placements (Figure 3.4).

**Metering at the edges of the network**

First, when looking from the left side of the Figure 3.4, meters marked as 'M' may be installed at the provider’s domain. Such arrangements, inherited from the telecommunication sector, seem to be the standard practise at the present and are often referred to as metering at the provider premises equipment, say as a part of the remote access server (RAS) software.
Providing quality of service guarantees per packet at the edge of the network, for instance, might require charging per packet. The gathering of packet level information at the provider domain introduces performance and storage issues. Because the provider's meters measure hundreds of simultaneous user sessions, fine-grained metering is hard to implement.

With commonly used accounting protocols like RADIUS [148] and DIAMETER [35], the provider's access server (a network element where a simple meter is embedded) handles coarse-grained measurements, e.g. volume counts, per user sessions. Eventually, this data will be stored and further processed in centralised database servers. From practical experience, it takes less than a week to fill up 1 Gbytes of accounting information for 5000 dial-up users, for a small scale access service provider. Performance also degrades on the access server itself as it handles traffic volume per user session. By extrapolating this to another scenario based on very fine-grained metering, say packet-level monitoring, it seems infeasible to cope with the resulting data of all sessions and keep the CPU load of the access server low. As a result, provider meter placements lack the flexibility in choices of granularities - coarse-grained metering is the only practical choice.

Another consideration is that in some countries a service provider must by law hold its customers accounting data for at least three years. The necessary storage space becomes infeasible without a strong backup support system, even though this data is already aggregated per accounting session. The extrapolation to per-flow or even per-packet measurements suggests that fine granularities requires more storage space should they be measured within the provider domain.

Besides this, the large deployment of broadband connections (xDSL) over copper telephone wire is seen as a healthy step for today's telecommunication sector [74]. Users are offered more bandwidth, which is correlated to the storage space necessary for its accounting.
3.3. Metering Issues

For instance, in order to implement the usage-based policy currently imposed by NTL volume counts need to be stored.

The other important aspect of provider based metering is the trust model inherited from the telecommunication sector which, until now, has governed the relationships between service providers and customers. In the future, users will question whether or not their service contracts have been met in respect to the quality of the service delivered. From this viewpoint, customers may opt to run meters on their own equipment (Figure 3.4b). The belief that moving metering functions to the customer domain is likely to dominate in a future scenario is supported by the arguments that follow:

- **Scalability**: fine-grained metering within the provider domain demands processing power and huge storage space as it deals with a large number of user sessions. It is not clear now whether or not the provider access servers and associated measurement/accounting database servers will be able to cope with this type of granularity. Instead, if metering is shifted to the customer domain, this new distributed setting can afford finer granularities at a low resource consumption. Distillation of measurement data would be done at each customer meter before essential data is transmitted to the provider’s databases.

- **Measurement data accuracy**: by increasing the granularity of the measurement toward fine-grained, the overall data accuracy is increased. With this rationale, metering at the customer domain can offer greater metering accuracy for an improved user perception of the service provided.

- **Purpose built devices**: the number of dedicated routers that have recently been bought by broadband users for home networks has increased rapidly. Since this type of device deals only with one customer’s traffic, it has spare processing power to not only forward packets but also to meter traffic at virtually any type of granularity. This may alleviate the concern of shifting critical provider’s business functions, in this case metering, to unreliable installation environments such as the customer’s personal computers [26].

- **Real-time feedback**: some applications running in the customer system often require instant feedback on the network state for various reasons. Services such as video on demand need this feedback in the order of milliseconds to accommodate the service rate to the network conditions. A meter could provide this feedback only if placed closer to where the application runs (customer system).
• **Customer verification:** unlike today’s service provision models, customers in the future will be keen to independently verify their own charges to challenge their providers’ ones. On the assumption of usage-based charging schemes being widely deployed, a meter can be placed at a customer system for checking purposes.

Perhaps the person most interested in the detailed usage data is the customer; whereas the provider may be satisfied with a general view of the usage in aggregated reports [26]. Hence, moving metering and accounting to customer machines is a better match to the processing and storage that these tasks impose.

### 3.3.2 Flexibility

There are many key points related to metering at the edges of the network that could be summarised in order to characterise the tacit flexibility of the scenario of Figure 3.4. Some of them are listed below:

• **Meter placement:** the measurement region (where meters are placed) is defined by three metering arrangements. First, the provider domain is one possible metering point. Second, meters can be placed at the customer domain. Finally, customers and providers may each have meters running simultaneously in their domains.

• **Granularity:** the level of detail needed for the measurement data is not the same for each of the possible meter placements. To improve scalability, the provider meter carries out coarse-grained measurements, whereas the customer is able to measure and is much more interested in fine-grained measurement.

• **Type of measurement:** some metrics are measured with passive techniques, for instance the throughput at a specific network point. In contrast, other metrics need active techniques, for example the delay of a segment.

• **Metric availability:** when metric implementation code for a service specification cannot be located in the metering system, it is important to the system to have the right capability of finding it. This can involve one system contacting other metering systems across the interface domains to discover a particular metric code.

This naturally prompts the question, “how would one deal with the issues posed by this flexibility?” Could they be dealt with by a multi-purpose system that caters for the provider and customer needs listed above? The next section will explore these issues in greater detail.
3.3.3 Issue: Control of the Metering System

The metering system performs and reports measurements in the context of a management application. Irrespective of the application objectives, e.g. charging, the system needs to be instructed to measure whatever parameters the application deems necessary.

Manual configuration of the system suffices to the point where the time-scale of changes in the measurement parameters is not an issue [18]. In other cases, automatic configuration seems a reasonable controlling approach, for instance when the dynamic negotiation of SLA parameters takes place between customers and providers [127] [69].

With the dynamic tariffing and price announcement mechanisms described in [150], manual configuration of the metering system is impracticable. The proposed scheme advocates that accounting for usage is the customer system's responsibility but under the provider's control. The particular configuration of the metering system is driven by what this electronic tariff requires in terms of metering. The tariff needs runs from simple per-byte counts at one end to possibly delay, loss and jitter metrics. The ability to control the metering system based on SLA and electronic tariff structures in dynamic tariffing and QoS environments becomes an important topic of research that deserves attention.

First, we shall briefly define these two structures. An SLA is the legal service contract between a customer and its provider. In particular, this thesis is interested in the technical aspects of an SLA such as the parameters to be measured that convey the acceptable quality levels of a service, often referred to as service level specification (SLS) [83]. The electronic tariff, on the other hand, consolidates charging algorithms that upon evaluation produce a charge for the service. The tariff structure may be represented as a sub-component of an SLA [11].

Though this is a reasonable conceptual approach, we ought to bind these two structures in a third one: the network service specification. In this sense, the network service is specified at a high-level of abstraction by its acceptable levels of qualities (SLA) with a description of how the charge is calculated (Tariff). Additionally, there are links back and forth between these two elements of the service. An example is monetary compensation of any kind due to SLA breach, which can be clearly stated in the tariff.

The service specification along with its structures describes, perhaps not entirely directly, what it is necessary to measure. The question is how the metering system will be controlled based solely on this high-level specification.
3.3. Metering Issues

One approach to the problem is to manually decompose the service structure into a metering configuration. This has a major drawback. The customers may wish to validate on their own meters the usage for which they are being charged and/or whether the service contracted is within the acceptable terms drawn by the SLA. The service specification in addition to its equivalent metering control can then be sent to the customer equipment. In this scenario, discrepancies may occur in what is described in the service specification and what is being measured (metering control) may happen, either intentionally or otherwise. This raises a security issue.

Automatically deriving the metering configuration (low-level) from the service specification (high-level) is an alternative approach that relies on a validated component, the SLA/Tariff translator. Nevertheless, the problem becomes how the translator derives the metering control based on the parameters for which the service specification needs measurements (Figure 3.5). Suppose the simplistic example of a linear usage-based tariff function \( f(K, V, \Delta T) = K \times V \times \Delta T \) where \( K \) is the constant price coefficient, and \( V \) is the volume transmitted in bytes within the time interval \( \Delta T \) [75]. In this case, measurable values such as the volume of bytes involved in the communication should be supplied by the meter component. To accomplish this, the argument \( V \) of the tariff function needs to be translated into metering control code to be understood by the metering system.

Furthermore, other higher level specifications, not shown in the figure, such as the legal contract document between the parties concerned, need to be produced by technical and legal representatives. The time-line to negotiate the SLA for a dedicated line service is from three to six months [144]. This might be biased because it reflects the data of one network provider only (AT&T). However, it shows that at this level the service specification is formal, takes time
3.3. Metering Issues

to produce and is written in plain English. This type of specification is outside the scope of this thesis.

3.3.4 Issue: Security

A number of security breaches in electricity pre-payment metering were discovered by accident and exploited on a large scale, proving very expensive to fix [9]. Despite the fact that utility metering is a simpler scenario and much more regulated than the one envisaged in this thesis, it exemplifies the broad security concerns of metering customers’ traffic for charging purposes.

No doubt there will be incentives for customers or perhaps providers to cheat when flat-fee pricing is replaced with some advanced charging scheme. Such fraud may lead to drastic results when achieved on a large scale. Generally, there are two security goals. The first one is to prevent frauds committed by malicious customers (e.g. meter tampering). The second goal is to protect customers from any damage that an installed meter code may cause to its local system.

Within the context of this thesis, there is in general partial trust between providers and customers because of the unilateral risks that both parties have to consider. Even with this trust model, conflicts of interests are very likely to arise. Clark et al termed this conflict a “tussle” between the provider and its customers [47].

The customer may trust its provider that what is meant to be measured is being done so faithfully. However, there will be occasions when customers will wish to challenge their providers’ charges. In that case, customers will be keen to independently verify their own charges.

From the provider viewpoint, there is a chance that the meter code running at the customer domain has been modified to under-measure the quality of the service provided. If the provider pays a penalty for any SLA breach and this is exploited by a large number of customers, the provider may go out of business.

Furthermore, it has earlier been argued that meters are preferred to be installed at the customer site to achieve full scalability. Such arrangements raise security issues that need to be addressed within the context of Figure 3.4. Chapter 6 deals with them in more detail. We give an overview of these issues in this section:

- Prevention of tampering: the measurement data is vulnerable to modification while lo-
3.3. Metering Issues

cally stored or communicated over the network. The incentive to modify data is high.

- **Binding of measurement to context:** there will be many versions of a service specification with a different range of tariffs and SLAs running at the same time in the meter system. The latter must be able to bind the measurement data to the correct service specification version in an attempt to prevent problems with data inconsistency.

- **Execution of authentic code:** lack of integrity assurances in the host system of meter code is a major issue. The situation is aggravated if the code is to be executed in the customer system. A common threat is the installation of malicious code, which may compromise the host execution environment. Such code may also interfere with other ongoing measurements in the system by consuming excessive machine resources. This requirement is mandatory and may require safe protection implemented in software.

- **Deviation of control:** the service specification (high-level) is mapped to a metering configuration (low-level) by a validated translator component. Therefore, there should exist links between these two entities. In this sense, the integrity of the metering configuration has to be preserved to avoid any unauthorised modification.

- **Mutual authentication:** principals should know in advance who the other end of the communication is before a data or control session begins. They must identify themselves using unique credentials.

- **Non-repudiation of measurements:** proving what another party claims to be the correct measurements is difficult. The customer wishes to be positive that what was used has been properly measured, perhaps within an acceptable margin of measurement error. The provider wishes to receive an accurate summary of its customer's usage. To illustrate this case, suppose a customer agreed to take the measurements at his domain. At a later point, the customer's meter system reports a summary of the measurement data to its provider. The provider, however, notifies this customer that no summary has been recently received. This prompts the question: "what actions should both customer and provider take to avoid such a situation?"

- **Prevention and recovery:** denial of service (DoS) attacks in a metering scenario interfere with the normal operation and management of the meter system. As a consequence of the metering applications being responsible for measuring the usage of resources, the meter system is likely to be a victim of such attacks. It may be even more tricky where money is the motivating force, for instance with billing applications.
3.3.5 Issue: Performance

Finding the limitations of a measurement system in terms of performance helps with the understanding of its operation in heterogeneous host environments. More interesting is whether these limitations can be characterised. Provided the characterisation is possible, the network services may take these restrictions into account, perhaps by adapting to the limitations of a given host system.

Virtually all the measurement processing relies on how the packets of interest are transferred from the network card to the metering system. The packet capture element, the metering system unit responsible for this transfer, deserves further analysis.

Although metering performance concerns a broad set of interesting issues, this thesis narrows down the spectrum to answer two basic questions. What is the resource consumption of the packet capture module of a metering system? Can we characterise the metering error pattern?

The first question is pertinent to the levels of resources, in particular CPU, demanded by a metering system. The host’s operating system may not produce accurate measurements other than for a specific range of CPU load, which covers the measurement conditions. The CPU cycles used in a host environment such as the customer machine should be quantified: (a) to provide grounds to avoid interference to/from other processes running and (b) to know whether the measurement results can be trusted.

In contrast, the second question creates the need to better understand metering error and its effects. Possible outcomes are curves of different error functions in addition to the sources of error. The overall goal is to investigate the performance limitations of a metering system.

3.4 Approaches to Metering

A metering system that is tailored to collect fine-grained measurement data is also flexible enough to provide coarser data granularities when this is deemed necessary. Three approaches from the pool of packet monitoring systems surveyed in Chapter 2 exhibit this flexibility. Systems such as IPMON [77] and CoralReef [98] were primarily designed to measure fine-grained properties inside the network but, in some cases, can be made applicable to the edges of the network. The third system, the tcpdump tool [163], is also able to collect data at a packet level detail from any point of the network. Unlike the previous two, tcpdump can measure a network
segment by just connecting a low cost PC.

As far as flexibility is concerned, the lack of an automated configuration mechanism stands as one of the disadvantages in fine-grained meter systems. Nevertheless, a middleware component that sits between the metering system and the management application can still be developed to address this issue. The problem becomes how to define this middle configuration layer.

Another granularity that has gained considerable interest is flow data. Though it is somehow constrained (flow data does not provide the details of per-packet measurements) flow-based systems certainly suit a vast range of management applications’ requirements. Superflows or aggregates, for instance, can be created from a set of minor flows to improve scalability [18]. Some flow-based systems embedded in routers have been in use for measurements from the provider domain towards the core network [46], whereas other systems that run on dedicated workstations or PCs such as the RTFM [30] have been commonly applied at the edges of the network.

Designed from its inception to meet the needs of traditional management applications, the RTFM architecture relies on the static Meter-MIB structure [29] and SNMP for providing a manageable control interface. The static nature of the control interface is one of the problems with the RTFM since the scenario envisaged in this thesis needs more than a static MIB control interface. However, a case for a flexible interface applicable to dynamic tariffing has been made in [142]. Security is another problem. Outside the working scope of the group that devised the RTFM system, security was partially addressed using the security extensions of the SNMPv3 protocol[159].

Bulk measurement based architectures, evidently less flexible in respect of measurement granularity than flow systems, are feasible options for metering either inside or at the edges of the network [107]. Also, some form of fine-grained data can even be collected with systems such as RMON2 [159]. Unfortunately, due to its complexity, few routers have implemented the RMON’s system specification in full for performance reasons.

Large scale active measurement projects, including AMP [109], Surveyor [96] and PingER [105], rely on optimised meter placement strategies within the network. As a result, metering at the edges is outside the scope of these infrastructures.

On the other hand, modular architectures offer most of the flexibility aspects discussed in
Section 3.3.2 [135, 164]. Modules of any type of metric may be hooked in to build a particular infrastructure in NIMI [135]. In this system, the installation of a module is the site administrator’s responsibility who may or may not grant the privileges for running a module. This will depend on whether or not it trusts the source of the module, thus making security a major concern. Some of the issues raised in Section 3.3.4 were addressed in the NIMI project and its future research agenda includes issues of resource control and code validation [133].

In a similar way, the Active IP Accounting Co-processor Environment (AIACE) system [164] accepts accounting plug-ins to be installed into the network nodes, e.g. routers, to perform accounting tasks on behalf of accounting servers. The idea is to push intelligence into the network by using active network concepts applied to network management. AIACE defines a naive metering layer, without much intelligence, which is placed below an accounting layer termed as a *virtual accounting device*. In contrast, the NIMI modules are stacked up in the meter itself, allowing filtering at this level for performance enhancements.

These two modular approaches perform well when the issue is collecting data from the network for further and often non-realtime processing. However, realtime metering to provide instant feedback on the network state to some applications poses a few challenges.

If we were to assume a scenario where a realtime application, say IP Telephony, running at a customer system continuously requires the current state of the network (e.g. congestion level) in order to adapt its rate, this feedback about the network state may be part of a metering service. Should the feedback be required instantly, for instance the latency time between the meter and the application being of the order of tenth of a second, the meter ought to be located close to the machine in which the application is running, if not in the machine itself. As AIACE meters reside inside the network, it is very unlikely that low-latency feedback would be offered.

Even when the meter can be positioned at the edges of the network, e.g. customer machines, fine-grained measurements break the design principle of the architectures that are only capable of collecting bulk statistics [11].

To summarise, AIACE and NIMI lack choices of meter positions for the reason that the customer domain is not considered as a valid placement position. Furthermore, security issues, such as the binding of accounting data to a particular accounting plugin, were neglected in AIACE.

Other proposals deal specifically with metering as the lowest hierarchical function of a
3.4. Approaches to Metering

broader charging system [160, 39, 63, 72]. Hartanto et al [86] discuss a charging and accounting reference model structured in layers. From bottom to top, they are: (a) a metering layer that records usage of resources; (b) a collecting layer, which gathers data from the metering layer to pass on to the accounting layer; (c) an accounting layer that consolidates the collected data and correlates it with the user identities; (d) a charging layer, which creates charge records based on the accounting data and pricing schemes and finally (e) a billing layer, which consolidates the usage of a customer over a period and presents the bill.

Carle et al [39] introduce an analogous model, Generic Charging and Accounting (GenCA). In this framework, the RTFM architecture is used within the metering layer. To overcome the limitation of controlling the RTFM manager and reader by means of the meter-MIB [29], configuration middleware was introduced. Although the metering configuration was obtained out of charging and accounting policies specifications, no further detail was given on how it was derived (translation issue).

Metering inside the network, carried out at router gateways, was proposed in the context of the INDEX project [63]. INDEX’s billing system charges for TCP flows only. Any TCP opening connection requests pass through an admission control that is based on whether or not a customer is willing to pay for the connection. As the system is flow-based (measuring TCP flows) it suffers from similar overhead issues as encountered in Cisco Netflow [46].

With more focus on service level agreements, the CATI project [160] is a charging and accounting approach to the issue of maintaining electronic SLAs within the service provider administrative boundaries and between adjacent neighbours. An SLA expresses the service delivered, its duration and price. By means of a service broker, SLAs can be negotiated to deal with customer requirements. An important point missed is how parameters requested by an SLA are to be measured since little detail has been given on a metering layer.

According to Trimintzios et al [69], some technical performance parameters of an SLA (specifically the delay, jitter, packet loss and throughput) define service guarantees that a network provider offers on a per-flow basis to its customers. Their proposed SLA monitoring infrastructure is controlled by network policies written in a high-level language. These policies are then translated to an object-oriented policy representation. Every time a new high-level policy is introduced, its specification should be refined into network policies for each layer of a management hierarchy. The technique that refines high-level policies, or translates from one layer to the layer below, was not described and it is ongoing work [76].
Similarly, the classifier specification notation (CSN) [40], a declarative policy language, is able to express meter control rules. CSN rules can be converted into different rule representation to be deployed in a variety of metering systems.

In a similar way, the active monitoring language (AML) [79] describes what to measure, without worrying about how to measure it. AML was discussed in the context of an end-user monitoring system named SoLOMon as a mechanism to discriminate what to measure. None of these two research projects have addressed the issue of converting AML or CSN specifications into a particular meter configuration.

Erena et al [95] also discuss a charging system but in a telecom operator context. This work is an outcome of a technical consortium between telecommunication companies. Consequently, the measurements in this proposal are to be taken inside the network, in particular within the segment owned by a network provider. The network performance level structure (NPL) conveys performance information in terms of delay, jitter and loss in order to support the service requirements. Scalability is the main concern identified by the authors since the collected measurement data have to be deposited in centralised servers. Security is the second major problem. As in the majority of measurement infrastructures projects, security was deliberately left outside the scope of this project.

Likewise, centralised SLS monitoring components running in the provider domain raise similar scalability issues with the system introduced in [11]. These components are failure prone or may build bottlenecks if the number of customers, and hence the number of monitored SLS, significantly increases.

3.5 Summary

This chapter has examined two supporting cases for metering by analysing a business context where service providers interact directly with end-customers. The deployment of a QoS mechanism on the Internet and mechanisms to charge for the QoS delivered were analysed. The importance of fine-grained usage metering and accounting to the widespread of these mechanisms was also presented. Wherever meters are installed in this context, they certainly sit at the edges of the network, introducing a region of two-meter placements - the provider or the end-customer domain.

The rationale for moving metering functions to the end-customer domain was discussed. This scenario can afford finer granularities at low resource consumption, making the overall
system scalable. The flexibility of meter placement, on the other hand, poses issues that need to be addressed. The issue of controlling the metering system based on high-level code (service specification) was presented. Security is another concern, as there will be incentives for customers and even providers to cheat when advanced charging schemes are used. Finally, performance issues associated with a metering system are important to understand, as to see how the business scenario presented in this thesis may be affected by metering inaccuracy.

Related work that address these issues were surveyed. Some of the systems were designed to run in managed elements inside the network (e.g. NIMI, AIACE and AMP), in order to collect coarse-grained measurement data. Other systems may be used at the edges of the network (e.g. RTFM and BPF-based tools), but lack security and an automated management mechanism.

The next chapter argues that the metering issues identified in this chapter are still open. It introduces the EdgeMeter infrastructure as a platform to facilitate the metering of networked services.
Chapter 4

EdgeMeter System

This chapter describes the EdgeMeter system and its design principles that aim to facilitate the accounting of network services and the measurement of the quality of service received by end customers.

To address the metering issues discussed in the previous chapter, EdgeMeter supports QoS measures within a certain accuracy, as well as safe transport and storage of measurement data. It also supports the discovery and installation of new metric modules and the flexible configuration of metering points based on electronic service specification (tariff/SLA) automatically distributed by service providers.

The components and mechanisms employed by EdgeMeter to accurately account for resources used by network services are described, and their importance to the scenario considered in this thesis (metering at the edges of the network) is discussed. Some of the ideas introduced in this chapter were previously presented in [138, 136].

4.1 Introduction

In the previous chapter, metering issues in the context of charging for the quality of service received by customers were discussed. It was also shown that metering at the edges of the network presents a viable placement arrangement to measure the performance of future network services such as IP Telephony.

Considering these issues, some of the architectures surveyed in Chapter 2 seemed suitable candidates for a measurement platform at the edges of the network. However, in their original design, these architectures present some characteristics that suggest the contrary, as follows:
• Some of the systems are designed to run in managed elements inside the network in order to collect bulk statistics of the passing traffic [109, 96, 105]. Taking this characteristic into consideration, these systems are neither appropriate to the process of metering at the edges of the network, nor to the fine grained measurements required by most realtime applications.

• Some of the architectures provide for the installation of plug-ins that implement a passive or active metering test [135, 164]. However, as just mentioned, these architectures have also been designed for scenarios where metering is crucial but is conducted inside the network. In principle, it rules out any measurement being undertaken at the edges of the network, say, for instance, in the customer domain.

• Flexibility of specifying the type of intended granularity in passive metering is not well addressed. Some of the proposed architectures deal solely with fine-grained measurement [98, 163, 30], whereas a few others are capable of providing coarse-grained measurements [107]. They fail specifically in presenting a versatile approach to the specification and the actual metering of a range of granularity.

• Any control exercised over the metering system should be derived from a network service specification, i.e. the combination of an electronic tariff and a service level agreement (SLA) components. None of the systems have explored this issue any further, apart from the work on the CSN language framework [40]. Details on the translation mechanism have not been given, though, leading to the conclusion that this problem is still open.

• Metering at the edges of the network is highly exposed to security threats as customers will eventually be billed for the usage of a service. Hence, security issues must be cautiously analysed and addressed. Unfortunately, these issues were neglected in the majority of the architectures surveyed, except for the subset discussed in Section 3.3.4 that have been tackled in the NIMI project [135].

• Last but not least, performance issues if not fully considered may degrade the normal behaviour of a metering system. The operating system of the host machine may not produce accurate measurements other than for a narrow range of load. In this case, finding the performance limitations of a metering system and its host environment helps to achieve greater measurement accuracy. Due to its complexity and its dependency on many variables, however, the performance related issues have been left on hold in the majority of the surveyed systems.
The EdgeMeter architecture, presented in this chapter, addresses the above issues.

### 4.2 Business Context

Dynamic charging environments, in which electronic tariffs are announced by means of multicast channels to billing and SLA management applications [150] [41], present a setting for the business model introduced in Chapter 1. To recap, the business roles involved consist of (a) **network providers** (bulk provider, wholesaler and retailer), which sell network-based services (e.g. data transportation, network access); (b) **application-level service providers**, which sell services such as audio/video streaming and IP Telephony; (c) **network management service providers**, including meter manufacturers that sell services such as metering software tailored for billing; and (d) **customers** who subscribe to a service at a certain level of quality guarantees.

The business interfaces between customers and service providers are established through a contract. Such legal documents have clauses, sometimes referred as service level objectives (SLOs), which specify the acceptable service performance (SLA), in addition to a tariff to be used for charging for the service. Eventually, this high-level specification of the service will be coded into an electronic form, the *service specification* code, which is the combination of an SLA and a tariff.

The service specification introduces the metering requirements of these business relationships. An electronic tariff, for instance, may require usage-based accounting of the service offered to a customer. Therefore, the metering service is an important functional piece in this business context.

This section introduces aspects of deployment of a metering system, the EdgeMeter, as part of a general charging scenario. The deployment follows two phases. In the initiation or code distribution phase, the EdgeMeter code is installed in the appropriate points of measurement. In the following operational phase, the metering system is instructed to measure the network traffic based on the requirements posed by a service specification code.

### 4.2.1 Initialisation: Code Distribution

The EdgeMeter code is distributed and installed during the initialisation phase. This phase comprises steps from the meter development (manufacturing) to its distribution to the intended metering points (provider, customer or both domains). These points are located at the edges of the network, as discussed in Chapter 3 (Section 3.3.1).
Apart from the standard means of code distribution (e.g., download from a web server), it is envisaged that active network themes may be helpful in this process. The system code can be embedded in a proxylet [80] and made available in some repository (proxylet servers). Such a practice would facilitate the dissemination of new code versions 'on the fly', at the expense of some additional management to correctly maintain these codes in the proxylet servers.

Figure 4.1 shows the three business roles involved. The meter manufacturer, a third-party player who is regarded as a management provider in our business model, develops the meter code under a contract with a customer or service provider. The latter in some circumstances may well play the role of a meter manufacturer by not only distributing the meter but also developing it. However, the customer might not accept this as it can create a conflict of interest.

The provider buys the EdgeMeter code to install in its own domain in order to measure the customer traffic (step 1). When the agreed metering point is the customer domain, the provider sends the EdgeMeter code to the customer (step 2).

This setting does not preclude customers from buying meter systems themselves directly from meter manufacturers for verification purposes. In this case, the network traffic will be measured simultaneously at two points, when leaving the provider domain (provider meter) and entering the customer domain (customer meter).

4.2.2 Operation

After initialisation, the second phase consists of the metering operation in a dynamic charging scenario, as sketched in Figure 4.2. It is important to note that the meter system was deliberately placed in the customer domain in the figure merely for illustration purposes. Moreover, any point at the edge of the network is a valid placement arrangement as part of the initialisation phase discussed in the previous section.
4.2. Business Context

This figure presents the business relation between two parties. On the right hand side, the customer system hosts EdgeMeter. Some of the provider's charging elements are shown on the left hand side. These elements expect to receive the output of the metering system, i.e. measurement data. The billing system is the one responsible for generating a bill to a customer based on the tariff structure assembled in the service specification code. The SLA maintainer strips out the SLA part of the service specification to continuously validate the service level contracted.

The service specification code is disseminated by the service distributor (provider side) to customers by means of either multicast or unicast (step 1). When a group of service receptors (customer side) are the target, multicast distribution is the preferred method; whereas unicast makes sense if communicating a service specification to a particular customer. Concomitantly, the service specification code is sent to the billing and SLA maintainer elements for the set up of the provider’s management tasks.

Upon receiving the service spec, the receptor element passes it on to the EdgeMeter system (step 2). This is a supported form of EdgeMeter's high-level configuration code that will be eventually translated into low-level metering configuration code. An internal element not shown in the figure, the service translator, performs the translation from high to low-level.

For the sake of clarity, Figure 4.2 shows a simpler scenario, where management elements (billing system and SLA maintainer) run on the provider side only. In real deployment, they...
could be replicated on the customer side for realtime billing and SLA maintenance. In such cases, a copy of the service specification code is sent to these elements during the second step.

The EdgeMeter architecture is modular allowing modules that take care of a metering function to be enabled on demand. However, these may need to be loaded from external repositories (module servers) as presented in the third step.

There will be a number of customer applications running in the customer system that might use measurements of the network. A video streaming application may need realtime feedback on the current state of the network, e.g. the number of congestion marks [138]. EdgeMeter properly configured takes such measurements and sends them to these applications (step 4).

Furthermore, the full collected measurement data can be locally stored (step 5). In fact, the customer itself is the party most interested in this extensive data so as to obtain itemised accounting of the network resources usage.

The provider, on the other hand, might generally be satisfied with just a summary of the measurement data [26]. As the sixth step in the figure, such a summary report is communicated to the billing system and SLA maintainer elements. The arrow between these two elements, as shown in the figure, presents the connection given by the terms of a service specification. The penalty a provider is liable to pay to a customer, for instance, if an SLA clause is violated may be described as a discount in the tariff part.

The operation phase pictured in Figure 4.2 considers EdgeMeter as a 'black box' that takes an input, the service specification (SLA/tariff), and provides tailored output, i.e. a set of measurements the service specification code requires. The following sections go a step further by introducing EdgeMeter inner components and their internal interactions.
4.3 Architectural Microscopic View

EdgeMeter employs a policy based approach in its internal structure. Details of this approach will be given in the sections that follow. In the meantime, Figure 4.3 presents an overview of the system structure. Based on their functionality, the internal components are shown enclosed in different functional groups:

- The configuration input group includes the components that export a set of interfaces tailored for configuring EdgeMeter. Currently, two types of configuration interfaces are available. The service translator interface allows high-level specification code to be input as pictured in Figure 4.2 (step 2). The other interface, the policy reader interface, accepts a low-level form of configuration.

- Configuration of EdgeMeter hinges on a policy specification that states what should be performed; it is not concerned at all with how tasks ought to be carried out. The policy management group deals with all aspects related to policies, ranging from their specification to their runtime maintenance.
• As mentioned previously, the architecture is modular, allowing modules to be hooked in on demand. Modules include code to implement metrics that specify what and how a measurement should be performed. The module management group provides the components that cope with the tasks of managing modules within the system, such as:

  - loading modules into the system,
  - scheduling modules, so that each can have a share of the system to perform its task, and
  - searching externally for modules that cannot be located in the system.

• Components of the network input group deal with information extracted from packets received from the network.

### 4.4 Configuration of EdgeMeter

This section deals with the issues of configuring EdgeMeter based on high-level specification code.

#### 4.4.1 Policy Centred Approach to Meter Management

Policy-based network management has been extensively discussed in the research community and standardisation bodies to address the issue of network configuration. Service providers recognise the need for clearly specifying at several levels of abstraction, from high-level (e.g. business goals) to low-level (e.g. device-specific configuration parameters), what a network policy is without the need to detail which mechanisms to use for its implementation. By doing so, there will be a separation between policy and mechanism.

This uncoupling allows extensibility in a new type of network management that spans the traditional approaches where otherwise policies and mechanisms were mixed together. For example, the description of QoS requirements for a realtime application in terms of high-level policies to be applied to the network can drive the low-level configuration of admission control mechanisms by means of a 'high-to-low' level mapping function. While high-level policy specification in this example may be unique, its low-level derivation is not. As network elements are different and produced by various vendors, they get configured differently, so there will be a relation of many to one low-level configuration codes (one for each type of network element) to a single high-level policy specification.
4.4. Configuration of EdgeMeter

In summary, a policy states what tasks to perform leaving the decision of how to perform them to the mechanism that implements such a policy. A general definition for a policy was given in [169]: a policy is a set of rules to manage and control access to network resources.

The policy framework currently under development within the IETF employs a policy-based approach to admission control decisions. Applying this framework to mechanisms that service providers deploy to enforce the quality of service requested by its users is one of its objectives. In this sense, the IETF framework has focused on DiffServ [18] and IntServ [21], two QoS frameworks developed by IETF working groups (refer to Chapter 3 for a discussion). Other proposals implement this policy framework in various contexts, policy-based accounting [176] and security [102] to cite a few.

The IETF framework at a glance adopts two architectural elements. First, the Policy Enforcement Point (PEP) element sits at network nodes and is responsible for the enforcement of a policy. The Policy Decision Point (PDP), the second element, makes policy decisions which are the evaluation of a policy rule. PDP is a remote entity that communicates to a PEP using the network. The interaction between these elements is described in [174]. PEP is triggered by a message, for instance an event occurrence, that needs a policy decision to be made. In a follow-up, PEP contacts a PDP passing on encapsulated objects to assist in the policy decision process. Once a decision has been reached, PDP sends back to PEP the policy decision to be locally enforced.

Even though the approach taken in EdgeMeter also relies on policy rules, there are significant dissimilarities from the above framework. To mention a few, the points of decision and enforcement have been simplified by merging them into the EdgeMeter’s internal system structure. Also, the specification of a policy rule and its management are simpler than the detailed policy specification of Policy Core Information Model (PCIM) [115], the data model extensively employed in the IETF policy framework. Furthermore, nothing precludes EdgeMeter being used in the context of the IETF framework. Because EdgeMeter is a network element customised to monitoring, it takes on the role of a PEP should the need for doing so arise.

What makes the policy framework of EdgeMeter uncomplicated is how each policy is represented. A policy rule statement is a composition of events, conditions and actions (ECA). The essence of ECA rules in EdgeMeter has been borrowed from rule-based approaches proposed in the context of active databases [171, 131]. These approaches have also motivated the design of some network policy languages [17, 59]. Damianou et al [59] describes a declarative and
object-oriented policy language, the Ponder language, that employs ECA rules in support to the specification of security policies for access control.

### 4.4.2 Service Related Control

There are two mechanisms for inputting configuration in the EdgeMeter system. Service specification code disseminated by a service distributor element is one of them (Figure 4.2). Since this code is written in a high-level language, it is currently specified in a XML-based (Extensible Markup Language) [23] language. This type of input mechanism is termed high-level configuration.

As a policy rule (ECA rule) is the basic configuration unit in EdgeMeter, any high-level configuration input needs to be first translated into a policyset. The latter is termed low-level configuration because it specifies a set of policy rules that enclose metering-specific configuration parameters. The *service translator* component does the actual translation (Figure 4.4). As such translation is vital to the scenario of tariff/SLA distribution presented in Section 4.2.2, Section 4.7 will cover its aspects.

It is not always the case that a high-level configuration will be the primary form of
4.4. Configuration of EdgeMeter

EdgeMeter configuration. The system is flexible enough to allowing direct management using a policyset as its configuration input. When the application scenario is not the dynamic dissemination of service codes, the policy reader component (top right of Figure 4.4) can be used as policyset receiver.

4.4.3 Policyset

A policyset is a small program in our proposed language - EdgePol (see Appendix B) that encloses policy rules in its body part. This part allows the specification of declarative statements. The idea is to describe when and what tasks are to be carried out in the form of ECA rules.

The ECA model has been contextually adapted to cope with the dynamics of a metering system. A ECA rule states what events, conditions and actions the system should react to (see Appendix B for a description of an ECA rule). A brief overview will be provided in this section.

- The event part of a rule describes occurrences that are of interest to the rule [131]. Such happenings convey the current state of a metering system and its surrounding elements. An event can be single or compound. The former type is raised by a single occurrence of a category list such as temporal and protocol related events. A compound event, on the other hand, is raised by some combination of primitive and composite events by using logical operators.

- The condition of an ECA rule is optional and it is evaluated when an event occurs. When no condition has been given, an event-action type of rule results.

- An action is a set of tasks to be executed when an event is triggered and the rule’s condition evaluates to true.

To manage the EdgeMeter system, a policyset is required. Each part of a policy rule is appointed to an internal component. The sections that follow will explore this division. The terms “policy rule” and “ECA rule” will be used interchangeably throughout this thesis.

4.4.4 Measurement specification

While some of the systems proposed in the literature collect network performance statistics at a very high level of aggregation, a few others present capabilities of collecting at finer granularities, often at packet level, offering the maximum level of detail that would be expected of a management application (see Section 3.4 for a discussion of these systems). A mechanism
4.4. Configuration of EdgeMeter

Figure 4.5: Example of TCP network field and flux

that is designed from its inception to cover all the measurement granularity spectrum, from one extreme (fine grained) to another (coarse grained), would be ideal.

To overcome this inflexibility, EdgeMeter introduces a novel notion of metering at virtually any granularity. This section unveils the network field measurement specification as a viable scheme to bring versatility in choosing, specifying and metering a variety of granularities. By tuning a few parameters, this scheme is capable of assembling most typical granularities, such as packet-level [163], flow-based [44] and bulk-based statistics [107].

The concepts of network field and its measure, the network flux, are detailed in Appendix B (EdgePol language). The example of Figure 4.5 briefly illustrates these new concepts. First, it shows a TCP network field created by all packets travelling from a source IP address, say 10.0.0.1, to any other host in the Internet. The protocol attribute value in each of the IP header is set to 6 (TCP protocol). The field lines visualised in the figure coincide with three TCP connections initiated by a host machine (e.g. source IP 10.0.0.1).

We now focus our attention on how these two concepts may be characterised in engineering terms to address the issue of specifying a measurement. This section presents an overview of two templates in EdgePol: one specifies a network field; whilst the other describes a network flux.

- **Network Field Template**: it defines the characteristics of a network field that will eventually be measured by a metering system. A network field can be regarded as a set of one or more streams of packets observed by a metering system.

- **Network Flux Template**: the observed network field needs to be intercepted by the metering system to collect the statistics required about the packet streams that form the network field. The measurement of a network field leads to a network flux. This template
4.4. Configuration of EdgeMeter

networkfieldtemplate NField1 {
  type {
    srcipaddr { (eq,10.0.0.1) } and
    protocolid { (eq,tcp) };
  };
  fieldline { newline }; 
  activeperiod {
    starttime { };
    stoptime { };
  }
}

Figure 4.6: Example of a network field template

describes the specifics of how a network field is to be measured. The template includes some attributes of the interception loop area, a time interval ($\Delta T$) and the metrics to be metered.

In order to specify the TCP network field of Figure 4.5(a), the template may be the one shown in Figure 4.6. The type of network field is described by the attributes within the type statement; it provides a form of packet stream filtering. Therefore, some filter expressions with attributes obtained from the packet header/content are allowed. The relational expression in the template example states that the network field of interest is TCP (protocolid = 6) and it originates from the IP address 10.0.0.1 (srcipaddr = 10.0.0.1).

The fieldline attribute determines the way new packet stream lines should be arranged. Many field lines can be spawned out of a network field template. They may be grouped into a single field line or multiple field lines. Wherever it is necessary to distinguish between packet stream lines, the keyword newline can be used. In the example, each packet stream that matches the field type results in a new field line. In contrast, the keyword sameline groups the packet streams that match the field type in a single field line.

Suppose a packet has 10.0.0.1 as its source IP address and destination IP 10.0.0.20. Streams of this packet will be regarded as a new field line when the template of Figure 4.6 is applied. In addition, another packet with identical source IP address but different destination, say 10.0.0.111, will generate another new field line. In contrast, a single field line to account for these two packet streams would be generated if the keyword sameline were used instead.

The last attribute inspected, the activeperiod, specifies a time period in which a network
4.4. Configuration of EdgeMeter

networkfluxtemplate NFlux1 {
    networkfieldtemp1 { NField1 };
    interceptionloop {
        dstipaddr { (eq,10.0.0.111) and (eq,10.0.0.20) };
    };
    deltat { (50,seconds) };
    metric { metric1 };
    meterprofile { mpl };
    report { };
}  

Figure 4.7: Example of a network flux template

field is active and operating. This attribute is largely used to stipulate when a measurement is to be taken, as this requires an active network field. Such a time representation is grouped in two categories. The first one, the absolute time, specifies precisely a list of one or more time intervals. In contrast, the periodic representation allows easy specification of intervals for a network field life-cycle. In the example, the active period of NField1 includes all of time since the attributes starttime and stoptime were left empty.

Figure 4.7 shows a possible network flux template for the example of Figure 4.5c. The networkfieldtemplate attribute lists the fields in which this network flux will be measured. Similar to the type attribute of a network field template, the interceptionloop holds filter expressions that combine attributes and values captured from packet headers and content. For instance, all packets that have destination IP addresses that match the interceptionloop filter expression will be measured by this network flux template. This corresponds to packets forwarded for destination hosts 10.0.0.111 and 10.0.0.20.

The time interval ($\Delta T$) is the time component of a network flux. It is specified as a tuple of a value and its unit; the value is 50 and its unit is seconds in Figure 4.7. Packets or bytes are examples of other acceptable units. Packets that belong to field lines intercepted by a loop area will be aggregated over a time interval set in the deltat attribute (4.5c).

The metric attribute determines what has to be metered within a network flux. How such a measurement should take place is described in the internals of the metric specification (metric1), which is not shown in this example (refer to Appendix B for more detail). The implementation code of metric1 is a module that is plugged in EdgeMeter (module management is discussed in
Section 4.5).

Finally, the *meterprofile* attribute contains properties of the meter, for instance the meter location and protocol to communicate, where a network field will be measured.

```plaintext
on (report networkflux1)
  if (dscopoint.ef == true)
    do {
      charge := charge + get_charge(p1);
    }
```

Figure 4.8: Example of an ECA rule

Figure 4.8 introduces an example of a policy rule that uses the resulting data of the template *NFluxI*. The event statement, in this case any report of the *NFluxI* template, is followed by the keyword `on`. The `if` statement corresponds to the condition part, while the action is enclosed by the `do` statement. When a report of *NFluxI* is produced (event), the condition is checked. Not only that but when the marked DiffServ codepoint [18] is Expedited Forward (EF) the action block will be executed. In the example, this action block updates the charges for the service.

### 4.4.5 Policy Management

The treatment given to a policyset at runtime is presented in this section. Within the *policy engine* element (Figure 4.4), a policyset should be analysed, decoupled and instantiated.

The standard unit of a policyset instantiation in EdgeMeter is a *metering session*. Sessions are owned by the policyset owner. The interface exported by the policy engine comprises operations for loading a policyset and subsequently creating the associated metering session:

- boolean *LoadPolicyset*(Policyset P): load a policyset P that has been received by either the service translator or the policy reader. Returns true if the operation was successful or false otherwise.

- SessionID *CreateMeteringSession*(Policyset P): creates an instance of policyset P, i.e. a metering session. To each session, an ID is assigned that is unique within the system.

A policyset may contain the specification of resources that are required to be attached to a metering session. As such, a session has instances of other specifications such as network
field and flux templates. The operation `CreateMeteringSession()` instantiates these templates. Should a module also be required, it is loaded and subsequently associated with this session.

It might be easier to imagine this association as a tree with metering sessions as root nodes; network field, flux and module instances would form the respective branches of a session tree. Links between two siblings are possible too. A network flux node in this tree, for instance, connects to a network field node since the former is the measure of the latter. On another front, a metric module would be linked to a network flux node and so forth.

Once a metering session has been created, the policy engine manages ECA rules contained in the policyset by adhering to the steps [131]:

- **Triggering** the corresponding policy rules upon receiving events signalled by the `event detector` element. A triggered policy rule should then be inserted in a `conflict set`.

- Policy rules in the conflict set should be scheduled for analysis. The `scheduling` step determines when the condition of a rule is evaluated and its subsequent action statements executed.

- The condition of a triggered policy rule needs to be `evaluated`. The condition evaluates to either true or false.

- In the fourth phase, the `execution`, actions of triggered rules are executed provided that the rule’s condition has been evaluated to true. Firing a policy rule can also be regarded as executing a rule.

The four steps above detail the processing algorithm of a policy rule. Initially, the policy engine investigates whether there are any triggered rules to process. If this is the case, the engine chooses a policy rule $R$ from the conflict set. Second, $R$'s condition is evaluated; if it is true, then $R$'s action part is executed.

Event triggering is outside the policy engine scope. Instead, the `event detector` captures the events raised by different types of sources and it signs them to the policy engine. It is compulsory to register the events of interest well in advance by invoking one of the event detector interface’s operations:

- `boolean RegisterEvt(Event E)`: registers an event of interest $E$, which is to be eventually signalled.
4.4. Configuration of EdgeMeter

- **boolean SignEvt():** signs registered events to the policy engine.

The event received in the engine that matches a policy rule triggers this rule. Figure 4.4 shows the registration of an event made by the policy engine to the event detector. Events can be categorised in respect to their source in the following groups:

- **Network Flux:** it includes events such as the creation, modification or deletion of a network flux. The example shown in Figure 4.8 has an associated network flux event, which is the start of a new report. It is important to note that the network flux concept is general. As such, it encapsulates any packet-related events, for example. Additionally, protocol-related events (e.g. IGMP join messages, RSVP setup message, DHCP allocation message) are also included in this category based on the same grounds.

- **Clock:** temporal events might cause an ECA rule to be triggered at an absolute time (e.g. 21/01/2004 at 18.00), or at periodic intervals (e.g. every 10 seconds) [141] [142].

- **External:** the event is raised by an occurrence outside the EdgeMeter system.

- **User Defined:** an application declares an event \( E \) (e.g. user authentication) which triggers some policy rules. The application in this case takes care of notifying the event to the event detector. This category allows flexibility for creating events tailored to the application needs [171].

- **Exception:** events in this category are raised as a result of some error or fault in the system. Suppose a situation where a requested metric module is not locally available. This may trigger an exception that leads to a search for the module.

- **Communication:** events raised by the communication mechanisms used in the system. To name a few, mechanisms such as the distribution of service specifications, gathering of measurement data and so forth are possible sources.

The scheduling phase deserves special attention in view of the fact that multiple rules may be triggered simultaneously. For instance, several policy rules can stipulate the same triggering event. Consequently, it may be necessary to select a rule among triggered rules. The selection process is known as conflict resolution. The rule scheduler is responsible for choosing the next policy rule to be fired (executed). Arbitrarily choosing a rule to execute seems the easiest way to resolve a policy conflict. However, it can lead to unfairness in the selection process. As a result, two other approaches were considered. The dynamic approach decides the next rule to
be executed based on the dynamic properties of rules, such as the history of rule triggering (e.g. time of event occurrence) [131].

The drawback of dynamic approaches lies in the non-deterministic behaviour of the system. Because the priority to be established is a function of the dynamic properties of rules, it makes any pre-runtime selection of rules hard to implement.

To avoid this the default policy scheduling in EdgeMeter prioritises rules, in case of a conflict, based on the priority specified in a static manner. A rule may be scheduled based on the explicit priorities specified in the definition of a policy rule, or it may be selected by taking into account other static properties of rules, such as the rule creation time.

Finally, as the measurement base adopted in EdgeMeter relies on network fields and flux, it seems relevant to put them in the context of policy management. When a rule is fired, the action statements are executed sequentially. If there is any network field and associated network flux templates to install, the policy engine will request the necessary resources to the module manager. The engine will then be served by the resource modules loaded in the system. A brief description of the module management is given in the next section, whilst Chapter 5 presents some of its design issues.

### 4.5 Module Manager

A module is the unit of code in EdgeMeter. Most metering resources will come embedded in a supplied module code. A delay metric of a network flux template, for instance, is implemented as a module.

Some module types (not an exhaustive list) are listed in Tables 4.1 and 4.2. Any number of module groups are permitted but the examples of the tables were arbitrarily categorised in two groups, the metric modules and special modules for data aggregation and specialised analysis.

Each module exports a service interface that can be further inspected by the policy engine.

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay</td>
<td>Measures one-way packet delay between two hosts</td>
</tr>
<tr>
<td>Loss</td>
<td>Measures packet loss of a communication link</td>
</tr>
<tr>
<td>Volume</td>
<td>Measures volume in bytes within a given time interval</td>
</tr>
<tr>
<td>Peak Rate</td>
<td>Measures peak packet rate over a time period</td>
</tr>
<tr>
<td>Congestion Level</td>
<td>Measures congestion level as experienced by the network using ECN marked packets [145]</td>
</tr>
</tbody>
</table>

Table 4.1: Examples of metric modules
4.5. Module Manager

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Aggregator</td>
<td>Further aggregates measurement data resulting from network flux</td>
</tr>
<tr>
<td>Data Miner</td>
<td>Analyses measurement data to suit a particular accounting application</td>
</tr>
<tr>
<td>Correlator</td>
<td>Correlates measurement data resulted from various network flux</td>
</tr>
</tbody>
</table>

Table 4.2: Examples of modules for data aggregation and analysis

or other module. The module's purpose determines its interface's set of operations.

However, the implementation of a module is not revealed. This allows different implementation of a module to be used. Suppose the one way delay metric is required in a network flux template. It might have two implementations, one using the IETF one-way-delay technique [5] and the other code measuring the round trip time (RTT) by pinging a host and dividing the time by 2. Though they are different codes, they do implement the same interface.

It is important to note, however, that a module should be implemented in a safe language such as Java [167]. The actual format of the module will depend, thus, on the language used. It is envisaged that a module can be implemented as an active service [8] that permits programmability of network elements, in this case the EdgeMeter system, but only at the application layer. The module may be executed in the network nodes by embedding it in a code unit called a proxylet [80]. Even the service specification code (Tariff/SLA) may be embedded in a proxylet to be distributed by a service provider to its customers [138]. A proxylet interface exports any operation as deemed necessary [80] but an earlier implementation of it [137] has limited the operations for security reasons.

Sethaput et al describes Regatta [151], a monitoring framework built on active network themes, that employs dynamic modules to supervise nodes inside the network. Two types of module were proposed: (a) a module for failure diagnostics, e.g. latency test, and (b) a module for failure repairs, e.g. alternate route module. Regatta's modules were also implemented as portable Java code. The modules are deposited in special repositories.

EdgeMeter takes a similar approach to Regatta for storing modules. Both frameworks employ a module repository. Unlike Regatta, the repository in EdgeMeter can be either local or in the network, preferably hosted in a trusted domain. This might be from the party providing the metering system, the service provider in most cases.

The module manager element offers an interface that is tailored to the maintenance of modules in the system. The exported operations are primarily implemented in the loader ele-
4.6 Network Input

Access to incoming packets from the network is an essential operation for EdgeMeter to support passive and active measurements (see 2.2.2 for a discussion on measurement techniques). The network input sub-architecture of EdgeMeter derives from the basic Berkeley Packet Filter (BPF) framework outlined in [106]. The motives that led to picking this framework to be part of EdgeMeter are:

- A vast number of running codes. Over the past years, BPF has been implemented in almost all operating systems, including Unix [92] and Windows [149].

- Optimisations on the original BPF architecture promise gains in performance [14]. BPF+, the optimised version, allows packet filters expressed in a high-level language to be optimised, for instance, removing filter redundancies. A compiler translates the resulting filters in a highly efficient native implementation to be delivered to an execution environment across the user-kernel boundary.

It is also acknowledged that BPF presents some performance limitations that necessitate careful analysis and quantification. As EdgeMeter relies on BPF, Chapter 7 evaluates the network reader element, which is a wrapper for BPF.

EdgeMeter allows sessions to gain access to captured packets by introducing the following interface:

- boolean InstallNetworkField(NetworkFieldTemplate NFT): installs a filter code which corresponds to a filtering expression written in the type attribute of a network field template (Figure 4.6). This is the only type of filtering carried out in
4.6. Network Input

Figure 4.9: Network reader

the kernel space. The second type of packet filtering, the filtering expression of an interceptionloop attribute specified in a flux template (Figure 4.7), is carried out in user space.

- PacketEvent ReportNetworkData (Mode M): a PacketEvent object which carries parts of packets, usually the initial $n$ bytes, can be continuously pushed out or polled. Mode $M$ parameter specifies how the event will be signalled. If it is push mode, the event object is sent from time to time to a set of registered listener elements. This can be performed by using previously registered call-back functions, for example. When $M$ is pull mode, the interested EdgeMeter elements should periodically poll the network reader element for more packet related events.

Figure 4.9 introduces the network reader and elements that surround it. At a glance, there are three boundaries: (a) the hardware space that includes a Network Interface Card (NIC); (b) the kernel space including the NIC device driver, the BPF kernel buffers and the OS protocol stack; and (c) user space where all EdgeMeter elements sit. The path taken by a packet begins in the NIC (Network Interface Card) when it copies the packet to a pool of system buffers (receive DMA ring) using DMA transfers. These buffers are accessible (using shared memory) to the NIC device driver, as shown in the figure. This process is termed the first copy.

The NIC generates an interrupt, which triggers the NIC device driver interrupt handler routine to interact with NIC and copy packets from the receive DMA ring to a kernel buffer (BPF
4.7 Translation Mechanism: Overview

The ability to control the EdgeMeter architecture through high-level service specification is one of the key aspects presented in this section. Such high-level code is specified using the SLA/Tariff specification language (STS). STS is an XML based language (see Appendix A for details), and it is translated into intermediate code, a policyset in EdgePol (see Appendix B). The service translator is also referred to as EdgePol translator in this section. An overview of the translation mechanism is shown in Figure 4.10.

![Figure 4.10: Overview of the translation mechanism](image)

The **front-end** takes the service specification in STS as the input. This block performs three functions: (1) **Service Partitioning** which separates the Tariff and SLA structures and builds a directed acyclic graph (DAG) associating them; (2) **Network Field Binding**, in which...
case network field and flux templates are separated. Required metric specifications are linked to network flux templates; (3) **Core Translation**, which is the stage when the intermediate code, the policyset, is generated.

The **middle block** performs the necessary optimisations and relevant safety checks in the intermediate code. Some of the optimisations might be the ones put forward in the BPF+ proposal [14]. In addition, the EdgePol translator may use a safety checker such as the proof-carrying code (PCC) mechanism proposed by Necula et al [117] [118]. PCC allows a consumer, in this case the EdgePol translator, to export a set of security policies to which any incoming code created by a producer must adhere. It is important to note that the intermediate sets of code will be created by the front-end and mainly optimised by the middle block. They represent the metering configuration (policyset), the Tariff and SLA structures. From the point of view of the EdgeMeter architecture, the metering configuration code represents the requirements expressed by the service specification, in particular the Tariff and SLA. Therefore, a policyset is enough to represent the metering configuration, whereas the SLA and Tariff code may be translated to a target code during the back-end processing. The latter translation suits billing applications, for example, but it is outside the scope of this thesis.

Finally, the **back-end** receives the intermediate code in some abstract machine representation and generates the target code. Final optimisations might be performed at this stage.

### 4.8 Module Discoverer

Extensibility and flexibility have been previously discussed as the key aspects of EdgeMeter. Modularity offers high flexibility in supporting new (perhaps unplanned) network services as the metering system can naturally evolve by extending its original pre-arrangement. To offer modularity, however, EdgeMeter strongly relies on the availability of modules to implement it.

Management of modules, therefore, has to be carefully considered. An important question to address is what follows if a requested module is not available in the system. The easiest answer would probably be for EdgeMeter to deny such a request irrespective of whether or not the unavailable module may be later located.

However, it is important to exhaust the list of alternatives before denying a module request. The alternative chosen in EdgeMeter has been discussed at a high level while introducing the module manager element in Section 4.5. To recapitulate the LoadModule() operation, when a module is locally unavailable, an external lookup follows. In case a module has been success-
fully located in any external module server (repository), the request is completed. Otherwise, the request is denied but at least an extensive discovery is carried out.

Attention is now turned to the mechanism employed for the discovery of modules. This prompts the question: "How does a meter find the closest module server to use?" This has the following requirements:

- The module discovery mechanism should return the topologically closest module servers to the meter issuing a request, i.e. closest in terms of delay time. The dynamic tariffing scenario may involve new service specifications being distributed. It is likely that such changes will trigger requests for modules. As a result, the topologically closest servers will be preferred to reduce the overall latency of downloading a module. The network location of a meter issuing the request and servers responding to it should be calculated.

- Flexible queries provide a means to search servers based on different types of information. By using boolean expressions, the search operation may return not only exact matches but a range of matched modules per server. Queries such as the closest servers that hold modules for data-mining rather than a particular module is one example.

Figure 4.11 outlines the discovery service offered by the module discoverer element. The discovery mechanism follows seven steps:

1. Each participating element in the discovery process, either a meter or a module server, has a location in the network. This step requests such a location for a module server from the network location service.

2. The module server's network location is calculated and returned.

3. Like the other participant, the meter requests its network location.
4. The meter's location is returned by the network location service.

5. Module servers hold information about the modules that they currently store. This information will be indexed in the distributed repository service.

6. Meters request the discovery of a module by issuing simple or boolean queries.

7. The distributed repository service returns references to requested modules when they have been successfully located.

The network location may be geographical. As one of the requirements is the search for topologically closest servers, however, it makes sense to use a location that describes the underlying network topology. To this effect, network measurements such as delay can be used as underpinning for a location.

The relative network location and information about the modules are described as points in a multidimensional vector space $V^d$, where the first $k$ dimensions comprise the network location and the remaining $d - k$ are reserved to specify information regarding the modules. Different dimensions of $V$ can accommodate different types of information (e.g. module group and version) or information regarding the module server (e.g. machine characteristics).

Suppose a particular module server has its network location relative to other servers as the point $(200, 100, 40)$. Additionally, this server uses the names of modules as the access key to a module. When applying, for instance, a function that maps the module name (string) into a key number, it results in the key 44734000, and the final point becomes $(200, 100, 40, 44734000)$.

It is convenient to address the search over these points in the vector space $V^d$ by using a multidimensional search mechanism [158, 15, 88]. In fact, three types of query seem applicable [19]:

- **Point Query**: specifies a point $Q$ and retrieves all points in the vector space with identical coordinates.

- **Range Query**: a query point $Q$, a distance $r$ and a distance metric $M$ are specified. The outcome of this query includes all points $P$ from the vector space, which have a distance smaller than or equal to $r$ from $Q$ according to the distance metric $M$. The latter can be the Euclidean metric ($L_2$ norm) or other appropriate distance metric.

- **K-Nearest-Neighbour Query**: this query type returns $K$ closest points to a query point $Q$ in the vector space.
4.8. Module Discoverer

A point query can be regarded as a special case of a range query with a radius \( r = 0 \). In this thesis, the range and K-nearest-neighbours queries apply to any dimension of the vector space. By employing boolean expressions, these types of queries can be constructed and searched for in any dimension required.

A meter may be acting as a module server by offering its cached modules to other meters. The contrary does not hold; a module server cannot be a meter. The dual role of a meter puts more flexibility in the system by introducing space storage otherwise limited by the availability of module servers.

The interface exported by the module discover element encompasses operations of a module server and a meter, yet considering the meter’s dual role:

- **Point FindNetworkLocation(EntryPointList E)**: operation invoked by either a meter or a module server. It returns the network location based on delay measurements of the requesting node to the points in the entry point list. This list holds the IP address of other module servers or meters that will act as the reference points for the joining node. The dimensionality \( d \) of the vector space will depend on the size of the entry point list \( \text{size} \) such that \( d = \text{size} + 1 \). This interface operation accounts for steps 1, 2, 3 and 4 of Figure 4.11.

- **boolean InsertModule(ServerID ID, Point P)**: inserts modules in the distributed repository. This operation is invoked either by a meter wishing to act as a module server or by a particular module server. This operation corresponds to step 5 of Figure 4.11.

- **ModuleServerList LookupModule(Query Q)**: returns a list of module servers that hold modules that matched query \( Q \). It is expected that the meters investigate the returned list so as to choose one appropriate module server to download the intended module. This operation corresponds to steps 6 and 7 of Figure 4.11.

Some of the design and implementation issues of a technique that implements the FindNetworkLocation() operation will be described in Chapter 5. This technique is used to calculate the network locations of module servers and meters as points in a coordinate system.
4.9 Summary

This chapter described the EdgeMeter system and its design principles, which aim to facilitate the accounting of networked services and the measurement of the quality of service received by end-customers. EdgeMeter supports QoS measurements within a certain accuracy, safe transport and storage of measurement data. It also supports discovery and installation of new metric modules, and the flexible configuration of metering points based on electronic specification (tariff/SLA) automatically distributed by service providers.

Deployment aspects of EdgeMeter as part of a general charging scenario were introduced. The deployment follows two phases: in the initialisation or code distribution phase, the EdgeMeter code is installed in the appropriate points of measurement; in the following operational phase, the metering system is instructed to measure the network traffic based on the metering requirements described in a service specification code.

The EdgeMeter inner components and their internal interactions were presented. The service translator receives service specification code (SLA/Tariff) from the provider's system through the network. This code is then translated to a policyset, which manages the EdgeMeter system. Once all required modules have been loaded, either locally or from external module servers, measurement data starts to be produced. The chapter concluded with a discussion of the service translation and the module discovery mechanisms.
Chapter 5

EdgeMeter Design Issues

The previous chapter described the EdgeMeter system architecture. This chapter details the internal data structures and algorithms of the following components of EdgeMeter:

- **Module Manager**: modules may be loaded from local or pre-defined external module servers, and are assigned to metering sessions at runtime. This component offers an interface for modules maintenance in the system.

- **Module Discoverer**: if a module cannot be located locally, the module discoverer performs a lookup within a set of servers. Lighthouse, a scalable network location framework, is proposed and evaluated as a possible implementation of this component.

- **Service Translator**: the design of the service translator, which is the translation mechanism of service specification code into a policyset, is introduced in the last section.

5.1 Module Manager

A module is the unit of code in EdgeMeter as discussed in Section 4.5. The module manager element offers an interface that is tailored to the maintenance of modules in the system. This section details the operations exported by this interface.

5.1.1 Loading a module

The `LoadModule()` and `BindModule()` operations are responsible for loading and binding, respectively, a module to the current metering session. A module is available from a local or external *module repository server*. The following types of loading are implemented by this operation:
5.1. Module Manager

- **Local loading**: the solicited module is available in the local repository. Apart from locally loading the module, no other action is required in this case.

- **External loading**: the module is not locally available, so discovery among a pre-defined set of external module servers is necessary. When the module is embedded in a proxylet, the repository is implemented as a proxylet server described in [80].

Module discovery, which involves lookup within a set of module servers, is carried by the module discoverer (See Section 4.8 for a discussion of this EdgeMeter element).

5.1.2 Session Scheduling

Ideally, a mechanism should be in place to isolate metering sessions, so that 'cross-talking' of activities of one session cannot interfere with another. Resource isolation in programmable networks is discussed in the RCANE architecture [110]. As an EdgeMeter metering session shares some similarities to an RCANE session, it makes the RCANE a suitable framework to approach the issue of metering session isolation.

RCANE employs vertical layering to separate sessions. Horizontal layering is used to create abstractions to control and reserve resources in advance (CPU, memory and network bandwidth). As resource scheduling occurs in a layer below the user session layer, each session can operate independently.

CPU management in RCANE is based on three abstractions [111]:

- A **virtual processor** (VP) is a regular allocation of CPU time. Tasks performed within a VP share the allocated CPU time.

- A **thread** is the basic unit of execution. A thread may be *Runnable* (in execution), *Blocked* (e.g. on a semaphore, or waiting for I/O processing) or *Idle* (awaiting the arrival of work items).

- A **thread pool** is a collection of threads in the system. A thread pool acts as a queueing and dispatch point for packets and events in RCANE. Each pool is associated with a particular VP.

The first two RCANE abstractions were applied in EdgeMeter for management of metering sessions. Each new metering session creates a new VP with a single thread and local memory associated with it. The session thread is responsible for:
5.1. Module Manager

- Organising packet data received from the network reader. The modules linked to the session store and classify data based on network field and flux templates. For example, data related to a TCP network field will be classified and stored accordingly.
- Storing network field and flux related data into a hash table for easy and quick access.
- Raising an event of new network flux data.

The scheduler applies a scheduling policy (e.g. WFQ [60]) to select between the thread pools (and threads) for execution.

5.1.3 Session Memory

Each EdgeMeter metering session has two types of memory available for data storage. One of them, the measurement memory, holds data associated with a set of measurements in a session lifetime. With the measurement specification discussed in Section 4.4.4, this data results from measurements of one or more network fields. It is expected, therefore, that such a memory be classified based on network field and flux templates.

As for the other type of memory, the auxiliary memory, it is useful for miscellaneous types of storage. When modules of a session request additional memory for a particular computation, auxiliary memory might be used. Besides, cases such as thorough analysis of network traffic also justify this type of memory.

The objective is to achieve efficient and structured storage of network flux related data. Because a metering session can be concerned with more than one network field (e.g. TCP and UDP fields), a flexible data structure that could evenly distribute network field related information would be useful in reality. An open hash table suits this purpose, as each network field creates a class or a slot in the hash table. Any related network flux data is inserted in its associated network field slot afterwards.

The network field hash table is a data structure of \( M \) classes, each class being a network field, numbered \( 0, 1, ..., M - 1 \). A hash function \( h_1(k_1) \) relates a network field (with key \( k_1 \)) to the class that it belongs to. Keys may be a combination of attributes used in expressions of the type attribute (of a network field template). For instance, the attributes \( \text{srcipaddr} \) and \( \text{protocolid} \) combined would make key \( k_1 \) for a TCP network field.

For each slot of a network field hash table there is a pointer to a linked list. These \( M \) lists assist the storage of network flux. Each element \( i \) of a list points to a second hash table that
Figure 5.1: Network field and flux hash tables

holds network flux data.

Figure 5.1 presents an example. All network field templates that have been associated with a metering session are inserted in the network field hash table, which can hold up to $M$ entries. Each element of this table holds a fixed portion of information (e.g. the active period of a field) along with a pointer to a list of network flux hash tables. In this example, the first element of the network field table points to a list in which the first element contains a pointer to a second table, the network flux table 1. Subsequent elements of this particular list will point to other network flux tables accordingly.

A second hash function $h_2(k_2)$ allows direct access to network flux hash tables. Attributes of filter expressions specified in the interceptionloop attribute of a flux template may generate the access key $k_2$.

The $j$th element of the flux table 1 holds pointers to other linked lists. One list for each metric of interest is specified in a network flux template. If the specified metrics are, for instance, delay and loss, there will be two pointers to two linked lists. Each list will store metric measurement data aggregated over a time interval. Unlike the lists pointed to by the elements of a network field hash table, these lists hold the actual data, as opposed to pointers only.

As far as a network flux is concerned, its temporal component is significantly important. To this effect, each element of a linked list pointed to by an element of a network flux table has already been aggregated using the time interval (deltat attribute of a flux template).

The average time per insertion and lookup operations for a hash table with $S$ slots and $N$ elements stored takes $O(1 + N/S)$ time. The constant 1 is the time to find the slot, and $N/S$ the time to search the linked list [2].

Finally, these two types of hash table organise network field and flux data in such a way
that any granularity of measurement data can be permanently kept.

For packet level granularities, the network field of interest is the one created by a unique stream of packets, so a single element of the network field hash table suffices to represent this type of granularity. This element points to a single flux table too. The latter holds a pointer to a linked list formed by individual packet headers/content that contains information such as a timestamp that supplies the time reference of a received packet.

The bulk-based granularity uses a similar arrangement and a number of tables. However, in this case it is important to store the aggregated data concerning various metrics. To achieve this, there will be as many lists as the number of metrics pointed to by a flux hash table element. In each of these lists, data aggregated over a time interval will be stored.

There will be more than one element in the network field table when storing flow-based measurement data as the type attribute of a network field template includes the microflow attributes. In complement, the interceptionloop attribute, which defines access keys to the flux table, might be present as well. Should this be the case, additional lists pointed to by this latter table will be essential (Figure 5.1).

5.2 Module Discoverer (Network Location)

The LoadModule() operation, exported by the Module Manager, does an external module discovery if the module is locally unavailable. In case a module has been successfully located in any external module server (repository), the request is completed; otherwise, the request is denied.

The interface exported by the module discover element encompasses three operations for locating a module code in the network. It is envisaged that distributed search algorithms such as the XenoSearch [158] or a distributed form of multidimensional binary search trees [15, 88] will be used as the implementation of two interface operations, namely the InsertModule() and LookupModule() (refer to Section 4.8 for further details of this interface specification).

This section describes in detail the framework to calculate a network location for a meter or module server. This is used as the implementation of the PointFindNetworkLocation(EntryPointList E) operation. A meter or module server will be referred to as a node in this framework.

Before turning to the details, an important concept should be introduced. Network prox-
5.2. Module Discoverer (Network Location)

imity, in the context of this thesis, refers to how close node A is to node B in respect to the underlying IP topology. We characterise it with measures of IP network performance. The propagation delay, for instance, can indicate whether or not two nodes are close neighbours.

To capture the proximity between nodes, their location in the Internet can be calculated using a set of coordinates. How such locations are computed is the main concern of this section.

Two definitions are needed. First, a **general space** \( M \) is defined by the pair \((X, d)\) where \( X \) represents the set of valid objects and \( d \) is a distance function, either metric or non-metric, i.e. does not follow the triangle inequality property, which represents the distance between these objects such that \( d : X \times X \rightarrow R \). In contrast, a **vector space** is a set \( V \) that is closed under appropriate vector addition and scalar multiplication operations.

These definitions have a broad scope. A general space represents objects and their mutual distances, whereas a vector space represents objects, their distances and locations. On the Internet, the space \( M \) may be a set of network nodes (objects) spaced according to a particular network performance metric (distance). For instance, properties such as propagation delay and bandwidth can define two types of distance measures, so under certain assumptions they create two metric spaces.

The problem then is defined as follows. We refer to it as the *mapping* problem and it consists of:

- finding a scalable mapping method to transform objects \( \{x_1, ..., x_n\} \), in our case, network nodes (meter and module servers), of the original space \( M \) onto points \( \{v_1, ..., v_n\} \) in a target vector space \( V^k \) (\( k \) is the dimensionality) in such a way that the distance measures (i.e. delay) are preserved, i.e. \( d(x_i, x_j) \sim D(v_i, v_j) \) for \( i, j > 0 \); where \( D \) is another distance function.

- **constraint**: only a few distance measures between these objects are known. This is because the system should scale and having a full distance matrix \( |X| \times |X| \) is impractical.

The constraint above leads to the use of **pivoting** techniques to map the location of an object in a general space onto a vector-space location. These techniques consider the distance from a given object to a number of pre-selected **pivots** \( \{p_1, ..., p_n\} \in X \). The pivots are the nodes in the EntryPointList parameter of the FindNetworkLocation() operation. Pivoting is the common framework for a large class of nearest neighbour algorithms [73, 152].
In what follows, Lighthouse, a technique that employs pivoting to solve the 'mapping' problem, is outlined. These ideas were previously published in [136].

5.3 Lighthouses

We start by introducing Figure 5.2a. The basis $G$ of this 3-D vector space comprises vectors $\{l_1, l_2, l_3\}$. The second observation is that $G$ must be formed by well-known pivot nodes, i.e., the same nodes must be contacted by every joining node. In fact, this is a characteristic of Global Network Positioning (GNP) [124], 'binning' [147] and 'beaconing' [100] frameworks in terms of how they manage reference points. It turns out that this characteristic has the disadvantage that it makes the system not fully self-organised. What happens if the pivot nodes (e.g. landmarks/beacons) are not available at a given instant of time? Who should a joining node contact instead to locate itself in the system?

To overcome the above issue of well-known pivots, Lighthouse$^1$ is presented, a technique that explores two concepts: multiple local bases together with a transition matrix in vector spaces. Lighthouse allows the flexibility for any host to determine its coordinates relative to any set of pivot nodes provided it maintains a transition matrix. Such a matrix does what the maritime chart does for navigation. It gives a basic instrument for gauging a global position when this is deemed necessary. With the idea of local positioning, better scalability of the system can be achieved. Figure 5.2b shows a follow-up configuration of the global basis scenario achieved with the Lighthouse framework. Now nodes $n_1, n_2, n_3, n_4$ are located in different local basis, $L$ and $L'$, in a decentralised manner.

Figure 5.3 presents an example of this technique applied to a 3-D real vector space. Points

---

$^1$Historically, lighthouses played a vital role in navigation. The first and most famous lighthouse, Pharos of Alexandria (Egypt), was built about 270 B.C. When looking at this unique tower with a bright light at the top, a ship's crew could compute their local position relative to it (local reference). Eventually, the position could be transformed into a global one by using maritime charts and the like. Nowadays, GPS (Global Positioning System) with its replicated service has made this method redundant.
at the left side plot network nodes, either meters or module servers, as they might be observed in the IP network (general space M). The right side shows the same points mapped onto a vector space $V$. With pivoting, arbitrary $k + 1$ local reference points are chosen, which we call lighthouses. Our framework relies on a set of nodes from which different joining hosts may select differently. However, each of these hosts has to preserve the invariant: a transition matrix $P$, which is only applicable to calculating a global position, has to be correctly maintained.

Details of the four step procedure followed by a joining host will be described in the sections that follow.

### 5.3.1 Finding Lighthouses

The bootstrap of the system occurs as follows:

- **Joining node**: a new node $n_i$ finds an entry point node $n_j$, i.e. any node that is already in the system. Node $n_j$ provides to $n_i$ a list of nodes that can potentially act as $n_i$ lighthouses. The joining node selects $k + 1$ nodes among those in this list. It then constructs a local basis $L = \{l_1, l_2, \ldots, l_k\}$, where each vector $l$ is a pair of lighthouses. This basis spans the $V^k$.

- **First nodes**: when $n_i$ is the $m$-th node such that $m \leq k + 1$, $n_i$ is considered as one of the first nodes. As $n_i$ cannot have other $k + 1$ lighthouses, it constructs a local basis with the lighthouses that have already joined. The idea is to build the first basis after $k + 1$ nodes have joined the system.

Once the joining node $n_i$ has been given a list of nodes that can act as its lighthouses, it measures a set of network performance metrics between itself and the lighthouses. The technique by which these measurements are undertaken will vary according to the context. The IDMAPS project [78] found that the propagation delay can be triangulated, so the delay between points $(a,c)$ can be estimated based on the delay between $(a,b)$ and $(b,c)$. As a result, the round-trip time (RTT) measured through ICMP ECHO packets may be a practical tool to incorporate delay as a metric. Additionally, techniques that measure the available bandwidth
5.3. Lighthouses

look promising. However, we have only explored the network delay metric for this work.

With a $k \times k$ matrix of network performance metric values, the joining node computes the coordinates of a local basis $L$.

![Diagram](image)

Figure 5.4: 2-D example

Figure 5.4 introduces a 2-D example. It is assumed that there are six nodes already in the system: $\{n_1, n_2, n_3, n_4, n_5, n_6\}$ (Figure 5.4a). Suppose a new node, $n_7$, wants to join in. As the first step, it contacts a node in the system, say $n_4$, in order to get a list of lighthouses. In this example, $n_4$ sends a list of three nodes to act as $n_7$ lighthouses: $\{n_4, n_5, n_6\}$. At this time, $n_7$ starts measuring the distance, propagation delay, between itself and the three lighthouses.

The following sections describe the method that calculates a local basis $L$ using the lighthouses.

### 5.3.2 Local Basis Coordinates

Any node that wants to take part in the system has to compute its own coordinates relative to a local basis. However, it must first determine the coordinates of the basis that it will be using. To do this, node $n_i$ calculates $L = \{l_1, ..., l_k\}$ where $l_i$ is a pair of lighthouse nodes $\overrightarrow{ni}, \overrightarrow{ni}$. It applies the Gram-Schmidt process (QR decomposition) [54] described as follows.

$$
\begin{align*}
    l_1 &= \text{proj}_{W_0} l_1 + \text{proj}_{W_0}^\perp l_1; \\
    l_2 &= \text{proj}_{W_1} l_2 + \text{proj}_{W_1}^\perp l_2; \\
    & \quad \vdots \\
    l_k &= \text{proj}_{W_{k-1}} l_k + \text{proj}_{W_{k-1}}^\perp l_{k-1}.
\end{align*}
$$

(5.1)

Where \(\text{proj}_{W_{k-1}} l_i\) is the projection of $l_i$ along the finite-dimensional subspace $W_{i-1}$ of $V^k$, whereas the vector $\text{proj}_{W_0}^\perp l_1$ is called the component of $l_1$ orthogonal to $W_{i-1}$. 
We shall explain the Gram-Schmidt process with a 3-D basis construction example. In the first step (Figure 5.5a), \( l_1 \) is projected into subspace \( W_0 \). Vector \( l_1 \) now spans the one-dimensional subspace \( W_1 \). In the second step (Figure 5.5b), vector \( l_2 \) is projected along and orthogonal to \( W_1 \). Over the last step (Figure 5.5c), vector \( l_3 \) is calculated as the sum of its component along the subspace \( W_2 \), spanned by \( l_1 \) and \( l_2 \), and by its component orthogonal to \( W_2 \).

The joining node, \( n_j \), uses the Gram-Schmidt process to compute a local basis \( L = \{ l_4, l_5, l_6 \} \) (Figure 5.4b).

### 5.3.3 Host Coordinates

At this stage, node \( n_i \) has fresh coordinates of its local basis. It may now calculate its own set of coordinates. However, as a side-effect of choosing arbitrary \( k + 1 \) lighthouse nodes to span the vector space \( V^k \), it is probable that these vectors will form an alternate basis, not necessarily an orthogonal one. Chances are that the computed basis will be oblique. In that case, it should be able to use any type of basis (oblique/orthogonal), so that node \( n_i \) coordinate vectors will be a linear combination of the local basis \( L \):

\[
\mathbf{n}_i = c_1 l_1 + c_2 l_2 + \cdots + c_k l_k
\] (5.2)

By taking the inner product \( < \cdot, \cdot > \) of both sides of Equation 5.2 with every vector in \( L = \{ l_1, l_2, \ldots, l_k \} \), we are left with the following system of equations:

\[
\begin{align*}
< \mathbf{n}_i, l_1 > &= c_1 l_1, l_1 > + \cdots + c_k l_k, l_1 > \\
< \mathbf{n}_i, l_2 > &= c_1 l_1, l_2 > + \cdots + c_k l_k, l_2 > \\
& \vdots \\
< \mathbf{n}_i, l_k > &= c_1 l_1, l_k > + \cdots + c_k l_k, l_k >
\end{align*}
\] (5.3)
Solving the system 5.3, the scalars $c_i$ are obtained. As the only given input to the technique is the distance measures (e.g. delay) from the joining node to a set of lighthouses, an expansion of the system above is essential. Thus, two formulae of Algebra are required.

$$< u, v > = ||u|| \cdot ||v|| \cdot \cos(\overrightarrow{u}, \overrightarrow{v})$$ (5.4)

$$< u, u > = ||u||^2$$ (5.5)

Formula 5.4 gives the cosine of the angle between two vectors $u$ and $v$, while formula 5.5 is a derivation of the first since the angle $\theta$ between identical vectors is 0. Substituting these two formulae in the system 5.3 yields:

$$\begin{align*}
  c_1 ||l_1|| + \cdots + c_k ||l_k|| \cdot \cos(\overrightarrow{l_1}, \overrightarrow{l_k}) = ||n_i|| \cdot \cos(\overrightarrow{n_i}, \overrightarrow{l_1}) \\
  \vdots \\
  c_1 ||l_1|| \cdot \cos(\overrightarrow{l_1}, \overrightarrow{l_k}) + \cdots + c_k ||l_k|| = ||n_i|| \cdot \cos(\overrightarrow{n_i}, \overrightarrow{l_k})
\end{align*}$$ (5.6)

In synthesis, the node’s coordinates are calculated by solving the system of linear equations (5.6). Geometrically, this represents the projections of the node distance measures along the vectors of the local basis $L$.

In the 2-D example (Figure 5.4c), node $n_7$ solves a simple linear system in two variables: $c_1$ and $c_2$. As a result, the coordinates of $n_7$ become $c_1 \cdot l_1 + c_2 \cdot l_2$, where $l_1 = \overrightarrow{n_4n_5}$ and $l_2 = \overrightarrow{n_4n_6}$.

### 5.3.4 Transition Matrix

Nodes are allowed to arbitrarily choose their lighthouse nodes (local basis) provided they preserve the invariant of rightly maintaining a transition matrix $P$. The question is how a joining node knows about the global basis $G$ without measuring any property between itself and the nodes that form such a basis. To answer this question, we bring the idea of basis changing into our technique.

If the basis for a vector space $V^k$ is changed from some old basis $B = \{u_1, ..., u_k\}$ to some new basis $B' = \{u'_1, ..., u'_k\}$, then the old coordinate matrix $[v]_B$ of a vector $v$ is related to the new coordinate matrix $[v]_{B'}$ of the same vector by the equation:

$$[v]_{B'} = P^{-1} [v]_B$$ (5.7)
where the columns of $P$ are the coordinate matrices of the new basis vectors relative to the old basis, that is the column vectors of $P$ are:

$$P = \begin{bmatrix}
[u_1^j]_B \\
\vdots \\
[u_k^j]_B
\end{bmatrix} \quad (5.8)$$

As a result, node $n_j$ computes a transition matrix $P$ between its local basis $L$ and the global basis $G$. This does not require any additional distance measurements. The only requirement is that the entry point node $n_j$ supplies either the coordinates of $G$ or its own $P$ transition matrix.

The transition matrix $P$ calculated by $n_7$ (Figure 5.4c) contains the coordinates of the local basis $L = \{l_1, l_2\}$ relative to the global basis $G$. These are the coordinates of the light-houses that compose $L$, i.e., $\{l_4, l_5, l_6\}$. Therefore, $n_7$ derives $P$ with nothing more than the information it already has.

Nodes are expected to re-calculate their coordinates from time to time due to frequent network topology changes (e.g. when a link shuts down). Such changes are captured by the network performance metrics used, such as the end-to-end delay. In this case, a participating node re-computes its coordinates following the four steps above. If for some reason a lighthouse node becomes unavailable during this re-calculation process, the participating node then chooses an alternative lighthouse to derive the transition matrix.

### 5.3.5 Experimental Evaluation

This section presents an analysis of Lighthouse accuracy. In this context, accuracy is how close the distance predicted by this technique is to the real distance measured. If a high level of accuracy is achieved, that means accurate node locations can be calculated. The accuracy of Lighthouse delay estimates were compared against the estimates of the Global Network Positioning (GNP) framework.

The GNP framework [124] for predicting Internet network distances is based on absolute coordinates computed by modelling the Internet delay as a real vector space. In outline, the GNP architecture is formed from two parts. First, a small set of well known hosts ( pivots) called landmarks locate themselves into a real vector space by measuring their mutual distances (delay). These coordinates are taken by hosts that wish to join the system as a global, and therefore unique, basis of the vector space. The landmarks’ coordinates are calculated through
5.3. Lighthouses

Table 5.1: Key Parameters

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>Distance</th>
<th>Probes</th>
<th>Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$\ell_2$ (Euclidean)</td>
<td>4</td>
<td>$10^{-5}$ (GNP only)</td>
</tr>
</tbody>
</table>

the solution of a relative error minimization problem: $\sum_{i,j} Error(d_{ij}, \hat{d}_{ij})$ where $d_{ij}$ and $\hat{d}_{ij}$ are the measured and estimated distances between the landmarks $i, j$.

The second part of the framework relates to how an arbitrary host calculates its own absolute coordinates based on the landmarks’ own coordinates. The joining host measures its round-trip delay to the landmarks and then casts the computation as an overall error minimization problem: $\sum_{i,j} Error(d_{ij}, \hat{d}_{ij})$ where now $d_{ij}$ and $\hat{d}_{ij}$ are the measured and estimated distances respectively from host $i$ to the landmark $j$. $Error()$ is an error measurement function.

The data used in this experiment was the global data set collected by the GNP project\(^2\). It consists of two matrices with delay measures. The probe matrix holds the mutual distance measures between 19 probes distributed around the world. Twelve of these probes were in North America, 5 were in Asia, and 2 were in Europe. The second matrix, called the target matrix, contains the delay measures between 869 target hosts and the 19 probes. The delay was measured by ICMP ECHO packets.

Table 5.1 shows the key parameters used in both implementation of these two techniques. Lighthouse was implemented in C++ and used the Gauss Elimination technique to solve the system of linear equations 5.6. The tolerance parameter, shown in the table, was the convergence error of the minimization method used by the GNP code.

The strength of Lighthouse, as explored in the previous sections, is its capability of working with multiple local bases through oblique projections. To fairly compare this technique to GNP, the experiment was limited to a unique and global basis.

Four arbitrary probes were chosen from the nineteen to serve as the lighthouses and landmark nodes. With the distance between four probes, Lighthouse code computed a local basis for a 3-D vector space, whereas the GNP code calculated a global solution for the distance error minimization problem.

A common framework was required to compare both techniques. Hence, the evaluation was divided in two sub-processes. The first one, called calibration, relates to how accurate a

\(^2\)Measurement data available at http://www-2.cs.cmu.edu/~eugeneng/research/gnp
technique is when computing the local basis (Lighthouse) or the global basis (GNP). Distance measures between the four chosen probes were required for this sub-process. The extrapolation, the second sub-process, tells how accurate a technique is at predicting distance measures between arbitrary nodes. Two accuracy metrics were used.

\[
\text{Relative Error} = \frac{|\text{Measured} - \text{Estimated}|}{\text{Measured}} 
\]

(5.9)

\[
\text{Ratio} = \frac{\text{Estimated}}{\text{Measured}} 
\]

(5.10)

Formula 5.10 gives the ratio of an estimated to a measured distance. The relative error metric (formula 5.9) results in zero when the estimated distance matches the measured distance.

In Figure 5.6, the Cumulative Distribution Functions (CDF) of the relative error of Lighthouse and GNP are shown for the calibration sub-process. As expected, both techniques achieved high levels of accuracy measured by their relative errors. It has to be pointed out that the measured distances between the four probes should match the distances computed by each technique as this is the given input information. This property determines how well the technique can extrapolate distance measures. Lighthouse presented almost the same average accuracy of GNP. Both techniques could estimate 99% of the distances within a relative error of 0.5 or less. Despite this margin of error, this result outperforms other Internet distance estimation techniques such as IDMAPS [78] as discussed in [124].
Figure 5.7 compares the CDFs of the ratios of Lighthouse and GNP delay estimates to those measured. Ideally, a curve resulting from this metric is a vertical line at \( x = 1 \). Despite the fact that the two techniques presented equivalent results, Lighthouse was slightly better than GNP for ratios less than 1. On the other hand, 70.34% of GNP estimates were within a 25% error margin as opposed to 69.61% of Lighthouse estimates. As much as 41% of Lighthouse and GNP estimates were within an error of 10%.

In total, 869+14 hosts were used in the experiments. Lighthouse could have used any local basis from a combination of 888 hosts taken 4 at a time, i.e. \( C(888,4) \). This yields a selection of \( 25.7 \times 10^9 \) bases that a joining node can choose as opposed to only one global basis offered by GNP. This gives numerical support for why Lighthouse should scale better than GNP.

### 5.4 Design of Service Translator

In this section, a detailed description of the service translator is given. The design of the translator is based on the three main blocks presented in Section 4.7 (front-end, middle block and back-end). There are six stages that transform the original service specification composed by tariffs and SLAs structures into a configuration code suitable for a metering system. Figure 5.8 shows the translator design.
5.4. Design of Service Translator

Figure 5.8: Service translator
5.4. Design of Service Translator

5.4.1 Stage 1: Service Partitioning

<?xml version="1.0" encoding="us-ascii"?>
<networkServiceSpecification ID="servicel">
  <SLA>
    [...]
    <SLO ID="SLO1">
      <dayTimeConstraint>
        <dow>Sun Mon Tue Wed Thu Fri Sat</dow>
        <startTime>00:00:00</startTime>
        <endTime>23:59:59</endTime>
      </dayTimeConstraint>
      <reportInterval>
        <every>1</every>
        <unit>minutes</unit>
        <dow>Sun Mon Tue Wed Thu Fri Sat</dow>
      </reportInterval>
      <metric ID="Metl">
        <description>Packet Loss</description>
        <metricUnit>%</metricUnit>
        <measureAt>Customer</measureAt>
        <lowerBoundValue>0.1</lowerBoundValue>
        <lowerBoundOperator>&lt;</lowerBoundOperator>
        <upperBoundValue>0</upperBoundValue>
        <upperBoundOperator>NA</upperBoundOperator>
        <dataCollectionSlot>1 minute</dataCollectionSlot>
      </metric>
    </SLO>
    [...]
    <Tariff>
      <T0 ID="T01">
        <chargingFuncFormula>
          (packets_sent + packets_received) * 0.02
        </chargingFuncFormula>
        <chargingFuncParam ID="Param1">
          <description>packets sent</description>
          <unit>packet</unit>
        </chargingFuncParam>
        <chargingFuncParam ID="Param2">
          <description>packets received</description>
          <unit>packet</unit>
        </chargingFuncParam>
        <dayTimeConstraint>
          <dow>Mon Tue Wed Thu Fri</dow>
          <startTime>00:00:00</startTime>
          <endTime>23:59:59</endTime>
        </dayTimeConstraint>
      </T0>
    </Tariff>
  </SLA>
</networkServiceSpecification>

Figure 5.9: Example of service specification in STS
A service as defined in this thesis is the composition of a service level agreement, electronic tariff and additional data. These high-level structures and their relationships can be specified in STS (SLA/Tariff Specification language - see Appendix A).

Figure 5.9 presents some parts of a service specification in STS. In this example, the service to be provided is Voice over IP (VoIP). The SLA part contains a few service level objectives (SLOs) but only the first is shown in the figure. SLO 1 states that packet loss must be less than 0.1% for a 1-minute sample, measured at the customer site. At every minute of the SLO validity period, a measurement report should be generated.

Additionally, the tariff has objectives (TO) that define different charging formulae. In this case, the first objective (TO1) states how much to charge for the service in a period of 24-hours, from Monday to Friday. The fictitious formula for the charging function is \((\text{packets}\_\text{sent} + \text{packets}\_\text{received}) \times 0.02\)’, where these parameters will be measured by the metering system. The value 0.02 corresponds to the unit price for transmitting a packet.

At some point, however, the SLA and tariff parts will have to be separated out in order to prepare for the translation process. Thus, the service partitioning stage conducts the partitioning of service specifications into two representations, one for the tariff and the other for the SLA structure.

Initially, a directed acyclic graph (DAG) is created representing a service with the associated SLA and Tariff. A DAG \(G\) is a triple \((V, E; f)\) where \(V \neq \emptyset\) and \(E\) are disjoint finite sets, and \(f : E \rightarrow 2^V\), is an incident map from \(E\) to the powerset of \(V\) such that \(1 < |f(e)| \leq 2\) for all \(e \in E\). Elements of \(V\) are called vertices or nodes and elements of \(E\) are called edges or branches. The map function \(f\) describes how edges in \(E\) connect vertices in \(V\). Therefore, the triple can be written as follows:

\[
V = \{v_1, v_2, ..., v_n\}
\]

\[
E = \{e_1, e_2, ..., e_n\}
\]

and \(f\) is the map \(e_1 \rightarrow \{v_m, v_n\}, ..., e_k \rightarrow \{v_i, v_j\}\)

The constraint \(|f(e)| = 2\) in the above DAG definition gives only regular edges. They are edges with different source and destination vertices. Additionally, with such a constraint, loop edges when \(|f(e)| = 1\) are not allowed.

Example: suppose two services, e.g. \(\text{service}_1\) (partially shown in Figure 5.9) and \(\text{service}_2\), are represented in STS.
Each service has a tariff and an SLA structure associated leading to the service DAG \( G = (V, E; f) \) where:

\[
V = \{ \text{service}_1, \text{tariff}_1, \text{sla}_1, \text{service}_2, \text{tariff}_2, \text{sla}_2, \} \\
E = \{ e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8 \}
\]

and \( f \) is the map described by: \( e_1 \rightarrow \{ \text{tariff}_1, \text{service}_1 \}, e_2 \rightarrow \{ \text{sla}_1, \text{service}_1 \}, e_3 \rightarrow \{ \text{tariff}_2, \text{service}_2 \}, e_4 \rightarrow \{ \text{sla}_2, \text{service}_2 \} \).

The graph \( G \) is shown in Figure 5.10. Each vertex has associated properties. One of them states whether the vertex must participate in the second stage of the translation mechanism. The SLA and Tariff type of vertices should normally be expanded from this point.

A **token** is a terminal symbol which represents a category such as keywords and literals. The sequence of characters that forms a single token is called a **lexeme**. During the last part of this first stage, a list of tokens is generated for each structure of interest in the service DAG, apart from the service vertices. Therefore, the vertices for SLA and Tariffs will point to a list of tokens. Such a list results from a lexical analysis which (a) removes white spaces and comments, and (b) recognises constants, identifiers and keywords.

The lexical analyser maintains a **symbol table**. This table helps the translation steps to be performed. Furthermore, operations to manage the table such as insertion, update and search have to be provided by the analyser.

Initially, a lexeme is inserted in the symbol table with the associated token. For instance, the lexeme \(<\text{SLA}>\) found in the source code, should be included in the symbol table as ('sla', keyword). This states that the lexeme 'sla' is a keyword. Thereafter, any token may be looked up in the symbol table by using a search operation.

### 5.4.2 Stage 2: Network Field Binding

As the first step of this stage, a network field DAG is created. This process involves a three step refinement of the service DAG. The first step creates a DAG called an **interdependency tariff-sla graph**, which represents the SLA and Tariff dependencies in terms of network field
templates. The second step builds another graph called an **interdependency network field graph**, which goes further and connects network flux templates and metric specifications to network field templates. Finally, the **interdependency event graph** connects events to network field templates.

**Example:** suppose now that the tariff structure of service\textsubscript{i} (Figure 5.10) requires measurements for two types of network fields, say HTTP and UDP fields, whereas its SLA structure needs measurements for a third network field (e.g. TCP). The graph \(G_1(V_1, E_1; f_1)\) that follows is the interdependency sla-tariff graph created, where:

\[ V_1 = \{\text{service}_1, \text{tariff}_1, \text{sla}_1, \text{networkfield}_1, \text{networkfield}_2, \text{networkfield}_3\} \]
\[ E_1 = \{e_1, e_2, e_3, e_4, e_5\} \]
and \(f_1\) is the map described by:

\[ e_1 \rightarrow \{\text{tariff}_1, \text{service}_1\}, e_2 \rightarrow \{\text{sla}_1, \text{service}_1\}, e_3 \rightarrow \{\text{networkfield}_1, \text{tariff}_1\}, e_4 \rightarrow \{\text{networkfield}_2, \text{tariff}_1\}, e_5 \rightarrow \{\text{networkfield}_3, \text{sla}_1\}. \]

![Figure 5.11: \(G_1\): Interdependency tariff-SLA graph](image)

Figure 5.11 shows the pictorial representation of graph \(G_1\). The **tariff\textsubscript{i}** has two linked network field templates: **networkfield\textsubscript{i}** that is the HTTP field and **networkfield\textsubscript{2}**, the UDP network field. In addition, the SLA **sla\textsubscript{i}** associates with **networkfield\textsubscript{3}** (TCP field).

**Example:** imagine that a network flux template, say **networkflux\textsubscript{1}**, specifies delay measurements in the metric specification **metricspec\textsubscript{1}** to be taken from the network field template **networkfield\textsubscript{i}**.

The **interdependency network field graph**, \(G_2(V_2, E_2; f_2)\) will be created when walking through the graph \(G_1\). \(G_2\) has the following properties:

\[ V_2 = \{\text{service}_1, \text{tariff}_1, \text{sla}_1, \text{networkfield}_1, \text{networkfield}_2, \text{networkfield}_3, \text{networkflux}_1, \text{metricspec}_1\} \]
\[ E_2 = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\} \]
and \(f_2\) is the map described by:
5.4. Design of Service Translator

$e_1 \rightarrow \{\text{tariff}_1, \text{service}_1\}, e_2 \rightarrow \{\text{sla}_1, \text{service}_1\}, e_3 \rightarrow \{\text{networkfield}_1, \text{tariff}_1\}, e_4 \rightarrow \{\text{networkfield}_2, \text{tariff}_1\}, e_5 \rightarrow \{\text{networkfield}_3, \text{sla}_1\},$

$e_6 \rightarrow \{\text{networkflux}_1, \text{networkfield}_1\}, e_7 \rightarrow \{\text{metricspec}_1, \text{networkflux}_1\}.$

Graph $G_2$, presented in Figure 5.12, expands $G_1$ up to network flux templates. As a result, structures such as metric specifications will be part of $G_2$.

Finally, the interdependency event graph $G_3(V_3, E_3; f_3)$ is generated based on the expansion of graph $G_2$. Graph $G_3$ is the last necessary step in order to perform the metering capability probing which queries the runtime system for available metrics.

![Figure 5.12: $G_2$: Interdependency network field graph](image)

**Example:** suppose that two events of any type of source need to be linked up (see Section 4.4.5 for a discussion of types of events). The first one is associated with the network flux template $\text{networkflux}_1$. The second one is associated with the metric $\text{metricspec}_1$.

$V_3 = \{\text{service}_1, \text{tariff}_1, \text{sla}_1, \text{networkfield}_1, \text{networkfield}_2, \text{networkfield}_3,$

$\text{networkflux}_1, \text{metricspec}_1, \text{event}_1, \text{event}_2\}$

$E_3 = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9\}$

and $f_3$ is the map described by:

$e_1 \rightarrow \{\text{tariff}_1, \text{service}_1\}, e_2 \rightarrow \{\text{sla}_1, \text{service}_1\}, e_3 \rightarrow \{\text{networkfield}_1, \text{tariff}_1\}, e_4 \rightarrow \{\text{networkfield}_2, \text{tariff}_1\}, e_5 \rightarrow \{\text{networkfield}_3, \text{sla}_1\},$

$e_6 \rightarrow \{\text{networkflux}_1, \text{networkfield}_1\}, e_7 \rightarrow \{\text{metricspec}_1, \text{networkflux}_1\}, e_8 \rightarrow \{\text{event}_1, \text{networkflux}_1\}, e_9 \rightarrow \{\text{event}_2, \text{metricspec}_1\}.$

Figure 5.13 shows the representation of graph $G_3$. 
As already stated, graph $G_3$ is the expansion of the nodes of graph $G_2$. Using the same logic, $G_2$ is the node expansion of graph $G_1$. Step 3 in the expansion chain should give the required metrics and events necessary for the next phase. However, more information is required in order to fully parse the source program. Such information can be formulated as a recursive expansion of the service graph until terminals are fully reached. Equations 5.11 and 5.12 formalise this recursion.

$$G_i \cup G_j = \{V_i \cup V_j, E_i \cup E_j\} \quad (5.11)$$

$$G_n = \bigcup_{i=1}^{n} (G_i \setminus G_{i-1}) \quad (5.12)$$

where $n$ is the number of graphs required for the translation.

The next step of this stage, the **metering capability probing**, aims to discover the metrics and resources available in the underlying runtime system (meter). The EdgeMeter system will provide a list of available resources to be used in a matching process. This process consists of finding whether the required metrics represented in graph $G_3$ are present in the system. In addition, metrics believed to be a substitute for a missing one will be linked in the corresponding vertex of $G_3$. The matching algorithm is based on the depth-first traversal, which recursively visits the children nodes in left-to-right order. It is described as follows:

1. Start from a metric $n$ (leaf)
2. Look up this metric in the list $m$ of available resources

3. If metric $n$ does not match $m.y$, then paint node $n$

4. Take the next metric and repeat from step 2 until all metrics are exhaustively verified

The metric nodes painted in graph $G_3$ indicate that they cannot be directly translated since the resource is not available in the runtime system. As a second process, there will be a negotiation which may substitute missing resources (modules) or even instruct the runtime system to fetch the missing parts from external servers (see Section 5.1 for a discussion on module loading).

Finally, parse trees for the SLA and Tariffs structures are produced. These trees are used during a process of verification to ascertain whether a string of tokens can be created by the EdgePol grammar. This step forms the syntax analysis of the presented translation mechanism. Top-down parsing, in which nodes of the parse tree are constructed starting from the root down to the leaves, is used in the syntax analysis. The generation of such trees takes place during the network field binding step. However, the expansion algorithm presented in 5.12 should be used in order to generate an entire parse tree.

As seen in Figure 5.8, the output of this stage is a list of structures. They are associated with each SLA or Tariff structures found in the source STS program. Syntax trees differ from parse trees as some minor irrelevant details for the translation do not appear in syntax trees. Thus, such trees when used as intermediate representations allow the translation to be separated from the parsing. This feature contributes to the modularity goal of the EdgePol translator.

5.4.3 Stage 3: Core Translation

The actual translation occurs using a technique called syntax-directed definitions. Such a scheme is composed of a grammar (STS grammar) and a set of semantic rules. The translation consists of a mapping function from the input $x$ to the output $y$. The function can be specified as the following steps [2]:

1. Take a syntax tree for $x$

2. Suppose a node $n$ in the syntax tree is labelled by the grammar symbol $z$. Thus, $z.a$ denotes the value of attribute $a$ of $z$ at that node.
5.4. Design of Service Translator

3. The value $z.a$ at $n$ is computed using the semantic rule for attribute $a$ associated with the $z$-production used at node $n$.

The result of this process is the translation into an intermediate representation which might be a three address code. Such intermediate representation is a sequence of the general form:

$$x := y \text{ op } z$$

where $x$, $y$, and $z$ are names, constants, or compiler-generated temporary structures; $\text{op}$ represents any operator such as a fixed or floating-point arithmetic operator. This representation, shown in Figure 5.8 with subscripts $\text{vm1 code}$, is suitable for SLA and Tariffs.

However, it seems more appropriate to represent the metering configuration in two other intermediate code representations. One should be in EdgePol, i.e. a policyset, to be used as the input for the policy engine. The choice of the other representation was to use an extension to the BPF+ virtual machine described in [14]. The latter is applicable to the filter configuration of the network reader discussed in Section 4.6. This is the reason why the metering configuration has two subscripts $\text{policyset}$ and $\text{vm2 code}$ (Figure 5.8).

5.4.4 Stage 4: Proof-carrying Code Checker

The Proof-carrying code (PCC) mechanism proposed by Necula and Lee [119] requires that the execution of a code follows well-defined safety policies. The consumer (the client's agent) publishes a formal set of security policies. The producer has to generate a proof that its code conforms to those policies.

If the optional PCC checker of the EdgePol translator is present, it will check whether the proof generated by the producer is valid. In case the proof attached to the source program can be verified, the intermediate code is signed as valid. Otherwise, the code is signed as invalid. Nevertheless, both types of intermediate code (valid or invalid) will go through to the remaining translation steps. The runtime system should take the decision of executing the target code or not. This will probably have to conform with the accepting policies in place in the runtime system.

5.4.5 Stage 5: Intermediate Code Optimisation

This stage applies well known optimisation techniques to the intermediate code representation. The idea is to introduce improvements in the execution time of an EdgePol program. Such improvements should happen as a result of the various transformations applied. In the BPF+
architecture [14], optimisation leads to an improved packet filter code. The transformations are applied to an intermediate code representation called the SSA form. This shows that optimisation at the intermediate code has a great impact on the running time of the target code. It is assumed that certain techniques can be used in this translator such as those proposed in [2] and applied in [14] to a packet filtering architecture.

5.4.6 Stage 6: Target Code Derivation (Metering configuration, Tariff, SLA)

In this final stage, the intermediate code is translated into native code. Final optimisations can still be done at this stage. The target environment for the metering configuration is the EdgeMeter interpreter. In contrast, the tariff and SLA structures are likely to be translated into native code for other target environments as well. Such environments would interact with the metering runtime system in order to get the data required by SLA and Tariffs.

5.5 Summary

The previous chapter described the EdgeMeter system architecture. This chapter detailed the design of data structures and algorithms of the main components of EdgeMeter.

First, the module manager component was discussed. Modules may be loaded from local or pre-defined external module servers, and are assigned to metering sessions at runtime. Session isolation issues were analysed so that 'cross-talk' activities between sessions cannot interfere with one another. Also, if the module cannot be located locally, the module discoverer element performs a lookup within a set of module servers. Such servers are located in the network as a point in a coordinate system. The implementation of the method to calculate a network location for a meter system, exported by the discoverer component's interface, is described. A scalable location framework, Lighthouse, is proposed and evaluated. It was shown that Lighthouse computes more accurate network locations, when compared to other related techniques. Finally, the design of the service translator, which is the translation mechanism of service specification code into a policyset was introduced.
Chapter 6

Metering Security

This chapter deals with the security issues identified in the scenarios of metering for charging and SLA validation in Chapter 3. The requirements for security are compiled, and a framework based on a trusted third-party model is introduced. This framework also introduces a set of protocols, tailored for each of these scenarios, that aim to fulfil the security requirements. Some of the key elements of our security framework were previously discussed in [139].

6.1 Introduction

Measuring a service between providers and customers raises security issues that need to be addressed. The problems may be aggravated by the terms of the charging policy agreed. When charging schemes that consider resource consumption, quality of service provision and correlated pricing policies are introduced to replace the current flat-fee pricing, there will be increased incentives for cheating. Security will be needed to prevent widespread fraud.

The first goal of our security framework is to prevent fraud committed by malicious customers, tampering with the meter system is one such case. The second goal is to protect customers from any damage that a meter installed in their systems might cause.

The trust model assumed in this chapter is as follows. A service provider (network, application-level and management) offers tailored services to customers and trusts them to pay for the charges incurred as part of the service provision. As part of the contract, the service provider may wish to measure the service at the customer premises for the reasons discussed in Chapter 3 (scalability is one of them). A customer, on the other hand, trusts its provider to deliver the agreed service and what is meant to be measured, as stated in the SLA and tariff, and to do so faithfully. The same customer may wish to have a meter installed to check its data
service provider’s charges.

The latter case, not very common today, opens up room for verification of what is referred to as a 'black box process' under the provider's total control. Billing and related measurements undertaken in the provider domain often creates disputes over hidden and erroneous charges at first unnoticed by customers. This is an unpleasant situation that could otherwise be independently verified by customers had they installed their own measurement systems.

There are unilateral risks that both providers and customers should take into account. If they agreed to install a meter system at the customer premises, the meter would ideally be independently supplied by a third-party that develops this type of software (meter manufacturer). In the majority of cases, however, the meter code itself will be distributed by the service provider.

This naturally prompts a question from a provider perspective: “how can the meter be protected from tampering?” For instance, the meter software may be modified to under-measure the traffic volume that is required by an installed service specification. Similarly, from a customer viewpoint, questions related to over-measurement and safety of the installed meter code are of great concern.

Figure 6.1, presented in Chapter 3 and replicated here, gives a general view of where these issues arise. As previously discussed, meters may be running in the provider domain (left side), customer domain (right side) and sometimes simultaneously in both domains. However, when the meter is running in customer equipment, it is expected that data will be passed across the boundaries between providers and customers. As a result, this data is “exposed” and should therefore be “sealed” for transit. In addition, there are issues related to the installation of a meter code in foreign environments (the customer system). To this effect, the sections that follow analyse what can go wrong when measuring services at the edges of the network, i.e.
either in the customer or provider domains.

6.2 Security Requirements

An analysis of the deployment of EdgeMeter in the context of the business model led to a list of security requirements. As for any system requirement, it is expected that they will be fulfilled by whoever implements the system.

In a competitive marketplace, say today's UK broadband market, providers face commercial disadvantages when cheating on their customers. For instance, a given provider can misbehave with a customer in the short-term, but not in the long-term, as customers will choose a competitor service provider. This is an example of structural security that may remove some of the security requirements.

The core requirements will be ranked as follows. \textit{Mandatory}: when the requirement must be fulfilled; and \textit{desirable}: the requirement may or may not be fulfilled. In this case, mandatory requirements correspond to the minimum security required in the deployment of EdgeMeter.

The security requirements were identified in two phases of EdgeMeter code deployment. First, the EdgeMeter system code is shipped out to the points of metering - these points may be a customer, a provider or both domains - during the initialisation phase. In the second phase (operation), the code is already installed and operational for metering the requested network services.

Table 6.1 presents the requirement classification. Some of them are also discussed for mobile agents in [42, 94, 16] and policy-based accounting [176].

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prevention of tampering</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Binding of measurement to context</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Execution of authentic code</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Deviation of control</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Mutual authentication</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Non-repudiation of measurements</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Denial of Service (DoS) prevention and recovery</td>
<td>Desirable</td>
</tr>
</tbody>
</table>

Table 6.1: Security Requirements

- \textit{Prevention of tampering}: service specification code (SLA/Tariff) disseminated by service providers and the measurement data generated are vulnerable to modification while
6.2. Security Requirements

locally stored or communicated over the network. The incentive to modify them is high since the former drives the meter system configuration and the latter is the result of a measurement session. In addition, the distribution of meter systems will often be via network media. Therefore, prevention of tampering with meter codes while they are being communicated is another key point to be considered. This requirement is mandatory.

- **Binding of measurement to context:** there will be many versions of a service specification code with a different range of tariffs and SLAs running at the same time in the meter system. The system must be able to bind the generated measurement data to the correct service specification version in an attempt to prevent data inconsistency. As the correct management of service specifications and related measurement data depends on this binding, this requirement is mandatory.

- **Execution of authentic code:** lack of integrity assurances in the host system of meter code is a major issue. The situation is aggravated if the code is to be executed in the customer system. A common threat is the installation of malicious code, which may compromise the host execution environment. Such code may also interfere with other ongoing measurements in the system by consuming excessive machine resources. This requirement is mandatory and may require safe protection implemented in software.

- **Deviation of control:** the service specification (high-level) is mapped to a metering configuration (low-level) by a validated translator component. Thus, there should exist links between these two entities. In this sense, the integrity of the metering configuration code and its associated service specification must be preserved to avoid any unauthorised modifications. Therefore, this requirement is mandatory.

- **Mutual authentication:** the principals (e.g. providers and customers) should know in advance who is at the other end of the communication before any data or control session begins. They must identify themselves using unique credentials. For instance, the service receptor element must authenticate the service distributor before authorising the reception of service specification code. This is a mandatory requirement.

- **Non-repudiation of measurements:** proving what another party claims to be the correct measurements is difficult. The customer wishes to be positive that what was used has been properly measured, perhaps within an acceptable margin of measurement error. The provider wishes to receive an accurate summary of its customer’s usage. To illustrate this case, suppose a customer agreed to take the measurements at his domain. At a later point,
the customer's meter system reports a summary of the measurement data to its provider. The provider, however, notifies this customer that no summary has been received recently. This prompts the question: "what actions should both customer and provider take to avoid such a situation?"

- **DoS prevention and recovery:** denial of service (DoS) attacks in a metering scenario can be regarded as a runtime issue that interferes with the normal operation and management of the meter system. As metering applications are responsible for accounting for resource usage, the meter system turns out to be a likely victim of such attacks, especially when billing is the reason behind the measurements. It might be the case that a customer benefits from a DoS attack unintentionally. This would lead to provider revenue losses, whatever the attack is. For instance, the customer is blocking the agreed report of usage summaries to its provider. On the other hand, the provider may benefit from a DoS attack. Just as with customers, this is often unintentional. Suppose the meter system at the customer side is intentionally packet flooded by an external attacker. The customer is charged based on volume communicated. Should this attack not be identified, the provider will generate a bill for this spurious traffic. In general, DoS is tricky to identify, let alone prevent. This requirement is desirable, which means it may or may not be fulfilled.

The next section introduces our security framework, which relies on a trusted third-party and some cryptographic protocols.

### 6.3 Security Framework

The deployment of EdgeMeter relies on a trust relationship between a provider (network, application-level and management) and its customer. Because the business relationship between them is in general established through a legal contract, it does simplify the form of trust required to address the security requirements identified and consequently minimises correlated risks.

To address the issues that arise from the partial trust between a provider and its customer, our security framework introduces a trustworthy third-party component, the Meter Inspection Authority (MIA). Third-party trust implies that both parties involved implicitly trust each other because they each share a relationship with a common third party, whose role in this case is to endorse the accuracy and safety of meter systems. The next section describes MIA along with
6.3. Security Framework

its certification services.

6.3.1 Meter Inspection Authority (MIA)

The MIA is best described by the services it offers to providers and customers. The services can be provided as single or compound certification to:

- Verify whether the meter system code is safe. This may be performed based on security proofs carried with the code [119].

- Check whether the meter is calibrated irrespective of the installation environment. On some occasions, it might be necessary to check whether the meter is producing results within a tolerable region of errors for heterogeneous environments, i.e. whether it complies with a meter specification.

- Check whether the code carrying a network service specification can be safely loaded in a meter system.

- Check whether the service translator element generates correct code. This service provides certification that the output of a service translator, metering configuration code, is consistent with the input (network specification code).

- Issue Meter Inspection Certificates (MIC) to indicate that any of the above inspections is positive. Two types of certificates are available: (a) Safety certificate and (b) Calibration certificate.

Table 6.2 proposes a format for MIA certificates. It is designed to be embedded in a X.509 certificate [89]. Most attributes are self-explanatory. However, a few of them deserve further clarification. Three types of target codes are allowed in the certificate: meter system, service specification and service translator. For each of them, a serial number is provided.

The meter manufacturer related attributes (ID and public key) are duplicate if this information is the same as it is in the provider attributes. This arises when the meter software is not outsourced, i.e. the provider itself develops it.
6.4 Applying the Framework

This section presents security protocols derived from the Needham-Schroeder authentication protocol [120], with added timestamps to address the security requirements. The notation shown in Table 6.3 will be used throughout this section to describe the protocols.

As part of key management, the distribution of keys is achieved with public-key encryption. It is assumed that services offered by Certificate Authorities (CA) will be available to bind entities to keys, as described in [61]. The steps required to assure that a given public key belongs to an entity are not explicitly shown in our protocols. They are regarded as the initial exchanged messages and are only infrequently required, as each of the parties involved can save the other’s key for future use.

The issue of key lifetime termination because of compromised keys leads to re-keying and certificate revoking. Although it is an important topic, standard solutions are well known for the re-keying process, and this will not be covered in this thesis.

The deployment process of Edgemeter involves two phases: initialisation and operation dealt with in the next two sections.
Table 6.3: Notation

6.4.1 Initialisation: Code Distribution

The EdgeMeter system is distributed and installed during the initialisation phase. This cycle comprises steps from the meter development (manufacturing) to its distribution to the intended points of metering (service provider, customer or both domains).

Figure 6.2 shows the three roles. We term them roles because different players may play different roles in different contexts. Providers as players may take the role of meter manufacturers, for instance.

The meter manufacturer implements the EdgeMeter architecture by following the design guidelines presented in Chapter 4 and 5. The resulting code, the EdgeMeter system, is shipped out to the provider (step 1) for eventual distribution to customers (step 2). Customers may also buy meter systems themselves from meter manufacturers for verification purposes.

Mutual authentication is mandatory over the interactions between these roles. The code communicated must be tamper-proofed. Finally, the code must be safe to be executed on the
customer system.

Trusting a service provider (network and application-level) not to load malicious code on the customer system is insufficient. The meter manufacturer (network management provider) may have written buggy code, for instance code that violates memory boundaries. There have been two fundamental approaches to prevent this. First, the language-based approach, which requires the code to be written in a safe language that constrains data type and memory access. Examples of language-based protection include Java [167], Safetynet [168], Caml [173] and more recently C# [162].

On the other hand, other schemes based on formal methods have been proposed to enforce security policies and at the same time allow code to be written in any programming language. Proof-carrying code (PCC) [119] provides a typing discipline for low-level code. The consumer (host system) publishes a formal set of safety policies. The producer (code developer) must generate proof that its code conforms to such policies. The consumer receives the code with an attached proof. If the proof is valid, the code is accepted.

PCC introduces some flexibility since the code can be written in any programming language, as long as there is a mechanism capable of generating a proof (theorem prover). However, PCC requires the producer to prove extensive conditions to indicate that the code conforms to the security policies specified by the host system. The theorem prover required for this is complex and difficult to implement for more elaborated predicates than the ones discussed in [119]. Besides, the proof size grows rapidly with the size of the code being proved safe.

Because PCC deployment is immature, the requirement for authentic and safe code can be fulfilled by a type-safe language such as Java. The Java Runtime Environment (JRE) includes a 'sandbox' providing portability and security policies that can be enforced at runtime.

Protocol Description (initialisation phase)

The scenario in Figure 6.3 presents a provider (P) buying a meter system from a meter manufacturer (MM). Eventually, this meter is installed at a customer domain. The figure and the accompanying table (Table 6.4) introduces the protocol for the initialisation phase.

Meter Manufacturer (MM) dispatches message 1 containing the EdgeMeter system to MIA. This message is signed by MM \(SIG_{MM}\) with an one-way hash function as a means for “MIA” to authenticate “MM”.
6.4. Applying the Framework

Figure 6.3: Initialisation phase (protocol steps)

Table 6.4: Security protocol: initialisation phase

(1) $MM \rightarrow MIA : \{\{\text{EdgeMeter code, version}, t_{1-MM}\} \text{SIG}_{MM}, ID_{MM}\} \text{KU}_{MIA}$

(2) $MIA \rightarrow MM : \{\text{sealed code, MIC}_1, MIC_2, t_{1-MIA}, ID_{MIA}\} \text{KU}_{MM}$

where: $\text{sealed code} = \{\text{EdgeMeter code, code ID}\} \text{SIG}_{MIA}$

(3) $MM \rightarrow P : \{\{\text{sealed code, MIC}_1, MIC_2, t_{2-MM}\} \text{SIG}_{MM}, ID_{MM}\} \text{KU}_P$

(4) $P \rightarrow C : \{\{\text{sealed code, MIC}_1, MIC_2, t_{2-MM}\} \text{SIG}_{MM}, ID_P\} \text{KU}_C$

Providers may assume that their competitors can capture the EdgeMeter code while in transit to analyse its functions. To avoid this happening, message 1 is encrypted with MIA’s public key ($\text{KU}_{MIA}$). The timestamp $t_{1-MM}$ informs MIA of the time this message was created. Distinct timestamp values will be used in the next steps for the purpose of message freshness.

MIA analyses the code upon receiving the message from MM. Then, it generates a certificate named MIC (Meter Inspection Certificate) with assurances. The certificate $MIC_1$ contains calibration and safety certification for the EdgeMeter system. $MIC_2$ does the same for the Service Translator element embedded in the EdgeMeter system. The Service Translator code not only has to be proven safe but must also contain assurance that its output (meter policyset) traces back to its input (service specification code). This is in fact the security requirement that any deviation of control must be promptly identified.

Also, MIA generates a form of meter seal by signing the code (EdgeMeter system) and this code’s unique identification (Code ID) with its signature ($\text{SIG}_{MIA}$), resulting in a sealed
6.4. Applying the Framework

code. The seal guarantees that the code will not be modified by anyone. By analogy, electricity meters are also sealed to prevent any customer tampering with them. MM sends a message containing the sealed EdgeMeter code and its code ID to P during the third step. A signature of MM ($SIG_{MM}$) is appended to the code, which guarantees its integrity. The message is encrypted using P’s public key ($K_{UP}$) to avoid eavesdropping.

During the last step (message 4), the customer (C) receives from P the code sealed by MIA. The certificates ($MIC1, MIC2$) give confidence to customers to trust the code as being safe. The sealed code is bound to its certificates by the Code ID information. Tamperproofing of this message is achieved because P signed it for authentication purposes and encrypted it with C’s public key ($K_{UC}$).

It is important to emphasise that the integrity and privacy of the EdgeMeter code to be shipped are also important issues to be addressed. How could it be guaranteed that the code, especially in a competitive market, was not modified or read by intruders? Thus, asymmetric cryptography to sign the messages and encrypt them is used throughout the protocol steps.

6.4.2 Operation: Measurements

![Figure 6.4: Operation phase](image)

At this stage, the EdgeMeter system should already be installed and operational to accept requests for measurements. This leads to the second phase of our system deployment.

Figure 6.4 introduces the operational cycle of EdgeMeter, identified along with the security
requirements that this dynamic tariffing/SLA scenario raises.

The service specification code must be protected from tampering of any kind while in transit from a provider's distributor to a customer's receptor. This code must also be safe to execute in the customer system. As such, the decision was to write the service specification code in a type-safe language [167] [162].

Though much of the protection checking is done statically at compile time by verifying whether or not the code circumvents the type system of the language, there is important checking to be performed at runtime. The recipient of this code should have a 'sandbox' environment to verify code loaded at runtime, for instance to rule out any possibility of access to resources not explicitly authorised (file system and so forth).

The use of a sandbox approach introduces some overhead that is worth mentioning. The code is likely to run substantially slower than code that executes without further checks, for instance code that is verified before it is converted into native code by a just-in-time compiler, such as the one used in the Common Language Runtime (CLR) [162]. However, the startup costs of the service specification code will be lower than those of code that must be verified before it is translated.

The service specification (high-level) is mapped to a metering configuration (low-level) by a translator component, as shown in the customer side (right hand side of Figure 6.4). There are some issues associated with this translation. First, in what circumstances may the low-level code be considered equivalent to the high-level one?

Suppose that the service specification code requires coarse-grained volume measurement at the unit of bytes. Furthermore, the electronic tariff of this service uses the measurement data to compute at runtime a customer's usage-based charge. However, for some reason, be it a buggy or a tampered translator, there was a deviation on the low-level code that led to the meter system being instructed to report volume in packets as opposed to bytes. The customer will be undercharged and its provider loses income. This episode illustrates the importance of the deviation of control issue as described in the previous section.

To address the above issue, providers and customers trust the MIA to undertake conformance tests on the service translator code during the initialisation phase. If the translator conforms to the specification, MIA issues a certificate validating the code. The software engineering mechanisms for checking the translator conformance are outside the scope of this thesis.
A naive approach is to have test sets of matching pairs of service specification code and policyset. By applying all of them and counting the failures, one may conclude whether or not the code has an acceptable behaviour. The translation may be considered flawed for unknown reasons. It may be a bug in the software or someone may just have altered it. To test whether or not the translator introduces any deviation, the service specification is input and the translated code (policyset) compared against the policyset paired with the input code. If they differ, the translator fails in one of the tests.

Another related issue arises when the service specification code (high-level) is not explicitly linked to its translated form, the policyset (low-level metering configuration). Had the service translator generated a matching input and output pair, the output code (policyset) would still be vulnerable to modification. It is important, therefore, to preserve the integrity of the policyset by linking these two codes up just after the translation, perhaps inside the service translator itself. This issue will be dealt with in the next section.

**Protocol Description (operation phase)**

![Protocol Diagram](image)

Figure 6.5: Operation phase (protocol steps)

This section presents a protocol to secure the operation phase as pictured in Figure 6.4. Some of the issues have been discussed in the previous section; the remaining will be addressed in this section.

Each of the protocol steps shown in Figure 6.5 addresses a particular issue. The type of asymmetric cryptographic transforms applied to every protocol step is presented in Table 6.5.

The service provide announces its services to customers, either using multicast or unicast. The service distributor (SD) component signs the service specification code (tamperproof code)
before transmitting it to the service receptor (SR) in the first step.

As the service provider will publicise its service and tariffs in the usual media channels (e.g. newspaper and TV), how the service terms are described in a service specification code will be no secret. Thus, message 1 is sent 'in clear' from SD to SR.

The MIA certified service translator (CST) converts service specification code to meter policyset. As the requirement of avoiding deviation of control is mandatory, there must exist a mechanism to bind these two codes together. To address this issue, the CST signs (step 2) the policyset code together with SD's signature applied to service specification and its version in step 1. This results in the binding of policyset and service specification: \( \{ \text{policyset, SIG(signed service specification)} \}_\text{SIG}_{\text{CST}} \)

where \( \text{SIG(signed service specification)} \) is the signature of SD that was appended to message \( \{ \text{Service Specification, version} \} \) (step 1). As the assumption is based on local communication within EdgeMeter, message 2 does not need to be encrypted for privacy purposes.

The third step corresponds to the interaction with a database, Measurement Storage (MS), where metering data will be deposited. This database holds data classified by the service specification and policyset. The idea is that once the metering data is generated, it cannot be modified but only accessed. To address this, the metering data is signed by an internal meter module, shown as 'EM' for EdgeMeter in Figure 6.5. Also, this signature binds the metering data to the EdgeMeter code ID and the service specification/policyset used to instruct the EdgeMeter system.

However, other components must gain read access to the metering data. Message 3, therefore, is encrypted using the group key \( (KG_{BS,SM,EM}) \). This enables a provider and its customer to read the signed metering data as the group key is shared by the billing system (BS) component, the SLA maintainer (SM) component and the EdgeMeter itself (EM). These keys
will be exchanged as part of the service setup between providers and customers.

The EdgeMeter system starts to push out data records to the billing system (BS) and SLA maintainer (SM) roughly at the same time. To accomplish this, message 4 is encrypted using the group key KG shared by BS, SM and EM ($K_{G_{MP,MC}}$). The metering data is signed by an internal EdgeMeter component (tamperproof).

Providers may wish to 'turn on' their own meter for auditing purposes, should they not be receiving signed metering data for reasons such as deletion of data from the measurement storage (MS), or network problems (congestion, traffic blocking). However, the arbitration process when there are discrepancies in metering data may need to be done through legal channels in some extreme situations.

6.5 Summary

This chapter has discussed the EdgeMeter security framework. An analysis of the deployment of EdgeMeter in a charging context led to a list of security requirements. Six of them were ranked as mandatory, meaning that they must be fulfilled.

It was argued that a trustworthy third-party component, the Meter Inspection Authority (MIA), along with cryptographic protocols are required to address these issues. MIA offers certification services to endorse accuracy and safety of metering systems prior to their installation on host machines. Once certification of this code has been done, MIA binds two certificates to it.

The requirement prevention of tampering of code has been tackled by binding the EdgeMeter code to the MIA digital signature (meter seal). As another example, the EdgeMeter system is capable of binding metering data to the SLA/Tariff code (service specification) and the EdgeMeter code version used. This has addressed another important requirement, binding metering data to context.

The chapter concluded that the cryptographic protocol steps established between customers, providers and MIAs are sufficient to address the security requirements.
Chapter 7

Experimental Work

7.1 Charging and SLA management

Business roles have been discussed in Chapter 1 along with a canonical business model applied to a multi-service network. Figure 7.1 shows part of the model that only includes providers selling services directly to end-users or corporations. The latter are customers connected at the edges of the network as discussed in section 3.3.1.

The network retailer offers access services (e.g. broadband, corporate leased lines) to customers; whereas application-level providers sell services at the application layer to be transported as part of the service a customer contracted with a network retailer. Application-level services include video/audio streaming and voice over IP - to mention a few.

The vertical dashed line of the figure represents the business interface between providers and customers through a contract. Eventually, such a legal document will be translated to a list of technical requirements that among others include necessary network measurements (refer to Section 3.3.3 for a discussion on service specification (SLA/Tariff)).

![Figure 7.1: Business Context](image-url)
Within this context, metering is vital to the deployment of SLA validation and charging components. This naturally prompts the questions: how does a metering infrastructure fare on providing an accurate metering service? What are the limitations of a metering service?

### 7.2 Metering Performance in a DiffServ Scenario

To find these limitations, we present the results of experimental work carried out using a Differentiated Service (DiffServ) [18] based scenario that uses SLAs to indicate the required performance of the network and electronic tariffs to charge for the service delivered. The DiffServ framework and some of its extensions proposals, discussed in Chapter 3, are likely to be the QoS framework deployed in large scale on the Internet. Evidence from router vendors in responding to a rising demand of service providers for a QoS mechanism has already shown this tendency.

The specific question to be addressed in the first part of the experimental work relates to the flexibility of meter placement. The measurement region for this experiment is defined by two metering points, either the provider or customer domain. We shall first define accuracy before formulating the question of interest. Accuracy is the degree to which a given quantity is correct and free from error. For instance, a quantity specified as 80 ± 5% has a relative accuracy of ±5%, meaning its true value can fall in the range of 76-84. Now, the question addressed is:

- **What is the accuracy achieved and what are its effects in a business context, when metering Internet services at the (a) provider domain and (b) customer domain?**

Consequently, it is clear how relevant it is to understand whether there are any errors related to the metering process in general and whether this invalidates the approach. When money is involved, it makes this process even more important.

Service providers should consider metering errors when validating SLAs. Suppose that an SLA for voice over IP service has one of its service level objectives (SLO) stating that packet delay above 75ms gives the customer a rebate of 5 pence should this objective not be met 5 times in 10 minutes.

Taking the above example further, if the metering service provides measures with an accuracy of ±10%, there is a range in which the measures may or may not be considered an SLA violation. For instance, if the value measured is 83.32ms, deducting 10% from it gives 74.98ms which would not constitute a violation. On the other hand, adding 10% to the measured value
would clearly raise a violation (90.55ms).

7.2.1 Services Available

In this hypothetical scenario, three categories of service were offered to customers by application-level providers in association with network retailers. Each service offered different quality guarantees and different prices were applied to each one. The service categories are fully described in [3]:

- **Interactive realtime:** this is a service offered when both ends interact with each other (two-way communication). Because of its realtime nature, it is very sensitive to any variation in the quality of service provided by the network. It requires quality guarantees, for instance, in terms of packet delay, jitter and loss. Examples of this category include voice over IP (VoIP) services.

- **Non-interactive realtime:** this is also a realtime service but has the characteristic of one-way communication, usually from a server (provider) towards a customer. The decision to have a new service category and not to use the category described above lies in the fact that its required metrics and respective thresholds differ from the ones used for interactive realtime services. A video/audio streaming service is classified in this category.

- **Non-realtime:** services in this category do not usually require QoS guarantees, although some may have special requirements in a few cases. An E-mail service is a typical example. There are customers willing to pay more in order to have emails delivered with time guarantees. Some form of QoS guarantee will be necessary (e.g. low packet delay and loss) to address this.

Within each service category above, two levels of QoS were defined. The *gold* level provides for higher level of quality guarantees within a service category, whereas the *silver* level offers a lower level of guarantees. Therefore, there were three service categories, each with two levels of QoS. This led to six independent levels to be guaranteed by the DiffServ mechanism. Each QoS level was termed a service class.

The next step was mapping the service classes to equivalent DiffServ services. Since there is no consensus yet on what a DiffServ service might be, the mapping was based entirely on finding a DiffServ class that best describes the QoS requirements of each service class.
7.2. Metering Performance in a DiffServ Scenario

<table>
<thead>
<tr>
<th>Service</th>
<th>QoS Level</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactive realtime</td>
<td>EF</td>
<td>AF₁₁</td>
</tr>
<tr>
<td>Non-interactive realtime</td>
<td>AF₁₂</td>
<td></td>
</tr>
<tr>
<td>Non-realtime</td>
<td>AF₁₃</td>
<td>AF₂₂</td>
</tr>
</tbody>
</table>

Table 7.1: Mapping to DiffServ classes

In the DiffServ framework, packets are aggregated in a small number of classes. Each DiffServ class is identified by a number known as the DiffServ Code Point (DSCP) and it has an associated per-hop behaviour (PHB) specifying how packets of this class should be treated in each network element (hop). The DSCP value is embedded in the Type of Service (TOS) field of the IPv4 header. Two PHBs are currently specified and they were extensively used here for the mapping from each of the six service classes to an equivalent DiffServ class.

The Expedited Forward (EF) PHB gives a low loss, low delay and low jitter service by ensuring that the service rate of EF packets on a given output interface exceeds their arrival rate at that interface over long and short time intervals [68]. To represent the service class (interactive rt, gold), the EF PHB was chosen on the basis that it provides the highest DiffServ quality guarantees.

The remainder of the service classes were mapped by using the Assured Forwarding (AF) PHB, which supports the delivery of packets in four forwarded DiffServ classes, which represent distinct levels of QoS. Within each AF class, an IP packet is assigned one of three levels of drop precedence: low, medium and high. The higher the drop precedence, the higher the discard probability should congestion in a router arise. An IP packet that belongs to an AF class \(i\) and has drop precedence \(j\) is marked with the AF codepoint \(AF_{ij}\), where \(1 \leq i \leq 4\) and \(1 \leq j \leq 3\) [87].

Table 7.1 shows the six service classes and their equivalent DiffServ mapped classes. Three levels of drop precedence (\(AF_{11}, AF_{12}\) and \(AF_{13}\)) were used for AF class 1, whilst two levels of drop precedence (\(AF_{21}\) and \(AF_{22}\)) were used for AF class 2. Should the order of priority among these classes be required, it will start with the (interactive rt, gold) service class and finish with the (non-realtime, silver) class.

7.2.2 Experimental Testbed

The experiments were carried out on a DiffServ testbed (outlined in Figure 7.2) at BTExact Labs (Ipswich, UK). Table 2 shows the specifications of each machine, all with Intel Pentium
processors. The same meter machine was used for measuring the customer and provider traffic.

Each router had the ALTQ (Alternate Queueing) software installed [43]. ALTQ implements the DiffServ traffic conditioners (e.g. packet classifier, marker and shaper) and the two defined PHBs, AF and EF. These are elements required to implement the service classes as described in the previous section.

As this experimental work was carried out in the early stages of this project, a prototype implementation of EdgeMeter was not yet available at that time.

To serve as the meter system, we used NeTraMet [28] which is a flow-based measurement software that implements the IETF RealTime Flow Measurement architecture (RTFM) \(^1\) [30]. NeTraMet is maintained by the University of Auckland \(^2\) and has been extensively used as the underlying measurement platform in various charging and QoS metering projects. The CADE-NUS [53] project used the RTFM as the monitoring system for the issue of SLA management. With focus on charging issues, projects such as M3I (A Market Managed Multiservice Internet) [25], CATI [160], GenCA [39] have also used RTFM as a feasible measurement infrastructure.

<table>
<thead>
<tr>
<th>Machine</th>
<th>OS</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provider</td>
<td>Linux 2.4</td>
<td>Pentium II 333MHz, 128 MBytes of RAM, 100Mb/s 3COM 3c59x NIC</td>
</tr>
<tr>
<td>RT1</td>
<td>FreeBSD 3.2</td>
<td>Pentium II 400 MHz, 128MBytes of RAM, 100Mb/s Intel Pro 100 NIC</td>
</tr>
<tr>
<td>RT2</td>
<td>FreeBSD 3.2</td>
<td>Pentium II 333MHz, 64MBytes of RAM, 100Mb/s Intel Pro 100 NIC</td>
</tr>
<tr>
<td>RT3</td>
<td>FreeBSD 3.2</td>
<td>Pentium II 266MHz, 64MBytes of RAM, 100Mb/s Intel Pro 100 NIC</td>
</tr>
<tr>
<td>Customer</td>
<td>Linux 2.4</td>
<td>Pentium II 266 MHz, 128 MBytes of RAM, 10Mb/s NIC</td>
</tr>
<tr>
<td>Meter (M)</td>
<td>FreeBSD 4.1</td>
<td>Pentium 133MHz, 24MBytes of RAM, 10Mb/s AMD PCNet/PCI NIC</td>
</tr>
</tbody>
</table>

Table 7.2: Machines specifications

NeTraMet passively records flow-based data in tables allocated in primary memory by capturing the traffic of the Ethernet segment to which it is connected. Another application,

\(^1\) The website http://www2.auckland.ac.nz/net/Internet/rtfm/ contains all information related to the RTFM architecture.

\(^2\) NeTraMet is freely available for most Unix-based systems at http://www2.auckland.ac.nz/net/NeTraMet. A Windows version of NeTraMet was developed within the context of this thesis and the work the author carried out in the M3I project (Market Managed Multiservice Internet) http://www.m3i.org
NeMaC, registers with NeTraMet to periodically collect flow data via SNMP, which is then flushed to log files on disk for analysis. NeTraMet relies on the BPF packet filtering architecture as its mechanism for packet capturing [106].

### 7.2.3 Methodology

As NeTraMet is a flow-based meter, six independent UDP flows were specified, one for each DiffServ class in Table 7.1, by adequately setting flow attributes in the IP and UDP headers of a packet. Each UDP flow was created by using the MGEN traffic generator [128]. To distinguish each flow, MGEN set the destination transport port number of each flow in the range of 8001-8006 with the class (interactive rt, gold) taking the first port number and so forth.

The packet size used in the experiments was 462 bytes. The breakdown of this size leads to 20 bytes (IPv4 header), 8 bytes (UDP header) and finally 434 bytes for the UDP payload.

<table>
<thead>
<tr>
<th>Service class</th>
<th>Generated (pkt/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1: Interactive rt, gold (EF)</td>
<td>750</td>
</tr>
<tr>
<td>Group 1: Non-interactive rt, gold (AF11)</td>
<td>500</td>
</tr>
<tr>
<td>Group 1: Non-interactive rt, silver (AF12)</td>
<td>375</td>
</tr>
<tr>
<td>Group 2: Non-realtime, gold (AF13)</td>
<td>375</td>
</tr>
<tr>
<td>Group 2: Interactive rt, silver (AF21)</td>
<td>250</td>
</tr>
<tr>
<td>Group 2: Non-realtime, silver (AF22)</td>
<td>250</td>
</tr>
</tbody>
</table>

Table 7.3: Packet rates generated

The traffic pattern of these flows followed a Poisson function with average rate determined by the packet rate as indicated in Table 7.3. For instance, the average packet rate specified for the (interactive realtime, gold) class was 750 pkt/s (packets per second).

As the general purpose was to find out levels of accuracy while metering at different points of the testbed configuration, two experimental setups were designed by varying the meter position to both the provider and customer sides. Although Figure 7.2 might give the impression that two meters were running simultaneously, in fact only one meter at a time was positioned in either the provider domain (provider setup) or the customer domain (customer setup).

For each experimental setup, two groups of three UDP flows in each were generated from the provider server towards the customer machine, traversing the DiffServ region; the end-to-end path taken by these flows was {provider, RT1, RT2, RT3, customer} (Figure 7.2). Each flow individually lasted for 300 seconds.

---

3 A flow is regarded as a 5-tuple (source IP address, destination IP address, source transport port, destination transport port and protocol ID). An example of a flow specification is (10.0.0.1, 10.0.0.2, 0, 80, UDP).
The first group of flows launched corresponded to the (interactive rt, gold), (non-interactive rt, gold) and (non-interactive rt, silver) classes, requiring an aggregated bandwidth of approximately \((750 + 500 + 375) \times 462 \times 8 \approx 6 \text{ Mb/s}\).

In the sequence, the second group included the (non-rt, gold), (interactive rt, silver) and (non-rt, silver) service classes. They used an aggregated bandwidth of \((375 + 250 + 250) \times 462 \times 8 \approx 3.2 \text{ Mb/s}\).

### 7.2.4 Measured Packet Rate

This section analyses measured packet rates by comparing them to the generated (expected) rates in a DiffServ business scenario.

The relative error is used for metering accuracy. Let the generated value of a metric be \(x\) and the measured value \(x_0\). The metering relative error \(\delta x\) is then defined by:

\[
\delta x = \left| \frac{x - x_0}{x} \right| = \left| 1 - \frac{x_0}{x} \right|
\]

Now, the accuracy of a measured value is defined in terms of a relative error within a closed interval \([+\delta x, -\delta x]\) in which the true value of a measured quantity can fall. A relative metering error of 0.02 (i.e. 2\%) means that accuracy has to be ± 2\%.

Figure 7.3 shows separate graphs of packet rates generated and measured for each service class described in Table 7.1. Two of the curves in each plot show the instantaneous packet rates generated (Poisson pattern) and their mean value over the time interval of 300 seconds. The remainder of the curves present packet rates measured at the provider and customer sites. Figure 7.4 shows the respective relative error graphs for each service class.

The UDP flows for the three service classes of Figures 7.3(a), 7.3(b) and 7.3(c) (first group) were launched at the same time, each flow lasting for 300 seconds; similarly, three other UDP flows for the service classes of Figures 7.3(d), 7.3(e) and 7.3(f) (second group) were launched after the completion of the first group.

In general, as can be seen from these graphs, there was a gap between the generated and measured packet rate curves, wherever the meter was placed (provider or customer). In particular, this absolute error is accentuated in Figures 7.3(a), 7.3(b) and 7.3(c). These results suggest that inaccuracy was present in the metering process. Figures 7.4(a), 7.4(b) 7.4(c) present the
7.2. Metering Performance in a DiffServ Scenario

relative errors for these service classes, respectively.

In Figures 7.4(a), 7.4(b), 7.4(c), the relative error curves start with their maximum values, rapidly decreasing to what was regarded as a mean value from the 80th second. This pattern is related to the start behaviour of Figures 7.3(a), 7.3(b) and 7.3(c), where the measured rate began very low, then rapidly increased to eventually stabilise within the interval [80, 300].

Although the relative error curves are not equal, they do have similar shapes, which suggest that relative errors occurred consistently, irrespective of where the measurement was taken. For instance, a decrease in the error value (customer side) was most likely to be related to a decrease in the error value associated with the provider measurements (see Figure 7.4(a) for an example). These results can be explained by observing Figure 7.3(a). The measured curves were nearly constant between [80, 300] seconds, while the generated packet rate curve did vary around their mean value in the same time interval.

The best accuracy achieved occurred with the (non-realtime, gold) service class measured at the provider site (Figure 7.4(d)). The mean relative error was 2.2% with a standard deviation of 1.5%. The measurement accuracy at customer for this class improved after the 80th second with a mean error of 8.48%, reaching a minimum value of 2.1% in the interval [224, 250] seconds. The standard deviation in this case was 9.2% due to a maximum value of 51.82%.

In Figures 7.4(d), 7.4(e), 7.4(f), the measurement error at the provider does not show the same level of variation as the error for customer measurements. Apart from the (non-realtime, silver) class (Figure 7.4(f)), the other two service classes presented a starting error below 10% (interval [0, 80]).

Table 7.4 contains some statistics of the relative error curves shown in Figure 7.4. It is important to note that measurement at the provider site for the classes (non-realtime, gold), (interactive rt, silver) and (non-realtime, silver) presented the lowest relative errors among all the measurements taken.

At the other extreme, measurement at the provider site for the classes (interactive rt, gold), (non-interactive rt, gold) and (non-interactive rt, silver) presented the highest relative error, in particular the class (non-interactive rt, silver) with a mean error of 14.96%.

The error associated with measurements at the customer presented high mean variation, which may suggest that this result cannot be interpreted as the mean for measurements at the customer if considered between the interval [0, 80], i.e. if not discarding the first part of the
7.2. Metering Performance in a DiffServ Scenario

time interval.

Figure 7.3: Comparison of measured against generated packet rate (absolute error)
7.2. Metering Performance in a DiffServ Scenario

Figure 7.4: Relative error curves
7.2. Metering Performance in a DiffServ Scenario

<table>
<thead>
<tr>
<th>Service class</th>
<th>Meas. Provider(%)</th>
<th>Meas. Customer(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactive rt, gold (EF)</td>
<td>12.03</td>
<td>8</td>
</tr>
<tr>
<td>Non-interactive rt, gold (AF11)</td>
<td>12.38</td>
<td>9.17</td>
</tr>
<tr>
<td>Non-interactive rt, silver (AF12)</td>
<td>14.96</td>
<td>9.88</td>
</tr>
<tr>
<td>Non-realtime, gold (AF13)</td>
<td>2.2</td>
<td>8.48</td>
</tr>
<tr>
<td>Interactive rt, silver (AF21)</td>
<td>3</td>
<td>10.61</td>
</tr>
<tr>
<td>Non-realtime, silver (AF22)</td>
<td>5</td>
<td>12.66</td>
</tr>
</tbody>
</table>

Table 7.4: Relative error statistics

7.2.5 Discussion

The results of Figure 7.3 show that measurements under-estimated true traffic. A discussion regarding what might have caused the metering errors in this experimental work is presented in this section:

1. **Network loss**: the assumption in the experiment was that the customer did receive all the traffic generated and that any error would be due to the measurement process itself and not to any packet loss occurring in the network. The DiffServ domain could be regarded as one source of error as packets could have been dropped within the network, for instance when the rate generated was higher than that agreed. However, it was confirmed that no losses occurred in the DiffServ routers and all packet rates for the DiffServ classes used were within the agreed rates. This leads to the conclusion that the customer machine did receive all packets generated by the provider server verifying the assumption, therefore ruling out the network as a possible cause of packet loss in this case.

2. **Operating system and meter code**: the packet capture process is responsible for getting parts of the packet (e.g. headers) and transferring them to the meter process in user space. The FreeBSD 4.1 system, in which the meter code (NeTraMet) was running, could not properly capture packets of the aggregate rates of the two groups of applications launched. The first group generated 1575 pkt/s, whilst the second group generated 1125 pkt/s. Even though the 10 Mb/s NIC used has a theoretical maximum packet rate of about 14880 pkt/s, packets were dropped in kernel space before being transferred to the meter process (user space). The reason was the lack of resources due to the high rate of NIC interrupts imposed on the system.

The measured rates rapidly increased at the beginning of the time interval (specially within the interval [0, 80]) which then became less visible from the 80th second onwards. These high startup error rates can be observed in all six graphs for the measured curves (Figure 7.3) apart
from the three curves of measurements at the provider in the last three graphs. The log files indicate that few resources were available to handle the Ethernet interface and packets were consequently discarded due to interrupt processing cost. This was particularly noticeable in the first 90 seconds suggesting that errors were more frequent at the startup.

From these results, it is clear that metering error is inevitable. The effects of the measurements taken in this experimental work on the maintenance of DiffServ SLAs and tariff clauses can be drastic if metering accuracy is not carefully considered. As an example, if we assume that the cost of the service class (interactive rt, gold) is 0.30 pence per packet then the errors on a VoIP service requiring 750 pkt/s, might amount to a revenue loss of £102/hour out of £810/hour. This error rate seems acceptable but mechanisms can be used to compensate for these errors.

The next section analyses the packet capture process of a prototype implementation of EdgeMeter. This is regarded as a key component of a metering system and possibly the reason for metering errors.

### 7.3 Packet Capture Evaluation

As briefly described in the previous section, possible causes of metering errors include the overhead on the system due to interrupt processing, while transferring packets from the NIC to the meter process in user space (NeTraMet).

This section has the objective of assessing the packet capture process of a metering system with further experimental work. A prototype implementation of the EdgeMeter architecture has been used to carry out experiments under a variety of controlled loads. The description of the results are followed by a discussion of findings and possible optimisations of the capture process.

#### 7.3.1 EdgeMeter Prototype Implementation

The *Network Reader* component was chosen to be implemented because it is responsible for receiving incoming packets from the network, selecting parts of the packet (e.g. headers) and delivering them to modules plugged in the metering system, which is running in user space. The network reader implementation will be described in this section as part of the EdgeMeter prototype implementation. The prototype has been developed over the MIT Click Modular Router [99]. The Click code used was the one available for FreeBSD systems.

---

4Software, including kernel patches, are freely available at http://www.pdos.lcs.mit.edu/click/
Click is a software architecture for building flexible and configurable routers from fine-grained components that represent a unit of router processing. Components, or Click elements, can be easily connected through a directed graph configuration. The graph’s edges, which are called connections, represent possible paths that packets can take within the router. An element represents a conceptually simple computation, such as extracting an IPv4 header field, rather than a large, complex computation, such as IP routing. This flexibility allows the creation of tailored packet forwarding systems by connecting basic and fine-grained elements.

The most important properties of a Click element are:

- **Element class**: each element belongs to one element class. This includes the code that should be executed when the element processes a packet.

- **Ports**: elements can have any number of input or output ports. A connection in the configuration graph is made from an output port on one element to an input port on another.

- **Method interfaces**: each element supports one or more method interfaces. Every element supports the simple packet-transfer interface, but elements can create and export arbitrary additional interfaces.

EdgeMeter was implemented in C++ as a Click element. Each element derives from the class `Element`, which has around 20 virtual functions. `Element` provides the default implementation for most of them, so the EdgeMeter element has overridden six of them, in particular the ones used during the operation of receiving and sending packets.

Figure 7.5 gives an overview of the EdgeMeter prototype implementation. Packets are read from the network interface `xlo` through the `FromDevice()` element. Its implementation uses the BPF architecture, thus allowing a similar experimental context to the one presented in the first part of this chapter with NeTraMet. The Ethernet header is removed from the packet (Strip()) and IPv4 header fields are checked regarding version, length and checksum. The IP addresses are then captured from the IP header.

The policy engine element keeps a table of measurements with packet and byte counters in memory. This table corresponds to a simplified version of the network field and flux hash tables discussed in Section 5.1.3.
7.3. Packet Capture Evaluation

7.3.2 Methodology and testbed

The testbed consisted of just a sender (Linux 2.4.9) and receiver (FreeBSD 4.2) systems, as shown in Figure 7.6. Both machines were Intel Pentium III 863 MHz, 256 MBytes of RAM, connected through a 100 Mb/s Ethernet crossover cable. The NIC card of the sender was an Intel EtherExpress Pro100, whilst the receiver had installed a 3Com 3c905C-TX Fast Etherlink. The EdgeMeter code was installed in the receiver machine in order to capture the receiving traffic.

The sender host generated UDP packets at specified rates towards the receiver. Minimum packet size has been used since it can stress the system producing the worst case scenario for capturing packets. Each 64-byte UDP packet included Ethernet, IP and UDP headers as well as 14 bytes of data and the 4-byte Ethernet CRC. When the 8-bytes preamble and 12-bytes of inter-frame gap are added, a 100 Mbit/s Ethernet segment can theoretically transmit up to 148.8k pkt/s. Each packet rate was generated for 60 seconds and repeated five times. The first and last 10 seconds were discarded leading to 40 seconds worth of data in each experimental run.

Intel Pentium III performance counters [52] were set to precisely measure CPU cycles in particular parts of the packet path within EdgeMeter. The counter CPU_CLK_UNHALTED provides the number of cycles spent in a specific operation, distinguishing between the ones spent in user-level and kernel-level. These counters were read through the RDPMC (Read
Performance-Monitoring Counter) instruction.

7.3.3 Results and Analysis

The results presented in Section 7.2 suggest that metering error is inevitable. Figure 7.7 sheds light on the issue of when metering errors may begin to occur. Packet rates generated and measured are shown on both graphs. The terms 'captured' and 'measured' rates have the same meaning in this context and will be used interchangeably; this also applies to the terms 'generated' and 'arrival rate'. Generated rate relates to the sender, while 'arrival rate' relates to the receiver. As there is no packet loss between these two in this experiment, the generated rate is the same as the arrival rate.

![Graphs showing comparison of generated and measured packet rates](image)

Figure 7.7: Comparison of measured against generated packet rate

The maximum packet rate generated was 48k pkt/s, with each packet being 64-bytes in size to guarantee maximum stress on the system. The measured packet rate was identical to that generated up to 30k pkt/s, when these two curves began to differ (Figure 7.7(a)). This point is called Maximum Error-free Metering Rate (MEFMR).

The meter suffered from poor overload behaviour at rates above 30k pkt/s when the measured rate began to decrease while the rate generated increased. This resulted in errors as shown in Figure 7.7(b). In the extreme case, packet losses were about 46% of the packets generated for rates of 48k pkt/s.

The onset of metering error started at 30k pkt/s (Figure 7.8(a)). Over a 60-second time interval, the relative error for this rate oscillated within 1% (Figure 7.8(b)).
7.3. Packet Capture Evaluation

![Graph showing packet capture evaluation](image)

Figure 7.8: Comparison of measured against generated for rates of 30k pkt/s

7.3.4 Source of Error

![BPF architecture diagram](image)

Figure 7.9: BPF architecture (packet path) at receiver machine

The EdgeMeter prototype implementation used the BPF system [106] available in the FreeBSD 4.2 kernel as its packet capture engine (FromDevice() implementation). Although implementations of BPF may vary among operating systems, they present similar characteristics as shown in Figure 7.9.

The path taken by a packet begins in the NIC (Network Interface Card) when it copies the packet to a pool of system buffers (receive DMA ring) using DMA transfers. These buffers are accessible (using shared memory) to the NIC device driver, as shown by the left side of the figure. This process is termed the first copy.

The NIC generates an interrupt, which triggers the NIC device driver interrupt handler routine to interact with NIC and copy packets from the receive DMA ring to a kernel buffer (BPF buffer), by passing packets headers and content to the BPF callback function bpf_mtap(). This is called the second copy in the figure.

NIC interrupts normally have high interrupt priority level (IPL) (FreeBSD system function...
7.3. Packet Capture Evaluation

splhigh() preempting all tasks running at a lower priority level. The NIC driver is normally assigned to an interrupt priority group by specifying it in its configuration files.

The interrupt may be generated in circumstances that solely depend on the NIC model and manufacturer. In low-cost NICs, it is common for each packet arrival event to generate a device interrupt. Dispatching an interrupt is an expensive operation as the CPU must save/restore its state and switch context. To avoid this cost, interrupts may be batched to allow more packets to be processed by the NIC driver before it returns from the interrupt.

Before the second copy reaches the kernel buffers, a filter is applied to each packet to verify whether it should be accepted or discarded. A noticeable performance gain is attained by filtering out unwanted packets within the interrupt handling code, as it avoids the cost of copying unnecessary packets [106]. The NIC driver also checks whether this packet is addressed to this host. If so, a copy of the packet is placed in the OS protocol stack for the usual protocol processing. Finally, the DMA descriptors of packets that the NIC driver has just received need to be replaced with new descriptors, so that the NIC may receive new packets.

Wanted packets are copied by the BPF code in the kernel process from the kernel-buffer to a user-buffer in user space through the function uio_move(). Once the packet data has been copied to user space, it becomes available to EdgeMeter. The EdgeMeter network reader element may notify event listener elements registered via the function ReportNetworkData() or wait for some element to poll the available packet data (see Section 4.6 for a discussion on the operations of the network reader interface).

By considering the path followed by a packet from the NIC to buffers in user space, the problem observed in Figure 7.7 was carefully studied by Mogul and Ramakrishnan in [113], as follows:

- Receiver interrupts take priority over all other tasks in the system. So, when the packet arrival rate is high, the system spends so much time processing receiver interrupts that no resources are left to send packet data to applications in user space (third copy). The amount of time spent in the NIC driver cannot be predicted or better controlled. This transient condition is called receive livelock. The sooner the packet arrival rate decreases sufficiently to allow useful progress of the system, the sooner the system is able to leave the livelock state and begin making progress again.

Figure 7.10 shows evidence of intense activity of processes running in the kernel/system
space (e.g. kernel itself and NIC driver). These measurements were averaged over one-second intervals and repeated five times. As the arrival packet rate increases, the total CPU utilisation linearly increases. Moreover, when the CPU time spent in user space processes increases, so does the CPU time spent in kernel space processes. For rates above 30k pkt/s, however, the system CPU time continues to increase linearly while the user time curve starts to decrease. This suggests that above this point, which has been identified as the MEFMR (Figure 7.7), the kernel space processes begin an intense consumption of system resources to handle the high rate of interrupts generated by the NIC of the receiver machine. This left few resources available to deliver data to applications running in user space (EdgeMeter). Extrapolation of this graph would result in a complete livelock with the capture rate eventually falling to zero.

Packet transmission is often done at a lower priority than packet reception. This suggests that there is a similar problem with packet transmission, as this process also generates interrupts, but at low Interrupt Priority Level (IPL). Thus, the equivalent to the MEFMR point for packet transmission may be lower than the observed MEFMR of 30k pkt/s.

Figure 7.10: Average CPU consumption for various packet rates
7.3. Packet Capture Evaluation

Figure 7.11 depicts the overall CPU cost for generated packet rates of 30k, 33k and 36k pkt/s. Costs were measured in nanoseconds by Pentium III cycle counters [52] as described in Section 7.3.2; each cost is accumulated from first and second packet copies (Figure 7.9). These results confirm the linear increase of system CPU utilisation when rates increase from 30k to 36k pkt/s, shown on average in Figure 7.10.

A number of techniques have tried to avoid the receive livelock problem. Reducing the rate at which interrupts are imposed on the system is one of them. The system disables interrupts temporarily should the interrupt processing be consuming more than a pre-defined share of resources [146].

Another scheme uses polling during high loads but retains use of interrupts under low loads to avoid the latency associated with pure polling systems. The NIC driver would act proactively by listening to incoming packets as opposed to being called upon by an NIC interrupt. With polling, the decision of when to copy packets is transferred to the NIC driver and kernel process and is not at the discretion of the NIC. This allows some flexibility to divide processing resources between the first, second and third copies.

Kohler et al [99] employed pure polling to eliminate the receive livelock on the basis that even infrequent interrupts in PCs are simply too costly to the system. The interrupts were completely disabled in this system. The results showed significant improvements in a Linux system when polling is implemented by the NIC driver. Not only was the MEFMR point shifted from 84k pkt/s (standard Linux MEFMR) to 333k pkt/s with DEC 21140 Tulip NIC 100 Mb/s
NICs, but the packet capture rate remained constant above MEFMR rather than declining as in Figure 7.7(a).

### 7.3.5 Optimisation

This section discusses some optimisations that can help improve the performance of a packet capture system as to avoid early packet loss and the receive livelock problem.

**Polling**

As discussed in the previous section, polling can be used to address the livelock problem. We present further aspects of this technique along with some necessary changes in the way major operating systems interact with network interface cards.

It has been observed in [146] and confirmed by experimental work in [113] that, as packet arrival rate increases, the system can behave in either of two ways:

1. In systems prone to livelock, the packet capture rate declines for arrival rates above the MEFMR point. The packet capture rate drops to zero if the condition persists, resulting in complete livelock. Pure interrupt-driven systems present this behaviour.

2. In improved systems, the packet capture rate keeps up with the arrival rate up to the MEFMR point. From this point, the packet capture rate is constant. The system is able to make progress, but some of the arriving packets will be dropped between the first and second copies (Figure 7.9). This is the expected behaviour of systems that use techniques based on polling to address the receive livelock problem.

These two behaviours are shown in Figure 7.12. Pure polling [99] or polling only in high loads [113] techniques definitely help in avoiding the receive livelock problem.

On the other hand, interrupt-driven systems such as the FreeBSD version 4.2 used in this experimental work perform poorly by eventually dropping the packet capture rate to zero. This tendency can be clearly observed in the results shown in Figure 7.7.

To implement polling, kernel and NIC driver codes of operating systems need to be modified. Latest versions of FreeBSD and Linux have included limited support to polling in selected NIC device drivers. However, polling support is immature. Not only does the device driver

---

5 FreeBSD NIC drivers patches for Intel EtherExpress Pro, DEC/Intel 21444 and Silicon Integrated Systems 900/SiS 7016 are maintained by Luigi Rizzo at [http://info.iet.unipi.it/~luigi/polling](http://info.iet.unipi.it/~luigi/polling). Linux
need to be modified but the kernel code must be manually patched. NIC drivers supporting polling in FreeBSD, for instance, provide a method (\texttt{poll}), which is periodically invoked by a kernel process to schedule polling of NICs.

**Maximising the Error Free Region**

Reducing the processing overhead is another possible optimisation of the system. If the processing cost, be it associated with either interrupt handling or polling, could be minimised this would certainly provide important resources available for transferring packets from NIC to applications in user space. As a result the MEFMR would be increased.

The last set of experiments carried out had the objective of finding these costs. The BPF/kernel and EdgeMeter codes were instrumented to measure CPU costs in nanoseconds by reading values of Pentium III cycle counters\(^6\). Values for the counters were measured at two points. One measured the summed cost of first and second copies, i.e. NIC to receive DMA ring and then to kernel buffers, and the other measured the third copy cost (kernel to user-space buffers). The results are the average taken over five experimental runs, and each run had a standard deviation of less than 1%.

Figure 7.13 presents the costs per packet. The x-axis shows the generated rates up to 30k pkt/s (MEFMR point), with each packet size being 64-bytes; the y-axis shows the CPU cost in nanoseconds. The curve for first and second copies started with a high cost of 2442ns suddenly

\(^6\)The Performance-Monitoring counter used was the CPU.CLK.UNHALTED.

patches for DEC 21140 Tulip NIC are maintained by the Click project (MIT) at \url{http://www.pdos.lcs.mit.edu/click}.
7.3. Packet Capture Evaluation

![Figure 7.13: CPU cost per packet copy](image)

<table>
<thead>
<tr>
<th>Task</th>
<th>Time ns(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>First and second copies</td>
<td>183 (7.6%)</td>
</tr>
<tr>
<td>Filtering and timestamp generation</td>
<td>2090 (86.9%)</td>
</tr>
<tr>
<td>Third copy</td>
<td>130 (5.4%)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>2403</strong></td>
</tr>
</tbody>
</table>

Table 7.5: Breakdown of CPU costs

decreasing to 2273ns. The reasons are not clear, but may involve interrupt batching in the NIC driver, which seemed to have amortised a fixed cost of processing one packet over several packets.

The third copy costs much less than the first and second ones, the average being 130ns. These results were expected as the first and second copies involve interrupt processing costs, packet filtering and timestamp generation in addition to the copies themselves. The third copy involves only the filling of a mbuf struct to be received by the (read()) function invoked in the user space by applications. 7.

Table 7.5 breaks down the cost of transferring a packet from the NIC to user space. Costs were measured per packet. It is no surprise that filtering and timestamp tasks were the most expensive among all tasks requiring 2.09 μs. The costs of all copies sum to 313ns.

From these tests, it is clear that there is space for further optimisation of the system aimed

---

7Fulvio Risso and Loris Degioanni at Politecnico di Torino found similar results with a Windows BPF implementation called NPF (Netgroup Packet Filter) [http://winpcap.polito.it/]. The reasons for this were that the second copy is performed on a packet-by-packet basis in Windows systems, while the third one usually aggregates several packets in a single transfer to the user space. They also found that timestamp and filtering were the most expensive operations of all in the process.
7.3. Packet Capture Evaluation

at reducing these costs:

- Filtering out packets as early as possible, preferably before the second copy, is one possible optimisation to slightly reduce the overall packet path cost. This can be achieved with BPF by setting packet filters that will eventually be converted into a virtual machine code described in [106]. However, few systems use this BPF feature, preferring to filter packets in user space instead. The packet cost in this case involves two unnecessary copies (second and third) until the application realises the packet is not wanted. Significant improvements in NeTraMet performance were achieved by using the BPF kernel filtering in [40].

- Filtering out unwanted packets in kernel space may reduce the cost of copying but does not eliminate the high cost of timestamp creation and filtering itself. Packets will be checked against filters and the acceptable ones will require timestamps. The most important and relevant optimisation seems to be the reduction of the number of copies along with the radical move of tasks such as filtering and timestamp generation to the NIC hardware itself.

Pratt and Fraser [140] describe the design and implementation of Arsenic, a user-accessible NIC interface which permits applications running in user space to access packet data directly from the NIC. Arsenic offers the combined benefits of packet filtering and reduction in the number of packet copies. Network-intense applications link to a shared user-space protocol library, whilst applications which require low-bandwidth may use the conventional protocol stack offered by the kernel.

Filtering is done in a rather radical manner. Packet filters are described in high-level specification, then transformed into MIPS binary form for direct execution on the NIC’s CPU. This adds the benefit of filtering packets within the NIC so as to avoid the high cost associated with this operation.

Each application accesses a protocol library that exports a standard BSD sockets interface. Protocol libraries access the NIC from the user space through a direct access library (DAL). DAL is responsible for instructing operating system processes, namely the IP registry and device manager, which will then create a virtual NIC interface for the application. The DMA engine of the NIC is programmed to read or write in the host memory allowing, for instance, metering systems to capture data directly from the NIC. This corresponds to the cost of a single
Moore et al [114] have also used packet filtering implemented in the firmware of the NIC with significant performance gains. This approach considerably shifted the MEFMR of their passive monitoring system (Nprobe). Experiments were carried out between a sender host and Nprobe. The latter was running on a Linux system on Intel Xeon 2.4 MHz with a 1GB/s SK-9843-SX Ethernet NIC. Results have shown that the MEFMR for the `tcpdump` [163], which uses BPF as its packet capture engine, was 36k pkt/s with small packets. Such a result confirms the MEFMR of 30k pkt/s achieved with EdgeMeter in a FreeBSD system, as presented in the previous sections.

Nprobe outperformed `tcpdump` with a MEFMR of 187k pkt/s. This improvement suggests that packet filtering at the NIC is a promising optimisation. However, we hypothesize that the major cost reduction which allowed the improvement of about 5 times over `tcpdump` resulted from packet filtering performed in the NIC itself.

It is important to note, however, that the authors recognised that the rate of interrupts generated was extremely high, suggesting that this was an area for future work in order to avoid the livelock problem.

### 7.4 Summary

This chapter has presented an experimental evaluation of the performance of a metering system. The general questions addressed were how such a system fares on providing an accurate metering service, and what the limitations of such a service are.

The experiments were divided into two parts. The first part relates to the flexibility of meter placement in a Differentiated Service (DiffServ) scenario. In this hypothetical scenario, three categories of services were offered to customers. Traffic for interactive realtime, non-interactive realtime and non-realtime services was generated from a sender to a receiver machine. From the results, it was clear that metering error is inevitable. The effects on the maintenance of DiffServ SLA and tariff clauses can be drastic if metering accuracy is not carefully considered. For instance, the results showed volume loss of approximately 12% for a realtime service at high packet rates.

In the second part of the experimental work, the packet capturing process was analysed. This is regarded as a key component of a metering system and possibly the main reason for
metering errors. A prototype implementation of the EdgeMeter architecture was used to carry out experiments under a variety of controlled loads.

It has been found that the source of error is the overhead in transferring packets from the NIC to the metering process in user space. This confirmed the hypothesis that the packet capturing process is one of the causes for errors. This is aggravated by a system design issue known as receive livelock, which makes the packet capture rate falls to zero. When the packet arrival is high, the system spends so much time processing receiver interrupts, that no resources are left to send packet data to applications in user space. Consequently packets are dropped in kernel space.

The cost of the path that is followed by a packet from the NIC to the metering process in user space can be considerably reduced in order to increase the MEFMR. Packet copying, filtering and timestamp generation were identified as the three central tasks performed in this packet path. The major costs found under a FreeBSD system were packet filtering and timestamp generation, which took 2090ns (86.9%) on average to complete. So, if these two tasks could be performed inside the NIC hardware, i.e. consuming the NIC's CPU cycles, they would cost nothing to the host CPU resulting in an increased MEFMR.

The experimental work presented in this chapter have shown that metering errors are inevitable for loads above the MEFMR point. The performance of metering systems has strong influence on accuracy. The next chapter discusses generally these research findings.
Chapter 8

Discussion

Metering performance and accuracy are important aspects to consider in charging and SLA maintenance scenarios. However, little attention has been given to how accurate and appropriate the current metering systems may be for these purposes.

The experimental work presented in the previous chapter showed that performance of metering systems has a strong influence on accuracy. A metering service may be inaccurate under certain loads. This chapter draws out broader conclusions from the experimental findings. It furthers the discussion with speculative interpretation of the results obtained as packet losses have drastic effects on business scenarios. In addition, security and system architecture are generally discussed.

8.1 Summary of the Experimental Findings

Major research projects in the area of charging and SLA maintenance have so far assumed error-free measurement systems [53, 25, 38]. Contrary to this belief, there are limitations on these systems leading to inevitable metering errors, observed when a particular limit of offered load is exceeded. For charging purposes, the following conclusions may limit the flexibility of this research in terms of how metering ought to be performed in practice:

- Measurements always under-estimate the true traffic above a certain offered load imposed on the system hosting the meter code. Such a point, called the Maximum Error-free Metering Rate (MEFMR), depends on the environmental conditions of the system. Elements that might affect the MEFMR, by either increasing or decreasing it, include the operating system, CPU speed, NIC model and its device driver implementation.

- The MEFMR point for NeTraMet [28] running on FreeBSD 4.1 with 10 Mb/s NIC card
was below 1225 pkt/s, as verified in the first part of the experimental work. In addition, the EdgeMeter prototype implementation produced a MEFMR of 30k pkt/s under a slightly different environment, it was running on FreeBSD 4.2 with a 100 Mb/s NIC card. Experimental work carried out independently by Mogul et al [113] and more recently by Moore et al [114] gives confidence in these MEFMR findings.

- The meter's responsiveness to changes in packet rates is poor if the rates change from a point below the MEFMR (low offered load) to a point above it (high offered load). This could be observed with the high error rates in the interval [0, 80] seconds of the experiments with the DiffServ testbed.

- Receiver interrupt processing costs can lead to a condition known as receive livelock, which makes the packet capture rate fall to zero. To address this issue, interrupt disabling and polling techniques can be used in order to give the operating system/NIC driver control over the timing of packet transfers from the NIC.

- The cost of the path that is followed by a packet from the NIC to the metering process in user space can be considerably reduced in order to increase the MEFMR. Packet copying, filtering and timestamp generation were identified as the three central tasks performed in this packet path. The major cost found under a FreeBSD system was packet filtering and timestamp generation, which took 2090ns (86.9%) on average to complete. So, if these two tasks could be performed inside the NIC hardware, i.e. consuming the NIC's CPU cycles, they would cost nothing to the host CPU, resulting in an increased MEFMR, as discussed in [140] and [114].

These are key findings to help advance the understanding of some of the accuracy issues associated with a metering service. The sections that follow discuss how these findings can be used to advantage in charging and SLA maintenance scenarios.

8.2 Customer versus provider measurement

It was argued in Chapter 3 that metering systems should be placed at the customer site mainly for the following reasons: (a) scalability as fine-grained measurements will often be required, (b) customer verification of the charges and (c) the customer's system is likely to have the spare CPU cycles for packet capturing that its providers will struggle to find in their systems, particularly when dealing with a large base of customers.
The results on metering accuracy adds another argument to this list. If a metering system is installed at the provider site, it might prove unfair to customers. Suppose in the cases where the provider’s meter machine accounts for a bandwidth that, although 'delivered' to the customer system, i.e. the packet has reached the NIC, some of the packets could still not make it to applications in user space because they were dropped in kernel space. In this case, fairness could play the deciding role in disagreements.

It can be argued that metering systems should be placed where the data is being sent, in this case, the customer system. This argument is on the basis that there is no way of measuring whether the customer application has really received the data without a measurement system installed at this end.

8.3 Further experiments

There are experiments that could have been done to advance the understanding of the metering accuracy problem. However, they were of lower priority, or would need many more available resources to be performed:

- From the start, the measurements were taken to demonstrate the inaccuracy of a metering system installed in ordinary PC machines, such as the ones used today by end customers. It would be interesting to perform similar experiments on measurement platforms such as the Sprint IP Monitoring System (IPMON), described by Kostantina Papagiannaki in her PhD thesis [129]. IPMON is designed to collect and analyse GPS synchronised packet level traces from selected links in the Sprint Internet backbone. The GPS time synchronisation provides the capability to measure one-way network delays and study correlations in traffic patterns. However, measurements for network backbones require expensive resources, e.g. specialised packet capture cards [49], not to mention access to an operational backbone, such as the Sprint network.

- To extend the validity of the results, further experiments could have been performed with different testing environments, including different NIC models/device drivers and operating systems, had the time allowed. Colleagues at Politecnico di Torino found similar results in Windows systems connected to a Gigabit Ethernet segment. Their results were not yet formally published.
8.4 Security

It has been argued in Chapter 6 that there will be incentives for customers or perhaps providers to cheat in scenarios of metering for charging and SLA validation. Such fraud may lead to drastic results when achieved on a large scale. Generally, there are two security goals. The first is to prevent frauds committed by malicious customers (e.g. meter tampering). The second is to protect customers from any damage that an installed meter code may cause to its local system.

A trusted third-party, the Meter Inspection Authority (MIA), was introduced as part of the framework to address the security issues associated with metering. MIA endorses the accuracy and safety of a meter code prior to its installation on host machines. Once certification of this code has been completed, MIA binds two certificates to it.

Six security requirements were ranked as mandatory, meaning that they must be fulfilled. The proposed framework has addressed these issues by means of cryptographic protocols and certification services offered by MIA. The requirement prevention of tampering of code has been tackled by binding the EdgeMeter code to the MIA digital signature (meter seal), for instance.

As another example, the EdgeMeter system is capable of binding metering data to the SLA/Tariff code (service specification) and the EdgeMeter code version used. This has addressed another important requirement, binding metering data to context.

Implementation details of the MIA’s services have not been provided. With the general specification of these services, as given in this thesis, one may be capable of implementing the MIA’s services. However, there are still open issues regarding how groups of MIAs ought to be structured in a business context. Should it be one MIA entity per service provider? Or should it be one MIA for various service providers and customers? If the latter is the preferred option, groups of MIAs are to be organised so as to avoid problems that may arise, such as a MIA server that becomes unavailable for certifying a meter code.

The use of asymmetric encryption is another aspect that deserves attention. The protocols discussed rely on public-key encryption, which is regarded as an expensive task when frequently performed. Other research [57] has found that 25% of the time spent on receiving and sending a packet carrying active code is used for authentication, i.e. generating and verifying digital signatures. This overhead may only be significant in our security framework for the protocol step of sending metering data from a meter at customer side back to its provider. Depending on
how frequent this might be required, the use of symmetric encryption may be more appropriate to
this step. Communication between these two ends would be encrypted using a temporary key
(session key). This key would be typically used for the duration of a metering data transmission.
As the remaining of the protocol steps occur infrequently, the cost of the asymmetric encryption
used can be amortised over time.

Finally, prevention of certain types of DoS attacks on a metering system is another open
issue. As this problem arises at runtime, a solution might require a “monitoring” component
in the EdgeMeter architecture that is able to continuously check the internal behaviour of the
system. Abnormal behaviour would trigger an action. What seems difficult, however, is how
to draw a line between normal and abnormal operation of the system. Potential DoS attacks,
such as packet flooding, would have to be characterised beforehand so as to include them in this
monitoring approach.

8.5 System Architecture

8.5.1 Flexibility

The EdgeMeter system architecture introduces flexibility for metering at the edges of the net-
work. Meters can be placed at either provider or customer domains. The provider meter can
carry out coarse-grained measurements, whereas the customer is able to measure at a fine gran-
ularity, and is in fact much more interested in this type of measurements. This adds flexibility
in choosing the granularity of metering data.

Because EdgeMeter is a modular system, modules can implement either passive or active
measurement techniques. When metric implementation code for a service specification cannot
be located within the metering system itself, EdgeMeter has the capability to find it through
external module discovery. This may involve one system contacting other metering systems
across the interface domains to discover a particular metric code.

Another important aspect is code distribution. Apart from the standard means of distri-
bution (e.g. download from a web server), it is envisaged that active network themes may be
helpful in this process. The meter code can be embedded in a proxylet [80] and made available
in some repository (proxylet servers). Such a practice would facilitate the dissemination of new
code versions 'on the fly', at the expense of some additional management to correctly maintain
these codes in the proxylet servers. Such a mobile meter code can be applicable to mobile IP
scenarios. Also, resource constrained mobile devices (e.g. cellphones, PDAs, car's control sys-
tems) may download code that only implements a subset of EdgeMeter. It is important to note that there is a trade-off between performance and accuracy that needs to be considered in such a scenario. When the mobile code is not native, performance issues may arise.

### 8.5.2 Implementation

All EdgeMeter components would have been implemented had the time allowed. However, it was necessary to choose the most critical to implement in this project. The network reader component was implemented as part of the EdgeMeter prototype described in Chapter 7 to assess the packet capture process. The sections that follow briefly discuss the implementation of another two architectural components.

### 8.5.3 Module Discoverer

The Lighthouse code was a C++ implementation of the method `FindNetworkLocation()` of the module discoverer component’s interface. The code allowed an analysis of this framework feasibility (refer to Section 5.2). However, this network location scheme faces questions associated with the problem of choosing the right performance metrics. So far, we have only experimented with Internet delay, as it can be triangulated [78]. What are the additional metrics that could be used? How could these metrics be *practically* measured? Is ‘available bandwidth’ a feasible metric? If so, could we characterise it as we did with network delay?

The choice of a distance function is another important aspect. In the experiments presented in Chapter 5, we tested the \( L_p \) family of functions with \( L_p = \left( \sum_{i=1}^{k} |x_i - y_i|^p \right)^{1/p} \). When \( p = 2 \) we have the \( L_2 \), which is the Euclidean distance. In contrast, for \( p = 1 \), we have the ‘Manhattan’ or block distance. Additionally, \( p < 1 \) results in a non-metric distance function used where the distances do not obey the triangle inequality [91]. We varied \( p \) from 0.0 to 6.0 in our experiments and found that the \( L_2 \) function has given better delay estimates than any other derivation of \( L_p \). However, the question is: could the \( L_2 \) distance function be applied to other network performance metrics such as the available bandwidth?

Network performance metrics may greatly differ among their *representational* dimensions \( k \) in a vector space and their *intrinsic* dimensionality. This is related to the real number of dimensions that have to be used while maintaining the original distance - called the *curse of dimensionality*. To match the *intrinsic* dimensionality of Internet delay distances, 5 to 7 dimensions were required [124]. This prompts the question: do we need to experimentally find out the intrinsic dimensionality of other network performance metrics, such as the available band-
width? Or could we assume that a vector space with 5 to 7 dimensions can model any network performance metric?

### 8.5.4 Service Translator

The configuration code of the metering system should be translated from a network service specification, i.e. the merging of an electronic tariff and an service level agreement (SLA) structures. Part of the EdgeMeter service translator component was implemented and validated within the scope of an MSc. project [7]. However, it has mainly addressed the issue of translating from a legal contract document written in plain english to the STS representation (STS is described in Appendix A). It is worth mentioning that any performance assessment of the service translator would be very dependent on implementation decisions.

### 8.6 Summary

This chapter has drawn out broader conclusions from the experimental work, security framework and the overall EdgeMeter system architecture. The experimental findings may limit the versatility of research on charging mechanisms, in terms of how metering ought to be performed in practise.

Measurements always under-estimate the true traffic above a certain offered load imposed to the system hosting the meter code. Such a point, called the Maximum Error-free Metering Rate (MEFMR), depends on environmental conditions of the system (NIC model and operating system).

The security framework was also discussed. There are still open issues with the structuring of MIA servers in a distributed system, which involves meters installed on provider and customers' host machines. Scalability issues arise from this distributed metering system. What happens when a MIA server becomes unavailable, so that a meter code cannot be certified by this server?

Also, the flexibility of the EdgeMeter system architecture was discussed. Meters can be placed in either the provider or customer domain. The end-customer placement introduces security issues that were addressed by the security framework.

With the proposed network field framework, various measurement granularities can be supported by EdgeMeter. To improve scalability, the provider meter may carry out coarse-grained measurements by specifying so in a network flux template, whereas the customer is able
to measure fine-grained measurement, such as packet level monitoring with this framework.

The modular approach of EdgeMeter allows metric modules to be hooked in on demand. Some metric codes may implement passive techniques, for instance, the throughput at a specific network point. Others may implement active measurement techniques, such as the delay of a segment. This introduces versatility of a hybrid system, that supports both types of measurement techniques.

When metric implementation code for a service specification cannot be located in the metering system, it is important to the system to have the right capability of finding it. This might involve one system contacting other metering systems across the interface domains to discover a particular metric code. Open problems with the proposed technique to locate meters in the network, Lighthouse, were also discussed.

The chapter concluded that the EdgeMeter provides a platform for end-customer metering of networked services, applicable to the general business scenario of the future market for Internet access services.
Chapter 9

Conclusions and Future Work

This dissertation has examined the issues associated with end-customer metering of networked services in a business context. An infrastructure to address these issues was presented, named EdgeMeter. Performance limitations of end-customer metering systems were examined with a prototype implementation of EdgeMeter. This chapter summarises the research and suggests areas for future research arising from this work.

9.1 Summary

Chapter 2 introduced the state of art in Internet metering by putting the architectures proposed in a common framework, which is based on three parts. The location of meters is one of them: meters can be placed at the edge of the network, i.e. close to senders and receivers; or inside the network, for instance, in cases where an operator’s network backbone is being monitored. The second part of the framework concerns the type of measurement, which can be either active or passive. Granularity, i.e. how detailed the measurement data can be, is the third part developed within this framework. A taxonomy of metering systems was developed using this framework. General observation was that meters are predominantly deployed inside the network and passive metering dominates the techniques employed. On the other hand, granularity types seem to be balanced, with a number of systems supporting both coarse and fine-grained metering. It was argued that passive measurements require processing and storage space when carried out inside the network at packet level (fine-grained). In particular, the IETF RTFM and Cisco NetFlow (flow-based passive metering systems) were presented as potential systems for usage-based accounting for billing purposes.

Chapter 3 examined two supporting cases for metering by analysing a business scenario where service providers interact directly with end-customers. The deployment of a QoS mecha-
anism on the Internet and mechanisms to charge for the QoS delivered were analysed. The importance of fine-grained usage metering and accounting to the widespread of these mechanisms was also presented. Wherever meters are installed in this context, they certainly sit at the edges of the network, introducing a region of two-meter placements - the provider or the end-customer domain.

The rationale for moving metering functions to the end-customer domain was discussed. This scenario can afford finer granularities at low resource consumption, making the overall system scalable. The flexibility of meter placement, on the other hand, poses issues that were also introduced. The issue of controlling the metering system based on high-level code (service specification) was presented. Security is another concern, as there will be incentives for customers and even providers to cheat when advanced charging schemes are used. Finally, performance issues associated with a metering system are important to understand, as to see how the business scenario presented in this thesis may be affected by metering inaccuracy.

Metering systems that address these issues were surveyed. Some of the systems were designed to run in managed elements inside the network, in order to collect coarse-grained measurement data. Other systems may be used at the edges of the network, but lack security and an automated management mechanism.

Chapter 4 described the EdgeMeter system and its design principles, which aim to facilitate the accounting of networked services and the measurement of the quality of service received by end-customers. To address the metering issues previously presented, EdgeMeter supports QoS measurements within a certain accuracy, safe transport and storage of measurement data. It also supports discovery and installation of new metric modules, and the flexible configuration of metering points based on electronic specification (tariff/SLA) automatically distributed by service providers.

Deployment aspects of EdgeMeter as part of a general charging scenario were introduced. The deployment follows two phases: in the initialisation or code distribution phase, the EdgeMeter code is installed in the appropriate points of measurement; in the following operational phase, the metering system is instructed to measure the network traffic based on the metering requirements described in a service specification code.

The EdgeMeter inner components and their internal interactions were presented. Internal components were grouped based on their functionality. The configuration input group includes components (service translator and policy reader) that export a set of interfaces tailored for
managing EdgeMeter. The policy management group deals with all aspects related to metering policies, ranging from their specification to their runtime maintenance. As the architecture is modular, it allows modules to be hooked in on demand, such as code to implement metrics that specify what and how a measurement should be performed. The module management group provides the components responsible for managing modules within the system. Components of the network group deal with information extracted from packets received.

The internal data structures and algorithms of a few of the EdgeMeter components were detailed in Chapter 5. First, the module manager component was discussed. Modules may be loaded from local or pre-defined external module servers, and are assigned to metering sessions at runtime. Session isolation issues were analysed so that 'cross-talk' activities between sessions cannot interfere with one another. Also, if the module cannot be located locally, the module discoverer element performs a lookup within a set of module servers. The implementation of the operation to calculate a network location for a meter system, exported by the discoverer component, is described. A scalable location framework, Lighthouse, is proposed and evaluated. It was shown that Lighthouse computes more accurate network locations, when compared to other related techniques. Finally, the design of the service translator, which is the translation mechanism of service specification code into a policyset was introduced.

In Chapter 6, the EdgeMeter security framework was discussed. An analysis of the deployment of EdgeMeter in a charging context led to a list of security requirements. Prevention of meter tampering, binding of measurement to context, execution of authentic code and mutual authentication are some of them. It was argued that a trustworthy third-party component, the Meter Inspection Authority (MIA), is required. MIA offers certification services to endorse accuracy and safety of metering systems. It was shown that a set of cryptographic protocol steps established between customers, service providers and MIA were sufficient to fulfil the security requirements.

Details of experimental work were presented in Chapter 7. The general questions addressed were how a metering infrastructure fares on providing an accurate metering service, and what the limitations of such a service are. The experiments were divided into two parts. The first part relates to the flexibility of meter placement in a Differentiated Service (DiffServ) scenario. In this hypothetical scenario, three categories of services were offered to customers. Traffic for interactive realtime, non-interactive realtime and non-realtime services was generated from a sender to a receiver machine. From the results, it was clear that metering error is inevitable. The effects on the maintenance of DiffServ SLA and tariff clauses can be drastic if
metering accuracy is not carefully considered. For instance, the results showed volume loss of approximately 12% for a realtime service at high packet rates.

In the second part of the experimental work, the packet capturing process was analysed. This is regarded as a key component of a metering system and possibly the main reason for metering errors. A prototype implementation of the EdgeMeter architecture was used to carry out experiments under a variety of controlled loads. The prototype was developed over the MIT Click modular router system on FreeBSD. Two machines were used, one sending and the other receiving traffic. The sender host generated UDP packets at specified rates towards the receiver. The results were presented and it was found that the source of error is the overhead in transferring packets from the NIC to the metering process in user space. This confirmed the hypothesis that the packet capturing process is one of the causes for errors. This is aggravated by a problem known as receive livelock, which makes the packet capture rate falls to zero. When the packet arrival is high, the system spends so much time processing receiver interrupts, that no resources are left to send packet data to applications in user space. These packets are consequently dropped.

The discussion of the experimental findings, security framework the EdgeMeter system was presented in Chapter 8. The experimental findings may limit the versatility of research on charging mechanisms, in terms of how metering ought to be performed in practise. It has been found that measurements, undertaken at the edges of the network, always under-estimate the true traffic above a certain offered load imposed to the system hosting the meter code. The security framework was also discussed. There are still scalability issues with the structuring of MIA servers in a distributed system, which involves meters installed on provider and customers' host machines.

9.2 Contributions

The main contributions of this dissertation can be summarised as follows:

- An infrastructure, named EdgeMeter, is proposed to address the issues associated with end-customer metering in scenarios of charging and SLA maintenance. A translation mechanism is introduced to tackle the problem of managing a metering system based on high-level code specification, such as electronic tariffs and SLAs.

- A security framework based on a trustworthy third-party component, the Meter Inspection Authority (MIA), is proposed. MIA endorses the accuracy and safety of metering
9.3 Current Limitations to the Edge Metering Approach

This thesis has considered some of the issues associated with Internet metering in depth. However, there are issues still remaining which may limit or prevent the deployment of EdgeMeter systems without perhaps further research. This section describes these areas of concern.

9.3.1 Distinguishing Application’s Data Streams

Traffic classification techniques such as grouping packet streams into flows\(^1\) are common means to distinguish the data streams generated by user applications. The protocol type and destination port number are usually the most appropriate information of a flow that can link an application.

---

\(^1\)Flow is a distinguishable stream of related packets that results from an user application. Flow can be specified as a 5-tuple (source IP address, destination IP address, source transport port, destination transport port and protocol ID). Flow information are captured from the packet headers.
to its set of data streams. For instance, a 5-tuple flow such as (10.0.0.1, 10.0.0.2, 0, 80, TCP) may be linked to an web application (port number 80) running in a host with IP address 10.0.0.1.

The EdgeMeter system is capable of accounting for data streams of user applications since the captured packet headers can be classified into flows (see Section 4.4.4). Analysing only packet headers in order to build flow tables, however, might not be sufficient to relate packet streams to applications in some cases. Some applications (e.g. peer-to-peer file sharing) dynamically assign port numbers, which creates an issue if the metering system relies on flow information only.

To illustrate this issue, we can consider the scenario of voice sessions (VoIP) monitoring for network management purposes. It seems acceptable in this case to identify the VoIP data streams by means of flow classification. Billing for voice sessions, however, may require storing protocol status information which will depend on the VoIP protocol used (e.g. ITU H.323 [90]). This may only be successfully achieved if packet content is disclosed for a deep analysis. With access to the full packet content, a finite state machine installed in the metering system can simulate the VoIP protocol steps. This would allow the metering system to distinguish, for instance, silence suppression from congestion.

In synthesis, the metering system should be able to inspect the packet content so as to better differentiate the data streams of a particular application. This is an area that was not explored in this thesis.

9.3.2 Security Difficulties

The metering placement scenarios presented in this thesis introduced three possible meter installation points. Meters may be running in the provider domain, end-customer domain and sometimes simultaneously in both domains. Some of the security issues that arise from these scenarios were discussed in Chapter 6. However, there are difficulties still remaining related to end-customer metering placement.

It is not a straightforward task to install and continuously run a meter system, i.e. a foreign code, in the end-customer system. There are issues with the host operating system, which can make any software-based tamper-proof technique hard to be adequately secure in practice. Ultimately, the end-customer owns and controls this system environment, which can be modified to circumvent the metering process.

Although aspects of the integrity of measurement originating in the user domain have been
addressed, on their own they would not be robust enough for an operational system. A solution space for this problem seems to be in the area of audit trails. For example, the provider’s default policy may be to trust the end-customer to do metering. The provider should then be able to ‘switch on’ a meter installed on its own authoritative domain for auditing purposes, should it believe its customer is cheating, e.g. tampering with the meter software. Finding the reason to trigger the provider’s checking meter is another issue that deserves attention. Also, this solution needs further insight if it is to be implemented with the framework proposed in this thesis. EdgeMeter has weaknesses in traceability and auditability of records being sent from the end-customer domain to billing components in the provider’s domain.

In addition, the cutover between user and provider measurements would result in difference. This is inevitable, and further investigation into minimising such discrepancies are still needed.

### 9.3.3 Monitoring of Global SLAs

Some network services are likely to be delivered via several providers. Such a service is likely to have a global SLA associated with, one that comprises clauses covering the performance levels experienced by the network service. When a service fails to meet its SLA the issue of whose fault it is must be tackled. Identifying the source of faults is a typical problem for which no effective solution is in place today in the Internet.

A video streaming service contracted from an application-level service provider (e.g. AOL), for instance, needs to be delivered to its customer through a chain of provider domains. The global SLA that covers such a service guarantees enough throughput, low jitter and packet delay, so that it can be delivered without disruption. However, the traffic passing through the various providers domains may be delayed or even dropped during its course. It is probable that this situation violates the SLA and as such the customer should be somehow compensated.

In this case, the application-level service provider is responsible for finding whose fault it was in order to recover the compensation given to its customer. One can say that there is no easy solution to the issue of identifying the source of a fault in the Internet should we rely on edge measurements only. This poses a serious limitation to the edge metering approach.

Perhaps one solution for the issue of identifying the source of service faults for global SLA monitoring is to place measurement systems within the service providers networks [129]. However, these systems need to exchange information and to be synchronised as to timely
identify sources of faults. Also, it needs worldwide deployment, which is indeed a major issue to address.

To avoid global SLAs, some broadband access providers are offering today services such as video streaming by caching the content of application-level providers. For this service, a local SLA is perfectly sound as the video content is locally fetched from the broadband provider's cache systems. Edge metering may be used in this case as this local SLA will cover only the performance of a manageable system environment, i.e. the broadband provider's network.

9.3.4 Bursty and Steady Data Streams

Some of the user application's data streams may be bursty, e.g. due to network conditions, or steady. The metering system is expected to accurately measure and report with enough granularity the application's flows in order to represent their original traffic characteristics. The EdgeMeter system is capable of reporting bursty and steady data streams. This capability is instrumented through the attribute report, as explained in Section 4.4.4. If this attribute is tuned to short timescales, the bursty characteristics of the arriving data streams may be preserved.

However, the timescale in which packets should be reported may be limited by performance issues. Such packets are required to follow the path from the NIC to the EdgeMeter process, which runs in user space (see Chapter 7). The shorter the timescale, the more difficult it will be to precisely report the arriving data stream. Further meter performance experimentation may assist in finding out the limitations of a system while measuring bursty traffic, so that the traffic characteristics are well captured.

9.4 Future Work

This section suggests other areas for future research arising from this work. Also, other applicability scenarios for EdgeMeter are presented.

9.4.1 Exploring logical processors

The results of this thesis showed the high cost of packet processing (copying, filtering and timestamp generation), specially in interrupt-driven systems. NIC interrupts are normally processed at the highest priority level, often starving the system which must then discard packets in response. Techniques that shift the control of the packet transferring process to the OS/NIC driver include polling with interrupts often disabled. A promising area to explore is the use
of new processor architectures that allow the creation of logical processors such as the Intel
Hyper-Threading architecture [52]. Logical processors share the real processor’s execution en-
gine and the bus interface. The fact that a logical processor has its own interrupt engine, an
advanced programmable interrupt controller (APIC), is rather relevant to this thesis. This might
be explored in many ways as to balance the control exercised by the operating system, device
driver and NICs over the packet transmission cycle. To address the livelock problem, priorities
can be fairly established without the need for device driver modifications.

9.4.2 Metering accuracy of flow-based systems

Flow-based systems like Cisco NetFlow [44] had a major software revision after ISPs realised
the degradation in router performance attributed to the NetFlow overhead. With the ‘aggrega-
tion version’ [45], packets are now filtered within the router replacing the previous mechanism
of sending the majority of captured flow traffic to remote machines (collectors) for filtering and
further processing. The question of what the MEFMR of a typical NetFlow system is remains
open to the best of our knowledge. Such systems are widely used in ISP backbones.

9.4.3 Lighthouses for network proximity exploitation in Peer-to-Peer Systems

The Lighthouse technique presented in Chapter 5 is a promising scalable framework that can
help addressing the important question of how to exploit network proximity in peer-to-peer
systems such as distributed hash table (DHT) protocols (e.g. Pastry, Chord and CAN).

Moreover, the peer-to-peer system scenario introduces new security requirements that need
to be addressed. This type of system is usually decentralised and fully distributed, therefore,
there should exist a mechanism to dynamically establish trust between nodes without relying
on centralised third parties.

For instance, Lighthouse assumes an environment where nodes cooperate to calculate their
own coordinates. A joining node trusts that a subset of the nodes already in the system will
inform their coordinates. However, the question is how the joining node assesses the risks of
trusting these nodes as the basis for its position calculation. This problem can be extended to
other scenarios such as Grid and mobile IP.

It might be interesting to investigate models of trust where nodes can independently decide
whether or not they can trust other nodes in the system. One approach is to define an additional
dimension of the coordinates computed by Lighthouse so as to indicate a trust level and perhaps
use this information in a multidimensional search algorithm (see below).

### 9.4.4 Multidimensional search to approach resource discovery

Several issues arise in the discovery of available resources in a peer-to-peer/Grid system. At least three key properties that a resource discovery mechanism should have to address some of these issues are: (a) **underlying topology**, routing a resource query request should consider the underlying IP topology, for instance, in terms of delay and bandwidth; (b) **flexible query construction**, the mechanism should provide a means to formulate efficient and flexible queries, not only exact but also partial match (e.g. k-Nearest Neighbour) queries; (c) **Scalability**, the discovery mechanism should be fully distributed. Search mechanisms in multidimensional spaces, where a given resource is specified as a point in a vector space, may be considered an approach to these requirements.
Chapter 10

Glossary of terms

Accuracy: refer to metering accuracy.

Active metering: technique that probes the network by injecting test packets (management traffic). Active metering relies on a pair of meters.

Aggregator: module that groups measurement records together. It can further classify and refine the data.

Application-level service providers: sell services such as audio/video streaming and IP Telephony. This provider interacts with network providers in order to have connectivity.

Berkeley Packet Filter (BPF) architecture: it carries out packet filtering in kernel space for improved performance. It is available for most Unix and Windows system versions. BPF-based tools (e.g. tcpdump) are widely used for passive metering at the network edges.

Billing system: responsible for generating a bill to a customer based on the tariff structure assembled in the service specification code.

Bulk network providers: own a massive network backbone infrastructure usually being geographically spread. Services offered include network-based services, such as backbone connectivity to network wholesalers.

Calibration: refer to metering calibration.

Coarse-grained granularity: the metering process classifies and aggregates IP packets based on groups of related traffic so as to generate statistics. Such statistics can be, for instance, counters for a router interface or information about a TCP connection.

Explicit Congestion Notification (ECN): it has been used in congestion charging mech-
anisms to indicate congestion, which can be evaluated by the number of ECN marked packets received. ECN relies on active queue management installed in routers, e.g. random early detection (RED), to make a decision of whether or not to set congestion bits in the TCP and IP headers.

**Differentiated Service (DiffServ):** a QoS mechanism proposed by the IETF. Traffic control is mainly done at the edges of the network, thus, the core routers are mainly left with the task of forwarding packets. Diffserv eliminates the need for per-flow state and per-flow conditioning by treating the traffic in aggregates of flows.

**DiffServ code point (DSCP):** number that identifies to which class of aggregated traffic a packet belongs to. The DSCP value is inserted into the Type of Service (TOS) field of the IPv4 header.

**Edges of the network:** it is the scenario where service providers, namely application level provider and network retailers, directly interface with end customers. Wherever meters are installed they certainly sit at the edges of the network introducing two regions of meter placements: either provider or customer domains. **Electronic tariff:** determines how a service ought to be charged.

**Expedited forward (EF) PHB:** one of DiffServ PHBs currently specified. The EF PHB gives low loss, low delay and low jitter to applications by ensuring that the service rate of EF packets on a given output interface exceeds their arrival rate at that interface over long and short time intervals.

**Fine-grained granularity:** the metering process receives IP packets of operational or management traffic as the input for the measurement. Fine-grained data such as packet-level metering can be obtained by correlating packet traces received from multiple meters.

**Flow-based approaches:** the meter classifies a stream of packets into flows based on a five-tuple set: source/destination IP and port numbers and protocol number.

**Global Positioning System (GPS):** satellite assisted location system. It can be used as a timing reference source with accuracy around 1 microsecond for metering systems. One drawback of GPS, however, is the installation cost of a receiver. It is not only the purchase cost that counts but also the need to install an antenna where it has a clear view of the sky.

**Granularity:** is how detailed the resulting measurement data can be. Measurement granularity may be fine-grained (packet level and flow-based) or coarse-grained (e.g. RMON).
Inside the network: metering carried out within a network backbone that belongs to a bulk or network wholesaler provider.

Integrated Services (IntServ): an IETF architecture that introduces a set of end-to-end services tailored to realtime applications. The architecture uses per-flow traffic conditioning at every network element along the end-to-end path. Network elements, components of the network capable of exercising QoS control over the data flowing through them, include individual devices such as routers and end-host operating systems.

Interception loop: attribute of a network flux template that holds filter expressions that combine attributes and values captured from packet headers and content.

Libpcap: standard API for packet capture applications, capable of configuring the BPF system at a reasonable level of abstraction. Libpcap has been used in most of today's network monitoring tools (e.g. tcpdump).

Management traffic: traffic injected in the network by management components (e.g. the metering system).

Manager: administers the meter using a configuration script. This process assumes a description of the set of metrics to be measured and how collectors will gather the results.

Measure: the numerical value of some attributes of a metric.

Measurement data: the resulting data generated by the metering system organised in data units called measurement records.

Meter: element that provides measures for a set of metrics by passive 'tapping' or active probing of the network. It is sometimes referred in the literature as a probe or trace.

Meter Inspection Authority (MIA): a trustworthy third-party component, whose role is to endorse the accuracy and safety of metering systems for providers and customers.

Meter Inspection Certificate (MIC): issued by MIA to indicate that a code inspection carried out is positive. Two types of certificates can be used: (a) safety certificate and (b) calibration certificate.

Meter manufacturer: refer to network management provider.

Metering/measuring: a process by which a measure is assigned to a metric. Measurement and monitoring are other terms used interchangeably.
**Metering accuracy**: is the degree to which a given quantity is correct and free from error. For instance, a quantity specified as \(80 \pm 5\%\) has a relative accuracy of \(\pm 5\%\), meaning its true value can fall in the range of 76-84.

**Metering calibration**: a technique to compensate for metering inaccuracy.

**Metering session**: standard unit of a policyset instantiation in EdgeMeter.

**Metering system**: a group of elements with the ability to gather and further refine the measures for a set of metrics. Meters and extra modules such as managers, collectors and aggregators form this system. They can be attached to the meter or reside externally.

**Metric**: an attribute of an entity that can be expressed qualitatively (what to measure) along with the specification of one possible way of calculating it (how to measure). Throughput, peak rate and packet loss are examples.

**MIB**: a tree-based database structure that stores statistics and related information accessible through the SNMP protocol.

**MIB-II**: MIB, largely supported by routers, that enables an overall understanding of the network. It supports variables that convey state information of the router resources (e.g. CPU and memory load) along with the network local environment. This information is most useful to long-term measurements, as variables will be updated in large time-scales (e.g. minutes to hours).

**Module**: standard unit of code in EdgeMeter.

**Module manager**: offers an interface tailored for the maintenance of modules in the EdgeMeter system.

**Module server**: repository of modules, located in a metering system or inside the network.

**NetFlow**: a passive metering system (flow-based) proposed by Cisco. It is tailored for measurements inside the network. Applicability scenarios include accounting for billing and network planning.

**Network field**: a set of one or more streams of packets observed by a metering system. It is a measurement scheme that brings versatility in choosing, specifying and metering a variety of granularities. By tuning a few parameters this scheme is capable of assembling most typical granularities such as packet-level, flow-based and bulk-based statistics.
**Network field template:** defines the characteristics of a network field that will eventually be measured by a metering system.

**Network flux template:** the observed network field needs to be intercepted by the metering system to collect the statistics required about the packet streams that form the network field. The measurement of a network field leads to a network flux. This template describes the specifics of how a network field is to be measured. The template includes some attributes of the interception loop area, a time interval and the metrics to be metered.

**Network management providers:** include providers that offer enhanced services for network management purposes. The list of services offered include but are not limited to, network monitoring and planning. Examples of providers include meter manufacturers that sell services such as metering software tailored for billing.

**Network providers:** consist of bulk providers, wholesalers or retailers, who sell network-based services (e.g. data transportation, network access).

**Network retailers:** sell network-based services directly to customers. Usual services include network access to the Internet, either via dedicated lines (ADSL/Cable) or dial-up access.

**Network service specification:** specification of a network service at a high-level of abstraction, as a binding of Electronic Tariffs and SLAs.

**Network time protocol (NTP):** protocol used to synchronise machine clocks to an external reference source with accuracy in the range of milliseconds. GPS receivers can provide better accuracy but at higher cost.

**Network traffic:** traffic which is observed at a metering point as the sum of the operational and management traffic.

**Network wholesalers:** own a network backbone, but on minor scale compared to bulk providers. They deal with network retailers, selling network-based services.

**Operational traffic:** traffic generated by a sender destined to a receiver element.

**Passive metering:** traffic is captured (e.g. tapping the network) at the metering points with minimal impact to the normal network operation. A large group of statistics (e.g. byte counters) can be calculated.

**Per-Hop behaviour (PHB):** specifies how packets from a certain traffic class should be
treated in each network element (hop).

**Policyset:** small program in EdgePol language that encloses policy rules in its body part.

**Policy rule:** composition of events, conditions and actions (ECA).

**Policy decision point (PDP):** element that makes policy decisions which are the evaluation of a policy rule in the IETF policy framework.

**Policy enforcement point (PEP):** element that sits at network nodes and is responsible for the enforcement of a policy.

**Real time flow measurement (RTFM) architecture:** it is an IETF standard for flow-level measurement system. The Meter-MIB compiles a list of common flow attributes (e.g. packet counters) in a MIB structure. The RTFM architecture proposes three components: (a) meters, which collect packets from the network; (b) readers, that gather measurement data periodically (pull mode) from the meters; and (c) managers, which configure meters through a rule set written in a high level language such as the SRL.

**RMON:** an IETF MIB applicable to network monitoring. It provides information about the network in short time scales. For instance, a traffic matrix at layer 2 can be collected.

**RMON2 MIB:** extends the RMON capabilities to allow traffic monitoring above layer 2. The network manager may monitor traffic based on network and transport layer protocols.

**RSVP:** protocol that provides an explicit signalling mechanism to request resource reservation within a network element. It has been used with the IETF Intserv architecture.

**Service level agreement (SLA):** legal contract between customers and service providers specifying acceptable thresholds of quality for the provision of a network service.

**Service level specification (SLS):** technical aspects of an SLA, such as the parameters to be measured that convey the acceptable quality levels of a service.

**Service translator:** performs the translation of the specification code from high to low-level (metering configuration code).

**SLA maintainer:** strips out the SLA part of the service specification to continuously validate the service level contracted.

**SLA/Tariff specification language (STS):** XML based language to specify the high-level
service specification code.

**SNMP**: protocol implemented in the majority of routers and management devices on the Internet. It is used to remotely collect values from MIB objects.
Appendix A

SLA/Tariff Specification language (STS)

The STS language was proposed within the scope of the MSc. project “High-Level Translation of Service Level Agreements and Tariffs for Internet Services” [7], Birkbeck College (University of London). This appendix presents the XML Schema for such a language.

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="networkServiceSpecification">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="SLA" type="SLAType" />
        <xsd:element name="Tariff" type="TariffType" />
      </xsd:sequence>
      <xsd:attribute name="ID" type="xsd:ID" use="required" />
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```xml
<xsd:complexType name="SLAType">
  <xsd:sequence>
    <xsd:element name="purpose" type="xsd:string" />
    <xsd:element name="scope" type="scopeType" />
    <xsd:element name="parties" type="partiesType" />
    <xsd:element name="validityPeriod" type="validityPeriodType" />
    <xsd:element name="version" type="xsd:string" />
    <xsd:element name="SLO" type="SLOtype" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
```

```xml
<xsd:complexType name="TariffType">
  <xsd:sequence>
    <xsd:element name="TO" type="TOtype" maxOccurs="unbounded" />
    <xsd:element name="penalty" type="penaltyType" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
```

```xml
<xsd:complexType name="scopeType">
  <xsd:sequence>
  </xsd:sequence>
</xsd:complexType>
```
<xsd:element name="scopeDescription" type="xsd:string" />
<xsd:element name="supportHours" type="xsd:string" />
<xsd:element name="responsabilities" type="xsd:string" />
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="partiesType">
<xsd:sequence>
<xsd:element name="customer" type="xsd:string" />
<xsd:element name="provider" type="xsd:string" />
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="validityPeriodType">
<xsd:sequence>
<xsd:element name="startDate" type="xsd:dateTime" />
<xsd:element name="endDate" type="xsd:dateTime" />
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SLOtype">
<xsd:sequence>
<xsd:element name="dayTimeConstraint" type="dayTimeConstrType" />
<xsd:element name="reportInterval" type="reportIntervalType" />
<xsd:element name="metric" type="metricType" maxOccurs="unbounded" />
</xsd:sequence>
<xsd:attribute name="ID" type="xsd:ID" use="required" />
</xsd:complexType>
<xsd:simpleType name="dayType">
<xsd:restriction base="xsd:NMTOKEN">
<xsd:enumeration value="Mon" />
<xsd:enumeration value="Tue" />
<xsd:enumeration value="Wed" />
<xsd:enumeration value="Thu" />
<xsd:enumeration value="Fri" />
<xsd:enumeration value="Sat" />
<xsd:enumeration value="Sun" />
</xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="dayTimeConstrType">
<xsd:sequence>
<xsd:element name="dow">
<xsd:simpleType>
<xsd:list itemType="dayType" />
</xsd:simpleType>
</xsd:element>
<xsd:element>
<xsd:element name="startTime" type="xsd:time" />
<xsd:element name="endTime" type="xsd:time" />
</xsd:element>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="reportIntervalType">
<xsd:sequence>
  <xsd:element name="every" type="xsd:positiveInteger" />
  <xsd:element name="unit">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="seconds" />
        <xsd:enumeration value="minutes" />
        <xsd:enumeration value="hours" />
        <xsd:enumeration value="days" />
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
  <xsd:element name="dow">
    <xsd:simpleType>
      <xsd:list itemType="dayType" />
    </xsd:simpleType>
  </xsd:element>
</xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="basicOperators">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="=" />
    <xsd:enumeration value="&gt;" />
    <xsd:enumeration value="&gt;=" />
    <xsd:enumeration value="&lt;" />
    <xsd:enumeration value="&lt;=" />
    <xsd:enumeration value="NA" />
  </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="metricType">
  <xsd:sequence>
    <xsd:element name="description" type="xsd:string" />
    <xsd:element name="metricUnit" type="xsd:string" />
    <xsd:element name="measureAt" type="xsd:string" />
    <xsd:element name="lowerBoundValue" type="xsd:double" />
    <xsd:element name="lowerBoundOperator" type="basicOperators" />
    <xsd:element name="upperBoundValue" type="xsd:double" />
    <xsd:element name="upperBoundOperator" type="basicOperators" />
    <xsd:element name="dataCollectionSlot" type="xsd:string" />
    <xsd:element name="nextMetricLogicalOperator" minOccurs="0">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="NA" />
          <xsd:enumeration value="AND" />
          <xsd:enumeration value="OR" />
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:sequence>
  <xsd:attribute name="ID" type="xsd:ID" use="required" />
</xsd:complexType>
<xsd:complexType name="TOtype">
  <xsd:sequence>
    <xsd:element name="chargingFuncFormula" type="xsd:string" />
    <xsd:element name="chargingFuncParam" type="chargingFuncParamType" maxOccurs="unbounded" />
    <xsd:element name="dayTimeConstraint" type="dayTimeConstrType" />
  </xsd:sequence>
  <xsd:attribute name="ID" type="xsd:ID" use="required" />
</xsd:complexType>
<xsd:complexType name="chargingFuncParamType">
  <xsd:sequence>
    <xsd:element name="description" type="xsd:string" />
    <xsd:element name="unit" type="xsd:string" />
    <xsd:element name="value" type="xsd:nonNegativeInteger" minOccurs="0" />
  </xsd:sequence>
  <xsd:attribute name="ID" type="xsd:ID" use="required" />
</xsd:complexType>
<xsd:complexType name="penaltyType">
  <xsd:sequence>
    <xsd:element name="numbViolations" type="xsd:nonNegativeInteger" />
    <xsd:element name="period" type="xsd:string" />
    <xsd:element name="deductionFormula" type="xsd:string" />
  </xsd:sequence>
  <xsd:attribute name="whichSLO" type="xsd:IDREF" use="required" />
</xsd:complexType>
</xsd:schema>
Appendix B

EdgePol Language

B.1 Introduction

EdgePol, a policy language proposed in this appendix, aims to represent network policies. It has arisen from the need to describe, at level of metering systems management, high-level electronic tariffs and their counterpart SLA (Service Level Agreement) structures.

Simplicity and the hybrid concept of the declarative and imperative programming language paradigms form the fundamental characteristics of this language. Clearly, a programmer will find parts of both paradigms. The ability to provide structures for triggering events represents one of the key aspects when defining network policies in EdgePol.

EdgePol comes from three distinct other languages. The first dates back to 1960 when ALGOL 60 was first proposed [13]. From this language, some of the imperative structures have been incorporated. Secondly, the essence of using event, condition and action (ECA) statements, such as objectives, has been borrowed from the rule-based languages proposed by the active database research community [171, 131]. Last but not least, some of the advantages of C++ [161] have also been integrated.

The language itself has been kept as simple as possible. The reason is that one of the envisaged applications is the deployment of mobile code (Tariffs/SLAs) to devices which lack resources such as CPU, storage space and bandwidth. Therefore, the EdgePol compiler/interpreter should be light enough to run on such equipment.

As a result of this trade-off, some of the object-oriented paradigm concepts such as classes and inheritance do not appear in this current version. Needless to say, such features might be incorporated in the near future if the community of users expressed the desire to do so.
B.2 Measurement Philosophy

The organisation of this specification document is as follows. Each language structure is presented with its semantics and syntax in BNF notation [13]. When it is appropriate, examples will be given to show how to use such structures.

B.2 Measurement Philosophy

While some of the metering systems proposed in the literature collect network performance statistics at a very high level of aggregation, a few others present capabilities for collecting at finer granularities, often at packet level, offering the maximum level of detail a management application would expect. A mechanism designed from its inception to cover all the measurement granularity spectrum - from one extreme (fine grained) to another (coarse grained) - would be ideal.

To overcome this inflexibility, EdgeMeter introduces a novel notion of metering at virtually any granularity. The subsections that follow unveil the network field as a viable scheme to bring versatility in choosing, specifying and metering a variety of granularities. By tuning a few parameters, this scheme is capable of assembling most typical granularities such as packet-level [163], flow-based [44] and bulk-based statistics [107].

B.2.1 Network Field

We take a novel approach to the measurement framework for EdgeMeter. The field theory, in particular electromagnetic fields, is applied to our work in order to model the observed traffic as a composition of packet streams (or field lines). Field lines can be intercepted and aggregated by a meter system (i.e. an observer) to attain the intended granularity. The terms “field line” and “packet stream line” may be used interchangeably throughout this document for a stream of packets.

The set and arrangement of the packet stream lines we call a network field (Figure B.1). Packets transmitted from a sender toward a receiver establish packet stream lines between these two nodes. A stream of HTTP request packets, for instance, creates a HTTP field line connecting a client to a server. Though these lines have no reality attached, they do provide a nice way to typify some patterns of packets that a meter system detects.
B.2. Measurement Philosophy

The two nodes shown in Figure B.1 set up a conceptual network field between themselves by communicating streams of packets that follow a particular direction. As each packet has a pre-determined direction, the field created is termed a vector field. It consists of a distribution of vectors denoting the directions taken by packets; each vector is associated with a packet being transmitted toward a particular direction.

For the Internet, the direction vector of a packet is unidimensional. In a given network element, the direction of a packet is either forward (toward the receiver) or backwards (toward the sender). Nevertheless, new research has shown that the Internet topology may be mapped onto an Euclidean vector space [136, 124]. Each packet might travel toward a direction expressed in a vector along various dimensions. Regardless of the dimensionality, the proposed network field framework is still applicable to multidimensional spaces.

We also observe that different packet stream lines originate different types of network fields in this model. With this rationale, any mechanism that is capable of clustering packets in field lines also creates a network field with its characteristics 'inherited' from the clustered packets that form the stream lines. A set of TCP streams, for example, create a TCP type of network field. It is one of a metering system's tasks to identify packet stream lines that produce network fields, which are deemed relevant for a customer's application.

Perhaps the pioneering research to apply field theory control to network traffic management issues was reported in [56]. It is said that any two nodes exercise repulsive or attractive forces as a result of the field produced by different types of packet streams communicated between themselves. Unlike this proposal, this document is not concerned with issues that arise when counter-measures are put to use as a means of controlling the effects of the magnitude and the direction of fields forces imposed on the network nodes. We rather limit the scope to the specification and measurement of a network field as a flexible yet powerful tool to achieve virtually any metering granularity of interest.
B.2. Measurement Philosophy

In the context of measuring network fields, the interception loop is a surface that captures the network flux, which is the amount of network field that the loop's area can intercept.

Figure B.2: Example of TCP network field and flux

B.2.2 Network Flux: Measure of Network Field

The network field framework models the observed traffic as packet streams appear/disappear in a metering system due to traffic dynamics. This alone is not sufficient to achieve the intended granularity flexibility, though. The actual measurement of network fields is dealt with in this section.

Specification of the field itself is the first step toward the metering of a network field. Packets will be related by this specification to create field lines. It is important to note, however, that a network field depends on how an observation point, the meter system in our case, specifies it. A metering system may be interested in a network field produced by UDP packet streams only. Another system positioned at a different location in the network may wish to meter TCP network fields. To accommodate these different interests, each observation point specifies network fields that need to be measured.

As the second step to measure fields, an interception loop has to be specified. Graphically, it is a closed surface of any shape that intercepts packet stream lines. The loop measures a network flux, which is the amount of network field that the loop's area can intercept.

The example of Figure B.2 illustrates these new concepts. First, it shows a TCP network field created by all packets travelling from a source IP address, say 10.0.0.1, to any other host in the Internet. The protocol attribute value in each of the IP header is set to 6 (TCP protocol). The field lines visualised in the figure coincide with three TCP connections initiated by a host machine (e.g. source IP 10.0.0.1).

Second, a metering system (an observer) intersects the TCP network field by using a squared interception loop with area $A$ (B.2b). The vector $A$ represents the area and its direction is exactly the same as the normal vector of the loop plane. In the figure, vector $A$ is
perpendicular to the loop plane, so it is its normal vector. In the example, the measure of two intercepted field lines account for the network flux.

The role played by an interception loop surface is rather important to the specification of a metering granularity. The loop provides a primary aggregation for the reason that it bounds the field lines that should be measured. We term this type of aggregation the *spatial component* of a network flux. In the Internet metering setting, the interception loop area determines the number of packet stream lines that account for a network flux.

When a secondary aggregation is deemed necessary, for instance to group over the time packets of a subset of field lines, the *temporal component* of a network flux can be used. Mostly, the temporal component is applicable to bulk-based measurements that often aggregate data over a time interval.

Figure B.2c shows the use of the spatial and temporal components of a network flux. The squared loop base intercepts, i.e. aggregates, two field lines of the TCP network field (spatial component). In addition, the temporal component aggregates packets intercepted over a time interval by the rectangular surface shown (parallelepiped).

Finally, when a specification of the desired behaviour of a system is necessary, a *policy* can be created. In this language, a policy has to be specified in the form of a rule. The rule expresses three dimensions, such as the time of a happening in the system, in what circumstances the system should react to this happening, and what the reactions are.

As far as the remaining sections are concerned, the document will drive throughout the engineering aspects of representing network fields and flux within a general framework of policy structures.

### B.3 Lexical Structures

A token in EdgePol can be any of the following non-terminal terms: identifier, keyword, literal, operator, separator and comment. They will be described in more detail in the next sections. Figure B.6 presents the BNF notation for these lexical structures.

#### B.3.1 Identifier (< id >)

Defined as any sequence of non digits/digits not starting with a digit.
B.3. Lexical Structures

var.car
var.motorcycle
threeBicycles

Figure B.3: Example of identifiers

B.3.2 Keyword (<keyword>)

A keyword can be any word from Table B.1.

and, array, body, do, defaultevent, date, else, false, float, for, footer, header, if, int, long, not, or, on, record, true, type, policysetid, policysetowner, record, while

Table B.1: Keywords

B.3.3 Literal (<literal>)

A literal can be an integer number, string, float or boolean.

Integer Literal: -34333
String literal: "hello world!"
Float literal: 32.34444
Boolean literal: true

Figure B.4: Example of literals

B.3.4 Operator (<operator>)

In this category are included any operator of arithmetic, relational and logical expressions.

B.3.5 Separator (<separator>)

A separator, also known as a delimiter, has the purpose of separating structures within the language such as the end of a statement. Figure B.6 shows the possible symbols for a separator in EdgePol.
B.3.6 Comment (\(<\) comment \(>)\)

Comments are very useful when adding further explanation inside a policyset. They must start with /* and end by */.

/* This is a comment */

Figure B.5: Example of a comment

```plaintext
<nondigit> ::= a | b | c | d | e | f | g | h | i | j | k | l | m |
    n | o | p | q | r | s | t | u | v | w | x | y | z |
A | B | C | D | E | F | G | H | I | J | K | L | M |
N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<token> ::= <id>|<keyword>|<literal>|<operator>|<separator>|
<bracket> | <comment>

Figure B.6: BNF notation for the lexical structures
B.4 General EdgePol Structure

B.4.1 Policyset

A policyset in EdgePol is a program with mainly three sections: a header, a body and a footer. The header contains meta data about the policyset such as as the owner of the policyset, the identifier of the policyset, date of last update and so forth.

In contrast, the body section encloses the ECA (event-condition-action) statements that specify declaratively an active behaviour. The body allows the specification of action (functions/procedures) calls. However, their declaration within the respective code implementation is left to the footer part. Basically, only declarative statements within the body section are permitted. The idea is to describe when and what tasks to perform, leaving the underlying details on how to do so for the footer section. Finally, variables and new types may be created within this section.

The footer comprises all the auxiliary action implementation written with imperative statements. It is worthwhile mentioning that the footer is optional. On some occasions, the footer code will be loaded dynamically from external storage such as proxylets servers [80] [37]. Figure B.7 gives an overall structure of a policyset, while Figure B.8 shows the BNF notation.

```plaintext
header {
    /* Here goes the general information about this policyset */
    ....
}

body {
    /* Only declarative statements are allowed here */
    ....
}

footer {
    /* Imperative and declarative structures allowed */
    /* The code may be local or external code */
    ....
}
```

Figure B.7: Example of a policyset's parts
B.4. General EdgePol Structure

B.4.2 Policyset Header

The header part has to begin with the keyword header. The header-part structure (Figure B.10) holds the required and additional headings enclosed by brackets {...}. The required headers are composed of the identifier and the owner of a policyset. The date of last update and environment-specific tags account for the additional headers. Figure B.9 presents an example with these two types of header information.

```
header {
  policysetid tariff_diffserv/142/1.0; /* Tariff no.142 v1.0 */
  policysetowner UCL; /* Owner of the tariff */
  date 12/12/2000; /* Date of the last modification */
}
```

Figure B.9: Example of the header section

```
<policyset> ::= <header><body> | <header><body><footer>

Figure B.8: Policyset: BNF Notation

B.4.3 Policyset Body

Dictating what is required and when without bothering with underlying details of how to perform the tasks is very desirable in policy languages [58]. Thus, allowing declarative statements
in the body part places this language in a context where hiding implementation details is a strong requirement. As a result, the programmer should incorporate any of the implementation specifics into the footer section using imperative statements.

The EdgePol takes into account the common properties of declarative languages when allowing three types of statement: expression, ECA or any declaration such as variables or types. Action implementation through function declaration is completely forbidden inside the policyset body. Action implementation must be declared in the footer section. However, action instantiation is well supported within the body.

Following up the above design decision, the example shown in Figure B.11 presents a fictitious tariff applied to the IETF DiffServ architecture [18] [125]. Figure B.12 gives the syntax for the body section.

```plaintext
body {
    /* Tariff algorithm applied to a DiffServ based service */
    float pi := 0.7;    /* price for the PHB EF */
    float p2 := 0.5;    /* price for the PHB AF11 */
    float p3 := 0.4;    /* price for the PHB AF12 */
    float p4 := 0.3;    /* price for the PHB AF13 */
    float price := 0.00; /* current price to pay */
    int report_interval := 120; /* report interval from the event */
    /* source set to 2 mins */
    on (dscodepoint.ef, "meter://128.16.6.143/"+report_interval)
    if (dscodepoint.ef == true)
        do {
            price := price + get-price(pi);
        }
    else if (dscodepoint.af11 == true)
        do {
            price := price + get-price(p2);
        }
    else if (dscodepoint.af12 == true)
        do {
            price := price + get-price(p3);
        }
    else if (dscodepoint.af13 == true)
        }
}
```

Figure B.11: Example of the body section

```plaintext
<body-part> ::= <body-stm> | <body-part> <body-stm>
<body-stm> ::= <exp-stm> | <ECA-stm> | <body-declaration-stm>
```

Figure B.12: Syntax of the body section in BNF notation
B.4.4 Policyset Footer

This optional part is meant to be entirely composed of imperative code. It is structured as a sequence of action implementation (functions/procedures) which assists the declarative statements inserted in the body section.

Furthermore, the programmer may wish to link external modules held in external repositories (e.g. proxylet servers) into this part. Consequently, the footer presents itself with two notions of code: local, which is inserted in the footer, and external, which is downloaded from external servers.

The example in Figure B.13 gives the footer section for the DiffServ based tariff (Figure B.11).

```c
footer {
    /* Footer section that contains the implementation for the action get_price with imperative statements */

    float get_price(float price_constant) {
        get_price := defaultevent.pkts_transf * price_constant;
    }
}
```

Figure B.13: Example of the footer section

```plaintext
<footer-part> ::= <block-stm> | <footer-part><block-stm>
<block-stm> ::= <block-stm> <stm>
```

Figure B.14: Syntax of the footer section in BNF notation

B.4.5 Network Field/Flux Specification

The concepts of network field and its measure, the network flux, have been introduced in the previous sections. We now focus our attention on how these two concepts may be characterised in engineering terms. This section presents an overview of two templates: one that specifies a network field, the other that describes a network flux.

- **Network Field Template**: it defines the characteristics of a network field that will even-
B.4. General EdgePol Structure

networkfieldtemplate NField1 {
    type {
        srcipaddr { (eq, 10.0.0.1) } and
        protocolid { (eq, tcp) };
    }
    fieldline { newline };
    activeperiod {
        starttime { }; 
        stoptime { }; 
    }
}

Figure B.15: Example of a network field template

- **Network Flux Template**: it describes the specifics of how a network field is to be measured. The template includes some attributes of the interception loop area, a time interval \(\Delta T\) and the metrics to be metered.

In order to specify the TCP network field of Figure B.2(a), the template may be the one shown in Figure B.15. The type of network field is described by the attributes within the *type* statement; it provides a form of packet stream filtering. Therefore, some filter expressions with attributes obtained from the packet header/content are allowed. The relational expression in the template example states that the network field of interest is TCP (*protocolid* = 6) and it originates from the IP address 10.0.0.1 (*srcipaddr* = 10.0.0.1).

The *fieldline* attribute determines the way new packet stream lines should be arranged. Many field lines can be spawned out of a network field template. They may be grouped in single or multiple field lines. Wherever it is necessary to distinguish between packet stream lines, the keyword *newline* can be used. In the example, each packet stream that matches the field *type* results in a new field line. In contrast, the keyword *sameline* groups the packet streams that match the field type in a single field line.

Suppose a packet has 10.0.0.1 as its source IP address and destination IP 10.0.0.20. Streams of this packet will be regarded as a new field line when the template of Figure B.15 is applied. In addition, another packet with identical source IP address but different destination, say 10.0.0.111, will generate another new field line. In contrast, it would be a single field line to account for these two packet streams had the keyword *sameline* been used instead.
The last attribute inspected, the **activeperiod**, specifies a time period in which a network field is active and operating. This meaningful attribute is largely used to stipulate when a measurement is to be taken as this requires an active network field. Such a time representation is grouped in two categories. The absolute time, the first one, specifies precisely a list of one or more time intervals. In contrast, the periodic representation allows easy specification of intervals for a network field life-cycle. In the example, the active period of \( NField1 \) includes all of time since the attributes **starttime** and **stopetime** were left empty.

```
networkfluxtemplate NFlux1 {
  networkfieldtemplate { NField1 };
  interceptionloop {
    dstipaddr { (eq,10.0.0.111) and (eq,10.0.0.20) };
  };
  deltat { (50, seconds) };
  metric { metric1 };
  meterprofile { mpl };
  report { };
}
```

**Figure B.16: Example of a network flux template**

Figure B.16 shows a possible network flux template for the example of Figure B.2c. The **networkfieldtemplate** attribute lists the fields in which this network flux will be measured. Similar to the **type** attribute of a network field template, the **interceptionloop** holds filter expressions that combine attributes and values captured from packet headers and content.

It is important to emphasise though that network flux mostly depends on this set of filter expressions to correctly intercept a network field. Although there exist significant differences, an analogy to vector fields in Physics will help to illustrate the **interceptionloop** attribute's relevance.

Filter expressions can be graphically regarded as the angle \( \theta \) between the field lines and the area vector \( \vec{A} \). If the field lines are perpendicular to the interception loop, i.e. \( \theta = 0 \) as in Figure B.2b), the network flux comprises all field lines intercepted. However, as more filter expressions are included in the **interceptionloop** attribute, the more inclined the loop plane becomes. It may well reach the point where no field line is intersected, i.e. the field lines are parallel to the interception loop. The angle \( \theta \) between the field lines and the area vector \( \vec{A} \) is \( \pi/2 \) in this case.
B.4. General EdgePol Structure

The time interval ($\Delta T$) is the time component of a network flux. It is specified as a tuple of a value and its unit; the value is 50 seconds and seconds its unit in Figure B.16. Packets or bytes are examples of other acceptable units. Packets that belong to field lines intercepted by a loop area will be aggregated over a time interval set in the \textit{deltat} attribute (B.2c).

The \textit{metric} attribute determines what has to be metered within a network flux. How such a measurement should take place is described in the internals of the metric specification ($\text{metric1}$), which is not shown in this example (refer to Appendix B for more detail). The implementation code of $\text{metric1}$ is a module that is plugged in EdgeMeter.

Finally, the \textit{meterprofile} contains properties of the meter - for instance the meter location and the protocol to communicate - where a network field will be measured.

B.4.6 Generalisation of Granularity

With the “spacetime” nature of a network flux, we can generalise the type of granularity by simply adjusting the values of the template attributes.

Packet level metering requires capturing every packet for further analysis. Because neither type of aggregation, be it spatial and temporal, is in fact necessary for packet level metering [163], this granularity can be obtained when a network field is formed by a single field line. This is specified by leaving the attribute \textit{type} of a network field template empty. In contrast, the interception loop area should be large enough, say that it tends to infinity, in order to intercept this single line. Since no temporal aggregation is necessary, the time interval tends to zero ($\Delta \to 0$). Figure B.17a shows the packet level granularity.

As for a flow-based granularity [30], the \textit{type} attribute embraces filtering expressions that use the microflow attributes: \textit{srcipaddr}, \textit{dstipaddr}, \textit{srcport}, \textit{dstport} and \textit{protocolid}. This will
Table B.2: Examples of granularity and their specification attributes

<table>
<thead>
<tr>
<th>Granularity type</th>
<th>Field Type</th>
<th>Interception Loop</th>
<th>Time Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet level</td>
<td>type is empty</td>
<td>$\text{interceptionloop}$ tends to $\infty$ ($A \rightarrow \infty$)</td>
<td>$\text{deltat}$ tends to zero ($\Delta \rightarrow 0$)</td>
</tr>
<tr>
<td>Flow-based</td>
<td>type has the microflow attributes</td>
<td>$\text{interceptionloop}$ might contain any value</td>
<td>$\text{deltat}$ greater than zero ($\Delta &gt; 0$)</td>
</tr>
<tr>
<td>Bulk-based</td>
<td>type is empty</td>
<td>$\text{interceptionloop}$ is $\infty$ ($A \rightarrow \infty$)</td>
<td>$\text{deltat}$ is much greater than 0 ($\Delta \gg 0$)</td>
</tr>
</tbody>
</table>

arrange field lines that resemble flows. The $\text{interceptionloop}$ attribute of a flux template might contain any value.

In some circumstances, however, the value of an interception loop will depend on whether the network flux of interest is a measure of a subset or the whole field. Suppose a network field covers all the flows initiated from the source IP address 10.0.0.1 but the intended network flux is only applied to flows destined to the IP address 10.0.0.35. In this case, the interception loop should contain a filtering expression to select packet stream lines that match the destination address 10.0.0.35 out of all lines originated from the address 10.0.0.1.

The time interval, the $\text{deltat}$ attribute, should be greater than 0 for flow granularity. It states for how long packets should be aggregated in a packet stream line (or flow) (Figure B.17a).

Finally, the simplest form of bulk measurements may be gathered by specifying an empty network field type, hence creating a single field line. The interception loop area is set to $\infty$, so that there is enough area to intercept the single field line. Unlike packet level specification, the time interval of a network flux template in bulk granularity does matter. As such, it should be much greater than 0 ($\text{deltat} \gg 0$) but naturally tuned to the intended interval of aggregation.

Table B.2 summarises some of the cases in which distinct specifications accommodate packet level, flow-based and bulk measurements.

B.4.7 Metric

A metric as defined in this document outlines a measurement template which can hold functional or non-functional requirements. Such a structure presents the properties to be measured along with how to get the measurement. It might be easy to think of a metric as the written specification of a process to be performed. By analogy, a metric is what a binary code is in a
computer system, whereas a process (binary code in execution) relates to a measurement.

The report interval property represented by the field reportinterval allows the creation of sophisticated data report mechanisms. Events might be combined in order to create compounds of interest. The basic format for a report interval is similar to the format of the report field of a session template.

Two parts compose a metric. First, the metric spec describes the requirements for a measurement without any implementation detail. Secondly, the metric impl structure provides the implementation code for the metric considered. This separation of what a metric is and how it is realised follows the original declarative notion behind EdgePol. Thus, it is perfectly sensible to have different metric implementations for the same metric specification. Such a feature will allow the reuse of downloadable code such as that conveyed through a programmable network.

A metric spec has the following fields: (a) a version string used to keep control of a set of metrics in a system; (b) a pmf standing for 'pattern matching filter' which is valuable to constrain the domain of events related to this metric; and (c) a reportinterval field that describes how the measurements taken should be reported by the runtime system.

However, the specification of a metric does not state how it is implemented. Thus, the second structure termed metricimpl comes to the stage. The structure name links to the correspondent metric specification (metricspec). The version field aims to describe the version of this implementation, which might differ from the version of the metric specification. The implementation encode (implemcode) states whether the metric algorithm is internal or external and gives further hints to get the metric code. The encode is given as a string with attributes of the metric algorithm. This field aims to introduce details on how to process the metric algorithm which can be written in EdgePol, or any other suitable programming language.

Figure B.18 shows an example of a metric spec structure with its correspondent implementation. It is important to note that the two structures are tied together by their name. The metric tokenbucket specifies a token bucket structure to hold the number of ECN marks (Explicit Congestion Notification) [145]. Such a structure might be configured accordingly to notify when there is congestion in the network. The $R$ parameter is the token rate, whereas the $B$ is the size of the token bucket. Therefore, in the example, packets carrying an ECN mark would flag the application when the marks arrival rate does not conform to the desired value for the token bucket rate ($R$).
B.4. General EdgePol Structure

metricspec ECNtokenbucket {
  version 1.0/UCL;
  type record of {R:float, B:long};
  pmf {};
  reportinterval {starttime=*.10; dom=*; month=; dow=; stoptime=120s}
}

metricimpl ECNtokenbucket {
  version 2.0/UCL;
  encode {External;
    source:'proxylet://192.168.0.1/metrics/tokenbucket'};
  algorithm {};
}

Figure B.18: Example of a metric with its implementation

B.4.8 Meter Profile

A meter profile contains the properties such as the location and the protocol to be used in order
to communicate to the meter. One important aspect lies in how the meters are configured. First,
they can be configured through unicast. In such a case, the meter profile will describe a single
meter. In contrast, when the configuration is done via multicast, the meter profile will specify
a set of meters which are located somewhere over the multicast tree. Figure B.19 presents an
example with one meter profile for unicast (mpl) and another for multicast (mp2).

meterprofile mpl {
  version '2.0/UCL/London';
  meterid 'meter1_unicast';
  meterhost '172.16.0.1/32';
  meterprot 'DMPv1.0';
}

meterprofile mp2 {
  version '1.0/Customers';
  meterid 'meter_multicast';
  meterhost '224.2.17.12/127';
  meterprot 'DMPv1.0';
}

Figure B.19: Example of a meter profile
B.5 Specific EdgePol Constructs

B.5.1 Expressions (<exp>)

Arithmetic Expression

Arithmetic expressions are statements which aim to calculate numerical values. This computation is assembled from parameters such as single numeric values, returned values from functions, and the value associated with the variables. Arithmetic operators are properly inserted in order to construct the expressions.

Furthermore, the evaluation of an expression produces the appropriate type according to the types of the operands. They can be integers (int and long) or float types. The precedence order when evaluating such an expression is given by the following rules:

1. Left to right within the same group of operators

2. Group of multiplication operators: *, /,

3. Group of addition operators: +, -

It is important to note that parentheses might be used for the purpose of changing the precedence order of the operators. Figure B.20 captures some valid expressions in EdgePol. Additionally, Figure B.21 presents the syntax of an arithmetic expression in BNF notation.

```plaintext
price / price_unit * coefficient + coefficient2
(price / price_unit * (coefficient + coefficient2))
x * y + 45.3 * h % 3
```

Figure B.20: Example of valid expressions
### B.5. Specific EdgePol Constructs

#### Boolean Expression

Boolean expressions are used for evaluating logical values. Two built-in boolean constants are available in EdgePol: `true` and `false`. Any nonzero number is considered true, and a zero value is false.

The precedence of boolean operators observes the rules as follows:

1. Calculation of arithmetic expressions
2. Relational operators: `<, <=, ==, >=, >, !=`
3. Logical operator `not`
4. Logical operator `and`
5. Logical operator `or`

Parentheses can be placed in the expression to force the desired order of operators. The syntax of a boolean expression is shown in Figure B.22.

```plaintext
<boolean-exp> ::= <boolean-and-exp>|<boolean-exp> or
<boolean-exp> ::= <boolean-negative-exp>|<boolean-and-exp> and <boolean-negative-exp>
<boolean-negative-exp> ::= <boolean-final-exp>|not <boolean-final-exp>
<boolean-final-exp> ::= <logical-value>|<variable>|<function-caller> <relation>|<boolean-exp>
```

Figure B.22: Syntax of a boolean expression in BNF notation
Assignment Expression

Values in EdgePol might be assigned to an identifier either directly or as the result of other expression evaluations. When this is the case, an identifier is required on the left hand side of the operator (\( := \)), whereas an expression should be written on the right hand side of the operator. The expression should afterwards evaluate to a value which is finally stored into the identifier. Figure B.23 presents some examples, while Figure B.24 gives the syntax for this type of expression.

\[
\begin{align*}
p1 & := 0.7; & /* price for the DiffServ PHB EF */ 
p2 & := 0.5; & /* price for the DiffServ PHB AF */ 
a & := (c\times b+3) 
\text{condition} & := (a^n b)
\end{align*}
\]

Figure B.23: Example of assignment expressions

\[
<\text{assignment-exp}> ::= <\text{id}> := <\text{exp}>
\]

Figure B.24: Syntax in BNF for an assignment expression

B.5.2 Statements (\(<\text{stm}>\))

The statements in EdgePol will normally be executed as they appear in the policyset. The event-condition-action (ECA) clauses correspond to the exception case. They will be performed in a top-down order as normal. However, they are associated with pre-registered events which will probably trigger more than one action simultaneously.

Several types of statement are currently supported. They have originated from the mixture of syntax from C++ [161] and ALGOL60 [13].

First, an expression statement is either one arithmetic expression, boolean expression or assignment expression. In contrast, iteration statements comprehend structures that create a block of repetitions assuming certain constraints. The \textit{while} and \textit{for} are good examples of iteration statements.

Furthermore, conditional statements include the \textit{if..else..} clauses. In fact, they are a subset
of ECA statements. The latter states what events, conditions and actions the system should react to. The event causes the ECA to be triggered. The condition is thereafter checked. After that, the actions are executed (fired) when the ECA statement is triggered and its condition is true. The event may have its internal and external sources. For instance, a network EdgeMeter (device that measures network traffic) may generate both types of event depending on its location.

Last but not least, the declaration statements associate properties with the identifiers.

```
<stm-seq> ::= <stm> | <stm-seq> <stm>
<stm> ::= <exp-stm>|<iteration-stm>|<conditional-stm>|<ECA-stm>|<declaration-stm>
```

Figure B.25: Syntax for statements

### Expression Statements

Refer to Section B.5.1 for further explanation and examples. The syntax of expression statements is presented in Figure B.26.

```
<exp-stm> ::= <arithmetic-exp>|<boolean-exp>|<assignment-exp>
```

Figure B.26: Syntax of expression statements

### Iteration (loop) Statements

The *for* and *while* are the two main blocks within this category. The *for* clause executes the sequence of statements zero or more times. The controlled variable makes the selection of the execution frequency. As for the *while* statement, repetition occurs until the condition evaluates to true. In the next cycle, the condition is then reevaluated.
B.5. Specific EdgePol Constructs

for x := 1 to 20 {
    /* Block statement here */
}

while (a>=2) {
    /* Block statement here */
}

Figure B.27: Example of iteration statements

<iteration-stm> ::= <for-stm> <while-stm>
<for-stm> ::= <for-clause> <stm-seq>
<for-clause> ::= for <variable> := <for-list> <stm-seq>
<while-stm> ::= while (  condition )  do <stm-seq>

Figure B.28: Syntax of iteration statements in BNF

Conditional Statements

When a condition is evaluated to true, the < action — stm > is accomplished. The true in this context means any nonzero value. In contrast, zero value constitutes the logical false.

When the if clause is accompanied by the else clause, the actions within the else are executed providing the condition is false. Furthermore, it is possible to create nested if statements. Figure B.29 shows an example of how to use the conditional statements. Figure B.30 gives the BNF syntax.

if ( a >= b) do {
    /* Statements performed when the condition (a >= b) is true */
} else {
    /* Statements performed otherwise */
}

Figure B.29: Example of a conditional statement
B.5. Specific EdgePol Constructs

<conditional-stm> ::= if ( condition ) do <action-stm> |
                   if ( condition ) do <action-stm> else <action-stm>

Figure B.30: Conditional statements: Syntax in BNF

B.5.3 Policy with the ECA (Event-condition-action) Model

In this section, a framework to represent a policy using the dimensions of ECA rules will be used. This model has been put forward in [131] and was contextually adapted to cope with the dynamics of a SLA/Tariff monitoring system. As far as the notation is concerned, the symbol \( \subset \) is used to denote that an attribute can take on more than one of the values presented, whereas \( \in \) indicates a list of alternatives [131].

Event

The state of the underlying distributed system has to be taken into consideration. Therefore, there will be two types of event state. One is the global event which refers to the global state of the system, such as when the communication links used to report the measured data are down. In contrast, the local event holds the state information at the local level on a single system. How to keep the global state information updated is a challenge. To consider this, it will be assumed that the global state is learnt from the interaction between the elements of the distributed system. Table B.3 shows the possible states associated with an event.

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>( \subset {\text{Global, Local}} )</td>
</tr>
</tbody>
</table>

Table B.3: Applicable states for an event

An event might be single or compound. It is single when raised by one occurrence that belongs to the Source category. In contrast, compound events are the conjunction of single events using logical operators such as and, or, not. Table B.4 shows the type and source of an event.

The source category arranges events in the following groups:

- Network Flux: it includes events such as the creation, modification or deletion of a network flux. It is important to note that the network flux concept is general, as such it encapsu-
lates any packet-related events for example. Additionally, protocol-related events (e.g. IGMP join messages, RSVP setup message, DHCP allocation message) are also included in this category based on the same grounds.

- **Clock:** temporal events might cause an ECA rule to be triggered at an absolute time (e.g. 21/01/2004 at 18.00), or at periodic intervals (e.g. every 10 seconds) [141] [142].

- **External:** the event is raised by an occurrence outside the EdgeMeter system.

- **User Defined:** an application declares an event $E$ (e.g. user authentication) which triggers some policy rules. The application in this case takes care of notifying the event to the event detector. This category allows flexibility for creating events tailored to the application needs [171].

- **Exception:** events in this category are raised as a result of some error or fault in the system. Suppose the situation where a requested metric module is not locally available. This may trigger an exception that leads to a search for the module.

- **Communication:** events raised by the communication mechanisms used in the system. To name a few, mechanisms such as the distribution of service specifications, gathering of measurement data and so forth are possible sources.

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>⊆ {Single, Compound}</td>
</tr>
<tr>
<td>Source</td>
<td>⊆ {Session, External, User Defined, Exception, Communication}</td>
</tr>
</tbody>
</table>

Table B.4: Source and type of events

**Condition**

A condition is checked when an event occurs. The context indicates the state of the system in which the condition is evaluated (Table B.5). It may possibly be associated with four states:

- **System.Event:** the system when the event took place;

- **System.Condition:** the system when the condition was evaluated;

- **System.Action:** the system when the action was executed;
### Action

An action is a set of tasks to be executed when the event is triggered and the condition evaluates to true. The Options property shown in Table B.6 represents the tasks allowed for execution. In such a case, any internal function (declared in the footer part) or external modules can be declared. The semantics of the action block are that any code has to be declarative. As a result, the statements should be calls/invocations to internal or external codes. The footer confines all the implementation details.

\[
\text{<action-stm>} ::= \text{do <stm>}
\]

Figure B.31: Syntax for an action in BNF

<table>
<thead>
<tr>
<th>Property</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options</td>
<td>{Internal functions, External modules}</td>
</tr>
</tbody>
</table>

Table B.6: Action

\[
\text{<ECA-stm>} ::= \text{on (event) <conditional-stm>} | \text{on (event)<action-stm>}
\]

Figure B.32: ECA Syntax

### ECA (Event-condition-action) Statements

This structure plays an important role in the language. The event, as already explained in the previous section, is a set of properties such as its name, source and the report interval. The event is registered within the source; each event instance sent by the source fires the statements associated with it. When conditions are presented, they are sequentially evaluated. For any condition that happens to be true, the action statements are executed.
B.5. Specific EdgePol Constructs

B.5.4 Declaration

The declaration part provides the means for type, variable and function definitions. Figure B.34 shows the syntax for any declaration in EdgePol.

\[
<\text{declaration-stm}> ::= <\text{type-declaration}|<\text{var-declaration}|<\text{function-declaration}>
\]

Figure B.34: Syntax for a declaration in EdgePol

Type Declaration

It is intended to associate identifiers with a given type. For this reason, EdgePol supplies built-in types such as integer (int), long integer (long), float number (float) and string. Further, records and arrays are also supported. The record fields should be enclosed within delimiters (\{...\}), with each field specified by the structure identifier: type. The array of any type may be constructed with \text{array of < type >}. Some examples of type usage can be seen in Figure B.35. Figure B.36 gives the syntax in BNF for a type declaration.

\[
\text{type newtype} = \text{int};
\text{type employee_record} = \text{record of} \{
\text{string name} := "";
\text{long ID} := 0;
\}\n\text{type vector_integer} = \text{array of int};
\]

Figure B.35: Example of type declarations
B.5. Specific EdgePol Constructs

\[
\begin{align*}
\text{<type-declaration>} & ::= \text{type <id> = <type>} \\
\text{<type>} & ::= \text{<local-type>} \mid \text{<record-type>} \mid \text{<array-type>} \\
\text{<local-type>} & ::= \text{int} | \text{long} | \text{float} | \text{string} \\
\text{<record-type>} & ::= \text{record of <record-fields>} \\
\text{<record-fields>} & ::= \text{id} : \text{<local-type>} | \text{id} : \text{<array-type>} | \text{id} : \text{<record-fields>} \\
\text{<array-type>} & ::= \text{array of <local-type>} | \text{array of <record-type>}
\end{align*}
\]

Figure B.36: Syntax of type declarations

Var Declaration

Each variable must be declared with a type, be it a built-in type or a user own-defined type. The variable may be initialised during the declaration through the assignment operator (\(=\)).

\[
\begin{align*}
\text{int a := 3;} \\
\text{float b;} \\
\text{string c := 'Hello';}
\end{align*}
\]

Figure B.37: Example of variable declarations

\[
\begin{align*}
\text{<var-declaration>} & ::= \text{<type> <id>} | \text{<type> <id> := <exp>} .
\end{align*}
\]

Figure B.38: Syntax of the variable declaration

Action Declaration

Any function should return one value. When this does not happen, the function is a procedure.

\[
\begin{align*}
\text{<function-declaration>} & ::= \text{type <id>\{ <param-list> \}} | \text{id}\{ <param-list> \} \\
\text{<param-list>} & ::= \text{<param>} \mid \text{<param-list>, <param>} \\
\text{<param>} & ::= [\text{id} : \text{<type>}] \\
\end{align*}
\]

Figure B.39: Syntax of function declarations
References


References


References


References


References


References


References


References


References


