Scalable Quality of Service Support for Mobile Users

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von
Günther Stattenberger
aus Deutschland

Leiter der Arbeit:

Prof. Dr. T. Braun

Institut für Informatik und angewandte Mathematik

Scalable Quality of Service Support for Mobile Users

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von
Günther Stattenberger
aus Deutschland

Leiter der Arbeit:

Prof. Dr. T. Braun

Institut für Informatik und angewandte Mathematik

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Bern, 19.12.2002

Der Dekan:
Prof. Dr. G. Jäger

Acknowledgements

This thesis has received the generous support of many individuals. I am grateful for this support and would like to thank the following persons in particular.

First of all I owe a debt of gratitude to my supervisor, Prof. Dr. T. Braun, who gave me the opportunity to do this research; he provided me with many new ideas and detailed comments on the topics of this thesis. Prof. Dr. Braun also encouraged and motivated me to publish my research work and gave me the opportunity to present the work on several conferences, where I learned many interesting new aspects.

I must also thank Prof. Dr. Stefan Fischer, who was willing to spend a lot of time expertising my work and Prof. Dr. Oscar Nierstrasz for serving as co-examinator.

Acknowledgement is also due to the members of the RVS group, especially Roland Balmer, Florian Baumgartner, Silvia Stattenberger, and Attila Weyland, for good discussions and the pleasant and stimulating working environment and to Ruth Bestgen for administrative support.

Finally I want to thank my family, especially my parents, who gave me the opportunity to study in the first place. Special gratitude goes to my wife, Silvia, whose continuous support and help I needed to complete this work.

The work presented here has been performed during a project financed by NEC Europe Ltd. and a project financed by the Swiss National Science Foundation.

Contents

Introduction **1**

Difficulties of Providing QoS in Mobile Environments 3

Approaches for Solving the Different QoS Problems 6

Bandwidth Brokers for Differentiated Services 6

Quality of Service Support in Wireless Networks 6

Authentication and Authorisation 7

Handover Optimisation in Mobile IP 7

QoS Signalling Protocol 7

Admission Control 8

Hierarchical Bandwidth Brokers 8

QoS-aware Handovers 9

Summary 9

Contribution of this Thesis 9

Structure of this Thesis 11

I Related Internet Standards and Research Work **13**

Overview **14**

1 Mobile IP **15**

1.1 Terminology 16

1.2 Mobile IP Operation 16

1.2.1 Registration 17

1.2.2 Routing 17
1.3 Handover in Mobile IP 19
1.3.1 Smooth Handover 20
1.3.2 Fast Handover 21
1.3.3 Effects of Layer 2 Technology 24
1.3.4 QoS-Aware Handover 24

2 Quality of Service 27
2.1 Integrated Services 28
2.1.1 Components of an Integrated Services Router 28
2.1.2 The Resource Reservation Protocol (RSVP) 30
2.1.3 RSVP and Mobile IP Interworking 33
2.2 Differentiated Services 35
2.2.1 Differentiated Services Codepoints 35
2.2.2 Service Level Agreements 36
2.2.3 Per Hop Behaviour (PHB) 36
2.2.4 Per Domain Behaviour (PDB) 38
2.2.5 DiffServ Components 39
2.2.6 DiffServ Router Types 42
2.2.7 Performance Evaluation of Differentiated Services 44
2.3 QoS-enabled IEEE 802.11 Wireless Access Networks 45
2.4 Signalling Protocol for Differentiated Services 47
2.4.1 Cross Application Signalling Protocol (CASP) 48
2.4.2 Context Transfer Protocols 49
2.4.3 Conclusion 50

3 Network Management 51
3.1 Network Management Architecture 51
3.2 ISO Network Management Model 52
3.2.1 Performance Management 52
3.2.2 Configuration Management 53

3.2.3 Accounting Management 53
3.2.4 Fault Management 54
3.2.5 Security Management 54
3.3 The IEEE P1520 Standards Initiative 55
3.4 Network Management using ScriptMIB 56
3.5 Bandwidth Broker Architectures 57
3.5.1 Overprovisioning 58
3.5.2 Hierarchical Bandwidth Broker Structures 59
3.5.3 Management of Heterogeneous Networks 61
3.5.4 Agent-Based Management 62
3.5.5 Conclusion 62
3.6 Bandwidth Broker Implementations 63
3.6.1 BB Implementation of the University of Kansas 63
3.6.2 Implementation of the Two-Tier Architecture by UCLA . 64
3.6.3 Implementation of Policy Based Networks by TUT 64
3.6.4 Implementation of a vendor-independent BB 64
3.6.5 Conclusion 65
3.7 Authentication, Authorisation, Accounting 65
3.7.1 AAA Architecture 66
3.7.2 System Components of the AAA Server 67
3.7.3 AAA for the Mobile Internet 69
3.7.4 Policy-based Approaches 70

Summary **71**

II Implementation and Performance Evaluation of a Diff-Serv Implementation for Linux Routers **73**

Overview **74**

4 DiffServ Implementation **75**
4.1 Traffic Conditioner Modules for the Linux Kernel 76
4.2 DiffServ Router Architecture 77
4.3 Linux Router Configuration 80
4.3.1 Ingress Router 80
4.3.2 Interior Router 81
4.3.3 Egress Router 81

5 Performance Results **85**
5.1 Test Network and Evaluation Methods 85
5.1.1 Test Network Design 85
5.1.2 Performance Measurement Procedures 86
5.2 Results 88
5.2.1 Tests without DiffServ 88
5.2.2 UDP experiments 89
5.2.3 TCP experiments 94

Summary **97**

III Design and Implementation of a Bandwidth Broker **99**

Overview **100**

6 A Generic Management API for QoS Support **101**
6.1 Data Classes for the API 104
6.2 The `Router` Class 106
6.3 The `Interface` Class 107
6.4 The `TrafficConditioner` Class 108
6.5 Dynamic C++ Classes 109
6.5.1 Dynamic Class Loading 110
6.5.2 Autoregistration 110
6.5.3 The Factory of Dynamic Classes 111

6.6 API Implementation for Linux Routers 111
6.6.1 API Child Classes for a Linux Router 112
6.6.2 Linux Router Class 112
6.6.3 Ethernet Interface Class 113
6.6.4 Traffic Conditioners 113
6.6.5 Other API Implementations 113

7 Bandwidth Broker 115
7.1 The Topology Database 117
7.1.1 Topology Auto-detection 118
7.1.2 The Naming Service 118
7.2 Bandwidth Management and Admission Control 119
7.3 Bandwidth Broker API Commands 119
7.3.1 Flow Establishment 120
7.3.2 Flow Deletion 121
7.3.3 Flow Modification 121
7.4 Communication Subsystem for the Bandwidth Broker 122
7.4.1 The Communication Server 122
7.4.2 The Communication Client 123
7.4.3 Serialisation 123
7.4.4 The Bandwidth Broker — Router Communication 124
7.4.5 The Bandwidth Broker — User Communication 125
7.5 Client Software and User Interface 126
7.6 Performance of the Architecture 126
7.6.1 Evaluation Scenarios 127
7.6.2 Results 128
7.7 Multi-ISP Support 135
7.7.1 Backward Reservation 137
7.7.2 New Architecture Components 138
7.7.3 Signalling Protocol 140
7.7.4 Multi-ISP-Capable Broker Commands 142

7.8 Hierarchical Bandwidth Brokers 145
7.8.1 Advantages of a Broker Hierarchy 145
7.8.2 Operation of a Broker Hierarchy 146
7.8.3 Conclusion 148

Summary **149**

IV Providing Quality of Service to Mobile Users **151**

Overview **152**

8 Mobile-Specific Extensions for the Bandwidth Broker **153**
8.1 Scenario Description 154
8.1.1 Negotiation of a new SLS 154
8.1.2 Migration to a new access network 155
8.2 Extensions to the Bandwidth Broker API 155
8.2.1 Request a Flow List 156
8.2.2 Automatic SLS Transfer 157
8.3 Inter-Domain Broker Signalling 158

9 User-initiated Handover **159**
9.1 Handover in IEEE 802.11 Wireless Access Networks 161
9.1.1 Link layer handover procedure 161
9.1.2 Measurement categories 162
9.1.3 Decision points 162
9.2 Link Layer Handover Statistics 164
9.2.1 Equipment 165
9.2.2 Environment 166
9.2.3 Measurements 166
9.2.4 Analysis 166
9.3 Implementation and Operation of the WLAN Monitor 167

9.3.1 Wireless LAN Monitor operation 169
9.4 Range-Based Bandwidth Allocation 171
9.4.1 Scenario 172
9.4.2 Renegotiation of the SLS 172

10 An AAA Architecture Extension **175**
10.1 Mobile AAA with Foreign Agent 176
10.2 Integration of SLP and the AAA Architecture 177
10.3 Mobile IP Node Negotiation Procedure for DiffServ 180
10.3.1 Initial Foreign Network Access 181
10.3.2 QoS Negotiation 185

Summary **186**

Conclusion **187**
New Components 187
Signalling Messages in the Home Network 189
Signalling Messages in the Foreign Network 190
Timing Considerations 194
Topology Updates 195
Outlook and Future Work 196

List of Abbreviations **199**

Bibliography **203**

List of Figures

I.1 A Mobile User with QoS Requirements 4

1.1 Mobile IP Triangular Routing 18

2.1 RSVP in Hosts and Routers [21] 29
2.2 Diagram of a Normal RSVP Reservation Setup and Clearance . . 31
2.3 Split Point in a RSVP Multicast Connection 32
2.4 DiffServ Codepoint in the IP Header [124] 35
2.5 Combination of DiffServ Components in a Router [16] 39
2.6 DiffServ Router Major Functional Blocks [12] 41
2.7 A simple DiffServ Network 43

3.1 A Typical Network Management Architecture [76] 52
3.2 Distributed AAA Model 67

4.1 Ingress Router Implementation Architecture 79
4.2 Interior Router Implementation Architecture 79
4.3 Ingress Router Configuration 80
4.4 Ingress Router Configuration Script 83

5.1 Testnetwork topology 86
5.2 Delay and jitter measurements 88
5.3 Clockskew Variation 88
5.4 Bandwidth for 5 MBit/s UDP 90
5.5 Delay for 5 MBit/s UDP 90
5.6 Jitter for 5 MBit/s UDP 90

5.7 Bandwidth for 5 MBit/s UDP 90
5.8 Delay for 5 MBit/s UDP 90
5.9 Jitter for 5 MBit/s UDP 90
5.10 Bandwidth for 5 MBit/s EF reservation (UDP traffic) 92
5.11 Delay for 5 MBit/s EF reservation (UDP traffic) 92
5.12 Jitter for 5 MBit/s EF reservation (UDP traffic) 92
5.13 Bandwidth for 5 MBit/s AF reservation (UDP traffic) 92
5.14 Delay for 5 MBit/s AF reservation (UDP traffic) 92
5.15 Jitter for 5 MBit/s AF reservation (UDP traffic) 92
5.16 Bandwidth for 5 MBit/s EF reservation (TCP traffic) 95
5.17 Delay for 5 MBit/s EF reservation (TCP traffic) 95
5.18 Jitter for 5 MBit/s EF reservation (TCP traffic) 95
5.19 Bandwidth for 5 MBit/s AF reservation (TCP traffic) 95
5.20 Delay for 5 MBit/s AF reservation (TCP traffic) 95
5.21 Jitter for 5 MBit/s AF reservation (TCP traffic) 95

6.1 The Bandwidth Broker Architecture 102
6.2 Overview of the API classes 103
6.3 Service Level Description format 105
6.4 Flow Description format 105
6.5 An application example for the abstract API classes 112

7.1 The Bandwidth Broker Architecture for an isolated network 116
7.2 Setup of the Topology Database 118
7.3 Communication Architecture for the Bandwidth Broker 122
7.4 The Router Communication Server 124
7.5 The Broker Communication Server 125
7.6 Message sequence for negotiating a new SLS 125
7.7 GUI Input Menu 127
7.8 An exemplary tiers-generated topology 128
7.9 Flow Database Size in kB 129

7.10 Flow Database Size for 1010 Nodes Topology 129
7.11 Memory Consumption of the Bandwidth Broker 130
7.12 Flow Setup Time of the Bandwidth Broker (Best Case) 132
7.13 Flow Setup Time of the Bandwidth Broker (Worst Case) 134
7.14 An exemplary multi-ISP scenario 136
7.15 The Bandwidth Broker Architecture for multi-ISP scenarios . . . 139
7.16 Signalling Protocol Messages 141
7.17 New SLS Signalling Packet Format 142
7.18 Changing an Existing Flow 144
7.19 Hierarchical Bandwidth Broker Example 146

8.1 Demo Scenario for QoS Provisioning to a mobile user 154
8.2 The New Bandwidth Broker Communication Server 156
8.3 Message sequence for SLS transfer to a foreign network 157
8.4 A scenario for inter-domain broker signalling 158

9.1 Requesting Neighbour Wireless Cells Before the Handover 160
9.2 Link layer handover message-time diagram 161
9.3 Link layer handover SNR-time diagram 163
9.4 Outline of the roaming path 166
9.5 Signal to noise ratio when roaming with low AP density 167
9.6 Signal to noise ratio when roaming with medium AP density . . . 168
9.7 Signal to noise ratio when roaming with high AP density 168
9.8 Interaction of the Wireless LAN Monitor components 169
9.9 Wireless LAN Monitor procedure message-time diagram 170
9.10 Scenario for a QoS-aware Handover 172
9.11 New Service Level Specification Format 173

10.1 AAA Message Sequence for a Mobile Node 176
10.2 Message Sequence in SLP 178
10.3 Message Sequence in SLP capable AAA Architecture 179
10.4 Message Sequence in Negotiation Procedure 181

10.5 An Authentication Option Extension in an Attribute Request Message 181
10.6 A Key Distribution IP Option Extension 184
10.7 An Account Number Option Extension in a Service Request Message 184

C.8 A Mobile User with QoS Requirements 188
C.9 Communication of a Mobile User in the Home Network 189
C.10 Communication of a Mobile User in the Foreign Network 191

List of Tables

2.1 The 3 Reservation Possibilities 33
2.2 Combination with Shared Explicit 33
2.3 Assured Forwarding Codepoints 37

4.1 Router Configuration Tables 82

7.1 The Laboratory Platforms 129
7.2 Performance on a single host 131
7.3 Performance on distributed hosts 131
7.4 Flow Setup Speed (Best Case) 133
7.5 Total Number of Routing Table Queries 133
7.6 Difference between Client and Broker Time per Flow Request . . 134
7.7 Flow Setup Speed (Worst Case) 135

9.1 WaveLAN/IEEE thresholds 163

C.1 Example for the new Routing Table Format 195
C.2 The new Routing Table after the Topology Changes 196

Introduction

In the last few years an astonishing amount of small, wearable electronic devices such as laptops, personal digital assistants (PDAs), and mobile phones have been observed. At the beginning of this trend, the former could not conveniently be used for mobile communication, whereas the latter had not enough computational power to solve difficult tasks. However, recently an affiliation of those initially separated branches is noticeable: mobile phones are equipped with larger displays and powerful CPUs, whereas PDAs and laptops contain a broad variety of communication equipment, such as Wireless LAN (IEEE 802.11) and Bluetooth (IEEE 802.15). Due to this fact, new applications that were restricted beforehand to fixed terminals are now becoming available to mobile users. The transfer of applications from fixed to wireless access methods nevertheless requires additional thought: The Quality of Service (QoS) wireless networks offer varies from the QoS usually available in wired networks:

- The bandwidth of wireless networks is much smaller than the bandwidth of wired networks (11 MBit/s in IEEE 802.11b [171] versus 100 MBit/s or 1 GBit/s Ethernet (IEEE 802.3u and 802.3z [172])).

- The delay in a wireless network is higher, since the medium access control (MAC) protocol is different: IEEE 802.11b uses CSMA/CA (Collision Avoidance) which adds a fixed Interframe Space (IFS) plus an additional random backoff time before sending a packet. IEEE 802.3 uses CSMA/CD (Collision Detection), that allows immediate sending after the medium is found free.

- Wireless media usually suffer from a bit error rate several orders of magnitude higher than wired media (10^{-5} compared to 10^{-13}), they are subject to interference and shadowing effects.

- Handovers between different base stations in wireless access networks cause transmission breaks, additional latency and packet loss.

- A wireless network uses a shared medium, the radio frequencies. Therefore, the QoS delivery depends on the presence of other nodes in the network.

For all these reasons, Quality of Service (QoS) issues have a much higher impact on applications running on a mobile device. Also, QoS guarantees are much harder to accomplish. Several modern applications, such as multimedia streaming, audio/video conferencing, or distance education, rely on continuous QoS provisioning of the underlying network. Adaptive applications may be able to decrease their bandwidth requirements themselves if they detect QoS degradation but this always results in a loss of quality for the user and is therefore not desirable. In this thesis we want to discuss the enhancements that are to be applied to the current Internet architecture in order to provide continuous Quality of Service to mobile users.

For that purpose we partly want to rely on already existing and well-known standards: Differentiated Services (DiffServ) [124, 16, 125] (see Section 2.2) for the Quality of Service support and Mobile IP [50, 123] (see Chapter 1) for mobility management. These two approaches are broadly discussed in the IETF and are still evolving. On the other hand, simply combining the two standards does not solve the problem: in [23] we investigate this topic and identify two key issues: The lack of a DiffServ signalling protocol and the need of the mobile node to be able to adapt to a large variety of possible access methods. The interoperability of the two standards remains a problem, since they were developed independently and do not support each other. We therefore need an entity that supports both standards and solves the interoperability problem: a mobile-aware bandwidth broker.

However, several other problems have to be considered: The access networks, which usually are wireless in a mobile scenario, have to support service differentiation and QoS as well. We do not discuss this problem in this thesis but refer to some possible solutions in Section 2.3. Another problem is the latency of handovers in the initial Mobile IP standard: since many signalling messages have to be exchanged, the user has to suffer a significant delay until the connection is reestablished. In Section 1.3.2 we present some existing approaches to speed up the handover in Mobile IP and decrease the latency.

In the following we will discuss the situation of a mobile user in a network that already offers Mobile IP and Differentiated Services, but no reconciling framework. We will also see that a simple combination of existing standards and proposals would result in a oversized architecture that is not easily understood, implemented and maintained. This discussion uses some of the terminology of Differentiated Services and Mobile IP, which is explained in detail in the corresponding chapters of Part I.

Difficulties of Providing QoS in Mobile Environments

In this section we want to investigate the difficulties an imaginary mobile user has to face if he would like to start an application, requiring a special Quality of Service (i.e. a guaranteed bandwidth, certain delay or jitter bounds, etc.). Such an application could for example be a video conferencing software. Furthermore, we will see, that even more problems arise, when this mobile user moves to one or several foreign domains and does not want to lose its initial QoS level. For our evaluation scenario, and all other scenarios that are used as an example throughout this thesis, we make the following assumptions:

- The Internet is separated into several, independently managed domains.
- Each node belongs to a unique domain, labelled “home domain” or “home network”. All other domains are considered “foreign domains”.
- Mobile IP is used for supporting mobile hosts, which means home and foreign agents are present in each network.
- All networks support wireless access for mobile hosts. For simplicity we assume that this access technology is similar to Wireless LAN (IEEE 802.11) technology.
- All routers support Differentiated Services and implement the expedited forwarding (EF) and assured forwarding (AF) per-hop-behaviours (PHBs) (see Section 2.2.3 and [40, 74]).

Figure I.1 shows the initial scenario with the different entities. The shaded numbers in the figure refer to the numbers in the enumeration below and indicate the location of the specific problem description. At the same time they refer to the problem-specific solutions presented in the next section. A mobile host (MH) is connected to a correspondent host (CH) by a bidirectional connection. For each direction a user wants a DiffServ reservation of 500 kbit/s. Afterwards, the mobile host moves to the two locations within a foreign access network at the bottom of the scenario. Naturally the user is not willing to quit the communication or to suffer from service degradation caused by the movement. We now want to identify the problems of continuous user-friendly negotiation of resource reservation.

1. At the present time, DiffServ lacks a signalling protocol which enables users to negotiate resource reservations “on the fly”. Currently, the negotiation is a rather time-consuming task of contacting a network’s system administrator via email or telephone and to negotiate the parameter settings of the edge

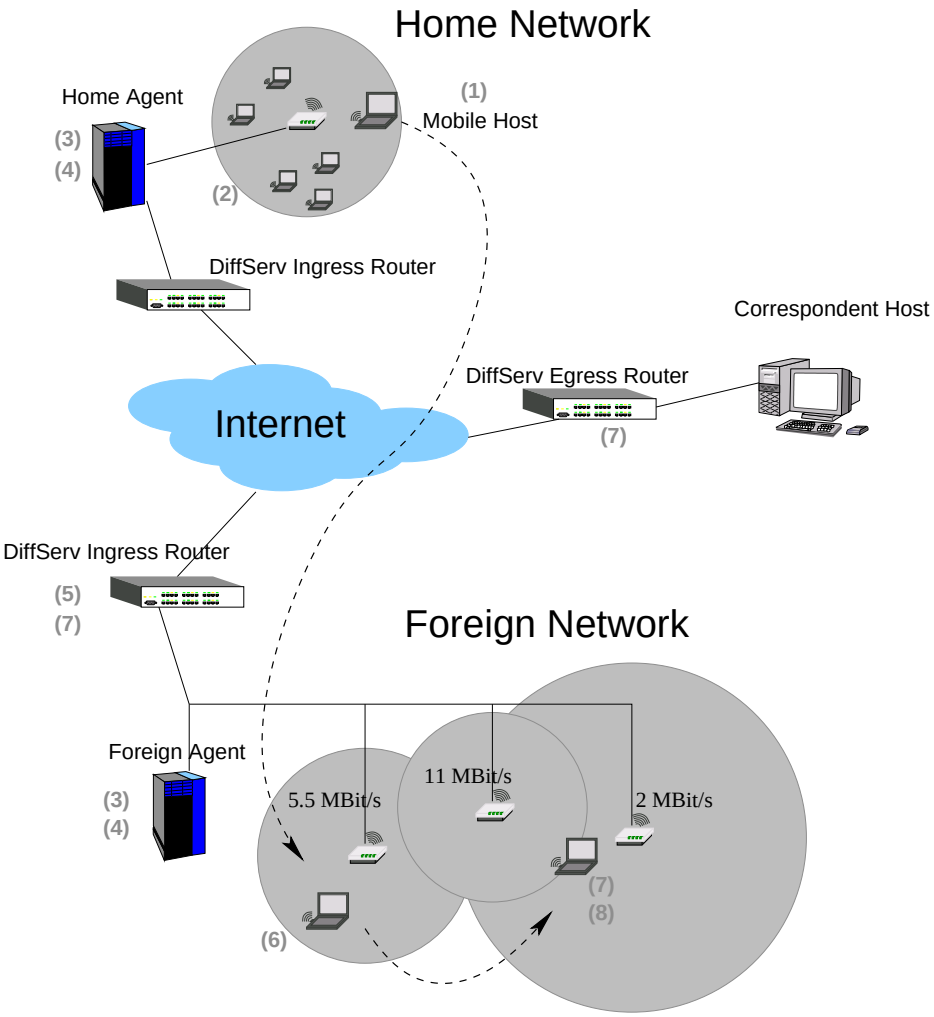

Figure I.1: A Mobile User with QoS Requirements

routers of some domain. At its home location, the mobile user therefore has to negotiate a fixed QoS-specification that applies to all applications the user would like to run.

2. If several mobile hosts share a wireless access network, there is some interference between them, thus the level of QoS that can be delivered depends on the presence of other mobile hosts and — more generally — the Quality of Service the lower layers are able to provide.

3. Handovers might happen between access networks under different administrative control. Therefore, some trust-relationship between the mobile user and the foreign network usually needs to be present or established: a preset relationship is unlikely since it requires the knowledge of all foreign networks a mobile host will visit on its journey in advance.

4. After the handover the mobile node suffers a large delay until its connection is reestablished, due to the Mobile IP handover signalling. This is especially costly, if home and foreign networks are separated by a transcontinental link. In addition, after a handover, packets from the mobile host may start using a new care-of-address. This might lead to difficulties recognising an ongoing session in some forwarding functions at intermediary nodes.

5. At the new access network no DiffServ reservation is established. Due to the missing DiffServ signalling protocol the user has to establish reservations in advance. This is difficult, since the user has to know exactly which foreign networks he will visit. It also wastes resources, since the reserved bandwidth is used for a short period of time only. In addition, effects of lower layers, such as handovers to different base stations, may completely prohibit this approach.

6. The scenario above assumes that a mobile host is always allowed to attach to a base station. This may not be the case, either due to invalid authentication / authorisation or if several mobile hosts already are attached to this base station and the local policy does not allow further visitors to prevent service degradation for the existing connections.

7. In case of an inter-domain handover between two foreign networks (i.e. a handover from one wireless cell in a foreign network to another wireless cell in another foreign network) the mobile user has to suffer the same latency as at the first handover (i.e. the Mobile IP tunnel setup and the DiffServ reservation latency). Usually such a handover only affects a small part of the overall path from the correspondent host to the mobile host, and should therefore be performed much faster.

8. After a handover more than one wireless cell could be chosen for association. Yet it could be possible that only a subset of the available base stations is able to continue the QoS support for the mobile host (e.g. there is not enough bandwidth available in other wireless cells).

Approaches for Solving the Different QoS Problems in Mobile Environments

Now we will evaluate how the problems mentioned above can be solved. For almost each problem there exists a proposed solution but so far no effort has been made to combine all the different proposals to a comprehensive framework that offers a simple user-friendly interface for mobile users to specify and hold their QoS requirements. Even worse, since most of the proposed solutions focus on a specific problem, future extensibility is often neglected. This makes the effort of combining the different proposals even heavier.

In the following sections we will shortly present the existing proposals and the contribution of this thesis, following the enumeration used before.

1. Bandwidth Brokers for Differentiated Services

The lack of a management entity for DiffServ has been mentioned quite shortly after the publication of the architecture [137]. In the meantime, a management framework for DiffServ, mostly built on an entity called “Bandwidth Broker” (BB) has been developed and implemented at several research groups [77, 92, 134, 135, 169, 193] (see also Sections 3.5 and 3.6). Unfortunately, the work so far does not take into account the special needs of mobile users.

In this thesis we present a framework that solves these two issues: A bandwidth broker, together with an appropriate signalling protocol, offering fast and convenient flow negotiation between the mobile user and the visited access ISP will be presented in Part III. In Part IV we will present a novel handover scheme, developed for supporting a continuous Quality of Service during the handover to a new access network.

2. Quality of Service Support in Wireless Networks

End-to-end provisioning of Quality of Service also includes service differentiation at the last hop. The last hop in mobile scenarios will usually be a wireless

link. This medium is shared because of physical reasons. This problem cannot be solved as easily as in Ethernet access networks, where modern installations usually use switches instead of hubs, thus providing non-shared access to the first-hop-router. Since the IEEE 802.11 standard is the most widely used today, several proposals exist about how to enable differentiation mechanisms and QoS support in this access technology. Some of these approaches will be discussed in Section 2.3.

3. Authentication and Authorisation

If a mobile node visits a foreign network and asks for a certain Quality of Service, the foreign ISP usually wants to get paid for its service. This leads directly to the problem of Authentication and Authorisation, since we must ensure that no one can use a service without being properly identified, that a user is actually entitled to use the service and is willing to pay for it. The standardisation of an Authentication, Authorisation and Accounting (AAA) framework has already taken place [42, 179, 180, 58], the special needs of Mobile IP users, however, are not fully taken into account. In Chapter 10 we propose an extension of the existing AAA architecture that supports mobile users specifying their QoS requirements in a DiffServ network.

4. Handover Optimisation in Mobile IP

The topic of optimising handovers is broadly discussed in the IETF [39, 41, 95, 97, 132, 154, 177]. However, this discussion almost exclusively focuses on minimising the delay and the loss of packets that results from the handover. Since we want to combine QoS issues and mobility support we have to focus on QoS-aware handover procedures, that are not investigated in detail within the IETF. A new handover strategy is proposed in Chapter 9. We develop a signal quality monitoring program that can initiate the actions needed to prepare a handover to a new access network. This includes the reservation of sufficient resources. Our approach additionally helps to accelerate the handover, similar to the approaches discussed in Section 1.3.2.

5. QoS Signalling Protocol

DiffServ networks, or broadly spoken QoS enabled networks, are expected to handle two different kinds of QoS granularities: per-flow QoS and per-class QoS. Per-flow QoS is usually used in access networks and there may be subject of QoS

signalling. In the core networks per-class - QoS has to be used due to scalability reasons. In DiffServ, for example, the ingress router is the only place where per-flow granularity is required (see Sections 2.2.5 and 4.2). A QoS signalling protocol has therefore to meet different demands depending on the part of the network it is operating on. A detailed overview of the requirements for QoS signalling protocols is given in [55]. In Sections 7.3 and 7.4 we propose our own signalling protocol that is designed for a very high speed of communication between the bandwidth broker and the user and between the bandwidth broker and the routers. The performance of this protocol has been analysed in Section 7.6. Our signalling protocol can also support mobile users by offering a new set of functions designed to automate the reseource reservations needed when a handover occurs. This is described in Section 8.2.

6. Admission Control

For each incoming reservation request, it has to be checked that the user is allowed to reserve resources and, if this is the case, that there are enough available resources. Checking the permissions of the user also includes checking the correctness of the provided identity. This results in a need of an AAA architecture. A more detailed discussion of how to include an AAA framework into the scenario is found in Chapter 10. The second question — about the availability of resources — can be delegated to the bandwidth broker: it can, based on the care-of address of the mobile user, decide at which access network the user is registered, and thus keep track of the resources used in this part of its network. If a configurable amount of resources is occupied, the bandwidth broker can reject the request.

7. Hierarchical Bandwidth Brokers

Usually a centralised approach, such as a bandwidth broker, has one serious problem: scalability. With scalability we mean the ability to perform well with an increasing number of participants / nodes. Generally speaking, centralised architectures do not perform well within a large network, since the number of messages, states and data to be processed and stored increases rapidly with the size of the network. However, we believe that in practice a centralised bandwidth broker does not suffer too much from scalability problems. The performance evaluation in Section 7.6 shows a good performance over a large range of network sizes. Nevertheless, if this performance is too low, there is the possibility of sharing the load between several bandwidth brokers that are organised in a hierarchical way (see [134, 191, 193] and Section 7.8). In this case, each so-called leaf broker

is only responsible for a small part of the network, aggregating the resource requests while the root broker only acts on traffic aggregates. A small increment in signalling (the messages between the root and leaf brokers) can result in a big performance boost of the architecture, provided the location of the leaf bandwidth brokers is chosen carefully.

8. QoS-aware Handovers

Up to now, the handover from one wireless cell to another is a pure layer 2 process: the radio network interface card decides to which wireless cell it next associates with no respect to upper layer protocols or applications. This can seriously downgrade existing Quality of Service. A lot of recent research work is presently done to accelerate the handover procedure. Unfortunately, the topic of resource reservation is neglected in this discussion. We therefore propose to transfer the responsibility of the handover procedure to a user program, which is aware of existing resource reservations and can direct the handover decision in a way that violating the QoS level is minimised. We propose to introduce a protocol between a bandwidth broker and the mobile user, that ensures the availability of resources at the new access network (if possible) and can additionally be used to pre-negotiate a transfer of the existing reservation to the new access network (cf. Chapter 9). To a certain extent this protocol also solves the problem of handover latency.

Summary

We can see that for many problems we encounter when looking for QoS support for mobile users, isolated solutions exist. Unfortunately, these solutions do not work together very well. Quite often, the support for mobile users has not been included from the beginning and adding it later so far usually resulted in big overheads and complexity. We want to present an architecture that considers the needs of mobile users from the beginning and therefore avoids complicated structures.

Contribution of this Thesis

In this thesis we will present an architecture and implementation of a central bandwidth broker architecture that is able to manage large DiffServ networks and offers a fast and convenient framework for mobile and immobile users to specify their QoS requirements. Although many components that could be used to build such

a framework already exist so far, there is no architecture that takes both, Quality of Service and mobility, into account.

We will present a flexible bandwidth broker which is easy to use and to administer and which will do most of the configuration work a network administrator of a DiffServ network has to do today. Additionally our broker also offers a convenient interface for mobile users in a way that they can specify their QoS requirements and hold this initial QoS level during their session no matter whether they are moving or not. The bandwidth broker is — like in many other designs — separated into two parts: a management layer and a configuration layer. The novelty of our approach is to introduce an object-oriented interface between the two layers. By using the polymorphism of this interface we can hide the heterogeneity of the network and offer a common management interface to the management layer.

The interface is built using a newly developed Application Programmers Interface (API) for Quality of Service management (Section 6). This API can be used to form a homogeneous image of the underlying network: each router and interface — no matter which kind of router hardware or manufacturer — is represented by an object-oriented class that offers a common interface for configuration. This way, the configuration of today's heterogeneous networks can be significantly simplified. This topic has not yet been widely addressed in the literature.

A bandwidth broker implementation can use our API to configure heterogeneous networks in a very simple way. Such a broker has been implemented (Section 7), performing topology auto-detection of large networks, initialising DiffServ configurations at the routers and offering a user interface for flow reservation. This implementation has been tested (Section 7.6) and proved to be able to configure large networks while offering a comfortable flow reservation rate. The networks in our tests have been considerably larger than test-networks in other publications: We have used several hundred nodes, whereas other simulation results are only based on 3 - 20 nodes and on a very simplified topology.

The great flexibility of our architecture can be shown by adding support for hierarchical brokers and mobile users step-by-step (Chapters 8, 9): without changing the internal design it is possible to provide a scalable way for Mobile IP users to negotiate reservations for all necessary links involved. Some additional commands are necessary and can easily be added to the broker's API. Even pre-handover negotiation and QoS-aware handover support is possible.

Our design might be quite similar to the design of other bandwidth brokers. Yet we have added some new functionalities into our approach that proved to be a major advantage. The QoS management API introduces an additional software layer in the architecture, separating the management part of the bandwidth broker from the configuration part. Therefore the flow management can be handled independently

from the router hardware. For simulation and performance evaluation even virtual routers can be used. Several new functionalities, which are not part of the basic bandwidth broker architecture have also been included in the implementation, e.g. topology auto-detection, backward reservation, over-provisioning support, and a link reservation / utilisation database.

Structure of this Thesis

This thesis consists of four parts. In the first part, existing standards and research work related to our work is discussed and taken into account. Each of the following parts discusses an approach to the final goal: a network where a mobile user is able to roam freely between different access networks without suffering from service degradation. Each part starts with a small overview that lists its contents in detail and will clarify the connections between the components listed in this, as well as in other parts. Similarly, each part closes with a short summary, listing again its most important results and the effects these results will have on the other work presented within.

The first part discusses well-known fundamental work that has mainly been published within the IETF workgroups, as well as current research work. This related work can be divided into three main topics: Mobile IP is discussed in Chapter 1. Several proposals to enhance the disappointing performance of Mobile IP handovers are included (Sections 1.3.1 – 1.3.4). A comprehensive introduction on Quality of Service follows in Chapter 2. First we will introduce the two main techniques used to implement QoS: Integrated and Differentiated Services. In the following sections we will focus on mobility-related topics of Differentiated Services, namely the DiffServ Support in IEEE 802.11 wireless access networks (Section 2.3) and a signalling protocol for DiffServ (Section 2.4). The third chapter is about network management from the ISO point of view. This chapter also includes recent research work on the main topic of network management that we are interested in: Bandwidth broker design (Section 3.5) and bandwidth broker implementations (Section 3.6).

The second part will investigate the behaviour of Differentiated Services in a test network. We will present our own implementation of DiffServ for Linux Routers (Chapter 4) and show a detailed performance evaluation (Chapter 5) that will prove the ability of DiffServ to protect high-priority flows from aggressive background traffic.

The third part will develop a novel application programmers interface (API) for programmers of network management software, focusing on Quality-of-Service

management (Chapter 6). Based on this API, a bandwidth broker architecture and implementation is developed and evaluated in Chapter 7.

The fourth part focuses on the changes that have to be made in the previously introduced bandwidth broker in order to be able to support mobile users (Chapter 9). In addition, an extension to the existing Authentication, Authorisation and Accounting architecture is presented to support mobile users (Chapter 10).

Finally, in the conclusion we will come back to the example presented in the Introduction (Figure I.1) and show, how a mobile user would be able to negotiate a resource reservation in a network using our architecture.

Part I

Related Internet Standards and Research Work

Overview

This part gives an overview of the Internet standards this thesis is built on, as well as a selection of Internet Drafts and other publications of various other authors that are related to the topics of this thesis. This related work can roughly be divided into three groups:

The first group (Chapter 1) is about the Mobile IP standard (Sections 1.1 and 1.2). The majority of publications concerning Mobile IP discusses the topic of handovers. In the Sections 1.3.1 and 1.3.2 we will discuss two approaches to improve the current Mobile IP handover in terms of packet loss and latency. These approaches have been started very shortly after the publication of the Mobile IP standard. However, these proposals try to solve the problem in the network layer only. More recent research work has shown, that some problems that occur during a handover can only be solved if information from lower layers is also taken into account (Sections 1.3.3 and 1.3.4).

The second group (Chapter 2) discusses the Quality of Service support in IP networks. The first two sections deal with the two main approaches : Integrated and Differentiated Services (Sections 2.1 and 2.2). Due to the well-known scalability problems of Integrated Services in the backbone, we focus on Differentiated Services and evaluate the extensions needed to support mobile users in a DiffServ network. Two important topics are handled explicitly in Sections 2.3 and 2.4: The QoS support of wireless networks, which we expect to be common for mobile users, and a signalling protocol for DiffServ, whose absence is the main obstacle for fast and convenient DiffServ resource reservation.

The third group (Chapter 3) concerns the network management. Here we first present the ISO network management architecture (Section 3.1). Afterwards we discuss the various designs and implementations of a network management architecture for a DiffServ network (Sections 3.5 and 3.6). In a DiffServ network, a network management and configuration entity is often called a bandwidth broker.

Chapter 1

Mobile IP

The increasing number of mobile and portable devices, such as laptops, personal digital assistants (PDAs) or mobile phones, presents problems regarding the configuration and administration of networks. Predefined installations and configurations of the device are not applicable to react to highly dynamic environments, since the conditions a user may encounter are not predictable. Additionally, such a pre-configuration and its maintenance leads to higher workload for the system administrator as well as for the user. Perhaps the most important topic, however, is the waste of allocated IP addresses occurring by using this approach.

An environment supporting Dynamic Host Configuration Protocol (DHCP) allows to provide a valid IP address to a device newly attached to a network. This is a very flexible approach, but it suffers a serious drawback whenever the user wants to stay online while moving around. If the movement of the user causes a handover by leaving one access network and connecting to another access network (e.g. due to radio coverage), the new access network is likely to be connected to another DHCP server. The mobile client will therefore require a new IP address. Unfortunately this leads to a disruption of all TCP connections the mobile node currently holds. This behaviour prohibits the use of DHCP in mobile environments with handovers. An alternative approach to this problem called Mobile IP is provided in [48].

Mobile IP allows a network device to connect to the Internet from various access network while maintaining a unique IP address (the *home address*). Obviously, a device with an IP address connected to a foreign subnet will not be able to communicate normally. Therefore, an additional IP address (the *care-of address*) from the address space of the access network is given to the mobile device. A special router in the home network, the *home agent* is responsible for capturing traffic addressed to the home address of the mobile node and for forwarding it to

the current care-of address. Usually the mobile node uses its own home address as the source address of all the packets it sends.

1.1 Terminology

The following terminology is used in Mobile IP; a more detailed overview of mobility-related terminology can be found in [113].

Home Address The static IP address owned by the mobile node. This address does not change wherever the mobile nodes attaches to the Internet.

Care-of Address A dynamically allocated IP address which a node uses to designate its current point of attachment to the access network.

Home Agent A router in the home network of the mobile node that keeps track of the current location of the mobile node. It also intercepts packets addressed to the mobile node, encapsulates them in a tunnel to the current care-of address and forwards them to the mobile node.

Foreign Agent A router in the network visited by the mobile node, which is only needed in Mobile IP version 4. It registers the presence of the mobile node, and forms the endpoint of the tunnel from the home agent. The foreign agent therefore decapsulates the packets coming from the home agent and forwards them to the mobile node via normal IP routing.

Tunnel Tunnelling in this case is the process of encapsulating an IP packet inside another (see [131, 152]). Tunnelling can also be thought of encapsulating an ordinary IP packet inside a special (perhaps encrypted) IP packet.

Correspondent Host The communication partner of the mobile host.

1.2 Mobile IP Operation

Mobile IP agents (i.e. home and foreign agents) advertise themselves by periodically broadcasting *agent advertisement* messages, which are ICMP (Internet Control Message Protocol) router advertisement messages with extensions. A mobile node may also explicitly request one of these messages by sending an *agent solicitation* message.

After a mobile node receives such a message it can determine, whether it is in its home network or in a foreign network. If the mobile node is at home, it will operate normally without the use of mobility services. Additionally, any previously registered location will be deleted by a registration request / registration reply sequence. If, however, the mobile node has moved to a foreign network, it obtains a care-of-address for the foreign network. This address can be obtained either by the foreign agent itself or by other mechanisms, such as DHCP. The latter case is also known as co-location of the care-of-address. This has the advantage of being independent from the presence of a foreign agent. The communication between the mobile node and a foreign agent has to take place at the link layer, since the mobile node's IP address does not match the foreign subnet address.

1.2.1 Registration

Once the mobile node has a care-of-address, it can register itself at the home agent. [48] defines two different ways of registration: the mobile node can register via a foreign agent, which relays the registration to the mobile node's home agent, or it can directly register at its home agent. The first case must be used if the mobile node got its IP address from a foreign agent. It should however, use this option, too, if it got an IP address via DHCP but also received an agent advertisement from a foreign agent in this subnet. On the other hand, the mobile node must directly register at the home agent if it has not received a foreign agent advertisement or if it returns to its home network.

After this registration procedure any datagram intended for the home address of the mobile node will be intercepted by the home agent and tunnelled (see [131]) to the care-of-address. The tunnel endpoint may be the foreign agent or the mobile node itself (in case of co-located care-of-address). The mobile node will respond to the datagram using standard IP routing mechanisms.

1.2.2 Routing

Triangular Routing

Any host that wishes to communicate with the mobile host ("Correspondent Host") will send all packets for the mobile host to the public IP address of the home network, since it is not aware of any mobility of the mobile host. The home agent will intercept any packets sent to mobile hosts that have registered at a foreign network, encapsulates the packets and tunnels them to the care-of-address. The foreign agent at the tunnel endpoint will decapsulate the packets and forward them

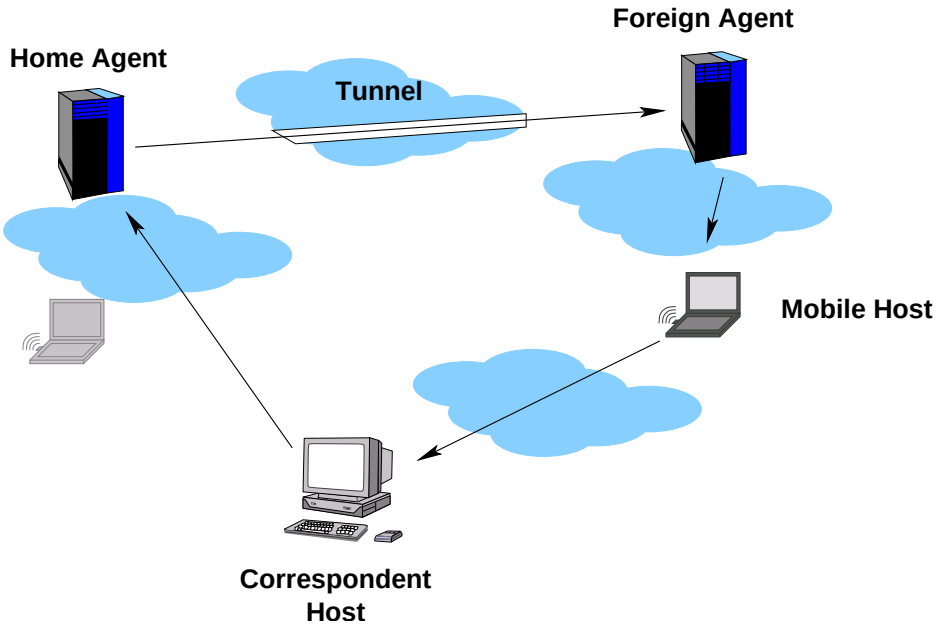

Figure 1.1: Mobile IP Triangular Routing

to the destination address via normal IP forwarding. If there is no foreign agent, the packets are delivered to the mobile host directly. The mobile node then decapsulates the packets itself. The packets from the mobile node to the correspondent node will take the route given by the normal IP routing settings. This routing operation is called triangular routing, since there are three different edges involved in the communication (see Figure 1.1).

Bidirectional Tunnelling

Internet router security guidelines recommend not to forward any IP packets out of an Autonomous System (i.e. a local network) if they have a source address not belonging to the Autonomous System. Since Mobile Nodes emit packets with their home address as source address this packets will be dropped leaving the network. This keeps them from communicating with any Internet host not in the network they are visiting.

To circumvent this obstacle, bidirectional tunnelling is necessary. In this case the packet to be emitted will first be tunnelled back to the Home Agent before being “let out” on to the Internet. Bidirectional tunnelling was not foreseen in the original Mobile-IP standard, but has been added later [123].

Route Optimisation

Usually detouring the packets via the home agent means wasting resources and generates unnecessary delay. Optimising the routing by telling the correspondent host the actual location of the mobile node is the better choice.

Mobile IPv4 route optimisation [133] is a proposed extension to the Mobile IPv4 protocol. It provides enhancements to the routing of datagrams between the mobile node and the correspondent node. The enhancements provide means for a correspondent node to tunnel datagrams directly to the mobile node or to its foreign agent care-of address.

Each time the home agent now receives a datagram that is addressed to a mobile node currently away from home, it sends a binding update to the correspondent node to update the information in the correspondent node's binding cache. After this binding update, the correspondent node can now tunnel packets to the mobile node directly. This way direct bi-directional communication is achieved with route optimisation. This reduces both, network load and also delays caused by routing. The optimisation is therefore valuable to mobile nodes that visit networks located far from their home agent.

Since both, the correspondent nodes and the foreign agents have binding caches, that change the routing of datagrams destined to mobile nodes, the binding updates must be authenticated. The authentication is performed in a similar manner as in base Mobile IPv4. All binding updates contain a route optimisation or smooth handover authentication extension. This extension contains a hash, which is calculated from the datagram and the shared secret [133].

The correspondent node and the mobile node's home agent need a security association [86]. This association is used for the authentication of the binding updates. Since the mobile node sends a binding update directly to its previous foreign agent, they also need a security association. If the security associations are not preconfigured, they can be established via a key management protocol.

1.3 Handover in Mobile IP

A quite important issue in mobile and wireless services is the topic of handovers. Due to the limited range of radio transmission, a mobile user will have to connect to several base stations during his movement. The procedure of passing all necessary information from the old base station to the new one is called a handover. Obviously, each time the user leaves the radio cell of a base station and enters the cell of a new base station, all traffic from and to the mobile node has to take

a new route. In addition some packets from a correspondent host to the mobile host might still be routed to the old destination shortly after a handover. In order to avoid the loss of those packets and the retransmission a transport protocol may require, a solution has to be found to forward those packets from the old access network to the new location. This problem is generally referred to as "smooth handover" (cf. Section 1.3.1).

A second problem in the handover procedure is the rather large amount of signalling messages in the original Mobile IP protocol: each time the mobile node changes the access network (i.e. radio cell) the home agent has to be notified and all tunnels have to be rebuilt. This adds a considerable amount of delay to the handover. In a scenario with small radio cells and thus frequent handovers (such as UMTS, IEEE 802.11) this prohibits a useful employment of Mobile IP. Handover procedures that provide a faster way to switch from one radio cell to another are therefore called "fast handover" (cf. Section 1.3.2). Of course it is desirable to have a combined solution to both problems described above, smooth and fast handover. Such a handover is called "seamless".

The fact that after a handover all packets from and to the mobile node have to take a new route obviously has a serious impact on all resource reservation scenarios: The reservations in the old access network have to be cancelled and new reservations have to be established in the new access network. The problems that arise due to those reorganisations of the network are similar to the two cases discussed above: registering a new reservation at the bandwidth broker takes a certain amount of time and has to be considered in an overall solution for the "fast" handover. In addition, important packets could be dropped due to non-existent reservations shortly after the handover and perhaps have to be retransmitted. This is quite similar to the problem solved by "smooth handover". Sometimes it could also happen, that no reservation is possible at the new access network. In this case we need mechanisms to select between several access network candidates. In addition, information about the possibility of a reservation in the candidate networks is needed. A handover mechanism providing this functionality is called "QoS-aware handover". We will discuss this in Section 1.3.4 and present our own strategy for such a handover in Chapters 8 and 9.

In the following sections we will investigate the different proposals to improve the handover in Mobile IP more in detail.

1.3.1 Smooth Handover

One Internet Draft [56] proposes a new handover method using the Explicit Multicast (xcast) technique for the Small Group Multicast (SGM). On the wired sec-

tion, control and user packets are multicasted using the xcast thechnology and sent to the Base Stations (BS) where mobile nodes (MN) can access. Packets are then passed on to the air-link activated between a base station and the mobile node. For smooth handovers, [94] specifies extensions to Mobile IPv6, which allow additional control structures that enable the transfer of the necessary state during handovers. This state transfer allows the applications running on the mobile node to operate with reduced latency, minimal disruption, and reduced packet loss during handovers. Moreover, Mobile IPv6 regional registration [111] reduces the binding update signalling latency and the signalling load for a mobile node moving within the same visited domain. The latency is reduced by localising binding updates to the visited domain and the signalling load is reduced by using a regional-aware router for a proxy care-of-address, the regional care-of-address, as seen by hosts outside the visited domain. This registration uses an Anycast Address for all regional routers, creates host routes for mobile nodes at relevant routers, allows arbitrary hierarchical topology without disclosing details to mobile nodes roaming from other domains, specifies an optimal method for forwarding and is compatible with smooth/fast handovers. An important issue that needs to be considered when supporting real-time applications like VoIP in mobile networks is the capability to provide smooth handovers. A critical requirement for smooth handovers is to minimise packet loss as a mobile node moves between network links. [98] defines a buffering mechanism for Mobile IPv6 by which a mobile node can request that the router on its current subnet buffers packets on its behalf while the mobile node completes registration procedures with the router of a new subnet. Once the registration is complete and the mobile node has a valid care-of address in the new network, the buffered packets can be forwarded from the previous router, thus reducing the possibility of packet loss during the transition. In networks with limited bandwidths, such as wireless cellular networks, compression of IP and transport headers may be employed to obtain better utilisation of the available spectrum capacity. When header compression is used along with handovers in such networks, the header compression context needs to be relocated from one IP access point (i.e., a router) to another in order to achieve seamless operation.

1.3.2 Fast Handover

Mobile IP describes a way of how a mobile node can change its point of attachment to the Internet while maintaining a unique IP address, a process referred to as handover. During this process, there is a time period during which the Mobile Node is unable to send or receive IP packets. This time period is referred to as handover latency. In certain scenarios, the handover latency resulting from

standard Mobile IP handover procedures can be greater than what is acceptable to support real-time or delay sensitive traffic. Mobile IP was originally designed without any assumptions about the underlying link layers over which it would operate so that it could have the widest possible applicability. This approach has the advantage of facilitating a clean separation between layer 2 (L2) and layer 3 (L3) of the protocol stack, but it has negative consequences for handover latency. The strict separation between L2 and L3 results in the following built-in sources of delay:

- The mobile node may only communicate with a directly connected foreign agent (FA). This implies that a mobile node may only begin the registration process after a L2 handover to nFA (new FA) has been completed.
- The registration process takes some time to complete as the Registration Requests propagate through the network. During this period of time the mobile node is not able to send or receive IP packets.

There exist several ways to achieve a low-latency Layer 3 handover: [54] distinguishes between the following methods:

- pre-registration handover method
- post-registration handover method
- combined method

The pre-registration handover method allows the mobile node to be involved in an anticipated IP-layer handover. The mobile node is assisted by the network in performing a L3 handover before it completes the L2 handover. The L3 handover can be either network-initiated or mobile-initiated. Accordingly, L2 triggers are used both in the mobile node and in the foreign agent to trigger particular L3 handover events. The pre-registration method coupled to L2 mobility helps to achieve seamless handovers between FAs. The basic Mobile IPv4 concept involving advertisement followed by registration is supported and the pre-registration handover method relies on Mobile IP security. No new messages are proposed, except for an extension to the Agent Solicitation message in the mobile- initiated case.

The post-registration handover method proposes extensions to the Mobile IP protocol to allow the oFA (old FA) and nFA (new FA) to utilise L2 triggers to set up a bi-directional tunnel between oFA and nFA that allows the mobile node to continue using its oFA while on nFA's subnet. This enables a rapid establishment

of service at the new point of attachment which minimises the impact on real-time applications. The mobile node eventually has to perform a formal Mobile IP registration after L2 communication with the new foreign agent is established, but this can be delayed as required by the mobile node or foreign agent. Until the mobile node performs registration, the foreign agents will setup and move bidirectional tunnels as required to give the mobile node continued connectivity.

The combined method involves running a pre-registration and a post-registration handover in parallel. If the pre-registration handover can be performed before the L2 handover completes, the combined method resolves to a pre-registration handover. However, if the pre- registration handover does not complete within an access technology dependent time period, the oFA starts forwarding traffic for the mobile node to the nFA as specified in the post-registration handover method. This provides for a useful backup mechanism when the completion of a pre- registration handover cannot be guaranteed before the L2 handover completion.

Two different handover mechanisms are proposed in [51], too:

- anticipated handover
- tunnel-based handover

In the first case the third layer initiates a handover to the new access network, while layer 2 connectivity remains at the old access network. Predictive information about the direction of the handover or the ability of forcing a handover to a specific access network is necessary in this case. In the second case, the mobile node defers the third layer handover until it is attached to the new access network. The traffic is tunnelled using the old care-of address to the new location.

The *anticipated handover* is quite similar to the handover technique adopted in our architecture, which we specify in Chapter 9: We predict the direction of the handover based on the signal - to - noise ratio, but leave the decision to the user (more precisely: a handover daemon) and force the handover to the chosen access network. The fact that the user can affect the handover decision by configuring the handover daemon is extremely important for QoS-aware handover (cf. Sections 1.3.4 and 9.4).

Another approach similar to the tunnel-based handover is presented in [82]: This approach proposes to set up a chain of tunnels between access routers for subsequent handovers. To minimise the latency introduced by the chain of tunnels a maximum tunnel length threshold is introduced. The authors claim a performance gain compared to the anchored tunnelling handover of [51], but unfortunately neither experiments nor results are available to proof this statement.

Conclusion The discussion above shows, that there are many improvements possible to provide a good and fast handover functionality in Mobile IP. The proposals can roughly be divided into two categories: a pre-handover negotiation approach from the old access network or a tunnelling, post-handover approach where the old care-of address is used in the new access network. The handover procedure proposed in this thesis (cf. Chapter 9) is quite similar to some of the solutions proposed for a pre-handover. Our proposal is also able to support the QoS demands of a mobile user, which is not the case in most fast-handover functions. However, due to the similarity it will share the benefit of low handover latency.

1.3.3 Effects of Layer 2 Technology

The Mobile IP working group has considered enhancements that significantly reduce the amount of service disruption involved in Mobile IP handover. The proposals treated by the working group so far are based on having layer 2 information available prior to the handover which allows the mobile node and/or the old foreign agent to prepare for handover in some fashion.

According to [54], a L2 trigger is a signal of a L2 event. One possible event is early notice of a upcoming change in the L2 point of attachment of the mobile node to the access network. Another possible event is the completion of relocation of the mobile node's L2 point of attachment to a new L2 access point. This information comes from L2 programmatically or is derived from L2 messages.

The requirements of layer 2 and layer 3 interoperation are discussed in [80, 190]. The authors state, that L2 triggers like the ”link tear down” and ”link establishment” can be used to indicate departure and arrival of a mobile node at a base station. Such indications can replace L3 signal exchange and therefore expedite the process. Timely receipt of this triggers is needed as protocol signalling needs to take place in parallel with the handover.

In [129] the author proposes a second wireless interface to solve the problem of smooth handovers: to prevent service interruption, a terminal with two wireless interfaces can keep existing connections alive over one interface while searching other frequency bands with the other. However, as the author already mentioned, this increases the cost for a mobile terminal.

1.3.4 QoS-Aware Handover

The QoS requirements for Mobile IP are summarised in [52]:

- Minimise the interruption in QoS at the time of handover

- Localise the QoS (re)establishment to the affected part of the packet path in the network
- Releasing after handover the QoS state (if any) along the old packet path
- interoperability with mobility protocols
- interoperability with heterogeneous packet paths as regards QoS paradigms
- QoS support along multiple packet paths
- interaction with wireless link-layer support for QoS

A QoS-aware handover is defined in [41] as a kind of handover that allows the mobile node to change the current point of attachment to the Internet without loosing the perceived QoS. In [41] the authors introduce a "Secondary Home Agent" that helps to set up a new reservation while the old one is still present.

In [63] we also find a basic signalling protocol and handover description that holds a QoS level during a handover by pre-registering the new route while staying connected to the old access network. The performance results published by the authors unfortunately do not handle this type of a handover.

Another approach is presented in [43]: two new services are introduced to replace expedited forwarding (called "Mobile Premium Service") in a mobile scenario and to complement this new "Mobile Premium Service". Using this new service, only a statistical handover guarantee can be granted. This service is implemented by reserving a certain amount of bandwidth for handovers. The second service, called "Best Effort Low Delay Service" utilises free Mobile Premium Service resources to offer cheap connectivity for real-time applications. However, the connection may break if the resources are used by the Mobile Premium Service users. A PHB for this service has been published as an Internet draft [44]. In another publication [59] the authors propose to implement service differentiation based on long-term traffic priorisation schemes. The authors suppose, that pre-negotiation is not applicable since the mobility of the user is unpredictable. Also, the user should be able to select between a priorisation level at the time when the application is started. However, the authors do not define a signalling protocol between the provider and the user to fulfil this task. In addition, such a signalling protocol could be used to negotiate the necessary parameters prior to a handover and also for directing the handover to a certain access point (for an implementation of such a signalling protocol, see Chapter 9).

Using the Hierarchical Mobile IPv6 (HMIPv6) architecture [154] the authors of [62] propose another QoS-aware handover (called QoS-conditionalised han-

dover). This approach also demands to perform a handover only if enough resources are available at the new access point. However, the authors fail to mention what happens, if no access point can provide the necessary resources and a handover has to be performed anyway (e.g. due to low signal quality). The signalling for the QoS-conditionalised handover is combined with the binding update messages to the HMIPv6 mobility anchor point. The authors assume some kind of QoS entities that are able to perform the necessary reservations to be distributed in the network (at least one at each access point and each anchor point). This QoS architecture, even though it is very important for the handover, is not mentioned in more detail.

Although the importance of QoS-aware handovers is emphasised in several publications [114, 115], no implementation is available to the time present. Some architectural considerations have been made so far [41, 63], but a detailed description and performance evaluation is still missing.

Chapter 2

Quality of Service

With the increased use of the Internet in many parts of every day's life, there has been a large focus on providing network resources to certain applications. A new field of multimedia applications (like IP telephony or video conferencing) has different demands on bandwidth, delay and delay fluctuations, and reliability than traditional Internet applications (like HTTP, FTP or telnet).

In todays networks bandwidth is a big issue. More and more people use the Internet for private and business applications. The amount of data transmitted by the Internet is increasing exponentially. However, there is only one service type available for all kinds of applications — the so-called best effort service. IP packets are forwarded hop by hop without any guarantee about bandwidth, arrival time or even delivery. All packets have the same priority and packets that cannot be delivered within a certain amount of time are dropped. There is no possibility to allocate network resources or to change the priority of certain packets.

While traditional Internet applications can tolerate the lack of service discrimination, many modern multimedia applications can not. A high-quality full-screen video transmission in MPEG2 consumes at least 2 MBit/s bandwidth and interactive applications, such as IP telephony, require a end-to-end delay of less than 150 ms. In fast Local-Area Networks (LANs), those requirements can usually be fulfilled. Due to the rapidly increasing transmission capacity of optical fibres this may soon be the case for wide-area networks (WANs), too. However, supporting Quality of Service by only providing faster links does not completely solve the problem: Short-lived congestion may still occur and for wireless access networks the gain in bandwidth in the near future will not be large enough to supply the required capacity.

Therefore, certain concepts for allocating network resources are needed to support those modern Internet applications. The Quality of Service (QoS) an application

requires can be specified as a set of parameters (e.g. bandwidth, buffer usage, priority, dropping probability, ...) of a certain flow. Here we define a flow as a distinguishable stream of related datagrams from a unique sender to a unique receiver that results from a single user activity and all datagrams require the same Quality of Service.

In this chapter we will first discuss the two most important proposals for Quality of Service delivery at the IP layer: Integrated and Differentiated Services. While the former has its benefits in an elaborate signalling, it suffers from a big scalability problem in the backbone. This problem is solved by the latter approach. Unfortunately, DiffServ has no signalling and management framework. This topic is addressed in Section 2.4. In Section 2.3, we will investigate the possibilities of how to offer QoS not only at the IP layer, but also at lower layers, such as in wireless access networks, since the successful QoS support may depend on such assistance.

2.1 Integrated Services

Integrated Services (IntServ) define extensions to the IP network model to support real-time transmissions and to guarantee bandwidth for specific flows. For example, a flow might consist of a video stream between a given host pair. To establish the video connection in both directions, two flows are needed. For example, if each flow requires a minimum of 128 kbit/s and a minimum packet delay of 100 ms to assure a continuous video display, such a QoS can be reserved for this connection.

The Integrated Services model was defined by an IETF working group. This Internet architecture model includes the currently used best-effort service and the new real-time service that provides functions to reserve bandwidth on the Internet. IntServ was developed to optimise network and resource utilisation for new applications, as for real-time multimedia, which requires QoS guarantees. Because of routing delay and congestion losses, real-time applications do not work very well on the current best-effort Internet. Yet, video conferencing, video broadcast and audio conferencing software need guaranteed bandwidth to provide video and audio of acceptable quality.

2.1.1 Components of an Integrated Services Router

To support the integrated services model, a router has to be able to provide an appropriate QoS for each flow, in accordance with the service model. The router

function that provides different qualities of service is called *traffic control*. It consists of the following components (see Figure 2.1):

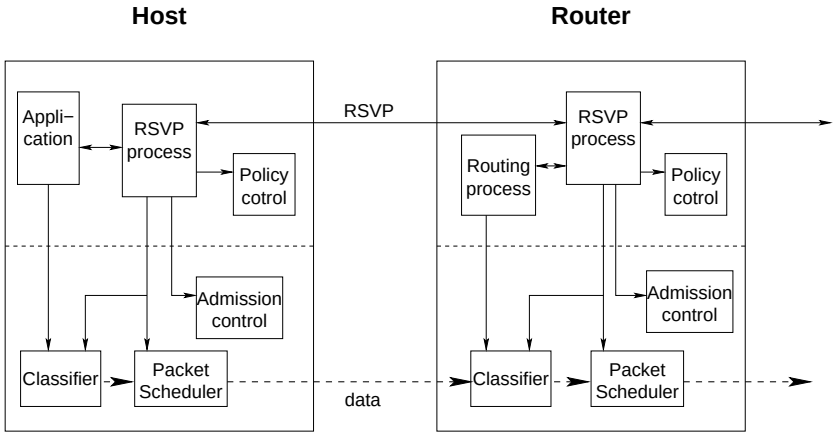

Figure 2.1: RSVP in Hosts and Routers [21]

Packet scheduler The packet scheduler manages the forwarding of different packet streams in hosts and routers, based on their service class, using queue management and various scheduling algorithms. The packet scheduler has to ensure, that the packet delivery corresponds to the QoS parameter for each flow. A scheduler can also police or shape the traffic to conform to a certain level of service.

Packet classifier The packet classifier identifies packets of the incoming traffic that will receive a certain level of service. To realize effective traffic control, each incoming packet is mapped into a specific class. All packets that are mapped into the same class get the same level of service from the packet scheduler. The choice of a class is based on the source and destination IP address and port number in the existing IP packet header or on an additional classification number, which has to be added to each packet. A class can correspond to a broad category of flows.

Admission control The admission control contains the decision algorithm that a router uses to determine whether there are enough resources available to accept the requested QoS for a new flow or not. If not enough resources are available, accepting a new flow would impact earlier granted guarantees and therefore the new flow has to be rejected. If the new flow is accepted, the reservation instance in the router assigns the packet classifier and the packet scheduler to reserve the requested QoS for this flow. Admission

control is then invoked at each router along a reservation path to make a local accept/reject decision at the time a host requires a real-time service. The admission control algorithm must be consistent with the service model.

Policy control Admission control is sometimes confused with policy control, which is a packet-by-packet function, processed by the packet scheduler. It ensures that a host does not violate its promised traffic characteristics. Nevertheless, to ensure that QoS guarantees are honoured, the admission control will be concerned with enforcing administrative policies on resource reservations. Some policies will be used to check the user authentication for a requested reservation. Unauthorised reservation requests can be rejected. As a result, admission control can play an important role in accounting costs for Internet resources.

2.1.2 The Resource Reservation Protocol (RSVP)

The Resource ReserVation Protocol (RSVP) is the most widely used implementation of the IntServ approach. In this section we will only give a short explanation of the RSVP basics. For a more detailed description see [21, 22, 112, 189].

Soft States All parameters that are used to specify Quality of Service are managed flexibly and have only a limited time to live (TTL), which means they are in Soft State. Therefore no explicit notification of closing a reservation is necessary. All data about the reservation is saved in Soft State. This includes previous and following RSVP routers, the flow description and the data about reserved resources.

Example of a Reservation Setup To clarify the idea of RSVP we will give an example (Figure 2.2) of a reservation setup followed by the transmission of data and finally the clearance of reserved resources.

In this example the following can be shown:

- The `Path` message is used to determine the route through the network. This message is sent directly to the receiver. Additionally, it contains specifications about the flow to be reserved.
- Since the `Path` message is sent to the receiver directly, it has to be specially marked so that intermediary routers can intercept the message. This is done by setting the *Router Alert* flag that initiates the initialisation of soft states

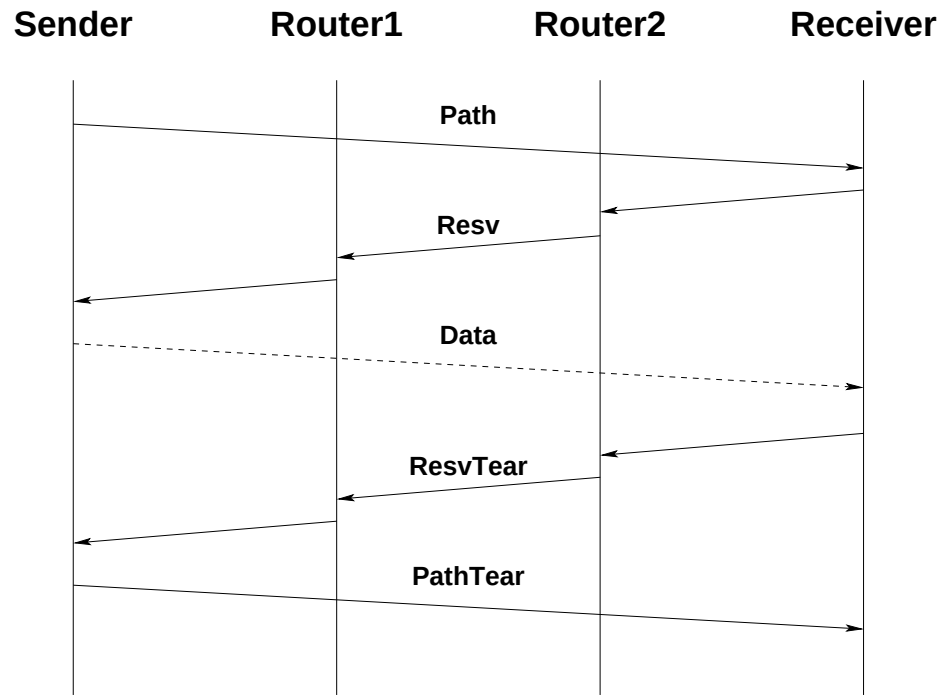

Figure 2.2: Diagram of a Normal RSVP Reservation Setup and Clearance

at each router. In addition, each router updates the information about the last RSVP router in the `Path` message. Therefore, after the `Path` message has arrived at the receiver, each router on the path knows the RSVP router preceding itself.

Since the generated soft states have a limited TTL, the `Path` message has to be repeated regularly. In order to avoid overloading the network, the interval is usually set to 30 s. To protect the reservation against being cleared because of a single loss of a `Path` message, the TTL of the soft states is thrice the interval of the `Path` messages. By repeating `Path` messages it is also possible to react to routing changes since the forwarding paths of the `Path` message and the data are identical.

- Following the `Path` message the `Resv` message performs the reservation of resources in the routers. In contrast to the `Path` message this message is sent hop-by-hop to the next RSVP router (each router knows the next hop because of the preceding `Path` message). At each RSVP router the possibility of setting up the reservation is checked. If the reservation is accepted the `Resv` message is forwarded, otherwise an error message is created and sent back to the receiver.

The receiver decides about the amount of resources to be allocated. Usually, the resources the sender has specified in the `Path` message are configured. The `Resv` message has to be repeated periodically, too.

- As soon as the `Resv` message has arrived at the sender, the sender can start the transmission of data over the newly established *unidirectional* reservation. If the sender transmits more data than allocated, this data is forwarded as best-effort.
- After the transmission is finished, there are multiple possibilities to end the reservation. The most simple one is not to repeat `Path` or `Resv` messages any more. As soon as the soft states are not renewed they are deleted. On the other hand one can also explicitly delete a reservation by releasing the resources via a `ResvTear` message or by deleting the soft states via a `PathTear` message.

Aggregation of Reservations A further property of RSVP is the support of multicast: in Figure 2.3 a router with four interfaces is shown. On the left hand side of the interfaces are the senders (S1,S2,S3), on the right hand side are the receivers. If sender S1 now transmits a packet to the receivers R1 and R2 the data is passed on to two different interfaces by the router. The contrary happens to the reservations: since the receiver asks for a reservation it also decides on the amount of the reservation. Both requests from the receivers R1 and R2 arrive at the router. If both requests would simply be forwarded and one of the senders would not need as much bandwidth as the other, the resulting reservation would be increased and decreased regularly. In order to prevent this, multiple reservations at a split point of a multicast connection are combined so that only one message is forwarded and the reservation is as large as the biggest reservation received.

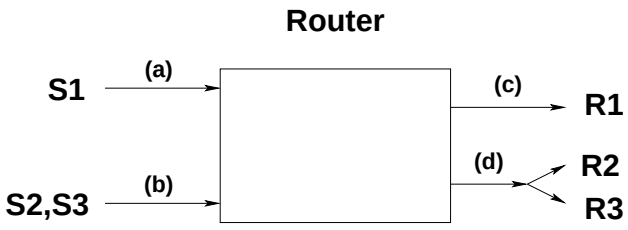

Figure 2.3: Split Point in a RSVP Multicast Connection

In addition, RSVP supports combining reservations from multiple senders to multiple receivers. Typical application scenarios would be telephone- or video conferences. One possibility would be to set up reservations for each possible sender

- receiver pair. This would result in a relatively large consumption of resources. Therefore, RSVP offers the possibility to combine multiple reservations to one. The end-user decides, how this has to happen. There are three possibilities, shown in Table 2.1:

Sender Choice	Reservation	
	Separated	Shared
Explicit	Fixed-Filter (FF)	Shared-Explicit (SE)
Wildcard	—	Wildcard - Filter (WF)

Table 2.1: The 3 Reservation Possibilities

It is apparent that there is — for obvious reasons — no possibility for a separated reservation with a wildcarded sender choice. An example of how the combination of reservations is performed is shown in Table 2.2. In this example we assume the following connections:

- S1 $\rightarrow$ R1,R2
- S2 $\rightarrow$ R2,R3
- S3 $\rightarrow$ R2

	OUT	Resv	IN	
(a)	SE(S1{3U})	(S1,S2) {U}	SE((S1,S2) {U})	(c)
(b)	SE((S2,S3) {3U})	(S1,S2,S3) {3U}	SE((S1,S3){3U}) SE(S2 {2U})	(d)

Table 2.2: Combination with Shared Explicit

We observe that at interface (d) the different senders are combined and the biggest reservation request is taken, which means for the data from senders S1, S2, S3, three units are reserved. Problems can arise, if all senders are transmitting at full bandwidth at the same time. Therefore, sharing reservations is only useful if not all senders are sending at the same time. This implies that WF and SE should only be used for audio transmission, for video transmission FF has to be used.

2.1.3 RSVP and Mobile IP Interworking

Interoperation between RSVP and Mobile IP is investigated in [57]. The main problem occurs, when Mobile IP operates over optimised routes: The address

translation in the binding cache creates a mismatch between the flow ID of the packets sent from the correspondent host to the mobile host and the flow ID signalled by RSVP. The authors discuss two solutions: Modifying RSVP at both, mobile and correspondent hosts, allows RSVP flows to be established. However, the authors recommend to add optional objects to RSVP messages to make handovers smooth and seamless. The same goal can be achieved with the second solution, using fixed flow IDs.

Mobile RSVP (MRSVP) has earlier been proposed in [166] as an extension to conventional RSVP. The idea was to provide QoS guarantees to mobile hosts independently of their movement throughout the access network. MRSVP suggests to make the required resource reservations in all the locations expected to be visited by the mobile host (mobility specification). MRSVP supports two service classes. The first one, called Mobility Independent, provides the agreed service in every location visited by the mobile host. The second, called Mobility Dependent, provides a high probability, but no guarantee, to receive the agreed service in the locations the mobile host may visit. MRSVP supports two different reservation styles. The mobile host makes active reservations from its present location towards all the correspondent hosts it is communicating with. It also triggers the establishment of passive reservations from all the locations it may visit. Such reservations are made by proxy agents (remote), operating on the behalf of the mobile host which is not present at their subnetwork. As passive reservations are not used by the mobile host (data is not flowing through them), they may be used temporarily by other connections of the Mobility Dependent class. When the mobile host roams, passive reservations are switched to active and vice versa.

[168] describes an approach for providing RSVP protocol services over IP tunnels. Currently, RSVP signalling over tunnels is not possible. RSVP packets entering the tunnel are encapsulated with an outer IP header that has a new protocol number and do not carry the Router-Alert option, making them virtually ”invisible” to RSVP routers between the two tunnel endpoints. The proposed solution is to recursively apply RSVP over the tunnel portion of the path. In this new session, the tunnel entry point sends PATH messages and the tunnel exit point sends RESV messages to reserve resources for the end-to-end sessions over the tunnel.

Conclusion Some feasible solutions for the interoperation of Mobile IP and RSVP seem to be available, however, the basic drawback of RSVP — the scalability — remains. Therefore, new solutions have to be found that provide scalability in the backbone and cooperate with Mobile IP.

2.2 Differentiated Services

Differentiated Service (DiffServ) is a well known way to support Quality of Service in the Internet. It is — unlike Integrated Services — based on the aggregation of flows by applying rules at the edges of a DiffServ domain. An aggregate is marked with a specific DiffServ Codepoint (cf. Section 2.2.1), which is coupled to a certain forwarding path treatment in the domain. By restricting the service differentiation at each hop to the few possibilities the codepoint offers, we do not need multi-field classification at each hop any more. This is the reason why DiffServ does not suffer from the scalability problem of IntServ and RSVP, since a core router has no detailed information about how to handle each single flow but only the information about the traffic conditioning of a very limited set of aggregates.

Reservations are made for any aggregation of flows (e.g. for all flows between two subnets). These reservations are rather static since no dynamic reservations for a single connection are possible for scaling reasons.

2.2.1 Differentiated Services Codepoints

IP packets are marked with different priorities, either in an end system or in a router. According to the different priority classes, the DiffServ routers reserve corresponding shares of resources (i.e. bandwidth and buffer space). Marking the packets is done by writing a DiffServ Codepoint (DSCP) into the Type of Service - byte (ToS byte) of the IP header (see figure 2.4).

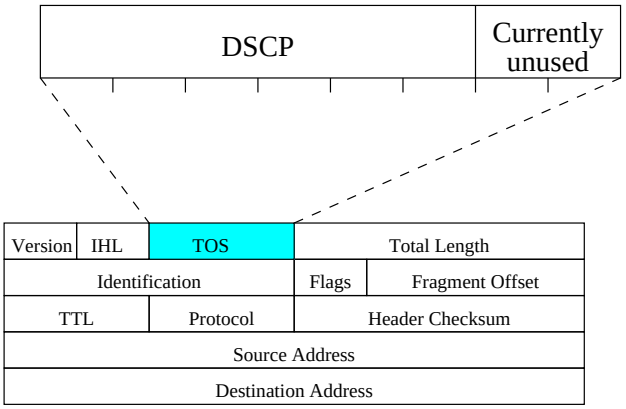

Figure 2.4: DiffServ Codepoint in the IP Header [124]

Currently the first six bits of the ToS byte are used for the DSCP. All router implementations should support the recommended DSCP-to-PHB mapping. A PHB (Per Hop Behaviour, cf. Section 2.2.3) is a forwarding behaviour which a router performs on a packet. In DiffServ such a forwarding behaviour is built of a combination of several components. These components are discussed in section 2.2.5. A chain of routers supporting the same PHB will provide an end-to-end Quality of Service.

2.2.2 Service Level Agreements

In the initial version of the DiffServ Architecture [16] a Service Level Agreement (SLA) describes a contract that specifies the services for a customer. Such a SLA may include a Traffic Conditioning Agreement (TCA) which in turn is an agreement specifying a set of rules of how to configure the forwarding elements of a router, for example shapers or queues. Later, people more and more believed, that an *agreement* also includes terms of pricing and other business contracts, as well as other completely technical specifications that are not addressed by the DiffServ Architecture (e.g. service availability). Therefore a new terminology has been introduced to restrict their meaning only to elements addressed by DiffServ (cf. [66]):

- A Service Level Specification (SLS) is a set of parameters and their values which together define the service offered to a traffic stream by a DiffServ domain.
- A Traffic Conditioning Specification (TCS) is a set of parameters and their values which together specify a set of classifier rules and a traffic profile. A TCS is an integral element of a SLS.

2.2.3 Per Hop Behaviour (PHB)

Several services and their corresponding codepoints have been defined in the Internet community. Nowadays, DiffServ is mainly based on the two traffic classes defined in [74] and [78]. We will now give a short overview of those two classes.

Expedited Forwarding / Premium Service

Premium Service refers to the traffic handling commonly known as expedited forwarding which is defined in [78]. This service shall provide low delay, low loss

Dropping Precedences	AF Classes: Class 1	Class 2	Class 3	Class 4
low	001010	010010	011010	100010
medium	001100	010100	011100	100100
high	001110	010110	011110	100110

Table 2.3: Assured Forwarding Codepoints

and low jitter at a fixed rate. It will appear to the endpoints like a “virtual leased line”. To fulfil those requirements, traffic marked for expedited forwarding has to meet very short queues. Therefore it has to be ensured that there is not more EF-traffic arriving at a router than the router settings allow to be transported. The departure rate at each hop should be independent of the intensity of any other traffic arriving at the router. DiffServ routers will for example give premium service packets priority over other traffic but in this case strictly police any traffic exceeding the negotiated limit to prevent premium service to starve any other traffic. Another very important topic is that reordering of EF packets must not appear. The recommended codepoint for the EF PHB is 101110.

Assured Service

Assured Forwarding defines a service which assures a high probability to forward the traffic through the network as long as the bandwidth does not exceed the negotiated limit. However, any traffic exceeding the profile will be forwarded too, but with higher probability to be dropped in case of congestion. It is also very important, that reordering of packets of the same microflow is strictly forbidden again.

There are four different assured service classes defined, each allocating a specific share of resources (i.e. bandwidth and buffer space) and thus having a different level of forwarding assurance. Within these classes packets can be marked with three possible drop precedence values. In case of congestion the in-profile traffic will be protected by preferably dropping packets with higher drop precedence. An overview of the codepoint values for the assured forwarding classes is given in table 2.3.

2.2.4 Per Domain Behaviour (PDB)

Scalable end-to-end QoS provisioning requires a definition of the behaviour of packets when they are grouped together with other packets as they traverse the Internet. The handling of packets within such an aggregation will be based on their DSCP only. A Per-Domain Behaviour (PDB) is defined in [125] as the expected treatment that an identifiable or target group of packets will receive from "edge to edge" of a DiffServ domain. A particular PHB (or, if applicable, list of PHBs) and traffic conditioning requirements are associated with each PDB.

Today, there exist two preliminary proposals for per domain behaviours: the virtual wire PDB and the assured rate PDB. The two definitions are closely related to the two existing PHB standards.

Virtual Wire PDB The expedited forwarding PHB [78] described above is intended for use in building a scalable, low loss, low latency, low jitter, assured bandwidth, end-to-end service that appears to the endpoints like a unshared, point-to-point connection or 'virtual wire.' The virtual wire PDB [79] provides the specifications necessary on that aggregated traffic in order to meet these requirements.

Creating a virtual Wire PDB has two parts:

1. Configuring individual nodes so that the aggregate has a well-defined minimum departure rate
2. Conditioning the entire DS domain's aggregate (via policing and shaping) so that its arrival rate at any node is always less than that node's configured minimum departure rate.

The first part is covered by the EF PHB. The second part is covered by [79], that describes how one configures the EF PHBs in the collection of nodes that make up a DS domain and the domain's boundary traffic conditioners (see [16]).

Assured Rate PDB This PDB [148] ensures that traffic conforming to a committed information rate (CIR) will incur low drop probability. The aggregate will have the opportunity of obtaining excess bandwidth beyond the CIR but there is no assurance. In addition to the CIR, the edge rules may also include other traffic parameters such as the peak information rate (PIR) to place additional constraints for packets to which the assurance applies or to further differentiate packets which exceed the CIR.

The possibility of obtaining excess bandwidth allows development of various novel SLA models. For example, excess bandwidth is charged at a higher rate

than assured bandwidth; excess bandwidth is cheaper than assured bandwidth; excess bandwidth is charged proportionally etc.

Applicability of the PDB to a particular application or traffic type is not restricted. However, it is also possible to use this PDB to create a service for an aggregate consisting only of TCP microflows or non-responsive UDP microflows. The provider may wish to create a TCP-only service for a variety of reasons such as traffic isolation, better treatment of individual short microflows within an aggregate, greater fairness among TCP and UDP microflows access to the excess bandwidth allowed for the aggregate.

2.2.5 DiffServ Components

Each implementation of a DiffServ router consists of a combination of different components (see Figure 2.5, [16]), which interact in a certain way to ensure the proper forwarding of traffic according to the requirements of the individual PHB. Not all components are required in each DiffServ node, this depends on the router and on the service type. Those components provide different traffic conditioning functions that range from simple marking to complex shaping and policing actions.

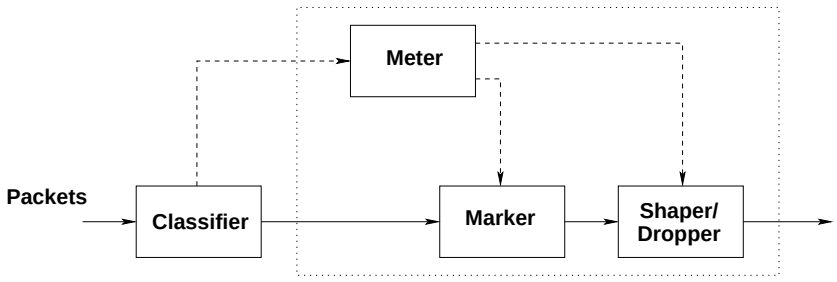

Figure 2.5: Combination of DiffServ Components in a Router [16]

Classifier A classifier matches packets according to its profile and forwards them to the corresponding component for further processing. There are two types of classifiers:

- **Multi-Field Classifier**: A multi-field (MF) classifier matches on a combination of IP header fields (addresses, protocol ID, ToS byte) or even port numbers.

- **Behaviour Aggregate Classifier**: Contrary to the MF classifier the behaviour aggregate (BA) classifier only classifies on the DSCP of the packets.

Marker Markers set the DSCP of IP packets. By setting the codepoint of packets, they are added to a traffic aggregate which provides important information for BA classifiers. Marking can be done statically (i.e. all packets are marked the same way) or depending on the state of some meter. This functionality is normally used in ingress routers for tagging the traffic flow of a single host or for re-tagging whole traffic aggregates coming from a connected DiffServ domain.

Meter Meters measure the amount of traffic that passes by. They are located in the forwarding chain of almost all traffic aggregates as they provide the basic information for many DiffServ components, like markers, shapers and policers.

Shaper Shapers delay some packets on their transmission path in order to bring the traffic flow into compliance with some Traffic Conditioning Specification (TCS). Those packets are stored in some queue and discarded if not enough buffer space is available. A properly configured shaper therefore provides some burst protection while not dropping the packets of the bursty traffic source. The usual location of a shaper is behind a classifier at the ingress router.

Policer Unlike the shaper a policer does not store any packets but simply drop any packets that do not meet the traffic conditioning specification. A policer can be implemented as a shaper with little or no buffer space. Policers are normally used in interior- and egress routers as they rely on the traffic being correctly shaped.

A higher-level view of a DiffServ router is presented in Figure 2.6 [12]. We can identify several functional blocks:

Datapath An ingress interface, routing core, and egress interface are illustrated at the centre of the diagram. The routing core moves packets between interfaces according to policies outside the scope of DiffServ. The components of interest at the ingress to and egress from interfaces are the functional datapath elements (e.g. Classifiers, Queueing elements) that support DiffServ traffic conditioning and per-hop behaviours. These are the fundamental components comprising a DiffServ router.

Configuration and Management Interface DiffServ operating parameters are monitored and provisioned through this interface. Monitored parameters include statistics regarding traffic carried at various DiffServ service levels. These statistics may be important for accounting purposes and/or for tracking compliance to Traffic Conditioning Specifications (TCSs) negotiated with customers. Provisioned parameters are primarily the TCS parameters for Classifiers and Meters and the associated PHB configuration parameters for Actions and Queueing elements. The network administrator interacts with the DiffServ configuration and management interface via one or more management protocols, such as SNMP or COPS, or through other router configuration tools such as serial terminal or telnet consoles.

Optional QoS Agent Module DiffServ routers may snoop or participate in either per-microflow or per-flow-aggregate signalling of QoS requirements [13] (e.g. using the RSVP protocol). Snooping of RSVP messages may be used, for example, to learn how to classify traffic without actually participating as a RSVP protocol peer. DiffServ routers may reject or admit RSVP reservation requests to provide a means of admission control to DiffServ-based services or they may use these requests to trigger provisioning changes for a flow-aggregation in the DiffServ network. A flow-aggregation in this context might be equivalent to a DiffServ BA or it may be more fine-grained, relying on a multi-field (MF) classifier. Note that the conceptual model of such a router implements the Integrated Services Model as described in [INTSERV], applying the control plane controls to the data classified and conditioned in the data plane, as described in [13].

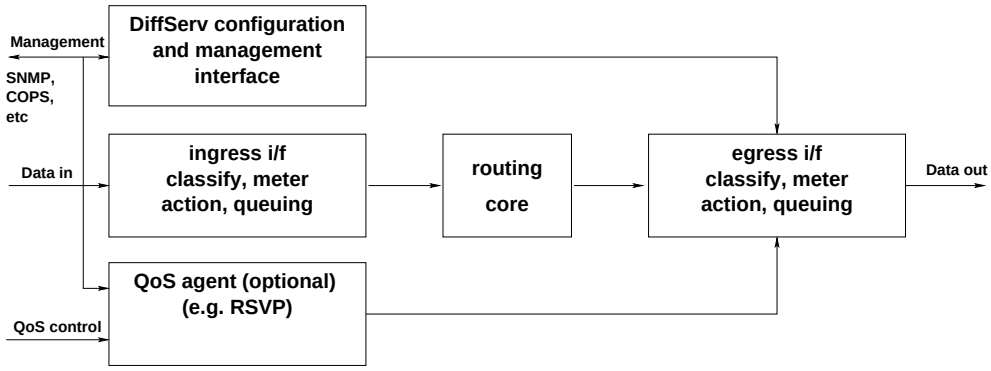

Figure 2.6: DiffServ Router Major Functional Blocks [12]

2.2.6 DiffServ Router Types

A simple DiffServ Network with 2 ISPs is shown in figure 2.7. It shows the four DiffServ Router types, that form the two DiffServ domains and connect the two hosts and provide an end-to-end Quality of Service. This router specifications are valid only for traffic from domain A to domain B. For traffic flowing in the opposite direction the boundary routers (ingress and egress routers) change their types. Note, that a router may also act in two ways, for example as interior router for one traffic flow while being ingress router for another. This can be seen in figure 2.7 for the router adjacent to Host B. The next sections describe the requirements of these four router types which follow the DiffServ architecture [16].

First Hop Router

A first hop router normally is responsible for marking incoming packets according to its profile. This profile allows to define a DiffServ Code Point (DSCP) for each flow specified by the six-tuple (source address / netmask, source port, destination address / netmask, destination port, protocol, DSCP). For small networks this functionality can be easily included in an ingress router.

Ingress Router

The configuration of the ingress router is the most complex one because at the ingress point each flow has to be handled separately and therefore not all properties of flow aggregation are available. The ingress router has to ensure, that the traffic entering the DiffServ domain conforms to any traffic conditioning specification (TCS) between it and the connected domain. Therefore, it will have to perform some traffic conditioning functions like shaping or dropping.

Interior Router

Routers not located at the border of a DiffServ domain can be very simple, as the most complex functions of classification and traffic conditioning are performed at the ingress and egress points of the network. However an interior node may perform some limited traffic conditioning like codepoint-remarking or policing to ensure the proper forwarding behaviour of the traffic classes. Normally the interior router should not have to perform any policing actions, therefore it is useful to trace those actions to detect serious misconfigurations at the border routers.

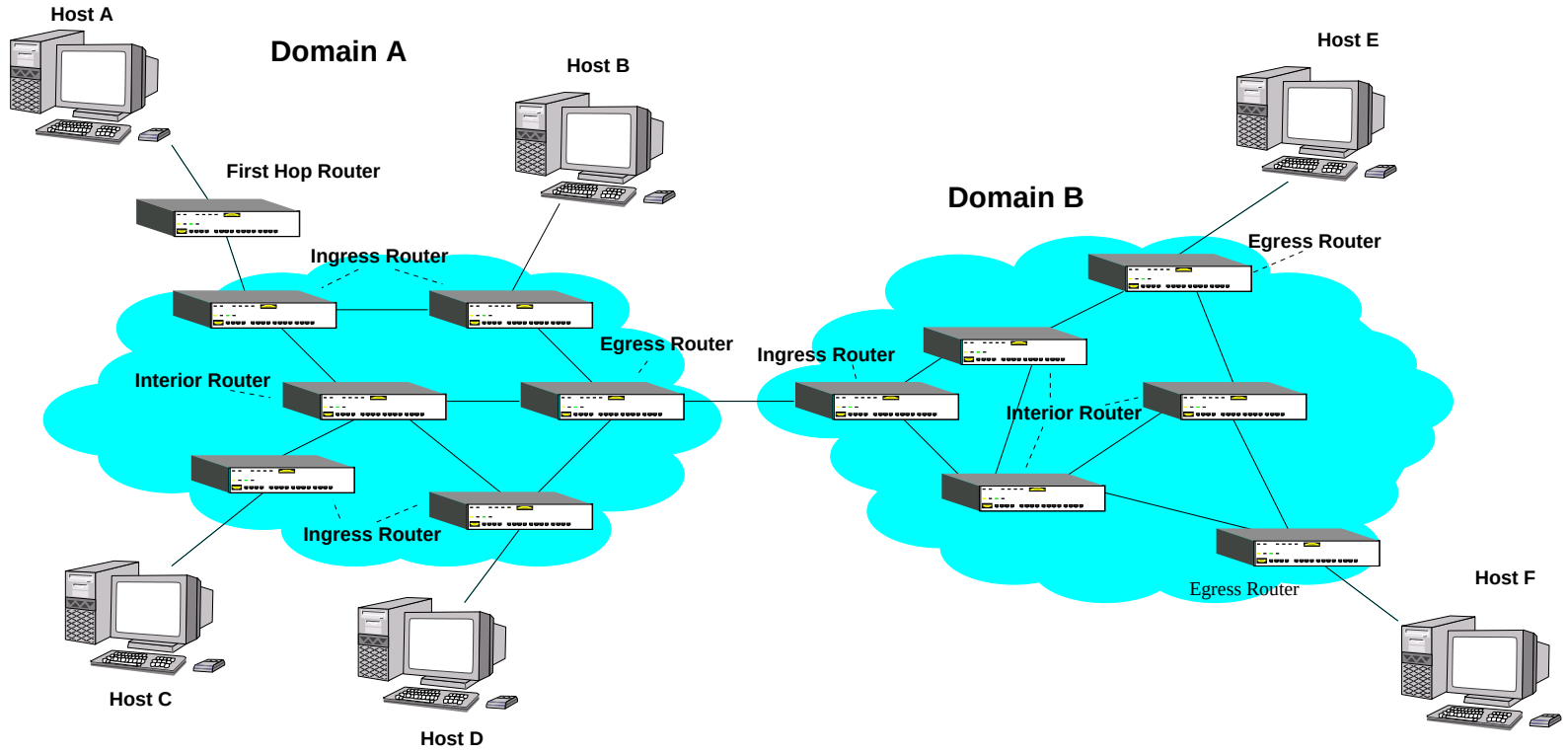

Figure 2.7: A simple DiffServ Network

Egress Router

An egress router can perform traffic conditioning functions on traffic leaving the DiffServ domain depending on the TCS between it and the connected domain. This functions normally will not depend on multifield classification but act on a behaviour aggregate. Therefore the configuration of the egress router is less complex than the configuration of the ingress router.

2.2.7 Performance Evaluation of Differentiated Services

Differentiated Services have been implemented for several platforms and multiple evaluations have been published. In addition to our own implementation, that we present in Chapter 4 and evaluate in Chapter 5, we want to present some other evaluations in this section.

An own implementation of Differentiated services has been evaluated in [9]. This DiffServ implementation offers three levels of QoS, and the available bandwidth is shared between those levels in a ratio of 1:2:4. The implementation has been performed on SUN Sparc Ultra 5 workstations. For the tests, two workstations have been connected with a 10 MBit/s Ethernet (effective throughput: 7 MBit/s). During the experiments the authors created 3 traffic streams (TCP and UDP) with a larger total bandwidth than the link was able to handle (8 –10 MBit/s). The authors were able to show, that the relative bandwidth of the three flows could be preserved. In further experiments they were able to show this for several flows in the same DiffServ class. If one traffic stream dynamically changed its SLS from level 1 to level 2, the relative share of the classes has been preserved, however the bandwidth of each single traffic stream in such a scenario can change dramatically.

In [145] the authors describe the behaviour of a DiffServ network consisting of Cisco 7200 routers. Several traffic generators with different transport protocols (UDP/TCP), burstiness, and application-like behaviour have been used. The authors evaluated two services: Premium and Olympic Service. Those two services are built on the EF and the AF PHB (cf. Section 2.2.3). The experiments showed a good delay of about 1 ms for both services in a not congested scenario. If the classes of Olympic Service are congested, the delay increases to 7 –22 ms, depending on the service class. In Premium Service congestion has no effect on the delay. Bandwidth measurements were only performed for TCP in the Bronze class (the lowest class) of Olympic Service. The results show that the configured amount of bandwidth is provided to the TCP stream regardless of bursts. This holds for different queue lengths.

Conclusion The authors of [9] have presented an own, class-based DiffServ implementation. They were able to show, that their implementation preserves a relative share of the bandwidth for each class in several scenarios. However, the bandwidth of each single traffic flow depends on the presence of other flows in its service class. Therefore, we believe, that this implementation is not well suited for modern multi-media applications, that rely on a constant delivery of QoS.

The second evaluation [145] has shown results for the DiffServ implementation of Cisco routers. Similarly to [9], the authors measured DiffServ behaviour in case of congestion *within* a traffic class. This is an improbable scenario, since proper traffic conditioning at the ingress of a domain should prevent this case. The delay results in the case of no congestion show the same behaviour as our performance tests: there is almost no difference between EF and AF behaviour in a properly provisioned network.

2.3 QoS-enabled IEEE 802.11 Wireless Access Networks

The IEEE 802.11 standard for wireless LANs is the most widely used method for WLAN communication today. Two services exist in this standard: the *distributed Coordination Function* (DCF) that supports delay-insensitive data (like FTP, e-mail) and the *Point Coordination Function* (PCF) which is optional and supports delay sensitive transmissions. The former function supports equal sharing of the channel between several hosts via CSMA/CA: a station with a packet ready for transmission waits a certain interval and only transmits the packet if the channel was idle throughout this interval. The latter function is a centralised polling-based approach that avoids contention by polling the mobile nodes individually for transmission. Naturally, this function offers very good possibilities to introduce service differentiation in wireless LANs: [106] studies the PCF among three other priorisation schemes: Distributed Fair Scheduling (DFS) [178], which uses the backoff mechanism of IEEE 802.11 to determine which station should send next. Each station has a backoff interval length according to its priority, so differentiation is granted while fairness is achieved by making the interval proportional to the packet size. Blackburst [153] has been designed with the main goal of minimising delay for real-time traffic. This scheme requires a high-priority station to access the channel at fixed time intervals. Furthermore a station may jam ("blackburst") the channel for a certain period of time if the medium was found busy. This ensures the channel being free for transmission after the burst. Finally, Enhanced DCF is included in the study, too. This access mechanism is part of the IEEE 802.11e standard.

The authors performed some ns-2 simulations in which they show, that Blackburst gives best performance to high-priority traffic but starves low priority traffic at high loads. EDCF is shown to perform better than PCF but also suffers from the starvation of low-priority traffic at high loads. Finally, DFS ensures better service to high priority traffic and still provides low priority traffic a fair share of the resources.

A new wireless data transfer protocol is introduced in [150]. Two new entities — the Mobile Network Adaption Terminal and the Mobile Network Adaption Master — are proposed. They ensure reliable data transfer, handover support and flow control. Unfortunately, no evaluation of the protocol is included that would allow a detailed interpretation of the benefits of the new protocol.

Another approach is presented in [1, 2]: The authors evaluate differentiation schemes based on different parameters of the IEEE 802.11 standard: the *DCF Inter Frame Space* (DIFS), Backoff Time, Maximum Frame Length, and Contention Window Size. Assigning different values of those parameters to wireless terminals provides service differentiation at the MAC layer using DCF only. This approach has the advantage of being applicable to ad-hoc networks too, since no centralised control is needed. In [1] the authors present some ns-2 simulations that show how TCP and UDP flows react to those priorisation schemes. The results show minor stability of the Backoff Time scheme and performance degradation in noisy environments for the Backoff Time and Maximum Frame Length scheme. Concluding, the authors propose DIFS differentiation.

The work presented in [105] follows a different approach: The authors propose to use two different service classes called instantaneous allocation and stable allocation. Instantaneous allocation provides better throughput while stable allocation provides better allocation stability. The cheaper instantaneous allocation class is intended to be used for short-lived connections. On the other hand an application can use the more expensive stable allocation class, if it is willing to trade lower bandwidth for better allocation stability. The authors present an implementation of those two service classes based on a Linux PC serving as an access point and the Linux `tc` (traffic control) function to restrict the bandwidth. A pricing scheme is presented and evaluated that — under certain assumptions — leads to cooperative user behaviour.

Conclusion The discussion presented here mainly focuses on per-terminal differentiation: a wireless terminal gets a certain priority assigned via a specific scheme. However, a wireless terminal may not be willing to send all its traffic with a higher priority (e.g. because of financial reasons). In [2] the authors apply differentiation schemes discussed earlier (cf. [1]) to per-flow differentiation.

Unfortunately, while UDP flows show a rather good differentiation, TCP flows perform badly because of the priority of the TCP-ACKs being constantly the priority of the base station.

2.4 Signalling Protocol for Differentiated Services

A survey of the main Internet Quality of Service protocols can be found in [149]. This paper defines several classification criteria, like scalability, complexity, and adaptability. The existing protocols are classified as follows:

RSVP The Resource ReserVation Protocol (cf. Section 2.1.2) has two major problems: complexity and scalability. In a backbone the bandwidth consumed by periodical refresh messages and the storage space needed to support a large number of flows is too large.

YESSIR YESSIR (YEt another Sender Session Internet Reservations) [130] generates reservation requests by senders to reduce the processing overhead, builds on top of RTCP, uses soft state to maintain reservation states, supports shared reservation and associated flow merging and is backward compatible with the IETF Integrated Services models.

Beagle Beagle [38] provides a way for applications to optimise resource allocation by expressing a wide range of policies to share resources amongst its flows. It allows applications to allocate computational and storage resources in the network.

COPS COPS is an out-of band protocol that is very useful for dynamic QoS environments. The drawback of COPS is its increasing complexity for large networks: More devices involve more PEPs (Policy Enforcement Point). Also, the complexity of the PIB (Policy Information Base) at each PDP (Policy Decision Point) is increasing.

The protocols discussed in this short summary unfortunately do not take into account the special needs of mobile users: the frequent changes of reservations that result from the handovers can get a bottleneck in the protocol even if only a small portion of the whole path is changed. Any protocol suitable for mobile users should support handovers with as little overhead of network signalling and reconfiguration as possible.

The main requirements of a protocol for mobile users are listed in [170]:

- Minimum changes to static nodes. A QoS protocol for mobile nodes must leave static nodes unchanged as far as possible.
- Support for *soft* QoS guarantees. In a highly dynamic mobile network QoS guarantees may not be easy to fulfil (e.g. due to external interference). Applications should be able to adapt to changing network conditions over the wireless link.
- Minimum disruption of service. If one tried to re-establish the full path of the reservation each time the mobile node changes the access network, reservations would have to compete (and possibly fail) for resources. This situation would create excessive disruptions of service and large latency.

The authors also demand a RSVP-like protocol since they want to develop a signalling protocol for an IntServ network. Their protocol works by combining pre-established RSVP tunnels with Mobile IP. RSVP messages are encapsulated in a tunnel connecting home and foreign agents. One big drawback of this protocol is, that it requires modifications in Mobile IP *and* RSVP to support resource reservations from mobile nodes. We believe that the development of a signalling protocol that handles mobile users without any changes in existing standard technology is possible. Such a protocol is described in Part IV

A most detailed list of requirements for QoS signalling protocols can be found in [55]. This Internet draft lists over 50 requirements, grouped into 11 categories. Unfortunately, in this document the requirements for mobile users are addressed very shortly: The only requirement is a quick re-establishment of the reservation after a handover. The authors also propose not to perform end-to-end signalling in such a case. Those requirements are fulfilled by our protocol.

2.4.1 Cross Application Signalling Protocol (CASP)

The Cross-Application Signalling Protocol (CASP) [147] is a general-purpose protocol for managing states in routers and other on-path network devices. It can be used for QoS signalling, middlebox control, topology discovery, measurement data collection, active network instantiation, and any other application where states needs to be established along a data path. CASP consists of a set of building blocks that can be used to construct protocol behaviour suited for a particular application.

It provides a generic signalling service by establishing state along the data path from a sender to one receiver for unicast data, or to multiple receivers for multicast data. CASP sessions can be initiated by the sender or by the receiver.

The following properties of the protocol are citations from [147]. Many other properties are listed in the specification, here we restrict to some of the most important:

- CASP is a layered protocol with two layers, the client and messaging layer. Each can be changed without affecting the other component.

- CASP uses the services of a reliable transport protocol that provides sequenced, reliable, flow- and congestion- controlled message transport between two CASP nodes.

- CASP provides a generic soft-state mechanism that can be used by all client protocols. Soft state is only used for logical state, not to deal with packet loss. To maintain soft state, requests are simply resent periodically by each node.

- CASP interfaces with route change detection mechanisms; IP mobility is also treated as a route change case.

2.4.2 Context Transfer Protocols

The topic of context transfer is discussed in detail in the IETF SeaMoby Working Group. This group investigates possibilities to develop seamless handovers which means handovers with both, low latency and minimal packet loss. In this case, a context is understood as information about services in the old network, such as AAA, header compression, and QoS. In order to obtain those services on a new network the host explicitly has to re-establish the services by performing the needed signalling messages from scratch [53]. This obviously will slow down the handover process. An alternative is to transfer the context information to the new network by using the context transfer protocol.

Such a context transfer protocol (CTP) is presented in [96]. This protocol defines several messages (context transfer initiate, context transfer request, context transfer reply, etc.) that can be used to perform the context transfer. Those messages can be used by trusted entities, such as the mobile node itself or a network server overseeing the mobile hosts handover. The protocol fulfils the requirements specified in [64], such as Layer 2 interoperability, flexibility (i.e. independence of content type), security and speed, to name just a few.

2.4.3 Conclusion

Although there are many existing QoS signalling protocols, most of them do not take into account the special needs of mobile users. The impact of frequent but small changes of the forwarding path, as it is common in handovers, is not investigated in detail. Most performance evaluations are done in a simulated network of a very limited size only. In Chapters 7 and 8 we will present a signalling protocol for cable-connected and for mobile users that allows fast changes in the forwarding path, especially in case of handovers.

The context transfer protocol is related to the possibility of our protocol to support SLS transfers from the old to the new access network (cf. Section 8.2.2). In our actual implementation we focus on transferring QoS information coded in a Service Level Description. This is called a QoS Profile Type (QPT) in [96]. If a extension to additional Context Profile Types, such as Header Compression Profile Types (HPT) is required, our architecture will support this by upgrading the Service Level Description Class with the necessary information. This can be done easily via derivation (cf. Section 6.1).

The CASP protocol is a very new protocol which takes many requirements of a general-purpose signalling protocol into account, such as mobility service discovery and multicast. However, due to its large functionality it uses a large signalling message format and a complicated architecture. It would be useful to estimate the benefits of this new protocol in a implementation and evaluation.

Chapter 3

Network Management

Network management is a topic that covers a broad range of possible application scenarios. At a very low level, it is sufficient to monitor the network performance with a protocol analyser, but in large networks, management involves distributed databases, automatic polling of network interfaces, and high-speed workstations to perform the necessary computations and to visualise the network topology and performance. The Internet Architecture Board (IAB) issued a RFC [37] detailing its recommendation, which adopted two different approaches:

- Simple Network Management Protocol (SNMP)
- ISO Common Management Information Services / Common Management Information Protocol (CMIS / CMIP)

SNMP became most popular and is therefore now industry-wide standard for reporting management data for an IP based network.

3.1 Network Management Architecture

Most network management architectures are based on a similar structure and on similar relationships between the components. Network elements, such as routers, switches or hosts, run software that enables them to alert a central management station when they recognise problems. Receiving these alerts the management station executes a certain procedure, for example logging, operator notification, automatic repair, shutdown, etc.

Management stations can by themselves poll the network elements to gather data for monitoring, statistics etc.

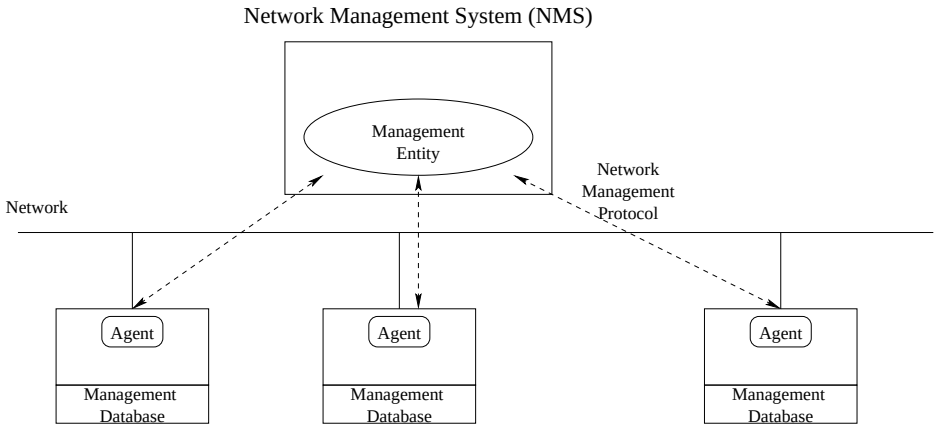

Figure 3.1: A Typical Network Management Architecture [76]

3.2 ISO Network Management Model

The network management model of the ISO is the primary means for understanding the major functions of network management systems. It consists of five conceptual areas, as discussed in the next sections [76]:

3.2.1 Performance Management

The goal of performance management is to measure and provide various aspects of network performance, in a way that an acceptable level of internetwork performance can be achieved. Examples of performance variables might be the network throughput, user response time, and utilisation.

Performance management can be divided into three steps: first performance data is gathered from the network. Second, this data is analysed to determine the normal state of the network. Finally, appropriate thresholds are defined for each variable in a way that exceeding this threshold indicates a serious problem. The functions of performance management include:

- Monitor performance indicators
- Activate controls to fine-tune network performance
- Generate reports and trend analysis

3.2.2 Configuration Management

The goal of configuration management (CM) is to monitor network configuration information. CM functions identify, control, and collect data for and provide data to networks for status accounting and auditing. Those tasks are closely related to both fault management and performance management for long-term planning of the network's topology, information processing systems configuration and inventory. CM functions may include:

- Initialise and terminate system operations
- Establish network connections
- Maintain real-time configuration status
- Distribute communication element network software
- Generate reports
- System administration

3.2.3 Accounting Management

Accounting management measures network utilisation parameters, so that individual users or groups of users can be regulated appropriately. This regulation minimises network problems (because network resources can be allocated) and maximises fairness across all users.

Similar to performance management it is important to measure all significant utilisation parameters of network resources. An analysis of this results provides usage patterns that can be used as a basis for usage limitations. Ongoing measurement of utilisation can produce billing information as well as information to assure fair and optimal resource utilisation.

The main steps of accounting management are:

- Specifying the usage data to collect
- Establishing and modifying accounting limits
- Collecting and storing usage data
- Controlling access and storage of usage data
- Report generation

3.2.4 Fault Management

Fault management tries to detect, log and as much as possible automatically fix network problems. Additionally the users have to be notified of important problems. Since faults can cause down-time or unacceptable network degradation, fault management is the perhaps most widely implemented form of network management.

Fault management involves:

- Prediction, detection and identification of faults
- Diagnostic testing
- Fault verification
- Error and event log review
- Fault correction

3.2.5 Security Management

Security management controls the access to network resources according to local guidelines. Its goal is to prohibit sabotage (intentionally or unintentionally) and assure that sensitive information cannot be accessed without appropriate authorisation. Security management subsystems part the network into authorised and unauthorised areas. For some users, access to any network resource is inappropriate, mostly because they are extern users. For other (internal) network users access to information outside a particular department may not be allowed. Access to human resources information has to be restricted to the users of the HR department.

Specific tasks of security management subsystems include:

- Authentication
- Authorisation
- Encryption
- Controlling the protection mechanisms
- Controlling access to resources and security information

3.3 The IEEE P1520 Standards Initiative

The IEEE P1520 proposed standard [15] aims to establish an open architecture in network control and management. The initiative sees architecture as a levelled entity with interfaces between different layers, each of them representing a certain resource type within the whole network architecture. The rationale behind having interfaces between different layers is to create a distributed programmable concept that interfaces to network abstractions that in turn allow signalling and service programming software to be realised in a distributed software environment with well-known paradigms. The interfaces proposed in the P1520 initiative allow views of network and switching hardware states to be utilised by independent and flexible signalling service creation.

In the reference model there are levels, entities in each of the levels and interfaces between levels. Above all of the levels there are the applications that are directly deployed by the end users. The reference model contains four levels, each of which presents certain kind of abstraction of services provided by the entities and entity instances within levels. The uppermost level is the value-added services level (VASL). In that level the entities are algorithms that provide end-to-end added value to the services of lower levels.

The level below the VASL is called network generic services level (NGSL). The scope of this level contains algorithms that take care of the functioning of the network, for example configuration and routing algorithms. The virtual network device level (VNDL) contains entities that are virtual representations of physical level entities. This means that these representation of entities provide a software interface to the physical elements that reside in the lowest, physical elements level (PE).

The interfaces between these layers provide a descending abstraction level. On top, the communication between the user and the VASL level provides convenient features only, hiding the necessary communication. In the middle, there are generic functions for accessing the network without explicit knowledge of the management implementation. The lowest interface is a collection of protocols that makes it possible to communicate with physical entities.

Our network management architecture presented in Part III follows this reference model quite closely. The lowest layer elements (the Linux routers) are represented by a configuration daemon running on the router. This daemon would be a VNDL-layer element. A bandwidth broker can configure the network by using generic commands of the QoS management API (cf. Chapter 6). The uppermost layer consists of the user interface that offers an input menu (cf. Section 7.5) to the user to specify its service level.

3.4 Network Management using ScriptMIB

The SNMP Management Framework is presently composed of five major components:

- An overall architecture, described in [71].
- Mechanisms for describing and naming objects and events for the purpose of management (Structure of Management Information (SMI) [117, 118, 119]).
- Message protocols for transferring management information [36, 30, 19].
- Protocol operations for accessing management information [35].
- A set of fundamental applications described in [101] and the view-based access control mechanism described in [186]

Managed objects are accessed by a virtual information store, referred to as the Management Information Base or MIB. Objects in the MIB are defined using the mechanisms defined in the SMI.

In [103] a part of the Management Information Base (MIB) to be used with network management protocols in the Internet is redefined. In particular, a set of managed objects that allow the delegation of management scripts to distributed managers is described.

The ScriptMIB module defined in [103] can be used to delegate management functions to distributed managers. Management functions are defined as management scripts written in a so-called management scripting language. The MIB makes no assumptions about the language itself and even allows distribution of compiled native code, if an implementation is able to execute native code under the control of this MIB.

The ScriptMIB defines a standard interface for the delegation of management functions based on the Internet management framework. In particular, it provides the following capabilities:

1. Capabilities to transfer management scripts to a distributed manager.
2. Capabilities to initiate, suspend, resume and terminate management scripts.
3. Capabilities to transfer arguments for management scripts.
4. Capabilities to monitor and control running management scripts.

5. Capabilities to transfer the results produced by running management scripts.

It does not address any additional topics like the generation of notifications or how to address remote agents from a ScriptMIB implementation.

The ScriptMIB has been implemented and evaluated in [162]. This evaluation shows, that most operations can be executed in less than 100 ms. In a highly dynamic mobile environment with many reconfigurations this might generate a serious bottleneck.

3.5 Bandwidth Broker Architectures

Several bandwidth brokers [128, 165, 167, 176] have been designed and developed throughout the last years since having been introduced in [126]. This initial architecture ever since has been the basis for a large amount of further developments of management structures for DiffServ networks: most architectures are quite similar, composed of interfaces and databases with an almost identical functionality. The differences between the individual architectures are quite hidden in the details. In addition, since then several enhancements have been added to improve the performance of such a bandwidth broker architecture in different environments, such as large networks. We now discuss this basic architecture in detail as our architecture is based on the same principle as well. The following sections present more elaborate functionality added to this initial architecture to work around some of its shortcomings.

According to [167] a bandwidth broker is defined as some kind of “oracle” that receives a resource allocation request (RAR) from one of two sources: either a request from an element in the domain that the broker controls, or a request from a peer (adjacent) bandwidth broker. However, [167] does not present an inter-domain broker protocol. The broker itself consists of three communication interfaces (user/application, intra-domain and inter-domain), of a routing interface that allows the broker to access routing information, and of a data repository that contains SLS information, current reservations and allocations, service mapping information, etc. However, the brokers of [128, 165, 167, 176] omit very important functions, like dynamic SLS negotiation. Our architecture does not vary much from this architecture but we specify the individual components in more detail and add more functionality. Usually, the access interface to routing information is described just very vaguely, while in our architecture there is an exact specification of how the topology information is retrieved and stored. Additionally, in contrast to our approach most architectures do not mention how the information in the data repository is structured or used.

We have also added a very flexible object oriented interface between the management and the configuration layer of the bandwidth broker. This is a completely new approach and has proven to be very successful in supporting various kinds of manageable elements (i.e. router hardware from different vendors but also entire subnetworks). W further add explicit support for mobile users, which cannot be found in any other architecture or implementation. Since performance is a main reason for user satisfaction, we have implemented our own signalling protocol for SLS negotiation. Our performance evaluation shows, that we can offer a very high speed of flow configuration, a feature that has been completely ignored in other implementations.

Our bandwidth broker further includes functionality also implemented in other existing bandwidth broker architectures. This is discussed in the following sections.

3.5.1 Overprovisioning

One critical topic of a centralised approach is the amount of signalling messages a central bandwidth broker has to process. [46] proposes a simple approach to decrease the amount of reconfiguration and signalling: after a reservation request from a user the bandwidth broker usually allocates more than the requested amount of bandwidth, hoping that subsequent requests can be admitted immediately with no reconfiguration necessary. The authors call this approach *path-oriented quota-based* (PoQ) bandwidth allocation. Yet simulation results show that the call blocking rate of the approach is comparable to the centralised-only scheme, where each flow is configured individually.

The idea of over-provisioning to reduce the configuration overhead is also presented in [134]. The authors propose a novel over-provisioning algorithm that depends on the amount of available resources. Unfortunately neither simulation nor experimental results about the performance of the algorithm are shown. The problem of dividing the domain into subnetworks that are under the control of a single bandwidth broker is not mentioned either.

In our architecture we provide a possibility to allocate more than the requested amount of bandwidth by setting two overprovisioning parameters. This allows a network administrator to adjust how much (in percentage *and* absolute values) bandwidth may be allocated, although not explicitly reserved. None of the architectures mentioned above provide this configurability, the algorithm is fixedly implemented in the broker's code without a possibility to adapt the overprovisioning to the actual situation in the network.

3.5.2 Hierarchical Bandwidth Broker Structures

A hierarchical structure for bandwidth brokers is presented in [134]. The basic architecture is similar to the one presented in the beginning of this section, and also to our own architecture: a bandwidth broker (in this context called Resource Control Point (RCP)) controls a set of Resource Control Agents (RCA), that in turn are responsible for controlling and configuring routers. Communication between the single entities and the hosts is performed via a special interface, called Application Middleware (AMW).

The hierarchy is used to reduce the interactions between the RCAs and the RCP. Each RCP is responsible for its "children" RCPs. Initially, the available bandwidth is distributed to the children according to a pre-defined configuration given by the network administrator. Afterwards, the children RCPs manage this amount on their own responsibility. If one child runs out of bandwidth it can request more from its parent RCP.

Another hierarchical bandwidth broker architecture is presented in [193]. This architecture is built on a basic central bandwidth broker architecture [46, 192, 191]. This broker consists of several modules, such as admission control, QoS routing and policy control. The broker maintains a number of management information bases (MIB) for the purpose of QoS control and management of the network domain (e.g. topology information base, policy information base, flow information base). Two additional MIBs are used to maintain the QoS states of the network: the *Path QoS state information base* and the *Node QoS information base*. The former contains parameters (i.e. hops, schedulers, propagation delay as well as dynamic QoS state information about different service classes) about paths between various ingress and egress routers of the network. These paths can be either preconfigured or dynamically set up. The latter maintains information regarding the routers in the network domain. Associated with each router is a set of static parameters characterising the router and a set of dynamic parameters representing the router's current QoS state.

When a new flow arrives at an edge router, requesting a certain amount of bandwidth to be reserved to satisfy its QoS requirement, the flow reservation set-up request is forwarded by the edge router to the bandwidth broker. The bandwidth broker then applies an admissibility test to determine whether the new flow can be admitted or not. More in detail, the bandwidth broker examines the path QoS state (obtained from the corresponding link states) and determines if there is sufficient bandwidth available along the path to accommodate the new flow. If the flow can be admitted, the bandwidth broker updates the path QoS state database and link QoS state database (as well as the flow information database) to reflect the new bandwidth reservation along the path. If the admissibility test fails, the

new flow reservation set-up request will be rejected, and none of the QoS information databases will be updated. In either case, the bandwidth broker will signal the ingress edge router its decision. For a flow reservation tear-down request, the bandwidth broker will simply update the corresponding link state database and path state database (as well as the flow information database) to reflect the departure of the flow.

The hierarchically distributed multiple bandwidth broker architecture consists of a central bandwidth broker (cBB) and a number of edge bandwidth brokers (eBB). The central bandwidth broker maintains the link QoS state database and manages quota allocation and de-allocation among the edge bandwidth brokers. Each of the edge bandwidth brokers manages a mutually exclusive subset of the path QoS states and performs admission control for the corresponding paths. If a flow arrives at an edge router, the flow reservation set-up request is forwarded by the edge router to the eBB that is in charge of the flow's path. The eBB will make admission control based on the path state it maintains such as the currently available bandwidth allocated to the path. If no sufficient bandwidth is available on the path, the eBB requests a new quota for the path from the cBB. If the request is granted, the eBB admits the flow and updates its path QoS state. When a quota request fails, the eBB will simply reject the flow reservation request instead of passing it to the cBB.

The topic of separating a network into parts and assigning a bandwidth broker to each part is also addressed in [134]: First, the RCP should represent a set of physical links that are topologically related. They can represent the links of a sub-area or sub-network, for example the network of a university laboratory. If two or more sub-areas are connected to the same router, a new RCP could be formed including the two RCPs that represented those sub-areas. Obviously, in a network that uses a fully meshed (or nearly fully meshed) topology, the concept of hierarchy can not be applied. Furthermore, it is not allowed for a RCP to include a link that is already member of another RCP of the same level: The level of a RCP should be taken into account because the parent RCP will always include links that are already members of its children RCPs. Also, the routing information could be an additional input to discuss. The sub-areas of the same level of hierarchy should not be directly linked. The local traffic for each sub-area should not use links that are members of another RCP, otherwise this will result in a leak of resources.

The hierarchy of [134] fails to clarify the way of how the bandwidth is allocated and distributed on a per-link level. The authors present an overprovisioning scheme that is to be applied if a child RCP runs out of bandwidth. Whether this new bandwidth is available at the congested link only or for the whole subnetwork is not specified. In our approach (cf. Section 7.8) we solve this problem by inverting the responsibility for the network: each "child" broker has the full knowledge

and control of all bandwidth and all links of its subnetwork. The "parent" broker allocates bandwidth from its children which it can allocate to inter-domain flows. This approach simplifies the network complexity for the "parent" broker a lot, a topic that is also not mentioned in [134]. The approach of [193] suffers from a missing investigation of distribution of paths to the edge brokers. In a large network especially, an edge broker should take advantage of spatial locality in order to decrease configuration delay. Furthermore, it remains vague how the bandwidth of a bottleneck link which is part of several paths through the network is distributed. The topic of topology changes, that would seriously affect the integrity of the path database is also not mentioned.

3.5.3 Management of Heterogeneous Networks

The topic of configuring heterogeneous networks is addressed in [128]: The standard of Differentiated Services specifies externally observable forwarding path behaviour (PHBs, cf. Section 2.2.3) only. Each vendor of a DiffServ-capable router may implement its own mechanism to support the PHB (e.g. either a priority round robin or a weighted fair queueing scheduler to implement Expedited Forwarding). The authors of [128] propose an architecture similar the way Java Applets are executed on heterogeneous clients running different operating systems: an OS-independent code is running on a *virtual machine*. The authors introduce a *Virtual Configuration Manager*, a program that runs at each edge router and receives a *Virtual Configuration Description*, generated in the bandwidth broker and translated into the router-specific configuration of the forwarding path. The bandwidth broker is not aware of different implementations on routers.

This approach is the contrary of our architecture which we will present in Part III: we try to keep the program running on the routers as simple as possible. For this reason we transfer the differentiation between the various router hardware to the bandwidth broker. Introducing an API for QoS management (see Chapter 6) we can translate generic router configurations to hardware-dependent commands, which just have to be transmitted and executed. With this approach we want to achieve a higher performance of the bandwidth broker and a smaller configuration latency: The translation from generic to specific configuration commands will be implemented in our architecture in a high-performance compiled language (C++) whereas the translation in [128] is interpreted at the virtual machine running at the router. Such an interpreter usually executes much slower than a compiled program. Unfortunately the authors of [128] did not publish any performance measurements.

3.5.4 Agent-Based Management

A different approach of management is proposed in [90]: again, the main focus is on the scalability problem of a centralised management architecture and the missing support of new technologies and protocols in the SNMP management protocol. Unlike the previously discussed architectures, this one approaches the scalability problem by using intelligent agents that are able to perform management functions by carrying code and executing tasks on any network node. This architecture also addresses the heterogeneity problem of configuring a network in a device-independent language.

This approach is based on prior work concerning VPN management in DiffServ networks [89, 88, 91, 24]. The use of VPNs essentially simplifies the view of the topology the broker has. However, no performance studies have been published to show how many flow requests can be performed by the bandwidth broker. In [90] this topic is explicitly mentioned as missing.

3.5.5 Conclusion

Most bandwidth broker architectures focus on the main problem of a centralised bandwidth broker approach: the large amount of reconfigurations that are necessary for a bandwidth broker to perform and the large number of signalling messages that follow in result. Two solutions have been proposed to this problem: over-provisioning and hierarchy. However, both solutions suffer from drawbacks: the former wastes bandwidth by allocating more than the requested amount, the latter introduces more brokers and more complexity to the scenario. In the publications cited here, the presented simulation results unfortunately are not extensive enough to thoroughly evaluate the disadvantages of those proposals. Our bandwidth broker architecture will also be centralistic, yet we can show, that the scalability problem is not as severe as it is assumed to be. To minimise the complexity and signalling overhead of a hierarchical architecture, we will propose a novel hierarchy structure where the root brokers have a very limited scope of duties only (cf. Section 7.8).

Although a distributed approach using intelligent agents for network management can solve the memory usage problem of a centralistic architecture, serious doubts concerning consistency and setup speed remain. On the other hand, the intelligence in [90] can be separated into two domains: service provisioning and handling heterogeneity. Those problems can, however, equally be solved in the centralistic, object-oriented architecture we propose: The heterogeneity is mapped to an object tree (see Chapter 6) and an intelligent algorithm for service provisioning is implemented into the bandwidth broker. The advantage of the centralistic

approach is the performance gain of a compiled language (like C++) compared to an interpreted script-like language. Therefore the overall flow setup speed of our central bandwidth broker is still very high (cf. Table 7.4), while no performance evaluation of the agent-based management architecture has yet been published.

3.6 Bandwidth Broker Implementations

An investigation in the Internet shows, that most publications just reference a very limited number of bandwidth broker implementations. This section will focus on the differences of the four bandwidth broker implementations most referenced [77, 169, 92, 135], and discuss some of their characteristics and drawbacks.

3.6.1 BB Implementation of the University of Kansas

The bandwidth broker implementation of the University of Kansas follows the basic architecture presented in Section 3.5. It consists of a broker database, a router configuration client, a SLA client and a bandwidth allocation request client. The broker database provides configuration and DiffServ codepoint for each SLA.

At startup the broker will load a default configuration to the network and then wait for incoming flow reservation requests. The broker implementation supports some heterogeneity by supporting two types of routers: a Linux-based DiffServ router and a commercial Cisco router. Linux routers are contacted directly via a TCP socket while Cisco routers are configured via an automated telnet session.

For flow specification and handling the broker provides a data structure containing all information needed (i.e. source, destination, protocol, service level information). This information is then sent to the broker via a host daemon. The broker offers a small set of commands (sla_add, sla_update, sla_delete, sla_list) that can be used for flow management. The host daemon receives the reply from the bandwidth broker and manages all flows to set up from this host.

Unfortunately, the tests presented only consist of a very small configuration (one router, two sources and a sink). Therefore, those tests are rather a functionality test showing the successful setup of a DiffServ configuration at the router than a kind of presentation of performance of the broker. Especially the question of how many flows can be set up per second and how many routers can be managed simultaneously remains unanswered.

3.6.2 Implementation of the Two-Tier Architecture by UCLA

Following the two-tier architecture, the bandwidth broker implementation of the UCLA [169] focuses on inter-domain brokering and bilateral negotiations between neighbouring domains. The bandwidth management within a single domain is not studied and covered by the implementation. The broker itself implements a twofold architecture: a flow database and a COPS server for communication with a COPS client at the routers to set up configuration parameters. Flows can be added and deleted to and from the flow database via a web interface.

On the router side, a COPS client exchanges flow information with the bandwidth broker and a second entity labelled Forwarding Path Driver (FPD) is used to configure the router. Unfortunately, detailed information on the FPD is missing. Also this paper does not address the problem of heterogeneity. Although experiments that show the ability of the implementation to set up DiffServ reservations are included, results about the speed of the flow setup procedure are missing. Since the core routers are not managed at all, the maximum topology size should not be a constraint, but this is not mentioned either.

3.6.3 Implementation of Policy Based Networks by TUT

The implementation of the Tampere University of Technology (TUT) [92] is quite similar to the implementation of UCLA described in Section 3.6.2. COPS is used for communication between the bandwidth broker (here called Policy Decision Point (PDP)) and the routers (Policy Enforcement Point (PEP)). A policy database is accessed via LDAP. Unfortunately the contents of the policy database are hardly mentioned. Up to now, no performance test of this implementation is available in the web.

3.6.4 Implementation of a vendor-independent BB

The implementation of Pop et al. [135] is the only one that directly addresses the topic of heterogeneity support by offering a vendor-independent interface for router configuration. Unfortunately the implementation of the independence is only examined theoretically, mentioning that SNMP or COPS might be used to configure the routers. This paper is also the only one that presents an implementation of the difficult problem of backward resource reservation, i.e. a host requests a flow whose source is not in the same management domain than the host itself. The authors propose to introduce special ICMP messages that can be intercepted

by an edge router to trigger the reservation. In Section 7.7.1 we will present a more detailed analysis of this problem.

3.6.5 Conclusion

Although there are many publications about bandwidth brokers in Differentiated Services networks, most of them reference a very limited number of implementations only. In addition, those implementations are often not thoroughly tested, especially regarding the number of routers an implementation is able to manage simultaneously. The performance of each implementation in terms of successful flow configurations per second has not been checked, too.

3.7 Authentication, Authorisation, Accounting

Authentication, Authorisation, and Accounting (AAA) can be defined as follows:

- Authentication is the process of identifying a user. Typically, a user proves its identity to the system by entering a valid user name and a valid password.
- Authorisation is the process of identifying what a user can do. For example, after logging in to a system, a user may try to issue commands. Authorisation determines whether the user is permitted to issue those commands. In some systems, authorisation and authentication are merged into a single process.
- Accounting is the process of measuring the resources a user has consumed. Typically, accounting measures the amount of system time a user has used, or the amount of data a user has sent and received.

By providing authentication, authorisation and accounting functions at the entry point of the network, one can control who can connect to the network and what they are allowed to do.

Even if we will have much more bandwidth in the future, the control of network resource utilisation remains essential for the support of applications with special demands and for the prevention of (malicious or accidental) waste of bandwidth. Charging provides a possibility to control utilisation and sharing of network resources. One challenge for the configuration of accounting services are heterogeneous metering and accounting infrastructures within provider domains. Also, the usage of different accounting and metering solutions used in different provider

networks complicates the sharing of configuration parameters (e.g. in mobile scenarios). To support mobile users in the Internet, adaptive network architectures and management of systems depending on monitoring the activity in this system are required. While customised user services, dynamic user behaviour, and user as well as device mobility increase, the importance of access control, authorisation, and security considerations arises significantly. Especially for dial-up or PPP (Point-to-point Protocol) connections Authentication, Authorisation and Accounting solutions exist in form of protocols and implementations, which integrate these AAA tasks. These tasks are commonly referred to as AAA systems. Presently, extensions to theses systems for other access scenarios like roaming or mobile users and access control extensions to communication protocols like Mobile IP are under discussion in the Internet Engineering Task Force (IETF) and the Internet Research Task Force (IRTF). Besides these protocol and data type parts, policies can be used as a mean for describing management goals and for the general management of networks.

In the following two sections we present some details of the AAA architecture with a special focus on mobile scenarios. Afterwards some additional research work on AAA in mobile scenarios and some policy-based approaches are discussed. A self-developed extension of the AAA architecture for use in mobile scenarios is presented in Chapter 10

3.7.1 AAA Architecture

Since an AAA framework is especially important for mobile scenarios, we will start our discussion not with the basic AAA model but present a more advanced model for distributed services [42]. An AAA infrastructure typically consists of AAA servers that interact with each other using an AAA protocol. The AAA servers authenticate users, handle authorisation requests and collect accounting data. Figure 3.2 shows the distributed AAA model.

A user wants access to a service or to a resource at the foreign domain. A foreign ISP's AAA server (AAAF), which authorises a service based on an agreement with the user home organisation, may not have enough information stored locally to verify the credentials of the user. However, the AAAF is expected to be configured with enough information to verify the client identity in collaboration with external authorities (e.g. a AAA server of the home network). This procedure can determine the nature of the service granted to the user.

A home domain's AAA server (AAAH) has an agreement with the user and checks whether the user is allowed to obtain the requested service or resource. This server might have information required to authorise the user, which might

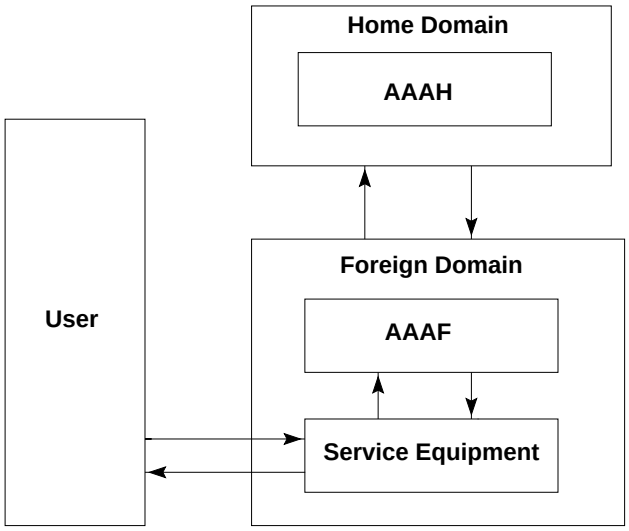

Figure 3.2: Distributed AAA Model

not be known to the foreign ISP.

An attendant, such as a foreign agent, which is an interface for the mobile node to the AAA server, often does not have direct access to the data needed to complete the transaction. Instead, it is expected to consult an authority (typically in the same foreign domain, e.g. AAAF) in order to request proof that the client has acceptable credentials.

3.7.2 System Components of the AAA Server

An AAA server needs several components in order to be able to handle AAA requests and supply QoS in mobile environments. With their implementation, the AAA server can inspect the contents of the request, determine what authorisation is requested and choose one of the following options to further process QoS requests:

- Query and retrieve policy rules from its SLA repository.
- Forward the policy component to another AAA server for evaluation.
- Let the policy be evaluated by the resource manager.

In DiffServ environments, customers are allowed to negotiate policies which define a fixed rate or a relative share of packets that have to be transmitted by the ISP

with high priority. All these policies can be put into a SLA repository which may reside on one AAA server or may be located elsewhere in the home network. Each policy should contain the following items: *user identification, password, service type, QoS parameters (rate, maximum burst, etc), source IP address, destination IP address, source port, destination port, duration of the request.* For evaluation and enforcement, each policy can also be retrieved by user name, by password, or by other attributes.

QoS support heavily depends on the allocation of a quantifiable amount of resources between a selected destination and source. However, network provisioning becomes very difficult and complicated in highly dynamic environments where the location and the QoS requirements of the end systems may change very quickly. So, a foreign ISP's AAA server needs to have an interface with the bandwidth broker (BB, Resource Manager Component) to check whether the user requirement can be satisfied or supported. As a part of the DiffServ architecture, the bandwidth broker is a software agent that automates the SLA negotiation and takes responsibility to allocate resources to users as requested.

Authorisation may be considered as the result of evaluating a SLA. While the policy definition typically is stored in the home domain of a visiting mobile node, it usually depends on the availability of the resource allocation in the foreign network whether its requirements can be satisfied or not. Due to the multiple administrative domain nature, a mechanism to forward messages between AAA servers is needed. Generally, any of the AAA servers involved in an authorisation transaction can retrieve or evaluate a policy (SLA) through an AAA protocol. This protocol is expected to be able to transport both SLA definitions and the information needed to evaluate SLAs and it also has to support queries for policy information.

A Network Access Server (NAS) is an interface for the mobile node to the AAA server, which allows access to network services to be managed on a per-user basis. The NAS may consult the AAAF in order to request proof that the client has acceptable credentials, to learn QoS and other network policies for the user via the AAA service and to apply QoS policies to the packets. A NAS may be an edge router which provides different qualities, types, or service levels to different users based on policy and identity information [121]. A NAS is like a QoS Policy Enforcement Point (PEP). Typically just the NAS knows the true dynamic session state. Therefore, the service equipment must be able to notify its resource manager as soon as a session terminates or the state changes in some other way. For auditing purposes, the generic server must have some form of database to store time-stamped events that occur in the AAA server. This database can be used to account for given authorisations. With the help of certificates, this database could support non-repudiation.

3.7.3 AAA for the Mobile Internet

In [72, 73] the authors discuss the need to enhance the existing AAA architecture in order to be able to support mobile equipment. Two different aspects have to be considered: security and economy. Based on an initial authentication and authorisation process, the mobile devices have to be allowed to consume distinct resources in the visited domain, for example to generate Internet traffic better than best-effort service. In the visited domain, the service definition will probably differ from Service Level Agreements (SLA) valid in the home domain. The need of this kind of service from a local domain requires authorisation of the mobile user, which directly leads to authentication. In many cases, the ISP of a visited domain only offers this service to a mobile user, if it is assured to get paid for the service. This requires an adequate accounting and charging concept, considering the quality of the service provided in the visited domain as well as other service-relevant characteristics. A client requiring resources of a visited domain is requested to provide credentials, which can be authenticated before access to these resources is permitted.

The authors propose an AAA architecture that has been derived from the generic architecture proposed by the AAAArch group [42]. It consists of AAA Systems that can either be an AAA server or an AAA client. The protocol to be operated between the AAA server and the AAA client is called AAA protocol, however, for a sufficient transfer of appropriate data it may be an enhanced version of either RADIUS [143, 142] or DIAMETER [84, 28]. An AAA client has no services to offer, instead it can request services only. Therefore it has to use the agent authorisation model [179]. An AAA server operates an interface to several Application-specific Modules (ASM), which provide either a service (e.g. Interface to Mobile IP, QoS, content service) or an accounting or charging functionality, which is considered as an important service on its own. The AAA server also has an interface to external authentication modules to be able to use different authentication techniques. The accounting component may get usage data input from a underlying metering component. Accounting can be a separate service or integrated within the service provided [195]. In case of integrated accounting, the service and accounting is performed by a single component, which interfaces the AAA server via one ASM. Charging is implemented as an external module which also communicates via an ASM with the AAA server. The charging module generates input for a subsequent billing process.

Our own AAA architecture extension, which we present in Chapter 10, focuses on the heterogeneity of services a mobile user might encounter when visiting a foreign network. Therefore, we propose not to use the AAA server of the foreign network as the first contact entity but to use the Service Location Protocol (SLP)

instead. This provides a generic framework to include new services: for example, after authenticating itself, the mobile user would like to negotiate a new QoS level at the foreign network. The address of a bandwidth broker can be found in our scenario much easier than in the scenario discussed above.

3.7.4 Policy-based Approaches

To offer services to customers, service providers have to manage distributed systems. This includes the configuration of networking devices (hardware) and the provision of various protocol mechanisms (software). Policies define one possible approach to constrain communication in networks and to manage networks. In [138] the traditional AAA, services are extended to contain auditing, charging, billing and pricing services as well. This new collection of services is called A^{x}. Such services are required by providers to offer transport and information services in a commercial environment. Policy-based A^{x} services can service descriptions in form of policies from mechanisms and system-specific information. Furthermore, policies enable the construction of inter-domain A^{x} services applicable to mobile scenarios. In particular, [195] proposes to use accounting policies to configure the accounting infrastructure and to use the Authentication, Authorisation and Accounting architecture to exchange and to deploy these policies.

Several drawbacks of the traditional AAA architecture are criticised in [138]:

- Missing separation of policy decision and policy enforcement
- Complicated extensibility
- No inclusion of QoS-related support services

The architecture proposed in [138] combines the traditional Policy-based Management and AAA architectures in order to form a generic A^{x} service architecture to solve the problems mentioned above.

Summary

This part presented the three basic principles this thesis is built on to achieve ubiquitous Quality of Service for mobile users: Mobile IP, Differentiated Services and bandwidth brokers.

Mobile IP is a well-developed and thoroughly investigated approach to provide mobility in the Internet. Several implementations are already available, including open-source projects for Linux [6, 194]. Handovers have soon been recognised as crucial for Mobile IP performance. Many proposals to improve handovers have been made. Unfortunately, no implementation supports those recent proposals. Another problem is the incomplete inclusion of lower-layer information. With this information the actions needed to prevent some of the drawbacks of a handover can be made in time. In Chapter 9 we present an implementation of a handover, that retrieves information from the MAC layer to be able to set up reservations in advance.

Differentiated Services is a concept to support Quality of Service in IP networks and to avoid the scalability problems of RSVP. Differentiated Services group packets with similar QoS requirements to a traffic aggregate in a way that they are treated equally at the routers. The algorithms for packet treatment are not complicated and can be applied even at high packet rates as will be shown in Chapter 5.

Differentiated Services suffer from a big drawback: There exists neither a management framework nor a signalling protocol as it is the case for RSVP. Therefore dynamic end-to-end reservations cannot be established. In Part III we present a bandwidth broker architecture and a signalling protocol that enables us to specify a detailed description of our QoS requirements in a way that the broker can configure the network accordingly.

Part II

Implementation and Performance Evaluation of a DiffServ Implementation for Linux Routers

Overview

Differentiated Services support Quality of Service in the Internet and are — unlike Integrated Services — based on flow aggregation. This avoids the need for multi-field classifiers at each hop, but network resources are reserved for any kind of traffic aggregation. This makes DiffServ a scalable technology, because only a limited set of aggregations needs to be supported in core routers. This part presents our implementation of Differentiated Services on Linux Routers and evaluates it by means of extensive performance measurements. These measurements are taken on an end-to-end basis for single traffic aggregates. We do not study the effect of aggregating various sources into one traffic aggregate because the number of sources has to be very large in order to get statistically meaningful results. For the study of traffic aggregation, simulation approaches seem to be more suited.

Our implementation of DiffServ routers on Linux PCs is described in Chapter 4. Section 5.1.1 presents the test network built in our computer networks laboratory. In Section 5.1.2 the measurement methods we used for detailed delay and performance evaluation are discussed. The results obtained regarding bandwidth, delay and jitter of 5 MBit/s UDP and TCP traffic using expedited and assured forwarding are shown and compared in Section 5.2.

Chapter 4

DiffServ Implementation

Now we will discuss some performance aspects of the Differentiated Services architecture. This architecture, as presented in Section 2.2, promises a scalable way of deploying Quality of Service for end users. However, the proof that DiffServ routers really are able to provide such bandwidth guarantee still has to be given. We have implemented Differentiated Services for Linux Routers and tested this implementation on a test network consisting of six nodes. This may not seem much but it is, however more detailed than most other experimental results (see [9, 17]). In addition, a survey of existing ISP backbones showed, that such a small hop number still is a reasonable approximation to reality (cf. Section 5.1.1).

Differentiated Services have been implemented for the Linux operating system shortly after the first publication of the concept by ourselves [26] as well as several other researchers [5, 17]. The implementation architecture described in [5] is similar to our implementation, because both are based on Linux traffic control [4, 136]. The implementations differ in details of some traffic conditioner implementations as well as in the way the implementations are configured: While [5] is based on traffic control command only, our implementation can be configured via tables that contain bandwidth, as well as other QoS-related parameters, such as queue size and scheduler parameters. By changing the parameters in the tables, the parameters of the traffic conditioners are changed during runtime. This accelerates the DiffServ configuration, simplifies the setup and helps to avoid inconsistent configurations.

The KIDS implementation [17] does not rely on Linux traffic control, but own queueing disciplines have been implemented. The performance measurements of KIDS have been performed with a network consisting of one single DiffServ router and focused on measurements of the internal implementation behaviour. No end-to-end measurements over several routers, in particular delay measurements have been published.

4.1 Traffic Conditioner Modules for the Linux Kernel

The implementation of Differentiated Services on a Linux router [26] provides a full set of traffic conditioning modules enabling a user to set up any kind of DiffServ router (i.e. boundary and core routers). Those modules include a marker, a classifier / scheduler, service handlers for EF and AF and several queueing disciplines such as token bucket filters, FIFO and TRIO queues. All traffic conditioners have been implemented as kernel modules that can be activated by the `tc` command, which is part of the `iproute` package [99]. This command can set the parameters of the queueing disciplines (bandwidth, buffer space ...) and can combine the traffic conditioners to form the configuration of a particular DiffServ router interface (boundary or interior router, see also Figures 4.1 and 4.2).

The information, which flows will get a certain service, is stored in a table within the router's kernel-memory. A kernel module together with a user interface program has been developed to support creating and changing those tables during runtime. The tables contain the source and destination subnet addresses of each privileged flow, marking information (the DSCP) and metering limits (bandwidth values) for the service handlers. IPv4 as well as IPv6 addresses are supported.

The **service_handler** is the marking module of our implementation. This module incorporates a multi-field classifier. It compares all incoming packets to the flows held in its table and writes the according DSCP into the IP header. Since this module has no metering functionality the dropping probabilities of AF packets are set by the `prec_handler` module (see below).

The **dsclsfr** module is a combination of a BA classifier and a scheduler. The classification procedure is executed when enqueueing a packet and forwards the packets according to their DSCPs to one of seven traffic conditioners. Those conditioners are intended to handle the four assured forwarding classes, expedited forwarding traffic, network control traffic and best effort traffic. The scheduling performed by the `dequeue` function is a combination of priority scheduling and weighted fair queueing. The highest priority is assigned to expedited forwarding traffic, the second highest priority to network control traffic and the third priority to the remaining five traffic classes. Those five classes — four for assured forwarding and one for best effort — are handled by the weighted fair queueing algorithm. The weights are configurable and can be specified via the command line.

The **prec_handler** is a colour - aware two-rate three colour marker [75]. The AF-PHB defines four independent service classes, each operating at three

levels of dropping probability. Traffic below the negotiated bandwidth limit has the lowest probability of getting dropped ("is marked green"). A packet is marked "yellow" (to a higher dropping probability), if it does not exceed a certain exceed-bandwidth. All other traffic is marked "red". The `prec_handler` specifies the colour-part of the AF-DSCP (see [74]), while preserving the colour of already marked incoming packets.

The **premium_shaper** is a conditioner for metering and classifying expedited forwarding traffic in ingress routers. According to [40] each expedited forwarding traffic must be shaped to the negotiated rate. This module offers the possibility of shaping a certain number of flows (currently up to 256) independently to different bandwidth values by offering multi-field classification based on IP addresses, port numbers and protocol ID. For each flow a separate token bucket filter that is configured to the maximum bandwidth / burst size parameters has to be installed.

The **TRIO** queue is a modification of the well-known RED queueing algorithm [60]. While the RED algorithm uses one dropping probability function, the TRIO queue uses three dropping functions, one for each colour of the three colour marker described above. By combining a properly configured TRIO queue with the `prec_handler` we can ensure increasing dropping probability for yellow and red packets while sustaining the order of the packets within the AF flow.

4.2 DiffServ Router Architecture

The combination of different traffic conditioners allows us to build several DiffServ router types forming the basis of our DiffServ implementation concept: An *edge router* is located at the borders of a DiffServ domain. Several subtypes can be distinguished according to the location of the router in the forwarding path:

- An ingress router as depicted in Figure 4.1 is located at the entry point of a traffic stream into the domain and performs most of the traffic conditioning functions. Our ingress router architecture consists of
  - A multi-field classifier, that is an integral part of the marker module. It decides which DSCP has to be written into the IP header of the packet by the marker. Marker and Classifier are the two parts of the `service_handler` module that serves as the *Marker* module in the ingress architecture. Since classification and marking of micro-flows

may be computationally expensive, this is only performed at ingress routers. Note, that the multi-field classification may also include previously set DSCPs, so that re-mapping of flows at the ingress is also possible.

  - A behaviour aggregate classifier (the `dsclsfr` module) forwarding the packets to the correct service handler, based on the DSCP only.
  - Several service handlers for the different PHBs (expedited and assured forwarding) (the `prec_handler` and `premium_shaper` modules.
  - A TRIO queue per AF class.
  - a token bucket filter plus a FIFO queue per EF flow registered at the ingress router.
  - FIFO queues for best effort and network configuration traffic.
  - A scheduler to ensure that each single service class has enough bandwidth available but does not exceed the negotiated limits. This scheduler is part of the `dsclsfr` module.

- An egress router is located at the exit point of a traffic stream. It has to ensure by traffic conditioning that the agreement about the amount of traffic leaving the DiffServ domain is met.

- An interior router (see Figure 4.2) can take full advantage of the traffic aggregation. They simply handle the traffic according to the DSCP set in the IP header. Traffic conditioning is mainly performed to detect misconfiguration and to minimise the damage caused by that (as e.g. required in [40]) The architecture of a core router is much simpler compared to an ingress router:

  - a BA classifier forwards the packets to the service handlers
  - there are no AF handlers, only TRIO queues perform minimal traffic conditioning for AF traffic
  - EF traffic is directly forwarded to a special EF handler (`premium_policer`): any EF traffic exceeding the limit that results from the aggregation of different flows *must* be dropped (cf. [40]). This is equal to a token bucket filter with zero queue length.
  - FIFO queues for best effort and network configuration traffic
  - the scheduler

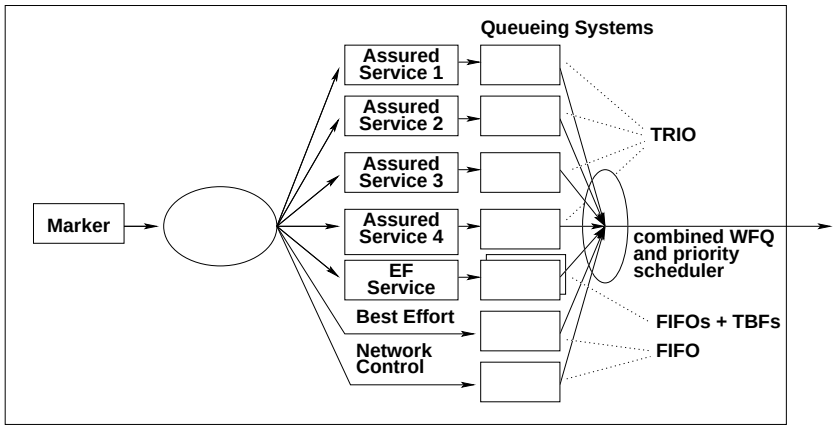

Figure 4.1: Ingress Router Implementation Architecture

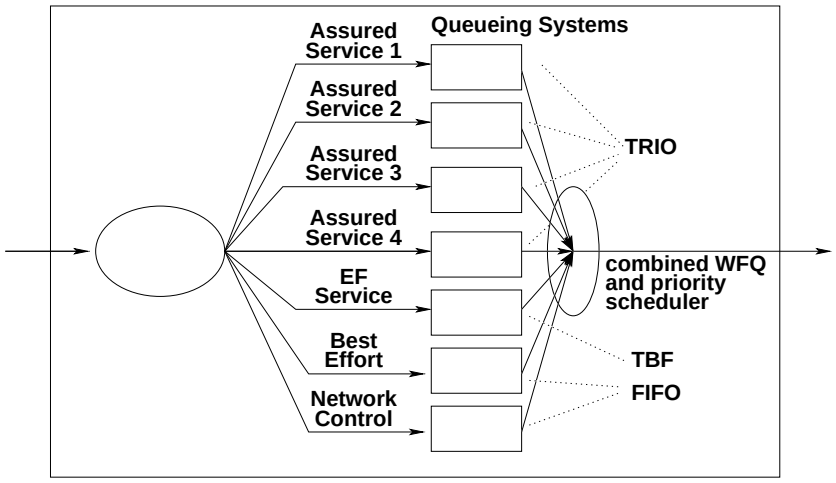

Figure 4.2: Interior Router Implementation Architecture

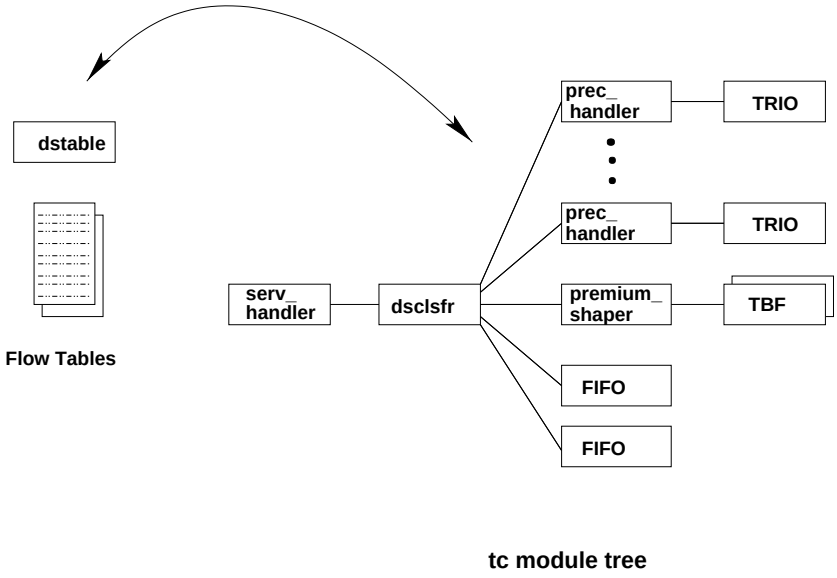

Figure 4.3: Ingress Router Configuration

4.3 Linux Router Configuration

4.3.1 Ingress Router

There is a major difference between the implementation architecture of the DiffServ routers shown in Figures 4.1 and 4.2 and the actual arrangement of traffic conditioners within a Linux router: While in Figures 4.1 and 4.2 all conditioners are passed only once by each packet, the arrangement of traffic conditioners by the `tc` command is a tree (see Figure 4.3), and therefore a packet has to pass each conditioner twice. Additionally the classifier has to perform the scheduling simultaneously (see the description of the `dsclsfr` module above). The different service classes are handled by appropriate service handlers (i.e. `prec_handler` or `premium_shaper`) or directly forwarded to queues (see Figure 4.3). To install a traffic conditioner at the correct location within the forwarding path the `tc` command needs three parameters: Each traffic conditioner has a unique ID (called `handle`), a `parent` (the `handle` of the parent conditioner in the tree), and a `class_id`, which indicates the position the conditioner occupies within multiple child conditioners. For example, if the `dsclsfr` in Figure 4.3 has `handle` 2, the `premium_shaper` would have `parent` 2 and `class_id` 5. Those three parameters, together with other, conditioner-specific parameters (e.g. queue length) are passed to the `tc` command.

Flow description information cannot be passed to the traffic conditioners via the command line. This information is stored in tables, allocated by the `dstable` module. This module can handle several table IDs and the conditioner modules can access the tables by using the correct ID. It is therefore possible to install a table with all EF flows and a second one with all AF flows and pass the table IDs to the correct service handlers. The communication between the tables and the traffic conditioning modules is done by a API provided by the `dstable` module.

In Table 4.1 we show, how the DiffServ tables for the `dstable` module are built. The first table (the `serv_handler.table`) contains all flows that have to be marked at this router with the corresponding codepoint. The second table (the `premium_shaper.table`) only contains the expedited forwarding flows, mapping each flow to its corresponding token bucket filter number. The bandwidth limit for the flows is not included in this table but in the configuration script (see Figure 4.4). The third table contains all assured forwarding flows and their bandwidth limitations for the dropping precedence handler. Figure 4.4 shows, how an ingress router has to be configured in order to support those reservations.

After we have investigated the ingress router in detail, we can handle the remaining router types quite shortly, since there are no architectural differences, only the number and the arrangement of the traffic conditioners differ. A detailed description of configuration with several examples of flow configuration tables and setup scripts can be found in [156].

4.3.2 Interior Router

The interior router configuration reflects the simple architecture of Figure 4.2. There is only a classifier (`dsclsfr`) and the queues. TRIO (`trio`) queues are responsible to handle AF traffic, while EF traffic is strictly policed to the configured limit by the `premium_policer` module.

4.3.3 Egress Router

Egress routers can be held as simple as interior routers if there is no special negotiation between two domains regarding the amount of AF traffic leaving a domain. In such a case, additional AF handlers (`prec_handler`) must be inserted in front of the TRIO queues in order to limit the amount of AF traffic.

Table 4.1: Router Configuration Tables

serv_handler.table

#	srcaddr	srcmask	srcport	dstaddr	dstmask	dstport	proto	srcdscp	outdscp	lprio	mprio
	10.1.1.1	255.255.255.255	5005	10.1.4.2	255.255.255.255	5005	*	*	ps	0	0
	10.1.1.1	255.255.255.255	5030	10.1.4.2	255.255.255.255	5030	*	*	ps	0	0
	10.1.1.1	255.255.255.255	5105	10.1.4.2	255.255.255.255	5105	*	*	as1	0	0
	10.1.1.1	255.255.255.255	5110	10.1.4.2	255.255.255.255	5110	*	*	as1	0	0

premium_shaper.table

#	srcaddr	srcmask	srcport	dstaddr	dstmask	dstport	proto	srcdscp	outdscp	class	unused
	10.1.1.1	255.255.255.255	5005	10.1.4.2	255.255.255.255	5005	*	ps	*	1	0
	10.1.1.1	255.255.255.255	5030	10.1.4.2	255.255.255.255	5030	*	ps	*	2	0

precedence_handler.table

#	srcaddr	srcmask	srcport	dstaddr	dstmask	dstport	proto	srcdscp	outdscp	lprio	mprio
	10.1.1.1	255.255.255.255	5105	10.1.4.2	255.255.255.255	5105	*	as1	*	625000	0
	10.1.1.1	255.255.255.255	5110	10.1.4.2	255.255.255.255	5110	*	as1	*	1250000	0

Figure 4.4: Ingress Router Configuration Script

```
#! /bin/bash
#
TC=/usr/bin/tc
TABLE0=~/diffserv/serv_handler.table
TABLE1=~/diffserv/precedence_handler.table
TABLE2=~/diffserv/premium_shaper.table
DEV0=eth1
#
dstab l 0 $TABLE0
dstab l 1 $TABLE1
dstab l 2 $TABLE2
#
$TC qdisc add dev $DEV0 root handle 1: serv_handler table_id 0
$TC qdisc add dev $DEV0 root handle 2: dsclsfr as1 0.4 as2 0.3 as3 0.1 as4 0.1 log_size 63
#
$TC qdisc add dev $DEV0 parent 2:1 handle 11: prec_handler table_id 1 dscp as1
$TC qdisc add dev $DEV0 parent 2:2 handle 12: prec_handler table_id 1 dscp as2
$TC qdisc add dev $DEV0 parent 2:3 handle 13: prec_handler table_id 1 dscp as3
$TC qdisc add dev $DEV0 parent 2:4 handle 14: prec_handler table_id 1 dscp as4
$TC qdisc add dev $DEV0 parent 2:5 handle 15: premium_shaper table_id 2 classes 2
#
$TC qdisc add dev $DEV0 parent 11:1 handle 101: trio limit 200 low_begin 0.8 low_end 1/
medium_begin 0.4 medium_end 0.6 high_begin 0 high_end 0.2
$TC qdisc add dev $DEV0 parent 12:1 handle 102: trio limit 200 low_begin 0.8 low_end 1/
medium_begin 0.4 medium_end 0.6 high_begin 0 high_end 0.2
$TC qdisc add dev $DEV0 parent 13:1 handle 103: trio limit 200 low_begin 0.8 low_end 1/
medium_begin 0.4 medium_end 0.6 high_begin 0 high_end 0.2
$TC qdisc add dev $DEV0 parent 14:1 handle 104: trio limit 200 low_begin 0.8 low_end 1/
medium_begin 0.4 medium_end 0.6 high_begin 0 high_end 0.2
$TC qdisc add dev $DEV0 parent 15:1 handle 105: tbf rate 5000kbit buffer 500kbit limit 500kbit
$TC qdisc add dev $DEV0 parent 15:2 handle 106: tbf rate 30000kbit buffer 3000kbit limit 3000kbit
```

Chapter 5

Performance Results

5.1 Test Network and Evaluation Methods

5.1.1 Test Network Design

The performance measurements aim to evaluate the end-to-end performance between a sender / receiver pair interconnected by a DiffServ network. Due to the limited resources available in a laboratory this interconnecting network is usually very small. To be able to compare our results in relation to existing networks we analysed the topologies of several European national research networks: SWITCH, the swiss education and research network [164], GEANT, the pan-European research network [65] and the European part of the Worldcom network [188]. We observed, that typically there are only very few backbone routers (mainly 3 – 6), between an ingress and an egress router. Therefore we set up a test network consisting of three routers: an ingress router, an interior router and an egress router (see Figure 5.1). The egress router has the same function as the interior router in such a scenario. Note that the sender-receiver relationships denote a sender and a receiver for a traffic aggregate but not for a micro-flow. The effects of several micro-flows competing at the edge for a single DiffServ service class is too complex to set up with a reasonable amount of hardware resources because a large number of senders would be needed for statistically meaningful results.

To simulate a highly congested router the network has been flooded by an aggressive UDP sender, transmitting 100 MBit/s traffic to the receiver on each of its three links. At each outgoing router interface the DiffServ traffic from the sender has therefore to be protected against a heavy background traffic load. Since the number of interior routers does not influence the service provisioning significantly we can estimate the behaviour of the DiffServ traffic classes (i.e. the provision of

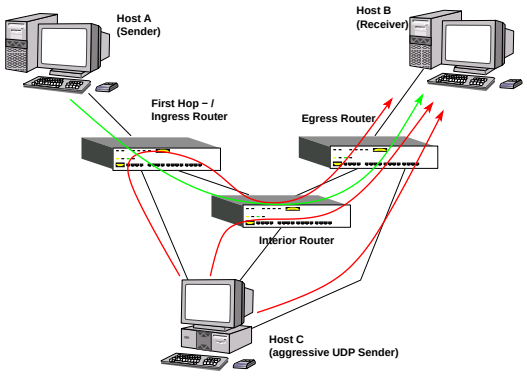

Figure 5.1: Testnetwork topology

bandwidth and certain delay and jitter limits) for large-scale backbones.

The nodes of our test network have been interconnected by full duplex 100BaseTx connections. Host A transmits UDP and TCP traffic to Host B. The first hop router marks the traffic with the EF and AF DSCP. The background traffic from Host C has three different routes to Host B. Therefore each DiffServ router has to drop 100 MBit/s (= 50%) of the incoming traffic at the outgoing interface (see Figure 5.1).

5.1.2 Performance Measurement Procedures

A couple of self-programmed tools have been used for the load generation. These allow to set various parameters, including ports, bandwidth and packet size (only for UDP traffic). The sender program transmits a packet of the specified size, then waits for a time t

$$t = \frac{\text{packet size}}{\text{bandwidth}}$$

and transmits the next packet. Waiting can be implemented in two ways: The first possibility is to get the actual time again and again, and send the packet as soon as the calculated time interval has passed (busy waiting). This consumes a lot of computing power, but gives an accuracy of about 10 μs. The second possibility is to wait for an operating system call to resume. The main drawback of this method is, that the accuracy of the Linux operating system is limited to 1 – 10 ms.

Both waiting algorithms have been implemented for the UDP sender, whereas only the second has been implemented for TCP. Since the TCP implementation queues the packets in order to form a traffic stream, higher timing accuracy would be lost in this case. The packet payload consists of an identification number to

calculate loss rates and two time-stamp fields, that have been used to calculate the end-to-end delay and jitter as shown in Figure 5.2.

Another problem with end-to-end measurements is the measurement of delay. In many other experiments, senders and receivers are identical machines, so that the same clock can be used to calculate the delay between transmitting and receiving a packet. Another alternative is to use sophisticated and expensive synchronisation hardware, for example based on GPS. In our case we developed a novel delay measurement scheme based on the estimation of the clock skew between two independent clocks (see Figure 5.2).

Before starting the tests, a two minute period of delay measurements without background traffic have been performed. The sender writes its local time t_0 to the first field, the receiver writes its local time T_0 to the second field and sends the packet back to the sender. Together with the arriving time of the packet at the sender t_1 and assuming that the transmission time is equal in both directions (we may assume that, since there is no background traffic and no congestion) we can easily calculate the delay

$$d = \frac{1}{2}(t_1 - t_0)$$

and the clockskew

$$c = T_0 - t_0 - d.$$

Figure 5.3 illustrates the drift between sender and receiver clocks before the first test. An obvious linear trend can be observed and assuming that this linear trend is constant during the measurements, we can compute the delay of each single packet. To estimate the trend, two different statistical methods were used: a standard least squares and a M-estimate method (minimising absolute deviation). The least squares method is obviously more sensitive to data that shows a large deviation from the linear trend because the error in this case is not normally distributed. Therefore, the results of the more robust method have been used in future calculations.

Now we can measure the delay and jitter during the tests (see packets 2 and 3 in Figure 5.2): estimating the clockskew

$$c(t) = a \cdot t + b$$

the delay of a packet is calculated by

$$d = T_1 - t_2 - c(t_2).$$

The jitter is calculated using two subsequent packets: according to [40] jitter is defined by

$$j = |T_2 - T_1 - (t_3 - t_2)|.$$

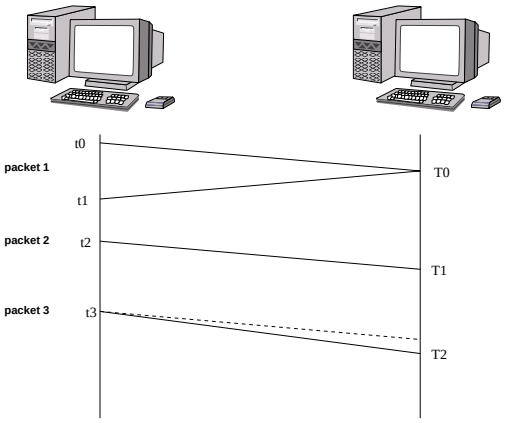

Figure 5.2: Delay and jitter measurements

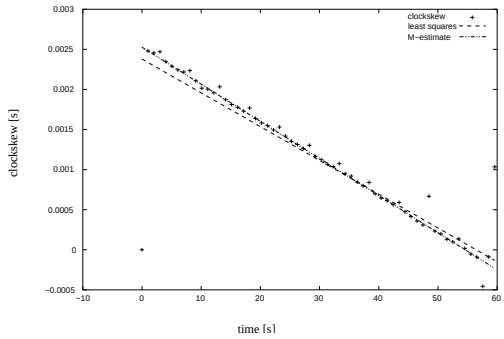

Figure 5.3: Clockskew Variation

This can be explained by Figure 5.2: packet 3 is a little bit slower (solid line) than packet 2 (dashed line). The difference of those two packets' speed is exactly the expression in the definition above. Taking the absolute value we ensure, that slower and faster packets do not average the jitter to zero.

The delay and jitter values are measured by the receiver during a configurable time interval and the average of that values is used in the graphs shown in section 5.2. The timescale used in the results section was 100 ms, which is a good compromise between the need of an accurate image of the behaviour of the DiffServ network and the limited timing accuracy of the Linux routers.

The overall duration of each test was 10 minutes to get a statistically significant amount of data, that allows us to predict the behaviour of a Linux DiffServ router accurately. As it can be seen from the results in Section 5.2, a duration of 10 min for each experiment seems to be more than sufficient to achieve this goal.

5.2 Results

5.2.1 Tests without DiffServ

In order to be able to compare our results to the results achieved with best effort forwarding, we performed some measurements without DiffServ. The throughput, the delay and the jitter have been measured for a 5 MBit/s UDP flow from Host A to Host B (see Figure 5.1).

During the first test there was no background traffic at all. Under these circumstances we can see, that the backbone of three routers creates a delay of just 1 ms

(see Figure 5.5). No packet loss can be recognised in Figure 5.4, and the jitter is at the lowest limit of the computer's time accuracy. For most of the time we get a jitter lower than 2 μs. The variance of the jitter can be explained by the randomness of the router's interrupts only.

The second initial test was designed to test unprotected UDP traffic from Host A to Host B against the 3×100 MBit/s background traffic generated from Host C. The flow's bandwidth was again 5 MBit/s.

In Figure 5.7 we can see, that the background traffic of Host C causes serious losses even to the connectionless UDP traffic. Approximately 1 MBit/s of the sender traffic can cross the network only. This is a loss of approximately 80%. Also, the variance of the throughput is considerably large.

The delay of the flow is approximately 28 ms (see Figure 5.8). Obviously, the traffic always meets full queues at each router due to the constant background traffic. Therefore, a packet has to wait until the maximum queue length has been forwarded. Since the default queue length of the FIFO queues in the routers is 100 packets we expect a delay of

$$d = 3 \cdot \frac{100 \cdot (1024 + 8 + 20 + 122) \cdot 8 \text{ Bit}}{100 \text{ MBit/s}} = 28.2 \text{ ms}$$

which shows the excellent accuracy of the delay measurement procedure. The values in the parentheses are the payload size (1024), the UDP header size (8), the IP header length (20) and the size of the Ethernet 802.3 header and tail (122) in bytes.

The jitter of this flow has a lower bound of 2 ms and a high variance (see Figure 5.9). Compared to the jitter values of the last series which has been lower by three orders of magnitude (Figure 5.6) we can see the big randomness of the packets being enqueued or dropped in the routers. This randomness results from the way, how the Linux-kernel internally handles incoming packets and forwards them to the outgoing queue.

5.2.2 UDP experiments

The UDP tests have been performed with a reservation of 5 MBit/s and four different runs, transmitting 2, 4, 5 and 10 MBit/s from Host A to Host B (see Figure 5.1) with UDP payload size of 200, 400, 500 and 1000 bytes. The results have been achieved by using the busy-waiting UDP sender (see section 5.1.2). Together with the varying UDP payload this ensures a constant bit rate sender measured over very short time intervals together with a constant packet frequency. This is especially important for explaining the results for assured forwarding later.

Results for UDP without DiffServ and background traffic

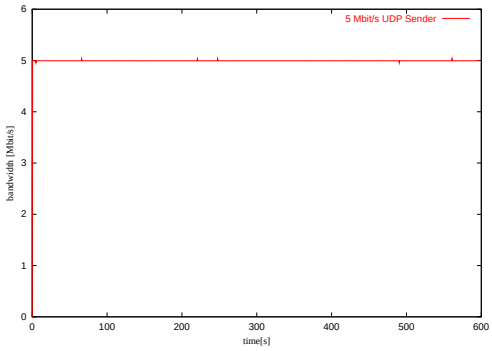

Figure 5.4: Bandwidth for 5 MBit/s UDP

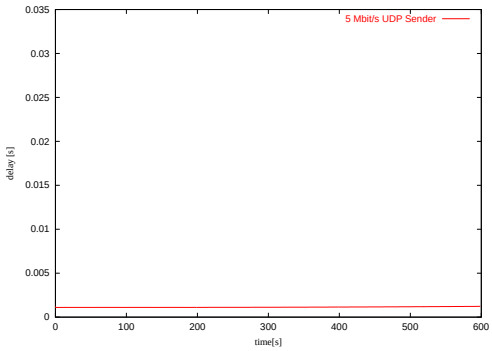

Figure 5.5: Delay for 5 MBit/s UDP

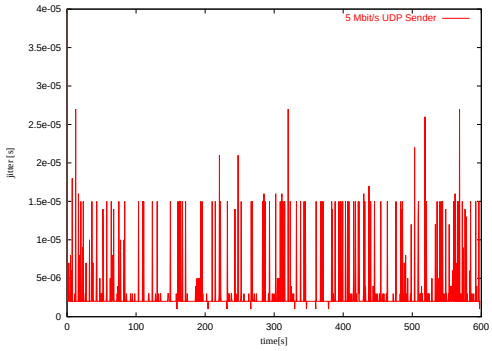

Figure 5.6: Jitter for 5 MBit/s UDP

Results for UDP without DiffServ but with background traffic

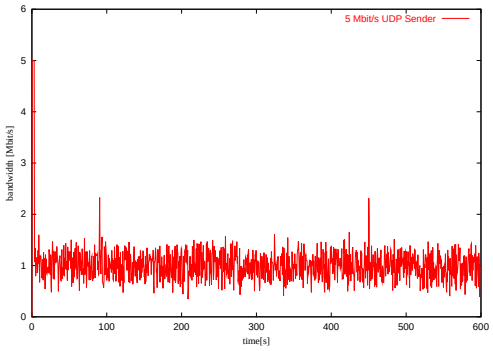

Figure 5.7: Bandwidth for 5 MBit/s UDP

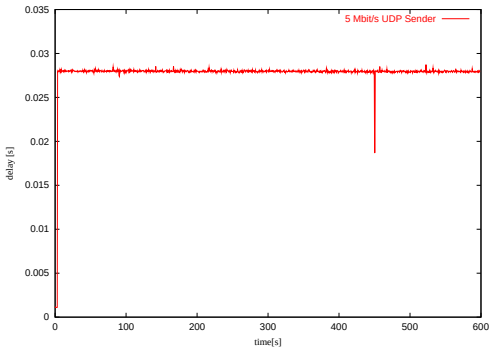

Figure 5.8: Delay for 5 MBit/s UDP

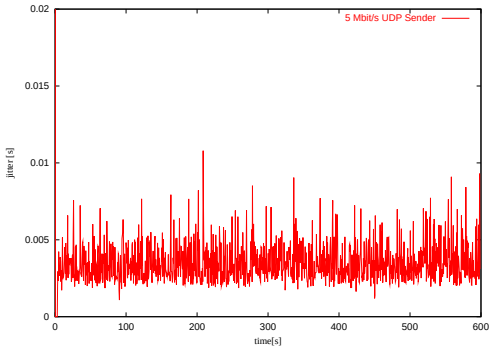

Figure 5.9: Jitter for 5 MBit/s UDP

Expedited Forwarding During the tests of the EF PHB all queues have been configured to a maximum throughput of 5 MBit/s and a maximum delay of 25 ms (this corresponds to 125000 bit queue length). This results in an increasing queue length for faster UDP senders, but it allows bursts to a certain degree and simplifies to compare the results.

Figure 5.10 shows the throughput for all transmission rates of 2, 4, 5 and 10 MBit/s. We can see, that in any case the bandwidth is provided up to the negotiated limit, regardless of the amount of EF traffic that flows through the router. This means, that the 2 and the 4 MBit/s sender suffers almost no loss, but the 5 MBit/s sender looses 7.66% of it's packets, while the 10 MBit/s sender looses even 52.1% of it's packets. The reason why the 5 MBit/s sender looses packets, although 5 MBit/s have been reserved, is, that the reservation is slightly lower than required, since the application generates 5 MBit/s data while the 5 MBit/s reservation is on IP level and includes packet headers.

The delay of the four flows is shown in Figure 5.11. As we can see, DiffServ provides a delay of less than 5 ms as long as the sender meets the bandwidth limitations. Most of this delay comes from the DiffServ processing inside a router. The queueing delay in this case is negligible, since the maximum fill-state of the queue was 2 packets. If the sender tries to transmit more traffic than allowed, the traffic shaper in the ingress router will create a delay according to the size of its token bucket filter. In the case of the third test (5 MBit/s Sender and 5 MBit/s reserved) the high delay is caused by the rather long queue size that is used to support busy traffic.

The jitter has a sharp lower bound of approximately 0.6 ms for the bandwidth values less than 10 MBit/s and a large variance which is decreasing when sending at a higher rate. For the 10 MBit/s sender the jitter increases to 0.8 ms but at a smaller variance. We assume, that the token bucket filter cannot dequeue packets in equidistant timesteps — especially at higher rates — and so causes a higher jitter.

Assured Forwarding The parameters for the four AF test runs have been chosen such that the results can easily be comprehended.This requires using a single AF class and two precedence levels, i.e. red and green. The `prec_handler` has been configured to mark all excess traffic as red. Dropping of red traffic starts at an empty queue and all packets are dropped, when the queue length has reached a fill level of 20%. Green traffic has been dropped between queue lengths of 80% to 100%. The queue length was 200 packets.

Figure 5.13 shows a similar behaviour for the flows below the limit but also clearly shows the difference between EF and AF for handling out-of-profile traffic. While

Results for UDP traffic with 5 MBit/s EF reservation

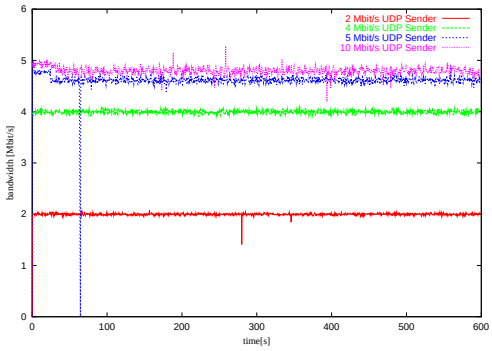

Figure 5.10: Bandwidth for 5 MBit/s EF reservation (UDP traffic)

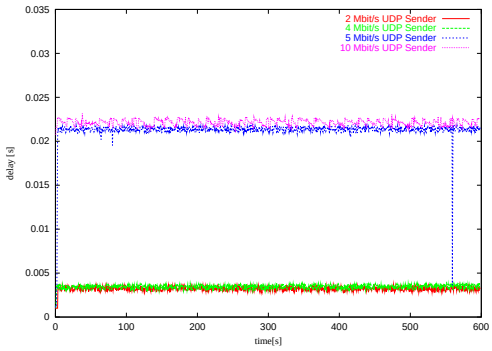

Figure 5.11: Delay for 5 MBit/s EF reservation (UDP traffic)

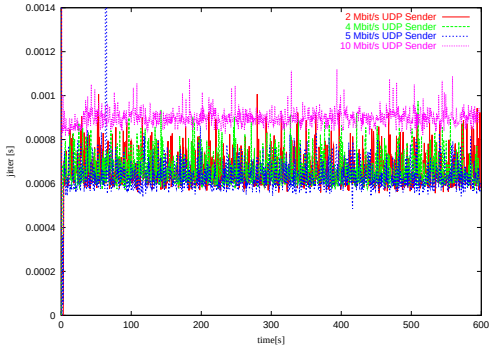

Figure 5.12: Jitter for 5 MBit/s EF reservation (UDP traffic)

Results for UDP traffic with 5 MBit/s AF reservation

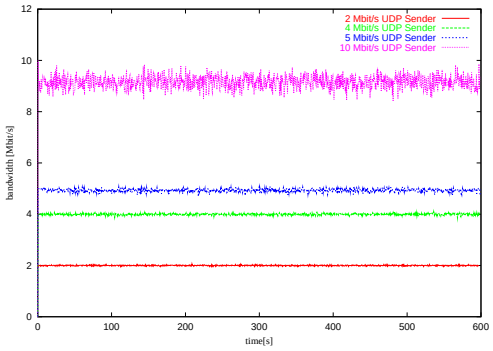

Figure 5.13: Bandwidth for 5 MBit/s AF reservation (UDP traffic)

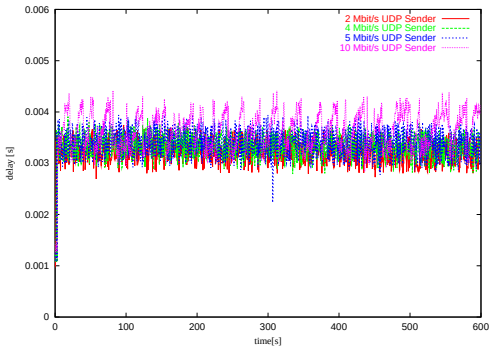

Figure 5.14: Delay for 5 MBit/s AF reservation (UDP traffic)

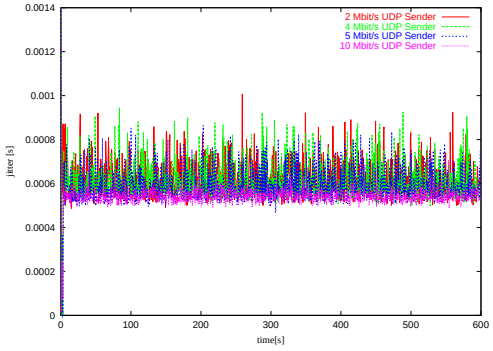

Figure 5.15: Jitter for 5 MBit/s AF reservation (UDP traffic)

traffic is strictly shaped in the former case (see Figure 5.10) we observe a significant amount of out-of-profile traffic crossing the routers in the latter case (see Figure 5.13). This amount heavily depends on the configuration of the TRIO queue and the sender behaviour.

We can assume, that the fill-state of the TRIO queue is very small, because we assigned a 40% weight to the assured service class and we transmitted with a low rate. Therefore, a sender with no or just little bursts has a higher probability of seeing no or little queues. Its high dropping precedence traffic will also not be dropped with a very high probability although the TRIO queue starts dropping packets at the very beginning. It is therefore possible for the 10 MBit/s sender to get almost 9 MBit/s across the network with a reserved rate of 5 MBit/s (see Figure 5.13).

The delays of the four flows in Figure 5.14 show a timely constant behaviour with values between 3 and 4 ms. The delay for senders with lower rates or exactly at the rate of the precedence handler show a random distribution, whereas the out-of-profile sender at 10 MBit/s has periodical oscillations. We assume, that those oscillations are caused by the random number functions used by the TRIO queue. The jitter for each test was about 0.5 ms at a large variance which became smaller for higher rates (see Figure 5.15).

Summary EF is a good mechanism to guarantee bandwidth for UDP traffic even under a heavy background load. The packets are forwarded almost without loss ($<$ 0.001%) as long as the sender rate is below the configured rate of the traffic shaper. The behaviour of our sender and the burst protection of the token bucket filter resulted in some packet loss and filled the queues during the test when sending exactly at the TBF's limit.

The delay shows an increase from about 1 ms to about 4 ms compared to the empty network (see Figure 5.5) but this is a large gain compared to the situation without DiffServ, when we see a delay of about 30 ms (see Figure 5.8). The jitter increases by three orders of magnitude when we have additional background traffic (Figures 5.6 and 5.9) but DiffServ is — compared to best effort forwarding — able to provide better results for the jitter by a factor of 5 – 10 (Figures 5.12 and Figures 5.9).

AF also proves to be able to protect UDP bandwidth against heavy background traffic. There is no obvious drawback of using assured instead of expedited forwarding for UDP traffic. The delay values are even smaller for AF traffic at a rate directly at the negotiated limit or higher, because there is no strict shaping of the flow that could result in filled queues. The jitter values don't even seem to depend on the DiffServ service type, they are almost the same for assured and expedited

forwarding.

5.2.3 TCP experiments

The TCP tests were performed with a bandwidth reservation of 5 MBit/s, too. Like during the UDP tests we had four test runs with TCP senders restricted to a maximum bandwidth of 2, 4, 5 and 10 MBit/s. The TCP implementation of the sending Host A determines the TCP payload size.

Expedited Forwarding The queue settings — especially the queue length — had to be adjusted to the specific behaviour of TCP. Since the packet frequency cannot be adjusted like for the UDP sender, we had to increase the queue size in order to allow bursts. Otherwise, if the queue length is too short, TCP packets would be dropped causing TCP's congestion control mechanism to reduce the bandwidth significantly below the reserved value. This would severely affect the DiffServ tests. Therefore we used a queue, that was four times larger than the queue used for the UDP tests. The queue length results in a constant maximum delay of 100 ms.

The resulting bandwidth diagram for expedited forwarding with TCP shows a very large variation (see Figure 5.16), especially for values at or above the rate limit of the token bucket filter. This behaviour is a result of the TCP congestion control function, which decreases bandwidth after a single packet loss. Averaging over a time interval of 100 ms those bandwidth breakdowns can be observed in the diagram, the use of larger intervals would smoothen the plot. For lower bandwidth values, the influence of the congestion control is not very significant.

The delay is approximately 5 ms for the two senders with 2 and 4 MBit/s (see Figure 5.17). Since the congestion control limits the bandwidth to values below the rate of the token bucket filter, the queue is not filled up and therefore the delay is also approx. 5 ms, even for the sender at the negotiated limit. In particular, TCP senders with higher rates fill up the queue before the congestion control can limit the bandwidth and so the maximum delay of 100 ms can occur in these cases.

The jitter values decrease with increasing bandwidth from 10 to 3 ms which can be explained by the stream-oriented TCP service. Due to that feature it is not so simple to influence the TCP packet size and the transmission frequency as it was in the UDP case. The TCP sender will send a few packets at a high rate and then wait, until the average bandwidth is below the token bucket filter's limit. Therefore, the queue length a TCP packet will encounter at the router is not constant for all packets, resulting in a higher jitter.

Results for TCP traffic with 5 MBit/s EF reservation

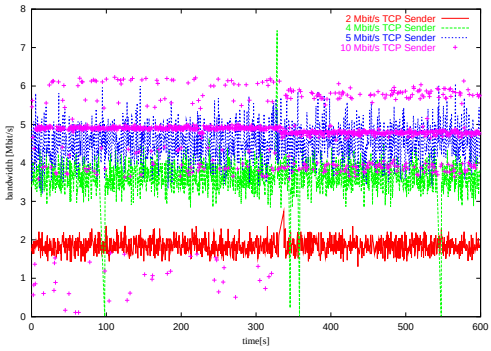

Figure 5.16: Bandwidth for 5 MBit/s EF reservation (TCP traffic)

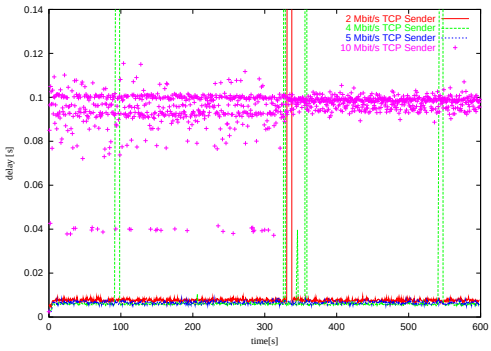

Figure 5.17: Delay for 5 MBit/s EF reservation (TCP traffic)

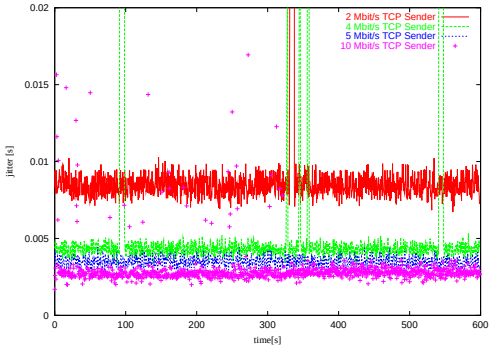

Figure 5.18: Jitter for 5 MBit/s EF reservation (TCP traffic)

Results for TCP traffic with 5 MBit/s AF reservation

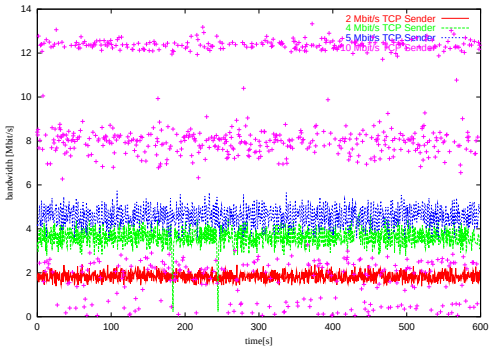

Figure 5.19: Bandwidth for 5 MBit/s AF reservation (TCP traffic)

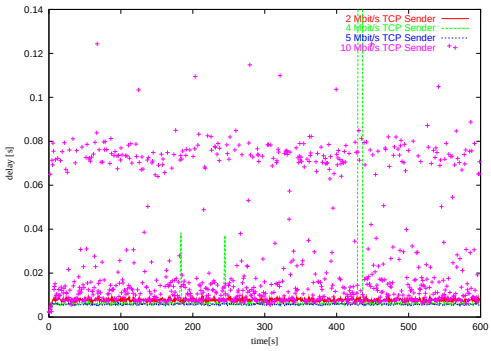

Figure 5.20: Delay for 5 MBit/s AF reservation (TCP traffic)

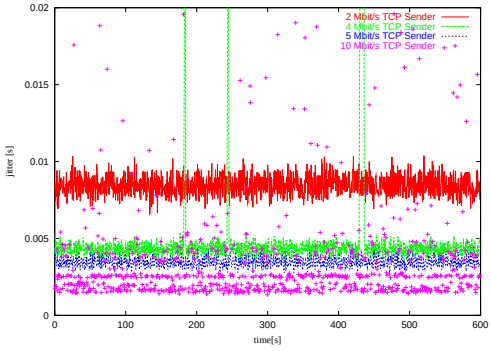

Figure 5.21: Jitter for 5 MBit/s AF reservation (TCP traffic)

Assured Forwarding The queue parameter configuration for assured forwarding is the same as in Section 5.2.2, because the TRIO queue length is configured in units of packets, not bytes. Therefore, we used the same 200 packet queue as during the UDP experiments.

Figure 5.19 shows that the bandwidth for AF is not as constant as for EF. This can be explained by the use of a TRIO queue, that randomly drops packets, even if the buffer is not filled up. Therefore, especially for higher bandwidth values there is a certain probability for packet loss and in that case TCP's congestion control decreases the bandwidth. On the other hand, the sender is now allowed to send more than the negotiated limit. All these issues give the large bandwidth variation for assured service.

As long as the sender meets the rate of the `prec_handler`, the size of the AF delay is 6 – 7 ms (see Figure 5.20). This means that the AF traffic is marked with low dropping precedence. For higher bandwidth, some packets will be marked as high dropping precedence and probably get lost. The retransmission of those packets results in a larger delay and a high delay variation. In this case, also the jitter varies significantly (see Figure 5.21) compared to the jitter during the former tests.

Summary The TCP results show, that even for rates below the negotiated bandwidth limit of the shaper the achieved bandwidth is much more irregular than with UDP. This is true for both, EF and AF. Averaging at a larger timescale would show better behaviour, but for applications which need a constant bandwidth at a small timescale the congestion control is a serious obstacle.

The delay of TCP is in the same range as the delay for UDP packets, but we have to reserve additional buffer space that is four times larger to handle the burstiness of the traffic. This results in high delay for higher transmission rates than negotiated. The jitter values for TCP are larger than for UDP. This is mainly due to the different algorithms we used for the sender programs. Since TCP does not allow to specify the packet size we could not increase the packet frequency and therefore the time between two subsequent packets could be up to 20 ms (see discussion in section 5.1.2).

AF is not the optimal choice for TCP, because TCP cannot take advantage of the possibility of sending out-of-profile traffic. This is due to TCP's congestion control, because out-of-profile packets will be dropped with a higher probability, causing TCP to reduce the bandwidth. Therefore, we can see frequent bandwidth breakdowns and because of retransmissions a great variance in delay will occur.

Note, that mixing TCP and UDP into a single service class is not recommended [10, 93, 146]. The problem lies in TCP which is backing off in cases of packet

loss. This allows UDP to send as much as desired and get most of the traffic sent through the network.

Summary

In this part we gave a short overview of our implementation of a DiffServ router based on Linux. A short description of the available kernel modules and a description of the router's configuration used during the performance evaluation have been presented. During this performance evaluation we tested the behaviour of UDP and TCP flows, that used either expedited or assured forwarding. Our results show, that it is possible to protect certain flows against aggressive UDP background traffic. Both, expedited and assured forwarding are able to provide a guaranteed bandwidth together with very low delay and jitter. Compared with today's best effort traffic forwarding we were able to lower delay and jitter by a factor of 5. We can conclude, that DiffServ is a reliable and scalable concept to support Quality of Service in the Internet.

Part III

Design and Implementation of a Bandwidth Broker

Overview

In order to support mobile users and their QoS requirements we propose a novel bandwidth broker architecture and also a QoS signaling protocol [158, 157] that provides enough functionality to solve the problems mentioned in the Introduction and has a flexible programming interface to add arbitrary extensions in the future. Our bandwidth broker architecture can be split into two different parts: a management part and a configuration part. Several other architectures use this separation, too [134, 193]. The novelty in our approach is in the interconnection of those two layers: We propose to use an object-oriented virtual representation of the underlying network. This interconnection is made available by the QoS Management API [160].

The architecture presented here has been implemented in C++ and evaluated to show which size of network can be managed. Our performance evaluation showed a very good call admission rate (CAR) (about 0.5 ms per flow) even at large network sizes (1010 nodes) [159]. As far as we know, no other bandwidth broker implementation has been tested with such a large network.

Even if the performance evaluation showed a good performance there may be situations where a higher CAR is required. In such an environment, distributing the load between several bandwidth brokers can provide better performance. A hierarchy of bandwidth brokers can be applied, where the leaf brokers manage a small part of the network independently.

In this part we will first present the design of the QoS Management API in Chapter 6: The main classes are handled in Sections 6.2 – 6.4, additional classes are presented in Section 6.1. An example for an implementation of the API and the use of the polymorphism of the classes can be found in Section 6.6.

Chapter 7 contains the design and implementation description of the bandwidth broker itself. The different components are specified in Sections 7.1 – 7.4. The performance evaluation is shown in Section 7.6. Interaction between multiple bandwidth brokers and a hierarchical composition of bandwidth brokers are described in Sections 7.7 and 7.8

Chapter 6

A Generic Management API for QoS Support

An important task for an ISP offering guaranteed services by using the Differentiated Services (DiffServ) approach to its customers is to configure the routers of its network according to the customers' requirements. The negotiation of technical service parameters between the customer and the ISP, also called a Service Level Specification (SLS), contains a description of the end-to-end Quality of Service the customer expects to be provided (even by routers that are not in the domain of the ISP). The resulting amount of configuration to own routers and communication to foreign networks is normally performed by an entity called a Bandwidth Broker (BB).

In Figure 6.1 we show an example of a user requesting a certain service for a connection across two ISP's networks by sending a SLS to the first bandwidth broker. This broker interprets the SLS in its management section and forms a SLS to be forwarded to the neighbouring bandwidth broker and a Traffic Conditioning Specification (TCS) for its own network. This TCS contains information about how each router has to be configured (on an abstract level) and will be forwarded to the configuration software that is able to create the corresponding configuration commands for each involved router.

From Figure 6.1 we can see, that our bandwidth broker architecture is separated into two layers: the management and the configuration layer. The QoS management API is used to connect the two layers with their different functionality and requirements. In order to specify the API we first have to specify the requirements the broker has to fulfil:

We will now define a generic QoS management API. A bandwidth broker may use the API to configure different types of routers within its own domain. It will be

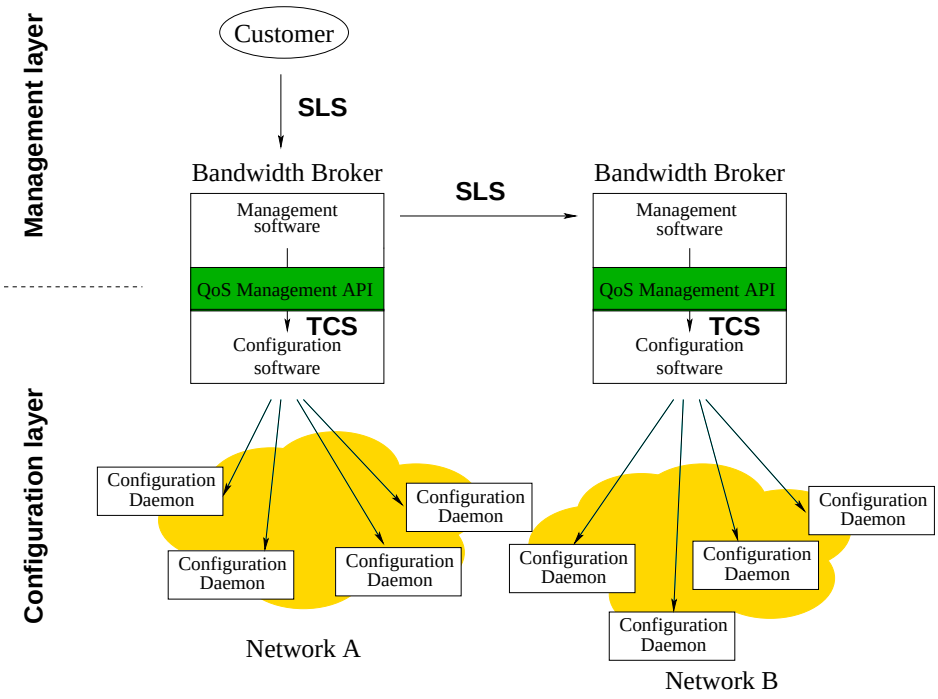

Figure 6.1: The Bandwidth Broker Architecture

an interface between the management software, which is not aware of the different types of routers within the underlying network and the configuration software, that is specially designed to optimally support each router type. The routers may differ in their capabilities or configuration interfaces. The main focus is on the independence of the API from router hard- and software and on an easy extensibility to any kind of new routers.

The independence of the API from router hardware or implementation details is best achieved using an object oriented design where the objects representing different implementations are derived from a common base class. The ideal solution for this polymorphism is to create abstract classes containing a certain set of virtual functions (methods) that have to be implemented by each derived class. This function set forms the interface by which the class can be accessed and conceals the different implementations from the application programmer.

Management software will likely be a critical part of an ISP's network management system. First, performance is an important issue. To be able to handle a large amount of user interactions together with the resulting configuration communication we have chosen C++ as the programming language offering object - oriented concepts at high performance. But another issue is also very important: Since an

ISP will certainly loose both revenue and customer goodwill if its network is temporarily unavailable for rebooting (e.g. to upgrade the system) this is undesirable yet normally inevitable. An effective way to meet this challenge is using dynamic code which ensures a up-to-date system running continuously. The use of dynamic C++ classes [127] combines the advantages of the object oriented design and dynamic linking to a management system that provides permanent availability together with the possibility to integrate new hard- or software features to the system during runtime.

The QoS management API consists of three abstract base classes (represented in C++ notation in Figure 6.2): the `Router`, the `Interface` and the `TrafficConditioner` (`TC`). Additionally some data classes representing e.g. IP addresses or flows are used in the functions. The base classes provide a generic interface to the programmer by their virtual functions. An application programmer can build a virtual image of the network that is to be configured consisting of objects derived from an API-`Router`. A configuration application can send generic configuration commands to each `Router` object. The objects will then translate them to the hardware-specific commands that are finally sent to the router.

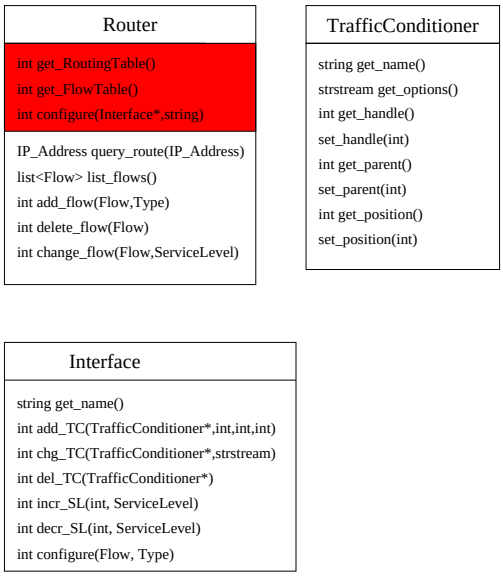

Figure 6.2: Overview of the API classes

In the following subsections we describe the member functions of those base classes together with the implementation-independent data classes, that are not shown in Figure 6.2.

6.1 Data Classes for the API

IP Addresses The `IP_Address` class is the parent class providing the common functionalities of IPv4 and IPv6 addresses. It is an abstract base class, i.e. no objects of that class can be instantiated; only pointers or references to `IP_Address` objects are allowed. An IP_Address object contains a version number an an address length field together with a binary address data field in network byte order. Since `IP_Address` is abstract, for constructing and copying special "virtual" constructors (see [163] pp. 424 f) are provided.

The class and all of its subclasses are declared in a namespace called `IP` that also includes several boolean operators (less, equal, not equal), input/output operators and functions for matching IP addresses with subnet masks. One important member of this namespace is the `IP::versions` map. This map holds a pointer to an object of each derived class indexed by the IP version number. This map enables us to transfer IP address classes from one network node to another: the sender transfers the IP version number, the address length and the address data, the receiver can choose a correct prototype of the IP address in its `IP::versions` map and call the virtual constructor of that prototype. The receiver has now a copy of the IP_Address object of the sender in a completely architecture-independent way. This is e.g. used in the (de-)serialisation function pair for IP addresses.

IP Routing The `Route` class represents a basic abstract concept of an IP route. Its only members are the destination address, a netmask and the next hop address. It completely ignores more advanced concepts of IP routing, such as metrics. Nevertheless, it is enough to help creating a image of the network topology.

The `RoutingTable` class is a simple extension of the STL `list` template. The only new member is a subscription operator, that allows simple access to the route towards a given destination address. This is a very intuitive approach to represent a routing table query, and is also open for better, more performant implementations of a routing table, not based on the rather slow STL `list`. Unfortunately, it is not possible to implement the routing table using the STL `map`, because the key of this map would be an `IP_Address`. Since the STL `map` requires a non-reference key type and the `IP_Address` class is abstract (i.e. only pointers and references to `IP_Address` are allowed) this simple improvement cannot be performed.

Service Level Specification The `ServiceLevel` class is a generic service level for a DiffServ Router. It contains mainly a bandwidth value, that specifies the average amount of traffic of a flow. Depending on the service that is requested, additional parameters can be specified: In detail, this class contains

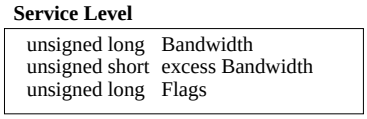

Figure 6.3: Service Level Description format

- a bandwidth value that specifies the average bandwidth of the flow in terms of kbit/s,
- an excess bandwidth value that gives the amount of bandwidth a short-time burst may need
- a realtime flag, that indicates delay and jitter sensitivity of the flow,
- a loss sensitivity flag, whether the flow is critical against packet loss or not,

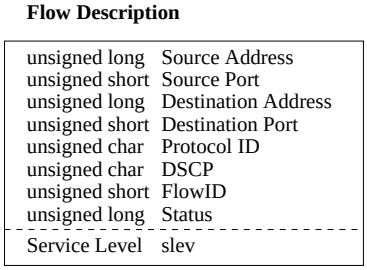

Figure 6.4: Flow Description format

Flow Description For the communication between the mobile host and the bandwidth broker and also for inter-broker communication an abstract flow description has to be specified. The flow description shown in Figure 6.4 can be mapped to different QoS strategies provided by the network by bandwidth brokers as long as the requirements of the flow are fulfilled. This packet contains the following information to specify a flow together with a certain service level:

- Source address and source port,
- Destination address and destination port,
- Protocol ID (TCP or UDP),

- DiffServ CodePoint (DSCP)
- a status word, providing information about the status of the reservation (e.g. in work, ready, in progress etc.)
- a flow identification number,
- a service level descriptor

The `Flow` class represents the concept of a flow, which is the set of packets going from a source to a destination. The source and the destination is determined by a (IP address , netmask , port) tuple. Each entry may also be wildcarded. Further differentiation can be made on the transport protocol which is used and on the DiffServ service codepoint.

A very important part of the flow description class is also the description of the service this flow requests / possesses. This description is implemented as a separate `Service Level` class (see Figure 6.3) for easy extensibility, if more elaborate service description is needed.

The `FlowTable` class is an extension of the STL `map` template. Since each flow possesses a unique flow identification number it's a natural choice to use that number as the key for the map. This `FlowID` is set by the bandwidth broker that has to ensure its uniqueness. In a scenario with multiple bandwidth brokers, each Flow ID consists of a (Broker ID, Flow ID) pair to ensure the uniqueness of the ID not only in the own subnetwork but also in the whole network (see Figures 7.15 and 7.17). Another advantage of the unique Flow ID will be seen in a mobile environment, when modifications of domain-crossing flow happen more frequently (Chapter 8).

6.2 The `Router` Class

The `Router` class provides a set of functions that every vendor of a DiffServ router has to provide in order to be manageable by our software. Its child classes will contain all data structures for interfaces and the routing- and flow tables. Those data structures can be the data classes presented in Section 6.1, but also hardware - dependent classes can be used as long as they provide an interface to the API functions.

The set of member functions of the `Router` class can be divided into two parts (see Figure 6.2): the dark marked members are used for communication with the real world router and its interfaces and their use is restricted. Management applications can only use the other functions to manage the router.

The *get_type()* function returns some type information about the router that can be used by the application to develop device-specific code.

The *get_name()* function returns the hostname of the router. The IP address is fetched by usual DNS resolve methods.

The *query_route(IP_Address)* function returns the IP address of the next hop for traffic to the given destination address. The translation of IP addresses to a pointer to the corresponding `Router` representation can easily be made by an external map.

The *get_RoutingTable()* function is used to fetch the routing table from the real world router it represents.

The *get_FlowTable()* function is designed for getting the flow table of the router. If there is more than one flow table in the router, they have to be mapped to a single one. Those flow table contains information about the flows that will be handled by the DiffServ implementation. The corresponding `FlowTable` data type is a list of `Flow` instances (see above).

The *configure(Interface*,string)* function is used for sending a router configuration command to the interface pointed to by the first argument. The interface related part is held in the `string` argument. The `Router` class can choose the way of communication and perhaps add the correct preamble to the command depending on the interface type of the first argument. This function is designed to be called by the `Interface` class after composing the correct command line.

The *list_flows()* function gives a complete list of all flows that are registered at this router.

The functions *add_flow(Flow, Type)*, *delete_flow(Flow)*, *change_flow(Flow, ServiceLevel)* are used to change the content of the router's flow table. The `Flow` data class is a router-independent flow description (see section 6.1). The `Type` argument of the `add_flow` function specifies how to configure the router type for this flow. This parameter refers to the particular flow only. It denotes whether the router acts as an ingress, intermediate or egress router for that flow. Those functions can also call the `Interface` member functions to add the necessary traffic conditioners and/or adapt the service level.

6.3 The `Interface` Class

The `Interface` class is designed to represent the whole set of network interfaces that could be part of a router. `Interface` objects are most likely created

by the child classes of the `Router` during initialisation. Each `Interface` provides member functions that are important for the QoS management:

The *get_name()* function returns an identifier of the interface. This identifier could e.g. be the name of the interface within the router.

The *add_TC(TC*,int,int,int)* function adds a traffic conditioner to the forwarding path of the interface. The location within the forwarding path is addressed by the three integer parameters: each traffic conditioner has its unique identification number called `handle` (the first parameter) and it is normally attached to some other traffic conditioner, whose identification number is given as the second parameter (the `parent`). If the conditioner is the root of the forwarding path this parameter is zero. Some traffic conditioners (e.g. classifiers) can have multiple conditioners attached. This can be indexed by the third parameter (called `position`). The `add_TC` function can create the necessary traffic conditioner together with the correct parameters, which can be calculated from the actual service level.

The functions *del_TC(TC*)* and *chg_TC(TC*,strstream)* are used to change the forwarding path of the interface. Since the information of the location of the traffic conditioner within the forwarding path resides in the conditioner itself there is no need for another parameter. The `strstream` parameter of the `chg_TC` function is a generic way of setting different numbers and types of parameters for different traffic conditioners.

The two functions *incr_SL(int,ServiceLevel)* and *decr_SL(int, ServiceLevel)* change the overall service level of a certain service at this interface. They increase or decrease the allocated bandwidth of the service and adapt the excess bandwidth or the delay limits.

The *configure(Flow,Type)* function arranges the correct traffic conditioners in the forwarding path of the interface for the given flow. The arrangement of the traffic conditioners depends on the `Type` parameter, that specifies, whether the interface should be configured as an ingress, an egress or any other type of interface.

6.4 The `TrafficConditioner` Class

The `TrafficConditioner` class represents the concept of modules that are used in the forwarding path of an interface in order to perform the necessary actions to comply with the DiffServ Per-Hop-Behaviours [74, 78]. For QoS management it is not necessary to simulate the exact behaviour of each conditioner. We only need but a class for each “real-world” conditioner that knows the way how to address its counterpart (e.g. command line syntax). The way how to compose

the traffic conditioners to support a certain service has to be programmed within the `Interface` class. Some common rules for boundary and core routers can be used to automate the setup of DiffServ routers.

The *get_name()* function returns the name of the traffic conditioner.

The *set_handle(int)*, *get_handle()*, *set_parent(int)*, *get_parent()*, *set_position(int)* and *get_position()* functions handle the location parameters of the traffic conditioner within the forwarding path of an interface. The `set` functions are used by the `Interface::add_TC` member function to specify the three parameters. The `get` functions can be used by any application.

The *get_options()* function returns a `string` that contains the command-line options which have to be passed to a configuration command to set up this special traffic conditioner. As traffic conditioners of different types may have multiple and very different parameters (e.g. rate and depth for a token bucket filter or queue weights for a weighted fair scheduler) it is the most generic way to support different types of conditioners.

Note that there is no function for setting the parameters, because this can be easily done by the traffic conditioner's constructor call. The command line string can then be composed from the constructor arguments. In order to change the parameters of already existing and installed traffic conditioners we can use the `Interface::chg_TC()` function.

We have chosen only one base class of traffic conditioners (traffic conditioning blocks in [14]). Conceptually, we could have derived different kinds of blocks from that base class, such as classifiers, meters, markers, packet schedulers, queues etc), and we could again derive from a packet scheduler a weighted packet scheduler and a priority scheduler, and so on. This would lead to an inflation of classes differing in details only. It is better to let the application programmer choose a suitable set of traffic conditioners to manage and configure the underlying network.

6.5 Dynamic C++ Classes

The main focus of the bandwidth broker design is on flexibility, extensibility and on having generic programming interfaces and data abstraction, that allow a programmer to develop different broker algorithms using high-level commands that hide the network topology and heterogeneity.

Nearly all those requirements can be fulfilled using the object-oriented concept of polymorphism. In short, polymorphism is the ability of an object belonging to a

derived class to act as an object belonging to the base class. While this has already some advantages in terms of extensibility and generic interfaces, the real power of polymorphism is unveiled when some functions of the base class are declared `virtual` or pure virtual. Thus we can provide an own version of those functions for each derived class. This allows us the full extensibility of our software. Now we can add new derived classes to provide whatever behaviour we desire. We therefore have separated the interface (the base class) from the implementation.

Despite the big advantages of this technique it suffers from a drawback: we must recompile (or at least re-link) our software after adding new components. In a scenario that relies on a highly available system (as a bandwidth broker should be) this is not desirable. It would be more convenient if we could load new classes at runtime.

Here we can give only a short overview to dynamic C++ classes, a more detailed article can be found in [127].

6.5.1 Dynamic Class Loading

Loading C++ class code at runtime has no direct support in the C++ language. However, there is the possibility to use the mechanism to dynamically load C code: the `dl` functions `dlopen`, `dlclose` and `dlsym`.

The `dlopen` function can be used to open a dynamic library at runtime. If it is opened with the RTLD_NOW flag, `dlopen` attempts to resolve all symbols found in the library. In our case, this means that all objects defined in the library are instantiated during the opening. The classes declared inside the library can be accessed by the `dlsym` function. It is nevertheless impossible to know the class name in advance. Therefore we cannot create new objects of that class directly. The library providing the new class must also provide a way to create new objects via a pointer to the constructor of the class.

6.5.2 Autoregistration

A set of functions returning a pointer to a newly constructed object (also called makers) is sometimes called a factory. A very convenient and flexible way to implement such a factory is using the STL `map` to assign key values to the functions (e.g. the class names) and access the makers via these values. Usually the names to access a specific class should be unique and commonly used. Therefore, it is necessary, that the programmer of the new class assigns a name for it and that this name is automatically registered at the factory.

The key is to use a “proxy” class that does the registration for us together with the RTLD_NOW flag mentioned above. Since all defined objects are instantiated, we only need to implement a class (the `proxy`) that does nothing but registering the classes in the library at the factory.

6.5.3 The Factory of Dynamic Classes

To ensure a common behaviour of different router types and for providing a generic configuration interface the approach of abstract base classes has been chosen. However, it is very important, that different derived classes implementing the functionality of e.g. new router types can be provided without recompiling and even without restarting the configuration software. For this purpose, the API provides three so-called factory maps — one for each abstract base class. Those factory maps provide the constructors of all derived classes, and the constructors can be indexed by the class name. Encountering a new class name the map tries to locate and open a dynamically loadable object file with that specific name. The programmer of that file has to ensure that it will register itself at the correct factory map at the time, when the dynamic object file is loaded by the linker. This can be done by implementing a proxy class which performs the registration within its constructor [127]. A `LinuxRouter` object can now be created by

```
Router* LR=factory_Router["LinuxRouter"];
```

6.6 API Implementation for Linux Routers

For each type of hard- or software element (i.e. routers and network cards from different vendors or traffic conditioners with different algorithms) that is used and managed within the network, an own class derived from one of the base classes must be provided, that implements the functions of the API according to the needs and the functionality of the element it represents. This set of classes must be dynamic, so that the management application can load them during run-time. This enables us to support network elements, that were unknown at the time when we were compiling the management software.

Another important issue is to specify a communication interface between the API classes and the routers. This depends on an implementation specific way to read and/or write certain variables of the router. For example it can be achieved by SNMP messages or command line interface (CLI) configuration for many commercial routers.

Using the dynamic classes the application programmer is now able to construct a logical image of the network he intends to manage by his application. All management is then performed by calling the API functions. Those calls are then translated to hardware / implementation dependent function calls and afterwards sent to the individual network elements.

6.6.1 API Child Classes for a Linux Router

Figure 6.5 illustrates how the API classes are used to derive classes for modelling a Linux DiffServ Router, so that the application can use it and communicate with it. Each of those child-classes must implement the API functions in a hardware-specific way. The generic data classes (see Section 6.1) can be used to model the important parts of a router (e.g. the routing table, the flow table or some information about the network interfaces) in a way suitable for the API. Using the communication interface to connect to its hardware - counterpart, the child class can execute the necessary commands and fetch and filter the desired information, so that the generic data classes can be initialised.

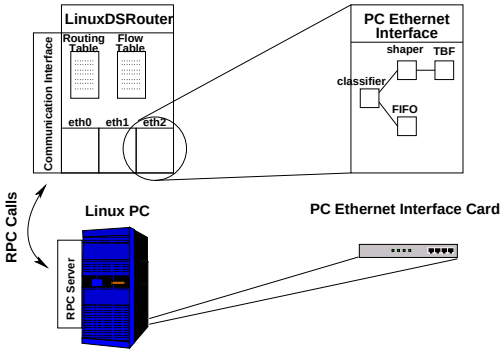

Figure 6.5: An application example for the abstract API classes

6.6.2 Linux Router Class

Any computer running the Linux operating system can easily be configured to act as a router (i.e. it forwards packets not destined to itself). Such a computer is referred to as Linux Router in this section. Since a Linux Router is not a commercial router, we need a special communication interface to send configuration commands and get some router settings (e.g. the routing table or a list of IP addresses).

6.6.3 Ethernet Interface Class

The `LinuxInterface` class is an implementation of the `Interface` class for a 100 MBit/s Ethernet interface for a Linux Router. This implementation contains the information, how a DiffServ service class is set up for a Linux router. This includes all command line syntax for the traffic conditioners as well as the knowledge about the conditioners' individual option syntax. In addition several variables for bandwidth management and admission control are included, such as the maximum link bandwidth of the interface or a maximum limit of the total amount of EF traffic that may not be exceeded. The usage of those variables is optional for the broker.

6.6.4 Traffic Conditioners

The implementation of `TrafficConditioner` classes for `LinuxRouter`s heavily depends on the implementation of the respective modules in the Linux kernel. All necessary information about command line syntax and options must be available via the API interface functions of the `TrafficConditioner` class. As an example we will focus the explanation on the `dsclsfr` class.

An exemplary command line syntax for the `dsclsfr` module is:

```
tc qdisc add dev eth1 parent 1:1 handle 2: \
   dsclsfr as1 0.05 as2 0.1 as3 0.1 as4 0.1 log_size 63
```

The three integer numbers after the `parent` and `handle` keywords specify the position of the conditioner in the forwarding path. Those values are passed to the function call placing the conditioner (i.e. the `Interface::add()` function). Since each conditioners' location is defined by those three values, the `TrafficConditioner` class provides an interface for reading and setting. The remaining numbers are special options for the `dsclsfr`. Of course, the number of those options will vary for each different conditioner, so the only generic way to pass options to different kinds of `TrafficConditioners` is via an option string, that is parsed by the object itself. This means also, that the object instantiating the `TrafficConditioner`s (i.e. the `LinuxInterface`) must know about the command line options for each conditioner it instantiates.

6.6.5 Other API Implementations

The Linux Router API implementation was not the only implementation that has explicitly been carried out. For the performance evaluation presented in Sec-

tion 7.6 we had to implement a "virtual Linux Router" class that we could multiply start at a single Linux PC. Of course, besides the disabled execution of configuration commands, the difference between the two implementations is negligible. However, there is also the possibility to implement classes for very different router hardware (i.e. routers from Cisco, NEC, Allied Telesyn etc.) or nodes that can only be managed by SNMP. We did not perform such implementations, but we shortly list the necessary steps that have to be performed:

- Implement a class derived from the `Router` class:
  - Investigate, how to retrieve routing information from the router and implement the `get_RoutingTable()` function accordingly.
  - Repeat the previous step for the `get_FlowTable()` function correspondingly.
  - implement the `configure` function, that it sends the command line formed by the `Interface` class to the router
- Implement a class derived from the `Interface` class:
  - The `Interface` class is responsible of setting up the traffic conditioners in a way that DiffServ can be supported. This is mostly done during the initialisation.
  - The `Interface` class must also provide a way of setup, deletion, and modification of a single traffic conditioner.
  - The `Interface` class must update and query the load and utilisation databases in case of flow setup, deletion or modification.
- Implement classes derived from the `TrafficConditioner` class: This might be the easiest part, since only the generic interface of the `TrafficConditioner` class has to be changed to parse the configuration parameters given to the traffic conditioner. In addition, the parameters of the conditioner's location within the forwarding path have to be adjusted.

Chapter 7

Bandwidth Broker for Large DiffServ Networks

The QoS Management API presented in Section 6 provides an object oriented way to configure different router hardware in a homogeneous way. It is based on three objects (`Router`, `Interface` and `Traffic Conditioner`) that are required to model a DiffServ network [16]. Those objects implement the different configuration facilities via derivation. In this chapter we present an architecture using the API to build a bandwidth broker able to manage large DiffServ networks. The bandwidth broker uses those three classes to build a representation of the underlying network topology. It can configure the router hardware by using common configuration commands (such as `Router::add_flow`). Those generic commands are translated into the correct configuration scripts by the corresponding `Router` instance. This solves the problem of missing router support by adding a software layer capable to deal with various routers but hiding the differences behind a generic interface.

The API additionally provides the necessary functionality to keep track of the amount of bandwidth reserved for a specific DiffServ class at each interface of each router. Therefore, the bandwidth broker is also capable to manage the bandwidth in order to either minimise the configuration/signalling overhead or to deny flows that cannot be served due to limited bandwidth resources (e.g. in a wireless access network).

Figure 7.1 shows the bandwidth broker architecture in detail. We will first focus on the simpler case, that the bandwidth broker manages an isolated network, i.e. all flows that have to be set up have its source and destination in the same network/domain. We also omit admission control for this first exemplary case to keep it as simple as possible. A policy database allowing admission control based

on subnet masks is presented in Section 7.7, where we develop a more elaborate bandwidth broker architecture that is able to support multiple ISP internetworks.

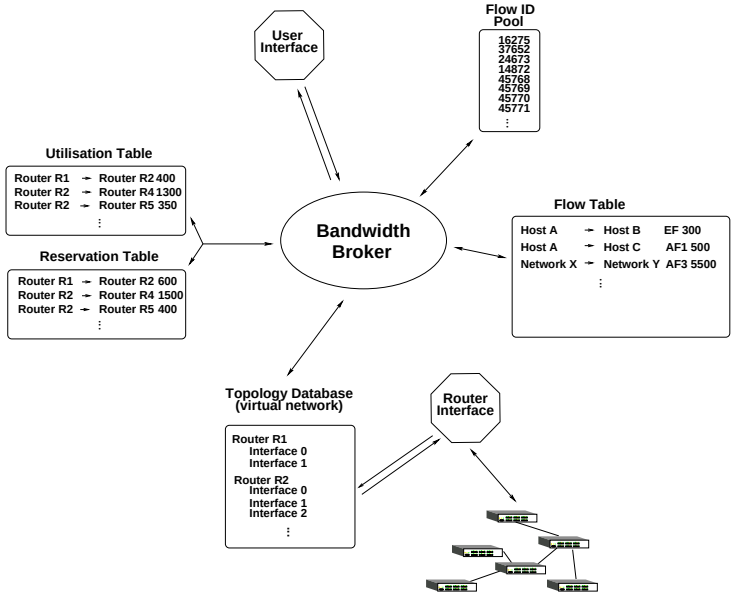

Figure 7.1: The Bandwidth Broker Architecture for an isolated network

The management layer of the bandwidth broker has to fulfil several requirements in order to be able to configure the network:

- it has to know the topology
- it has to keep track of the resources that are currently allocated
- it has to perform call admission control
- it has to communicate with adjacent bandwidth brokers
- it has to communicate with users to negotiate resource reservations

Several building blocks are thus needed to form the management layer of the bandwidth broker:

- topology database
- reservation and utilisation tables
- policy database

- flow table
- user and broker communication interfaces

The topology database naturally forms the interface between the management layer and the virtual network. It is automatically built during the startup phase and contains all IP routing information about the network. This information is held in routing tables that are part of the `Router` objects of the virtual network.

Both communication interfaces — broker-broker and user-broker are built on top of a TCP/IP server-client model that offers a functionality similar the the SUN RPC model. The server offers a certain set of functions (e.g. `add_flow`, `del_flow`) the client can call; the client can pass parameters to the functions and receives a result. Broadcast functionality is included, too.

The bandwidth management part consists of the reservation and utilisation tables, together with the policy database. Those components form the decision base about admission or rejection of a flow: a flow can only be admitted if the resulting bandwidth utilisation on any link is lower than the limit configured in the policy database. The policy DB contains such limits for each ingress link and each connected foreign subnet according to the contracts negotiated with neighbouring bandwidth brokers.

The configuration layer consists of hardware - dependent configuration daemons running one at each router. The communication between the configuration layer and the virtual network is unspecified. Virtually any means of configuration protocol can be used (e.g. SNMP, CLI). In our implementation we use a TCP/IP socket based protocol offering a command line interface to the Linux `traffic control` configuration API. The variety of this communication is again hidden from the management layer by the polymorphic virtual network.

7.1 The Topology Database

During the initialisation the broker automatically generates a network representation. This can be done either by reading a local topology database from the hard disk, or by broadcasting a broker advertisement message. Each router has to reply to this advertisement message giving its host name and its router type. We chose this approach because of its stability against changes in the topology, although we had to accept the resulting need of a router daemon running on each router. Since we nevertheless depend on a router daemon to perform the configuration commands from the bandwidth broker, this is not a big drawback.

After this step the bandwidth broker fetches all routing tables from the network. It now has the full knowledge of the topology on the IP layer, which is sufficient for our task (i.e. configuring DiffServ flows). In addition, all necessary traffic conditioners needed to support DiffServ are already set up and a default configuration is loaded. This default configuration simply consists of empty DiffServ reservations at each interface.

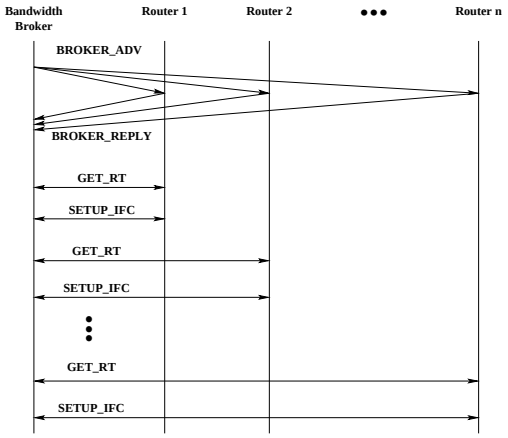

Figure 7.2: Setup of the Topology Database

7.1.1 Topology Auto-detection

This feature enables the bandwidth broker to detect the topology of the network it is managing. It relies on a communication server running on each router of the network. This server has to reply to a broadcast call answering its hostname and the router type. This type information may for example contain the router vendor and series and is the basis of the decision how the router is addressed for configuration. After the broadcast the broker has a list of hostname / type pairs and can now build an image of the topology, consisting of one `Router` object per node. The correct derived type of the `Router` class is chosen by the router type information, and the routing table is fetched automatically by the `Router`'s constructor.

7.1.2 The Naming Service

The virtual network representation within the bandwidth broker consists of a collection of pointers to `Router` objects. Those objects are unique and should never

be copied since they can become very large. To map the hostnames to the `Router` pointers, a DNS-like service has been implemented, that returns the pointer corresponding to a given IP Address / hostname. This service is initialised during the startup phase of the bandwidth broker.

7.2 Bandwidth Management and Admission Control

It is not advisable to reconfigure a router every time a flow is registered that crosses that router. This would lead to a large communication overhead and thus significantly decrease the overall network performance. There are some alternatives to that approach:

One way is to over-provision the different service classes, i.e. allocate more resources to a service class than there is currently reserved. This has the advantage of reducing the signalling overhead since we do not have to reserve additional bandwidth for each flow, but there possibly is enough bandwidth reserved for some flows by the over-provisioning algorithm. Only when a certain threshold is exceeded we adjust the overall reservation on the router (see also [67, 68]).

This approach has the possible drawback of wasting some of the reserved bandwidth and of allowing the users of a service class to transport more prioritised traffic than they negotiated. The latter case should not occur in a properly designed DiffServ network, since traffic shaping at the ingress point inhibits the hosts to send more than the negotiated limit. The first case, however, is also not important for the DiffServ implementation on Linux routers, because all of the unused DiffServ bandwidth can be used by best effort traffic and therefore is not lost at all.

7.3 Bandwidth Broker API Commands

The bandwidth broker offers several flow handling commands to a user that hide the heterogeneity of the network and necessary actions per router thus providing a simple, generic configuration API. Usually these commands have a flow descriptor argument and also a flow descriptor return value. The client can specify the flow it would like to set up as a parameter to the API function call and check if the setup was successful by looking at the flags field in the return value. Also the flow ID is returned in the appropriate field of the returning flow descriptor. This flow ID must be used for future reference (e.g. when using the delete or change commands).

The client software supports a customer in contacting the bandwidth broker to negotiate and manage resource reservations (see [158, 157]). The customer can specify a flow description — i.e. a {source address, source port, destination address, destination port, transport protocol} tuple — and additionally a service level description. This service level description contains in its basic form just the bandwidth and excess-bandwidth information together with two flags indicating the necessity of the flow to be handled by a low-delay or a low-loss service. This flow description is implemented as a separate object in the flow description class and can therefore be adapted to new demands very easily.

7.3.1 Flow Establishment

The `add_flow` function must be called for establishing a new flow. The function expects a `Flow` object to be passed via the communication interface. If the network can be configured to handle the flow, a new, unique flow ID is created, the status of the flow is set to FLOW::ACCEPT, and the flow is returned to the calling process via the communication interface.

To set up the flow and configure the network the following steps are performed:

- choose an appropriate DSCP. We will use the `set_dscp` function for this purpose.
- create a new flow ID. This function ensures, that this flow ID is unique for the management domain of the bandwidth broker.
- perform a traceroute to get all routers that eventually need to be reconfigured
- configure the ingress router. The ingress router always has to be reconfigured, as it is responsible for marking and shaping the flow. The configuration is performed by the `Router::add_flow` function.
- configure the core and egress routers. To reduce the signalling expenses the bandwidth broker uses over-provisioning.
- the actual traffic load and the actual reservation are updated for each interface involved in the flow setup
- the global flow table is updated

7.3.2 Flow Deletion

The `del_flow` function is used for deregistering a previously installed flow. This function expects the `Flow` object via the communication interface and performs the following tasks:

- check the correctness of the flow ID and the flow.
- delete the flow from the ingress router (using `Router::del_flow`)
- reduce the load at all core and egress routers.
- eventually adapt the over-provisioning.
- return the flow ID to the global pool
- delete the flow from the global flow table.

7.3.3 Flow Modification

The `change_flow` function can be used for changing an already registered flow. It is possible to change bandwidth, service level and also parts of the forwarding path. Also this function expects a `Flow` class as an argument. This flow class must contain the flow ID of the original flow in its flow_id field. In particular, the `change_flow` function does the following:

- The original flow is fetched from the global flow table, using the flow_id variable of the new flow.
- The difference between the old and the new path is computed.
- If the ingress router has changed, the flow is deleted from the old ingress router by the `Router::del_flow` function, and the new ingress router is configured via `Router::add_flow`.
- The bandwidth used by the old flow is released and the bandwidth used for the new flow is allocated if necessary. If there is a part of the route that is used by both flows, we have only to update the bandwidth on that part.

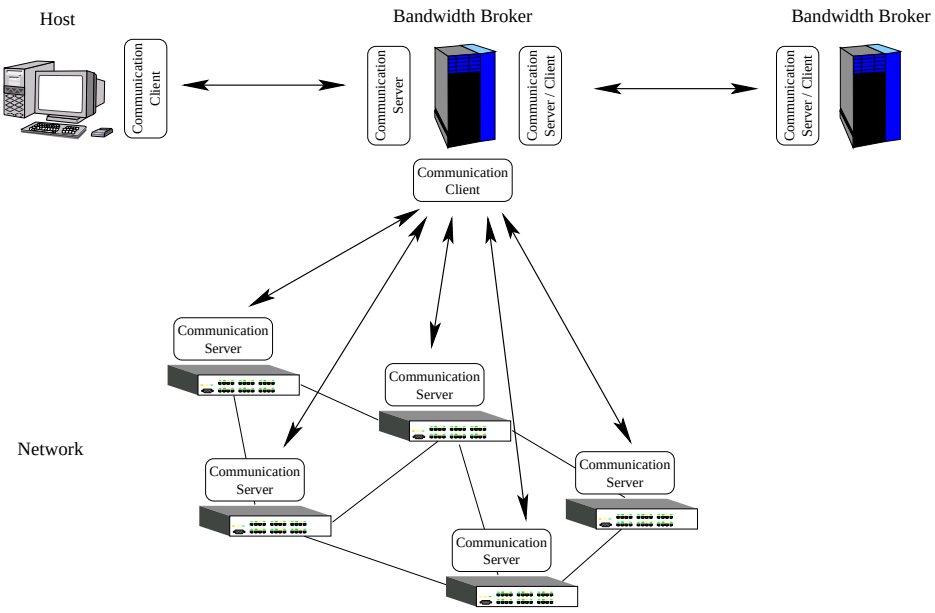

Figure 7.3: Communication Architecture for the Bandwidth Broker

7.4 Communication Subsystem for the Bandwidth Broker

For the communication between the broker and the other entities (e.g. hosts, network elements or other brokers) a threefold communication architecture has been developed (see Figure 7.3): Broker-router communication, broker-user communication and broker-broker communication. The architecture is based on the server - client model and offers a functionality very similar to the SUN RPC [120, 155] model. Each communication server provides a set of functions a client can call. The server executes the function with some parameters specified by the client and delivers the result back to the client. Internally, our implementation uses RPC function calls to carry out the communication. This can be changed to a implementation based on TCP sockets only.

7.4.1 The Communication Server

The set of functions a communication server provides is usually a subset of member functions of a certain class (e.g. a bandwidth broker class or a Linux router configurator class). Therefore, if we want to implement a general communication server class, we have to deal with member functions of different classes. One so-

lution could be to establish static wrapper functions in each class and ensure, that only one object of each class is instantiated (which would be a valid assumption for a bandwidth broker within a network). We chose a more elegant solution to this problem: The communication server is implemented as a template class indexed by the type of the class it belongs to. This way the type of the member functions is known to the server and it can access the member functions using pointers.

Each communication server opens two sockets: one for unicast connections and one for broadcast. Since we have to use UDP for the broadcast socket we can assign the same port number to those two sockets. The port number the server has to listen to must be provided by the correspondent main class. During the construction of the main class all member functions of the main class that have to be accessible from outside are inserted in a list of function pointers within the server.

7.4.2 The Communication Client

The communication client class connects to a communication server on a given host and port. The communication client class provides two functions, a static broadcast function and a unicast function. Using these functions, the user can specify a call parameter, that defines the function he would like to execute on the remote system. In addition, a single parameter can be given to the remote function and a result can be passed back. For binary or higher-level functions that require more parameters, a separate data structure should be defined that contains all parameters and can be passed as one single parameter. The broadcast function contacts each communication server in the network and gathers the results in a list. This function is static, which means it does not belong to a single instance of a communication client and therefore does not have a specified hostname. Depending on the port number the broadcast function uses, a different class of communication servers is called.

7.4.3 Serialisation

Serialisation is the term commonly used for converting an object into a stream of bytes for the purposes of storing it onto the disk or transmitting it over the network. Basically, serialising an object involves serialising each of the object's data members in a well-defined order. Each of the data members should support serialisation, thus serialising a high-level data structure is done recursively.

If an object dynamically allocates memory, the amount of this memory is transmitted first, to allow the client side to reconstruct the object by allocating memory correctly. Serialisation in our scenarios therefore includes the serialisation of the heap state of an object but not the execution stack and the register state. This would only be necessary if we would like to serialise objects that perform long computations, to avoid a restart of the computation from the beginning. In our case, to transmit call parameters to functions, pure heap state serialisation is sufficient.

The implementation of the serialisation consists of two function templates (one for serialisation, one for deserialization) with specialisations for each object that is not a built-in type. The specialisations take care of the correct memory allocation and object dereference for higher-level objects and cannot be deduced from a general implementation. The choice of using function templates instead of the usual overloading of input/output operators provides a greater flexibility in the number and type of call parameters to the serialisation functions (e.g. different file descriptors, flags, etc.).

7.4.4 The Bandwidth Broker — Router Communication

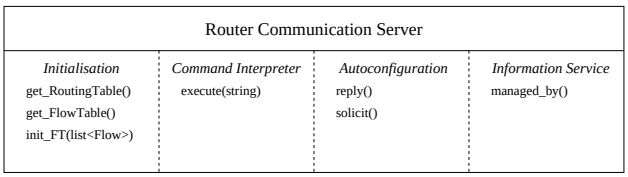

Figure 7.4: The Router Communication Server

Each router in the network has to provide a configuration daemon to execute the functions needed to configure this router. This daemon is a communication server providing the functions to the bandwidth broker. The broker calls these functions each time it initialises or (re-)configures the router. The functions of the communication server can be divided into four parts. The first part includes the functions corresponding to the private member functions of the `Router` class. These functions are only called during the initialisation. The second part consists of one function only (the `execute` function) which provides a way to execute shell commands or scripts directly at the router. It is called from the various flow configuration functions of the bandwidth broker. The third part consists of two functions (`reply` and `solicit`) that are used by the topology auto-detection feature of the broker (see Section 7.1). The last part also consists of just one func-

tion that returns the name of the bandwidth broker configuring this router. This function will be used in a multi-ISP scenario (see Section 7.7).

7.4.5 The Bandwidth Broker — User Communication

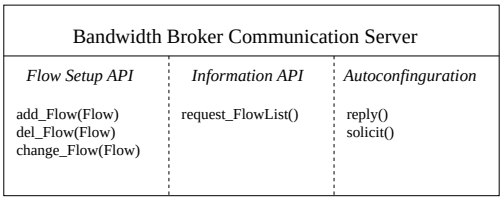

Figure 7.5: The Broker Communication Server

The broker itself provides a communication server offering all broker commands (cf. Section 7.3) to the user. A graphical interface offers a user-friendly way to call the different broker commands and to keep an overview about the flows that are currently set up.

Negotiation of a new SLS The messages that need to be exchanged between the host and the bandwidth broker in order to set up a new SLS are shown in Figure 7.6. The messages are exchanged using the TCP protocol. Authentication information also has to be included, but this issue is not considered in this scenario.

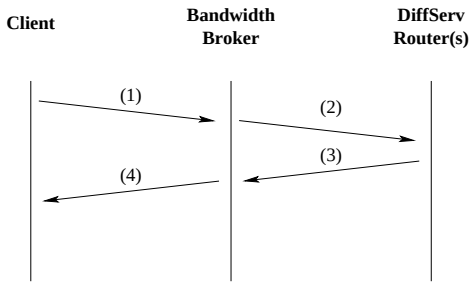

Figure 7.6: Message sequence for negotiating a new SLS

The messages needed to negotiate a new SLS as the following:

1. The initial request message defining the Service Level of the new flow. This message contains the data structure shown in Figure 6.4, describing the flow

specification in an abstract way. With this message, the broker can decide on how to configure the routers in a way that best fits the current network topology.

2. The bandwidth broker translates the abstract packet data into a concrete router configuration. Now it tries to set up the routers that are involved during the transmission of the flow. The bandwidth broker can also check in advance, if there is enough bandwidth reserved to accept the flow and reject the SLS if this is not the case (see message (4)). The API described in [160] manages all the translation and configuration and also provides the functionality to manage the bandwidth that is reserved.

3. Each router reports success or failure of the configuration back to the bandwidth broker.

4. The bandwidth broker reports the status of the SLS back to the mobile host. Failure can be caused by errors during the configuration or — most likely — by a unavailable amount of bandwidth.

7.5 Client Software and User Interface

Each client that wants to contact the bandwidth broker to negotiate a reservation needs an implementation of a communication client (cf. Figure 7.3) that calls the correct broker functions. In order to support the user and to make the negotiation more convenient, a graphical user interface (GUI) has been implemented. The GUI offers an input menu (see Figure 7.7) where the user can fill in his requirements. All other necessary actions are performed by the client software. In particular this actions are:

- search for a bandwidth broker via broadcast calls
- call the appropriate broker API functions
- display the flows after successful reservation

7.6 Performance of the Architecture

The performance evaluation of our architecture shows how fast the bandwidth broker can handle flow requests while managing a reasonably large network. In

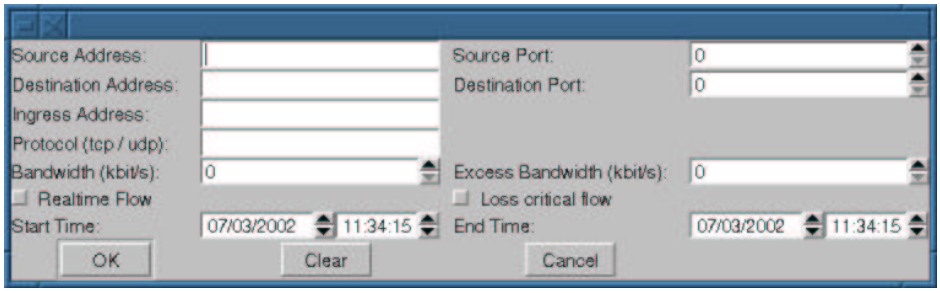

Figure 7.7: GUI Input Menu

addition, we were interested in the memory consumption of the bandwidth broker, which we expect to be quite large because of the central topology and flow databases. For the evaluation we had to emulate a large number (about 200) of routers (more precisely: router configuration daemons) on a single PC, since no existing DiffServ network consisting of several hundred nodes was available. Therefore we had to restrict the actions of the router daemons: no flow tables have been written to the kernel memory or to the hard disk, and no configuration commands have been actually executed. Otherwise a huge overload on the PC hosting the router daemons would arise. On the other hand these restrictions do not compromise the reliability of our results because of the parallelism of these actions: executing one action after another on a single machine produces a large delay but executing them in parallel on several machines is done very fast.

7.6.1 Evaluation Scenarios

For the performance evaluation several test topologies have been created using the tiers [45] program. This program randomly generates a topology consisting of a single WAN with several MANs and several LANs per MAN. The number of routers per network can be chosen. We generated seven topologies from 157 nodes up to 1010 nodes and tested our broker with those topologies. As an example, Figure 7.8 shows the topology consisting of 535 nodes. The routing tables were generated by the Bellman-Ford distance vector algorithm [11, 61]. Each routing table contains the distance (in a given metric; here: hops) to each router in the network.

The measurements have been performed on Linux PCs in our network laboratory. The different platforms available are listed in Table 7.1. All PCs are interconnected via full-duplex 100 MBit/s Ethernet lines.

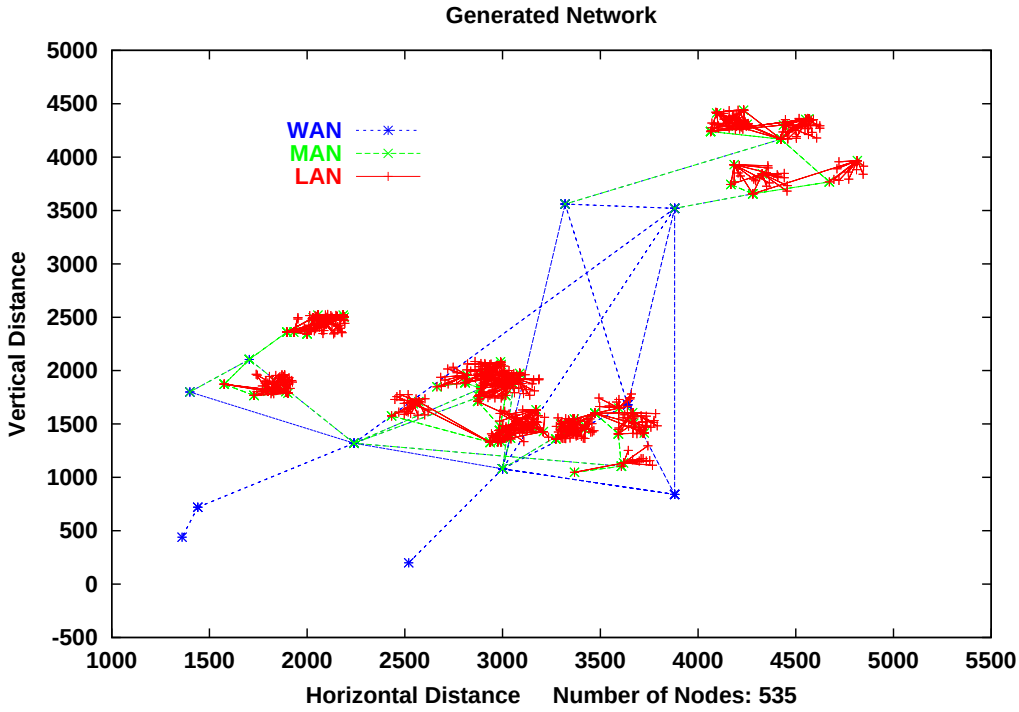

Figure 7.8: An exemplary tiers-generated topology

7.6.2 Results

Memory Consumption

The memory consumption of a centralised application is always a critical issue. Our bandwidth broker relies on two databases that can grow very large: The topology database containing the routing tables and the flow database. Of course, the amount of memory used does not depend on the platform. Therefore, we only present the results for one platform.

The memory consumption of the flow database is not very critical because only a very small amount of data has to be stored per flow, and the memory consumption of this database grows linearly with the number of flow requests (see Figure 7.10). The actual amount of memory used per flow during the experiments with several topologies can be found in Figure 7.9. This table shows the amount of memory that has been used for the flow database and other allocations that have been made during a test performing 1000 flow requests.

However, in our scenarios the routing tables will grow quadratically with the number of nodes in the topology (see Figure 7.11). This is an effect of the RIP pro-

CPU	Memory	Bus Clock	external bus	name
2x AMD Athlon 2000+ MP	2048 MB	266 MHZ DDR	U3-SCSI	europa
2x Pentium III 800 MHz	512 MB	133 MHz	U2W-SCSI	atlas
AMD K6-2 525 MHz	192 MB	100 MHz	EIDE	buran
AMD K6-2 400 MHz	192 MB	100 MHz	EIDE	challenger
Pentium II 333 MHz	192 MB	100 MHz	U2W-SCSI	saenger

Table 7.1: The Laboratory Platforms

Topology Size	Flow DB Size (kB)
157	1596
208	1760
305	1480
535	1368
710	1620
928	1332
1010	1652

Figure 7.9: Flow Database Size in kB

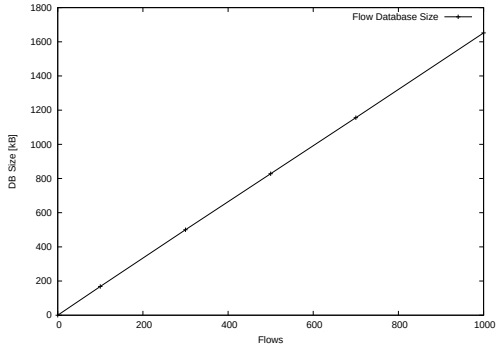

Figure 7.10: Flow Database Size for 1010 Nodes Topology

tocol, which creates a routing table entry for each router of the network. Nevertheless we can estimate, how much memory a router instance consumes when its routing table contains a reasonable number of entries (ca. 10000 - 30000). This is approximately the size of the routing table of a backbone router in a large ISP network. We assume a simple quadratic equation of the form $y = ax^2 + c$ and estimate the memory of a router that contains 30000 entries. As a result, only about 15 MB would be needed for such a backbone router. A workstation equipped with 4 GB memory therefore would be able to hold the topology database of more than 250 backbone routers, each having a routing table of 30000 entries. To manage a large network consisting of 1000 backbone routers, a single server with 16 GB memory would be sufficient. Another way would be to separate the network into four parts and assign each part a workstation with 4 GB memory (see Section 7.7). Larger networks can be managed accordingly (assuming the size of the routing table does not change).

Flow Setup Time

For measuring the speed of the bandwidth broker, a small client application has been implemented to request small flows as fast as possible. This means we send a

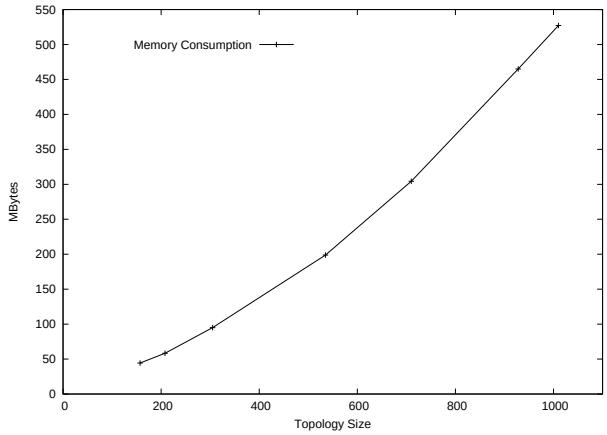

Figure 7.11: Memory Consumption of the Bandwidth Broker

new flow request as soon as the reply of the bandwidth broker arrives at the client. The bandwidth broker processes the incoming requests in sequential order, since no parallel communication server has been implemented yet. This is subject to future work and could further improve the performance. With this application we were able to measure the flow reservation speed. Up to 1000 flow requests have been performed for each topology.

For timing measurements two different strategies were used. The goal is to get the performance of the bandwidth broker. The first strategy is to measure the time between the sending of the flow request and the receiving of the reply at the client. To do so, a simple stopwatch timing is sufficient, therefore we used the UNIX `time` command. This command measures the total (wall clock) time a command needs for execution, as well as the "user time", the time that is spent in the program code, and the "system time", the time that is spent by the system, for example to perform I/O calls.

The second strategy is to measure each internal function call of the bandwidth broker and to add up these times. To do so we used the Tuning and Analysis Utilities (TAU,[151]). This is a set of tools to analyse the performance of C, C++, Fortran, and Java programs. By using this toolkit, we were able to measure the execution speed of the broker's `add_flow` function (see Table 7.6).

Performance of the Different Platforms First we will investigate, how the response time of the bandwidth broker depends on the computing power of the host the bandwidth broker runs on and / or on the computing power of the configurator host. For this purpose we performed tests using all the different platforms avail-

able. The first experiments consisted of co-located tests where bandwidth broker and configurators ran on the same machines. This eliminates possible effects of the interconnection network. We did an experiment setting up 1000 flows on the smallest topology (157 nodes). The following results have been achieved (see Table 7.2):

Host	Wallclock time	User Time	System Time
europa	0.715s	0.070s	0.060s
atlas	0.887s	0.080s	0.040s
buran	2.857s	0.220s	0.140s
challenger	3.436s	0.400s	0.140s
saenger	2.314s	0.190s	0.070s

Table 7.2: Performance on a single host

By distributing the broker and the configurator host, it can be seen (cf. Table 7.3), that the performance of the broker host has a much higher impact on the overall performance than the configurator host (all flow requests have been sent from host europa):

Broker Host	Configurator Host	Wallclock time	User Time	System Time
europa	challenger	0.837s	0.070s	0.050s
europa	saenger	0.839s	0.090s	0.040s
challenger	europa	2.813s	0.030s	0.010s
saenger	europa	2.203s	0.090s	0.050s
saenger	buran	2.416s	0.050s	0.010s
buran	saenger	2.583s	0.020s	0.010s
enterprise	challenger	2.714s	0.050s	0.020s
europa	atlas	0.780s	0.080s	0.040s

Table 7.3: Performance on distributed hosts

In the last two lines it can be seen, that depending on the computing power the influence of the interconnection network can vary. Using a reasonable fast broker host, the interconnection network is no bottleneck and can almost be neglected. For this reason, we performed the remaining experiments using the host europa solely.

Best-Case Scenario The fastest response a client will get from a bandwidth broker (if we assume a positive answer) will be if there is enough bandwidth allocated along the path already and therefore just the ingress router has to be reconfigured

(this has to be done anyway). In order to measure the performance of our bandwidth broker under such circumstances we performed an initial configuration of the network providing enough bandwidth on each link so that all subsequent flow requests were able to be admitted after the ingress router reconfiguration. Figure 7.12 shows the results we got for each of the seven topologies:

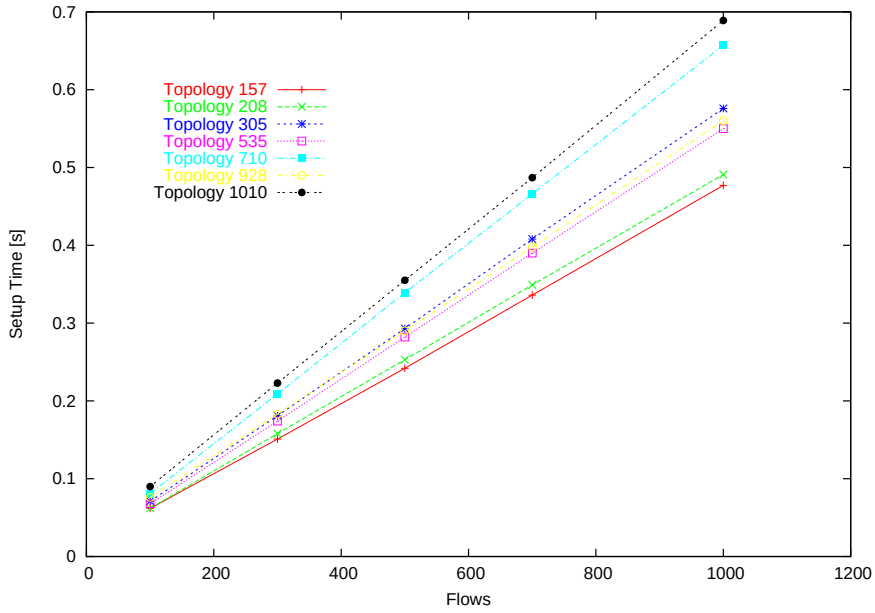

Figure 7.12: Flow Setup Time of the Bandwidth Broker (Best Case)

We can see that there is a linear correlation between the number of flow requests and the total setup time. This denotes a constant flow setup time for a given topology. Table 7.4 shows the speed averaged over 1000 flow requests for three topologies.

We can also recognise a small dependency of the topology: The topologies with 710 and 1010 nodes have a slightly lower speed than the one with 928 nodes. This can be explained if we look at the profiling results which give not only the time of a function call but also the number of times this function has been called. In Table 7.5 we can see that the topologies with 710 and 1010 nodes have a quite higher number of routing table queries. This indicates a bigger diameter of the topology which is a result of the randomness of the topology generation process.

Another interesting topic is the difference between the Client time and the Broker time. The client measures its time starting the sending of the reservation request up to the reception of a reply; the broker starts at the reception of the request and

Topology Size	Flow Setup Speed (Flow/s)
157	2096.4
208	2036.6
305	1736.1
535	1818.1
710	1522.0
928	1785.7
1010	1451.4

Table 7.4: Flow Setup Speed (Best Case)

Topology Size	Routing Table Queries	Diameter
157	9368	9.36
208	11508	11.5
305	16476	16.5
535	13228	13.2
710	20536	20.5
928	12218	12.2
1010	21328	21.3

Table 7.5: Total Number of Routing Table Queries

stops when the sending the reply. This difference tells us tell the speed of the communication interface. Table 7.6 shows the differences between Client and Broker time measured, averaged over 1000 Flow requests. We can see that the difference is about $10 - 20$ μs. This indicates a very high speed of the communication interface.

Worst Case Scenario In the worst case, a bandwidth broker has to reconfigure all routers along the forwarding path of a flow. This obviously results in a higher latency but fortunately the use of overprovisioning can reduce this case to a rare exception. To measure the bandwidth broker under such extremely bad circumstances we disabled the overprovisioning. As a result, each router had to be reconfigured. This case will be a rare exception. Normally a user will have a much lower latency.

Figure 7.13 shows the results for this test. As can be seen, that the time needed to set up a given number of flows is clearly higher than in the previous experiments shown in Figure 7.12. In Table 7.7 can be seen, that the Flow Setup Speed has dropped by almost 40%. However, taking into account the large number of

Topology Size	Client Time [ms]	Broker Time [ms]
157	0.477	0.451
208	0.491	0.480
305	0.576	0.554
535	0.550	0.521
710	0.657	0.640
928	0.560	0.527
1010	0.689	0.675

Table 7.6: Difference between Client and Broker Time per Flow Request

reconfigurations that have been performed, this is still quite fast. The reason is, that reconfigurations of core routers are much simpler to be performed than setting up an ingress router. A core router just needs reconfiguration of bandwidth limitations for the service classes, whereas an ingress router needs a new traffic conditioner to be inserted and configured (cf. Section 2.2.6).

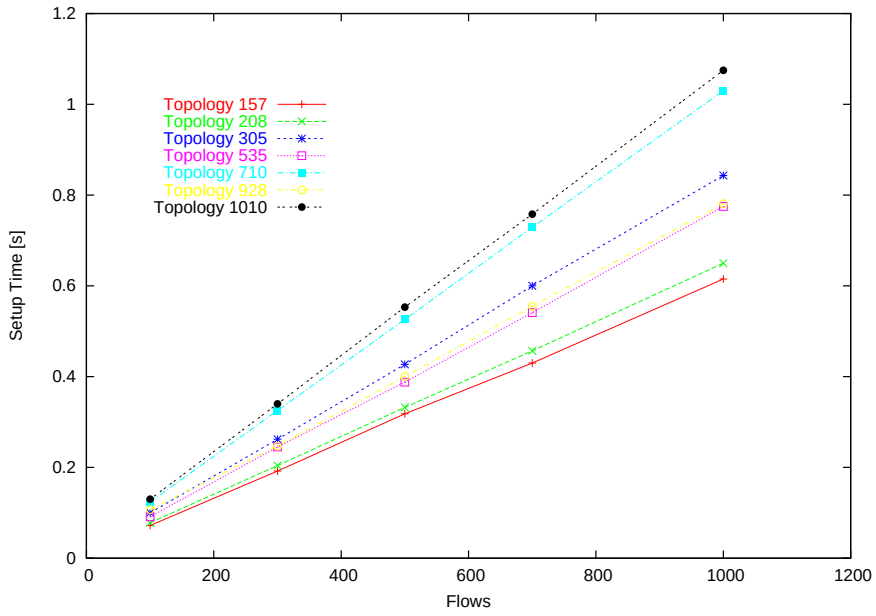

Figure 7.13: Flow Setup Time of the Bandwidth Broker (Worst Case)

Topology Size	Flow Setup Speed (Flow/s)
157	1626.0
208	1538.5
305	1186.2
535	1290.3
710	970.9
928	1280.4
1010	930.2

Table 7.7: Flow Setup Speed (Worst Case)

7.7 Multi-ISP Support

If we want to support reservations crossing multiple ISP domains, that are managed by different bandwidth brokers, we need several extensions to the previously presented architecture. In the following sections we will discuss the changes in the signalling protocol and we also need to adapt the implementation of the flow reservation and setup procedure since we need to communicate with neighbouring bandwidth brokers if a flow crosses several domains.

As an example we want to evaluate the scenario shown in Figure 7.14. Let us assume that Host A wants to set up a *bi-directional* reservation from itself to Host B. The route of this flow shall be: Router A1, Router A3, Router X1, Router X2, Router X4, Router B1, Router B3. In addition, for simplicity and scalability reasons there are only bilateral agreements between neighbouring bandwidth brokers, i.e. only neighbouring brokers may communicate with routers or brokers of a domain.

Several new steps have to be performed and new problems arise in the setup and configuration procedure:

1. After having received the flow description from Host A, Broker A has to detect that the flow requires a domain-crossing request.

2. Broker A has to make sure, that other bandwidth brokers along the path to Host B (Broker X and Broker B) set up the flow correctly and has to provide them with the necessary information. This is especially important, if the route from Network A to Network B is ambiguous: Broker X has no knowledge of the routing within Network A and therefore cannot know at which possible ingress router the flow from Host A to Host B enters its domain.

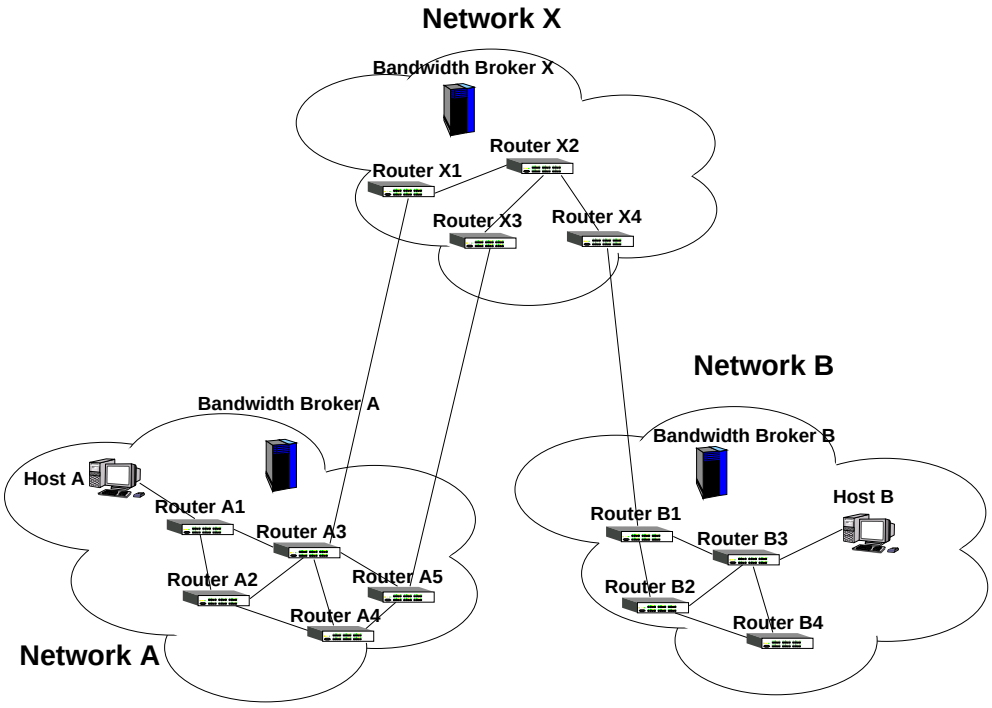

Figure 7.14: An exemplary multi-ISP scenario

3. Brokers X and B have to set up a reservation from the ingress to the egress of their domain.

4. Since the routing may be asymmetrical, the reservation for the reverse direction must be started from Broker B.

We have implemented the solutions to those problems as follows:

1. To detect a flow request crossing or leaving the domain is quite easy: As an example, the Naming Service of Broker A (cf. Section 7.1.2) does not know Host B, therefore this host cannot be in the domain of Broker A. The flow from Host A to Host B must be leaving the domain.

2. For the request of a resource reservation to neighbouring domains, Broker A constructs a new reservation request. To do so, two steps have to be performed:

   (a) Broker A discovers the first router in the next domain using the traceroute command on its topology database (in our example this is Router X1). This router serves as the ingress router of the following domain

and is inserted in the therefore newly generates `Ingress Router` field of the SLS Signalling Packet (as shown in Figure 7.17, see also Section 7.7.3).

(b) Broker A has now to detect the broker responsible for the subsequent flow setup. To do so, it requests from the ingress router (Router X1) the address of its bandwidth broker (Broker X).

Broker A can now send the reservation request to Broker X which in turn can continue with the flow setup. Broker X has to perform similar actions to form a correct reservation request to send to Broker B.

3. The knowledge of both, the source address and the ingress address allows both Brokers X and B to set up the reservation along the correct path as well as to perform admission control or policing based on the source address.

4. The problem of how to signal the Broker B to start the reservation for the reverse direction is handled separately in the next section.

7.7.1 Backward Reservation

Quite a difficult topic in multi-ISP (and thus multi-broker) scenarios is the topic of backward reservation: a user would like to reserve bandwidth for a flow from a specific sender (perhaps a multimedia server for digital video) to itself. This reservation is also called “receiver-initiated reservation”. In general this sender can not be expected to be in the same domain as the user, therefore the bandwidth broker of the server’s home domain has to be found. Since the routing may be asymmetrical, the simple solution of setting up reservations on the known path from the receiver to the sender is also not possible.

In RSVP all reservations are backward reservations, but in our case we prefer not to rely on support from the foreign ISP other than provided by the bandwidth broker. Another approach [135] uses modified ICMP echo request packets to record the route from the sender to the receiver. Our solution does not change the routers of the networks but is merely implemented in the code of the bandwidth broker.

If a bandwidth broker detects a backward reservation (i.e. it can find neither the sender’s IP address nor the ingress router’s IP address in its topology database) it does not perform any reservation but will pass the flow request unchanged to the next upstream bandwidth broker. It does not matter whether this is the correct bandwidth broker on the route from the sender to the receiver or not; the only purpose is to find the correct bandwidth broker in the home network of the

sender. The path the reservation request takes is the same path a forward reservation would take, but without any changes in the routers. Finally, the reservation request arrives at the broker of the sender. This bandwidth broker is now able to find the correct route and to perform the necessary configurations. Afterwards the correct downstream brokers are contacted, and finally the reservation is fully established.

The disadvantage of this approach is, that the reservation request first has to travel to the sender's network before the reservation can be established. This leads to an additional delay, but this penalty is met in other implementations, too. However, since no reconfiguration of routers has to be done, the additional delay will be comparably small. The main advantage of our approach is the independence of any support from third parties; the backward reservation is implemented in the bandwidth broker core and does not rely on special installations on the routers of the network.

7.7.2 New Architecture Components

Figure 7.15 shows the bandwidth broker architecture for multi-ISP internetworks. We can identify several new components:

Broker - Broker Communication Interface

An additional communication interface for broker - broker communication has to be added. The protocol between the brokers nevertheless can be the same as in the user-broker communication, as described in Section 7.4.5. Using this protocol the brokers can reserve aggregations of flows by masking the source / destination address just like any user can reserve a flow at the broker.

Policy Database

Usually it is necessary to restrict the amount of traffic entering a domain from the outside. Thus, a policy database containing the information about the amount of traffic foreign subnets are allowed to send is needed. This rather coarse access restriction can surely be improved, but it is sufficient to limit the number of users of a bandwidth-limited access network (e.g. a wireless access point). Using this policy database we can provide admission control based on the network identification of the sender / receiver address of a flow.

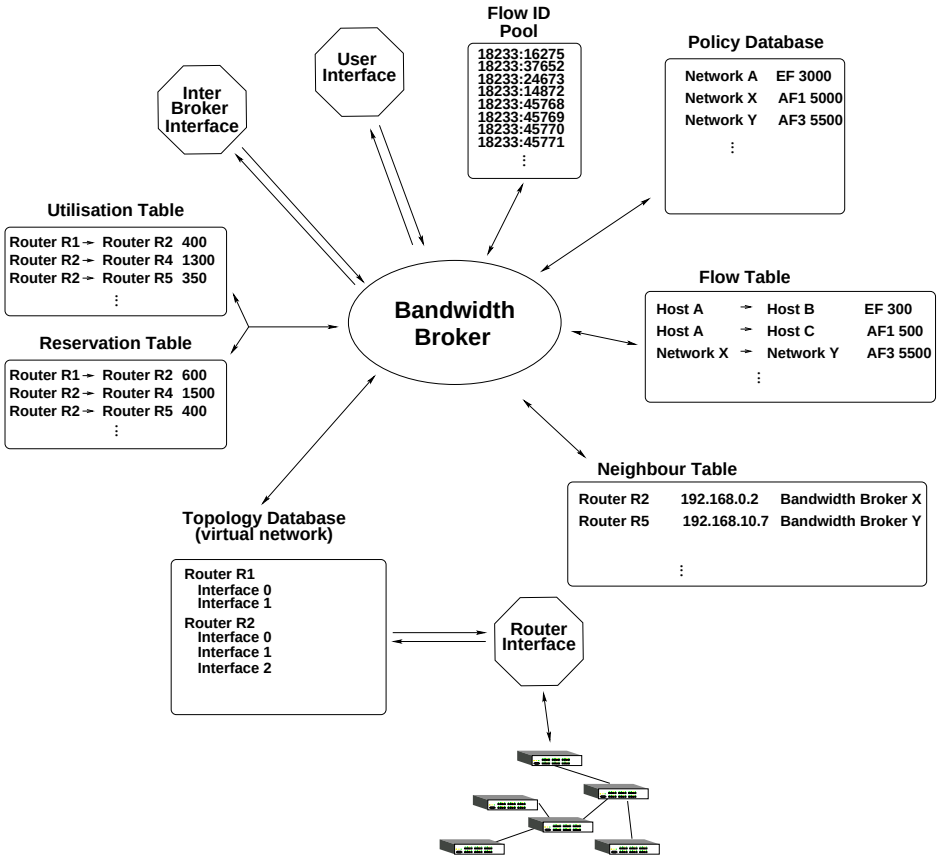

Figure 7.15: The Bandwidth Broker Architecture for multi-ISP scenarios

Neighbour Table

As already mentioned the bandwidth broker needs to ask ingress routers from foreign networks, about the bandwidth broker responsible to manage this foreign network. In order to minimise the signalling traffic needed, the bandwidth broker can save the name of a neighbouring bandwidth broker together with the corresponding egress router / ingress router pair in the neighbour table.

FlowID Pool

The FlowID Pool is not a new component of the multi-ISP capable bandwidth broker, however it had to be changed to provide not only network-wide unique flow IDs but globally unique flow IDs. Therefore, the format of the flow IDs has been changed to a pair of integer values, the first indicating the bandwidth broker,

which issued the ID, the second being created by the same algorithm as the one used in the single-ISP broker.

7.7.3 Signalling Protocol

The signalling protocol has to be extended in order to help the bandwidth broker to find the correct entry point of a flow into its network: suppose a user wants to set up a flow from Host A to Host B, thus crossing the networks of several different ISPs (see Figure 7.14). The user only specifies the source and the destination of its flow (see Section 7.4). With this information, Bandwidth Broker A can set up the flow until it reaches the border of its network (e.g. via Router A1, Router A3). Broker A has to ask Broker X to continue with the reservation. Unfortunately, Broker X cannot continue the flow setup without additional information: Broker X has no possibility to decide, where the flow from Host A to Host B enters its network. There are two possibilities (Router X1 and Router X3) and one has to know the internal routing of Network A to predict. Therefore the signalling protocol between the bandwidth brokers has been extended by an additional field: the ingress router (see Figure 7.17). This field contains the IP address of the ingress router where the flow enters a network. In the example above, the ingress router for Network A would be Router A1, and assuming the routing mentioned above, the ingress router for Network X would be Router X1. With this information, Broker X can continue the correct flow setup to Host B. The same signalling has to happen between Broker X and Broker B for the setup of the flow in Network B.

The implementation of the signalling protocol tries to minimise the amount of signalling messages needed in case of a setup failure: when receiving a reservation request, the bandwidth broker has to ensure, that there is enough bandwidth available for the flow request after the downstream domains have configured the flow. If in the meantime a flow has allocated an amount of bandwidth such as the initial request has to be rejected, all downstream domains must be notified, and all signalling and configuration work has been useless. To avoid this, the brokers previously check the possibility of the request being admitted before sending the request to neighbouring bandwidth brokers: first the policy database is asked, if the flow meets the policy of the network. Afterwards the requested amount of bandwidth is entered in the reservation table. This is a very fast operation compared to the reconfiguration of a router. Yet it ensures, that in the meantime no new flow request can allocate the bandwidth of the pending flow request. After the admission of the downstream domain arrives, the broker can now safely configure its network, and report the successful reservation setup to the user. Using this procedure the user is notified much faster if the reservation can't be served.

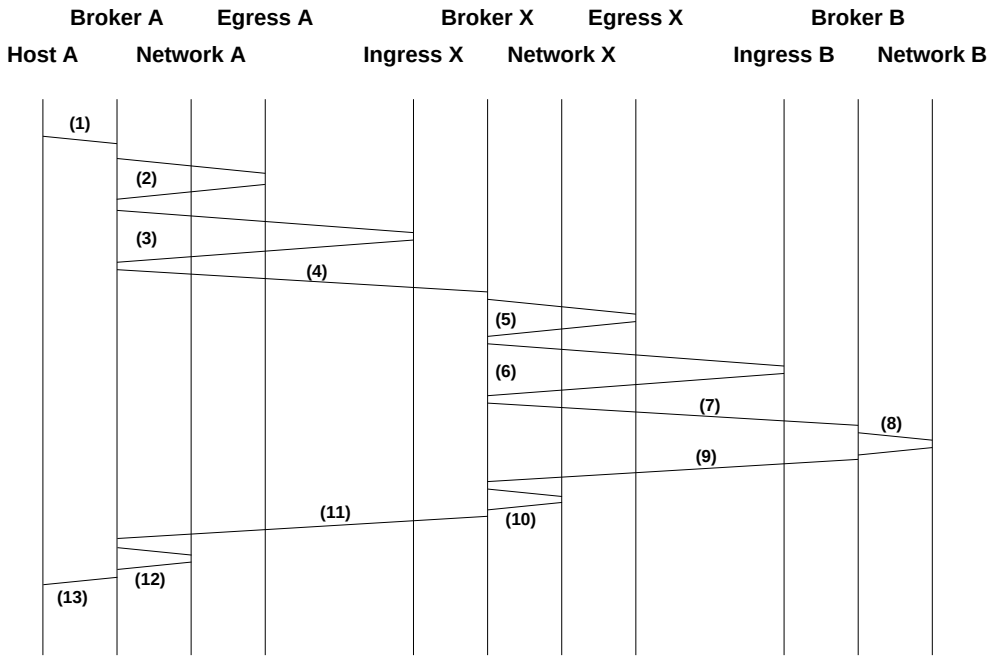

Figure 7.16: Signalling Protocol Messages

The messages exchanged during the reservation process are summarised below:

(1) Host A sends a reservation request to its home broker

(2),(5) The broker asks the egress router's routing table (or the neighbour table) for the ingress router address

(3),(6) The broker asks the ingress router of the foreign network (or the neighbour table) for the bandwidth broker of the foreign network

(4),(7) The broker forwards the reservation request with the updated ingress router address to the neighbouring bandwidth broker

(8),(10),(12) The broker configures the network

(9),(11) The broker reports the successful completion of the setup

(13) Host A receives the report about its reservation request

Flow Description (II)

unsigned long	Source Address
unsigned short	Source Port
unsigned long	Destination Address
unsigned short	Destination Port
unsigned long	Ingress Router
unsigned char	Protocol ID
unsigned char	DSCP
unsigned long	FlowID
unsigned long	Status
Service Level	slev

Figure 7.17: New SLS Signalling Packet Format

7.7.4 Multi-ISP-Capable Broker Commands

In this section we now summarise all changes made to the simple scenario described in the beginning of this chapter in order to get the multi-ISP scenario.

Flow Establishment

The `add_flow` command (cf. 7.3.1) is changed as follows:

- Choose an appropriate DSCP. If a DSCP is already set, map it to the DSCP used in the own network.
- If no flow ID is set, create a new one.
- Ask the naming service, if the sender or the ingress address is part of this domain.
  - If both are unknown, this is a backward reservation. Find the egress router to the sender.
  - Get the next-hop router (i.e. the ingress of the neighbouring domain) for the flow by looking at the egress routers routing table.
  - Ask this ingress router for its managing bandwidth broker.
  - Send the reservation request unchanged to the neighbouring bandwidth broker.
- Ask the naming service, if the destination is part of this domain.

  - If yes, configure the flow along the path from the ingress address to the destination address. If the ingress address is empty, use the source address instead.
  - If not:
    * Find the egress router of the flow in this network.
    * Get the next-hop router (i.e. the ingress of the neighbouring domain) for the flow by looking at the egress routers routing table.
    * Ask this ingress router for its managing bandwidth broker.
    * Write the ingress routers address into the ingress field of the flow description.
    * Send the reservation request to the neighbouring bandwidth broker.
    * If the neighbouring bandwidth broker accepts the request, configure the flow along the path in this network.

- Update the load and reservation tables.
- Update the flow table.

Flow Modification

The `change_flow` command is a little more complicated: Yet this command is very important in case of mobile hosts (see Part IV), if the location and/or IP address of a host may change and existing reservations nevertheless have to remain. Four different scenarios are possible: Changing the destination address of a flow, or changing the source address of a flow, both as a forward or a backward reservation. Figure 7.18 shows the different scenarios: in this figure each network is managed by an individual bandwidth broker. In each of the four scenarios those bandwidth brokers have to decide whether to add a part of the new branch to their network, to delete an old branch, to change an existing part in their network, or to do nothing at all. This decision is based on the existence of the globally unique flow ID and on a simple comparison of the sender/ingress and receiver address.

If the broker receives a `change_flow` command, it executes the following:

- If the flow cannot be found in the database
  - If the sender address or the ingress address is part of the broker's own network, it sends the `change_flow` command to the neighbouring broker in the direction of the receiver; after a successful setup, it configures the flow in the own network (this is exactly the same procedure as setting up a new flow).

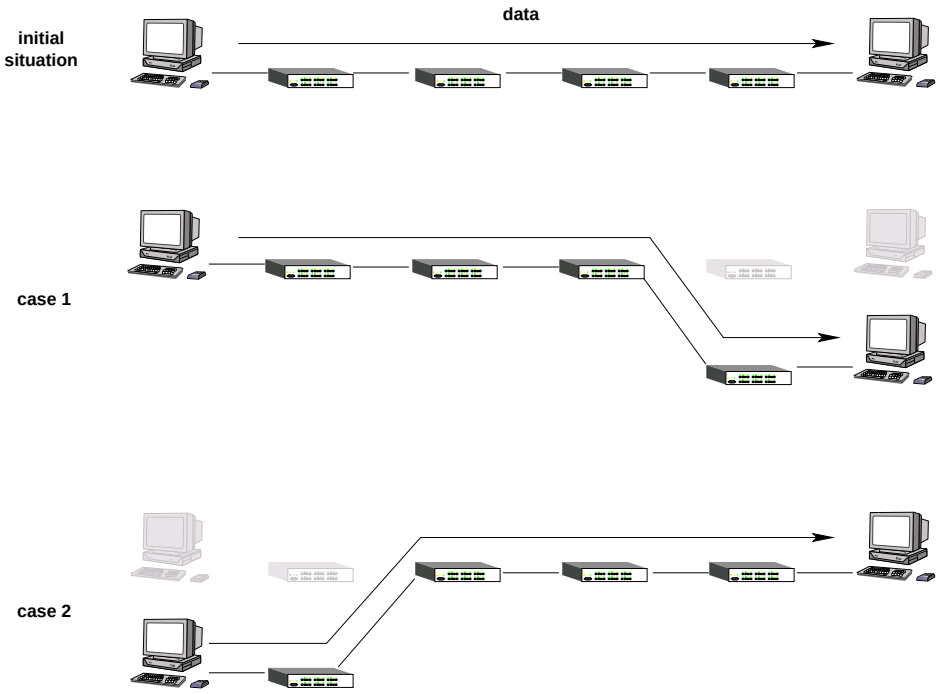

Figure 7.18: Changing an Existing Flow

  - If both, sender and ingress address are unknown to the broker, it sends the `change_flow` request to the next broker in direction of the sender (cf. backward reservation, Section 7.7.1).

- If the flow ID is found in the database:

  - If the receiver address has changed, it sends a `del_flow` command to the broker in direction of the old receiver, it sends a `add_flow` command in direction of the new receiver, and performs a `change_flow` in the own network.

  - If the sender address or the ingress router have changed, it updates the own flow table in a way, that the entry with the flow ID now contains the new sender / ingress address. Afterwards it sends a `del_flow` command to the source bandwidth broker. Deletion of this flow will be terminated at this broker, since the flows are no longer equal (the new source address has been inserted before). Finally it performs a `change_flow` in the own network.

7.8 Hierarchical Bandwidth Brokers

In Section 7.6 we have seen, that the performance of a single centralised bandwidth broker is limited by several factors: The memory consumption for a network consisting of several hundreds or even thousands of nodes is quite high. In addition, the processing speed of a request is limited by the CPU power of the bandwidth broker, thus only a limited number of reservation requests can be served per second. The idea of distributing the load among several bandwidth brokers, each managing an independent subset of a domain has already been discussed in Section 3.5.

From an architectural point of view a bandwidth broker in a hierarchy does not differ much from a bandwidth broker in a multi-ISP environment: like a "multi-ISP" broker such a leaf broker performs the local flow setup in its own network independently from other brokers, and if a flow traverses another network it will call the responsible bandwidth broker. The big difference between a broker hierarchy and a collection of peer brokers in a multi-ISP environment is the way how the routers in the network report to the responsible bandwidth broker: in a multi-ISP scenario the unique address of the bandwidth broker managing the network is returned (cf. Section 7.7), in a broker hierarchy two cases exist: if the request comes from a router that belongs to the same (bigger) network, the address of the leaf broker managing the subnetwork is returned. If the request comes from a foreign network, the address of the root broker is returned. This separation keeps the amount of signalling messages as small as possible.

7.8.1 Advantages of a Broker Hierarchy

The problem of load sharing between bandwidth brokers in a very large network with a high reservation request frequency could be solved by simply subdividing the network into smaller subnetworks, each managed by its own bandwidth broker. This would result in a simple multi-ISP scenario. A broker hierarchy has the obvious drawback of introducing an additional entity that has to be considered and more signalling is required to ensure the integrity of the network configuration.However, a root bandwidth broker can decrease the amount of signalling needed for flows crossing the network. We can see this in the example in Figure 7.19:

Each network is divided into several subnetworks, each managed by a bandwidth broker (small black square). The upper network implements a broker hierarchy. A Root Bandwidth Broker (big black square) is responsible for handling all external flow requests. The root bandwidth broker does not need to know the topology

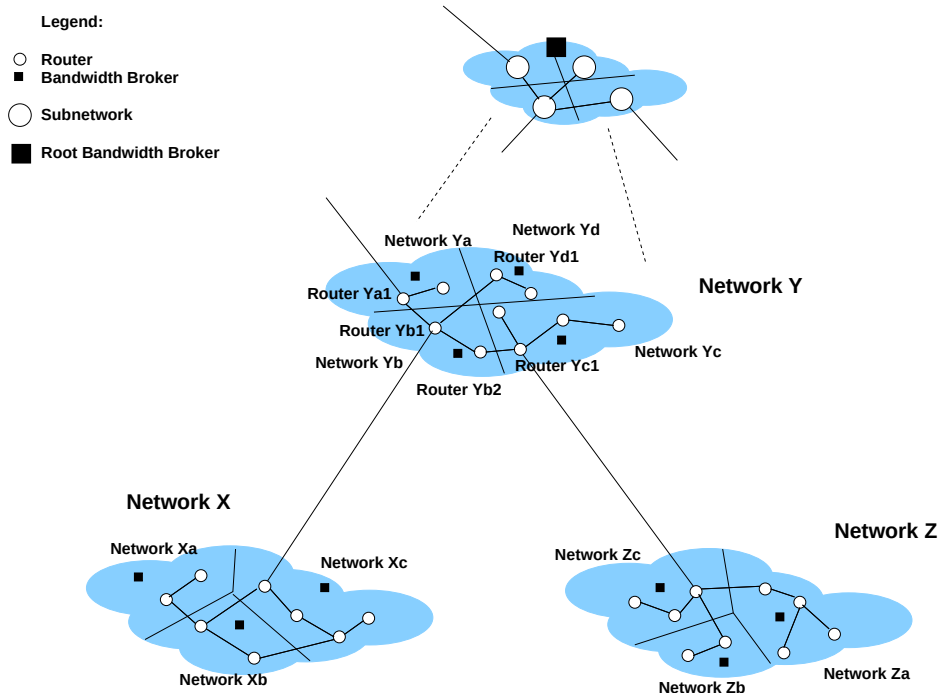

Figure 7.19: Hierarchical Bandwidth Broker Example

as much in detail, as the leaf brokers do, but it can handle this requests with topology knowledge of the subnetworks. Obviously such a network is much easier to manage than the complicated fine-grained network. Thus, the management of a network abstraction in the root bandwidth brokers has the advantage of being able to react much faster to transitional flows than a collection of peer brokers could.

7.8.2 Operation of a Broker Hierarchy

Creating Subnetworks

A large network has to be divided into smaller subnetworks, which are to be managed by a single leaf bandwidth broker each. This has to be done by a network administrator in advance. The question of how to allocate routers to subnetworks is quite difficult. It can only be solved with explicit knowledge of the surrounding situation of the network. Many parameters have to be taken into account: First, so-called “hot spots” have to be located, where many users with multiple and frequently changing QoS requirements are found. Those users will cause a big load to the bandwidth broker and thus this load has to be shared. Also the kind of the

access networks used will influence the way the network is subdivided: in wireless networks with small cells there is usually a high frequency of handovers. Therefore, many reconfigurations of edge routers may be necessary. In this situation, the grouping of some geographically *and* topologically close access networks may help to diminish the signalling overhead and improving the overall performance of the bandwidth broker hierarchy.

Initialisation of the Brokers

Each leaf bandwidth broker can initialise itself as mentioned in Section 7.1. The root broker has a different initialisation phase: since it cannot auto-detect the topology of the abstract network it has to read it from an external configuration file. Since such a partitioning of the network is assumed to be stable, this is not a drawback. The objects that form the topology database of the root broker are also not representing routers but *subnetworks*, that are, however, configured via the same configuration API like a normal router. This example again shows the outstanding flexibility of the object-oriented interface between the management layer of the bandwidth broker and the entities to be managed. Those entities usually are routers, but as we have seen, they can also be entire subnetworks.

The root bandwidth broker performs an initial reservation of bandwidth between the egress and ingress points of the subnetworks. This reservation is based on fixed contracts with neighbouring bandwidth brokers and/or given knowledge about the expected usage. This initial reservation and all following reservations on those links should be big enough to minimise the signalling overhead. That means, the Root Bandwidth Broker also acts as a client requesting a large amount of bandwidth from the brokers managing the subnetworks and distributing this bandwidth pool at its own responsibility.

For example let us assume, that in the network of Figure 7.19 all links are 100 MBit/s Ethernet links and the Root Bandwidth Broker for Network Y has the policy to initially reserve 10% of the total bandwidth for “foreign” flows. It would then initiate three flow requests: from Router Yb1 to Router Ya1, from Router Yb1 via Router Yb2 to Router Yc1 and from Router Yb1 to Router Yd1, each requesting 10 MBit/s bandwidth.

Flow Establishment

To continue the example, let us now assume, that two flow requests are sent from Network Xa to Network Yd and from Network Xa to Network Za. To simplify the example we will not specify the host names of the sender and the receiver.

The first flow request from Network Xa to Network Yd will be sent to Broker Xa. This broker detects, that the flow will leave its domain and initiates the flow establishment procedure as described in Section 7.7.4. Therefore it will forward the flow request to Broker Xb, which in turn will forward it to Broker Xc. This broker now detects, that the flow will leave its domain and thus requests the address of the responsible broker from the ingress router of the neighbouring domain (in this case Router Yb1). In contrast to the procedure presented in Section 7.7 the router does not send the address of Broker Yb but — since the request comes from outside the domain — the address of the Root Bandwidth Broker. Therefore, the flow request is sent to the Root Bandwidth Broker, which will use some of its initially allocated amount of bandwidth for this flow between Router Yb1 and Router Yd1 and finally sends the flow request to Broker Yd to set up the last part of the flow within the destination subnetwork.

The second flow request from Network Xa to Network Za will initially follow the same path. After passing Brokers Xa, Xb and Xc it will be forwarded to the Root Bandwidth Broker of Network Y. Now this broker detects, that the flow will only transit Network Y (neither source nor destination address belong to network Y), so this flow request is not forwarded to any of the leaf brokers of Network Y. The Root Broker configures the flow from Network Yb to Yc using only its preallocated bandwidth and afterwards forwards the flow request to Network Z. In this example we can see the difference and the benefit of a broker hierarchy: In a pure multi-ISP scenario without a hierarchy such a flow request would have to be handled by Brokers Yb1 and Yc1, whereas in a hierarchy scenario only the Root Broker is involved. Assuming that the subnetworks consist of several hundred nodes each, while the subnetwork topology is very simple this results in a very fast flow setup and decreased number of signalling messages compared to the multi-ISP scenario.

7.8.3 Conclusion

Our approach to organise a hierarchy of bandwidth brokers differs a lot from the ones found in the literature. Usually, the root broker manages the network in a centralistic way and the leaf brokers allocate large amounts of bandwidth they can distribute at their own responsibility [193, 134]. In our approach the control of the bandwidth fully remains at the leaf brokers. The root broker just allocates an amount of bandwidth it can distribute to reservation requests that are crossing the network. In case of a flow going from one subnetwork to another in the same “big” network, this has the advantage of not detouring the reservation request to the root broker.

Summary

In this part we presented the design and the implementation of our bandwidth broker architecture. First, in Chapter 6 the QoS management API has been presented. This API is the important interface between the management and the configuration layer in the bandwidth broker architecture. The object oriented implementation of this interface allows us to support various router hardware. In Chapter 7 the bandwidth broker architecture is explained in detail. We first presented a simple single-domain broker architecture and the associated flow reservation API. A detailled performance evaluation showed, that this architecture is able to manage large networks ($>$ 1000 nodes) at a high speed ($\sim$ 0.6 ms per Flow). Furthermore we presented the necessary extensions for a broker architecture in multi-domain networks. Several new components for this advanced architecture are discussed. In addition, modifications in the flow setup procedure have been necessary to support receiver-driven reservations and to perform the communication with other bandwidth brokers. Finally, a novel bandwidth broker hierarchy structure has been presented. This hierarchy can reduce the amount of signalling needed for a flow crossing a domain, and therefore minimise the configuration latency.

Part IV

Providing Quality of Service to Mobile Users

Overview

In this part we will further develop our bandwidth broker architecture to finally achieve the goal of being able to support the QoS demands of mobile users. This will require several enhancements in the bandwidth broker architecture itself, that are presented in Chapter 8. First the broker API has to be extended with a new command to give information about existing flows to mobile users (cf. Section 8.2). The possibility to transfer the existing reservations to a new network is prepared as well. In addition, the inter-domain broker signalling protocol has to be developed. This is done in Section 8.3.

As already mentioned before, the most important topic in QoS provisioning in mobile and wireless networks is the handover procedure. A new kind of handover is developed in Chapter 9, which allows a mobile user to pre-negotiate resource reservations with the new network. This new handover heavily relies on information about the current link quality, which is the only hint of a coming handover the mobile host can get autonomously (i.e. without the help of a bandwidth broker or without additional information, like travelling speed and geographical information about the location of base stations). In Section 9.1 we therefore start by investigating the handover procedure as it nowadays is implemented in IEEE 802.11b-compliant hardware. In several experiments in our laboratory we were able to validate the handover parameters found in technical manuals of our hardware (Section 9.2). Based on those results we implemented a monitoring daemon that controls the signal level and interacts with the client software running on the mobile host. This way it can negotiate resource reservation before the handover has to be performed due to low signal quality (Section 9.3).

The mobile node will usually try to attach to the base station offering the best signal quality. This can be done by a flow modification request to the bandwidth broker as shown in Section 7.7.4. However, relying solely on the signal quality may not give satisfying results in every case. Problems could occur if there are not enough resources available at the new access network (e.g. many other users have already reserved bandwidth). In such a case the mobile node would not be able to perform the handover and finally the connection would abort. To prevent this, we propose to introduce an additional parameter that specifies how hard the user insists on getting exactly the resources specified. This value is interpreted in percentage of how much the user is willing to degrade its service to the benefit of the handover. The bandwidth broker might then be able to perform the necessary reservations for the handover with a higher probability. This renegotiation of a existing SLS is described in Section 9.4.2.

Chapter 8

Mobile-Specific Extensions for the Bandwidth Broker

Now we will investigate the changes needed in the bandwidth broker architecture to provide Differentiated Services to a Mobile IP user. A mobile user might visit several access networks managed by different ISPs, but desires to get a certain level of Quality of Service wherever he is connected. Since the user usually has negotiated a Service Level Specification (SLS) with his home-ISP, there exist two different possibilities. Either the user has to negotiate a new SLS with each new ISP individually, or the initial SLS has to be transformed and transmitted to the foreign networks the mobile user visits. In each case the bandwidth broker managing the foreign network will then configure the network according to the SLS of the user. In addition, we assume, that the mobile host can get a valid IP address for each foreign network (e.g. via DHCP or foreign agent registration).

The first case does not imply any difficulties or changes in the architecture, since there is no difference to the negotiation of a SLS with the home broker. The second case, however, implies various extensions to the current architecture, including for example changes in the signalling protocol format and a new inter-broker protocol. Those extensions will be covered in the following sections.

The main reason to investigate this approach is to simplify the mobility for the user as much as possible. While in the first case the user has maximum flexibility to adapt to changing environmental situations (such as radio coverage, availability of bandwidth ...), each time the user attaches to a new ISP's network, a new negotiation has to be performed. This includes too long an interruption in the communication to be acceptable. The second case avoids the negotiation of a new SLS and is therefore much faster and more convenient for the user. We should, however, try to get some of the flexibility of the first approach, without the draw-

backs of frequent user interaction.

8.1 Scenario Description

Using the small network shown in Figure 8.1 as an example, we can show the major points where the reconfiguration happens when the mobile user establishes a SLS at home and afterwards migrates from one access to another. This scenario contains the home network (a home agent and a home bandwidth broker) and the foreign network (a foreign agent and a foreign BB). In addition, a correspondent host is connected to both networks. For simplicity we assume, that both, home and foreign network belong to the same management domain, that they are managed by the same bandwidth broker (e.g. running on the home agent host).

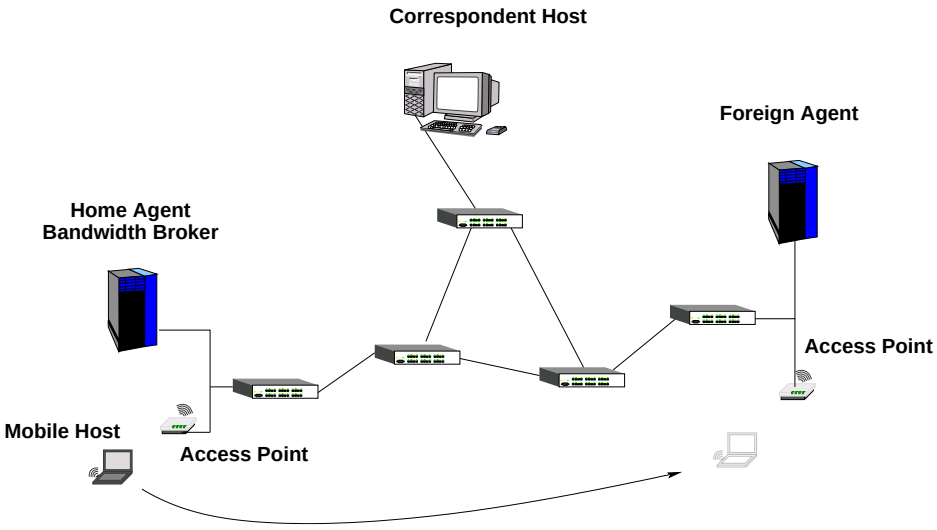

Figure 8.1: Demo Scenario for QoS Provisioning to a mobile user

8.1.1 Negotiation of a new SLS

After registering at the home agent the mobile host can send the information about the desired SLS to the home bandwidth broker. The negotiation starts when the mobile user sends a packet containing the bandwidth and some high-level information about the desired service [8] (e.g. delay-sensitivity, loss-sensitivity ...). The broker's communication interface translates this information to the internal, technical-oriented flow description of the broker and submits the result to the

bandwidth broker. The bandwidth broker tries to set up the routers according to the user's requirements and reports success or failure back via the communication interface.

8.1.2 Migration to a new access network

When the mobile host moves to a foreign domain it first has to get a care-of address (CoA) by either a foreign agent or DHCP. Using this CoA, the mobile host can now request the transfer of its home SLS to the new location. The transfer is initiated by signalling the request to the bandwidth broker in the foreign domain. The broker can then perform the authentication separately and afterwards contact the home domain's bandwidth broker to get the user's SLS. Together with the CoA of the mobile user, the foreign bandwidth broker can now establish the service in the foreign network.

Alternatively the mobile user could establish a totally new SLS with the foreign bandwidth broker without using its home SLS. The procedure is then — set aside AAA (cf. Chapter 10) issues — identical to the procedure in the Section 7.7.

8.2 Extensions to the Bandwidth Broker API

In Figure 8.2 the new bandwidth broker API is shown. The new "Mobile Support" part consists of three functions. Two of them are explained in detail in the following Sections (Sections 8.2.1 and 8.2.2), the third (`neighbouring_ESSIDs`) will be explained in the next Chapter. This function will be used for Qos-aware handovers.

Since Mobile IP is transparent to any correspondent host (with the exception of route optimisation), flows that come from the correspondent host to the mobile host will always go to the home network. This implies, that any reservation the correspondent host has set up also ends at the home network. If the mobile host moves to a foreign network and wants to get the same level of QoS in the foreign network, it also has to take care of the fact, that there may be reservations it is not aware of. Therefore we must provide a way for the mobile host to query the bandwidth broker to deliver a list of reservations that have to be changed in order to maintain the current QoS level.

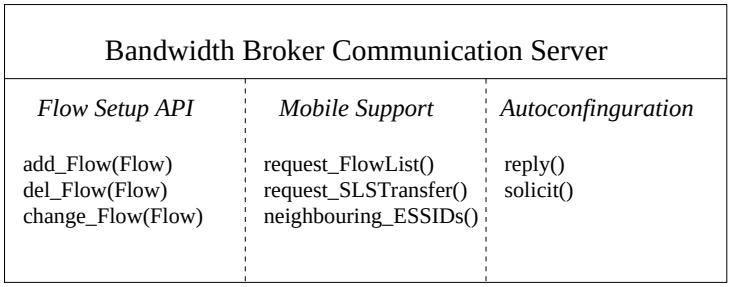

Figure 8.2: The New Bandwidth Broker Communication Server

8.2.1 Request a Flow List

The `request` function can be used to query the bandwidth broker's global flow table for flows matching an input mask. This input mask consists of a `Flow` object and compares the source-, destination- and ingress addresses and -ports. If any flow in the broker's global flow table matches this input, it is appended to the result list.

Assume, that the mobile host is attached to its home network and that there are two reservations: one from the mobile host to the correspondent host, and one from the correspondent host to the mobile host. When the mobile host moves to a foreign network it has to distinguish between the following three actions, depending on the Mobile IP routing method:

Triangular Routing: In this case the mobile host needs to establish a reservation from its care-of-address to the correspondent host and a reservation for the tunnel from the home agent to the CoA. The reservation from the home address of the mobile host to the correspondent host can be released.

Reverse Tunnelling: Here the mobile host has to establish a reservation for the tunnel from the home agent to the CoA and for the reverse tunnel from the CoA to the home agent. The reservation for the flow between the home address and the correspondent host must remain.

Route Optimisation In this case, the mobile host only needs to set up a reservation from the CoA to the correspondent host and can release the reservation from the home address to the correspondent host. Since the correspondent host will be notified that the location of the mobile host has changed, it can update the reservation itself.

Of course this can easily be automated and is executed by a daemon monitoring the change in the CoA and performing the necessary actions.

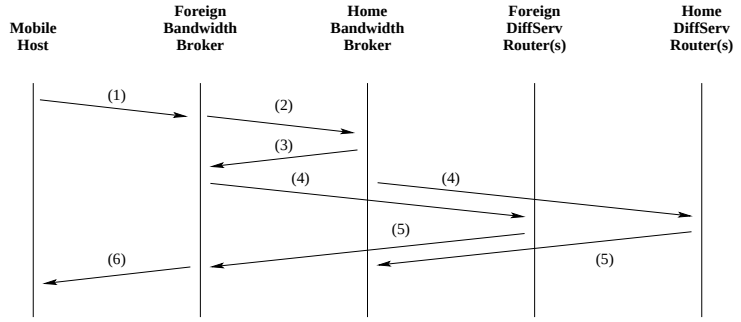

Figure 8.3: Message sequence for SLS transfer to a foreign network

8.2.2 Automatic SLS Transfer

If the mobile user connects to a foreign domain, the SLS of its home domain has to be transferred to the foreign network. The mobile host can check if it is connected to a foreign network by checking for a care-of address. The message sequence for this case is shown in Figure 8.3.

As mentioned before, the protocol shown in Figure 7.6 can also be used, for example to change the home-SLS to adapt it to the new environment.

1. The mobile host requests the foreign bandwidth broker to transfer its home SLS to the new location. Therefore, a special packet format is used, including the home address of the mobile host.

2. The foreign broker asks the home broker for the SLS of the mobile host. It has to use the home address of the mobile host for the query.

3. The home broker transmits the SLS to the foreign broker using the packet format shown in figure 6.4.

4. The foreign broker replaces the home address of the mobile node with the care-of address and configures the routers in its network. The home broker reconfigures the routers in the home network to release the resources used by the mobile user.

5. The routers report success or failure of the configuration back to the bandwidth brokers.

6. The foreign broker informs the mobile host about success or failure of the SLS transfer.

8.3 Inter-Domain Broker Signalling

A second, more complex scenario is presented in Figure 8.4. For this scenario the bandwidth brokers in the home and the foreign networks also need to contact the bandwidth broker in the correspondent host's network, because some of the routers are not in the domain of the home broker. In addition to configure the routers in their own domains, the home and foreign brokers must signal the correspondent hosts's broker the modified flow containing the egress router's address. The bandwidth broker can determine this address by tracing its topology database (see [160]). It is important to signal the egress router's address, because the broker in the correspondent host's network has to be able to determine where the new flow enters its network. Since a bandwidth broker usually only knows the topology of its own network and additionally the addresses of the neighbouring egress/ingress routers, this is the only way to set up the flow between two adjacent domains correctly. The packet format for this message can be the same as in the first scenario (Figure 6.4). This fact extremely simplifies the broker signalling protocol.

If the mobile host roams toward the foreign domain, the reservation towards the home domain has to be deleted and a new reservation towards the foreign domain has to be established.

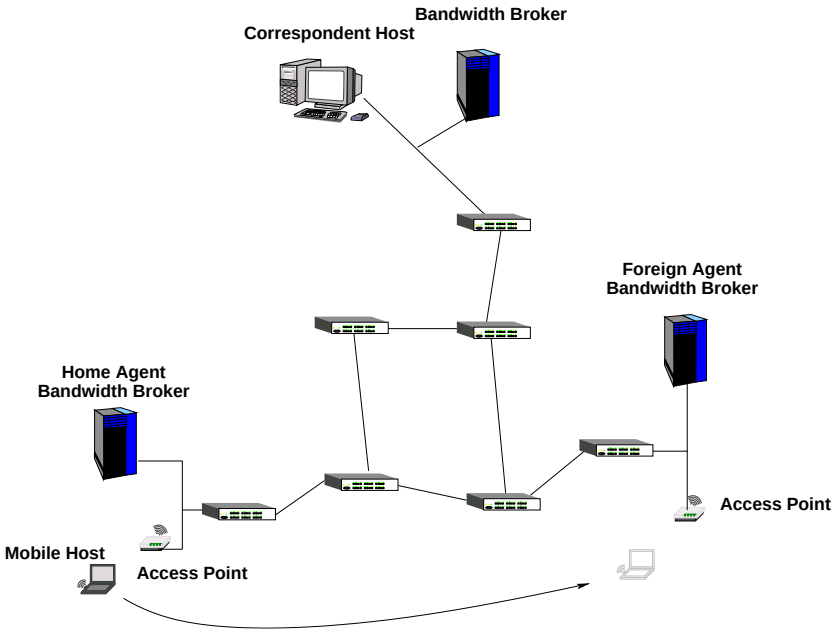

Figure 8.4: A scenario for inter-domain broker signalling

Chapter 9

User-initiated Handover

In this chapter we will present a way for the mobile user to gain control of the handover. This will enable the mobile user to take advantage of negotiating QoS parameters prior to the handover. Using the extensions to the bandwidth broker presented in Chapter 8, the mobile user will be able to receive Quality of Service with minimal interruption.

For the development of this new "User-initiated Handover", we assume, that the mobile node uses some kind of wireless LAN access technology. The wireless LAN technology has been in use for a long time now (over 10 years) and mature, standardized products are available. Due to the implementation in hardware, the link layer handover process concludes very quickly. However, most manufacturers do not provide enough interface control to their products, making it difficult to adjust or improve the wireless LAN hardware's behavior. One of the major concerns is, that while the wireless LAN technology and Mobile IP operate at different layers (the first at the physical and the link layer, the latter at the network layer), there is no definition for the inter layer communication between link and network layer. In practice, this is reflected in the Mobile IP implementations just following the behavior of the wireless LAN hardware. Of course, this leads to delays or unwanted and unpredictable results. Yet for applications relying on QoS in particular, the well timed transfer of network management information (i.e. flow descriptors) to the new point of attachment is very important. Additionally, the setup of new reservations (e.g. for the tunnel between the home and foreign agent) will take some time. Therefore, a pre-handover reservation setup will reduce the service interruption due to a handover, thus improving the overall service quality an access network offers.

However, to be able to precisely predict the time a handover is going to happen, we have to get control over the link layer handover process. The most obvious

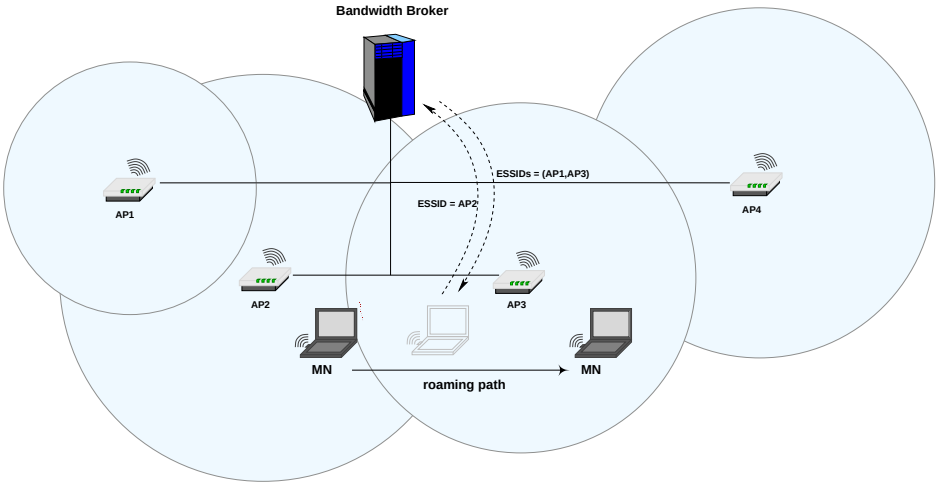

Figure 9.1: Requesting Neighbour Wireless Cells Before the Handover

parameters that influence the wireless LAN hardware's behavior are indicators of the signal quality. These quality parameters are usually provided by the driver of the wireless LAN hardware and can be gathered easily. This leads to the main idea of continuously monitor the signal quality at the application layer and, upon exceeding or falling below certain thresholds, alert concerned applications.

The mobile node may now ask the bandwidth broker for a list of neighbouring access points (see Figure 9.1): The mobile node passes its current ESSID to the broker (AP2). The bandwidth broker having a list of all access points in its network and their locations available can in turn search for access points that are likely to be in range of the mobile host (that is AP1 and AP3) and pass this list of ESSIDs to the mobile node. The mobile node can now perform a search for wireless cells offering a better radio transmission quality. It will restrict its search to the list of ESSIDs it has received from the bandwidth broker. In Figure 9.1 the mobile node will only find AP3. Now the mobile node can ask the bandwidth broker if any of the wireless cells it found offers enough network resources to satisfy the SLS of the mobile user. If there are several access points available, the mobile node will choose the one offering the best signal quality.

The implementation [181] requires access to the wireless LAN hardware, a specification of the thresholds (Section 9.2) as well as a communication interface to other programs. With these abilities a mobile node is in control over the handover process and can perform fast handovers.

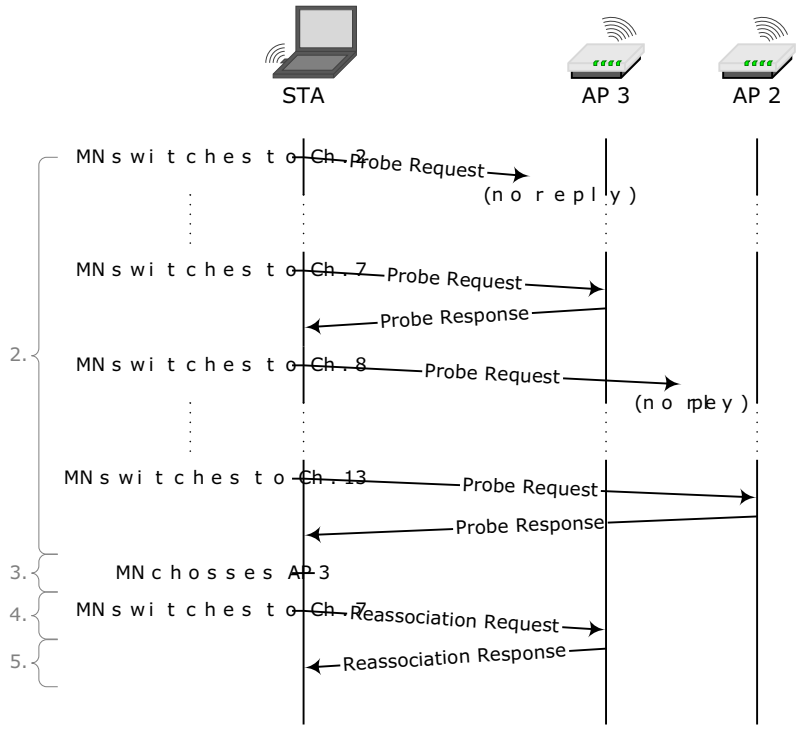

Figure 9.2: Link layer handover message-time diagram using active scanning

9.1 Handover in IEEE 802.11 Wireless Access Networks

WaveLAN is a commercial wireless LAN implementation from Lucent Technologies conforming to the IEEE 802.11 standard.

The following description of the link layer handover process is based on the WaveLAN implementation (see [108]).

The wireless LAN card permanently monitors the signal quality of its current link to the access point. These values are compared reiteratively with predefined thresholds and lead to appropriate actions.

9.1.1 Link layer handover procedure

When a station moves to the border of the coverage area (so-called wireless cell) of its current access point into the coverage area of another access point (see Figure 9.1), the signal quality of the current link drops and a process called *Handover*

is invoked. It guarantees the seamless transition between different wireless cells up to the link layer.

These steps describe the link layer handover procedure shown in Figure 9.2 in detail:

1. A station decides (see Section 9.1.3) that the signal quality to the current access point is poor.
2. It starts to look for other access points with the same ESSID using a sweep.

   (A sweep characterizes a series of scans on different channels. These are maintained by the station in a channel-list).
3. The station then selects the best access point found by evaluating the signal quality.
4. It sends a reassociation request to the selected access point. The access point determines if access can be granted.
5. If the station is allowed to access the access point, the access point sends a reassociation response.

9.1.2 Measurement categories

When monitoring the signal quality, the *Signal-to-Noise Ratio* (SNR) is of substantial interest. It is based on the signal level and the noise level. The signal level is obtained from the beacon messages sent by all access points at a rate of ten messages per second. Information about the noise level is taken from the data traffic the station is engaged in.

9.1.3 Decision points

A well timed identification of the station's movements is achieved by introducing several decision points (thresholds) based on the SNR values (see Section 9.1.2).

Table 9.1 lists the thresholds depending on the density of installed access points. The access point density value specifies the distance between access points in the wireless network. These values are taken from Lucent's Technical Bulletin 023/B [109].

Carrier Detect threshold: A signal level value in dBm representing a lower limit. A station will only accept received signals that exceed the Carrier Detect threshold.

Threshold	AP Density		
	Low	Medium	High
Carrier Detect [dBm]	-95	-90	-85
Defer [dBm]	-95	-85	-75
Cell Search [dB]	10	23	30
Out of Range [dB]	2	7	12
Delta SNR [dB]	6	7	8

Table 9.1: WaveLAN/IEEE thresholds

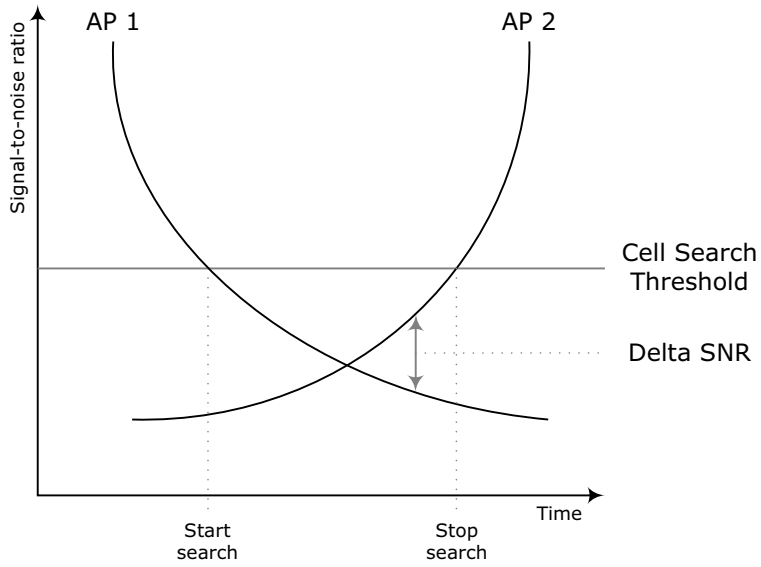

Figure 9.3: Link layer handover SNR-time diagram

Defer threshold: A signal level value in dBm marking a upper bound. A station will delay its own signal to be transmitted until the incoming signal falls below the Defer threshold, thereby not being recognized as a modem signal anymore.

Cell Search threshold: A SNR value in dB used as lower bound. A station will start to look for another access point as soon as the SNR falls below the Cell Search threshold.

Out of Range threshold: A SNR value in dB describing a lower limit. A station will be unable to receive a signal properly if the signal's SNR falls below the Out of Range threshold.

Delta SNR: A SNR value in dB representing a minimal distance. A station will only change to another access point if the difference of both, the old and the new access point's SNRs exceed the Delta SNR.

Figure 9.3 depicts the thresholds impact in the scenario seen in Figure 9.1.

Over time the station shifts from the Wireless Cell 1 to the Wireless Cell 2. Thereby it removes from the first access point and thus the SNR decreases more and more. Soon the SNR falls below the *Cell Search* threshold, triggering the scanning functions of the wireless LAN card. When it recognizes the second access point, the wireless LAN card does not connect instantly to the new access point. Instead it waits for the difference between the SNR from AP1 and AP2 to exceed the *Delta SNR*. After having changed to AP2, the wireless LAN card stays in the search state until the SNR passes the Cell Search threshold again.

If the station moves to an area without any access point coverage, the SNR falls below the *Out of Range* threshold, causing more intense scans. Another reason for not finding any access points might be an overloaded network.

9.2 Link Layer Handover Statistics

Now we want to determine and validate the Signal-to-Noise Ratio thresholds used by wireless PCMCIA cards to initiate a layer 2 handover as described in Section 9.1.1. As no standard thresholds have been prescribed each manufacturer uses its own values. The threshold values for different density settings shown in Table 9.1 are taken from Lucent's Technical Bulletin 023/B [109].

The wireless equipment used in the tests is manufactured by Lucent Technologies' Microelectronics Group [1] and available documentation allowed the verification of the obtained results.

The AP Manager [3] provides a limited control over the threshold values to be used. Three length values (large, medium or small) in the *Distance between APs* field describe the AP density (low, medium or high) of the wireless network.

9.2.1 Equipment

The test equipment consists of two access points (Lucent WavePOINT-II V3.83) and a Laptop acting as the mobile node, all using WaveLAN cards (Lucent WaveLAN/IEEE Turbo, Firmware 7.52).

A small program called *iwstats* has been written to gather the signal's quality values. These consist of the signal level, noise level and the resulting Signal-to-Noise ratio (SNR). It runs on the mobile node and logs the accumulated data. Iwstats is based on the Wireless Tools [175] and adds some functionality to let the user specify the length of the gathering period and the interval between the collection of two records. Additionally the output is formated to allow easier evaluation of the data with gnuplot.

The channel setup for the access points provides sufficient channel separation between two access points as described in [107].

The public access to the wireless test network has been disabled using the *Close Wireless System* option in the AP Manager. This feature is non-compliant to the 802.11 standard and implements a stricter handling of association requests from stations. More information about WaveLAN security options can be found in [110].

Furthermore, all involved access points and the mobile node have been configured to use the same ESSID. When the Close Wireless System option is activated, a link layer handover can only be carried out between access points and mobile nodes using the same ESSID. The mobile node can also be configured to connect to any access point by omitting the ESSID specification (i.e. using the value ANY instead) and disabling the Close Wireless System option in the concerned access points.

[1] now called Agere Systems

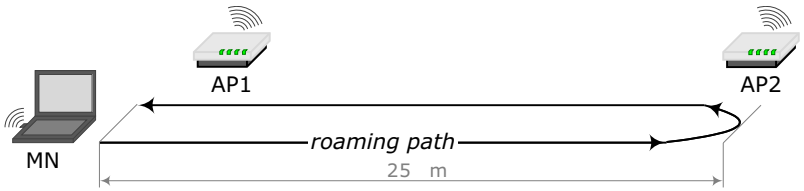

Figure 9.4: Outline of the roaming path

9.2.2 Environment

The test site is located under the roof of a brick wall building. The access points have been placed in two separate rooms, connected by a long hall. The first room is used as standard office, the second as test lab and contains therefore lots of technical equipment. The hall in between these two rooms is almost empty.

9.2.3 Measurements

In accordance with the available access point density settings three series of measurements have been carried out, each consisting of 20 runs. The signal's quality parameters have been gathered at 300 ms intervals.

The roaming took place along the path shown in Figure 9.4. Commencing one meter before the first access point (AP1) the mobile node moved in front of the second access point (AP2) and from there - following the same path - back to the starting point. With a walking speed of approximately 3 km/h the distance of 50 meters was covered in about 60 seconds.

The averaged results are shown in Figure 9.5, 9.6 and 9.7. They illustrate the performance of the signal's quality parameters (signal level, noise Level and SNR) over time.

9.2.4 Analysis

While moving away from the AP1 the SNR decreases more and more until it reaches the Cell Search threshold. Now the layer 2 handover process (see Section 9.1.1) starts and as result the mobile node associates with the AP2 and thereby connects to the foreign link. When returning back to the home link, the analogue procedure takes place.

The altering of the handover points throughout the three graphs is due to the changes of the access point density parameter and - at the same time - keeping

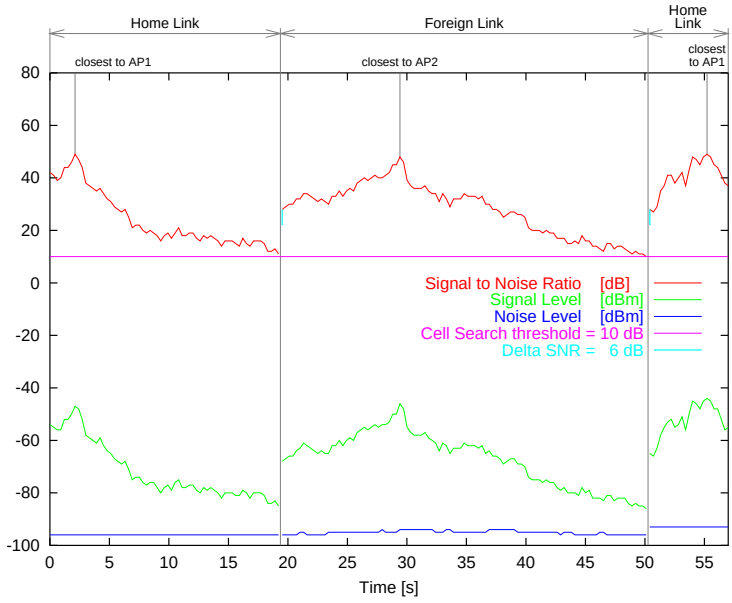

Figure 9.5: Signal to noise ratio when roaming with low AP density

the experimental setup physically untouched.

Any fluctuations (e.g. local peaks) in the graphs can be most often ascribed to surrounding objects made of interfering material.

The graphs show that the thresholds listed in Table 9.1 on page 163 are accurate values and will therefore be included in the development process of the mobile-controlled handover (see Chapter 9.3).

9.3 Implementation and Operation of the WLAN Monitor

[Implementation and Operation of the WLAN Monitor] The necessary functionality to keep track of the SNR behaviour and alert the user to perform the necessary pre-handover actions a Wireless-LAN monitoring program has been developed and implemented [181]. Here we will only give a short overview of the operation of this program.

The implementation was done in C++ and consists of two seperate daemons: The first provides the data transmission from the bandwidth broker's mobile client to the Wireless LAN Monitor, the latter is responsible for the data transmission

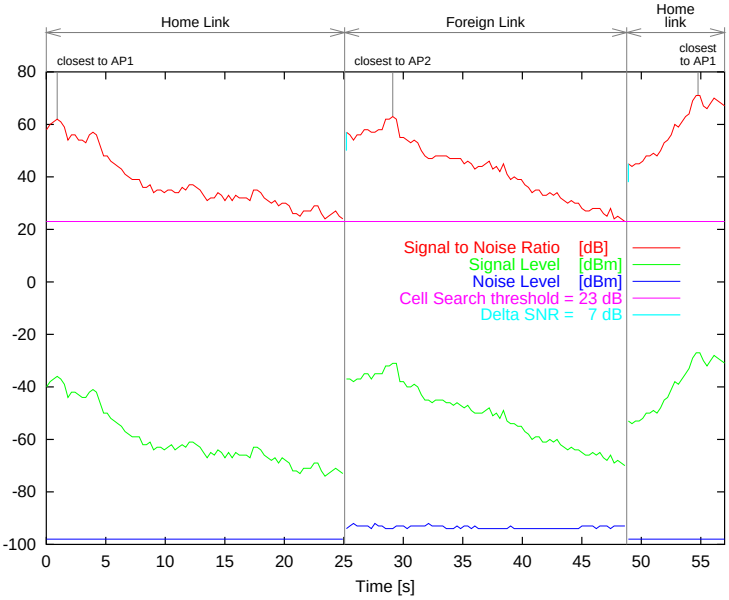

Figure 9.6: Signal to noise ratio when roaming with medium AP density

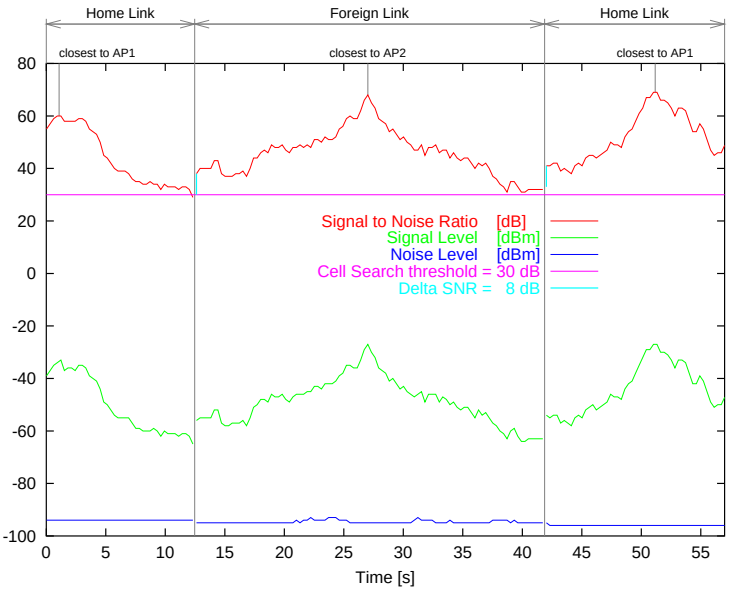

Figure 9.7: Signal to noise ratio when roaming with high AP density

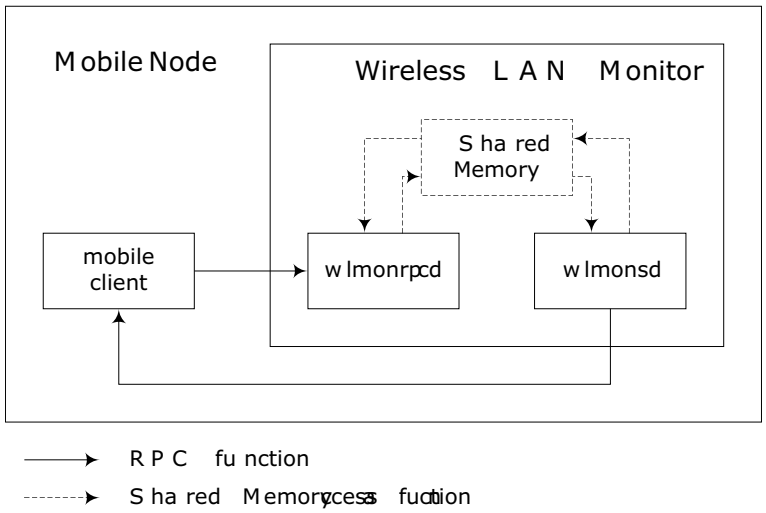

Figure 9.8: Interaction of the Wireless LAN Monitor components

to bandwidth broker's mobile client and for the monitoring the signal's quality parameters of the mobile node, where all three programs are running on (see Figure 9.8). For interprocess communication between the different programms RPC calls as well as shared memory has been used.

9.3.1 Wireless LAN Monitor operation

The cycle of the Wireless LAN Monitor's components interacting with the mobile client of the bandwidth broker are described with the following steps and illustrated in Figure 9.9.

To synchronize the shared memory access of the wlmonsd and the wlmonrpcd the first character in the shared memory field has been turned into a status flag.

1. When the Wireless LAN Monitor Service Daemon (SD) is started, it invokes the Wireless LAN Monitor Remote Procedure Call Daemon (RPCD).

2. The bandwidth broker's mobile client (BBMC) passes an ESSID list to the RPCD.

3. After reception, the RPCD requests the transfer of the list to the SD.

4. The SD acknowledges the request and hereon the RPCD places the list into the shared memory and confirms the transfer's completion.

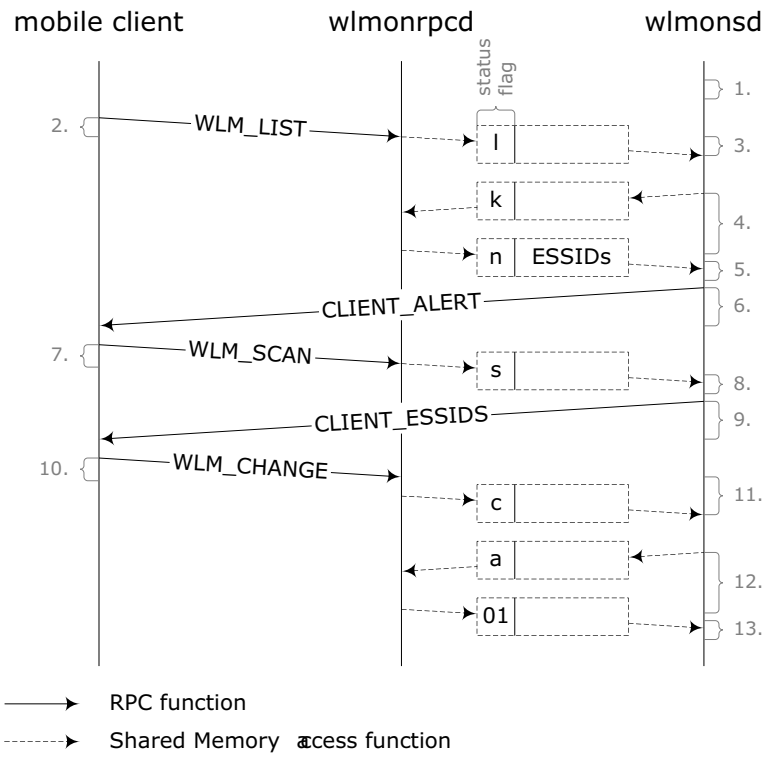

Figure 9.9: Wireless LAN Monitor procedure message-time diagram

5. Having awaited the confirmation, the SD now reads the list from the shared memory.

6. Assuming that the signal quality falls below the predefined threshold (e.g. because the mobile node may have moved), the SD alerts the BBMC.

7. Next the BBMC sends an order to scan for better serving access points to the RPCD, which in turn places a specific mark in the shared memory to be read by the SD.

8. The SD becomes aware of the request mark and starts to scan other wireless cells identified by their ESSID from the list.

9. After finishing the scans, the SD informs the BBMC about the results by passing a list of indices. Sorted descending by signal quality, each index represents an ESSID from the original list submitted by the BBMC.

10. The BBMC sends an order to change the wireless cell with a certain delay to the RPCD.

11. After having waited for the specified delay, the RPCD announces the requests to change to the SD.

12. When the SD acknowledges the request, the RPCD puts the index, to which the SD should change to, as unsigned character in the shared memory.

13. The SD reads the index and carries out the change of the ESSID.

9.4 Range-Based Bandwidth Allocation

We have introduced a mechanism that allows us to initiate a pre-handover SLS negotiation in order to achieve continuous QoS support in wireless access networks. The current implementation allows us to select an access point out of a list the bandwidth broker provides. We now want to present certain criteria to ensure the best possible QoS for a mobile user roaming several wireless access networks.

The scenario we want to investigate consists of a mobile host that would like to communicate while moving through the coverage areas of several wireless access points (AP). Each of those access points offers different QoS levels. Given the initial QoS requirements of the mobile user, a decision to which wireless cell the mobile host connects has to be made. If the mobile user has fixed QoS requirements that must be kept, the mobile host may not be able to change to a wireless

cell offering a better SNR. Perhaps, if the signal quality is too low for communication, no access point at all can serve the user's requirements. If, however, the user allows a certain deviation of the QoS specifications, we may be able to choose the best access point from a bigger list of possible candidates.

9.4.1 Scenario

As in Section 9 we assume, that all wireless cells have a unique identification number (the ESSID). Figure 9.10 shows the scenario more in detail: A mobile user reserves a flow requiring 2.5 MBit/s bandwidth at the bandwidth broker. Since the first wireless cell provides a maximum bandwidth of 5.5 MBit/s, this flow can be set up. Afterwards the mobile user moves over to the second and third wireless cell, providing 11 MBit/s and 2 MBit/s respectively. Performing the first handover is easy, since more bandwidth than required is available. If, however the mobile user moves to its final location, the service initially granted cannot be satisfied any more. Section 9.4.2 will cover this problem more in detail.

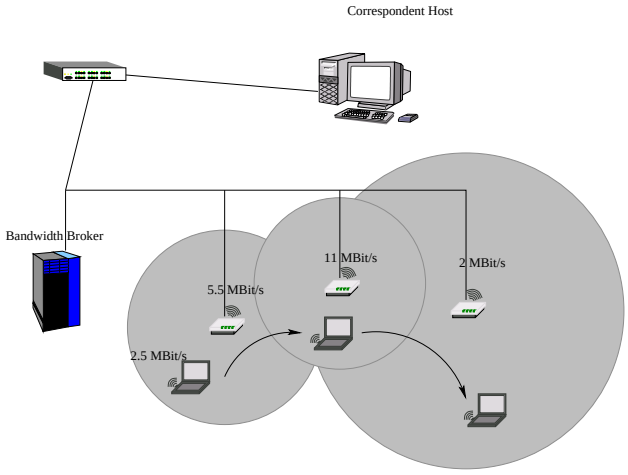

Figure 9.10: Scenario for a QoS-aware Handover

9.4.2 Renegotiation of the SLS

Each time, a handover occurs, the bandwidth broker has to guarantee, that the new wireless cell is able to fulfil the requirements of the new user. If no wireless cell capable to serve the new flow is in range, a renegotiation of the SLS has to be performed in order to maintain the connectivity of the user. Otherwise, the

mobile user would not be able to perform a handover and finally move out of the transmission range of the base station, unable to communicate any more. To avoid this, during the pre-handover negotiation (see Section 9) the BB will also propose some wireless cells, that have not enough resources to serve the flow. The mobile host can now choose the best wireless cell available to attach. This choice may consider parameters outside the scope of networking, for example geographical information.

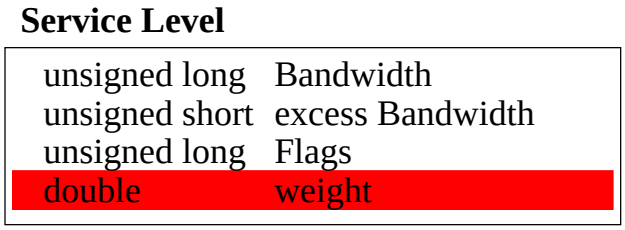

Figure 9.11: New Service Level Specification Format

The decision of the bandwidth broker which cells to propose is based on an additional entry in the SLS description: the weight field (see Figure 9.11). This newly introduced value will be interpreted as a percentage value, how far the mobile user is willing to deviate from its initial service level specification. If for example the value of weight equals 0.2, any wireless cell offering more than 80% of the specified bandwidth will be proposed. Given the values of Figure 9.10 and a weight of 0.2 the third cell could just be proposed to the mobile host, thus providing connectivity at the mobile host's final destination.

Chapter 10

An AAA Architecture Extension for Providing DiffServ to Mobile IP Users

Throughout the discussion of our bandwidth broker architecture and the extensions for mobile users (Chapters 6 – 8) we neglected the topic of security. In a mobile scenario, however, this topic gets more and more important. Usually, an ISP will not offer its resources to a foreign user for free. Therefore, accounting mechanisms have to be implemented if the support for mobile users is desired. To prevent abuse and fraud, this automatically leads to authentication and authorisation.

In this chapter we will discuss, how an AAA architecture, like the ones presented in Section 3.7, could be modified or extended to fit the requirements of a mobile user requesting Quality of Service in a foreign domain.

In Section 10.1 we present a simple architecture and message sequence for AAA in a Mobile IP scenario using a foreign agent. This architecture is derived from the roaming pull model for distributed services [179]. In Section 10.2 we discuss, that it is necessary to depend the reception of a care-of address on valid authentication and authorisation. A new architecture is developed by combining the Service Location Protocol (SLP) with the traditional AAA architecture. Of course in such a scenario the access to the bandwidth broker has to be restricted, too. This is discussed in Section 10.3.2.

10.1 Mobile AAA with Foreign Agent

We now apply the roaming pull model [179] to a Mobile IP scenario which is one of the main usages of the roaming pull model. Furthermore we specify the messages that are exchanged between the mobile host and the entities of the AAA architecture. After a mobile user arrives in a foreign domain, the following messages have to be exchanged in order to provide connectivity to the user:

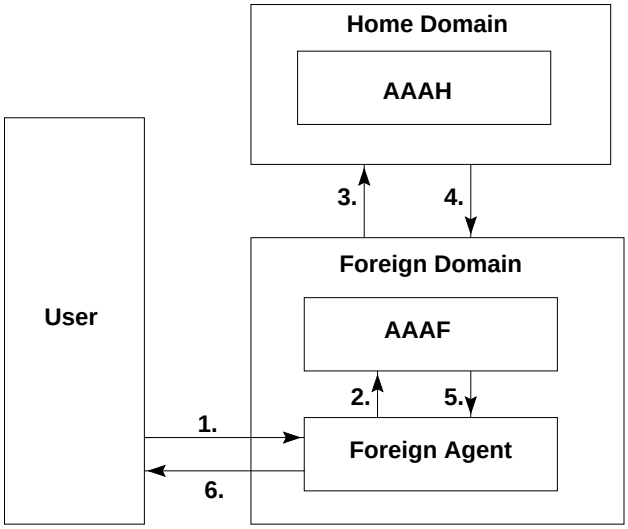

Figure 10.1: AAA Message Sequence for a Mobile Node

1. The user visiting a foreign network wants to use a certain level of QoS. Therefore it requests a quantifiable amount of resources between a selected destination and itself. First, he/she needs to issue a registration request to the foreign agent, including the authentication information. At this point the mobile node still has not yet gained access to the network, as it still doesn't have a valid care-of-address. Thus it can not send the requests to the home AAA server directly.

2. The foreign agent parses this request and forwards the authentication information to the foreign service provider's AAA server (AAAF).

3. By receiving the registration of the mobile node, the AAAF checks the realm part of the NAI provided by the mobile node to see whether the mobile node belongs to its own network. The Network Access Identifier (NAI) Extension [27] is the user ID submitted by the client during authentication and has the format of username@realm. The purpose of the NAI is

to uniquely identify the mobile node. Usually the authentication information of a mobile user cannot not be validated locally. Therefore the AAAF needs to contact the appropriate external authority (AAAH) to evaluate the request. This AAAH (the AAA server of the home domain) is found with help of the NAI.

4. The AAAH looks up the corresponding policy in its SLA repository based on the user name and forwards it to the AAAF for evaluation.

5. Once the authorisation has been obtained by the AAAF, it decides whether to accept a user with the specific parameters or not. The AAAF will notify the foreign agent about the result.

6. After a successful authorisation of the mobile node, the service equipment has to set up a policy enforcement and to tell the user that the required service is available. Now, the foreign agent is able to continue the Mobile IP registration procedure without further involvement of the AAA servers.

10.2 Integration of SLP and the AAA Architecture

The main drawback in the Mobile IP AAA architecture described above is that it depends on the existence of a foreign agent: A foreign agent is not always available in foreign network environments. Sometimes the mobile node uses some other automatic configuration mechanisms to get a new IP address such as IPv6 stateless address autoconfiguration [173] or DHCP. However, valid IP addresses are also network resources. If any mobile node is allowed to get an IP address by automatic address configuration, it can do anything at will. This has a tremendous impact on the network security. Therefore we need to make specific restrictions on methods to obtain an IP address. To do so, it is necessary to further develop an AAA architecture which is working in a uniform manner for IPv4/IPv6 no matter whether the foreign agent exists or not. In order to solve these problems we integrate the Service Location Protocol (SLP) [69] into our architecture.

SLP simplifies the discovery and selection of network services such as printers etc. There are three components involved in SLP: User Agents (UA) acquire service handlers for user applications; Service Agents (SA) advertise service handlers; Directory Agents (DA) collect service handlers in a local network. The signalling procedures are illustrated in Figure 10.2.

1. The DA periodically multicasts DA advertisements to indicate its presence to all SAs and UAs.

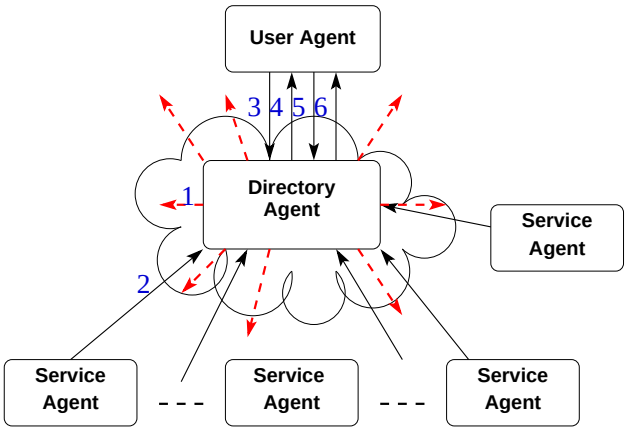

Figure 10.2: Message Sequence in SLP

2. The SAs advertise themselves by registering at a DA. The registration information includes a list of all the keywords and attribute-value pairs that describe their service. Registration information also includes a time-to-live after which the service information is removed by the DA. Explicit deregistration can also remove service information. The DA has to return an acknowledgement or a deregistration message on receipt of a registration.

3. Whenever a client application requests a type of service, the user agent will send an attribute request to the DA to find out the characteristics of a particular service.

4. The DA sends an attribute reply which gives a list of available services matching the requested information.

5. The client chooses a service out of this list and the UA sends a service request to notify the DA of its choice and acquires a service handler (i.e., service addresses and access information).

6. The DA sends a service reply to the user agent to provide the service handler.

Finally, the client application can communicate directly with the SA, the DA's assistance is no longer needed. In order to make the Mobile IP AAA architecture independent of the foreign agent, we further make an extension by deploying SLP as shown in Figure 10.3. A mobile node can search any service agent which is available in a foreign domain (such as foreign agent, DHCP server, printer etc.) via SLP according to the user's requirements.

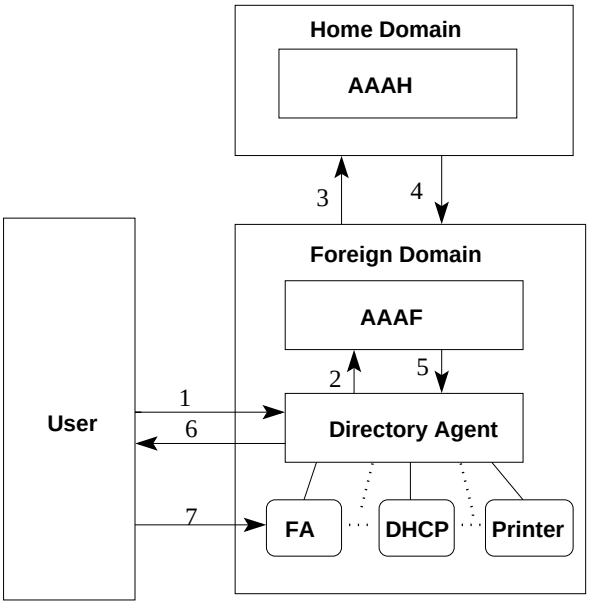

Figure 10.3: Message Sequence in SLP capable AAA Architecture

1. The foreign agent and the DHCP advertise themselves by registering at a Directory Agent, which periodically multicasts DA advertisements. Note that it is required in advance to disable the stateless autoconfiguration in IPv6 routers. Once the mobile node receives a DA advertisement, it will send an attribute request to obtain an IP address, which includes the authentication information.

2. The DA acts like a guard having the responsibility of authentication. In particular, it has to make sure that only valid mobile users can freely use resources or services and leave the other malicious users outside. Before the DA answers the query of the mobile user, it first needs to check whether the identity of the mobile user is valid. Then, the authentication phase begins. In order to accelerate this procedure, the DA needs to have a database with the valid visitors. This information can be dynamically obtained via the AAA service. By receiving an attribute request the DA checks its user database to see if there has been a record of this user according to the username part of the NAI provided by the mobile node. If it is not available, the DA is expected to forward authentication and NAI information to the local AAA server.

3. The AAAF first checks the realm part of the NAI to see whether the mobile user belongs to its own network. If the user is a visitor, it contacts the

external AAAH to further verify the user identity.

4. QoS specifications are typically located in the home AAA server, which may be indexed by username, password etc. Therefore, the home AAA server checks the validity of the user identity based on the confidential information, then gives a proper response to the foreign AAA server. Of course, it needs to forward the SLA policy in its positive reply to the foreign domain in order to facilitate the later authorisation, to minimise latency, and to avoid too frequent control message exchange if the mobile user micro-moves between different subnets in the same ISP domain.

5. If the foreign AAA server receives a positive reply from the home AAA server, it will store the SLA specification to establish a customer record in its database. Meanwhile it will inform the DA about the authentication result.

6. As soon as the DA is informed about the user validity, it will add the user information to its user records database. If this user wants to re-use the various network services at a later time again, the DA can then assure the identity by directly checking the database and no time-consuming authentication is needed anymore. Now, the DA can send an attribute reply that gives a list of the available services able to allocate an IP address, such as a foreign agent or DHCP.

7. The client agent of a mobile computer chooses automatically or manually one service, gets a service handler from the DA and then contacts a foreign agent or DHCP to get an IP address. Each SA has to contact the AAAF to further authenticate the user's identity for more security. Finally, the mobile node will issue a registration request to require access to the network.

10.3 Mobile IP Node Negotiation Procedure for DiffServ

In order to achieve a complete impression about the way our extended AAA architecture works in detail and how various components interact with each other to establish Differentiated Services for a Mobile IP node, this section will give a more detailed description of the message sequence shown in Figure 10.4.

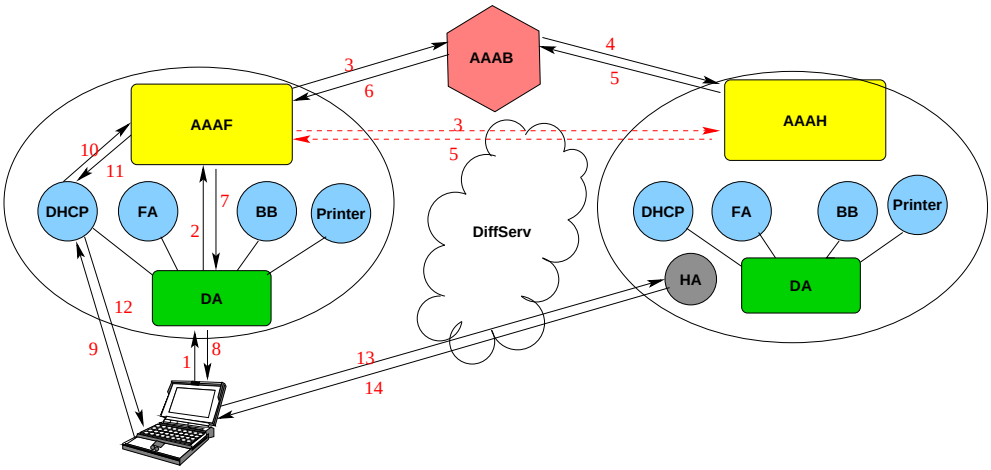

Figure 10.4: Message Sequence in Negotiation Procedure

10.3.1 Initial Foreign Network Access

1. In order to obtain a temporary connectivity, IP address, subnet mask, default router, DNS server, and other informations are required. As soon as a mobile node receives a DA advertisement, it will send an attribute request to obtain an IP address. Because the foreign ISP needs to assure that he/she will pay for the connectivity, the user has to send some confidential information such as username, password, ID etc. to identify himself/herself. This information should be encrypted with the AAAH's public key and appended as an option extension to every attribute request message sent to the DA (Figure 10.5). These authentication information tends to be valid for a long period, is difficult to forge, and has a strong authentication process to establish the owner's identity. It can be considered as a passport to identify the owner. Meanwhile, in order to map the home domain and facilitate the later distribution of the shared key between AAAF and mobile node, the mobile node's NAI extension and its public key also need to be included.

Type=1111	Length	Security Parameter Index
Confidential Information		

Figure 10.5: An Authentication Option Extension in an Attribute Request Message

2. The DA checks its user database to see whether there is a record of this

user according to the username part of NAI. If it is not available, the DA is expected to forward the authentication data, NAI and mobile node's public key information to the local AAA server.

3. The AAAF also needs to have the two following tables in addition to the SLA repository with the native customers.

   - A visiting record table with all mobile nodes who are presently visiting this foreign domain. Each item in this table has to contain the following information: *the username part of the NAI, account number and shared key which is produced by the AAAF for the valid user, mobile node's public key and the SLA specifications*. The item has a lifetime that is setup appropriately by the network operator.
   - The other table is a security association table with all foreign domains this AAAF has established security associations with. Each item in this table has to contain the following information: *foreign domain name, IP address of AAAF, IP address of AAAH, shared key*.

   Usually, if a mobile user moves into a new domain, the AAA has no corresponding record to verify him/her. Therefore, it will contact the appropriate external AAA server (AAAH). By reading the realm portion of the NAI, the AAAF can determine if and where to the information should be forwarded. The basic operation is as follows:

   - The AAAF first checks the realm part of the NAI to see whether the mobile user belongs to its own network or not. This is in case the mobile user micro-moves around in different subnets of the same administrative domain. If the user is a native customer, it will directly decrypt the authentication option with its own private key and check the validity of the user in its SLA repository.
   - If the mobile user is a visitor, the AAA has to check its visiting record table according to the username of the NAI in order to see whether a record about this mobile user already exists or not. If his/her record is available, this user has been authenticated before and he/she is accepted as is a valid user.
   - If the record is not available, the mobile user is a newcomer. The AAAF then needs to further check the security association table to see if a security association between the local AAA server and the AAA server of the home domain indicated by the NAI of the mobile user exists already.

- If so, the AAA server directly sends the authentication option extension of the IP packet to the AAA server in the home domain (AAAH). The AAAH uses its private key to decrypt the authentication information (e.g. user name and password) and checks the validity of the user identity in its SLA repository. For valid users, the AAAH returns a copy of the SLA specification which needs to be encrypted with the shared keys between the AAAF and the AAAH. Here, the security association is assumed to be a trust relationship by which the AAA server in the foreign domain can make sure the AAAH will definitely pay for the services used by the mobile users who belong to it.
- Otherwise, support from the AAAB is required. If the AAAF has an interface to the AAAB, it can send the authentication option extension and NAI extension of the IP packet to the AAAB.

4. The AAAB checks the realm part of the NAI to see if it is able to map this domain name into an IP address of an AAA server. If it is not possible, the AAA broker has to reject the service to the mobile node in the foreign network by giving a negative response to the AAAF. Otherwise, the AAA broker needs to send an inquiry message including this authentication option extension to the AAAH in order to require a copy of the authorisation message from the home domain.

5. The AAAH is responsible for storing all user names and SLA specifications about the mobile users who belong to this home domain. If the AAAH receives the inquiry message from the AAAB, it will decrypt the authentication with its private key and look up its SLA repository. This database not only contains the user's identification but also specifies the maximum amount and type of traffic the user can send and/or receive. Finally, the AAAH checks the security association table, uses a proper key (AAAB's public key or the shared-key between the AAAH and the AAAF) to encrypt the SLA information and sends the result to the correspondent node (the AAAF or the AAAB). Of course it here refers to the AAA broker.

6. The AAA Broker will decrypt the received message and send a packet to the AAAF. It encrypts the result with its private key and adds it to the IP packet as IP Authentication Header [7]. This packet is more like an entry visa, because it is typically issued by a different authority than the passport issuing authority and it does not have as long a validity period as a passport. The structure of this packet is a digitally signed set of attributes defining the DiffServ service level of the mobile user when at home.

7. If the AAAF receives a positive reply from the external AAA server (AAAH

or AAAB), it will decrypt the message with the proper key and store this SLA specification to establish a customer record in its visiting record table. Meanwhile it will establish an account number and generate a shared key for this valid user. This information should be encrypted with the mobile node's public key and appended as a key distribution IP option extension (Figure 10.6) to the message the AAAF sends to inform the DA of the result of authentication.

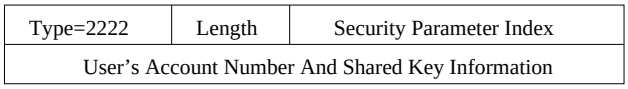

Figure 10.6: A Key Distribution IP Option Extension

8. As soon as the DA is informed about the validity of the user, it will add the user information (the username of the NAI) to its user records database and send an attribute reply to the mobile node. This message includes the original key distribution IP option and gives a list of available services for IP address allocation, such as foreign agent or DHCP servers. Note that each item in the user records database of the DA also has a restricted lifetime. Therefore, a periodical refreshment is necessary.

9. The mobile node first decrypts the key distribution IP option with its private key to achieve its account number and shared key in the AAAF. The UA on the mobile node chooses one service and gets a service handler from the DA. Finally, the mobile node will directly issue a service request message to the corresponding SA. In each service request message sent to the SA, the mobile node has to offer the NAI extension and its account number option extension shown in Figure 10.7. This account number has to be encrypted with the shared key between the mobile node and the AAAF.

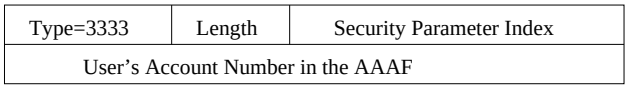

Figure 10.7: An Account Number Option Extension in a Service Request Message

10. Each SA has to contact the AAAF to further authenticate the user's identity for more security. If a SA receives the service request from the mobile node, it first has to send the NAI and the user's encrypted account number to the AAAF.

11. According to the username of the NAI, the AAAF uses the appropriate key to decrypt and check the validity of the user's account number. If the account number is consistent with the information in its visiting records table, the AAAF will inform the SA to supply the desired service to the mobile node. Otherwise, the AAAF will send a negative response to the SA and ask it to reject the request of this user.

12. For a valid user, the SA (now a foreign agent or DHCP server), will allocate an IP address for the mobile node and deliver it in its service reply.

13. After this, the mobile node will continue with the Mobile IP registration procedure and inform the home agent about its current location.

14. Finally, the home agent sends a registration reply to the mobile node indicating that the registration was successful and now access to the network is possible.

10.3.2 QoS Negotiation

After the mobile node has successfully got access to the network, it might further desire to get QoS. When we introduced our signalling protocol in Section 7.7.3 we assumed, that the host sending a reservation request has a valid IP address and is allowed to set up a QoS reservation. This might not hold in a mobile scenario where the mobile host is unknown to the foreign domain. The mobile node has to authenticate itself following the procedure presented earlier in this section.

First, the mobile node has to find an available SA providing Differentiated Services via the DA. The SA specifies what types of Differentiated Services are supported in this foreign domain. The basic operation is as follows:

- The mobile node sends an attribute message to the DA to request QoS.

- When the DA finds the record of the mobile node in its user database based on the username of the NAI, the DA will assume that this user has already been authenticated and subsequently send an attribute reply to the mobile node, which gives the IP address of the SA supplying DiffServ, such as a bandwidth broker.

- Then, the mobile node communicates with the bandwidth broker to negotiate its QoS level. For that purpose the mobile node can use the signalling protocol presented in Section 7.7.3. The bandwidth broker contacts the AAAF to further verify the user with the same procedure described above

(initial foreign network access). For a valid user, the bandwidth broker will perform the reconfigurations and negotiations with the other bandwidth brokers needed to configure the reservation. At the same time, the bandwidth broker also sends a signal to the AAAF to trigger an accounting procedure.

In this example we can see, that after a successful authorisation and authentication the full signalling protocol presented in the preceding chapters can be used.

Summary

In this part we focused on the needs of mobile users if they want to maintain an initial QoS level during roaming through several foreign domains. In Chapter 8 we were able to show, that with a few small additions, the multi-domain bandwidth broker architecture is ready to support mobile users. In Section 8.2 we present new functions in the broker API which enable a mobile user to perform the necessary reservations. An additional function to support the mobile node's QoS-aware handover has been used in Chapter 9. By using this handover technique, the mobile node is notified as soon as the signal quality is too low to stay connected any longer. To achieve this, several daemons have been implemented to permanently monitor the signal quality and to notify the client software. The client then has the possibility to set up a new reservation at the bandwidth broker for the new subnetwork. To improve the chances of a successful handover, the mobile node also has the option to diminish the amount of the reservation.

The last chapter of this part deals with security architectures needed for mobile scenarios. We combine the existing AAA architecture with the SLP and this way can assure that no mobile user without valid authentication and authorisation gets access to a network.

Conclusion and Outlook

In order to summarise our work we would like to come back to the scenario presented in the Introduction (Figure I.1). In this scenario we pointed out the problems a mobile user has in todays networks if he/she wants to sustain an initially negotiated QoS level. Based on this scenario we want to develop a figure of how these problems could be solved with the bandwidth broker framework developed in this thesis. The new components we want to add to the existing infrastructure are shown in Figure C.8 and their interaction is shortly described in the following section. Showing all the signalling protocols and all the messages used to configure a resource reservation and to sustain a QoS level in one single figure would lead to a figure too complicated and too overloaded. In the following Sections we will therefore split the figure into two figures, one showing the home and one the foreign network. In addition to simplify the figures we will skip the Mobile IP registration procedure, since we did not change anything in it. We will also only show the most important communication channels and refer to the description given or to the section where this topic is handled in detail.

New Components

We can see several components newly introduced in the scenario: bandwidth brokers and AAA servers, one for each network or subnetwork, and configuration daemons (CD) running at each DiffServ router. The bandwidth brokers control and manage the routers in their networks (here represented by a single DiffServ ingress router) but also all wireless access points, as it is shown in the “virtual network” picture of the brokers. The control of the access points allows a broker to perform admission control based on the access point, i.e. reject a flow if the wireless cell is already overloaded. Each time the broker receives a reservation request from a unknown host, it asks the AAA server, if the host is allowed to

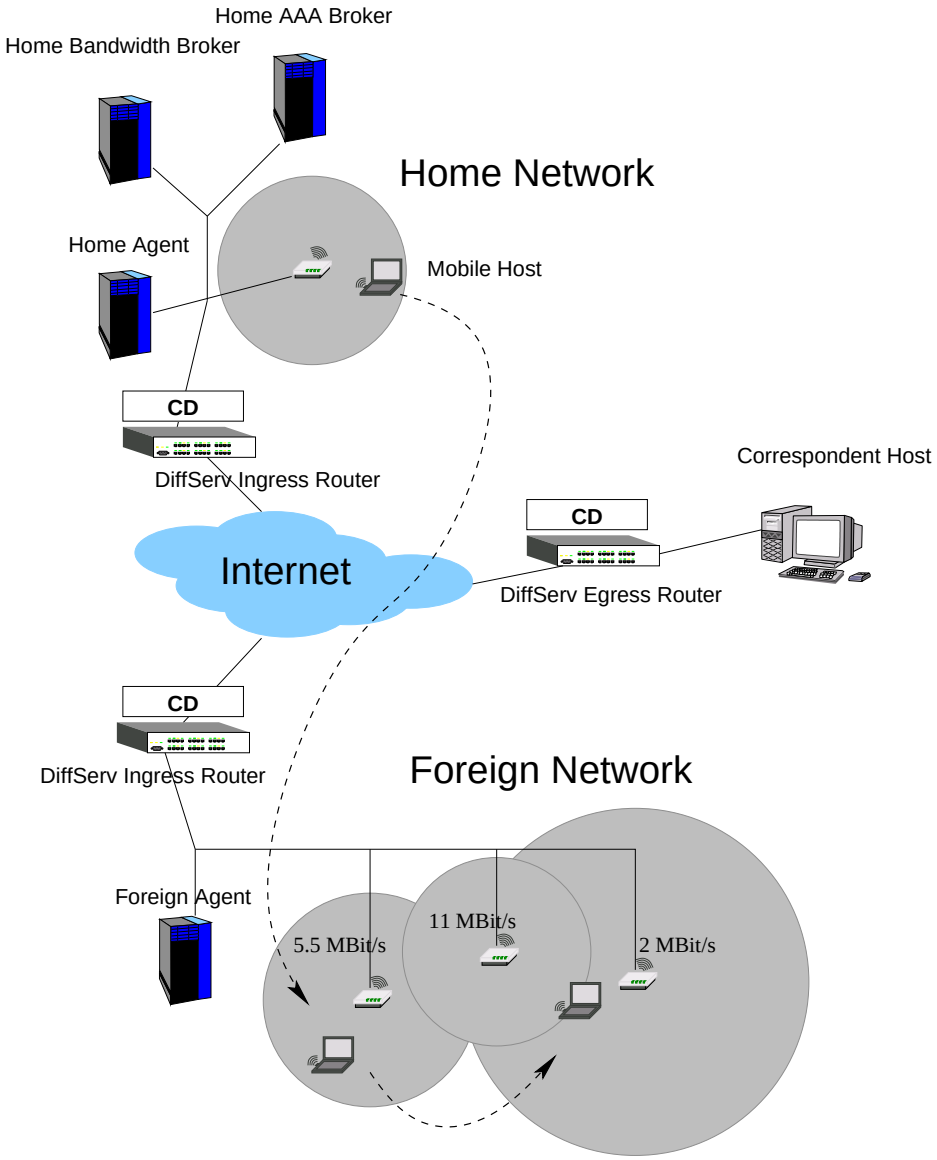

Figure C.8: A Mobile User with QoS Requirements

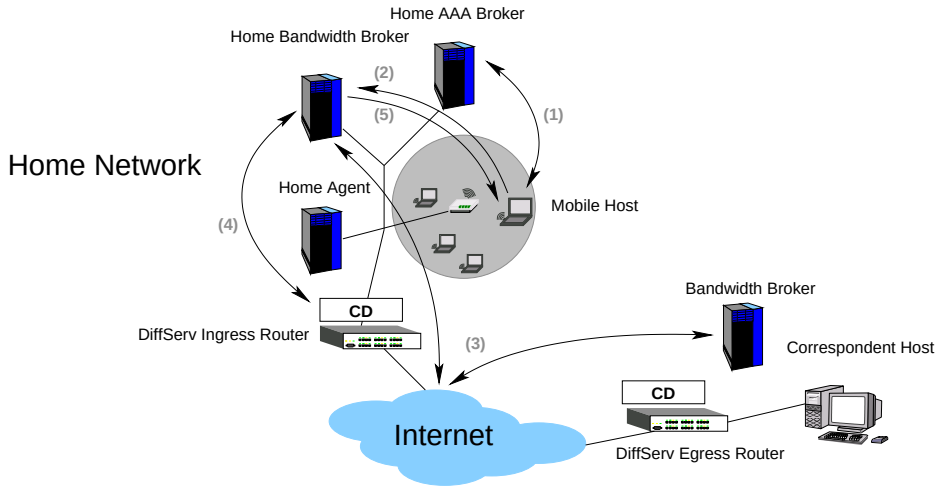

Figure C.9: Communication of a Mobile User in the Home Network

allocate resources. If the answer is positive, the bandwidth broker prepares a configuration for the DiffServ routers and sends it to the configuration daemons. The daemons configure the routers according to the commands of the broker.

Signalling Messages in the Home Network

The messages that are needed to set up a new reservation for a mobile host in its home network may be summarised as follows (cf. Figure C.9):

1. **Registration at the AAA Broker** The mobile host authenticates itself in order to prove its authorisation to set up a reservation. It gets the address of a AAA broker by asking a Directory Agent (not included in the picture) and registers at the AAA broker. The mobile host receives a valid IP address and the address of the bandwidth broker where it can specify its QoS requirements (cf. Section 10.3).

2. **QoS negotiation with the bandwidth broker** The mobile host sends a service level specification to the bandwidth broker to request the transfer of its home SLS to the foreign network. The broker will check in its policy database whether the flow can be admitted or not (cf. Section 7.7.2). Afterwards it will forward the reservation request to the next downstream bandwidth broker.

3. **Downstream setup of the reservation** The reservation request is now forwarded from one bandwidth broker to the next downstream broker as described in Sections 7.3.1 or 7.7. The brokers have a list of neighbouring brokers available, either built via signalling or explicitly configured by the network administrator. The configuration of the own network is performed *after* all downstream domains have already configured the flow in order to avoid useless configurations. The bandwidth is nevertheless allocated at the beginning, this way the successful configuration can be granted.

4. **Configuration of the own network** After the successful reservation in the downstream domains the broker configures the own network. It will query its topology database to find the routers that possibly need reconfiguration. The load table and the reservation table show, if the routers really have to be reconfigured. Finally the configuration is created and sent to the configuration daemons.

5. **Reply to the user** After a successful configuration the user is notified. If somewhere the configuration fails, the user gets a negative reply. Since no configuration at all is performed in this case, the negative answer is much faster.

Signalling Messages in the Foreign Network

For the example of signalling in a foreign network we will assume, that all networks now implement a hierarchical bandwidth broker architecture (cf. Section 7.8). Furthermore we assume that there exists a reservation from the correspondent host to the mobile host's home network which needs to be prolonged to the foreign network. Finally, the wireless access networks in the foreign network might be overcrowded and not enough bandwidth is available in all wireless cells. We will also investigate, how the protocol can deal with that situation.

The signalling in this scenario is much more complicated: First we need a `change_flow` request in a multi-ISP scenario (cf. Section 7.7.4) to change the existing flow from the mobile host to the correspondent host. Second, we need a backward reservation (i.e. initiated by the mobile host) from the home network to the new care-of address for the Mobile IP tunnel. This signalling will all happen within a broker hierarchy.

The mobile host will first connect to the foreign network at location (A). At this point the authentication procedure is performed and access is granted. Now the

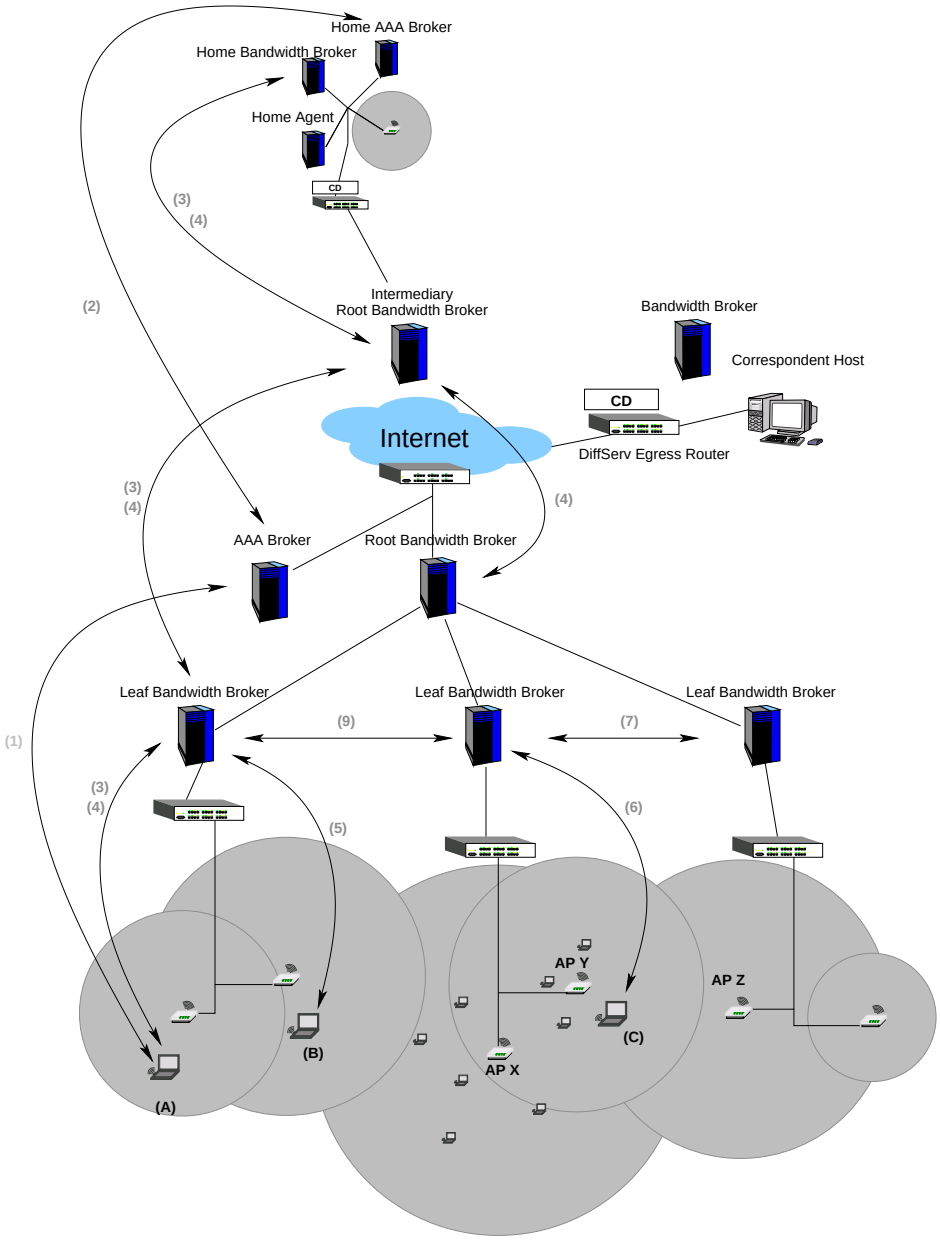

Figure C.10: Communication of a Mobile User in the Foreign Network

mobile host can perform all necessary steps to establish all reservations. This way it has exactly the same QoS level it had in the home network. Afterwards the mobile host will proceed to location (B). At this point a intra-domain handover, which means a handover within the same subnetwork, has to happen. Finally the mobile host moves to location (C). Here we have an overlapping of three wireless cells. Two of them are quite crowded, so that there may not be enough bandwidth available. On the other hand, the third cell is the one offering the worst signal-to-noise ratio, since the access point is a good distance off. Here the mobile user can decide (e.g. by specifying decision parameters to a handover daemon) which QoS parameter is of greatest importance.

If the mobile user follows the path starting from (A) via (B) to (C), the following signalling messages have to be exchanged:

1. **Registration at the AAA Broker** The mobile host authenticates itself at the foreign AAA Broker.

2. **Validation of the authentication** Since the mobile host is unknown in the foreign network the AAA Broker has to check the validity of the authentication of the mobile host at the AAA Broker of the mobile host's home network.

3. **Change the existing reservation** The existing reservation from the mobile host to the correspondent host has to be changed to reflect the new care-of address of the mobile node. The change request establishes a new reservation and travels upstream until it finds the split point of the two paths. At this point the old branch to the home network is deleted (cf. Section 7.7.4). Note, that in a broker hierarchy this request starts at the leaf broker in the foreign network, bypasses the root broker of the foreign network (since it is not a crossing flow, see Section 7.8) and directly goes to the neighbouring root bandwidth broker. The rest of this flow changing procedure only affects root brokers with the exception of the home network. This is a big reduction of signalling messages and therefore strongly affects the total setup speed.

4. **Reservation for the Mobile IP tunnel** If a mobile host wants to get the same QoS level it had in its home network it also has to take care of a reservation for the Mobile IP tunnel, if there exists a reservation from the correspondent host to the home network. Since Mobile IP is transparent to the correspondent host (except of explicit binding update messages in the route optimisation operational modus) the mobile host has to take care of this reservation itself. This is a receiver-initiated (or backward) reservation (cf. Section 7.7.1). Again the request is started at the leaf broker in the

foreign network, goes directly to the neighbouring root broker and is passed upstream until it reaches the home broker. This broker can now start to set up the reservation. Again we can see, that by having broker hierarchies the number of brokers involved and thus the number of signalling messages necessary is greatly reduced.

5. **Intra-domain handover** When the mobile host travels to point (B), it has to perform an intra-domain handover. The necessity of performing a handover is recognised by the SNR-monitoring daemon running on the mobile host (cf. Section 9.3). The mobile host will thereupon scan the available wireless cells, find a new one and query the bandwidth broker if it is possible to perform a handover to this cell while maintaining the QoS level. Since the cell is not crowded, the broker gives a positive reply and the handover can be performed. Note, that there are *no* reconfigurations necessary at all, since the forwarding path has not changed. Nevertheless, the bandwidth broker has to be notified since it has to keep track of the utilisation of the wireless cells in order to admit / reject future resource reservations in this cell.

6. **Inter-domain handover in a broker hierarchy** When the mobile host arrives at point (C) it has to perform another handover. Again the mobile host will scan for available cells. This results in three possible access points (APs X, Y, and Z). This list is passed to the bandwidth broker.

7. **Admission control** The bandwidth broker queries the utilisation and policy databases if the flow can be admitted. If necessary, it can also query other bandwidth brokers.

8. **QoS-aware handover decision** The bandwidth broker passes a list of possible access points back to the user. In our example this list may exclude AP X, since there is no more bandwidth available due to the number of mobile hosts already attached to the access point. AP Y might be included due to a higher availability of bandwidth or the service degradation parameter (cf. Section 9.4.2). We now want to assume, that AP Y offers only 90% of the bandwidth the mobile user would like to reserve. The user has now two possibilities: It can accept the service degradation for the benefit of a good signal-to-noise ratio, or it chooses the higher bandwidth, but has a lower signal-to-noise ratio. This decision is not an easy one to make and it may require additional information to make a good choice. One possibility could be for the bandwidth broker to provide the geographical location of the access points. The mobile host could then easily predict a travelling path, if such a route is not already configured by the user. In our example, a prediction would most likely propose AP Z for attachment, whereas a

preconfigured route could go north and choose AP Y instead.

9. **Handover between leaf brokers** The necessary changes of resource reservations for this handover affect the leaf brokers in this hierarchy only. Redirecting the reservation requests to the root broker would only result in additional signalling and yield no performance benefit. The changing of the reservations to the new care-of address is therefore performed as previously described (cf. Section 7.7.4).

Timing Considerations

Now we want to estimate the latency a mobile user has when performing a handover from its home network to a foreign network. Of course this delay heavily depends on the distance between the home and the foreign network: if those networks are separated by a transcontinental link, the delay of one single signalling message can be 40 ms and more. This again signifies the importance of managing handovers locally as far as possible, as it is described in the previous section. The fact, that handover latency depends on delays that are outside the scope of a bandwidth broker, or more generally out of the QoS framework, means, that we can only estimate the amount of latency additionally caused by our framework.

Bandwidth Broker

The latency added by our bandwidth broker hierarchy mainly depends on two parameters: the number of bandwidth brokers that are involved and the possibility of a fast reservation (where only the ingress router has to be reconfigured). The size of the topology the individual bandwidth brokers manage does *not* influence the result in a significant way (cf. Section 7.6). The worst case, in which every router along the path has to be reconfigured is negligible as it hardly ever occurs. We may therefore assume, that the majority of the users has an increased latency which is the sum of all latencies of bandwidth brokers along the path This latency is approximately 0.5 ms per broker. If we take into account, that a broker can manage more than 1000 nodes, the additional delay due to QoS signalling will not be more than 5 ms, even for transcontinental distances. Compared to the link delay of 40 – 200 ms, this is almost negligible.

Usually the biggest amount of handovers can be performed locally, which means that only one or two bandwidth brokers are involved and need to be informed. In this case, the additional latency will be approximately 1 – 2 ms. In this case the

delay introduced by the broker is of the same magnitude as the link delay which is also about 2 ms for an IEEE 802.11b access network.

Topology Updates

Throughout this thesis we assumed the topology to remain static after the startup of the bandwidth broker. This allowed us to simplify the design of the topology and the reservation databases. However, this assumption may not be valid in a large network over a longer interval of time. Although the routes to popular destinations seem to be reasonable stable — despite the large number of BGP(Border Gateway Protocol) updates [139] — it might happen that a route changes while a reservation is still relying on the forwarding given by this route. We now want to shortly discuss possible solutions to this problem, which has not been solved in any bandwidth broker implementation yet.

The problem of changes in the routing is mainly caused by the aggregation of flows in the core routers: the core routers do not store per-flow information because of scalability reasons they only store the overall sum of bandwidth kept. If a link goes down, the underlying routing protocol might change the routing in such a way, that one part of the flows is now routed via one interface, but another part via another interface of this router (and might even not cross this router at all). Since the core router only has information about the aggregated sum of bandwidth we cannot know how to divide this sum into parts needed for the new routes.

One possible solution would be the allocation of the total sum of bandwidth on the new outgoing interfaces. This of course would result in a large overprovisioning and waste of bandwidth and is therefore not desirable. We propose another solution that breaks the rule of only storing aggregate information in the core routers in a way, that the bandwidth broker maintains a reservation value not only for each interface of a core router but for each routing table entry (see Table C.1).

Destination	Gateway	Interface	Reservation
$Dest_1$	$Gatew_1$	Ifc_1	500
$Dest_2$	$Gatew_1$	Ifc_1	300
$Dest_3$	$Gatew_2$	Ifc_2	700
$Dest_4$	$Gatew_1$	Ifc_1	700
$Dest_5$	$Gatew_3$	Ifc_3	500
$Dest_6$	$Gatew_2$	Ifc_2	200

Table C.1: Example for the new Routing Table Format

With this additional information we can correctly change the topology and reservation database in case of routing changes: Assume, that the link of Interface 1 breaks and the routing protocol provides us with the following new routes: destinations $Dest_1$ and $Dest_2$ are now routed via interface Ifc_2, while $Dest_4$ is routed via Ifc_3. The new routing table now looks like Table C.2.

Destination	Gateway	Interface	Reservation
$Dest_1$	$Gatew_2$	Ifc_2	500
$Dest_2$	$Gatew_2$	Ifc_2	300
$Dest_3$	$Gatew_2$	Ifc_2	700
$Dest_4$	$Gatew_3$	Ifc_3	700
$Dest_5$	$Gatew_3$	Ifc_3	500
$Dest_6$	$Gatew_2$	Ifc_2	200

Table C.2: The new Routing Table after the Topology Changes

The bandwidth to be configured newly at the interfaces can easily be compute from the routing table: Ifc_2 needs 1700 units, while Ifc_3 will be configured 1200 units.

Deletion of a routing table entry is also quite simple: we will only have to check tin he new routing table, over which gateway the addresses of the deleted subnetwork will now be routed. The only thing that has to be done is to add the bandwidth used for the deleted network to the reservation entry of this gateway.

Outlook and Future Work

The implementation of the bandwidth broker architecture for mobile users is not carried out in detail yet. There are many things still to be done to accomplish a fully developed framework:

- Although the QoS management API offers the flexibility to support various kinds of router hardware, only an implementation for Linux routers has been implemented. Other implementations for commercial routers (Cisco, NEC, Allied Telesyn, etc.) are still missing.
- The implementation of security mechanisms has not yet been performed. The AAA architecture presented in Chapter 10 provides basic guidelines for such an implementation. These guidelines can be used as a basis for a complete security packet

- The bandwidth broker – router communication has to be secure, too in order to prevent malicious reconfiguration of network elements. This could perhaps be implemented using a `ssh` tunnel.
- The user interface for the bandwidth broker could still be improved. A web-based interface would be very convenient for the user. However, the notification of the user in case of necessary interaction (e.g. a handover could not be performed using the parameters given to the client software) is a problem.
- Although our QoS-aware handover procedure should be able to provide reasonable fast and smooth handovers, the handover techniques presented in Section 1.3 should also be included in the framework.
- Performance evaluations for the multi-domain and for the mobile scenarios would be interesting to locate possible bottlenecks and further optimise the implementation.

The bandwidth broker architecture and the QoS signalling protocol can also be extended and used for other purposes.

- Our present admission control algorithm is not fully developed yet and can still be improved. For example, measurement-based admission control can improve the bandwidth utilisation of an access network significantly, if several users are not using their full amount of reserved bandwidth at the same time.
- Presently, our implementation can only be used to set up DiffServ reservations. However, the architecture could be extended to configure non-DiffServ reservations, such as MPLS paths.
- The handover decision, as it is now implemented, is only based on signal quality and on the availability of resources. We have already mentioned, that it could be advantageous to include information outside the scope of networking, such as geographical information (e.g. from a GPS receiver) or route information. By taking such information into account, the mobile client would be able to come to a better decision about the new access point.
- Our signalling protocol only supports the setup of unicast flows. However, it would also be interesting to support multicast flows. To do so, we would have to extend the bandwidth broker to query the multicast routing information from the routers and to calculate the multicast tree. Another possibility would be to calculate an optimal multicast tree for an existing group and explicitly force the routing of this tree to the network.

- The reservation and utilisation tables of the bandwidth broker could also be used to monitor the validity of the network state. With the use of monitoring information from routers we could easily detect a unexpected increase of traffic to a certain destination, thus recognising Denial of Service attacks.
- Accounting is still a big issue for value-added services like Differentiated Service.

List of Abbreviations

AAA Authentication, Authorization and Accounting

AAAB AAA broker (trusted third party)

AAAF AAA server in the foreign domain

AAAH AAA server in the home domain

AF Assured Forwarding

API Application Programmers Interface

BA Behaviour Aggregate

BB Bandwidth Broker

CAR Call Admission Rate

CH Correspondent Host

CLI Command Line Interface

CMIP Common Management Information Protocol

CMIS Common Management Information Services

CPU Central Processing Unit

CSMA/CA Carrier Sense Multiple Access / Collision Avoidance

CSMA/CD Carrier Sense Multiple Access / Collision Detection

CoA Care of Address

DA Directory Agent

DHCP Dynamic Host Configuration Protocol

DNS Domain Name Service

DSCP Differentiated Services Code Point

EF Expedited Forwarding

ESSID Extended Segment Set Identification

FA Foreign Agent

FTP File Transfer Protocol

GUI Graphical User Interface

HA Home Agent

IAB Internet Architecture Board

ICMP Internet Control Message Protocol

ID Identification Number

IEEE Institute of Electrical and Electronics Engineers

IETF Internet Engineering Task Force

IP Internet Protocol

ISO International Organization for Standardization

ISP Internet Service Provider

LAN Local Area Network

MAC Medium Access

MF Multi Field

MH Mobile Host

MPEG2 Motion Pictures Expert Group Standard 2

NAI Network Access Identifier

PDA Portable Digital Assistant

PDB Per Domain Behaviour

PHB Per Hop Behaviour

QoS Quality of Service

RFC Request For Comment

RPC Remote Procedure Call

RSVP Resource Reservation Protocol

SA Service Agent

SLA Service Level Agreement

SLP Service Location Protocol

SLS Service Level Specification

SL Service Level

SNMP Simple Network Management Protocol

STL Standard Template Library

TCA Traffic Conditioning Agreement

TCP Transmission Control Protocol

TCS Traffic Conditioning Specification

TC Traffic Control

TTL Time To Live

ToS Type of Service

UA User Agent

UDP User Datagram Protocol

UMTS Universal Mobile Telecommunications System

VoIP Voice over IP

Bibliography

[1] I. Aad and C. Castelluccia. Differentiation mechanisms for IEEE 802.11. In *IEEE Infocomm 2001,Anchorage, Alaska,*, April 2001.

[2] I. Aad and C. Castelluccia. Remarks On Per-flow Differrentiaition In IEEE 802.11. In *European Wireless 2002, Florence, Italy*, February 2002.

[3] Agere Systems. AP Manager. Available online from http://www.orinocowireless.com.

[4] W. Almesberger. Linux network traffic control — implementation overview. ftp://lrcftp.epfl.ch/ pub/ people/ almesber/ pub/ tcio-current.ps, April 1999.

[5] W. Almesberger, J. Salim, and A. Kuznetsov. Differentiated Services on Linux. Internet Draft, draft-almesberger-wajhak-diffserv-linux-01.txt, June 1999. work in progress.

[6] B. Andersson, D. Forsberg, J. Hautio, H. Kari, J. Malinen, J. Malinen, K. Mustonen, and T. Weckström. Dynamics - HUT Mobile IP. http://www.cs.hut.fi/Research/Dynamics/.

[7] R. Atkinson. RFC 1826: IP Authentication Header, August 1995. Obsoleted by RFC2402 [87].

[8] R. Balmer, M. Günter, and T. Braun. Video Streaming in a DiffServ/IP Multicast Network. In *Workshop Advanced Internet Charging and QoS Technology at Informatik 2001 (ICQT)*, September 2001.

[9] M. Bauer and H. Akhand. Managing Quality-of-Service in Internet Applications Using Differentiated Services. *Journal of Network and Systems Management*, 10(1), March 2002.

[10] F. Baumgartner and T. Braun. Fairness of Assured Service. In *Modelling and Simulation, 13th European Simulation Conference'99, Warsaw*, volume 1, 1999. ISBN 1-56555-171-0.

[11] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[12] Y. Bernet, S. Blake, D. Grossman, and A. Smith. RFC 3290: An Informal Management Model for Diffserv Routers, May 2002.

[13] Y. Bernet, P. Ford, R. Yavatkar, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, J. Wroclawski, and E. Felstaine. RFC 2998: A Framework for Integrated Services Operation over Diffserv Networks, November 2001.

[14] Y. Bernet, A. Smith, S. Blake, and D. Grossman. A Conceptual Model for DiffServ Routers. Internet Draft, March 2000. work in progress.

[15] J. Biswas, A. A. Lazar, J. F. Huard, K. Lim, S. Mahjoub, L.-F. Pau, M. Suzuki, S. Torstensson, W. Wang, and S. Weinstein. The IEEE P1520 Standards Initiative for Programmable Network Interfaces. *IEEE Communications Magazine*, 36(10):64–72, October 1998.

[16] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC 2475: An Architecture for Differentiated Service, December 1998.

[17] R. Bless and K. Wehrle. Evaluation of Differentiated Services using an implementation under Linux. In *Seventh IFIP International Workshop on Quality of Service (IWQOS'99), London, UK*, June 1999. ISBN 0-7803-5671-3.

[18] U. Blumenthal and B. Wijnen. RFC 2264: User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3), January 1998. Obsoleted by RFC2274 [19].

[19] U. Blumenthal and B. Wijnen. RFC 2274: User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3), January 1998. Obsoletes RFC2264 [18]. Obsoleted by RFC2574 [20].

[20] U. Blumenthal and B. Wijnen. RFC 2574: User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3), March 2000. Obsoletes RFC2274 [19].

[21] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, and S. Jamin. RFC 2205: Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification, September 1997.

[22] R. Braden and L. Zhang. RFC 2209: Resource ReSerVation Protocol (RSVP) – Version 1 Message Processing Rules, September 1997.

[23] T. Braun, C. Castelluccia, and G. Stattenberger. An Analysis of the DiffServ Approach in Mobile Environments. In *Proceedings of 1st Workshop of IP Quality of Service for wireless and mobile networks (IQWiM'99), Aachen, Germany*, 1999.

[24] T. Braun, M. Günter, and I. Khalil. Management of Quality of Service Enabled VPNs. *IEEE Communications Magazine*, May 2001.

[25] T. Braun, Li Ru, and G. Stattenberger. An AAA Architecture Extension for Providing Differentiated Services to Mobile IP Users. In *Proceedings of the 6th IEEE Symposium on Computers and Communications (ISCC 2001), Hammamet, Tunesia*, July 2001.

[26] T. Braun, M. Scheidegger, H. Einsiedler, G. Stattenberger, and K. Jonas. A Linux Implementation of a Differentiated Services Router. In Sathya Rao and Kaare Ingar Sletta, editors, *Next Generation Networks — Networks and Services for the Information Society*, volume 1938 of *Lecture Notes in Computer Science*, pages 302 – 315, October 2000.

[27] P. Calhoun and C. Perkins. RFC 2794: Mobile IP Network Access Identifier Extension for IPv4, March 2000.

[28] P. R. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J Arkko. Diameter Base Protocol. Internet Draft, draft-ietf-aaa-diameter-15.txt, October 2002. work in progress.

[29] J. Case, D. Harrington, R. Presuhn, and B. Wijnen. RFC 2262: Message Processing and Dispatching for the Simple Network Management Protocol (SNMP), January 1998.

[30] J. Case, D. Harrington, R. Presuhn, and B. Wijnen. RFC 2272: Message Processing and Dispatching for the Simple Network Management Protocol (SNMP), January 1998. Obsoletes RFC2262 [29]. Obsoleted by RFC2572 [31].

[31] J. Case, D. Harrington, R. Presuhn, and B. Wijnen. RFC 2572: Message Processing and Dispatching for the Simple Network Management Protocol (SNMP), April 1999. Obsoletes RFC2272 [30].

[32] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. RFC 1442: Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2), April 1993.

[33] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. RFC 1449: Transport Mappings for version 2 of the Simple Network Management Protocol (SNMPv2), April 1993. Obsoleted by RFC1906 [36].

[34] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. RFC 1902: Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2), January 1996. Obsoletes RFC1442 [32]. Obsoleted by RFC2578 [117].

[35] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. RFC 1905: Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2), January 1996.

[36] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. RFC 1906: Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2), January 1996. Obsoletes RFC1449 [33].

[37] V.G. Cerf. RFC 1052: IAB recommendations for the development of Internet network management standards, April 1989.

[38] P. Chandra, A. Fisher, and P. Steenkiste. Beagle: A Resource Allocation Protocol for an Advanced Services Internet. Technical Report CMU-CS-98-150, Carnegie Mellon University, 1998.

[39] H. Chaskar and R. Koodli. A framework for QoS support in mobile IPv6. Internet Draft, March 2001. work in progress.

[40] B. Davie, A. Charny, J.C.R. Bennet, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. RFC 3246: An Expedited Forwarding PHB (Per-Hop Behavior), March 2002. Obsoletes RFC2598 [78].

[41] A. De Carolis and L. Dell'Uomo. QoS-Aware Handover for Mobile IP. Internet Draft, draft-decarolis-qoshandover-02.txt, April 2001. work in progress.

[42] C. de Laat, G. Gross, L. Gommans, J. Vollbrecht, and D. Spence. RFC 2903: Generic AAA Architecture, August 2000.

[43] J. Diederich, T. Lohmar, M. Zitterbart, and R. Keller. A QoS Model for Differentiated Services in Mobile Wireless Networks. In *Paper Digest of the 11th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN'01), Boulder Co., USA*, March 2001.

[44] J. Diederich and M. Zitterbart. An expedited forwarding with dropping PHB. Internet Draft, draft-dieder-diffserv-phb-efd-00.txt, October 1999. work in progress.

[45] Matthew Doar. Tiers topology generator. http://www.geocities.com/ ResearchTriangle/ 3867/ sourcecode.html.

[46] Z. Duan and Z.-L. Zhang. A Scalable Bandwidth Management Architecture for Supporting VoIP Applications Using Bandwidth Broker. In *Paper Digest of the 11th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN'01), Boulder Co., USA*, March 2001.

[47] D. Eastlake. RFC 1455: Physical Link Security Type of Service, May 1993. Obsoleted by RFC2474 [124].

[48] C. Perkins (Ed.). RFC 2002: IP Mobility Support, October 1996. Obsoleted by RFC3220 [49].

[49] C. Perkins (Ed.). RFC 3220: IP Mobility Support for IPv4, January 2002. Obsoletes RFC2002 [48].

[50] C. Perkins (Ed.). RFC 3344: IP Mobility Support for IPv4, August 2002. Obsoletes RFC3220 [49].

[51] G. Dommety (Ed.), A. Yegin, C. Perkins, G. Tsirtis, K. El-Malki, and M. Khalil. Fast Handovers for Mobile IPv6. Internet Draft, draft-ietf-mobileip-fast-mipv6-04.txt, March 2002. work in progress.

[52] H. Chaskar (Ed.). Requirements of a QoS Aolution for Mobile IP. Internet Draft, draft-ietf-mobileip-qos-requirements-03.txt, July 2002. work in progress.

[53] J. Kempf (Ed.). Problem Description: Reasons for performing Context Transfers Between Nodes in an IP Access Network. Internet Draft, draft-ietf-seamoby-context-transfer-problem-stat-04.txt, August 2002. work in progress.

[54] K. El-Malki (Ed.), P. Calhoun, T. Hiller, J. Kempf, P. McCann, A. Singh, H. Soliman, and S. Thalanany. Low Latency Handoffs in Mobile IPv4. Internet Draft, draft-ietf-mobileip-lowlatency-handoffs-v4-04.txt, June 2002. work in progress.

[55] M. Brunner (Editor). Requirements for QoS Signaling Protocols. Internet Draft, draft-ietf-nsis-req-03.txt, November 2002. work in progress.

[56] Y. Ezaki and Y. Imai. Mobile IPv6 handoff by Explicit Multicast. Internet Draft, draft-ezaki-handoff-xcast-01.txt, May 2001. work in progress.

[57] G. Fankhauser, S. Hadjiefthymiades, N. Nikaein, and L. Stacey. RSVP Support for Mobile IP Version 6 in Wireless Environments. Internet Draft, draft-fhns-rsvp-support-in-mipv6-00.pdf, November 1998. work in progress.

[58] S. Farrell, J. Vollbrecht, P. Calhoun, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. RFC 2906: AAA Authorization Requirements, August 2000.

[59] A. Fieger, M. Zitterbart, R. Keller, and J. Diederich. Towards qos support in the presence of handover,. In *Proceedings of 1st Workshop of IP Quality of Service for wireless and mobile networks (IQWiM'99), Aachen, Germany*, April 1999.

[60] S. Floyd and V. Jacobsen. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4), August 1993.

[61] L. R. Ford and D.R Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[62] X. Fu, H. Karl, and C. Kappler. QoS-Conditionalized Handoff for Mobile IPv6. In *Proc. of the Second IFIP-TC6 Networking Conf. - Networking2002*, pages 721 – 730, Pisa, Italy, May 2002. Springer Verlag.

[63] J. A. Garcia-Macias, F. Rousseau, G. Berger-Sabbatel, L. Toumi, and A. Duda. Quality of Service and Mobility for the Wireless Internet. In *Proc. First ACM Wireless Mobile Internet Workshop, Rome, Italy*, 2001.

[64] editor Gary Kenward. General Requirements for Context Transfer. Internet Draft, draft-ietf-seamoby-ct-reqs-05.txt, October 2002. work in progress.

[65] http://www.dante.net/geant/about-geant.html, September 2002.

[66] D. Grossman. RFC 3260: New Terminology and Clarifications for Diffserv, May 2002.

[67] M. Günter. *Management of Multi-Provider Internet Services with Software Agents*. PhD thesis, University of Bern, June 2001.

[68] M. Günter and T. Braun. Evaluation of Bandwidth Broker Signaling. In *Proceedings of the International Conference on Network Protocols ICNP'99, Toronto, Canada*, pages 145 – 152. IEEE Computer Society, November 1999. ISBN 0-7695-0412-4.

[69] E. Guttman, C. Perkins, J. Veizades, and M. Day. RFC 2608: Service Location Protocol, Version 2, June 1999.

[70] D. Harrington, R. Presuhn, and B. Wijnen. RFC 2261: An Architecture for Describing SNMP Management Frameworks, January 1998. Obsoleted by RFC2271 [71].

[71] D. Harrington, R. Presuhn, and B. Wijnen. RFC 2271: An Architecture for Describing SNMP Management Frameworks, January 1998. Obsoletes RFC2261 [70]. Obsoleted by RFC2571 [184].

[72] Hasan, J. Jähnert, S. Zander, and B. Stiller. Authentication, Authorization, Accounting, and Charging for the Mobile Internet. Technical Report TIK-Report No. 114, Eidgenössische Technische Hochschule Zürich, June 2001.

[73] Hasan, J. Jähnert, S. Zander, and B. Stiller. Authentication, Authorization, Accounting, and Charging for the Mobile Internet. In *IST Mobile Communications Summit 2001, Barcelona, Spain*, September 2001.

[74] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. RFC 2597: Assured Forwarding PHB Group, June 1999.

[75] J. Heinanen and R. Guerin. RFC 2698: A Two Rate Three Color Marker, September 1999.

[76] Cisco Systems Inc. *Internet Technology Handbook*. Cisco Press, third edition, December 2000.

[77] Bandwidth Broker Implementation. www.ittc.ukans.edu/˜kdrao/ BB/ bbreport.html.

[78] V. Jacobson, K. Nichols, and K. Poduri. RFC 2598: An Expedited Forwarding PHB, June 1999. Obsoleted by RFC3246 [40].

[79] V. Jacobson, K. Nichols, and K. Poduri. The 'Virtual Wire' Per-Domain Behaviour. Internet Draft, draft-ietf-diffserv-pdb-vw-00.txt, July 2000. work in progress.

[80] S. Jamadagni and S. Pandey. Optimized IP Mobility - Requirements for Underlying Systems. Internet Draft, draft-satish-l2-mobilereq-00.txt, February 2002. work in progress.

[81] R. Jayaraj. Cell-Search List Indications for Seamless Anticipative, Resource-Mindful Handovers. Internet Draft, draft-rjaya-seamoby-cslist-ind-00.txt, May 2002. work in progress.

[82] H. Jung, H. Kang, and C. Lee. Fast Handoff with Chain Tunneling for Mobile IPv6. Internet Draft, draft-jung-mobileip-fastho-chain-00.txt, June 2002. work in progress.

[83] J. Kempf, D. Blair, P. Reynolds, and A. O'Neill. Leveraging Fast Handover Protocols to Support Localized Mobility Management in Mobile IP. Internet Draft, draft-kempf-mobileip-fastho-lmm-00.txt, June 2002. work in progress.

[84] J. Kempf, P. Calhoun, and D. Frascone. The Diameter API. Internet Draft, draft-ietf-aaa-diameter-api-03, November 2002. work in progress.

[85] J. Kempf, Y. Gwon, D. Funato, A. Takeshita, and J. Wood. Post-handover Mobile Initiated Tunneling for Fast Mobile IPv4 Handover. Internet Draft, draft-kempf-mobileip-postmit-handover-00.txt, June 2002. work in progress.

[86] S. Kent and R. Atkinson. RFC 2401: Security Architecture for the Internet Protocol, November 1998.

[87] S. Kent and R. Atkinson. RFC 2402: IP Authentication Header, November 1998. Obsoletes RFC1826 [7].

[88] I. Khalil and T. Braun. A Range-Based SLA and Edge Driven Virtual Core Provisioning in DiffServ-VPNs. In *The 26th Annual IEEE Conference on Local Computer Networks (LCN'2001), Tampa, USA*, November 2001.

[89] I. Khalil and T. Braun. Implementation of a Bandwidth Broker for Dynamic End-to-End Capacity Reservation over Multiple Diffserv Domains. In *4th IFIP/IEEE International Conference on Management of Multimedia Networks and Services (MMNS), Chicago, USA*, November 2001.

[90] I. Khalil and T. Braun. Automated Service Provisioning in Heterogenous Large-Scale Enviroment. In *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS 2002), Firenze, Italia*, April 2002.

[91] I. Khalil and T. Braun. Edge Provisioning of VPN-DiffServ Networks,. *Journal on Network and Systems Management*, 10(1), March 2002.

[92] P. Kivimäki. Policy Based Networks & Bandwidth Broker. www.atm.tut.fi/workshop01/ workshop01-bb.pdf.

[93] A. Kolarov. Study of the TCP/UDP Fairness Issue for the Assured Forwarding Per Hop Behaviour in Differentiated Services Networks. In *Proceedings of the IEEE Workshop on High Performance Switching and Routing (HPSR 2001), Dallas, Texas, USA*, May 2001.

[94] R. Koodli and C. Perkins. A Framework for Smooth Handovers with Mobile IPv6. Internet Draft, draft-ietf-koodli-smoothv6-00.txt, July 2000. work in progress.

[95] R. Koodli and C. Perkins. Fast Handovers in Mobile IPv6. Internet Draft, draft-perkins-mobileip-fastv6-00.txt, October 2000. work in progress.

[96] R. Koodli and C. Perkins. A context Transfer Protocol for Seamless Mobility. Internet Draft, draft-koodli-seamoby-ct-04.txt, August 2002. work in progress.

[97] G. Krishnamurthi, R. Chalmers, and C. Perkins. Buffer Management for Smooth HandOvers in Mobile IPv6. Internet Draft, draft-krishnamurthi-mobileip-buffer6-00.txt, July 2000. work in progress.

[98] G. Krishnamurthi, R. Chalmers, and C. Perkins. Buffer Management for Smooth HandOvers in Mobile IPv6. Internet Draft, draft-krishnamurthi-mobileip-buffer6-00.txt, July 2000. work in progress.

[99] Alexey Kuznetsov. iproute2 release 990824. ftp://ftp.sunet.se/pub/Linux/ip-routing/iproute2-2.2.4-now-ss990824.tar.gz.

[100] D. Levi, P. Meyer, and B. Stewart. RFC 2263: SNMPv3 Applications, January 1998. Obsoleted by RFC2273 [101].

[101] D. Levi, P. Meyer, and B. Stewart. RFC 2273: SNMPv3 Applications, January 1998. Obsoletes RFC2263 [100]. Obsoleted by RFC2573 [102].

[102] D. Levi, P. Meyer, and B. Stewart. RFC 2573: SNMP Applications, April 1999. Obsoletes RFC2273 [101].

[103] D. Levi and J. Schoenwaelder. RFC 2592: Definitions of Managed Objects for the Delegation of Management Script, May 1999. Obsoleted by RFC3165 [104].

[104] D. Levi and J. Schoenwaelder. RFC 3165: Definitions of Managed Objects for the Delegation of Management Scripts, August 2001. Obsoletes RFC2592 [103].

[105] R.-F. Liao, R. H. Wouhaybi, and A.T. Campbell. Incentive Engineering in Wireless LAN Based Access Networks. In *Proc. 10th International Conference on Network Protocols (ICNP 2002), Paris, France*, November 2002.

[106] A. Lindgren, A. Almquist, and O. Schelen. Evaluation of Quality of Service Schemes for IEEE 802.11 Wireless LANs. In *Proceedings of the 26th Annual IEEE Conference on Local Computer Networks (LCN'2001), Tampa, USA*, November 2001.

[107] Lucent Technologies. . . . *IEEE 802.11 channel selection guidelines*: Using multiple channels in WaveLAN / IEEE 802.11. WaveLAN Technical Bulletin 003/A, Lucent Technologies, November 1998.

[108] Lucent Technologies. . . . *roaming with WaveLAN/IEEE 802.11*. WaveLAN Technical Bulletin 021/A, Lucent Technologies, December 1998.

[109] Lucent Technologies. . . . *Planning Large Scale Installations*: Installation Guidelines. WaveLAN Technical Bulletin 023/B, Lucent Technologies, April 1999.

[110] Lucent Technologies. . . . *security*: WaveLAN / IEEE 802.11 Security Layers. WaveLAN Technical Bulletin 002/A, Lucent Technologies, September 1999.

[111] J. Malinen and C. Perkins. Mobile IPv6 Regional Registrations. Internet Draft, draft-malinen-mobileip-regreg6-00.txt, July 2000. work in progress.

[112] A. Mankin, Ed., F. Baker, B. Braden, S. Bradner, M. O'Dell, A. Romanow, A. Weinrib, and L. Zhang. RFC 2208: Resource ReSerVation Protocol (RSVP) – Version 1 Applicability Statement Some Guidelines on Deployment, September 1997.

[113] J. Manner and M. Kojo (Eds.). Mobility Related Terminology. Internet Draft, draft-ietf-seamoby-mobility-terminology-00.txt, August 2002. work in progress.

[114] V. Marques, R. Aguiar, F. Fontes, J. Jähnert, and H. Einsiedler. Enabling IP QoS in Mobile Environments. In *IST Mobile Communications Summit 2001, Barcelona, Spain*, September 2001.

[115] V. Marques, R. Aguiar, J. Jähnert, K. Jonas, M. Liebsch, H. Einsiedler, and F. Fontes. A Heterogeneous Mobile IP QoS-aware Network. In *accepted for oral presentation at CRC 2001*, November 2001.

[116] P. Martinez, M. Brunner, J. Quittek, F. Strauß, J. Schönwälder, S. Mertens, and T. Klie. Using the Script MIB for Policy-based Configuration Management. In *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS 2002), Firenze, Italia*, April 2002.

[117] K. McCloghrie, D. Perkins, and J. Schoenwaelder. RFC 2578: Structure of Management Information Version 2 (SMIv2), April 1999. Obsoletes RFC1902 [34].

[118] K. McCloghrie, D. Perkins, and J. Schoenwaelder. RFC 2579: Textual Conventions for SMIv2, April 1999.

[119] K. McCloghrie, D. Perkins, and J. Schoenwaelder. RFC 2580: Conformance Statements for SMIv2, April 1999.

[120] Sun Microsystems. RFC 1050: RPC: Remote Procedure Call Protocol specification, April 1989.

[121] D. Mitton and M. Beadles. RFC 2881: Network Access Server Requirements Next Generation (NASREQNG) NAS Model, July 2000.

[122] G. Montenegro and Ed. RFC 2344: Reverse Tunneling for Mobile IP, May 1998. Obsoleted by RFC3024 [123].

[123] G. Montenegro and Ed. RFC 3024: Reverse Tunneling for Mobile IP, revised, February 2001. Obsoletes RFC2344 [122].

[124] K. Nichols, S. Blake, F. Baker, and D. Black. RFC 2474: Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, December 1998. Obsoletes RFC1455 [47].

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

[126] K. Nichols, V. Jacobson, and L. Zhang. RFC 2638: A Two-bit Differentiated Services Architecture for the Internet, July 1999.

[127] J. Norton. Dynamic class loading for C++ on linux. *Linux Journal*, 73, March 2000. http://www2.linuxjournal.com/lj-issues/issue73/3687.html.

[128] J. Ogawa and Y. Nomura. A Simple Resource Management Architecture for Differentiated Services. In *Proceedings of the 10th Annual INET Conference, Yokohama, Japan*, July 2000.

[129] M. Ohta. Smooth Handover over IEEE 802.11 Wireless LAN. Internet Draft, draft-ohta-smooth-handover-wlan-00.txt, June 2002. work in progress.

[130] P. Pan and H. Schulzrinne. YESSIR: A Simple Reservation Mechanism for the Internet. *Computer Communication Review*, 29(2), 1999.

[131] C. Perkins. RFC 2003: IP Encapsulation within IP, October 1996.

[132] C. Perkins. Fast Handovers for Mobile IPv6. Internet Draft, draft-perkins-mobileip-handover-00.txt, November 2000. work in progress.

[133] C. Perkins and D. Johnson. Route Optimization in Mobile IP. Internet Draft, draft-ietf-mobileip-optim-10.txt, November 2000. work in progress.

[134] G. Politis, P. Sampatakos, and I. Venieris. Design of a Multi-Layer Bandwidth Broker Architecture. In Sathya Rao and Kaare Ingar Sletta, editors, *Next Generation Networks — Networks and Services for the Information Society*, volume 1938 of *Lecture Notes in Computer Science*. Springer Verlag, October 2000.

[135] O. Pop, T. Mahr, T. Dreilinger, and R. Szabo. Vendor-Independent Bandwidth Broker Architecture for DiffServ Networks. In *Proceedings of the IEEE International Conference on Telecommunications (ICT 2001) Bucharest, Romania*, June 2001.

[136] S. Radhakrishnan. Linux - Advanced Networking Overwiew. Available online from http://qos.ittc.ukans.edu/howto.

[137] F. Reichmeyer, L.Ong, A. Terzis, L. Zhang, and R.Yavatkar. A two-tier resource management model for differentiated services networks. Internet Draft, November 1998. work in progress.

[138] C. Rensing, Hasan, M. Karsten, and B. Stiller. A Survey on AAA Mechanisms, Protocols, and Architectures and a Policy-based Approach beyond: A^x. Technical Report TIK-Report No. 111, Eidgenössische Technische Hochschule Zürich, May 2001.

[139] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang. BGP Routing Stability of Popular Destinations. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop, Marseille, France*, November 2002.

[140] C. Rigney, A. Rubens, W. Simpson, and S. Willens. RFC 2058: Remote Authentication Dial In User Service (RADIUS), January 1997. Obsoleted by RFC2138 [141].

[141] C. Rigney, A. Rubens, W. Simpson, and S. Willens. RFC 2138: Remote Authentication Dial In User Service (RADIUS), April 1997. Obsoletes RFC2058 [140]. Obsoleted by RFC2865 [143].

[142] C. Rigney, W. Willats, and P. Calhoun. RFC 2869: RADIUS Extensions, June 2000.

[143] C. Rigney, S. Willens, A. Rubens, and W. Simpson. RFC 2865: Remote Authentication Dial In User Service (RADIUS), June 2000. Obsoletes RFC2138 [141].

[144] Li Ru, T. Braun, and G. Stattenberger. An AAA based Architecture for Providing Differentiated Services to Mobile IP Users. Technischer Bericht IAM-00-009, Institut für Informatik, Universität Bern, Schweiz, November 2000.

[145] V. Sander and M. Fidler. Evaluation of a Differentiated Services Based Implementation of a Premium and Olympic Service. In B. Stiller, M. Smirnow, M. Karsten, and P. Reichl, editors, *From QoS Provisioning to QoS Charging*, volume 2511 of *Lecture Notes in computer Science*. Springer Verlag, August 2002.

[146] S. Sato, K. Kobayashi, H. Pan, S. Tartarelli, and A. Banchs. Configuration Rule, Performance Evaluation of DiffServ parameters. In *17th International Teletraffic Congress (ITC17), Salvador da Bahia, Brazil*, December 2001.

[147] H. Schulzrinne, H. Tschofenig, X. Fu, J. Eisl, and R. Hancock. CASP - Cross-Application Signaling Protocol. Internet Draft, draft-schulzrinne-nsis-casp-00.txt, September 2002. work in progress.

[148] N. Seddigh, B. Nandy, and J. Heinanen. An Assured Rate Per-Domain Behaviour for Differentiated Services. Internet Draft, draft-ietf-diffserv-pdb-ar-01.txt, July 2001. work in progress.

[149] R. Serban, S. Gara, and W. Dabbous. Internet QoS Signaling Protocols. *IEEE Communications*, December 2000.

[150] J. Sevanto, M. Liljeberg, and K. Raatikainen. Introducing Quality of Service and Traffic Classes into Wireless Mobile Networks. In *Proceedings of*

First ACM International Workshop on Wireless Mobile Multimedia (WOWMOM 1998) Dallas, Texas, USA, 1998.

[151] S. Shende, A. Malony, J Cuny, K. Lindlan, P. Beckman, and S. Karmesin. Tuning and Analysis Utilities (TAU). Available from http://acts.nersc.gov/tau/main.html.

[152] W. Simpson. RFC 1853: IP in IP Tunneling, October 1995.

[153] J. L. Sobrinho and A. S. Krishnakumar. Quality-of-Service in ad hoc carrier sense multiple access networks. In *IEEE Journal on Selected Areas in Communications*, August 1999.

[154] H. Soliman, C. Castelluccia, K. El-Malki, and L. Bellier. Hierarchical MIPv6 mobility management (HMIPv6). Internet Draft, draft-ietf-mobileip-fast-hmipv6-06.txt, July 2002. work in progress.

[155] R. Srinivasan. RFC 1831: RPC: Remote Procedure Call Protocol Specification Version 2, August 1995.

[156] G. Stattenberger and T. Braun. Implementation and Configuration of a Linux Differentiated Services Router. Technischer Bericht IAM-00-010, Institut für Informatik, Universität Bern, Schweiz, 2000.

[157] G. Stattenberger and T. Braun. Providing Differentiated Services to Mobile IP Users. In *Proceedings of the 26th Annual IEEE Conference on Local Computer Networks (LCN'2001), Tampa, USA*, November 2001.

[158] G. Stattenberger and T. Braun. QoS Provisioning for Mobile IP Users. In H. Afifi and D. Zeghlache, editors, *Conference on Applications and Services in Wireless Networks, ASW 2001*, Paris, France, July 2001.

[159] G. Stattenberger and T. Braun. Performance of a Bandwidth Broker for DiffServ Networks. In *Fachtagung Kommunikation in Verteilten Systemen (KiVS), Leipzig, Germany*, February 2003.

[160] G. Stattenberger, T. Braun, and M. Brunner. A Platform - Independent API for Quality of Service Management. In *Proceedings of the IEEE Workshop on High Performance Switching and Routing (HPSR 2001), Dallas, Texas, USA*, May 2001.

[161] G. Stattenberger, T. Braun, M. Scheidegger, M. Brunner, and H. Stüttgen. Performance evaluation of a Linux DiffServ implementation. *Computer Communications*, 25(13), August 2002.

[162] F. Strauß. Script MIB Performance Analysis. *Simple Times*, 7(2), November 1999.

[163] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, third edition, 1999.

[164] http://www.switch.ch/network/map/SWITCHlanbb1.gif, May 2002.

[165] R. Szabó, T. Henk, V. Rexhepi, and G. Karagiannis. Resource Management in Differentiated Services (RMD) IP Networks. In *Proceeding of the International Conference on Emerging Telecommunications Technologies and Applications (ICETA 2001), Kosice, Slovak Republic*, October 2001.

[166] A. Talukdar, B. Badrinath, and A. Acharya. MRSVP: A Resource Reservation Protocol for an Integrated Services Packet Network with Mobile Hosts. Technical Report DCS-TR-337, Rutgers University, 1997.

[167] B. Teitelbaum and P. Chimento. Qbone bandwidth broker architecture. http://qbone.internet2.edu/bb/bboutline2.html, 2000. Work in Progress.

[168] A. Terzis, J. Krawczyk, J. Wroclawski, and L. Zhang. RFC 2746: RSVP Operation Over IP Tunnels, January 2000.

[169] A. Terzis, J. Ogawa, S. Tsui, L. Wang, and L. Zhang. A Prototype Implementation of the Two-Tier Architecture for Differentiated Services. In *Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium (RTAS99), Vancouver, Canada*, June 1999.

[170] A. Terzis, M. Srivastava, and L. Zhang. A Simple QoS Signaling Protocol for Mobile Hosts in the Integrated Services Internet. In *INFOCOM (3)*, May 1999.

[171] The Institute of Electrical and Electonics Engineers, Inc. (IEEE). *ANSI/IEEE Standard 802 Part 11: Wireless LAN Medium Access Control (MAC) and Pysical Layer (PHY) Specifications*, 1999. http://standards.ieee.org/getieee802/download/802.11-1999.pdf.

[172] The Institute of Electrical and Electonics Engineers, Inc. (IEEE). *ANSI/IEEE Standard 802 Part 3: Carrier Sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*, 2002. http://standards.ieee.org/ getieee802/ download/ 802.3-2002.pdf.

[173] S. Thomson and T. Narten. RFC 1971: IPv6 Stateless Address Autoconfiguration, August 1996. Obsoleted by RFC2462 [174].

[174] S. Thomson and T. Narten. RFC 2462: IPv6 Stateless Address Autoconfiguration, December 1998. Obsoletes RFC1971 [173].

[175] Jean Tourrilhes. Wireless Tools. Available online from http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html.

[176] P. Trimintzios, I. Andrikopoulos, G. Pavlou, P. Flegkas, D. Griffin, P. Georgatsos, D. Goderis, Y. T'Joens, L. Georgiadis, V. Jacquenet, and R. Egan. A Management and Control Architecture for Providing IP Differentiated Services in MPLS-Based Networks. *IEEE Communications Magazine*, May 2001.

[177] G. Tsirtis, A. Yegin, C. Perkins, G. Dommety, H. Soliman, and M. Khalil. Fast Handovers for Mobile IPv6. Internet Draft, draft-designteam-fast-mipv6-00.txt, June 2001. work in progress.

[178] N.H. Vaidya, P. Bahl, and S. Gupta. Distributed fair scheduling in a wireless LAN. In *Proceedings of ACM International Conference on Mobile Computing and Networking (MobiCom 2000), Boston, Massachusetts, USA*, August 2000.

[179] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. RFC 2904: AAA Authorization Framework, August 2000.

[180] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. RFC 2905: AAA Authorization Application Examples, August 2000.

[181] A. Weyland. Mobile-Controlled Handover in Wireless LANs. Master's thesis, University of Bern, 2001.

[182] A. Weyland, G. Stattenberger, and T. Braun. Mobile-Controlled Handover in Wireless LANs. In *IEEE Workshop on Local and Metropolitan Area Networks (LANMAN'02), Stockholm, Sweden*, August 2002.

[183] A. Weyland, G. Stattenberger, and T. Braun. User-Controlled Handover in Wireless LANs. In *IEEE Workshop on Applications and Services in Wireless Networks (ASWN 2002), Paris, France*, July 2002.

[184] B. Wijnen, D. Harrington, and R. Presuhn. RFC 2571: An Architecture for Describing SNMP Management Frameworks, April 1999. Obsoletes RFC2271 [71].

[185] B. Wijnen, R. Presuhn, and K. McCloghrie. RFC 2265: View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP), January 1998. Obsoleted by RFC2275 [186].

[186] B. Wijnen, R. Presuhn, and K. McCloghrie. RFC 2275: View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP), January 1998. Obsoletes RFC2265 [185]. Obsoleted by RFC2575 [187].

[187] B. Wijnen, R. Presuhn, and K. McCloghrie. RFC 2575: View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP), March 2000. Obsoletes RFC2275 [186].

[188] http://www1.worldcom.com/global/about/network/maps/europe/.

[189] J. Wroclawski. RFC 2210: The Use of RSVP with IETF Integrated Services, September 1997.

[190] A. Yegin, D. Funato, K. El-Malki, Y. Gwon, J. Kempf, M. Pettersson, P. Roberts, H. Soliman, and A. Takeshita. Supporting Optimized Handover for IP Mobility - Requirements for Underlying Systems. Internet Draft, draft-manyfolks-l2-mobilereq-02.txt, June 2002. work in progress.

[191] Z.-L. Zhang, Z. Duan, L. Gao, and Y. Hou. Decoupling QoS Control from Core Routers: A Novel Bandwidth Broker Architecture for Scalable Support of Guaranteed Services. In *ACM SIGCOMM 2000, Stockholm, Sweden*, August 2000.

[192] Z.-L. Zhang, Z. Duan, and Y. T. Hou. On Scalable Design of Bandwidth Brokers. Technical report, University of Minnesota, July 2000.

[193] Zhi-Li Zhang, Z. Duan, and Y. T. Hou. On Scalable Network Resource Management Using Bandwidth Brokers. In *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS 2002), Firenze, Italia*, April 2002.

[194] X. Zhao. Linux Mobile IP. http://gunpowder.stanford.edu/mip/.

[195] T. Zseby, S. Zander, and G. Carle. RFC 3334: Policy-Based Accounting, October 2002.

Curriculum Vitae

1973	Born on September, 22^{nd} in Regensburg, Germany
1980 – 1984	Elementary School Konradschule Regensburg
1985 – 1992	Albrecht Altdorfer Gymnasium Regensburg
1992 – 1993	Military Service
1993 – 1999	Study of Mathematics and Physics at the University of Regensburg, Germany
1999	M. Sc. in Physics
1999 – 2002	Research assistant and Ph. D. student at the Institute for Computer Science and Applied Mathematics, University of Bern