Project Number:	**IST-1999-10077**
Project Title:	**AQUILA** **Adaptive Resource Control for QoS Using an IP-based Layered Architecture**
Deliverable Type:	**PU – public**

Deliverable Number:	**IST-1999-10077-WP1.2-SAG-1201-PU-O/b0**
Contractual Date of Delivery to the CEC:	**June 30, 2000**
Actual Date of Delivery to the CEC:	**June 30, 2000**
Title of Deliverable:	**System architecture and specification for first trial**
Workpackage contributing to the Deliverable:	**WP 1.2**
Nature of the Deliverable:	**O – Other**
Editor:	**Martin Winter (SAG)**
Author(s):	**Gerald Eichler (DTA); Ralf Widera (DTA); Soritis Maniatis (NTUA); George Politis (NTUA); Petros Sampatakos (NTUA); Haris Tsetsekas (NTUA); John Karadimas (QSY); Giovanni Benini (SAG); Andreas Kirstädter (SAG); Axel Hof (SAG); Peter Schneider (SAG); Martin Winter (SAG); Falk Fünfstück (TUD); Anne Thomas (TUD); Zbigniew Kopertowski (TPS); Rafał Sobiczewski (TPS); Marek Dąbrowski (TPS); Wojciech Burakowski (WUT); Andrzej Bąk (WUT)**

Abstract:	This deliverable details the system architecture used for the prototypes in the first trial.
Keyword List:	AQUILA, IST, architecture, resource control, QoS

Executive Summary

This deliverable details the system architecture used for the prototypes in the first trial. The focus is on a single-ISP scenario. The architecture specifies the function split and the black-box behaviour of the components developed in WP 2.1. The deliverable is used for internal and external communication of the system architecture.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

The purpose of this document is to specify the system architecture of the AQUILA resource control layer. This includes basic principles as well as definition of the components, which make up the AQUILA architecture. The interaction of these components is described. The interfaces between the components are specified.

The document is structured as follows:

After this introduction, chapter 2 provides a general description of the architecture principles. This chapter also contains definitions of the terms used within this document.

Chapter 3 describes the overall structure and architecture. The components are identified and their general interaction is specified. The structure is summarised in a set of overall analysis diagrams.

Chapter 4 provides a functional specification and a coarse design of each component. The assignments of each component are identified. A first design class diagram defines the classes, that make up each component as well as the relationship and interaction among them.

Chapter 5 describes the interfaces. This chapter is divided into subchapters regarding internal interfaces, external interfaces and management interfaces. Internal interfaces are those between components of the resource control layer. External interfaces are interfaces from the resource control layer elements to the network elements (routers). Management interfaces are used to control and manage the resource control layer.

Chapter 6 contains an alphabetically sorted list of abbreviations used in this document.

Finally, chapter 7 lists the references to other AQUILA documents as well as to external papers, RFCs and drafts.

2 General Description

2.1 Approach

2.1.1 The Vision

The AQUILA project aims at the provision of Quality of Services features for end-users over the existing Internet. AQUILA will develop an architecture that allows end-users to have application sessions where the communication is of higher quality than nowadays, and to request for such session characteristics.

For that, the network will offer network services to the customers of the network and implement them internally by different traffic classes. Network services can be seen as products for which customers have to establish contracts, so-called Service Level Agreements (SLAs). When a contract is established, customers may use it by initiating QoS requests for different application requirements, different periods of time, different reservation scenarios, and different levels of guarantee.

It is the task of the end-user's equipment (in terms of a customer premises equipment – CPE) to map the needs of the end-user and of the application into corresponding network services. And it is the task of the network to map these requests for network services into corresponding traffic classes.

This two-level-mapping ensures a maximum of flexibility and scalability. While the network provider is free to design the mapping in a manner that a set of network services is internally translated into a set of traffic classes with specific parameters, the end-user is free to use different applications with more or less specific requirements. Therefore, both mapping processes are an essential part of the approach.

In general, the AQUILA project aims to develop a flexible, extendable and scalable Quality of Service architecture for the existing Internet. This will be done by considering existing QoS approaches such as Differentiated Services (DiffServ), Integrated Services (IntServ), and Multi Protocol Label Switching (MPLS) which are currently under discussion at the IETF.

In particular, the core network will be an enhanced DiffServ network providing several dynamically manageable traffic classes with specific QoS parameters, per hop behaviours, and other "guidelines" that realise different traffic handling for different network services to be requested. In the access networks, RSVP can be used as one possible solution to request for QoS. On the other hand, other solutions will be based on CORBA. Moreover, MPLS may be a solution to provide Virtual Private Network (VPN) awareness.

In order to provide QoS support for normal end-users, the overall approach is based on real Internet scenarios, where several (client and server) hosts[1] are connected via access nets (e.g. LANs) to a core network (e.g. a WAN) (Figure 2-1).

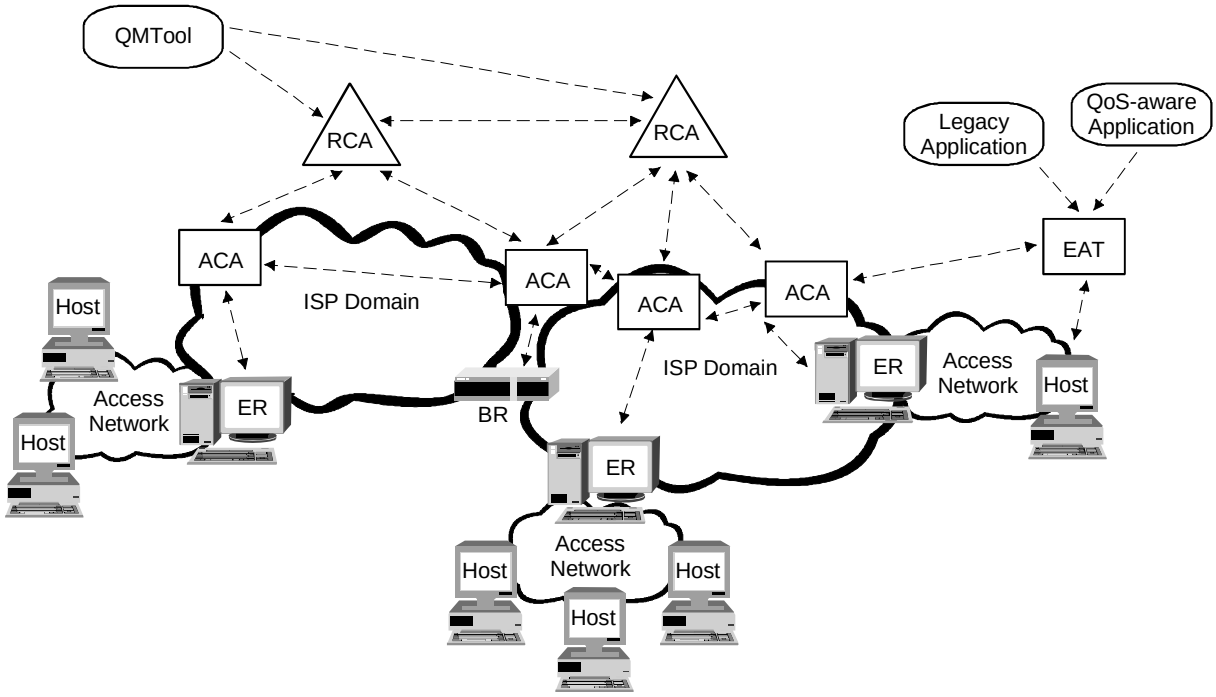


Figure 2-1: Overall network architecture

The core network consists of several Internet Service Provider (ISP) domains connected by so-called border routers (BRs). On the other hand, the access networks are connected to their domains by so-called edge routers (ERs).

A new layer will be developed on top of this physical architecture that is the Resource Control Layer (RCL). The task of this layer is: the control of the underlying network in order to provide QoS features to the customers of the network. The RCL mainly consists of:

- Resource Control Agents (RCAs) which are responsible for the control and the management of the overall resources of "their" DiffServ domains. Therefore they act as bandwidth brokers [RFC2638]. RCAs communicate with other RCAs to allow QoS requests over different domains, and with ACAs to share network resources for them. The standard configuration is to have exactly one RCA per ISP domain.

[1] From the network point of view hosts are clients as well as servers in terms of client/server applications, for instance.

- Admission Control Agents (ACAs) are managed by RCAs in order to perform local admission control and policy control for resources assigned to an edge router or a border router, i.e. there is one ACA per edge or border router. ACAs communicate with other ACAs to allow reservations across the domain. Moreover, ACAs communicate with EATs in order to distribute network services, to allow the negotiation of SLAs, and to process QoS requests initiated by the EATs.

- End-user Application Toolkits (EATs) are a kind of middleware between end-user applications and the network infrastructure. On behalf of end-users they request for network resources in order to support applications to get the proper QoS for their communication. Each EAT can communicate with its corresponding ACA (i.e. the ACA of the edge router of its access network) but is not aware of any RCAs.

Note that RCA, ACA, EAT components are not physical devices but logical components which may be placed anywhere within the network.

Further components of the architecture are legacy as well as new QoS-aware applications running on the hosts. Both will use the EAT middleware to benefit from the QoS capabilities of the AQUILA approach, i.e. the EAT is always the QoS portal to the RCL for them.

Moreover, the QoS Management Tool (QMTool) is a software for network operators providing access to the network. It allows the management of resources and services, the establishment and the maintenance of a distributed database (not shown in the figure above). This database is responsible to keep resource and service relevant information.

One important goal of the AQUILA approach is to ensure platform independence. Therefore, as many components as possible will be realised by using the Java programming language, while the communication between the components will probably be based on OMG's CORBA standard. However, existing reservation and policy control protocols will be considered. For example, RSVP could be used for QoS requests between the applications and the EAT as the frond end toolkit of the RCL. The EAT will intercept the RSVP requests and translate them into requests that fit to the AQUILA approach. In this way AQUILA will consider existing approaches on the market, e.g. the RSVP interface of Windows 2000.

Another decision concerns the use of the Unified Modelling Language (UML) for the analysis and the design of the AQUILA components. It may help to support the developers to have a common understanding of all components as well as to document the project results.

2.1.2 The Approach for the First Trial

In order to activate the development and to get the first results, the 1st trial approach will focus on some important issues but not consider all aspects of the overall approach. Particularly, while the overall approach considers inter-domain scenarios, the 1st trial approach focuses on a single domain scenario. Physically, the core network will consists of one administrative domain including network elements like core routers (CRs) and edge routers. Inter-domain re-

lated issues such as the control of border routers are out of the scope of the 1st trial (Figure 2-2).

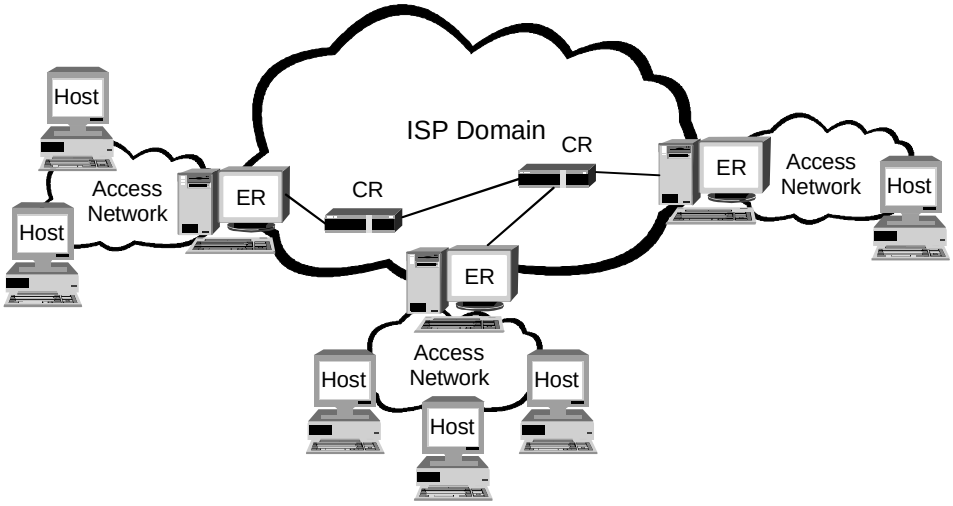


Figure 2-2: Physical architecture for the 1st trial

Therefore, the network architecture of the 1st trial approach is a subset of the overall approach architecture. The overlaying layer will include one RCA; a set of ACAs, one per edge router; and a set of EATs which supports legacy applications in the first stage (Figure 2-3).

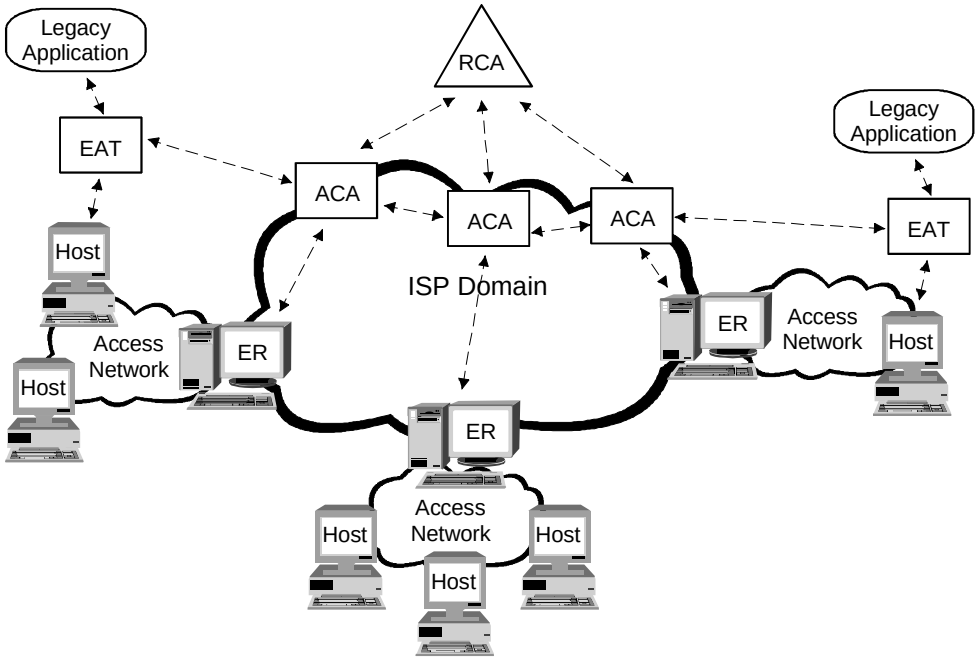


Figure 2-3: Network architecture for the 1st trial

In detail, the RCA manages the whole resources of the domain and assigns resources to the ACAs. Additionally, the RCA establishes the network services as well as the DiffServ traffic classes. Another restriction for the 1st trial is that these traffic classes are fixed, i.e. no dynamic reconfiguration is possible. It is the responsibility of the ACA to offer the network services to the EATs and to map them into traffic classes.

The EAT negotiates contracts with the ACA on behalf of the network. These SLAs are for network services. It is the responsibility of the EAT to map the end-user's and application session characteristics into network services corresponding to the existing contracts.

If a mapping is possible, the EAT requests for QoS by requesting for network services. A request can be made for different periods of time and different levels of guarantee. AQUILA will further offer the possibility to allow different reservation styles and modes as described in the next chapters. However, reservation can only be made for the core network. Access network reservations will be considered at most in the 2nd trial.

The EAT sends its request to the ACA that carries out a local admission control for the core network. (Admission control for the access network as well as policy control are out of the focus of the 1st trial.) It is the decision of the ACA whether the amount of locally managed resources can fulfil the request or whether the RCA has to be contacted. Another assignment of the ACA is to identify other ACAs in order to negotiate responsibilities. Finally, if the request is admitted or not, the EAT will be informed.

The approach, here roughly depicted, is described in more detail in the following chapters.

2.2 Definitions

The following glossary defines general important terms used throughout this document and the whole project.

Administrative Domain. A collection of network elements under the same administrative control and grouped together for administrative purposes. It is usually managed by a single corporate entity. For QoS enforcement purposes, a **network domain** refers to any domain that shares a common QoS policy. It may or may not overlap with other kinds of domains like IP or NT domains.

Admission Control. The process of determining whether a flow can be granted the requested QoS [Ferg98]. Admission Control is processed by the network and can be resource and/or policy based.

> **Local Admission Control.** Admission control based on locally managed resources and/or policies.

Admission Control Agent (ACA). A logical entity of the RCL. The ACA performs policy control and local admission control. There is a 1-1 relation between the logical entity ACA and the physical edge device.

Application. In terms of AQUILA an **end-user application** that uses a network (i.e. the Internet) for communication-based purposes. Such applications mainly consists of two levels: Firstly, the underlying user program, and secondly, the online service to be made available.

> **Legacy Application.** An end-user application which is not QoS-aware but can *indirectly* benefit from the QoS capabilities of the network.
>
> **QoS-aware Application.** In terms of AQUILA an application that benefits *directly* from the QoS capabilities of the network by using either an API of a QoS middleware or an appropriate signalling protocol such as RSVP. (Applications that use the API of the EAT middleware are called **EAT-based applications** in the following.)

Autonomous System (AS). A self-connected set of networks that are generally operated within the same administrative domain.

Border Router. See edge device.

Content. The multimedia data offered to users of an online service, e.g. a video in a Video-on-Demand service, dynamic financial information in an online banking service, etc.

Content Provider. Somebody who offers contents for online services.

Core Router. A router that is deployed at the core of an administrative domain.

Customer. An entity that purchases a specific network service. The customer acts either as an intermediate entity between the network provider and the end-user or as the end-user itself. In AQUILA a customer is equivalent to an end-user.

Customer Service. Depending on the customer being the end-user itself it is equivalent to end-user service or being an intermediate entity it is equivalent to network service. Not to be used for AQUILA.

Edge Device. A device such as a router or a gateway that is deployed at the border of an administrative domain. This can be an inter-domain border (then also called **border router**) or the border to the hosts. Two specialisations exist specifying whether the edge device belongs to the core (provider edge, **edge router**) or to the access side of a network (customer edge, access router).

Edge Router. See edge device.

Egress. The point where traffic *leaves* the network or the domain. The receiver is located at this point.

End-user. A person or a group of persons external to the network that utilises the network to work on a task, to offer something, etc., by using so-called end-user applications.

End-user Application Toolkit (EAT). A logical entity of the RCL. The EAT mediates between the user programs of the host and the ACA on the network.

End-user Service. See Service.

Flow. In terms of AQUILA a set of packets belonging the same application session.

Guarantee. The level of probability that an end-user gets the QoS he/she requested. While **hard guarantee** means the probability of 100%, a **soft guarantee** means a lower probability.

Host. Computer system on a network belonging to an end-user.

Ingress. The point where traffic *enters* the network or the domain. The sender is located at this point.

Link. A network communications channel consisting of a circuit or transmission path and all related equipment between a sender and a receiver. Most often used to refer to a WAN connection. Sometimes referred to as a line or a transmission link. In AQUILA the latter meaning is used, i.e. the connection from one hop to the next.

Middleware A software that occupies a position between an infrastructure (e.g. an operating system or a network) and applications , particularly in a distributed system.

Network Provider. An entity that controls a network infrastructure and offers network services. A provider can act as an **access provider** to prepare network access and/or as a **service provider** to offer network services with a specific behaviour, and perhaps to charge and account them.

Network Resource. The capacities of a network infrastructure to be shared between several utilisation. Main resources are bandwidth of links and buffers within routers, for example.

Network Service. A product that a network provider offers to its customer. In detail, it describes how customer's traffic is handled across the network as it is implemented by one or more traffic classes. Usually, there will be a set of pre-defined services but also the possibility to request for special parameters.

Online Provider. An end-user that offers online services via a network. An online provider may own a SLA with a network provider that allows the offer of different levels of quality for the online services.

Online Service. A product that somebody (e.g. an online provider) offers to another end-user or a group of end-users of any network. It requires user programs. Online services can be online and multimedia documents, client/server programs, or electronic commerce products, for instance, which may benefit from the use of network services.

Per Hop Behaviour (PHB). The forwarding treatment given to a specific class of traffic, based on criteria defined in the DiffServ field. Routers and switches use PHBs to determine

priorities for servicing various traffic flows [Star00]. There are currently two standard PHBs defined by the IETF: **Expedited Forwarding (EF)** [RFC2598] and **Assured Forwarding (AF)** [RFC2597].

Policy. (QoS Policy.) The binding of traffic recognition and registration profiles to specific network behaviours including, though not exclusive to:

- Admittance/denial of identified traffic getting anything better than best-effort QoS.
- Simple prioritisation or specific bandwidth reservation for identified flows or aggregated flows. [Cisco99]

Policy Control. The process of determining whether access to a particular resource should be granted.

Quality of Service (QoS). An overall measurement of the service quality based on certain key parameters [Black99]. QoS can be seen on two levels: In terms of end-user applications, it is expected to get the data in a sufficient manner with minimal delay or latency, minimal variations of delay (jitter), and error freeness.

In terms of a network, QoS is used to describe a connection, on which data are transmitted in a manner better than best-effort by using the network resources efficiently, and with minimal data loss.

> **QoS Scenario.** A use case where applications request for QoS in order to have a better service quality for their communication.

Request. (QoS Request). An explicit demand for getting QoS from an infrastructure. Usually, signalling protocols such as RSVP are used for the request. However, requests could be based on APIs and CORBA as well.

Reservation. Part of a resource that has been dedicated for the use of a particular traffic type for a period of time through the application of policies. [Star00]

> **Reservation Mode.** The assignment of the requester role. Three modes can occur: sender-initiated (forward reservation), receiver-initiated (backward reservation), as well as third-party-initiated.
>
> **Reservation Style.** The amount of senders/receivers involved in a reservation. Generally, there are three types: point-to-point (p2p; one sender, one receiver), point-to-anywhere (p2a; one sender, loads of receivers), and anywhere-to-point (a2p; loads of senders, one receiver).

Resource. See network resource.

Resource Control Agent (RCA). A logical entity of the RCL. The RCA controls resources and distributes them to the ACAs.

Resource Control Layer (RCL). An overlay network layer which monitors and controls the resources of the core DiffServ and access network as well as offers a QoS interface to the applications. The RCL consists of: ACAs, RCAs, and EATs.

> **RCL Platform.** Physical platform, on which one or several ACAs and/or RCAs are running. May be a separate hardware entity or integrated into a router.

Role. A host that participates on a network session can play the role of the **sender** of QoS traffic, or the role of the **receiver** of the QoS traffic. Additionally, the role of the **requester** which reserves for the QoS traffic can be played either by the sender, the receiver, or a third party.

Service. By default service is meant as synonymous to **end-user service**: A set of functions offered to an end-user by an organisation (service provider). From the network point of view, end-user services are products offered to the host.

However, these products are seen on two different abstraction levels from an end-user point of view: The end-user subscribes a set of network services, but chooses more abstract services by using his/her applications. *Due to these different views, it is proposed to avoid this term but to use the terms network services and session characteristics, respectively, instead.*

Service Level Agreement (SLA). A contract between a network provider and a customer defining provider responsibilities in terms of network services. In detail, a SLA includes QoS properties (throughput, loss rate, delays and jitter) of the network service and times of availability, method of measurement, consequences if network services aren't met or the here defined traffic levels are exceeded by the customer, and all costs involved. [D1101, Chapter 9.2, SLA]

Service Level Specification (SLS). A set of parameters and their values which together define the service offered to a traffic stream by a DS domain. Specific term for DiffServ. [D1101, Chapter 9.2, SLA]

Service Provider. See network provider.

Session. Time during which an application uses a network with QoS.

Session Characteristics. An end-user does have the possibility to individualise his/her applications in order to choose their session quality. For example: He/she can either choose between different pre-defined video qualities or directly set the parameters such as frame rate, picture size, etc. These characteristics have to be mapped into network services.

Subscriber. Used within the RCL to identify a customer that has been subscribed a SLA with the network provider.

Traffic Class. In terms of AQUILA the **implementation** of a network service, i.e. the network view on that product. A traffic class contains rules how to handle the traffic belonging to this class such as per-hop behaviour, rules for traffic conditioning as well as for admission control.

User Program. In terms of AQUILA a standard or special software that allows the use and the offer of online services. Typical examples for standard software are Web browsers like Netscape Communicator and Microsoft Internet Explorer as well as Web servers, FTP and E-mail programs as well as multimedia conferencing programs like Microsoft NetMeeting and MBone tools, etc.

3 Overall Structure and Architecture

This chapter should define and describe the overall architecture for the first trial of AQUILA. It should identify the main components and show their interaction.

3.1 Overview

The general approach of AQUILA is to achieve end-to-end Quality of Service (QoS) support on IP networks by providing an appropriate logical entity to manage and control the network resources. The architecture is divided in a data-plane that consists of a core DiffServ network and an overlay control-plane called Resource Control Layer (RCL) as depicted in Figure 3-1. The primary focus of this project is the control plane, which can be seen as a distributed Bandwidth Broker (BB) in the DiffServ architecture. Moreover, a detailed description of the provided services, the network architecture and some basic scenarios that clarify the interactions between the logical entities are delineated in this chapter.

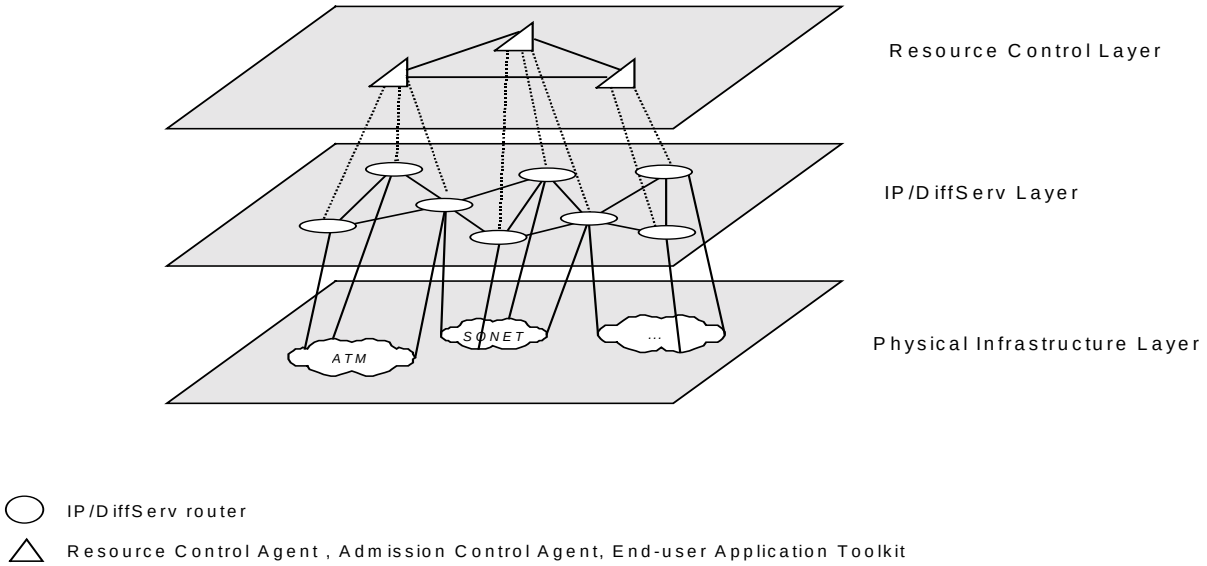


Figure 3-1: Layered architecture of AQUILA

The RCL is responsible for two major tasks: to control and monitor the network resources, and to perform policy and admission control in order to control the access to the network resources. Additionally, an interface to the end-user applications should be offered facilitating them for reserving network resources. In order to perform these tasks the RCL contains three functional components: the Resource Control Agent (RCA), the Admission Control Agent (ACA) and the End-user Application Toolkit (EAT, EAToolkit). These entities are hierarchically organised as depicted in Figure 3-2.

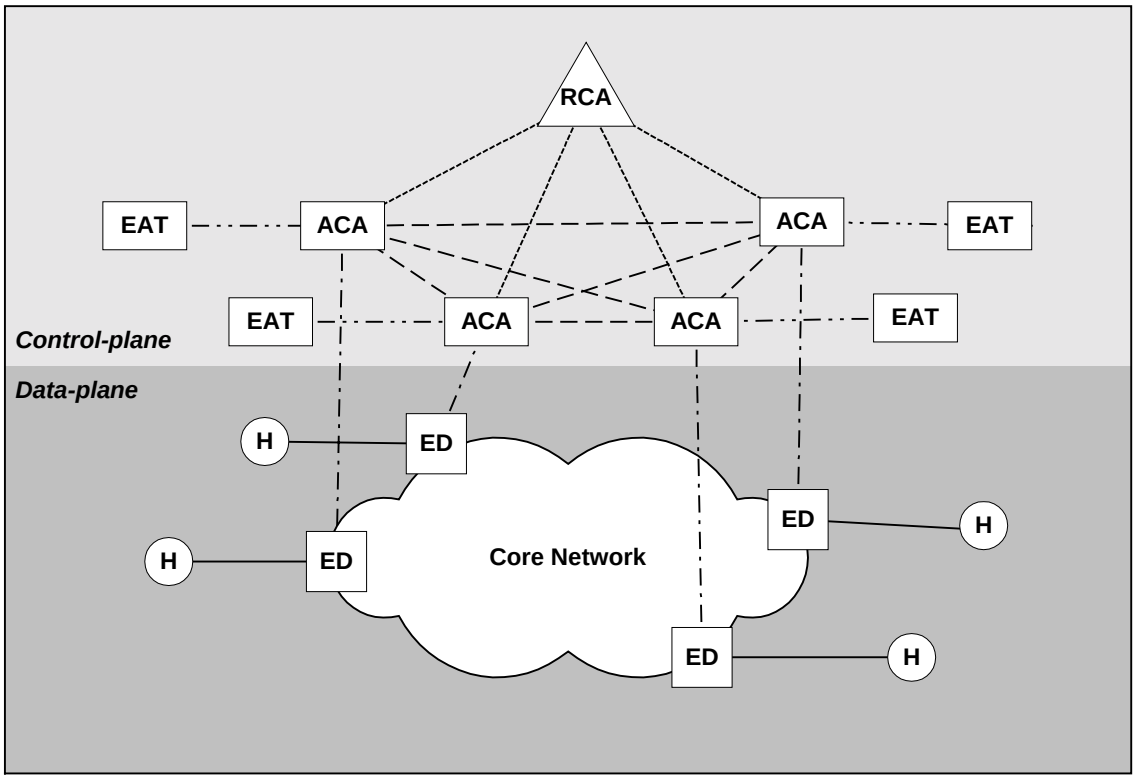


Figure 3-2: Resource Control Layer overview

Each administrative domain is controlled by one RCA, which is responsible for the control, distribution and monitoring of the network resources. The ACAs are responsible to control the access to the network, they perform policy and admission control at the edges of the core network. Each ACA controls one Edge Device (ED) or Border Router (BR). All the ACAs of a single domain are under the supervision of the RCA, which is the final authority of this domain. Finally, the EAT is the middleware that provides the interface of the QoS mechanism to the applications.

For the first trial a single DiffServ domain is assumed, while the inter-domain aspects are not covered in this specification.

3.2 Services

3.2.1 Network Services

An Aquila network offers a number of transport options for user IP traffic. These are called network services. Each network service provides a certain QoS, expressed by statistical statements about e.g. delay and packet loss. Also, for each network service it is exactly defined, which parameters must be passed in a reservation request requesting this network service, and what values are allowed for these parameters. As part of these parameters are used for policing the traffic, the traffic allowed to use a given network service is forced to obey certain characteristics.

The network services and their characteristics are defined by the network operator. They are made available by some kind of database to the other parts of the RCL. ACA and RCA only care about network services, not about specific applications and their demands. The latter is covered by the EAT. The EAT has access to the network service data, and on the other hand knows the application demands. It maps application demands to network services. So, a reservation request from EAT to ACA specifies in particular the requested network service.

3.2.2 Characterisation of a Network Service

The following information characterises a network.

- name or id identifying the network service
- requirements for requests

  - required reservation style (p2p or p2a) (i.e. is "any" as destination address allowed or not)

  - required traffic description

    Procedure: There is a fixed, common, "maximal" traffic descriptor, comprising at least the TOKEN_BUCKET_TSPEC of RFC 2215 (currently under discussion). For the individual network service, the following is defined:

    default values for all parameters

    parameters, for which explicit values can be specified

    ranges for values of the explicitly specifiable parameters

    (This description contains some redundancy, which has been left intentionally!)

  - required description of reservation time

    Procedure: analogous to traffic description

    Example 1 (preferred solution for the first Aquila trial): The default is "from now until explicit cancellation" and this default cannot be overridden.

    Example 2 ("specification of duration"): Start time and end time as the only parameters, with default values "from now on" and "until explicit cancellation", respectively. Start time default cannot be overridden, end time default can be overridden by a time/date value in minute granularity.

- QoS: statistical information about

  delay

  jitter

  loss probability / degree of bandwidth guarantee

packet ordering/disordering

Example: 95% of packets experience no queuing delay, at least 99% of packets are not discarded due to queue overflow, no packet disordering.

- accounting info: What is relevant for doing accounting for the usage of this network service. (Together with subscriber specific accounting information, from this service specific accounting info, it can be derived how the accounting will be done for this service for this subscriber.)

3.2.3 Implementation of Network Services

While the RCL knows about network services, routers don't. Routers know DSCPs, and they have scheduling mechanisms treating packets according to their DSCPs. (Edge devices also may look at other header fields in order to classify traffic.)

The idea is to implement each network service by using one or more DSCPs. The routers are configured in such a way, that their IP forwarding behaviour for such traffic results in the QoS defined for the corresponding network service.

3.2.4 Traffic Classes

The term traffic class shall be used to describe the implementation of a network service.

To allow the ACA to do admission control independent of traffic class features, each traffic class should provide rules how to add and to compare traffic descriptors, e.g. traffic class dependent formulas to compute an "effective bandwidth".

So the following characterises a traffic class:

- Per-hop behaviour
- Rules for traffic conditioning
- Rules for admission control
- Computation rules for traffic descriptors in this traffic class

3.2.5 Mapping of Network Services to Traffic Classes

The ACA performs the mapping from network service to traffic class. Therefore, the RCL may comprise a kind of database containing available

- network services
- traffic classes

- mapping rules (allowing to derive the traffic class from the requested network service plus other parameters of a request)

(This database will be also used to provide the available network services to the EAT.)

3.2.6 QoS Indicators

The IP header and higher level 4 headers include some fields which can be analysed for appropriate QoS handling decisions. Principally each defined bit pattern of an IP packet could be traced, but some fields of the IP header are more useful. They are marked grey in Figure 3-3.

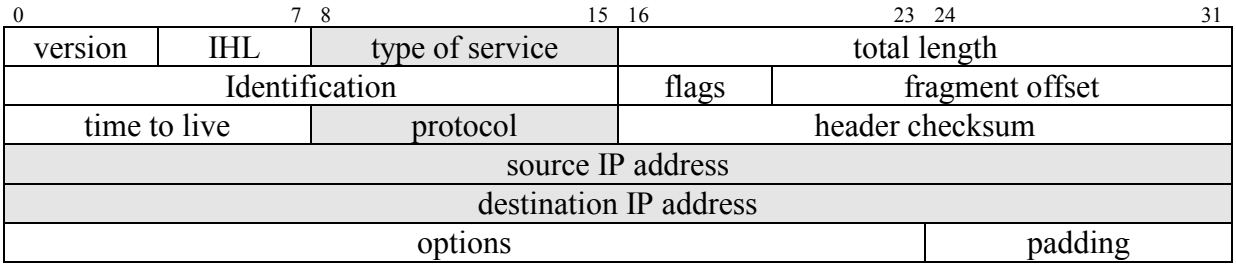


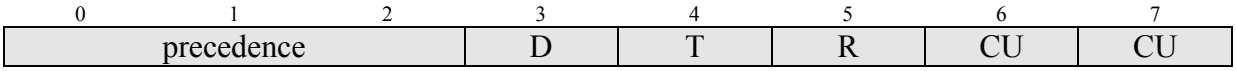
Figure 3-3: IPv4 Packet Header

Address based QoS Indication: Address based QoS indication allows to identify packets from a certain source and/or to a certain destination. This is only applicable if no address translation is performed. Beside single addresses also addresses of a sub-network will be supported.

Protocol based QoS Indication: On layer 4 IP packets use a protocol for transmission. This protocol is indicated within the IP header protocol field. Selective on the protocol, typically TCP or UDP appropriate network services could be selected.

Port Number based QoS Indication: The TCP and UDP header contain source and destination port numbers which typically specify the required service, e.g. TCP port 21 = FTP. Instead of single port numbers port number ranges are sometimes required.

Host Marking: Traditionally, the type of service (ToS) field was designed to carry information about precedence (3 bit) as well as delay, throughput and reliability (1 bit each) as depicted in Figure 3-4.



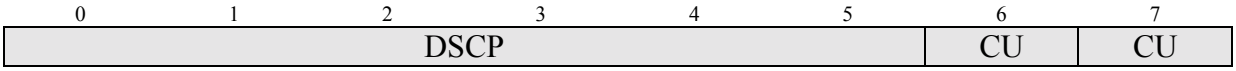
D ... delay T ... throughput R ... reliability CU ... currently unused

Figure 3-4: Traditional use of the Type of Service Field

Within an access domain the interpretation of the bit settings of the host is up to the EAT/ER. One possible exploitation is direct coding of the AQUILA service classes within the precedence 3 bits of the ToS field. As host marking allows different interpretation of the ToS bits,

an interpretation scheme must be associated with hosts using this method. Use of the remaining CU bit or address dependent interpretation are two possible solutions.

Differentiated Services Model (DiffServ): The traditional type of service approach has been changed several times and came up with the DiffServ model. The idea is to handle the first 6 bits together as Differentiated Services Code Point (DSCP), see Figure 3-5. The type of service byte is now called DS-Byte. A maximum of $2^6 = 64$ classes will be supported. The DiffServ approach is a special case of host marking.



DSCP ... Differentiated Services Code Point CU ... currently unused

Figure 3-5: DiffServ Interpretation of the Type of Service Field

Resource Reservation Protocol (RSVP): A more specific request is possibly given by using protocol based methods. The EAT in co-operation with the ACA will be able to handle simple RSVP requests. Requests using the PATH/RESV mechanism are terminated and generated locally within the access domain by the EAT within the AQUILA architecture.

Table 3-1 shows a systematised summary of QoS indicators related to header and protocol based indication on one hand and static and dynamic indicators at the other hand which are of interest for the project.

	Static indicators		Dynamic indicators	
	Host/Network sensitive	Session sensitive	During Session changeable	Per Packet changeable
Header based (implicit)	Source IP address Destination IP address IP address sub-network	Source protocol number Dest. Protocol number Source port number Dest. Port number Port number ranges		DiffServ Host Marking
Protocol based (explicit)			RSVP	

Table 3-1: Classification of QoS Indicators

3.3 Network Architecture

3.3.1 Overview

The general AQUILA network architecture is based on the DiffServ network concept. The objective of the project is to enhance the original DiffServ architecture by adding a new layer (RCL – Resource Control Layer) above the DiffServ network to provide dynamic access to

QoS network services. The RCL layer controls the distribution of network resources between different QoS control entities as well as the user access to the network. The underlying IP network provides means to realise the network services defined by AQUILA with assumed quality of service.

The overall AQUILA architecture consists of two functional areas: the data plane that is responsible for transmitting IP packets and practically implements the AQUILA network services and the control plane namely the Resource Control Layer that performs admission control and distributes resources between admission control entities.

The AQUILA transmission infrastructure is divided into access network and core network. The access and core networks can utilise different architectures. For example, in the access network where scalability issues are not of the main concern the IntServ network architecture can be used. The core network architecture is solely based on the DiffServ architecture as this solution provides for network scalability in terms of network size and capacity. AQUILA architecture distinguishes four types of network elements: Hosts, Edge Devices, Border Routers and Core Routers. The overall network scenario assumes that the user terminal (Host) is connected through the access network to the edge router (Edge Device) that provides access to the core network (see Figure 3-6) while Border Routers provide the access to other IP networks.

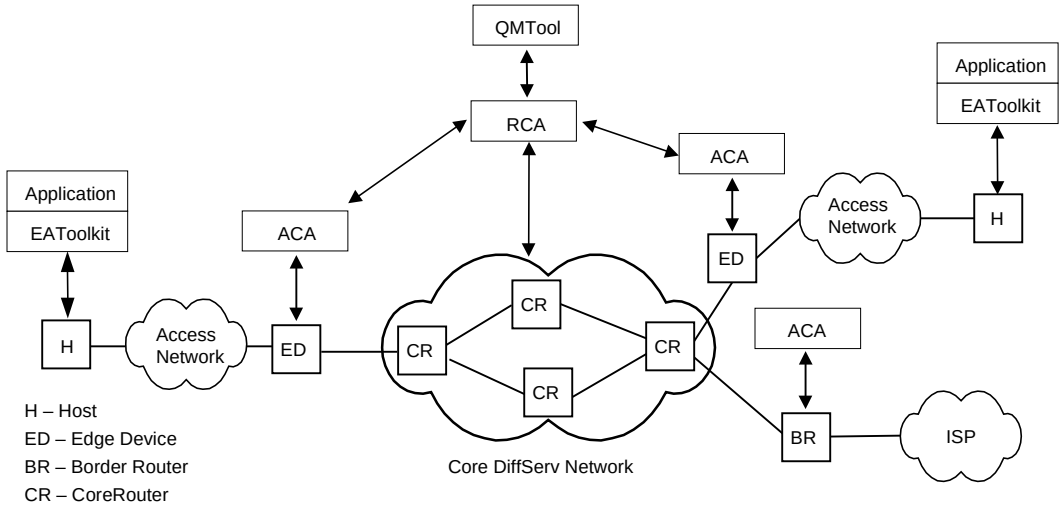


Figure 3-6: General AQUILA Network Architecture

3.3.2 Access Network Architecture

The access network connects user terminals (hosts) with edge devices (ED). If the reliability of access network is not of the main concern the tree-like structures are natural choice for access network topology. This type of access network topology is recommended for AQUILA projects in the first trial.

The access network has to provide adequate level of performance to not nullify the quality of AQUILA network services. The solutions that provide point-to-point resource reservation for packet streams between hosts and edge device should be preferred.

To provide required quality of service in access network the following approaches can be considered:

- The QoS guarantees are provided by layer 3 mechanisms e.g. IntServ, DiffServ, MPLS
- The QoS guarantees are provided by layer 2 mechanisms e.g. ATM, Frame Relay
- The QoS guarantees are provided by over-dimensioning e.g. dedicated Ethernet connections, Fast Ethernet etc.

The access network has to support wide range of access interfaces (in edge device):

- ATM
- Frame Relay
- Packet over SONET/SDH
- Ethernet/Fast Ethernet/Gigabit Ethernet

3.3.3 Core network architecture

The DiffServ is one among the many propositions to introduce the QoS into the IP networks. It provides scalable service differentiation in IP networks. *The service defines some significant service characteristics of packet transmission in one direction across a set of one or more paths within a network* [RFC2475]. These characteristics can be specified in quantitative (or statistical) or relative terms. The quantitative terms can relate to such parameters as throughput, delay, jitter or packet loss. Relative terms can describe the relative priority in access to network resources of IP packets using different services. However the service definition is not the part of DiffServ network specification. Instead this architecture provides rather the framework for defining network services.

The main requirement for AQUILA core network is scalability and reliability. Therefore the DiffServ architecture is adopted for AQUILA project. This model assumes that all per flow processing is performed at the network boundary (in AQUILA terms in the Edge Device). The Edge Device is thus the boundary node in the DiffServ terminology and implements such functions like traffic policing, marking, shaping and dropping. The Core Routers (interior nodes in DiffServ) should not be engaged with complex traffic processing (i.e. per flow processing). The core network can be divided into different administrative domain each controlled by its own RCA. However in the first trial only a single administrative domain is considered.

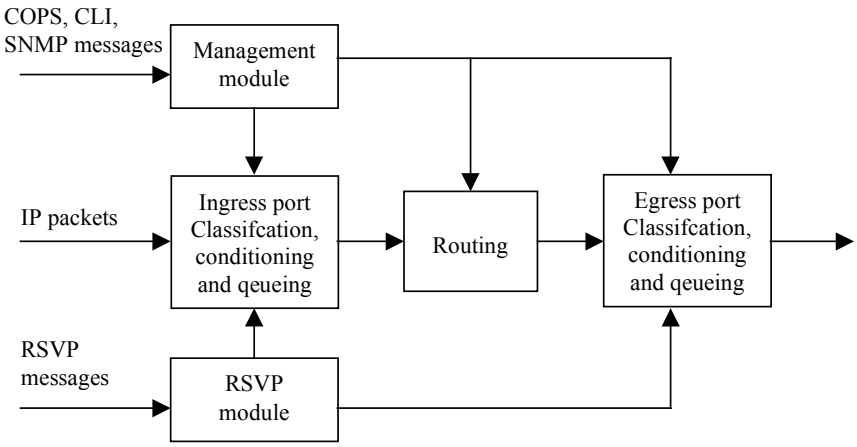


Figure 3-7: Router architecture

The general router architecture (edge device and core router) is shown in Figure 3-7. It consists of the following modules:

- Management module for configuration and management purposes. It should support the COPS, CLI or SNMP protocols for communication between RCL layer and network devices.
- RSVP module for compatibility with IntServ architecture. This is not mandatory for the AQUILA architecture, but can provide compatibility with RSVP aware applications.
- Routing module implementing routing protocol (recommended OSPF protocol).
- Ingress/Egress port modules. In the core router a Packet-over-Sonet/SDH (PoS) interface is recommended at physical and data-link layers.

The port module is composed of the following functional elements:

- Packet classifiers including both the MF Classifier as well as the BA Classifier,
- Traffic conditioning functions, including packet metering, marking, shaping and dropping for the traffic classes defined in AQUILA,
- Per-hop packet forwarding behaviours (PHB), implemented with the aid of scheduling and buffer management algorithms.

The packet handling in the forwarding path of the Edge Device (DiffServ capable boundary router) is depicted on Figure 3-8. The incoming packet stream is classified by the MF Packet Classifier into different packet flows. A flow represents a stream of IP packets that receives QoS guarantees and is therefore an analogue to a connection in the connection oriented packet networks, e.g. ATM. The packet classification function in the Edge Device selects IP packets based on the contents of the IP header (MF Classifier) and assigns them to different packet streams for further processing (conditioning and scheduling). The classifier module of the

Edge Device is configured by the ACA, which submits it with traffic filters constructed on the basis of user request for QoS services. Exactly one traffic filter is present in the Edge Device for each user request (being accepted by the ACA). The ACA constructs filters using the information contained in request messages send by the EAT.

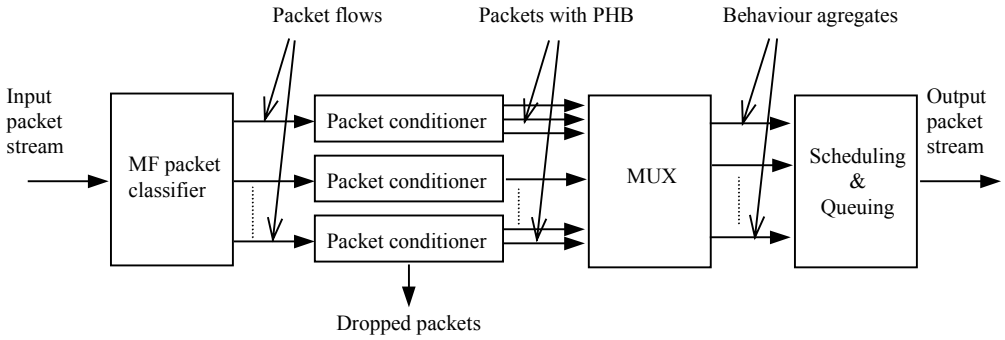


Figure 3-8: Packets forwarding in Edge Device

After passing classification step the packet flows are steered to traffic conditioning blocks (TCB). Each TCB block is configured according to the traffic class of a given flow. The TCB block ensures that the packet stream entering the AQUILA network conforms to the traffic contract. Traffic conditioners are typically deployed at the DS boundary node (in context of AQUILA architecture this function is deployed in the edge device and the border router). The traffic conditioner may mark/re-mark packets according to traffic contract and discard or shape packets to alternate the properties of packet flow. The traffic conditioner may consist of the following elements: meter, marker, shaper and dropper.

The following TCB blocks are required in the AQUILA network architecture for the first trial:

- Premium CBR – single token bucket with dropper
- Premium VBR – dual token bucket with dropper
- Premium Multimedia – single token bucket with marker
- Premium Mission Critical – dual token bucket with marker

The conditioner is instantiated and configured by the ACA for each accepted user request.

The traffic stream leaving the conditioner is marked with a specific code point or code points (PHBs). The packets marked with the same code point (independently of the original flow) are merged forming the so-called Behaviour Aggregate. Each Behaviour Aggregate is associated with the specific PHB that determines the forwarding treatment of its packets. Thus packets with the same DS code point will receive the same treatment in the network. The PHB are meaningful in relation to other PHBs (the performance of a single PHB depends only on the load of the link). The PHBs are implemented by means of scheduling and buffer management

algorithms. In the first trial the scheduling and queuing block parameters are administratively set by the network operator.

The packet handling in the forwarding path of the Core Router is depicted on Figure 3-9. The per-flow classification and conditioning functions are not present inside the network. So these mechanisms are not required in Core Routers. The incoming traffic stream is classified by a BA classifier (on the basis of the DSCP field) into flow aggregates that correspond to one of the four AQUILA traffic classes. The scheduling and queuing block implements buffer management and service disciplines required for each traffic class (this functional block is the same as in the Edge Device).

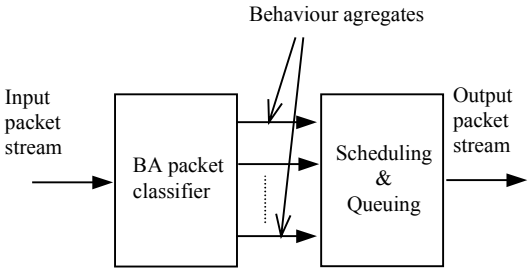


Figure 3-9: Packets forwarding in Core Router

The model of the scheduling and queuing block in the edge and core router is described in [D1301].

3.3.4 Routing protocols

IP routing protocols are divided into two groups: Interior Gateway Protocols (IGPs) and Exterior Gateway Protocols (EGPs). Interior Gateway Protocols perform routing in networks that are under common network management. The examples of IGPs are: Routing Information Protocol (RFC 1058, RFC 2453), Open Shortest Path First (RFC 2328), Interior Gateway Routing Protocol (Cisco) etc. Unlike IGPs, Exterior Gateway Protocols are used to exchange routing information between networks that do not share common administration (so called Autonomous Systems, AS). The most typical is the Border Gateway Protocol (RFC 1771).

During the designing process of a QoS IP network the existing routing rules should be taken into account. However it is not so clear, what kind of relation shall we have between QoS architecture and routing protocol. Two options are possible:

- Routing supports QoS mechanisms (QoS routing),
- Routing does not support QoS mechanisms.

In the first solution, routing should co-operate with other mechanisms, such as admission control and bandwidth allocation in QoS provisioning. Unfortunately, during the evolution of IP technology QoS provisioning was hardly taken into account. Quality of network services was squeezed out by the connectivity paradigm, which led to a specific design of routing and

switching mechanisms as well as protocol data unit formats. For that reason the second solution is for now the only one choice. It means that there is no direct interaction between routing and QoS provisioning, and it has to be done in some intermediate way. For example it is highly desired that QoS provisioning mechanisms are aware how the routing paths run. Similarly frequent route flaps should be avoided.

As it was stated before, one of most fundamental paradigms of IP technology design was connectivity – it was assumed, that the network will stay consistent in spite of any failure that do not lead to a division into separate, unconnected parts. To achieve such reliability the network offers only connectionless packet transfer service. For that reason each packet contains an explicit destination address that is the only information needed to find out its route through the network. It is assumed that the network nodes will route the packet on the basis of the destination address thanks to having properly configured routing tables.

Configuration of routing tables can be done in a static and dynamic manner. Static routing assumes that the network manager manually creates routing table entries. Typically such configuration takes place in small networks or at the network edge where nodes are single-connected. The full power of IP connectivity can be obtained using dynamic routing, when routing table entries are created dynamically through operation of special routing protocols. Routing protocols offer mechanisms allowing distributed topology discovery in order to achieve self-healing property in case of network failures and some kind of routing path optimisation.

From the point of view of a particular traffic flow, lacks of a common route as well as flow integrity are disadvantageous from QoS provisioning perspective. For flows belonging to some traffic classes, especially those requiring hard guarantees, the mentioned drawbacks could disturb packet transfer quality. The reasonable solution to this problem is to keep the route path with respect to given traffic flow as stable as possible. This stability requirement can be extended to the whole traffic class when per flow state is not maintained in core routers (e.g. DiffServ architecture). Assuming that routing is the same for all traffic classes, hard QoS requirements provisioning leads to more or less static routing in the whole network. Such situation could take place if routing mechanisms are not capable of distinguishing different traffic classes.

3.3.4.1 OSPF - Open Shortest Path First

The Open Shortest Path First routing protocol is based on advertising states of each link by the routers – for that reason it is called ‘link state algorithm’. Link state information is flooded through the network and on that basis each router maintains a database describing the Autonomous System’s topology. Further calculation of routing paths is based on performing Minimum Spanning Tree Algorithm on topology database.

In OSPF, link state contains metrics for routing path calculation. In the current version (ver.2) of this protocol the only simple additive cost metric is used. This link cost is assigned administratively and can be different for traffic classes characterised by the value of the ToS octet.

At the moment it is not consistent with DSCP interpretation, so it's application to DiffServ is implementation specific. So, this solution can be applied in AQUILA.

Additionally, the OSPF has three properties that can be very useful for AQUILA: it floods network topology information, allows splitting the AS into sub-areas, and can be managed by SNMP.

Flooding link state information allows the RCL to learn underlying network topology thus allowing some kind of dynamic configuration. Further topology information can be obtained through SNMP.

Splitting the AS allows grouping the hosts and nodes into areas, which maintain separate topology databases and perform their own interior routing. The topology of area is invisible from external areas, what offers reducing routing traffic and prevents inter-area traffic being routed by the backbone. Thus it fits well to the concept of RCL hierarchy and creation of resource pools – when some parts of the network are assigned to OSPF areas, information about their topology is not flooded through the rest of the network. In that case controlling such areas can be performed only through management mechanisms, e.g. SNMP, which is more difficult and more bandwidth consuming, than monitoring OSPF messages. For that it is reasonable to assign the areas to separate resource pools.

Thanks to standardisation of its MIB, network routing performed by OSPF can be modified by external management entities through SNMP. It allows changing link metrics for particular classes thus offering some control, i.e. by RCL.

3.3.4.2 QOSPF – QoS extensions to Open Shortest Path First

Recently the experimental RFC was published describing QoS extensions for OSPF. The most important changes comparing with the standard version of OSPF are the following:

- New ToS encoding is compatible with DiffServ DSCP interpretation.
- Two new link metrics were introduced: link available bandwidth and link propagation delay (the latter is used only for pruning high latency links)

Supporting QoS routing requires three main components: obtaining the information needed to compute QoS paths, establishing and maintaining the path for a given flow. Expressing link metrics in term of bandwidth will cause frequent link updates and, in effect, possible route changes. In order to satisfy QoS guarantees, routing paths calculated by QOSPF must be somehow maintained during flow lifetime. Unfortunately such maintenance is not possible in DiffServ without additional mechanisms such as MPLS. For that reason using QOSPF is not recommended for AQUILA architecture at least for the first trial.

3.3.4.3 Recommended approach

As it was stated in AQUILA, developing new routing protocols and appropriate mechanisms is out of scope of the project. For that reason we must assure, that network routing will not disturb QoS provisioning in developed architecture. This can be achieved in two possible ways:

- Using static routing for the traffic classes requiring hard QoS guarantees.
- Control routing by changing metrics used by dynamic routing protocol.

For the first trial we recommend to use static routing, at least for QoS provisioning, since it will allow better understanding how the other mechanisms work. Route changes lead to recalculation of admission control limits and reacting in appropriate way, when new limit is lower than needed to serve accepted reservations. Omitting routing information during calculating bandwidth limits could be very undesirable since it leads to uncertainty in the evaluation of AC mechanism – for example we cannot be sure what was the cause of bad AC decision: lack of network bandwidth or bad AC mechanism. For that reason we recommend to use static routing in the first trial (at least for traffic classes that require the hardest QoS guarantees) until we develop the RCA with full functionality.

Controlling the dynamic routing protocol (i.e. OSPF) is for further study.

Of course in real network static routing configuration is not desired and could be not feasible, so dynamic protocols should be used in target network.

3.3.5 Network Topology

3.3.5.1 Access network topology

Access to AQUILA network should not be limited to any specific solution. Therefore various technologies in the access have to be supported. Such a situation takes place in existing networks, where subscribers are connected by LANs, dedicated ISDN links, Frame Relay etc.

At the lowest level of network hierarchy usually the tree topology is used. The protection against failures at the access using multi-homing is used only in specific cases. The most important reason for that is significant cost of redundant links. Since the failure of an access link affects only one subscriber, it is his decision whether to protect the access or not and how the protection is realised. In many cases subscribers decide to use cheaper solution, e.g. dial-up ISDN backup links.

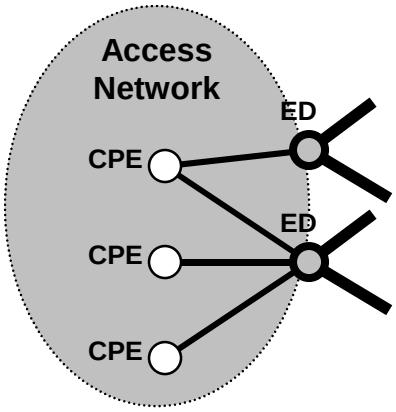


Figure 3-10: Typical access network topology.

3.3.5.2 Core Network Topology

The core part of AQUILA network could be decomposed into some sub-areas following the RCL architecture. These sub-areas correspond to the concept of resource pools managed by the RCA. Notice, that each sub-area should also handle its local traffic without using resources allocated to the higher levels.

The highest level is the backbone network, which connects all the regional networks thus offering full connectivity between all network subscribers.

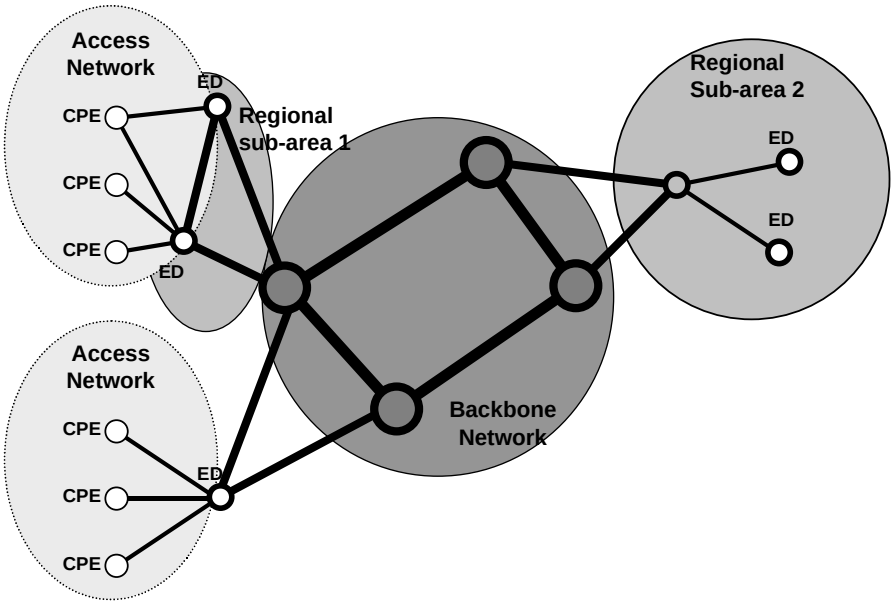


Figure 3-11: Example of network topology.

The core network consists of core routers connecting edge devices. Since full network has to provide full connectivity and reliability, the topologies used at these levels have to be characterised by at least 2-connectivity. The division of core network onto sub-areas is caused by the existence of resource pools created by the RCL, but also covers other aspects such as geographical location, routing hierarchy etc.

In case of the core network tree-like structures are not recommended because they do not provide the necessary reliability. The general candidates for this part of network are the ring or the mesh structures.

The ring is the simplest topology offering 2-connectivity, i.e. there are always two candidates for forwarding path. In case of a single link or node failure the topology still allows full connectivity. For that reason ring structures are widely used in data-link technologies such as FDDI or SDH. Unfortunately, IP does not directly benefit from the ring configuration – dynamic routing protocols perform the same topology discovery process without any assumption on underlying topology.

The mesh topology is typically used in IP networks. In general, the mesh structures can offer K-connectivity (where $K \geq 2$) and are very flexible in the realisation of network protection.

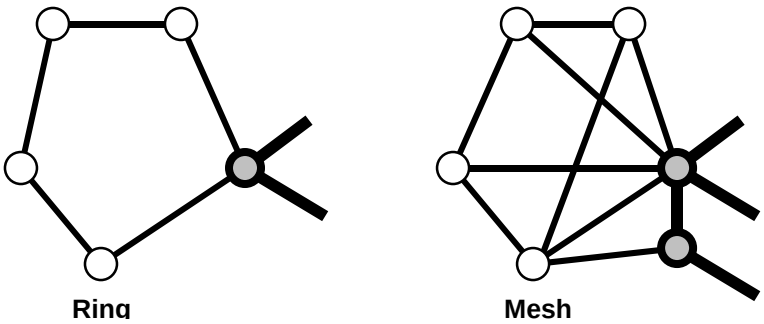


Figure 3-12: Examples of core network structures.

The limitations on feasible topologies are the following:

- The number of routers,
- Physical interfaces – type, number, number of ports in modules.

From the network survivability point of view the ED should be at least 2-connected. Three situations are possible:

- ED is connected to at least two CRs (2-connectivity with the core)

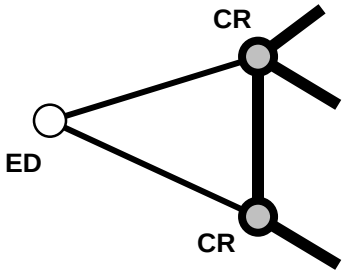


Figure 3-13: Single ED is directly connected to two Core Routers.

- ED is connected to one CR, but it is also connected to other EDs. In this situation, traffic is typically routed directly to the core, but in the case of any link failure, the other ED's links are used. Such EDs probably should be members of the same resource pool in order to avoid "stealing" the bandwidth. Inter-ED links could be also used to handle local traffic. Notice, that after the link failure the process of local bandwidth re-provisioning is required.

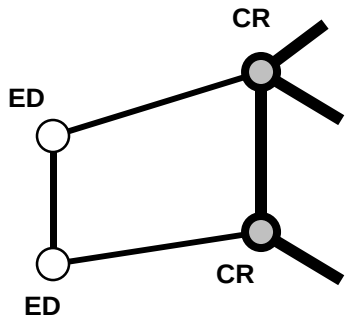


Figure 3-14: ED is connected to single CR, but in case of link failure it can route traffic through other ED

- Alternatively, EDs can be connected using both approaches.

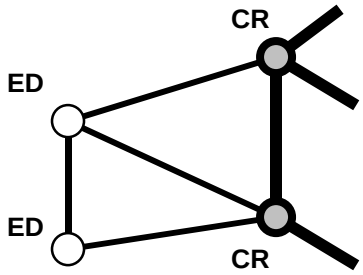


Figure 3-15: ED connected to 2 Core routers and to the other ED

3.3.5.3 Low bandwidth links

At present, access to IP core network is often provided by low-bandwidth links, using such technologies as ISDN, Frame Relay or synchronous private lines. Therefore it is inevitable, that the AQUILA network architecture should support such technologies. Low-bandwidth links (ranging from 64 to 1024 kbps) are always the problem in providing QoS guarantees, especially in case of interactive traffic, which is susceptible to increased latency. When the traffic generated in the subscriber's network is composed of large packets created by e.g. FTP applications, large queuing time of such packets could strongly influence delay-sensitive traffic generated by applications such as Voice-over-IP, Telnet etc. For that reason two additional mechanisms are used in order to solve delay problems:

- Link fragmentation and interleaving
- Header compression

Link fragmentation and interleaving is a method of splitting, sequencing and recombining large packet across low-bandwidth links. Arriving packets are classified and sorted into queues. Next, the large packets are fragmented to packets of small size and interleaved with

time-sensitive traffic by queuing discipline. Fragmenting and reassembling the packets is performed using a special encapsulation based on Multilink PPP protocol.

Header compression depends on reducing of number of bytes needed to carry IP, UDP/TCP and/or RTP headers by eliminating information that is not changing in each packet belonging to a particular flow and thus reducing bandwidth needed to carry such packets. In case of RTP traffic header compression allows reducing 40 bytes constituting IP, UDP and RTP headers to only 5 bytes. The decompressor located after the link is then able to reconstruct the original headers without any loss of information.

The existence of low-bandwidth access links has some serious impact on the AQUILA network architecture. First, the admission control should be performed before such links, because there is a high probability, that it will be the bottleneck on the forwarding path of the flow. Recall, that usually we cannot get significant multiplexing gain on low-bandwidth links with adequate QoS guarantees.

For that reason two-stage approach is recommended. At the first stage, admission control should be roughly performed at the customer access router (ED) in order not to exceed the bandwidth of the access link. The access router should be suitably configured in order to prioritise QoS traffic. The actual admission control is performed by the first Core Router.

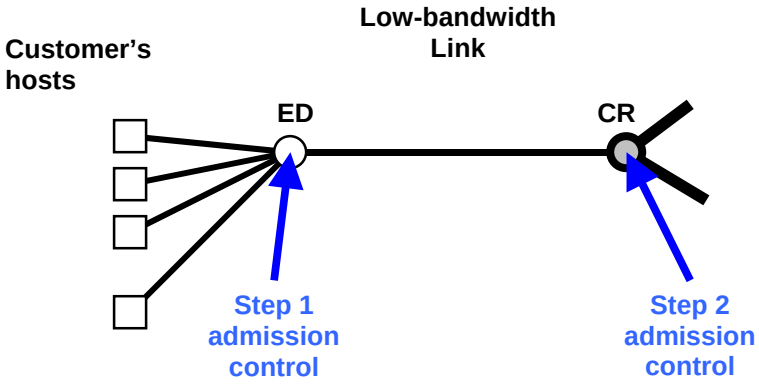


Figure 3-16: Access to the core network using low-bandwidth links

3.3.5.4 Topological Aspects of Hierarchical Resource Distribution

Taking into account hierarchical distribution of resources performed by RCA the most important question is when and where we create resource pools. One can expect that knowledge about network topology and routing should be helpful in solving this problem. First of all, we must assume some mapping between logical resource pool elements and elements of physical network. The following assumptions can be made:

- Resource pool maps into the set of physical links,
- Each link can belong only to one resource pool,

- ACA resource pool is mapped to ED (indirectly also to physical links connecting ED with the core, since there is a relationship between AC limit given and link bandwidth).

The following rules determine the creation of sub-areas:

- They should contain at least two EDs,
- The sub-areas' sets of links must be consistent.
- The sub-area should close its local traffic – traffic directed from one member ED to another member of the same pool should not cross the links that not belong to the given pool.

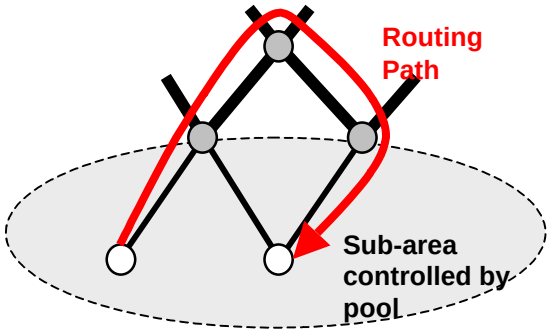


Figure 3-17: Example of disadvantageous situation when local traffic influences other pools

- The sub-areas of the same level of hierarchy should not be directly linked to avoid "stealing" resources one from another (see figure below). Such regions should create a single sub-area.

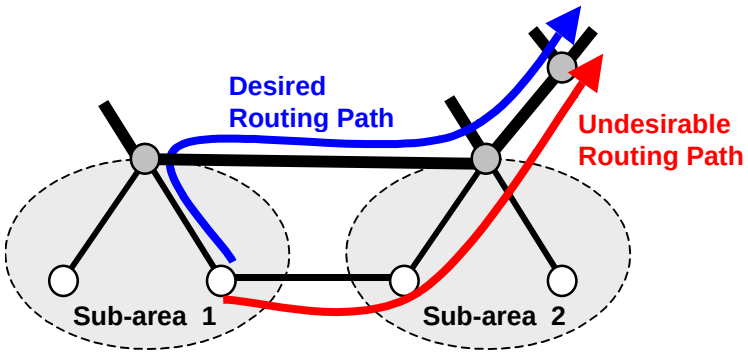


Figure 3-18: Example of undesired routing between two directly linked sub-areas.

Summarising the above discussion, we may conclude that the structure of the sub-areas should be organised in the hierarchical form following the resource pools structure. Each sub-area can be directly connected only to the higher or lower level, and as a consequence this leads to the tree-like structure (see Figure 3-19). EDs and BRs constitute the leafs of the tree. The

number of levels in this hierarchy can be chosen as needed, depending on real network topology.

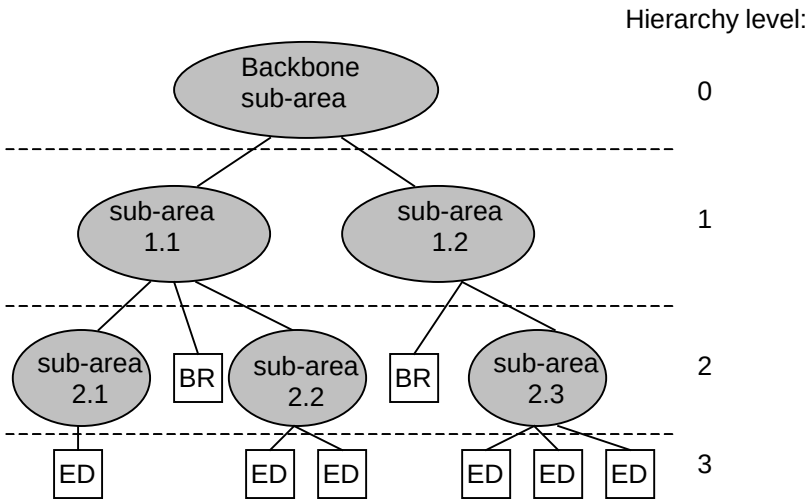


Figure 3-19: Exemplary hierarchical network topology

The ability to structure the network in this manner is unfortunately limited to strict hierarchical configurations. For instance, the full mesh topology, which is not of hierarchical type, does not fit well to such partitioning. Anyway, within a given sub-area the mesh topologies are preferable.

3.3.6 QoS Indication Architecture

While the core network handles all packets based on a limited number of AQUILA service classes autonomously, various methods within the access network are foreseen. Figure 3-20 mentions the involved components. The set of methods is extendable on-demand. Rules for the mapping decision onto a predefined AQUILA service class resulting from single or a combination of indicators have to be defined carefully later.

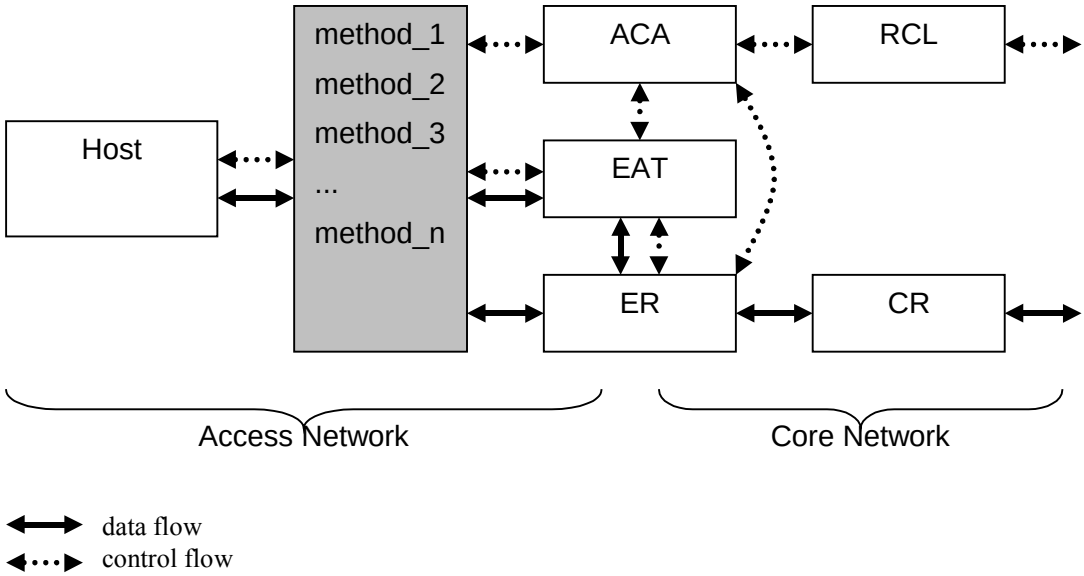


Figure 3-20: QoS Indication Handling

Independently of the used single method the handling will be terminated or generated locally within the access domain, using the RCL infrastructure (ACA, EAT). This approach implies the chance of inter-communication between hosts, using different QoS indication methods. Figure 3-21 shows an example where an RSVP sensitive Host A has a connection with Host B which triggers on protocol and port numbers.

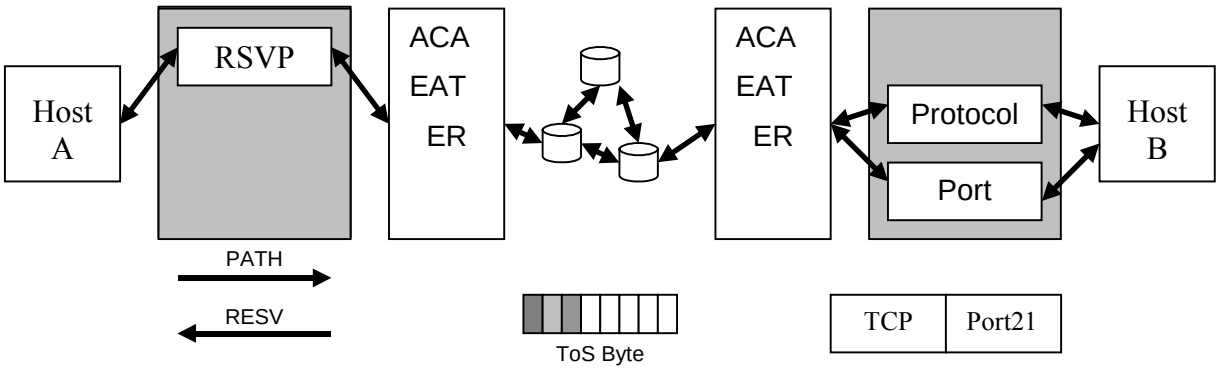


Figure 3-21: Mixed QoS Indication Handling

3.4 Resource Control Layer

The resource control layer (RCL) is an overlay network on top of the DiffServ core network. The RCL mainly has three tasks, which are assigned to different logical entities:

- To monitor, control and distribute the resources in the network. This task is assigned to the **resource control agent (RCA)**.
- To control access to the network by performing policy control and admission control. This task is assigned to **admission control agents (ACA)**. Each edge router or border router is controlled by an ACA. As each access request necessarily means usage of resources, the RCA may be directly or indirectly involved in handling admission requests.
- To offer an interface of this QoS infrastructure to applications. This task is assigned to the **end-user application toolkit (EAT)**. From the network point of view the EAT acts as a RCL front-end. From the user point of view, the EAT provides a QoS portal.

The entities defined above are associated to network elements within the underlying domain as shown in the following figure:

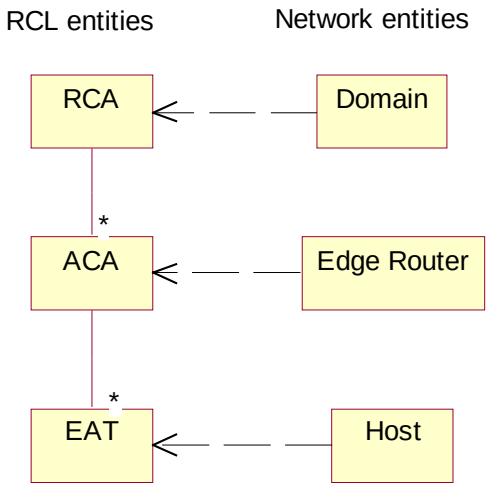


Figure 3-22: Mapping of RCL entities to the underlying network entities

Please note, that an EAT instance can be responsible for a single host as well as for a set of hosts. The latter might be the case, when not a single host, but a whole sub-network is connected to an edge router.

The resource control layer assumes an underlying DiffServ network. The DiffServ code points (DSCP) and the PHBs of this network are assumed to be predefined by management. They are not under control of the RCL for the first trial. For each traffic class (see chapter 3.2.4) however, there is a specific amount of bandwidth available in each link of each edge router, border router or core router. So bandwidth is the main resource, which is handled by the RCL.

In the first trial, the RCL implements a dynamic admission control by distributing the pre-configured, static resources of the core network among edge routers and border routers. In the second trial, dynamic reconfiguration of core network resources is also taken into consideration.

For the following discussion, a single DiffServ domain is assumed. Inter-domain aspects are not covered in this specification.

3.4.1 Admission control

A DiffServ network can only provide quality of service, if it is accompanied by an admission control, which limits the amount of traffic in each DiffServ class. Admission control checks, whether the resources requested from a user are available in the network and admits or rejects the request.

The AQUILA architecture uses a **local admission control** located in the ACA, which is associated with the ingress and egress edge router or border router. To enable the ACA to answer the admission control question without interaction with a central instance, the RCA will locate objects representing some share of the network resources nearby the ACA. Resources are assigned to these objects proactively. For the ACA, these objects represent a "consumable ResourceShare".

Admission control can be performed either at the ingress or at the egress or at both, depending on the reservation style as defined in chapter 3.4.4.

3.4.2 Resource distribution

The ACA will just allocate and de-allocate resources from its associated consumable ResourceShare. The ACA is not involved in the mechanisms used by the RCA to provide this resource share, to extend and to reduce it.

Resource distribution is performed on a per DiffServ class basis. In the first trial, there will be no dynamic reconfiguration of DiffServ classes. So, the resources of each class can be handled separately and independently of each other. This per class distribution however is not appropriate for edge devices, which are connected via small bandwidth links to the core network. In this case, additional mechanisms apply, which are described in chapter 3.4.3.

Resources are handled separately for incoming traffic (ingress) and for outgoing traffic (egress). The following description of resource distribution applies to both.

Resource distribution is performed by the RCA in a hierarchical manner using so called **resource pools**. For this purpose it is assumed, that the DiffServ domain is structured into a backbone network, which interconnects several sub-areas. Each sub-area injects traffic only at a few points into the backbone network. As described later, this structuring may be repeated on several levels of hierarchy.

When considering the resources in the backbone network, all traffic coming from or going to one sub-area can be handled together. So it is reasonable to assign a specific amount of bandwidth (incoming and outgoing separately) to each sub-area.

Depending on the topology of the backbone network, it may be useful to add some degree of dynamic to this distribution. The RCA may assign a larger bandwidth to one specific sub-area, when the bandwidth is reduced in other sub-areas. This dynamics may be described by the following formulas:

$$r_i \leq R_i$$
$$\sum_i r_i \leq R$$

where r_i is the resource limit actually assigned to ACA_i and R_i is an upper bound for this value. R is the overall limit of all resources distributed to all ACAs. These formulas express the following behaviour:

- The bandwidth assigned to each lower level entity r_i must not exceed an individual limit for this entity R_i. This limit R_i reflects the linkage of the lower level entity (e.g. sub-area) to the upper level entity (e.g. core network).

- The sum of the bandwidth assigned to all lower level entities must not exceed an overall limit R.

Depending on the values chosen for R_i and R, a more or less dynamic behaviour can be achieved.

Please note, that describing bandwidth with a single value (bits per second) is not sufficient in all cases. The characteristics of the traffic have to be taken into account. This may lead to an "effective bandwidth" formula, which is specific for each traffic class. It may also be necessary to describe bandwidth with a much more complex data structure, for which "addition" and "comparison" may be defined as rather complicated operations.

Resource shares are completely managed by the RCA. The resource share object itself is responsible to manage its resources and to check, whether a new bandwidth allocation request fits into the available bandwidth. If the amount of available bandwidth crosses some low-water-mark, the resource share object may precautionary request more resources from the resource pool. On the other hand, the resource share object will return unused resources to the pool.

Within a sub-area, there may be further subordinated sub-areas, which could be handled similar. Each resource share r_i assigned to a sub-area can be handled again as a resource pool R, which is distributed in a similar way among the sub-areas. Finally, resources can be used by ACAs as "consumable ResourceShare".

The depth of this hierarchical structure may be chosen as needed. It is also possible to mix several degrees of hierarchy, e.g. to break down the structure near edge routers more deeply than the structure of border routers, which are likely to be directly connected to the backbone.

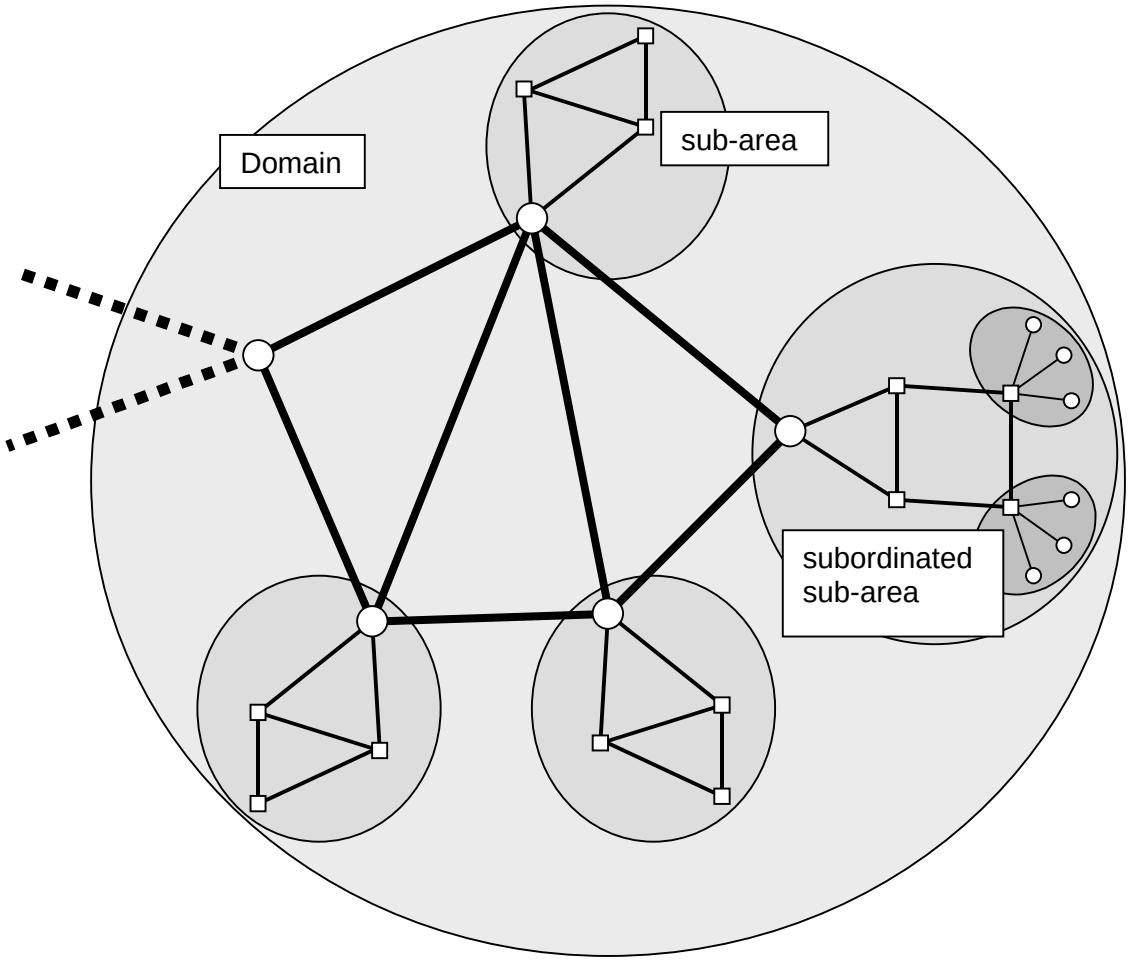


Figure 3-23: Hierarchical resource pools

The figure above illustrates this. It shows an example domain, which contains four sub-areas and one border router. In one of the sub-areas, the further division into subordinated sub-areas is illustrated.

Obviously, the ability to structure a domain like this strongly depends on the topology. In the access area of a network however it is likely, that tree-like structures exist, which enable the definition of such a structure.

3.4.3 Small Bandwidth Links

In typical network scenarios there might be edge devices, which are connected to the core network by links with a relatively small bandwidth (e.g. 256 kbit/s). In this case, the resource distribution scheme described above must be slightly modified.

It is not reasonable to split this capacity into different traffic classes, at it is done with larger bandwidth links in the network. Instead, there should be a mechanism, which assigns the bandwidth to those traffic classes, which request them.

Consider the following example network:

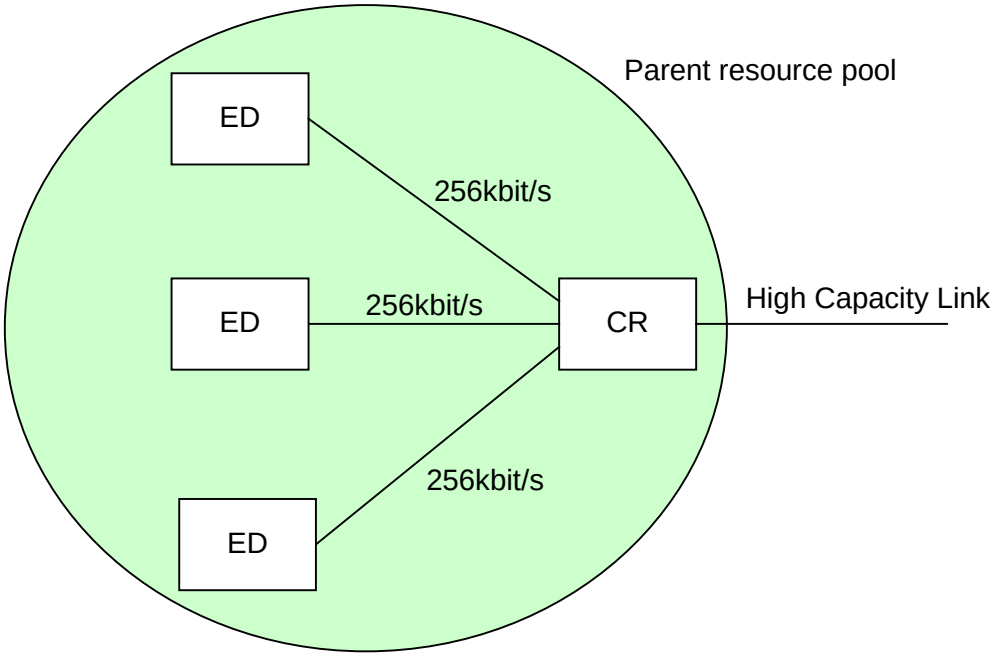


Figure 3-24: Example network with small bandwidth links

The parent resource pool covers the area of the network shown above. The children correspond to the resource shares assigned to the edge devices (respectively the ACAs related to the edge devices).

During initial assignment of resources, the parent will assign empty resource shares to the children, which have an upper bandwidth limit of zero. This is done, because any nonzero resource assignment would split the low bandwidth of the links into even smaller pieces, which may be unusable at all.

When an ACA gets a request, it will try to allocate resources from its associated resource share. According to the assignment described above, this request will not be successful at the first step. Instead, the child will try to increase its resource share to fulfil the request and will in turn request the appropriate amount of resources from its parent.

If the parent is able to fulfil the request, it will give the requested resources to the child, which then can “consume” them and admit the user request.

Likewise the child will return any resources released by the user to the parent. The child’s bandwidth cushion will in fact always be zero.

Up to now, the behaviour described here is nothing new. It is just the general resource pool algorithm. However some parameter values take on specific values (e.g. zero initial resources, zero bandwidth cushion).

To avoid unnecessary resource requests to the parent, the child should control and limit the sum of the requested resources for all traffic classes and reject a request, if it exceeds a certain limit, which depends on the bandwidth of the link to the core router. This is an additional functionality, which can especially support small bandwidth links.

With small bandwidth links it is especially important to consider, that bandwidth in general is not just a number, but should somehow include the characteristics of the traffic. As mentioned before, this affects the way, how bandwidth is added or compared. This problem exists on every level of the resource pool hierarchy and not only for small bandwidth links, but is a general behaviour of the resource "bandwidth".

One way to attack this problem is the use of an "effective bandwidth" formula, which takes into account the statistical multiplexing gain. The effective bandwidth however depends on the overall link capacity. So a value computed at some stage in the network (e.g. in the access) is not valid for other parts of the network (e.g. in the core). When the link capacities are large enough in some part of the network, then the effective bandwidth will not differ very much in this area and it might be sufficient to compute the effective bandwidth at the first resource pool, where a considerable number of flows is merged and so a considerable statistical multiplexing gain is achieved.

- For edge devices with a high capacity link to the core network this will be the child associated with this edge device's ACA.

- For edge devices with a low capacity link to the core network this will be the parent resource pool.

A better way would be to provide a formula for adding, subtracting and comparing traffic descriptors, so that the characteristics of the traffic is available at each stage in the resource pool hierarchy.

3.4.4 Roles

In a general QoS scenario, three different roles of actors can be defined:

- Requester: The requester sends the QoS request to the network. He/she determines, which service will be requested from the network and who may use it. The requester has to be authenticated to the network, because he/she will be charged for of the reservation. In AQUILA, the role of the requester is always played by the EAT.

- Sender: The sender injects the QoS traffic into the network. Admission control has to be performed for the sender. To control the injected QoS traffic, a policer and a marker has to be assigned.

- Receiver: The QoS traffic leaves the network at the receiver side. A reservation of network resources is also performed at the receiver side. A policer, however, is not necessary there.

Depending on the scenario, the sender and/or receiver is not always known. The requester role, however, must always be present. So, the following **reservation styles** may be implemented by this approach:

- p2p, point to point. The sender and the receiver are known.
- p2m, point to multipoint. The sender and a set of receivers are known.
- p2a, point to anywhere. Only the sender is known. QoS data may be sent to any destination.
- a2p, anywhere to point. Only the receiver is known. Traffic from any sender is prioritised.

Also depending on the scenario, the requester and the sender or the requester and the receiver may be located in the same host. This generates the following **reservation modes**:

- Sender oriented. The requester is identical to the sender.
- Receiver oriented. The requester is identical to the receiver.
- Third party. The requester is neither the sender nor the receiver, but a third party.

Reservation modes and reservation styles are independent of the requested network service. The admission control may however restrict the possible combinations of network service class and reservation style. E.g. a network service class providing guaranteed services should be restricted to p2p reservations.

The following figure shows a general third party p2p request. The request is initiated by the requester EAT and sent to its associated ACA, called manager ACA in the figure below. This manager ACA controls the process of reservation. First the manager ACA determines the ingress and egress ACA. For a discussion on the mechanisms used refer to chapter 3.4.5.2. Then the manager ACA calls the ingress and egress ACA to perform admission control on either side.

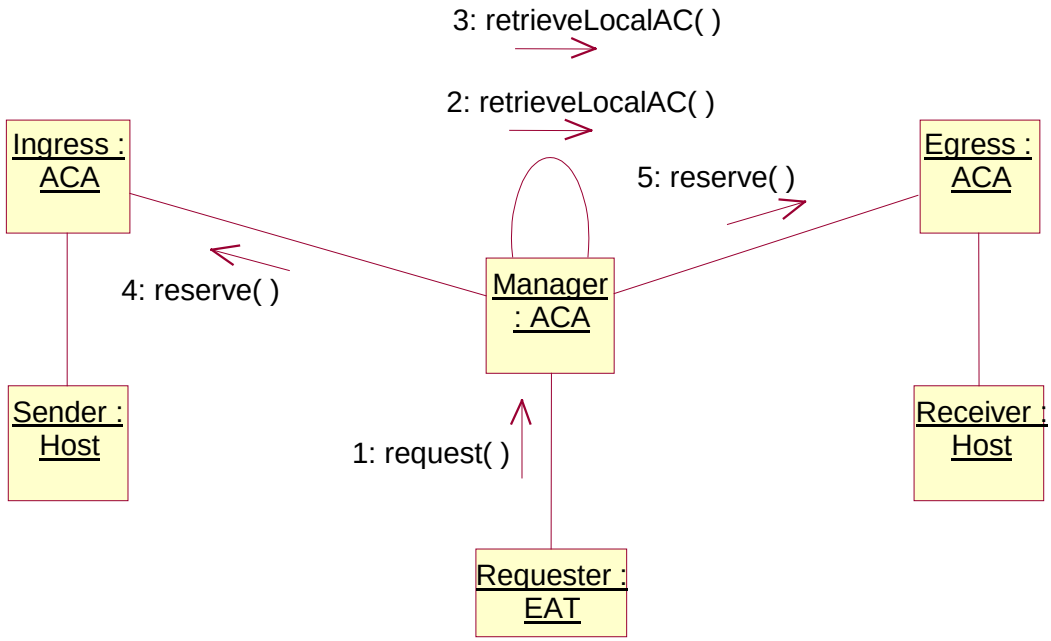


Figure 3-25: General third party p2p scenario

For sender oriented or receiver oriented reservations the ingress ACA or the egress ACA also takes the role of the manager. The scenario is then simplified as follows:

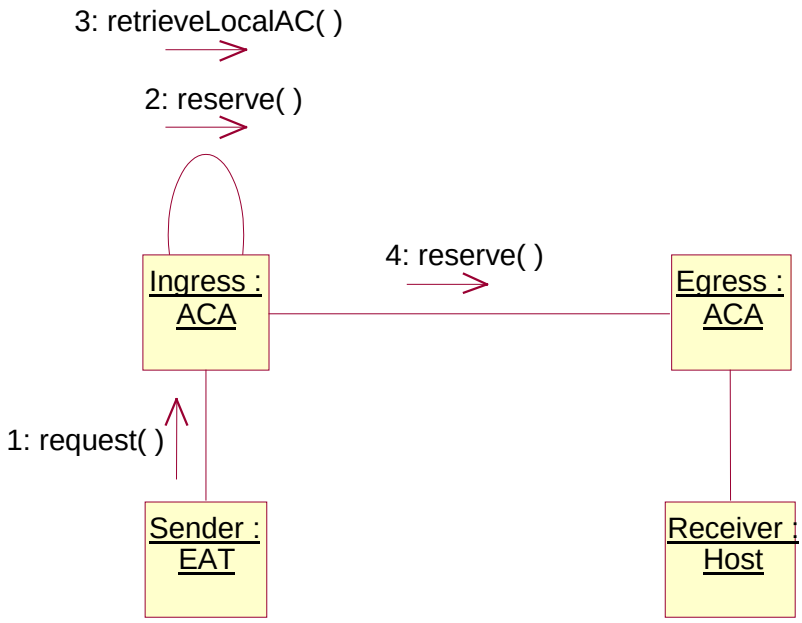


Figure 3-26: Sender oriented p2p reservation

For p2a or a2p style reservations admission control can only be performed either at the ingress (p2a) or at the egress (a2p). If such reservations are mixed with p2p reservations within one class, this leads to incorrect resource usage information at the network side, where no admission control can be performed.

To overcome this problem, one of the following solutions have to be chosen:

- Allow only a single style of reservations in a class. E.g. it may be useful to allow only p2p style reservations in a class, which is intended to provide hard guarantees.
- Use ingress (or egress) reservation only also for p2p requests, if they are mixed with p2a (or a2p) requests.

In p2m scenarios, the reservation is generally divided into two parts. First, the sender requests resources at the ingress of the network. The sender may the begin to send its multicast traffic. Still no receivers are known, so no egress reservation is performed.

With each receiver, which joins the multicast tree, an egress reservation is performed with the same QoS parameters as the sender. This egress reservation accounts for the traffic, which is generated at the forking points of the multicast tree within the network.

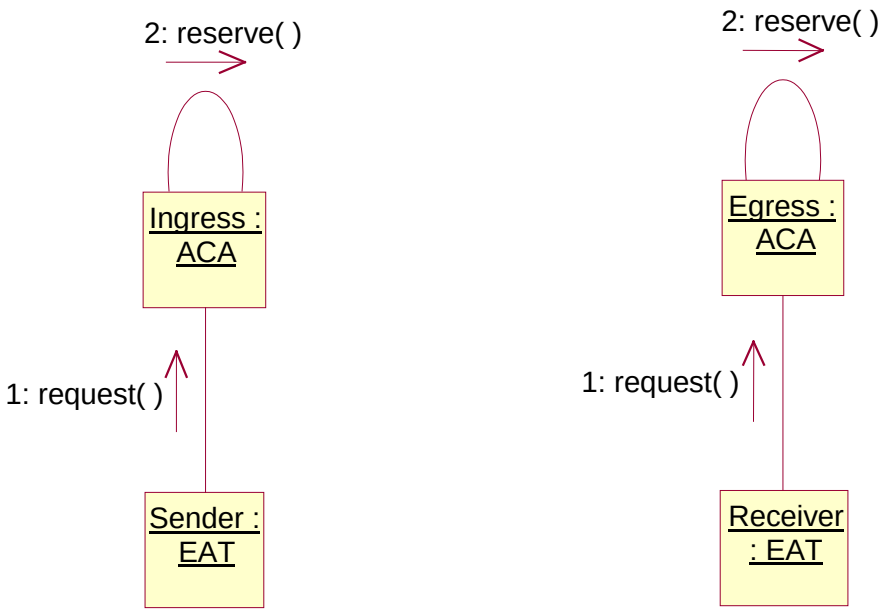


Figure 3-27: Multicast reservation

3.4.5 Communication

At the present stage, this chapter tries to identify the information, which must be available to make a QoS reservation and to specify requirements for communication between the entities within the RCL.

A basic requirement for all communication within the RCL is robustness against failure of single components. This implies, that either soft states with dynamic refresh or keep-alive mechanisms have to be used for all communication protocols.

3.4.5.1 ACA identification

When a new instance of an EAT is started, it has to identify its corresponding ACA. This may be performed by dynamic negotiation, similar to DHCP or PPP. For AQUILA, static configuration is also a possible choice.

Suggestion: Use static configuration for AQUILA in the first trial.

3.4.5.2 Finding ingress and egress ACA

To perform admission control, the ingress and/or the egress ACA has to be contacted, depending of the reservation style (p2p, p2a, a2p).

The requester generally does not know the ACA associated to the sender and receiver EAT. The requester however knows the addresses of the sender and/or receiver host. A mapping mechanism is needed to find the associated ACAs.

This mapping can be performed in two ways:

- By interaction between the hosts. This requires, that some part of the EAT is running at each host. If this condition is met, the approach also works in an inter-domain scenario. The requester EAT may contact the sender and/or receiver EAT and asks them for their associated ACA. A modification to this approach is, that the requester ACA instead of the requester EAT contacts the sender and receiver EATs.

- By a mapping mechanism implemented in the ACA. Each ACA may have access to a list of subnets connected to each ACA. From this list, the requester ACA can determine the sender and/or receiver ACA. For an inter-domain scenario, this approach requires, that the border router is taken as the egress point.

In the second approach, please note, that it is not the original task of the ACA to perform such a mapping. Another instance within the domain should be responsible to collect the necessary information and to provide a mapping service for the ACAs.

To decide between these two approaches, the following considerations have to be taken:

- Are we willing to assume, that at least some part of an EAT is running on the sender, receiver and requester in all cases? Are there possibly other arguments, which require an EAT at each side, e.g. information of the sender and receiver about the reservation?

- Is the EAT indeed always running on the host? The current EAT architecture also supports scenarios, where the EAT is running as some kind of proxy on an intermediate instance between the host and the ED.

- What would a mapping implemented in the ACA mean for administration and management of routing information?

- In a inter-domain scenario it is likely, that the reservation is split into several parts: An intra-domain reservation in the source domain (from sender to border router), an inter-domain reservation between source and destination domain (on an aggregated base) and an intra-domain reservation in the destination domain (from border router to receiver). It is unlikely, that the requester ACA has knowledge of the network configuration in both the source and the destination domain.

- To enhance security, all communication relevant for the reservation should be initiated by the ACA. This allows a more secure authentification of information.

Suggestion: Use the second approach (mapping in the ACA) for AQUILA. This enables scenarios, where an EAT is not available at the sender and/or receiver. This information should be stored in a common database or directory server.

3.4.5.3 Sender and receiver information

Especially with 3rd party reservations, but also with sender and receiver oriented reservations it may be required, that sender and the receiver are informed about the establishment and release of a reservation.

The requester is the role responsible for the reservation. So the requester is also responsible to inform the sender and receiver. It is obvious, that this is only possible, if an EAT is running at the particular side. There are two options:

- The requester EAT may inform the sender and receiver EATs

- The requester ACA may inform the sender and receiver EATs

Suggestion: As above, the requester ACA should do this.

3.4.5.4 Resource distribution

As described before, resources are distributed in a hierarchical manner. The RCA consists of a tree of resource pools. A resource pool is some entity managing some amount of resources. It receives resources (from somewhere) and/or distributes part or all of these resources to other resource pools.

(The question how to establish a suitable tree of resource pools is not covered in this chapter. Anyway, the root corresponds to the resources for the overall network, and the leaves correspond to the resources assigned to single edge devices. The resource pools in between may correspond to sub-areas etc. so that the resource pool tree reflects the hierarchical structure of the network.)

In the resource pool tree forming the RCA, the root resource pool receives resources by static provisioning, e.g. by providing a configuration file containing information about what resources are available for that pool.

All other resource pools are children within the tree and receive their resources from their parent pools.

All pools that are not leaves in the tree have children. A parent pool distributes resources to its children. Of course, a pool can distribute at most as much resources as it has received itself.

A leaf pool doesn't distribute the resources it has received. Instead, each leaf pool corresponds to an ACA, which "consumes" the pool's resources (for reservations requested by users.)

Each resource share represents a resource in a specific traffic class and a specific direction (ingress or egress). The resources each pool receives from its parent pool (or, in case of the root pool, by configuration) consist of up to two shares per traffic class, one for ingress reservation and, if this traffic class supports egress reservation, one for egress reservation. These objects are created during initialisation of the corresponding resource pool and live as long as that pool.

When a pool makes the initial assignment of resources to a child, it creates new resource shares, at most one per traffic class and direction. It passes these objects to the child. Of course, in each traffic class and direction, the sum of resources represented by the objects passed to children must not exceed the amount of resources the parent has originally received.

Example of a resource pool together with the resource shares this resource pool knows:

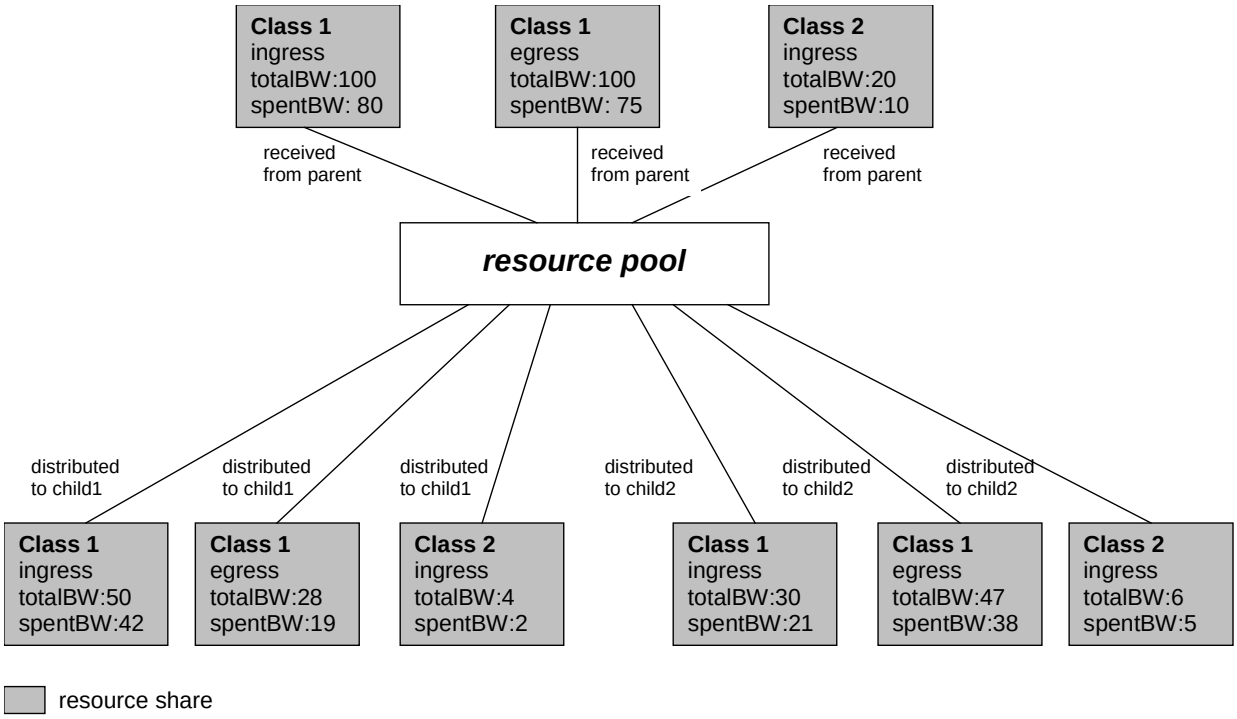


Figure 3-28: Example of a ResourcePool

To perform the resource distribution, a communication between a parent resource pool and its children is necessary. This communication may be initiated by both sides:

- The parent may proactively distribute resources among the children. This might be useful for an initial distribution of resources.

- The children may request or release resources from the parent. A resource request is triggered, whenever the remaining bandwidth within a child crosses some low-water mark or if the child is unable to fulfil a specific host resource request. The child then tries to increase its resource share, to fulfil the current and/or allow for further resource allocations.

  A child returns resources to the parent, if the remaining bandwidth crosses some high-water mark.

- The parent may specify the thresholds (high-water and low-water marks) of the resource allocation mechanism of the children. E.g. the parent may ask the children to limit their resource “cushion” to a specified percentage or amount.

Suggestion: All of these mechanisms should be implemented.

3.4.6 Deployment of logical entities

The previous chapters describe the RCL mainly on a logical level. For an implementation, however, a mapping of these logical entities to physical components has to be defined. Two opposing examples will show, how broad the possible range of mappings can be. For the AQUILA trial, it is likely that none of these extreme approaches is used. A good mixture of both will be the most suited mapping.

3.4.6.1 Mapping to associated physical entities

As defined in the foreword of chapter 3.4, there is a somehow natural mapping of logical to physical entities:

- The EAT is associated with the end-user host
- The ACA is associated with the edge router or border router
- The RCA is a more abstract instance, not associated to a specific network element.

This associations suggest, that

- The EAT is running on the end-user host
- The ACA is running on the edge router/border router or on a host closely related to this router.
- The RCA is running on a separate platform.

This is however not the only possible mapping. In a fully different approach, one may also define a single RCL platform, as described below.

3.4.6.2 Single RCL platform

The complete opposite of the previous approach is the single RCL platform approach. Here, all logical entities of the RCL are mapped to a single RCL host. This host runs all the EATs, the ACAs and the RCA.

End-user hosts communicate with their EATs e.g. by using CORBA, RMI, RSVP, HTTP or another protocol. The location of each component is fully transparent to their clients.

The architecture described in this document allows the full range of mappings as shown above.

3.5 Applications

3.5.1 Overview

This chapter contains descriptions of existing internet applications that can be used as a base for the AQUILA development and specifications for business and residential users software platforms. The main focus of this chapter is the specification of a selection of integrated RSVP-aware internet applications and other legacy applications:

- improved MBONE tools based on the functionality of *vic* and *vat*
- WWW/FTP applications for browsing and file transfer
- QoS applications in Windows 2000
- Toolkits (API's) used for implementation of QoS – aware applications

3.5.2 Operating Systems and QoS

The AQUILA implementation approach must be Operating System independent but as a lot of Operating Systems vendors are integrating QoS functionality in their Operating Systems we will give a brief description of the QoS support for the most common Operating Systems in the market :

- Sun Sparc (SunOS 4.x and Solaris 2.5, X11R6)
- Linux 2.1.90 or later
- Windows 2000

3.5.2.1 Sun Sparc QoS Support

Solstice Bandwidth Reservation Protocol 1.0 is a toolkit based on the standard network control protocol RSVP (ReSerVation Protocol), which allows Internet / intranet applications to reserve special quality of service for their data.

When an RSVP-enabled multimedia application receives data for which it needs a certain quality of service, it sends an RSVP request back along the data path, to the sender application.

At each stage along the route, the quality of service is negotiated with the routers or other network components. Non-RSVP routers simply ignore RSVP traffic and take no part in the negotiation.

Benefits

- Solstice Bandwidth Reservation Protocol inter-operates with RSVP-enabled applications and network devices, such as routers from other vendors, to ensure that resources are reserved from the emitting to the receiving device, right across the connection.
- An autostart/autostop feature stops Solstice Bandwidth Reservation Protocol automatically when it is no longer needed and restarts it as appropriate - saving memory and resources.
- Several levels of troubleshooting and logging features, which can be configured on-line, enable easy debugging and speed up the development of applications based on Solstice Bandwidth Reservation Protocol.
- A simple, robust command line interface makes administration easy.

Product Components

Solstice Bandwidth Reservation Protocol has two components:

- An API for development of RSVP-enabled applications.
- A run-time environment for cost-effective deployment of RSVP-enabled applications.

Solstice Bandwidth Reservation Protocol 1.0 implements the Proposed Standard RSVP protocol as defined in RFC 2205, and its application to Integrated Services as defined in RFC 2210.

3.5.2.2 Linux QoS Support

Linux, a shareware operating system, supports a number of advanced networking features, thanks largely to the huge Linux networking community. Besides the reliable TCP/UDP/IP protocol suite, a number of new features like firewalls, QoS, tunnelling etc. has been added to the networking kernel.

The support for quality of service is available from Linux kernel versions 2.1.90. However, the support is more comprehensive in the more recent kernels. This document is written with reference to the kernel version 2.2.1. This kernel also has support for differentiated services in the form of a patch. This patch needs to be applied in order to exercise all the QoS features supported in Linux.

The basic principle involved in the implementation of QoS in Linux is shown in Figure 1. This figure shows how the kernel processes incoming packets, and how it generates packets to be sent to the network. The input de-multiplexer examines the incoming packets to determine if the packets are destined for the local node. If so, they are sent to the higher layer for further processing. If not, it sends the packets to the forwarding block. The forwarding block, which may also received locally generated packets from the higher layer, looks up the routing table and determines the next hop for the packet. After this, it queues the packets to be transmitted on the output interface. It is at this point that the Linux traffic control comes into play. Linux

traffic control can be used to build a complex combination of queuing disciplines, classes and filters that control the packets that are sent on the output interface.

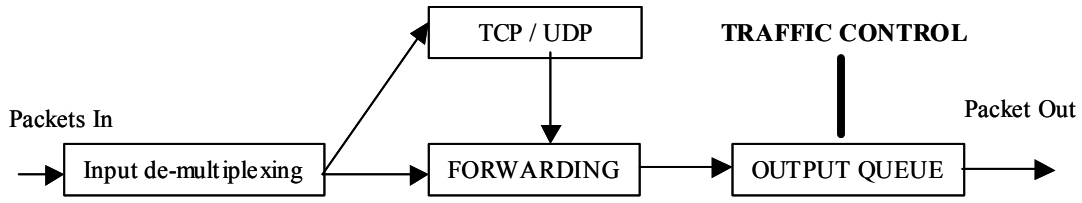


Figure 3-29: Linux Traffic Control

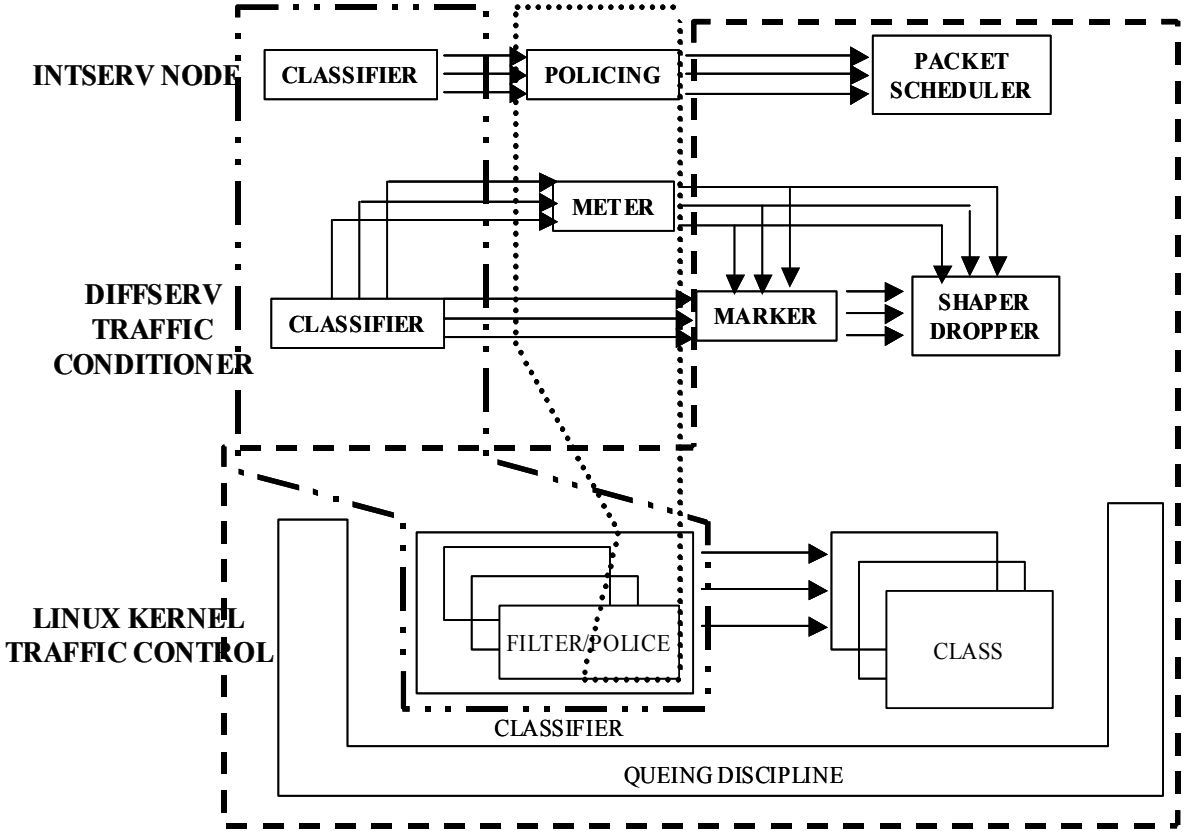


Figure 3-30: Framework for developing "intserv" and "diffserv"

As shown in the above figure, the QoS support in Linux consists of the following three basic building blocks, namely:

- Queuing discipline
- Classes
- Filters/Policer

Queuing disciplines, form a basic building block for the support of QoS in Linux. Each network device has a queue associated with it. There are 11 types of queuing disciplines that are currently supported in Linux, which includes:

- Class Based Queue (CBQ)
- Token Bucket Flow (TBF)
- Clark-Shenker-Zhang (CSZ)
- First In First Out (FIFO)
- Priority
- Traffic Equaliser (TEQL)
- Stochastic Fair Queuing (SFQ)
- Asynchronous Transfer Mode (ATM)
- Random Early Detection (RED)
- Generalised RED (GRED)
- DiffServ Marker (DS_MARK)

Queues are identified by a handle <major number: minor number>, where the minor number is zero for queues. Handles are used to associate classes to queuing disciplines. Queuing disciplines and classes are tied to one another. The presence of classes and their semantics are fundamental properties of the queuing disciplines. In contrast, filters can be arbitrarily combined with queuing disciplines and classes, as long as the queuing disciplines have classes. Not all queuing disciplines are associated with classes. For example, the Token Bucket Flow (TBF) does not have any classes associated with it.

Each class owns a queue, which by default is a FIFO queue. When the enqueue function of a queuing discipline is called, the queuing discipline applies the filters to determine the class to which the packet belongs. It then calls the enqueue function of the queuing discipline that is owned by this class.

Filters are used to classify packets based on certain properties of the packet, e.g., ToS byte in the IP header, IP addresses, port numbers etc. It is invoked when the enqueue function of a queuing discipline is invoked. Queuing disciplines use filters to assign the incoming packets to one of its classes. Filters can be maintained per class or per queuing discipline based on the design of the queuing discipline. Filters are maintained in filter lists.

The interface between the kernel and the user space is achieved using netlink sockets.

'tc' (traffic controller) is the user level program that can be used to create and associate queues with the output devices. It is used to set up various kinds of queues and associate classes with each of those queues. It can also be used to set up filters based on the routing table, u32 classi-

fiers, tcindex classifiers and RSVP classifiers. It uses netlink sockets as a mechanism to communicate with the kernel networking functions.

3.5.2.3 Windows 2000 QoS Support

Quality of Service in Microsoft® Windows® 2000 is a collection of components that enable differentiation and preferential treatment for subsets of data transmitted over the network. QoS components constitute the Microsoft implementation of Quality of Service.

Quality of Service is a defined term that loosely means subsets of data get preferential treatment when traversing a network. QoS technology has implications for the application programmer and the network administrator.

Windows 2000 Quality of Service achieves this through programmatic interfaces, the cooperation of multiple components, and communication with network devices throughout the end-to-end network solution.

3.5.2.3.1 Windows 2000 Quality of Service Defined

The Microsoft implementation of Quality of Service enables developers to use generic Windows Sockets 2 calls to create QoS-enabled applications. With the Windows 2000 QoS capabilities, developers do not need to consider how the various operating system components interact to achieve quality of service. The components that constitute QoS implementation are instead abstracted from the QoS application development effort, allowing a single or generic QoS interface — instead of individual interfaces — for each QoS component. This provides a generic interface for the developer, and also provides a mechanism by which new QoS components (perhaps with increased functionality) can be added, without the need to completely rewrite existing QoS applications

3.5.2.3.2 How Windows 2000 QoS Works

To achieve manageable and predictable quality of service from one end of the network to the other, the collection of components that must communicate and interact results in a fairly complex process. Microsoft® Windows 2000 QoS has the ability to facilitate priority along every step of a packet's journey: in the sender's network stack, at the switch, and even at each QoS-enabled router hop. Quality of Service also has the ability to facilitate how much data can and should be sent in a given unit of time, maximum burst rates, and overall bandwidth utilisation rights. These can be configured based on administratively configurable policies. These functional capabilities only scratch the surface of quality of service.

Windows 2000 QoS is comprised of a number of components. The following figure shows where many of the QoS components reside in relation to the network stack, where communication occurs between and among them, and where certain interfaces, such as APIs, facilitate developing QoS services.

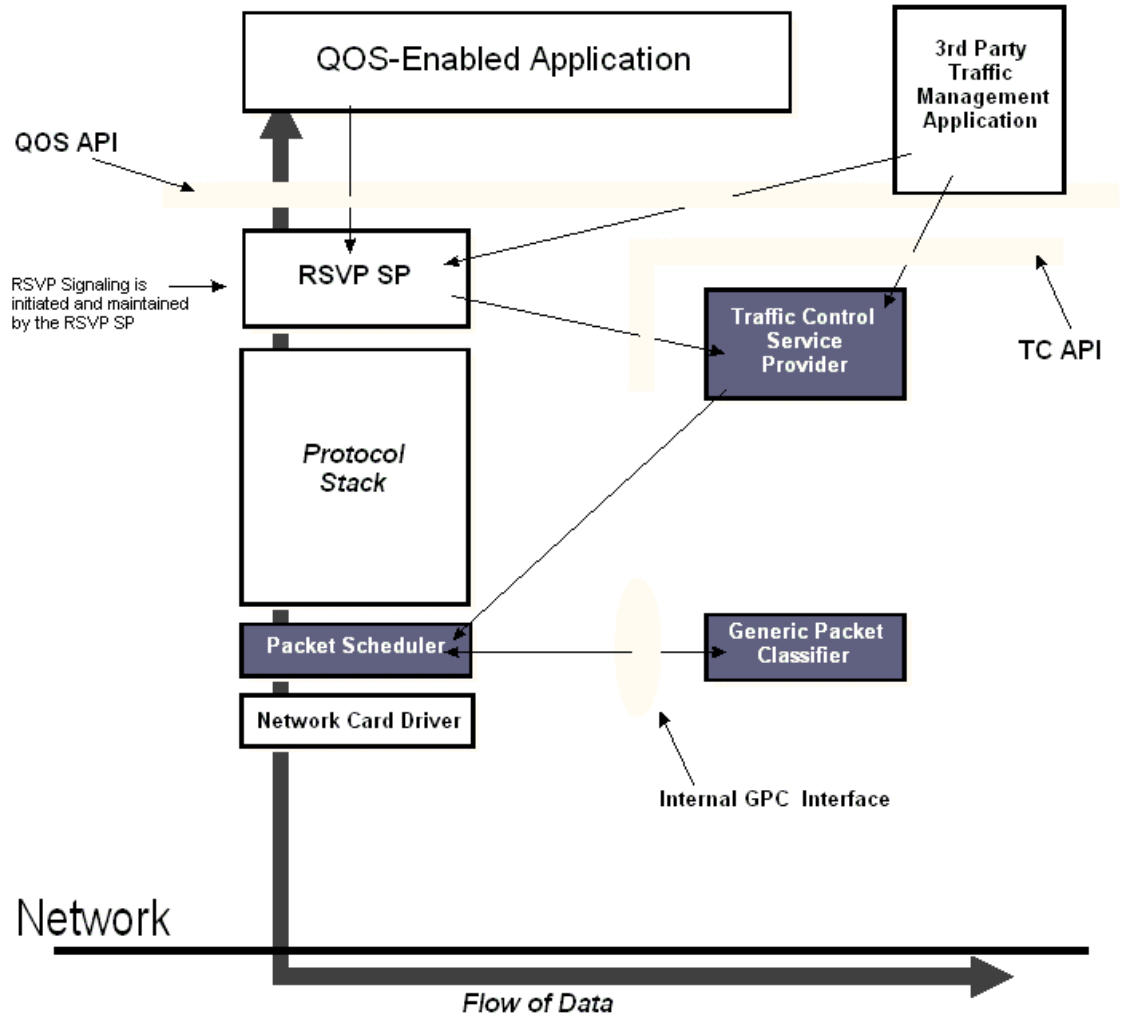


Figure 3-31: Windows 2000 QoS

To facilitate explanation, the Microsoft® Windows 2000 QoS documentation divides the QoS components into the following three topics:

- Application-Driven QoS Components
- Network-Driven QoS Components
- Policy-Driven QoS Components

3.5.2.3.3 Application-Driven QoS Components

RSVP Service Provider

The RSVP service provider (RSVP SP) is the name for the QoS component that invokes nearly all resulting QoS facilities. The RSVP SP communicates Windows Sockets 2 QoS se-

mantics to the RSVP SP. The RSVP SP DLL is loaded by Windows Sockets when a QoS-enabled socket is opened.

Traffic Control Modules

Traffic control (TC) plays a central role in the provision of quality of service. With traffic control, packets are prioritised both inside and outside the node on which TC is used. The implications for such granular control (or preferential treatment) of packets as they flow through the system and through the network, reach across the entire network realm or enterprise. Traffic control is realised through two modules, the Generic Packet Classifier and the Packet Scheduler.

Generic Packet Classifier

Packet classification provides a means by which packets internal to a specific network node can be classified, and consequently prioritised, within and by both user and kernel-mode network components. These classification and prioritisation uses include activities such as CPU processing attention or transmission onto the network. The Generic Packet Classifier (GPC) is utilised through the Generic Packet Classifier Interface, or GPC Interface, which facilitates an information store that can be used or associated with specified (defined) subsets of packets.

The importance of GPC hinges on its ability to provide lookup tables and classification services within the network stack, and is thus the first step in an overall and ubiquitous prioritisation scheme for network traffic.

Packet Scheduler

Packet scheduling is the means by which data (packet) transmission-governing—a key function of quality of service—is achieved. The packet scheduler is the traffic control module that regulates how much data an application (or flow) is allowed, essentially enforcing QoS parameters that are set for a particular flow. The packet scheduler incorporates three mechanisms in its scheduling of packets:

- A conformer
- The packet shaper
- A sequencer

The conformer and sequencer are discussed in more detail in the traffic control documentation. Since the packet scheduler's role is essential to overall traffic control understanding, it is defined here.

The packet scheduler considers the classification provided by the Generic Packet Classifier (GPC), and provides preferential treatment to higher-priority traffic. Consequently, the packet scheduler is the first step (in a sequential view) to ensuring that the prioritised network transmission of packets begins with data that has been deemed most important.

Part of the packet scheduler's responsibility is shaping the way packets are transmitted from a network device, a capability often referred to as packet shaping. Though often referenced by its own name, the packet shaper is simply a part of overall packet scheduler functionality.

The packet shaper mitigates the burst nature of computer network transmissions by smoothing transmission peaks over a given period of time, thereby smoothing out network usage to affect a more steady use of the network. The significance of the packet shaper becomes apparent: one factor that contributes to network congestion is the burst nature of computer data transmissions, a side-effect of the inherent "send it all out right now" nature of IP transmission. Packet shaping can help alleviate at least some of the effects of such activity by spacing out QOS-enabled packet transmissions.

Resource Reservation Protocol (RSVP) Service

The RSVP service is a single instance Windows 2000 service that runs on a Windows 2000 computer. The RSVP service instigates traffic control functionality (if appropriate), and implements, maintains, and handles RSVP signalling for all Windows 2000 QoS functionality.

The RSVP service, by virtue of the fact that it implements and maintains RSVP and is the initiator of traffic control, is at the heart of Windows 2000 Quality of Service.

QoS API

The QoS API is the programmatic interface to the RSVP service provider (RSVP SP). Under most circumstances, the QoS API is the only interface that programmers will require to create QoS-aware or QoS-enabled applications. Most operations that happen on behalf of an application in the QoS sequence are a result of QoS API calls communicating requests down (and sometimes back up) through QoS components, creating a QoS-enabled flow of data that keeps important data moving through the network with preferential transmission consideration.

Traffic Control API (TC API)

The traffic control application programming interface (TC API), is a programmatic interface to the components that regulate network traffic on local hosts; both from an internal perspective (within the kernel itself), and from a network perspective (prioritisation and queuing of packets based on transmission priority).

Traffic control is implicitly invoked through calls made to the QoS API and subsequently serviced by the RSVP service provider (RSVP SP). However, applications that require further or specific control over traffic control can use the TC API. The RSVP SP also uses TC API calls. The use of functions in the TC API requires administrative privilege

3.5.2.3.4 Network-Driven QoS Components

802.1p

Responsibility for QoS provisions on the local segment, and avoidance of the "all packets are treated equally" issue, falls onto the hub or switch servicing the segment. At such a level, the issue of differentiating between network packets, and perhaps treating them differently, must fall into the realm of the media access control (MAC) header. The MAC header (the lower half of Layer 2 in the ISO OSI Model) is the only part of a packet that hubs or switches investigate in their scope of work.

802.1p provides prioritisation of packets traversing a subnet by the setting of a 3-bit value in the MAC header. Thus, when the local segment becomes congested and the hub/switch workload results in the delay (or dropping) of packets, those packets with flags that correspond to higher priorities will receive preferential treatment, and will be serviced before packets with lower priorities.

Note that implementing 802.1p for QoS requires an 802.1p-aware network interface card, an 802.1p-aware device driver, and an 802.1p-aware switch.

Differentiated Services

Differentiated services enables packets that pass through network devices operating on Layer 3 information, such as routers, to have their relative priority differentiated from one another. Differentiated services uses 6 bits in the IP header to specify its value, called the DSCP (DiffServ code point); the first 6 bits of the ToS field, the first three of which were formerly used for IP precedence. Differentiated services has subsumed IP precedence, but maintains backward compatibility.

With differentiated services marking, Layer 3 devices can establish aggregated precedence-based queues and provide better service (when packet service is subject to queuing, as is the case under significant traffic loads) to packets that have higher relative priority. For differentiated services to be effective, Layer 3 devices must be DSCP-enabled.

L2 Signalling

WAN technology manipulates Layer 1, Layer 2, and to a certain extent Layer 3 information, as it transmits data over the telecommunications network. Since quality of service is an end-to-end solution that provides quality of service for data transferred across the network, there must be a means by which data passing through WAN interfaces can be associated with some sort of preferential or non-preferential treatment. Such a requirement necessitates the mapping of RSVP or other QoS parameters to WAN technology QoS interfaces.

Layer 2, however, is where QoS technology interacts most with the WAN's underlying signalling, since it is in Layer 2 where existing WAN technologies implement their own native QoS components. L2 signalling, in Windows 2000 QoS terms, takes QoS information such as parameters that are carried in RSVP messages to or through each network node between end de-

vices, and maps that QoS information to native WAN technology QoS interfaces. For example, the classical IP over ATM (CLIP) module in Windows 2000 specifically maps Windows 2000 QoS settings to an appropriate ATM class of service.

Subnet Bandwidth Manager (SBM)

The Subnet Bandwidth Manager (SBM) is the QoS component that provides resource management and policy based–admission control for QoS-aware applications using shared media subnets (Ethernet, for example). The SBM is based on an IETF draft that defines SBM functionality and its general implementation. SBM is implemented in Windows 2000® through the Admission Control Service (ACS). ACS is a Windows 2000 service that resides on a Windows 2000 Server.

Resource Reservation Protocol (RSVP)

Carries and disseminates QoS information to QoS-aware network devices along the path between a sender and one or more receivers for a given flow, and also to senders and receivers.

3.5.2.3.5 Policy-Driven QoS Components

Admission Control Service (ACS)

Admission control service (ACS), is a Windows 2000 QoS component that regulates subnet usage for QOS-enabled applications. The ACS exerts its authority over QOS-aware applications or clients by placing itself within the RSVP message path. With this placement, ACS effectively intercepts RSVP PATH, RESV, PATH_ERR, RESV_ERR, PATH_TEAR, and RESV_TEAR messages and passes the messages' policy information to Local Policy Modules (LPMs) for authentication. This exertion of ACS authority occurs on each interface (or shared medium) over which a given QoS flow must traverse. For a simplified example, if ACS is functioning on a source subnet and a (different) destination subnet for a given flow, policy restrictions are enforced by the ACS on each subnet.

ACS regulation is based on available network resources and on administratively-configurable information on users, or group policy. ACS is implemented as a Windows 2000 service on a Windows 2000 Server.

Local policy modules (LPMs) fall within the fold of ACS functionality, and can be considered an integral part of the ACS. With the default LPM, Microsoft Identity LPM (MSIDLPM) user information in the intercepted RSVP message is used to look up user policy in Windows 2000 Active Directory services. MSIDLPM then makes policy decisions based on information found in Active Directory services.

Another ACS component, the Policy Control Module (PCM), actually mediates the interaction between the ACS and LPMs. If there are multiple residential LPMs, the PCM will send all policy data objects contained in the received RSVP messages to each LPM, gather all re-

sponses, perform logical checks on the information, aggregate it, and return the combined response to the ACS.

If network resources are available and if the policy check succeeds, the RSVP message and its policy information is sent to the next hop (or the previous hop, if it is a PATH or RESV message). In this way, ACS acts as the logical gatekeeper for RSVP message propagation across the network by rejecting requests under the following conditions:

- If local segment resources aren't available to provide the requested level of QoS (the SBM functionality of the ACS)
- If the sender or receiver doesn't have appropriate policy permission to transmit data with the requested parameters

When such conditions occur, no network nodes beyond the ACS (in the appropriate direction) receive any of the RSVP messages rejected by the ACS. However, the error messages due to the rejection will traverse the network to get to the network mode that made the request.

This provides twofold service. It keeps unnecessary RSVP signalling traffic from traversing the network by keeping lame-duck RSVP messages from running across the network, and it preserves processing resources for routers and WAN Interface Cards (WANICs) since they will not have to handle such RSVP messages. Note that any node that declines requests based on policy failure, however, will return an RSVP error message to the sender, indicating failure. Clients will not transmit anything if their request is rejected by ACS.

Though ACS is a Windows 2000 QoS component, its services include other QoS components, such as the Subnet Bandwidth Manager (SBM) and its LPM interface.

Local Policy Module (LPM)

The Local Policy Module (LPM) is a QoS component responsible for retrieving and returning policy-based decisions used by the Admission Control Service (ACS). A default LPM provides an interface to policy information that is configured and stored in Windows 2000 Active Directory services. LPM is a generic term used to supply policy-based admission control decisions for ACS. An LPM makes these decisions using policies that are generally configured by network administrators, and stored in policy databases. An LPM is usually implemented as a DLL.

LPMs are used on the ACS server. Client QoS components that generate the policy element reside on the client, and are capable of creating policy information that is carried in RSVP messages to the ACS (which then gets forwarded down to the LPM). It is possible to install an LPM on the ACS server for which there is no corresponding component on the client; for example, an ACS-based LPM could enforce "time-of-day" policies, or could be capable of communicating with a COPS server.

It is important to note the difference between IETF-draft technologies and the Microsoft implementation of them. Subnet Bandwidth Manager (SBM), for example, is a technology de-

rived from an IETF-draft proposal for regulating access to 802 subnets (draft-ietf-issll-is802-sbm-08.txt). ACS is a Windows 2000 service (and a QoS component) that incorporates SBM technology within its fold to incorporate regulation of access to 802 subnets in accordance with policy control.

Policy Information

Microsoft provides identity policy information through a DLL installed on QoS-enabled Windows clients such as Windows 2000 Server and Windows 2000 Professional.

The policy information is incorporated in RSVP resource reservation requests, which in turn get sent out onto the network in the form of RSVP messages. This policy information is intercepted by the Admission Control Service (ACS), which includes SBM functionality as part of its fundamental service suite. The ACS services such requests by checking whether the requesting client has authorisation to use network resources on the local subnet. If successful, the policy information is forwarded to the next network node for subsequent policy checking, and such activity continues toward the intended receiver, along the data path of network nodes, until the request is rejected or the intended receiver (the node with which the sending client wishes to communicate) is reached.

RSVP

Carries policy data between end nodes and the ACS/SBM. RSVP also carries rejections to admission requests back to the requesting node.

Local Policy Module API (LPM API)

The local policy module application programming interface (LPM API) is the programmatic interface by which LPMs communicate and interact with the Admission Control Service (ACS). The LPM API also specifies how LPMs are registered and initialised within the constructs of the ACS.

Such interaction is actually regulated by an abstraction module called the Policy Control Module (PCM). Because it is possible to have multiple LPMs, the PCM manages the policy-based decision information that LPM modules return. Note that LPMs may selectively accept or reject flows. For example, an LPM can receive an RSVP-based request from the PCM that has multiple flow requests; the LPM can then selectively accept or reject individual flows within that request, and return the results to the PCM. Note, too, that the PCM can manage information returned from multiple LPMs (if multiple LPMs are installed on the system), perform logical aggregation of their results, and then return aggregated information to the ACS.

The LPM API consists of a handful of functions used to allow its interaction with the ACS. The interaction of an LPM with its corresponding policy store or server is excluded from this interface; such interfacing would be proprietary to the LPM and the policy store or server.

3.5.3 Existing Applications

3.5.3.1 Legacy Applications

3.5.3.1.1 MBone Tools

A wide range of MBONE tools exists today. This section describes the selected tools from the list provided above

3.5.3.1.1.1 Stand-alone Tools

With the absence of multicast, the appropriate tools to demonstrate multimedia conferencing should be based on the point-to-point versions of *vic* and *vat.*

3.5.3.1.1.1.1 Vic

Vic is a video conferencing tool that allows users to transmit video over an IP multicast network [VIC]. A conference session is specified by the destination address and the UDP port used. Point-to-point conferences are initialised by supplying host IP addresses, whereas multiparty conferences need a Class D group address. To transmit video data, the Real Time Transport Protocol (RTP and RTPv2) is used. A variety of video formats are supported, depending on the host video hardware and the available bandwidth. The format can be chosen by the user. Some of the supported features are: compression/decompression for video data, voice switched viewing windows, dithering algorithms, interactive title generation and encryption. Audio conferencing is not included in this tool.

3.5.3.1.1.1.2 Vat

Whereas Vic supplies the video part of a multimedia conference, Vat allows users to create point-to-point or multiparty audio conferences [VAT]. Besides the widely used sound equipment, no special encoding/decoding hardware is required. A conference is specified as described above: unicast IP addresses are needed for point-to-point sessions and, multicast address is required for a multiparty session. This tool supports a number of audio coding standards: PCM, DVI, GSM and LPC. Audio data is transported over RTP or RTPv2.

In general, both Vic and Vat

- are widely used by the Internet community.
- fulfil the needs of business as well as residential users.
- together can generate a wide range of traffic characteristics due to the support of many different audio and video codings. In addition, the users can adjust some of the parameters.
- provide a high compatibility with other MBONE tools used nowadays.

3.5.3.1.2 Existing FTP applications

3.5.3.1.2.1 Existing FTP architecture

The file transfer application based on FTP (File Transfer Protocol, the Internet standard) copies a complete file from one system to another.

The user protocol interpreter initiates the control connection. At the initiation of the user, standard FTP commands are generated by the user protocol interpreter and transmitted to the server process via the control connection. (The user may establish a direct control connection to the server-FTP, from a terminal, and generate standard FTP commands independently, bypassing the user-FTP process.) Standard replies are sent from the server protocol interpreter to the user protocol interpreter over the control connection in response to the commands.

The control connection is established in the normal client-server way. The server does a passive open on the well-known port for FTP (21) and waits for a client connection. The client does an active open to TCP port 21 to establish the control connection. The control connection stays up for the entire time period the client communicates with this server. This connection is used for commands from the client to the server and for the server's replies.

The FTP commands specify the parameters for the data connection (data port, transfer mode, representation type and structure) and the nature of file system operation (store, retrieve, append, delete, etc.). The user or its designate should "listen" on the specified data port, and the server initiates the data connection and data transfer in accordance with the specified parameters. It should be noted that the data port need not be in the same host that initiates the FTP commands via the control connection, but the user or the user-FTP process must ensure a "listen" on the specified data port. The data connection may be used for simultaneous sending and receiving.

The protocol requires the control connections to be open, while data transfer is in progress. It is the responsibility of the user to request the closing of the control connections when finished using the FTP service, while it is the server who performs this action. The server may abort data transfer if the control connections are closed.

The interactive user normally doesn't deal with the commands and replies that are exchanged across the control connection. In fact the interactions are left to the protocol interpreter. The user interface is a type of interface selected by the interactive user (full-screen menu selection, line-at-a-time commands, etc...) and converts these into FTP commands that are sent across the control connection. Similarly, the replies returned by the server across the control connection can be converted to the format required by the interactive user. The two protocol interpreters invoke the two data transfer functions when there is a request for data transfer.

3.5.3.1.2.2 Connection management

Data connection can be used for:

1. Sending a file from the client to the server
2. Sending a file from the server to the client
3. Sending a listing of files or directories from the server to the client

The FTP server sends file listings back across the data connection, rather than multi line replies across the control connection. A new data connection is required for every file transfer or directory listing. The normal procedure is as follows:

1. The creation of the data connection is under control of the client, because it's the client that issues the command that requires the data connection (get a file, put a file, or list a directory).
2. The client normally chooses an ephemeral port number on the client host for its end of the data connection. The client issues a passive open from this port.
3. The client sends this port number to the server across the control connection using the PORT command.

The server receives the port number on the control connection, and issues an active open to that port on the client host. The server's end of the data connection always uses port 20.

3.5.3.1.3 Existing Web applications

The World Wide Web is the vision of programs that can understand the numerous different information-retrieval protocols (FTP, Telnet, NNTP, WAIS, gopher, ...) in use on the Internet today as well as the data formats of those protocols (ASCII, GIF, Postscript, ...) and provide a single consistent user-interface to all.

World Wide Web is a subset of the Internet, the information being stored in computers all over the world, as a distributed hypermedia. Web documents are written in Hypertext Mark-up Language (HTML). The network protocol for requesting and transmitting Web documents is the Hyper Text Transfer Protocol (HTTP), which both Web servers and browsers can understand. The Web allows users to use hypertext (point and click) to view multimedia web pages, incorporating texts, pictures, sounds, video sequences, Java applets, etc. A group of related web pages compose a site. Web sites are housed on a Web server.

The World Wide Web main features are listed below.

- Software has been developed to allow numerous different computers to act as Web servers and browsers. A Web server is a computer which stores page information and other files and transmits that information across a computer network. A browser is a software pack-

age which requests information from a server and than processes it to display it. Browsers allow a user to view and interact with web pages.

- Graphic images, sound files and movies can easily be incorporated within a document and displayed on a range of different computers in similar formats.

Hyperlinks can be incorporated from anywhere within a document to another document or file (such as a graphic, sound or movie file) and any other Web server on the Internet. Any web document has it's own address, the location of the document is specified by the Uniform Resource Locator (URL). URL is a series of words, slashes, and "dots" which make up the Internet address. It is possible to represent nearly any file on the Internet with an URL. The first part specifies the method of access, the second the address of the computer on which the data or service is located. Further parts may specify the names of the files.

HTTP functions on a client-server basis, and it is implemented in software in the end systems. The connection between Web servers and Web browsers is based on the simple application layer protocol HTTP. This protocol connects a HTTP client (that is, the web browser) and a HTTP server (that is, the web server). Recall from the protocol stack applet that the application layer is at the top of the TCP/IP protocol stack.

3.5.3.2 QoS Aware Applications

3.5.3.2.1 Microsoft's QoS enabled applications

3.5.3.2.1.1 Microsoft NetMeeting

NetMeeting is a Windows feature that provides real-time collaboration through standards-based data/audio/video Internet conferencing client support. Data conferencing includes application sharing, chat, whiteboard, and file transfer features. Other features include remote desktop sharing and gatekeeper support.

NetMeeting 3 provides components for a wide range of developers. The NetMeeting 3 SDK provides a scripting guide for Web developers and other scripting application developers, a low-level T.120 application guide, and a COM guide. Because most of the components are implemented using the Component Object Model (COM), they can be called from any COM-supporting languages, such as C/C++, Microsoft Visual Basic ®, and Java.

To take full advantage of NetMeeting, one should be familiar with the following technologies: H.323 and T.120 conferencing standards, Microsoft ActiveX® technologies, HTML, the Windows Sockets API, and COM.

3.5.3.2.1.1.1 NetMeeting Components

3.5.3.2.1.1.1.1 Conferencing API

The NetMeeting conferencing API, takes advantage of all the functionality provided at lower layers, including, optionally, the NetMeeting user interface (UI).

3.5.3.2.1.1.1.2 Control Layer

The following components tie together the NetMeeting functionality provided by lower layers.

Node Management	Keeps track of conferences and conference participants.
NetMeeting UI	Displays NetMeeting functionality to users.

3.5.3.2.1.1.1.3 Audio/Video Components

The following components manage the audio and video functionality in NetMeeting.

Call Control	Sets up the audio and video portion of the NetMeeting call using the H.245 standard for call control.
RTP/RTCP	Handles real-time streaming of audio and video over the Internet using the H.255.0 standard.
H.263 Video	Compresses and decompresses video data using a H.263-compliant video codec.
G.723 Audio	Compresses and decompresses audio data using a G.723-compliant audio codec.
Media Stream Engine	Co-ordinates capture, compression, and transmission of audio and video data. On the receiving side, this component receives, decompresses, and replays audio and video.

3.5.3.2.1.1.1.4 Data Conferencing Components

The following data conferencing components handle transmission of data among different conference participants. This includes file transfer, application sharing, whiteboard, and chat data, as well as applications written using the NetMeeting SDK's Data Channel.

T.120 data	Sends and receives data between different conference participants using the T.120 protocol for data conferencing.
Built-in data applications	Use the T.120 data component to send and receive information. (These applications include chat, file transfer, and whiteboard.)

3.5.3.2.1.1.1.5 Internet Locator Service Component

The Internet Locator Service (ILS) server component communicates with ILS on the network (LAN or Internet) to get directory listings and find users.

3.5.3.2.1.2 Window Media Services

Windows Media Services is a core feature of the Windows NT and Windows 2000 Server platforms. Windows Media Services is a platform for delivering high-quality digital media across the Internet and corporate intranets. Whether you're delivering the latest music videos on demand to thousands of consumers, or streaming a CEO broadcast to thousands of employees, Windows Media Services provides the most scalable, reliable, and manageable server to meet your digital media demands.

- **Windows Media Unicast service**

This service provides the ability to stream content to a specific client and enables on-demand streaming. On-demand streaming is the playback of a stream that is controlled by the client.

- **Windows Media Station service**

This service sends a stream to multicast-enabled routers. This enables the Windows Media server to provide a single stream to multiple end users.

- **Windows Media Program service**

You can use this service to control how many times a group of streams is played. This service is used when you are serving a Windows Media station.

- **Windows Media Monitor service**

You can use this service to monitor clients connected to publishing points. You can also use this service to monitor other servers that are connected to your Windows Media server.

- **Windows Media Rights Manager**

Windows Media Rights Manager, a digital rights management (DRM) application, lets content authors deliver songs, videos, and other media over the Internet in a packaged, encrypted file format. Windows Media Rights Manager consists of the Windows Media Packager and the Windows Media License Manager. Windows Media Rights Packager packages and encrypts media files by locking them with a "key." End users need a separate license containing the key to play a packaged media file with the Windows Media Player. Media files and licenses are stored separately, making it easier to manage the entire system.

3.5.3.2.1.3 TAPI–3.0

Though not an application itself, TAPI-3.0 provides QoS services to numerous third party telephony applications.

TAPI allows applications to support telephone communication. TAPI facilitates include:

- Connecting directly to a telephone network.
- Automatic phone dialling.
- Transmission of data (files, faxes, electronic mail).
- Access to data (news, information services).
- Conference calling.
- Voice mail.
- Caller identification.
- Control of a remote computer.
- Collaborative computing over telephone lines.

3.5.3.2.1.4 Kerberos authentication protocol

Kerberos is a network authentication protocol. It is designed to provide strong authentication for client/server applications by using secret-key cryptography. A free implementation of this protocol is available from the Massachusetts Institute of Technology. Kerberos is available in many commercial products as well.

The Internet is an insecure place. Many of the protocols used in the Internet do not provide any security. Tools to "sniff" passwords off of the network are in common use by systems crackers. Thus, applications which send an unencrypted password over the network are extremely vulnerable. Worse yet, other client/server applications rely on the client program to be "honest" about the identity of the user who is using it. Other applications rely on the client to restrict its activities to those which it is allowed to do, with no other enforcement by the server.

Some sites attempt to use firewalls to solve their network security problems. Unfortunately, firewalls assume that "the bad guys" are on the outside, which is often a very bad assumption. Most of the really damaging incidents of computer crime are carried out by insiders. Firewalls also have a significant disadvantage in that they restrict how your users can use the Internet. (After all, firewalls are simply a less extreme example of the dictum that there is nothing more secure then a computer which is not connected to the network --- and powered off!) In many places, these restrictions are simply unrealistic and unacceptable.

Kerberos was created by MIT as a solution to these network security problems. The Kerberos protocol uses strong cryptography so that a client can prove its identity to a server (and vice versa) across an insecure network connection. After a client and server has used Kerberos to prove their identity, they can also encrypt all of their communications to assure privacy and data integrity as they go about their business.

Kerberos is freely available from MIT, under a copyright permission notice very similar to the one used for the BSD operating and X11 Windowing system. MIT provides Kerberos in source form, so that anyone who wishes to use it may look over the code for themselves and assure themselves that the code is trustworthy. In addition, for those who prefer to rely on a professional supported product, Kerberos is available as a product from many different vendors.

In summary, Kerberos is a solution to your network security problems. It provides the tools of authentication and strong cryptography over the network to help you secure your information systems across your entire enterprise. We hope you find Kerberos as useful as it has been to us. At MIT, Kerberos has been invaluable to our Information/Technology architecture.

The Kerberos system employed by the Windows 2000 RSVP stack enables a PDP (which in this case is the Windows 2000 ACS) to authenticate an end user (called the Kerberos client) who is requesting network resources through a trusted KDC called the Kerberos server. A DC can also serve as a KDC, but this paper shows that the process works with any Kerberos 5 KDC. To simplify administration, it is recommended that both the end user and the PDP device belong to the same Windows 2000 domain or Kerberos realm. All ACS systems in a domain share the same ACS account and all password changes for the ACS account must be synchronised. The ACS domain is the set of ACS servers that can access a directory for a given DN.

The authentication process requires that both the client and the server (in this case, the PDP) be able to authenticate to the KDC and acquire credentials for the PDP service account. For Windows 2000 QoS, this service account is a user account named `AcsService`, or the ACS Kerberos principal name. To authenticate itself to the PDP, a client requests a Kerberos ticket with a target name of `AcsService`. This ticket contains a session key. The client then transmits the ticket, which contains an encrypted version of the client's identity and the session key, in the PATH message. The PATH message is intercepted by the PEP and forwarded to the PDP. Having acquired its own credentials for the AcsService account, and using the session key from the PATH message, the PDP is now able to extract the client principal name from the ticket, reveal the user identity, and use it to authenticate the user.

Figure 1, below, shows elements of the authentication process using the Microsoft ACS server as a combined policy decision and policy enforcement point. It is intended to illustrate the procedure used by any RSVP-aware network element to extract the Windows 2000 user ID for authentication. Non-Microsoft implementations may include additional elements and/or steps, but the basic process is the same.

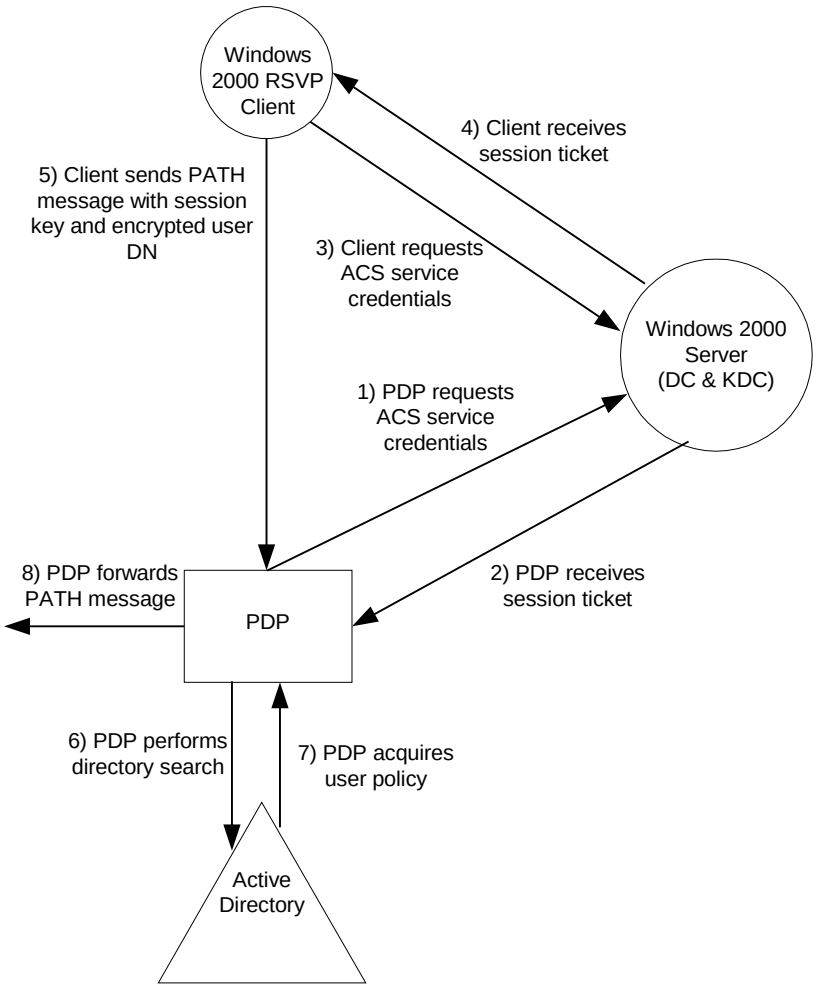


Figure 3-32: Kerberos authentication process

1. At start-up, the PDP (in this case, the Windows 2000 ACS) authenticates itself to the KDC and acquires the credential handle for service principal AcsService.

2. The KDC returns the credential handle for AcsService to the PDP.

3. Prior to sending a PATH message, the client requests a Kerberos ticket for a target service of AcsService.

4. The KDC returns a session ticket to the client for AcsService.

5. The client sends a PATH message that is intercepted by the PDP. The PATH message includes the Kerberos ticket for AcsService without Kerberos authdata (the authdata is not needed, requested, or used, in this implementation). Included in the PATH message is the user DN, which is encrypted using the session key.

6. Acting as the PDP, the Windows 2000 ACS, extracts the Kerberos ticket, decodes it, and uses the session key to decrypt the user's DN (using the GSSAPI call GSS_unwrap.). The DN is used for a LDAP search of the DS to retrieve the user's policy for RSVP.

7. The PDP returns the policy decision to the PEP. If the PDP has accepted this PATH message then the PEP configures the RSVP state for this PATH message and then forwards the PATH request onto the network. After the request is processed, it is forwarded to the next hop, toward the destination system. For the router to be able to process the identity in the PATH request, it has these alternatives:

    - Have a PDP running internally in the router. This PDP would have to use the same Kerberos principal name as other ACSs in the domain and participate in password synchronisation.

    - Send the PATH message to the PDP to retrieve the policy for the user. The PDP would be running with the ACS identity.

The preceding description is between the end system and the first policy-aware hop. A different encryption key can be used beyond this hop. This key can be pre-shared, a public key, or obtained using a trusted third party.

For more details on the Kerberos authentication scheme, read RFC 1510 “The Kerberos Network Authentication Service.”

ACS - Admission Control Service

AD - Active Directory, the Windows 2000 implementation of a directory service

ADT – Abstract Data Type

DC - Windows 2000 Domain Controller

DES - Data Encryption Standard

DN - Distinguished Name

DS - Directory Service

GSSAPI - Generic Security Service Application Program Interface

KDC - Key Distribution Centre

PATH - RSVP message that follows the data path

PDP - Policy Decision Point

PE – Policy Element

PEP - Policy Enforcement Point

QoS - Quality of Service

RESV - RSVP reservation request message

RSVP – Resource Reservation Protocol

SSPI – Security Support Interface

3.5.3.2.2 Toolkits for the implementation of QoS – aware applications

3.5.3.2.2.1 Java Media Framework

The Java Media Framework (JMF) is an application programming interface (API) for incorporating audio, video and other time-based media into Java applications and applets. It is an optional package which extends the multimedia capabilities in the J2SE platform.

The latest release, JMF 2.1, adds optimisations for Sun Ray and Linux platforms, and adds greater RTP/RTSP support to inter-operate with standards-based, third-party video streaming servers from Entera, Apple and Sun.

JMF's ability to capture, playback, transmit and transcode audio, video and other media gives multimedia developers a powerful toolkit to develop cross-platform applications and applet. Note, there is no change in the JMF 2.0 API to JMF 2.1 API, only the underlying implementation has been updated.

3.5.3.2.2.2 Continuos Media Toolkit

The Continuous Media Toolkit (CMT) is a toolkit for multimedia applications. It is built on top of Tcl/Tk, a scripting language and graphical user interface toolkit, and Tcl-DP, which provides network tools included a remote procedure call package and a name server. CMT is freely distributed and is very portable. It has been compiled and tested on the following platforms:

- DEC Alpha (Digital Unix 3.x, X11R6)
- HP 9000/700 (HPUX 9.0x, X11R6)
- Sun Sparc (SunOS 4.x and Solaris 2.5, X11R6)
- FreeBSD 2.2.2-Release
- Linux 1.2.8 or later
- SGI (Irix 5.3, X11R6)
- Windows NT

CMT supports several audio and video encoding formats, including Sparc style audio (8-bit µ-law compressed or 16-bit linear), MPEG video, MJPEG video, and H.261 video. It contains support for a number of audio interfaces including the Sparc, Linux, and Irix devices, as well as DEC's AudioFile. It also contains software MPEG, MJPEG, and H.261 decoders as well as the capability to perform hardware assisted decompression using the Sun Parallax, SunVideo, DEC J300, or SGI Cosmo board.

The toolkit is implemented as a collection of objects, each of which handles a specific task, for example, reading MPEG encoded video from a file or decoding and displaying MPEG encoded video. Objects can be easily created and connected to build applications. Aside from objects that read or decode audio and video, a number of other interesting objects are available, including:

- Objects to support the construction of distributed applications.
- Objects to transmit and receive data across a TCP/IP network using Cyclic-UDP, a best effort protocol.
- Objects to transmit and receive data using the Real-time Transport Protocol, the protocol used by the MBONE tools.
- Objects to filter uncompressed video.

CMT also comes with the CMplayer, a sample CMT application that can be used to play audio and video files locally or from a CMT video file server.

3.5.3.2.2.3 RAPI: an RSVP Application Programming Interface

An Internet application uses some "API" (Application Programming Interface) in order to request enhanced quality-of-service (QoS). A local RSVP control program will then use the RSVP protocol to propagate the QoS request through the routers along path(s) for the data flow. Each router may accept or deny the request, depending upon its available resources. In the case of failure, the local RSVP control program will return the decision to the requesting application via the API.

A particular RSVP API implementation known as "RAPI" is described. RAPI is based on a client library linked with the application.

The following diagram shows the RAPI implementation model. RSVP is implemented on a host by a user-level daemon program. The procedures of the RSVP client library module interact with the local RSVP daemon program through a Unix-domain socket. RAPI refers to the interface between the application and the RSVP client library.

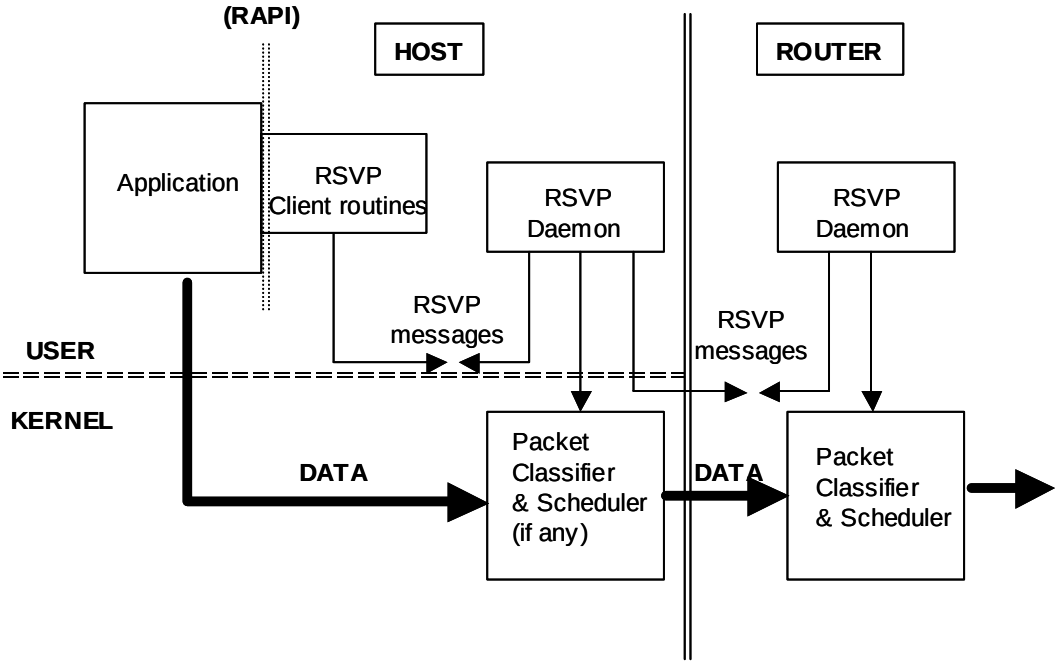


Figure 3-33: RAPI implementation model

3.6 Overall Class Diagram

The following diagrams result from the object-oriented **analysis** of the different requirement documents produced by the AQUILA teams. This corresponds in a first time to a static view of the system. We used the CASE tool Together 3.2 / Together 4.0 for the modelling and UML [UML1.3] as modelling language.

For a better comprehension and reading we split the system in many entities, or diagrams corresponding to the different items of this contribution. For a better readability we kept the title of the paragraphs.

This model is aimed to show many different views of the system.

The part entitled "general description" represents the main view of the system and the main interactions between the different partners of the system in a very high abstraction level.

The part entitled "Network architecture" represent many views of the whole network: the physical one, the logical one, as well as the access network and the core network.

The part entitled "services" models the services.

3.6.1 General description

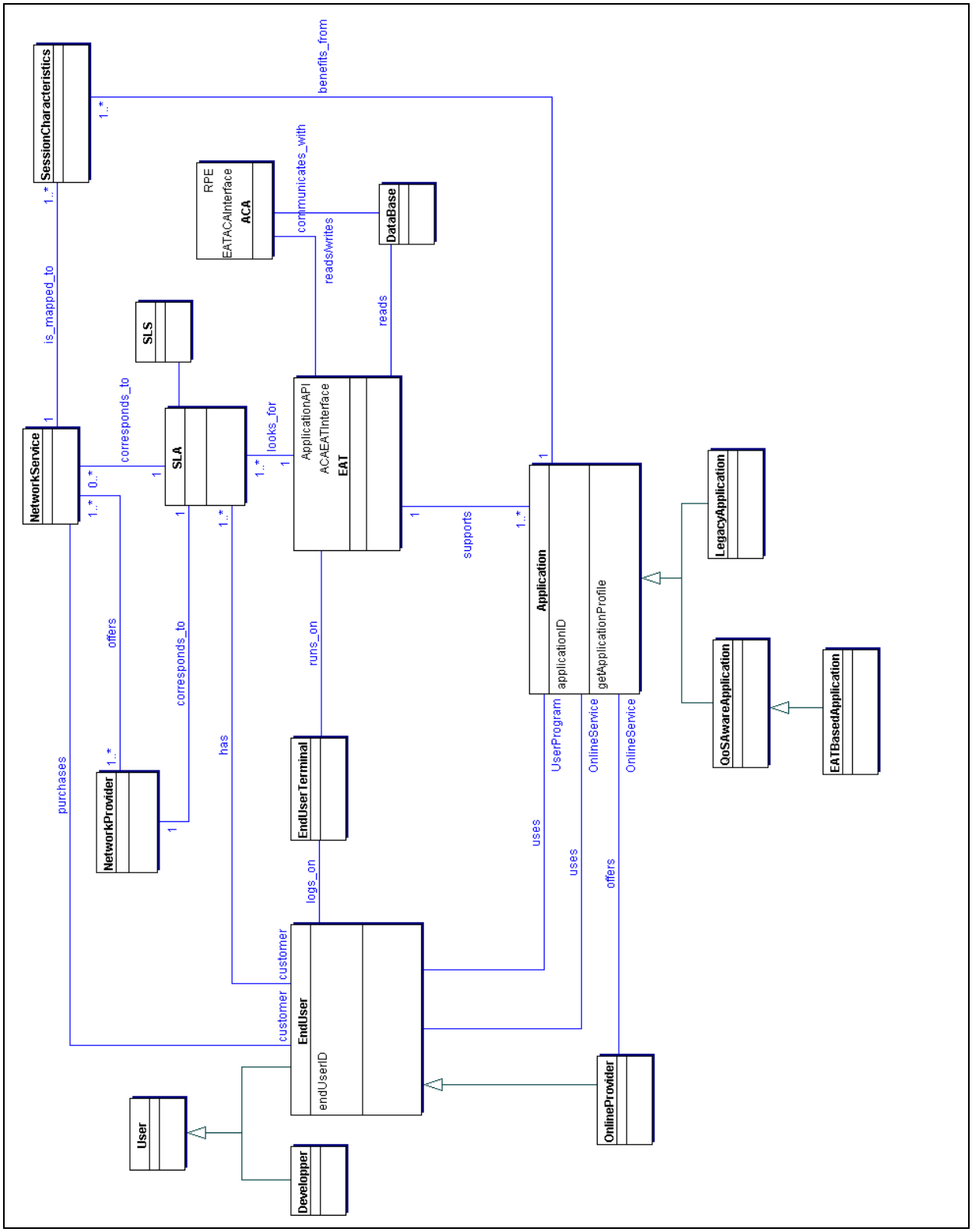


Figure 3-34: Class diagram for the general description

3.6.2 Network architecture

3.6.2.1 Network elements

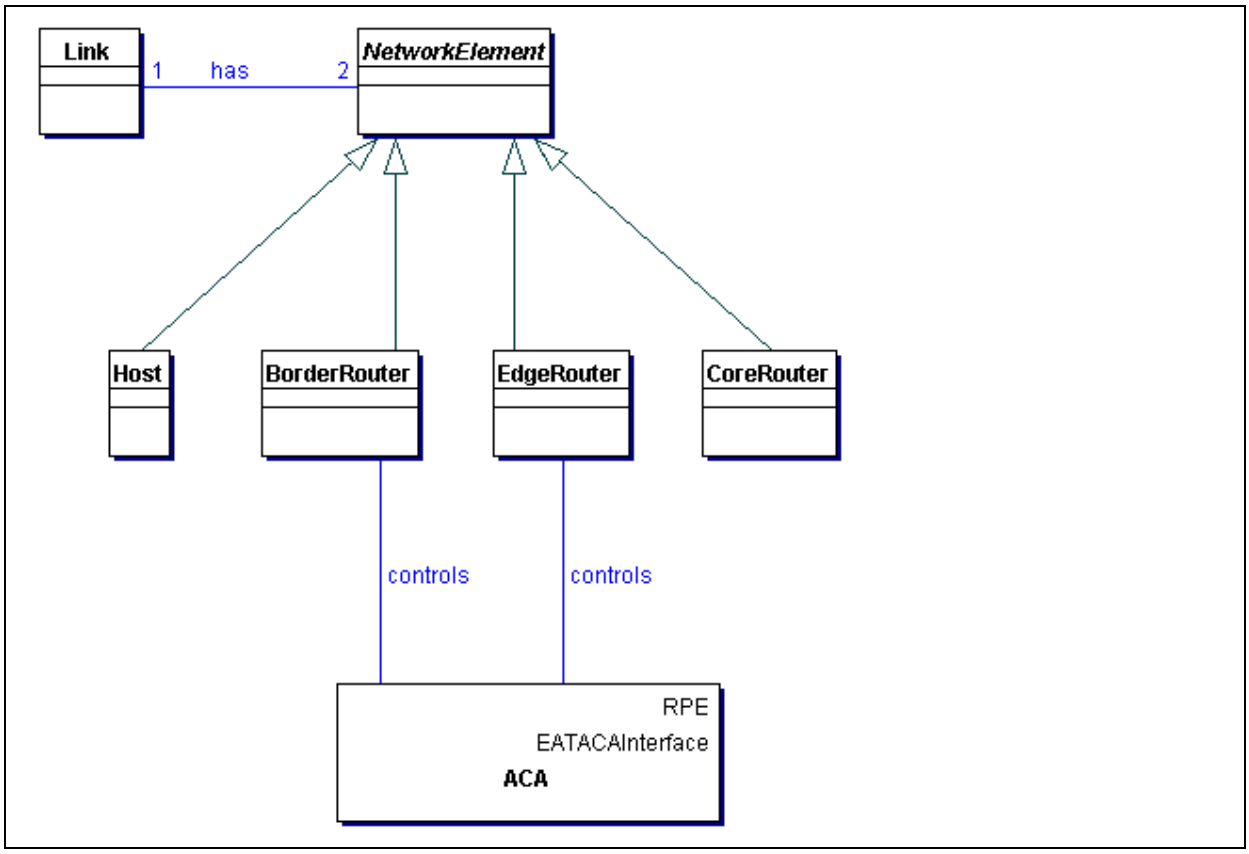


Figure 3-35: Class diagram for the network elements

3.6.2.2 Network topology

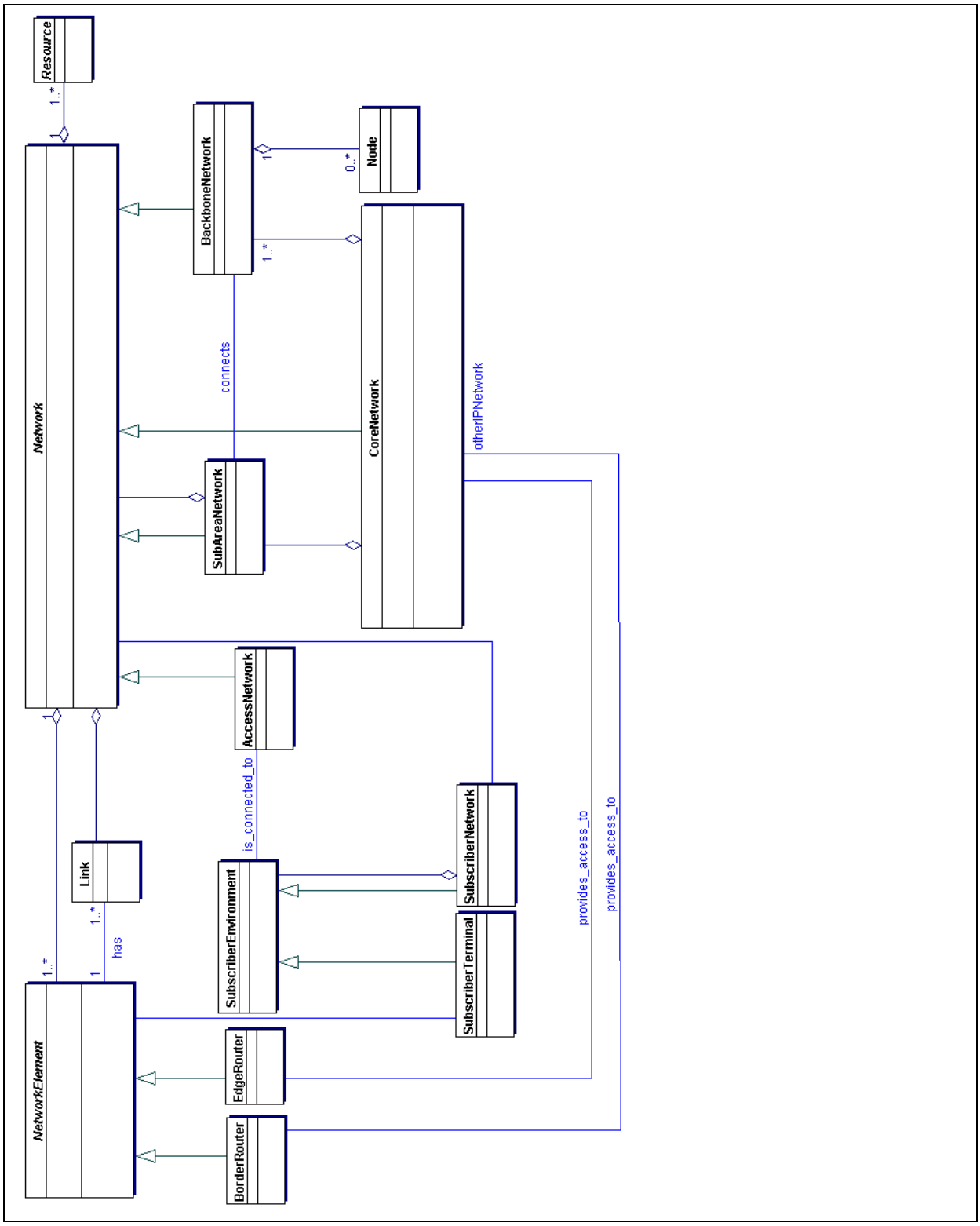


Figure 3-36: Class diagram for the network topology

3.6.2.3 Logical network

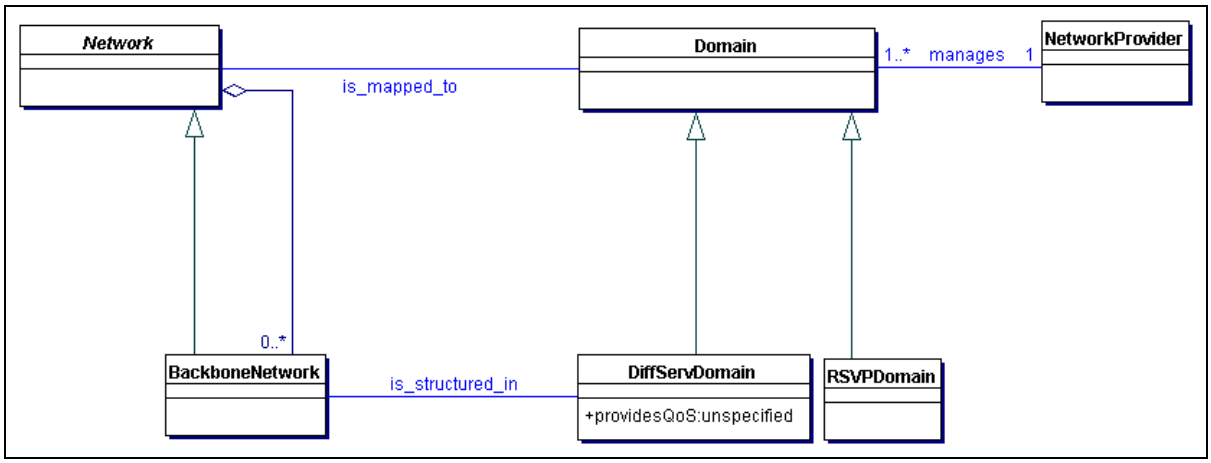


Figure 3-37: Class diagram for the logical network

3.6.2.4 Core network architecture

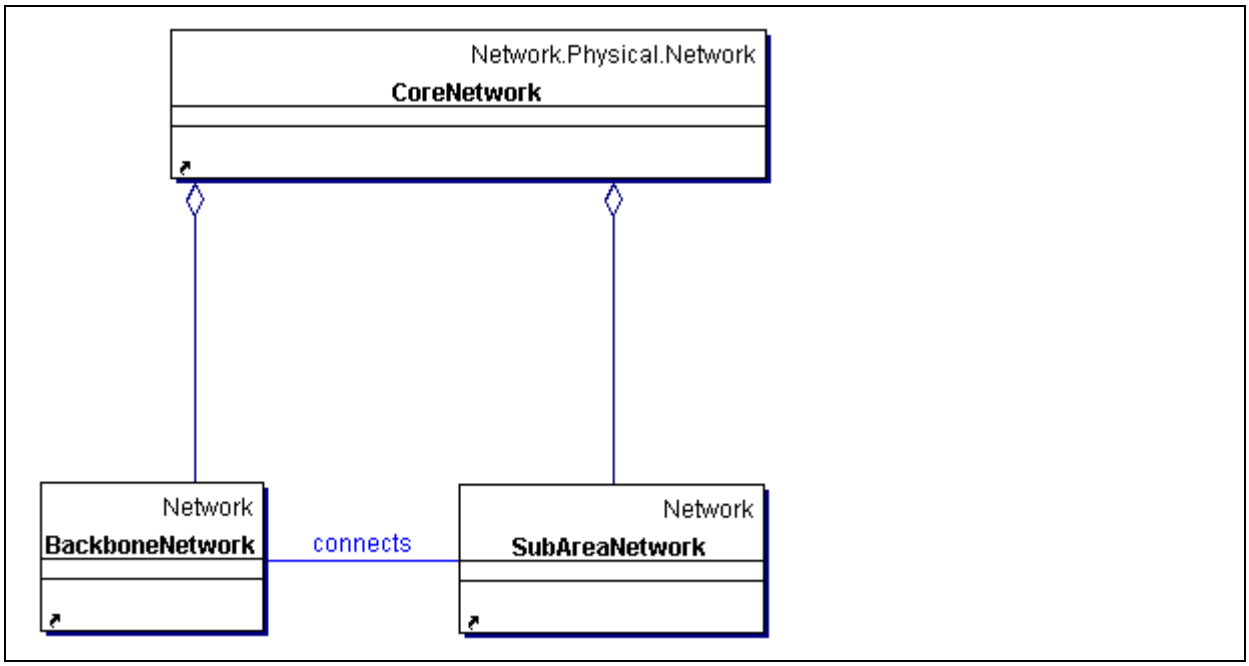


Figure 3-38: Class diagram for the core network

3.6.3 Services

3.6.3.1 Network services

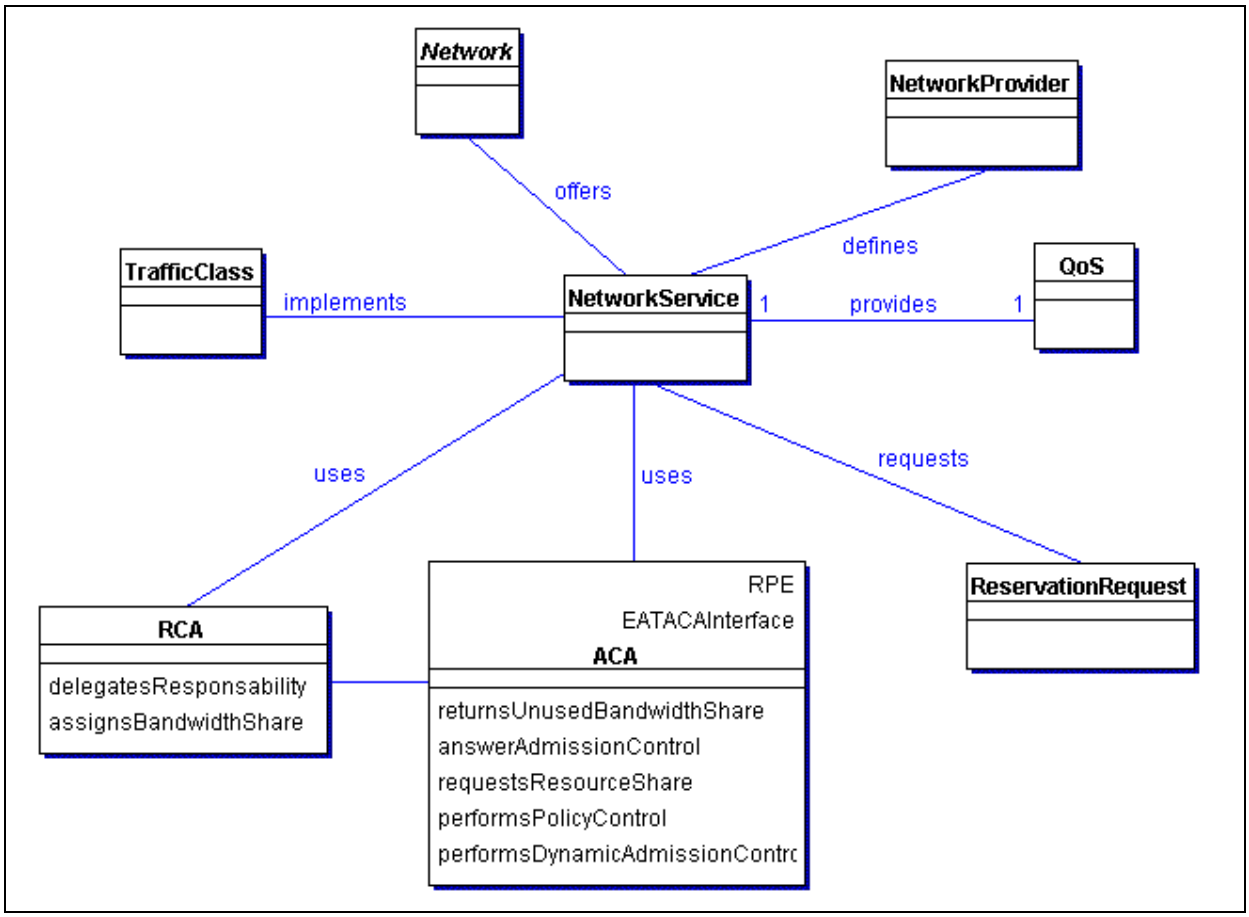


Figure 3-39: Class diagram depicting the Network Services

3.6.3.2 Traffic class

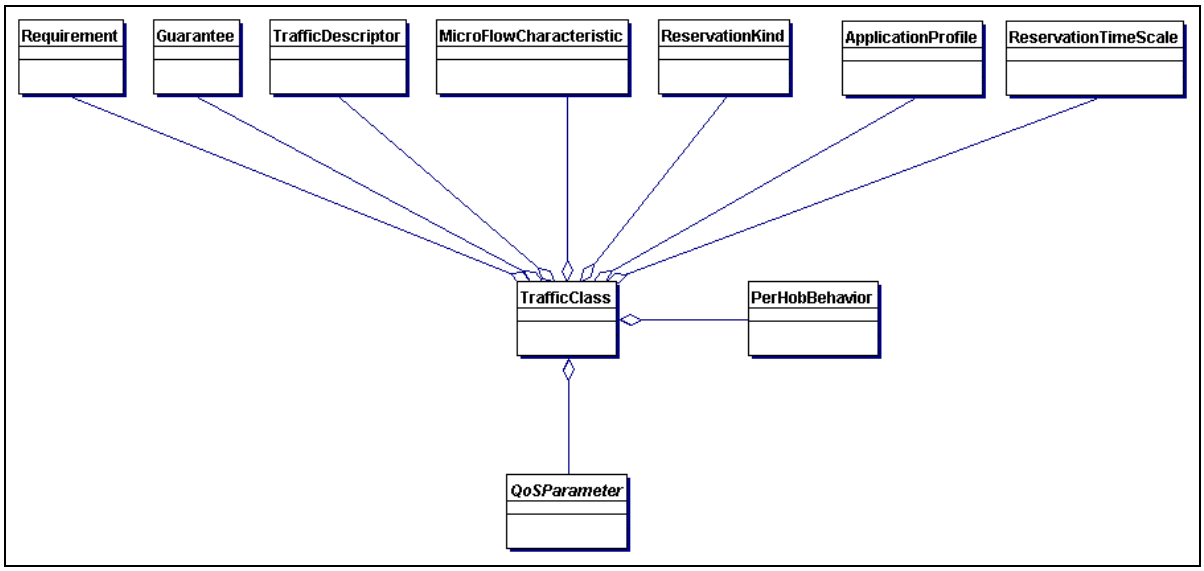


Figure 3-40: Class diagram for the traffic class

3.6.3.3 QoS

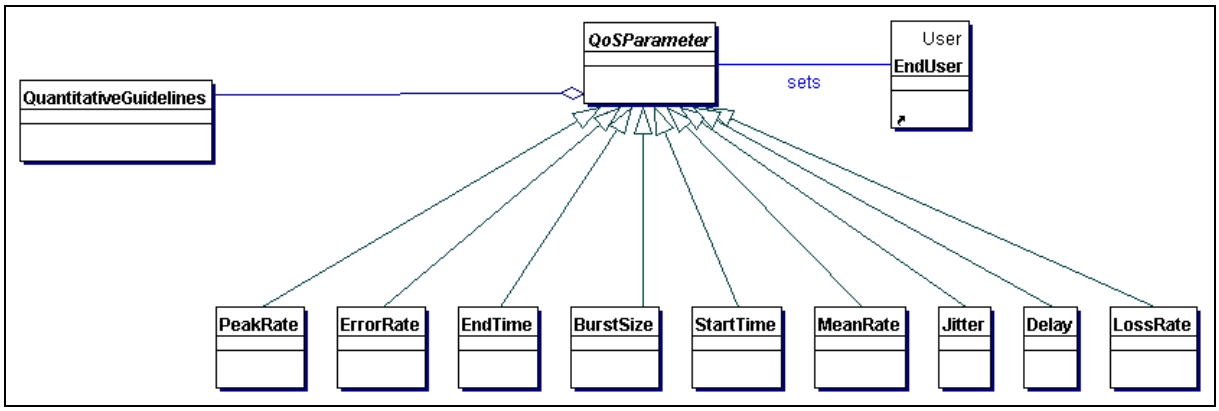


Figure 3-41: Class diagram for the QoS

3.6.4 Resource control layer

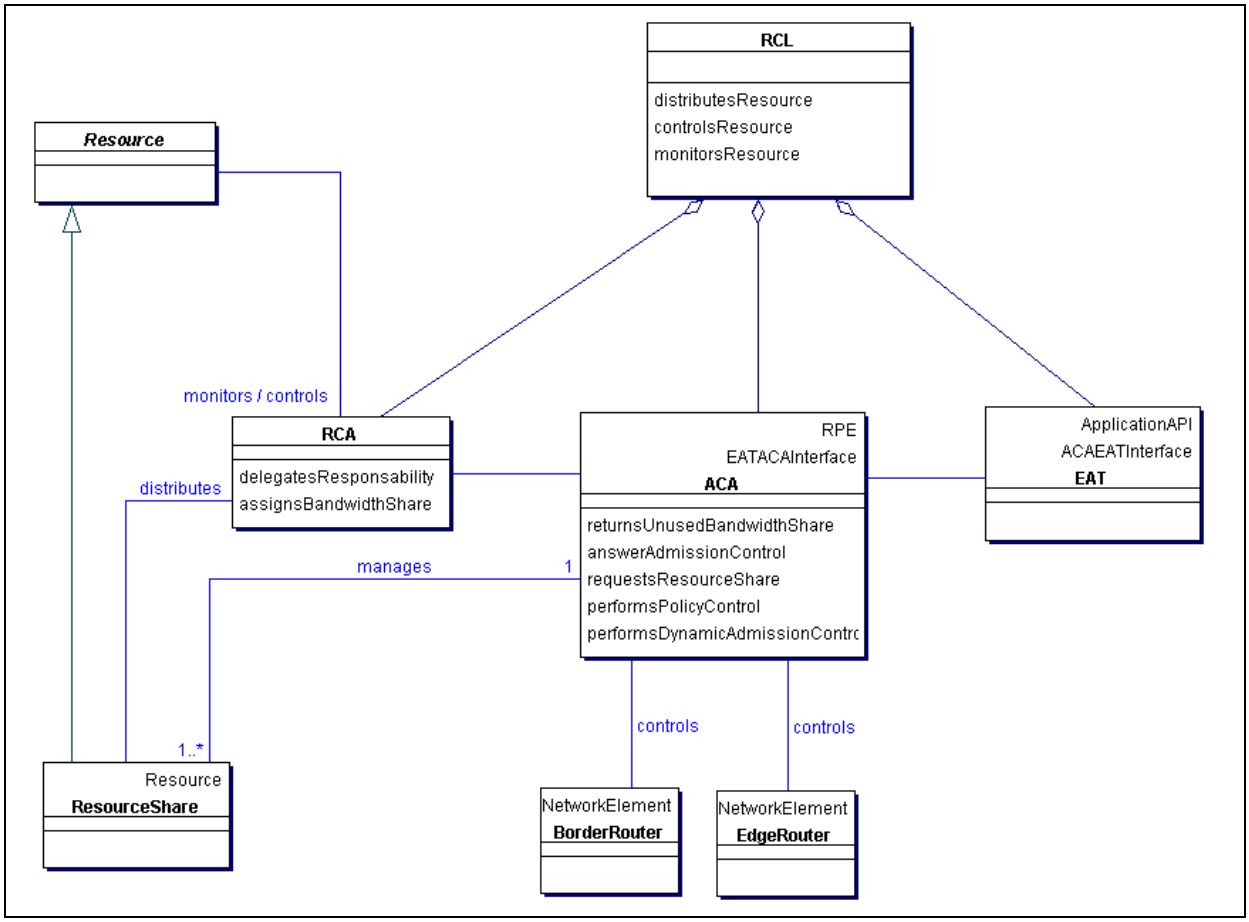


Figure 3-42: Class diagram for the resource control layer

3.6.4.1 Resource structure

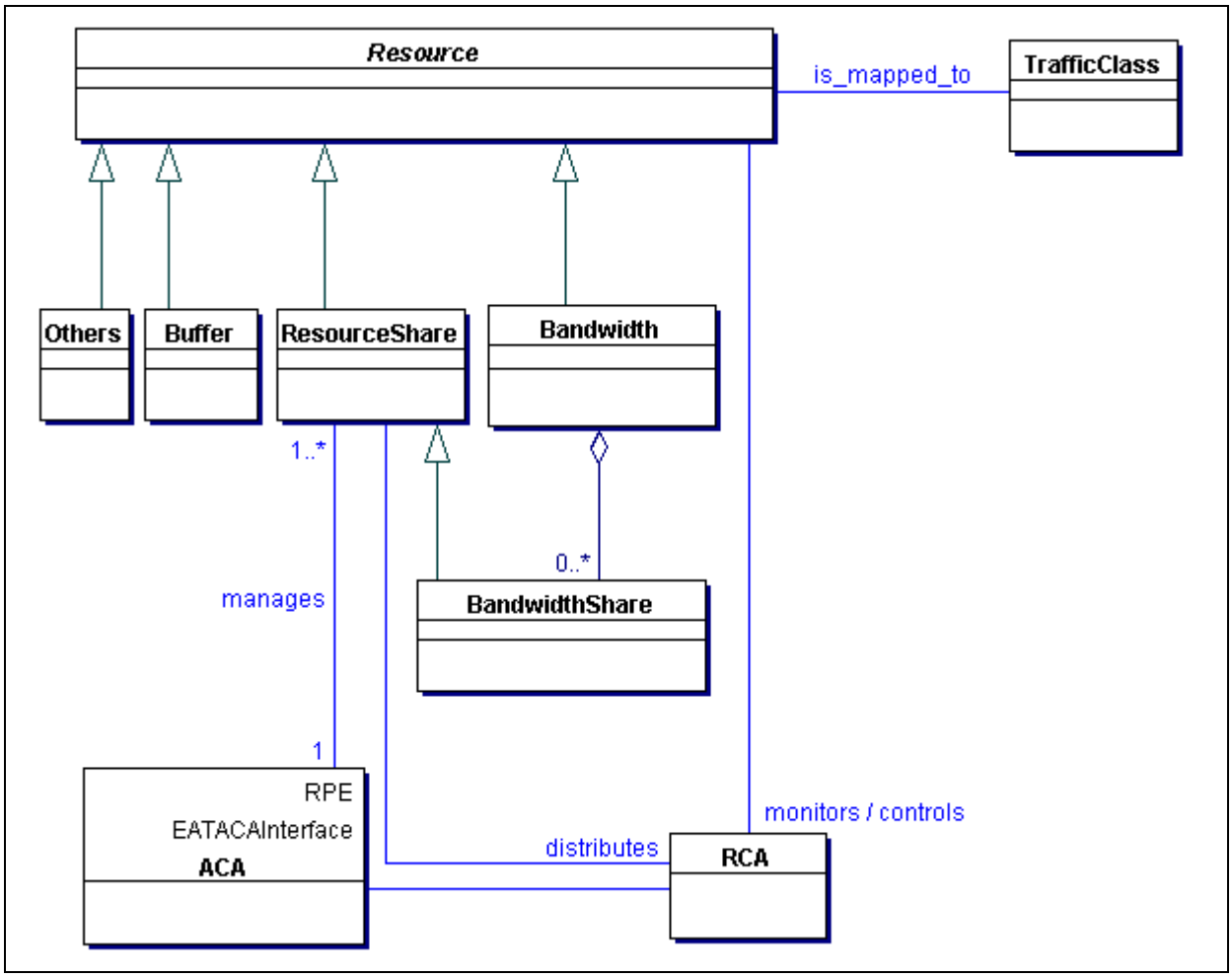


Figure 3-43: Class diagram for the resource structure

3.6.4.2 Resource Pool / distributor structure

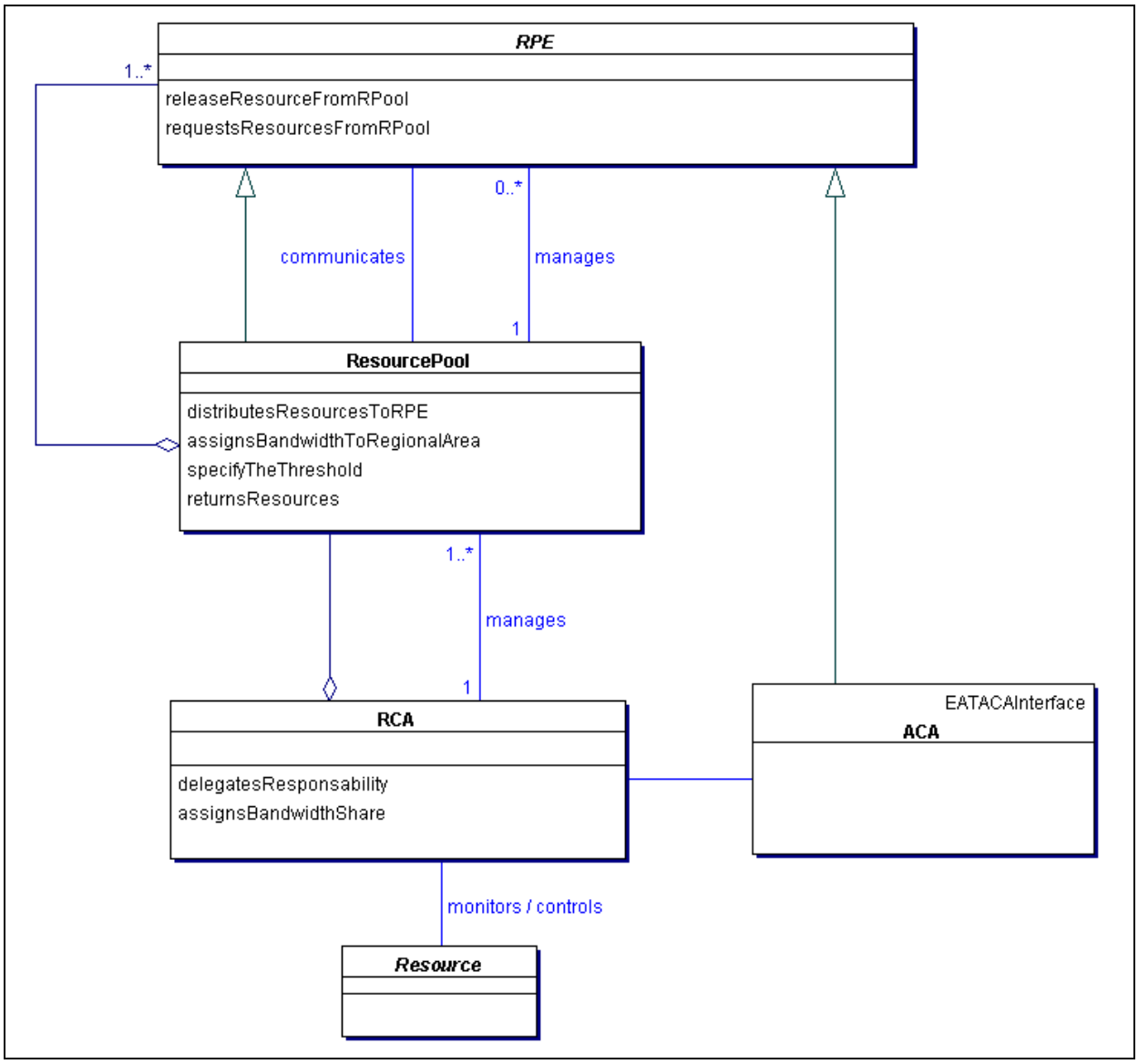


Figure 3-44: Class diagram for the resource pool structure

4 Coarse Design of System Components

This chapter is a functional specification of each part of the AQUILA architecture. Each chapter should show the assignments of each part, the functionality, which is included in each part and how it is implemented. This chapter should be useful for the implementation work packages.

4.1 Edge Router, Core Router

4.1.1 Overview

The general AQUILA network architecture is based on the DiffServ network concept. The main requirement for the AQUILA core network is scalability and reliability. Therefore the network elements should provide a general mechanism for the following features to support this functionality ([Cisco00/1], [Verma99]):

- Policy Routing
- Packet Filtering (dropping / shaping)
- QoS Weighted Service
- QoS Classification and Marking
- Metering
- Rate Limiting
- Committed Access Rate
- Congestion management, avoidance

The general router architecture (edge device and core router) is shown in Figure 3-7. It consists of the following modules:

- Management module for configuration and management purposes. It should support the COPS, CLI or SNMP protocols for communication between RCL layer and network devices.
- RSVP module for compatibility with IntServ architecture (only in edge devices)
- Routing module, it implements routing protocol (OSPF protocol).
- Ingress/Egress port modules (in core router PoS interface at physical layer).

The port module is composed of the following functional elements:

- Packet classifiers (MF Classifier and BA Classifier)
- Traffic conditioning functions, including packet metering, marking, shaping and dropping for the define in AQUILA service classes (PVO, PMM, PMC)
- Per-hop packet forwarding behaviours (PHB), implemented with the aid of scheduling and buffer management algorithms.

Figure 4-1 shows a logical view of a packet classifier and traffic conditioner within a router.

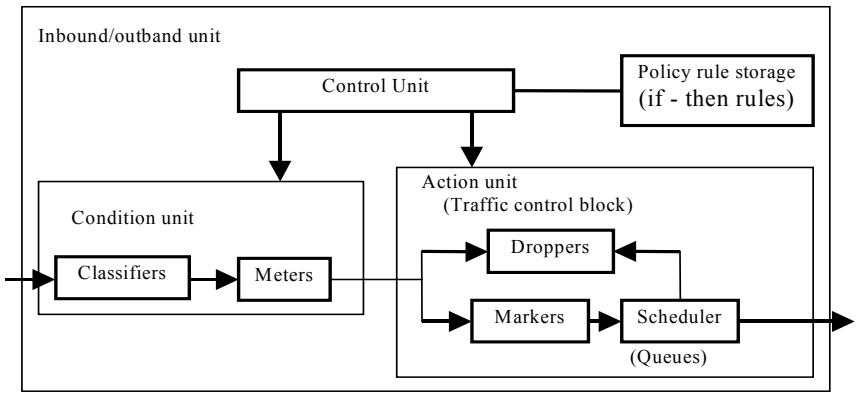


Figure 4-1: Router functionality

Within the AQUILA project different routers are available for the field trial:

Within the AQUILA project different routers from two vendors are available for the field trial:

- Cisco 1700
- Cisco 2514, 2513, Cisco 2621, 2611, Cisco 2900
- Cisco 3600, 3810
- Cisco 4700
- Cisco 7000, Cisco 7010, Cisco 7204, Cisco 7505, 7500
- Cisco 12000
- Catalyst 6000, 2924
- Cisco LS1010
- Cisco AS5300

- Unisphere Redstone ERX

The edge router are the ingress and egress of an ISP network. They build together with the connected core router the network. The functionality of a core router and an edge router is therefore different.

Edge router

In general the edge routers perform the following QoS functions:

- Packet classification
- Admission control
- Configuration management

The admission control is done at the ED and the accepted packets are classified and marked by either values in the packet header or by source interfaces. The forwarding of the marked packets is managed in the traffic shaper and packet dropper. Figure 4-2 gives an logical view of the edge router. The meter records the incoming packets and stores the statistics in data-bases. The data can be used in the marker and the shaper.

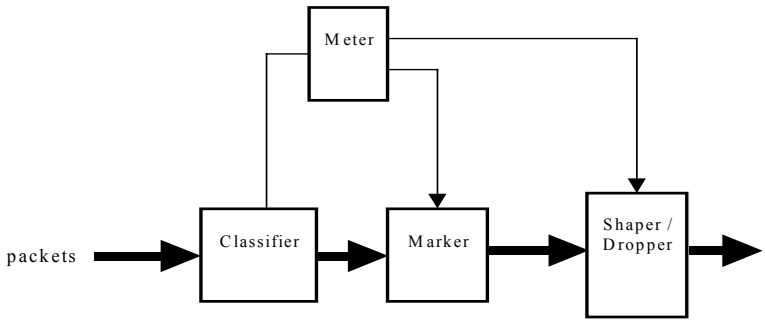


Figure 4-2. Router functionality

Core router

In general, backbone routers in the core of the network perform the following QoS functions:

- Congestion management
- Congestion avoidance

The admission control and the classification of the packets is part of the ED. The core router has to store and forward the incoming packets. The core router offer different queuing mecha-nisms to avoid congestion in the router.

4.1.2 Cisco Router

The CISCO routers provides a wide range of different QoS functionality's. The offered functionality of a router depends on the type and the used software (IOS release). The following subchapters give an overview of the different QoS functionality.

4.1.2.1 Classification

Packet classification features provide the capability to partition network traffic into multiple priority levels or classes of service (CoS). One can use Cisco IOS QoS policy-based routing (PBR) and the classification features of Cisco IOS CAR to classify packets ([Cisco00/3]). Border Gateway Protocol (BGP) policy propagation to propagate destination-based packet classification policy throughout a large network via BGP routing updates.

Committed Access Rate

The Committed Access Rate (CAR) and Distributed CAR (DCAR) is a multifaceted feature, that implements classification services and policing through rate limiting. The classification service of CAR allows the partition of a network into multiple priority levels or classes of service. Devices within a network can use the adjusted IP precedence to determine how to treat the traffic.

For the classification can policies based on physical port, source and destination IP or MAC address, application port, IP protocol type, or other criteria specifiable in access lists or extended access lists be used.

For a complete description of the CAR commands, refer to the Cisco IOS Quality of Service Solutions Command Reference ([Cisco00/1)].

CAR is supported on the following platforms:

- Cisco 1720
- Cisco 2600 series
- Cisco 3600 series
- Cisco 4500 series, Cisco 4700 series
- Cisco 7200 series
- Cisco 12000 series

Distributed CAR is supported on Cisco 7000 series routers with a Route Switch Processor (RSP)-based RSP7000 interface processor or Cisco 7500 series routers with a Versatile Interface Processor (VIP)-based VIP2-40 or greater interface processor. A VIP2-50 interface proc-

essor is strongly recommended when the aggregate line rate of the port adapters on the VIP is greater than DS3. A VIP2-50 interface processor is required for OC-3 rates.

Policy-Based Routing

Cisco IOS QoS PBR allows the following:

- Classify traffic based on extended access list criteria.
- Set IP Precedence bits.
- Route specific traffic to engineered paths, which may be required to allow a specific QoS service through the network.

Classification of traffic through PBR enables to identify traffic for different classes of service at the perimeter of the network and then implement QoS defined for each class of service in the core of the network using priority, custom, or weighted fair queuing techniques. This process obviates the need to classify traffic explicitly at each WAN interface in the core-backbone network.

Some possible applications for policy routing are to provide equal access, protocol-sensitive routing, source-sensitive routing, routing based on interactive versus batch traffic, or routing based on dedicated links.

4.1.2.2 Traffic Shaping and Policing Tools

Cisco IOS QoS includes traffic policing capabilities implemented through the rate-limiting aspects of CAR and traffic shaping capabilities provided by the Generic Traffic Shaping (GTS) and Frame Relay Traffic Shaping (FRTS) protocols.

The rate-limiting feature of CAR provides the network operator with the means to define Layer 3 aggregate or granular access, or egress bandwidth rate limits, and to specify traffic handling policies when the traffic either conforms to or exceeds the specified rate limits. Aggregate access or egress matches all packets on an interface or sub-interface. Granular access or egress matches a particular type of traffic based on precedence. The network operator can designate CAR rate-limiting policies based on physical port, packet classification, IP address, MAC address, application flow, and other criteria specifiable by access lists or extended access lists. CAR rate limits may be implemented either on input or output interfaces.

Generic Traffic Shaping (GTS) provides a mechanism to control the traffic flow on a particular interface. It reduces outbound traffic flow to avoid congestion by constraining specified traffic to a particular bit rate (also known as the token bucket approach), while queuing bursts of the specified traffic. Thus, traffic adhering to a particular profile can be shaped to meet downstream requirements, eliminating bottlenecks in topologies with data-rate mismatches.

4.1.2.3 Congestion Management

The router can use different queuing algorithms depending on the used IOS. The Cisco IOS software includes the following queuing tools ([Cisco99]):

- First In First Out
- Priority Queuing (PQ)
- Custom Queuing(CQ)
- Weighted Fair Queuing (WFQ)

FIFO provides basic store and forward capability. FIFO is the default queueing algorithm in some instances, thus requiring no configuration.

Priority Queuing is designed to give strict priority to important traffic and ensures that important traffic gets the fastest handling at each point where PQ is used. PQ can flexibly prioritize according to network, incoming interface, packet size, source/destination address, and so forth.

Custom Queuing reserves a percentage of the available bandwidth of an interface for each selected traffic type. If a particular type of traffic is not using the bandwidth reserved for it, then other traffic types may use the remaining reserved bandwidth.

For AQUILA the recommended scheduling algorithm is of WFQ type (e.g. WFQ, WF^2Q, etc.). WFQ is one of Cisco's premier queuing techniques. It is a flow-based queuing algorithm that does two things simultaneously: It schedules interactive traffic to the front of the queue to reduce response time, and it fairly shares the remaining bandwidth between high bandwidth flows. WFQ is designed to minimize configuration effort and adapts automatically to changing network traffic conditions.

The WFQ algorithm also addresses the problem of round-trip delay variability. If multiple high-volume conversations are active, their transfer rates and interarrival periods are made much more predictable. WFQ ensures that queues do not starve for bandwidth, and that traffic gets predictable service.

Cisco IOS software also provides distributed weighted fair queuing (D-WFQ), a special high-speed version of WFQ designed initially for IP-only networks. D-WFQ is currently available only on VIP processors and only in Cisco IOS release 11.1cc, a special version for 7500 VIP processors.

4.1.2.4 Congestion Avoidance

Congestion avoidance techniques monitor network traffic loads in an effort to anticipate and avoid congestion at common network bottlenecks, as opposed to congestion management

techniques that operate to control congestion once it occurs. The primary Cisco IOS congestion avoidance tool is Weighted Random Early Detection (WRED.

These techniques are designed to provide preferential treatment for priority class traffic under congestion situations while concurrently maximising network throughput and capacity utilisation and minimising packet loss and delay.

Router behaviour allows output buffers to fill during periods of congestion, using tail drop to resolve the problem when WRED is not configured. During tail drop, a potentially large number of packets from numerous connections are discarded because of lack of buffer capacity. This behaviour can result in waves of congestion followed by periods during which the transmission link is not fully used. WRED obviates this situation proactively by providing congestion avoidance. That is, instead of waiting for buffers to fill before dropping packets, the router monitors the buffer depth and performs early discards on selected packets sent over selected connections.

WRED is the Cisco implementation of the Random Early Detection (RED) class of congestion avoidance algorithms. When RED is used and the source detects the dropped packet, the source slows its transmission. RED is primarily designed to work with TCP in IP internetwork environments.

4.1.2.5 Overview

Table 4-1 and Table 4-2 give an overview of different IOS releases ([Cisco00/2]).

QoS Technique	Cisco Systems Device	11.1	11.2	11.3	11.1(cc)
Priority Queuing (PQ), Custom Queuing (CQ)	Cisco 7500, 7200	X	X	X	X
	RSM (Catalyst 5000),Cisco 4700, 4500, 4000, 3600, 2500	X	X	X	
	Cisco 2600			X	
Weighted Random Early Detection (WRED)	Cisco 7500, 7200, RSM (Catalyst 5000), 4700, 4500, 4000, 3600, 2500		X	X	
	Cisco 2600			X	
	Cisco 7500 VIP (uses DWRED)				X
Weighted Fair Queuing (WFQ)	Cisco 7500, 7200	X	X	X	X
	Cisco 4700, 4500, 4000, RSM (Catalyst 5000), 3600, 2500	X	X	X	

QoS Technique	Cisco Systems Device	11.1	11.2	11.3	11.1(cc)
	Cisco 2600			X	
Class-Based Weighted Fair Queuing (CBWFQ) Using Distributed Weighted Fair Queuing (DWFQ), Using Fair Queuing and QoS Group DWFQ	Cisco 7500 VIP				X
Policy-Based Routing (PBR) (also called colouring)	Cisco 7500, 7200, RSM (Catalyst 5000), 4700, 4500, 4000, 3600, 2500		X	X	
	Cisco 2600			X	
Generic Traffic Shaping (GTS)	Cisco 7500, 7200, RSM (Catalyst 5000), 4700, 4500, 4000, 3600, 2500		X	X	
	Cisco 2600			X	
Frame Relay Traffic Shaping (FRTS)	Cisco 7500, 7200, RSM (Catalyst 5000), 4700, 4500, 4000, 3600, 2500		X	X	
	Cisco 2600			X	
Committed Access Rate (CAR) Classification	Cisco 7500, 7500 VIP, 7200				X
Committed Access Rate (CAR) Rate Limiting	Cisco 7500, 7500 VIP, 7200				X
Resource Reservation Protocol (RSVP)	Cisco 7500, 7200, RSM (Catalyst 5000), 4700, 4500, 4000, 3600, 2500		X	X	
Resource Reservation Protocol (RSVP)	Cisco 7500, 7200, RSM (Catalyst 5000), 4700, 4500, 4000, 3600, 2500		X	X	

Table 4-1: Supported devices and QoS Techniques for Cisco IOS Software Release 11.x Devices

QoS Technique	Cisco Systems Device	12.0	12.0(5)T	12.0(5)XE
Priority Queuing (PQ), Custom Queuing (CQ)	Cisco 7500	X	X	X
	Cisco 7200	X	X	X
	Cisco 7100			X
	Catalyst RSM (Catalyst 5000), 4700, 4500, 4000, 3600, 2600, 2500	X	X	
Weighted Random Early Detection (WRED)	Cisco 7500, 7200, RSM (Catalyst 5000), 4700, 4500, 4000, 3600, 2500	X	X	X
	Cisco 7500 VIP (uses DWRED), 7200	X	X	X
	Cisco 7100			X
	Catalyst RSM (Catalyst 5000), RSM VIP (Catalyst 5000), 4700, 4500, 4000, 3600, 2600, 2500	X	X	
Weighted Fair Queuing (WFQ) (or Fair Queuing (FQ) where indicated)	Cisco 7500, Cisco 7500 VIP (uses FQ)	X	X	X
	Cisco 7200	X	X	X
	Cisco 7100			X
	Catalyst RSM (Catalyst 5000), RSM VIP (Catalyst 5000; uses FQ), 4700, 4500, 4000, 3600, 2600, 2500	X	X	
Class-Based Weighted Fair Queuing (CBWFQ) Using Distributed Weighted Fair Queuing (DWFQ), Fair Queuing and QoS Group DWFQ	Cisco 7500 VIP, RSM VIP (Catalyst 5000)		X	
Class-Based Weighted Fair	Cisco 7500		X	X

QoS Technique	Cisco Systems Device	12.0	12.0(5)T	12.0(5)XE
Queuing (CBWFQ)				
	Cisco 7500 VIP			X
	Cisco 7200		X	X
	Cisco 7100			X
	Cisco 4700, 4500, 3600, 2600, 2500		X	
IP RTP Priority	Cisco 7500		X	X
	Cisco 7200		X	X
	Cisco 7100			X
	Cisco 4700, 4500, 3600, 2600, 2500		X	
Policy-Based Routing (PBR) (also called colouring)	Cisco 7500, 7500 VIP, 7200, 7100, RSM (Catalyst 5000), RSM VIP (Catalyst 5000), 4700, 4500, 3600, 2600	See note		
	Catalyst 4000	X	X	
	Cisco 2500	X	See note	
Generic Traffic Shaping (GTS)	Cisco 7500	X	X	X
	Cisco 7200	X	X	X
	Cisco 7100			X
	Catalyst RSM (Catalyst 5000), 4700, 4500, 4000, 3600, 2600, 2500	X	X	
Frame Relay Traffic Shaping (FRTS)	Cisco 7500	X	X	X
	Cisco 7200	X	X	X
	Cisco 7100			X

QoS Technique	Cisco Systems Device	12.0	12.0(5)T	12.0(5)XE
	Catalyst RSM (Catalyst 5000), 4700, 4500, 4000, 3600, 2600, 2500	X	X	
Enhanced FRTS with Frame Relay Fragmentation (FRF12), Frame Relay Fair Queue, and Frame Relay Voice Bandwidth	Cisco 7200		X	X
	Cisco 3600, 2600		X	
Committed Access Rate (CAR) Classification (also called colouring)	Cisco 7500, 7500 VIP	X	X	X
	Cisco 7200	X	X	X
	Cisco 7100			X
	Catalyst RSM (Catalyst 5000), RSM VIP (Catalyst 5000), 4700, 4500, 3600, 2600	X	X	
	2500		X	
Committed Access Rate (CAR) Rate Limiting	Cisco 7500, 7500 VIP	X	X	X
	Cisco 7200	X	X	X
	Cisco 7100			X
	Catalyst RSM (Catalyst 5000), RSM VIP (Catalyst 5000), 4700, 4500	X	X	
	2500		X	
Weighted Round Robin (WRR)	Cisco 8510, 8540	X		
Resource Reservation Protocol (RSVP)	Cisco 7500	X	X	X
	Cisco 7200	X	X	X

QoS Technique	Cisco Systems Device	12.0	12.0(5)T	12.0(5)XE
	Cisco 7100			X
	Cisco 4700, 4500, 3600, 2600, 2500	X	X	
Network-Based Application Recognition (NBAR)	Cisco 7200, 7100			X

Table 4-2: Supported devices and QoS Techniques for Cisco IOS Software Release 12.x Devices

With the IOS version 12.1 WFQ and PQ can be used together. The configuration and the functionality's in IOS 12.1 are nearly platform independent. For the field trials the IOS version 12.1 should be used, if it is available for all operators.

4.1.3 Redstone Router

The Redstone Policy Manager supplies DiffServ mechanism. An intuitive CLI and the ability to provide policy templating allow a network provider to easily create or reuse an already configured policy on a new user. The Policy Manager provides for the following services:

- Policy Routing
- Packet Filtering
- QoS Weighted Service
- QoS Classification and Marking
- Rate Limiting
- Committed Access Rate

Policy Routing is a service whereby a packet flow's destination port is predefined or nailed-up. A route lookup is not performed on the packet. On ingress, the packets are classified into a packet flow and sent to the pre-configured destination port.

Packet Filtering is a service whereby a packet or set of packets are dropped based on values in it's header.

QoS Weighted Service is a service where packets are treated differently through the chassis. Treated differently means one packet flow gets preferential treatment with regard to buffer management and scheduling time. A typical application would be to provide two types of

services to a customer, gold and bronze. Under load, the gold service would get a proportionately higher resource usage and statistically better throughput than the bronze service.

QoS Classification and Marking is a service where packets are classified by either values in its header or by source interface and are correspondingly marked with a pre-configured value. This service is important to network providers that wish to process packets differently outside the context of our chassis.

Rate Limiting is a service where rates below the physical port line rate can be configured and enforced. The out of profile packets are dropped.

Committed Access Rate is a service that provides a two rate three colour marker mechanism where packets are marked on ingress based on with of the three rate categories they fall into. On egress, when the uplink becomes congested, the packets are queued based on their marking. For example, green marking would occur on in profile packets, yellow on between in profile and out of profile, and red for out of profile packets. This way, committed rates, green coloured packets, can be guaranteed.

The Policy Manager provides a rich CLI, providing the capability to combine some of the above services to map into a service level agreement (SLA). For instance a user could request a rate-limit service on certain types of classified packet flows with some packets flows being destined for a gold service and the others for best effort service.

4.2 Admission Control Agent

Figure 4-3 gives an overview over the structure of the ACA

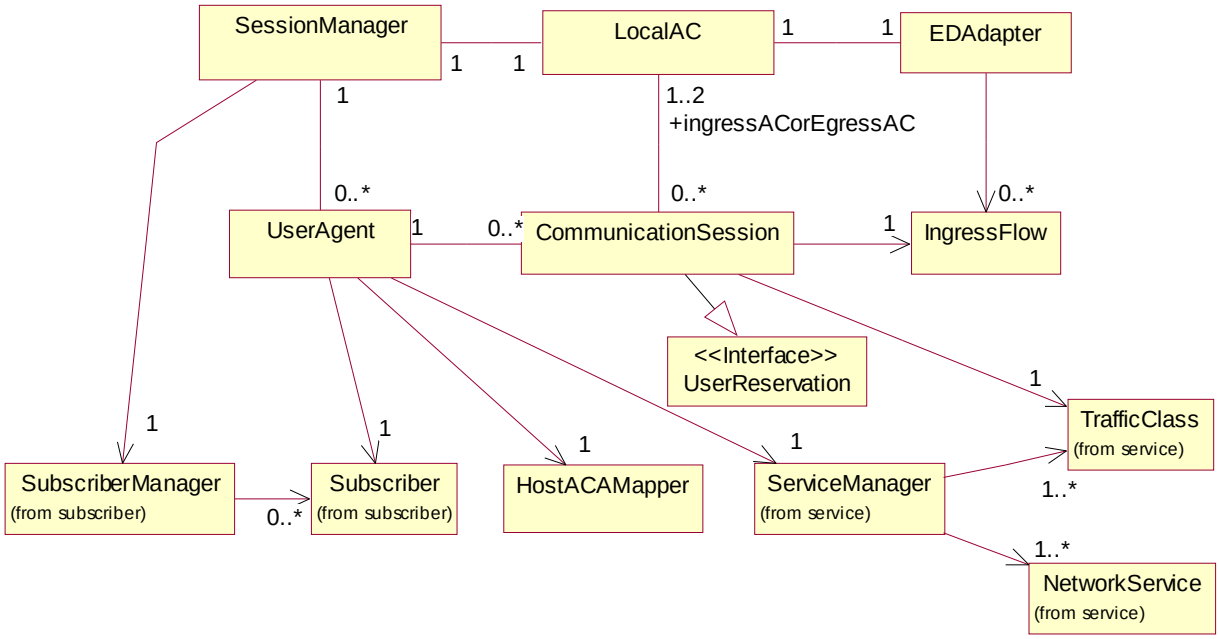


Figure 4-3: Structure of the ACA

The main components, instantiated only once per ACA and existing during all the lifetime of this ACA are SessionManager, LocalAC and EDAdapter.

For EATs running on hosts connected to the network via an ACA's associated ED, this ACA is the manager ACA. The EAT uses the **SessionManager** of its manager ACA to start a login session.

LocalAC performs local admission control. For each request, it tries to allocate resources out of the resource pool assigned to this ACA. If this succeeds, the request can be granted.
If the link between the edge device corresponding to this ACA and the core is a "low speed link" i.e. has a capacity below some defined value, then LocalAC performs on each request an additional check to find out, whether this low speed link supports the request. (In contrast to the resource allocation, where only the resources available in a specific traffic class are considered, this check takes into account allocations of all traffic classes on this specific link.)

LocalAC does not process directly requests from the EAT. Instead, it processes requests coming from other ACAs (the manager ACAs of the requesting EATs).

EDAdapter is an encapsulation of the components performing the communication with the ED. A LocalAC of an ingress ED uses its EDAdapter to set up classifier, marker and policer for a flow for which a reservation has been made.

UserAgent, CommunicationSession and IngressFlow are classes for objects that can be dynamically instantiated and destroyed again during the lifetime of an ACA.

Each time an EAT performs a successful login via the SessionManager, a **UserAgent** is instantiated, which represents an active, authenticated end-user connected to this EAT. The UserAgent provides the interface used by the EAT to establish reservations.

Each reservation is represented by a **CommunicationSession**, which bundles all the information relevant for this reservation. It is associated with up to two LocalACs, the LocalAC of the sender ACA and optionally the LocalAC of the receiver ACA. CommunicationSession implements the interface **UserReservation**, which is the view that the EAT has of a CommunicationSession. (The EAT can retrieve usage data for each UserReservation and it can release the reservation.)

An **IngressFlow** represents an established classifier/marker/policer at an ingress ED. An established CommunicationSession is associated with such an IngressFlow. Also, the EDAdapter knows all IngressFlows of the corresponding ED.

The **HostACAMapper** allows to retrieve the ACA responsible for a given host. This class is an encapsulation of the mechanism which provides the mapping. It may exist once per ACA, or one HostACAMapper may serve several or all ACAs in a domain.

SubscriberManager, Subscriber, ServiceManager, TrafficClass and NetworkService don't belong to the core parts of the ACA. They exist independently of ACA instances and by this form rather independent parts of the RCL. As they are used mainly by the ACA, they are described in this chapter.

The **SubscriberManager** provides access to the subscriber database. It might exist only once per domain, which would mean a central subscriber database. Another solution would be one SubscriberManager per ACA, which would be more suitable when a distributed database shall be used. In any case, the SubscriberManager allows to retrieve instances of class **Subscriber**, which represents the information associated to individual subscribers. Subscriber objects are persistent. There is one object per subscription which exist as long as this associated subscription.

The **ServiceManager** provides access to the service/traffic class database, which holds the persistent **NetworkService** and **TrafficClass** objects. It allows to retrieve all NetworkService-objects (a feature that is used by the EAT) and to map a NetworkService to a TrafficClass. A NetworkService object holds all the information describing an individual network service. A TrafficClass object holds information belonging to a traffic class. This will cover information needed by the EDAdapter (e.g. DSCPs) as well as rules how to compute e.g. an effective bandwidth.

In the following subchapters of this chapter 4.2, if operations are described using Java syntax, this is to be regarded to be pseudo code, not the final operation description on implementation level.

4.2.1 Processing of a Reservation Request

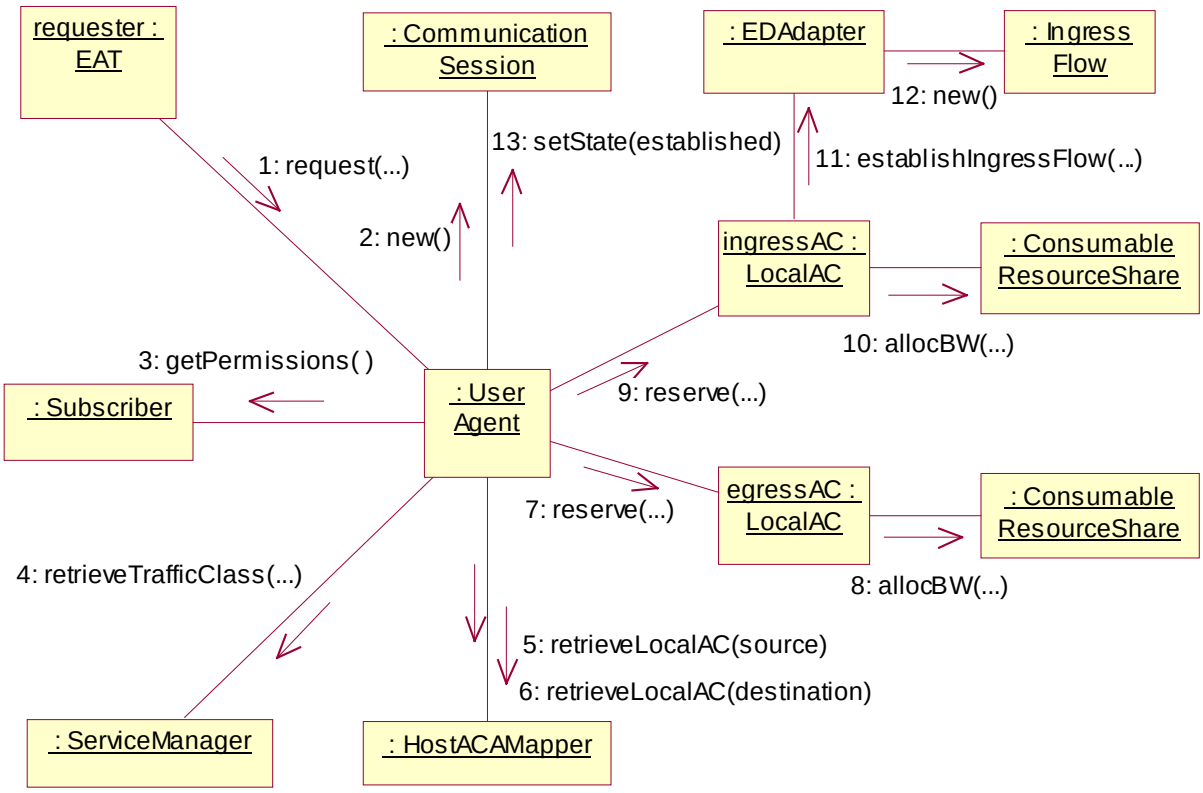


Figure 4-4: Processing of a Reservation Request

The sequence from Figure 4-4 is described in the following. It assumes that every call is successful. (In case of a negative response the sequence would be interrupted, everything already established would be reset, and the EAT would receive a negative response on its request.)

1. A reservation request is initiated by an EAT.

2. The UserAgent representing this ACA creates a CommunicationSession object, with a status like “being established” at the beginning.

3. The UserAgent checks whether the corresponding subscriber has the proper rights to do such a request.

4. The UserAgent finds out the TrafficClass (from the network service requested by the EAT, possibly using other request parameters, too.)

5. The UserAgent retrieves a reference to the LocalAC object of the sender ACA (ingressAC).

6. Assuming that egress reservation is done in this traffic class, the UserAgent retrieves a reference to the LocalAC object of the receiver ACA (egressAC).

7. The UserAgent calls the egressAC to make the egress reservation.

8. The egressAC uses its local resources to make the egress reservation. If the egress ED is connected to the core with a low speed link, the egressAC performs an additional, link specific check to make sure the link is not overloaded.

9. The User UserAgent calls the ingressAC to make the ingress reservation.

10. The ingressAC uses its local resources to make the ingress reservation. If the ingress ED is connected to the core with a low speed link, the ingressAC performs an additional, link specific check to make sure the link is not overloaded.

11. The ingressAC calls its EDAdapter to establish the flow, i.e. set up classifier/marker/policer state at the ingress ED.

12. The EDAdapter establishes classifier/marker/policer state, which is represented in the model by a new IngressFlow object.

13. When all these actions have been successfully completed, the CommunicationSession is considered to be established.

4.2.2 Subscriber database

The class **SubscriberManager** offers a method

```
Subscriber subscriberByLoginInfo(String loginInfo);
```

which is used by the ACA's SessionManager to retrieve a Subscriber object for a given loginInfo, comprising e.g. the account name and some authentification info like a password. The method may return a null reference in case of invalid loginInfo, or it may throw an exception (which would allow to transmit some kind of "failure reason").

Objects of class **Subscriber** represent subscriber information. This comprises the following:

- account name
- password (or "shared secret" for authentification protocols using such a thing, e.g. CHAP)
- subscribed network services, and for each network service:

- usage restrictions (e.g. maximal resource usage, time of day or day of week restrictions)
- accounting rules (how is accounting done for this network service for this subscriber)

- current billing info
- "offline user data" like name, postal address etc.
- marketing profile data (e.g. age, education, profession, interests etc.)

Not all information listed above is relevant to the RCL. It has to be decided what information will be available within the Subscriber objects.

4.2.3 Network service / traffic class database

The class **ServiceManager** offers the methods

```
NetworkService[] getAllNetworkServices();
NetworkService   getNetworkService(int serviceID);
```

which allow to retrieve all network services as well as a single network service (by its id). The methods are for usage by the EAT.

An object of class **NetworkService** represents information about a specific network service. This comprises the information described in chapter 3.2, as far as relevant for the RCL.

Another method of ServiceManager is

```
TrafficClass[] retrieveTrafficClass(NetworkService, TrafficDescription);
```

which is used by the UserAgent to retrieve the appropriate traffic class to be used for a given reservation request.

As described in the overview, a **TrafficClass** object holds the information needed to establish a marker within the ingress ED of a flow (i.e. DSCPs). Furthermore, the class offers methods how to calculate with traffic descriptions. This could be an effective bandwidth formula or operations like "plus", "minus", or "lessOrEqual". These calculation rules are an essential part of the RCL's main task, the resource management.

4.3 Resource Control Agent

The Resource Control Agent (RCA) for the first trial is responsible for the distribution of the network resources to the ACAs and for performing admission control decisions upon reservation requests. To be more specific, the network topology is divided in a hierarchical manner to various levels of sub-networks: backbone network, sub-areas, subordinated sub-areas. The task of creating this tree structure is not a responsibility of the RCA, but another component has to be specified. This network structure might be defined by the network administrator or by an algorithm that might take into account routing information, traffic utilisation, local policies etc. For the first trial this information will be determined by the network administrator

and provided to the RCA statically e.g. using configuration files or a database, in order to perform the resource distribution.

The RCA makes an identical tree structure based on the concept of the resource pools, where each resource pool could be mapped to a sub-network (sub-area etc.) of the network topology and it is responsible for computing the resource allocations for the resource pools attached below it (its children). The inner nodes of the tree are resource pools (ResourcePoolComposite), while the leafs (ResourcePoolLeaf) represent ACAs. Therefore, the resources assigned to each ResourcePoolLeaf are determined by the ResourcePoolComposites above it (i.e. its parent). The algorithm that determines the bandwidth allocations has to be specified.

Finally, the RCA should configure appropriately the ResourcePoolLeafs and be able to answer the requests for bandwidth allocations or de-allocations. The ResourcePoolLeafs perform the admission control decisions based on the bandwidth allocations determined by the resource pools and on some parameters (high, low watermarks) specific to the adopted admission control mechanism. These parameters can be dynamically specified and this is a task of the RCA. Although, if the ED associated to an ACA has a low speed "edge link", the ACA should check to find out whether the edge link supports this reservation before calling the alloc().

The diagram illustrated in Figure 4-5 describes the above concepts using the UML notation (it is **not** a design model).

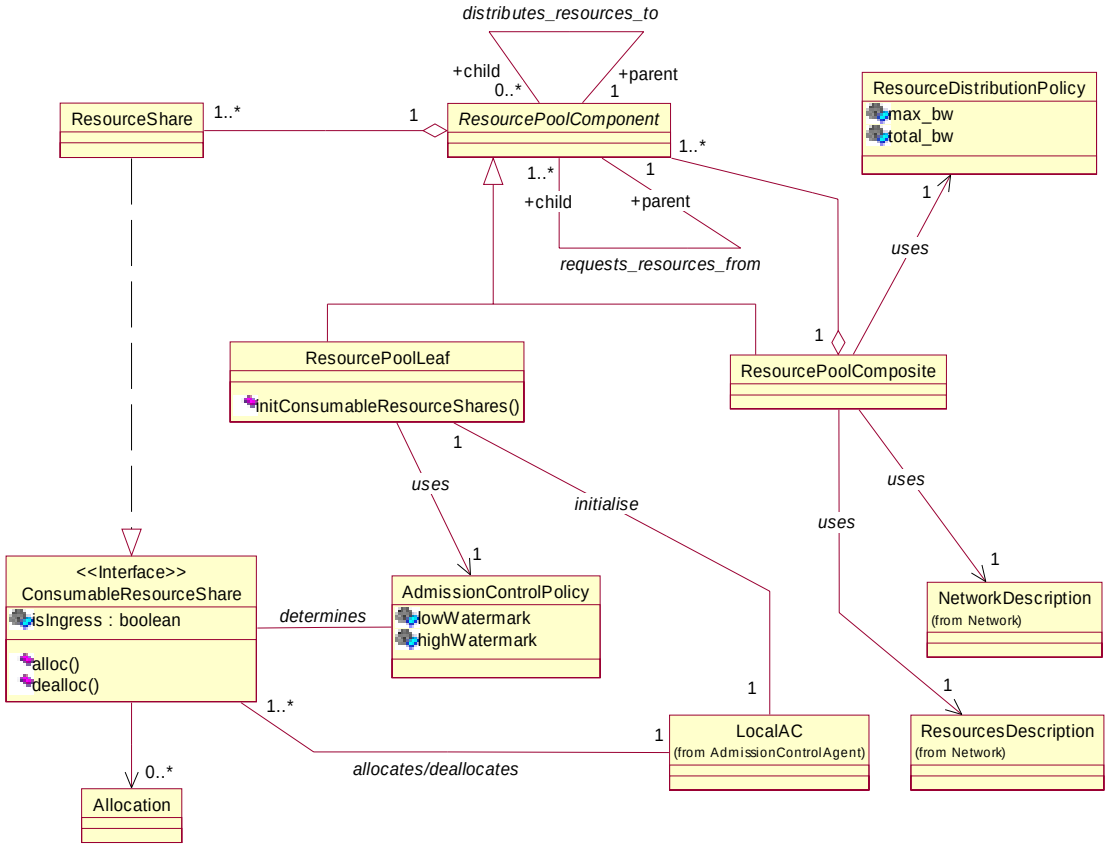


Figure 4-5: Structure of the RCA

In order to depict the tree structure of the resource pools the composite pattern is used [GHJV95], where the following classes are defined: **ResourcePoolComponent**, **ResourcePoolComposite** and **ResourcePoolLeaf**. The tree of the resource pools is created using the information retrieved by the network package: **NetworkDescription** and **ResourcesDescription**. This information is used to create the resource pools and assign initial resources to each of them. The network package is managed by the network administrator.

The parents ResourcePoolComponents distribute resources to the child ResourcePoolComponents, except from the leafs of the tree structure that receive resources from the resource pools above them and initialise the corresponding ACAs appropriately. Moreover, each ResourcePoolComposite uses a **ResourceDistributionPolicy** to distribute resources to the resource pools below it.

Each ResourcePoolComponent has been assigned some resources by the network administrator (start-up configuration) or the resource pool above it, represented by the class ResourceShare. Each **ResourceShare** object refers to a specific traffic class and a specific direction (ingress or egress).The interface **ConsumableResourceShare** is implemented by the ResourceShare objects and represents, what an ACA sees from a ResourceShare object, i.e. ConsumableResourceShare will only be used at the interface between the ACAs and the RCA.

Whenever, an ACA "runs out" of resources the corresponding ResourcePoolLeaf, based on the **AdmissionControlPolicy** reconfigures the ACA accordingly. If needed the ResourcePoolLeaf might request more resources than the allocated ResourceShare from its parent resource pool and so on.

The ResourcePool has the following values for each traffic class and direction (ingress, egress):

- Max_bw: the maximum amount of resources assignable to this pool
- Total_bw: the total amount of resources assigned to this pool
- HighWatermark: a high watermark (only the ResourcePoolLeafs)
- LowWatermark: a low watermark (only the ResourcePoolLeafs)
- Spent: the amount of resources currently spent

The goal is to have HighWatermark >= spent >= LowWatermark. If after an allocation or a release the above formula is true no redistribution of resources is required. If after a reservation the high watermark is exceeded the ResourcePoolLeaf will request additional resources from its parent pool. If after a release the spent value is smaller than the low watermark the ResourcePoolLeaf will return resources to its parent pool. A request is immediately rejected if spent will become greater than the Max_bw.

4.3.1 Network Database

As is already mentioned the RCA needs information in order to construct the resource pool tree and assign the appropriate parameters (Max_bw, Total_bw, watermarks, …) to each resource pool. This information should be contained in a database (*network database*). The RCA analysis model shows the two entities that are responsible for the interaction with the database (NetworkDescription, ResourcesDescription).

The exact contents of such a database have to be specified.

4.3.2 Resource Distribution Algorithm

To be discussed.

4.4 End-user Application Toolkit

The end-user application toolkit (EAT) is an application that aims to provide access to end-user applications to QoS features. The EAT is a middleware between the end-user applications (for example a video conferencing tool or a video on demand service) and the network infrastructure (for example the AQUILA network).

The tasks of the EAT are: to allow legacy applications (QoS-aware and non-QoS-aware) to benefit from QoS features (1st trial), and to allow, by the way of an API (to be developed for the 2nd trial), the implementation of QoS-aware EAT-based applications.

This toolkit should provide reusable and generic components for both client and server sides of applications. The toolkit should be used by end-users as well as by application developers.

The EAT should support a variety of operating systems, network and reservation protocols, by providing a logical abstraction that hides the low level details of particular systems. The EAT should be structured in such a way that it will allow easy update. For this reason a number of adapters or plug-ins is used that are specific to a particular system or protocol. In detail, the EAT may include some platform-specific components while the whole toolkit is platform independent. On the other hand, it supports different protocols, for example QoS protocols.

The EAT will not be transparent for the legacy applications, but will be mostly transparent for new EAT-based applications (using the API).

4.4.1 Goals definition

4.4.1.1 Must criteria

Towards the end-user the EAT will:

- Offer QoS features/functionality.
- For non-professional end-users: Offer in ergonomic form (the session characteristics) the network services (corresponding to the SLAs concluded by the end-users, and mapped into corresponding quality criteria for the chosen applications) to the end-users.

  The session characteristics parameters should correspond to the end-user's perception of the application. As it is difficult for a normal end-user to be aware of the technicality of the network services (like PVO and PMM), their peculiarities, and the technical parameters, the EAT should encounter the user's perception of a network service. A strategy could be to use terms like "CD quality" to describe an audio quality feature, "big window size" for video, "ISDN audio quality" for IP telephony, etc.
- For professional end-users: Let the end-user requests for precise session characteristics.

  Map the network services (corresponding to the SLAs concluded by the end-users) into a set of parameters or session characteristics. The professional end-user should be able to set the values corresponding to the parameter set of the QoS request.
- Support the end-user in the utilisation of legacy applications, by providing special GUIs, and functionality.
- Provide a set of GUIs to support the activities of the end-user.

Towards the overall architecture the EAT will:

- Be a middleware application.
- Be a platform independent application.
- Enable the migration of legacy end-user applications to QoS-awareness (already for the 1st trial).
    - Legacy applications could be:
        - Streaming video applications:
            - Microsoft Streaming Media.
            - Real (Player + Server).
            - Apple Streaming QuickTime.
        - Voice over IP: WinSIP from Siemens
        - Multiplayer network games
    - Provide some kind of proxy mechanisms (for example to identify the applications, to extract control plane information).
    - Provide a GUI to support the input of control plane information (chosen by the end-user, and at this stage represented by the session characteristics).
- Enable the provision of existing QoS-aware end-user applications with AQUILA QoS features (perhaps already for the 1st trial).
    - Existing QoS-aware applications could be:
        - Applications which are based on the Windows 2000 QoS-API, e.g. the video conferencing tool NetMeeting, Phone Dialler by Microsoft.
    - Provide a proxy + daemon mechanism (for example to intercept control plane information).
- Enable the construction of QoS-aware end-user applications (only for the 2nd trial).
    - New EAT-based applications could be:
        - Based on the Java Media Framework (JMF).
        - Enhancements of Q-Systems applications.
    - Provide a QoS API.
- Provide scalable mechanisms for applications, which produce a large number of short-lived sessions with unclear requirements (e.g. WWW) and where dedicated reservations are inappropriate.

- Communicate with the ACA.
- Enable information exchange with the network/ACA.
- Play the role of the requester.
- Request resource reservation from the network/ACA.
- Map the chosen session characteristics into the corresponding network services and request them by the ACA.
- Know the SLAs concluded between the end-users and the network providers and take them into account when mapping the network services from the ACA into session characteristics.
- Ensure compatibility with various methods and protocols for communicating QoS requests between the end-user applications and the Resource Control Layer.

  (Special schemes for integrating application level protocols (e.g. H.323, SIP) with QoS control will be taken into consideration.)

4.4.1.2 Wished criteria

- The EAT should have a flexible architecture to add further application adapters as well as further network adapters as needed.
- The EAT can be a distributed application.
- The EAT can reside on another host than the application one.
- The EAT may intercept RSVP and its messages (PATH, RESV, …).

  Concerning the RSVP topic: In general, the EAT is aimed to support applications and systems that use RSVP for QoS signalling. Examples are applications for Windows 2000 like NetMeeting, for instance.

  On the other hand, the EAT uses a unique well-defined interface to request for QoS reservations at the ACA. Therefore, the preferred solution is *not* to have an additional request interface for RSVP in parallel but to intercept the RSVP messages at the EAT and to map them into reservations requests (by CORBA) that fit to the AQUILA approach. Because of the receiver orientation of RSVP, the translation should be mainly based on the RESV message. A possible scenario might be:

  1. The sender application/system sends its PATH message directly to the receiver. That means, the message will be ignored by *every* AQUILA component (EAT, ACA, ED, …) in between.

2. The receiver creates a RESV message if it is RSVP-aware and if it wants to reserve.

3. The receiver's EAT (i.e. the egress EAT) intercepts this message by using a proxy mechanism for example.

4. The receiver's EAT tries to map the RSVP reservation into an AQUILA one and requests for it at the ACA.

5. If the reservation succeeds, the receiver EAT may send a RESV CONF message towards the receiver.

There are still some open points to be discussed:

- The (sender, ingress) EAT may intercept the PATH message for sender oriented reservations as well. That mainly depends on the applications.

- The EAT may intercept only the first RESV(/PATH) message in order to create an AQUILA reservation from that. The following RESV messages may be ignored and just answered and forwarded.

4.4.1.3 Restriction criteria

- The EAT will not support every (kind of) legacy application.

- The EAT will not include every possible components on every platform.

4.4.1.4 Opened questions

- Is an EAT on every side requested?

4.4.2 Target groups

The EAT enables a differentiated utilisation of the network. Therefore users wanting to have access to a network, which provides differentiated classes of services, by the way of different kind of end-user applications are potential users of the toolkit.

There are many kinds of users:

- Private users for: playing, online-banking/brokerage, web surfing... These users are mostly not network specialists/professionals. They are mostly using the capabilities of the network for entertainment purposes.

- For work: this group of users is not consisted of network specialists. They are using applications designed for special purposes (remote diagnosis, remote control, etc.).

- Content/online providers (business users): providing contents, QoS-aware applications, and/or online services to other end-users (private users) of the network.

- Application developers: they will use the API of the EAT to develop their applications.

The range of user is very bright (from non-professional end-user of the network to professional end-user of the network over entertainment purposes to business purposes). Therefore the EAT should be highly adaptive and flexible.

4.4.3 Product interfaces

The EAT has interfaces to the following elements:

- Applications:
  - QoS-non-aware legacy applications.
  - QoS-aware applications:
    - RSVP applications.
    - DiffServ Applications.
    - EAT-based Applications.
- ACA: Admission Control Agent.
- Persistence Layer.
- Service package (management of network services, WP 2.1).

4.4.4 EAT in context

The following UML class diagram represent the EAT, and its relations to the other system elements, and to the different participants of the process.

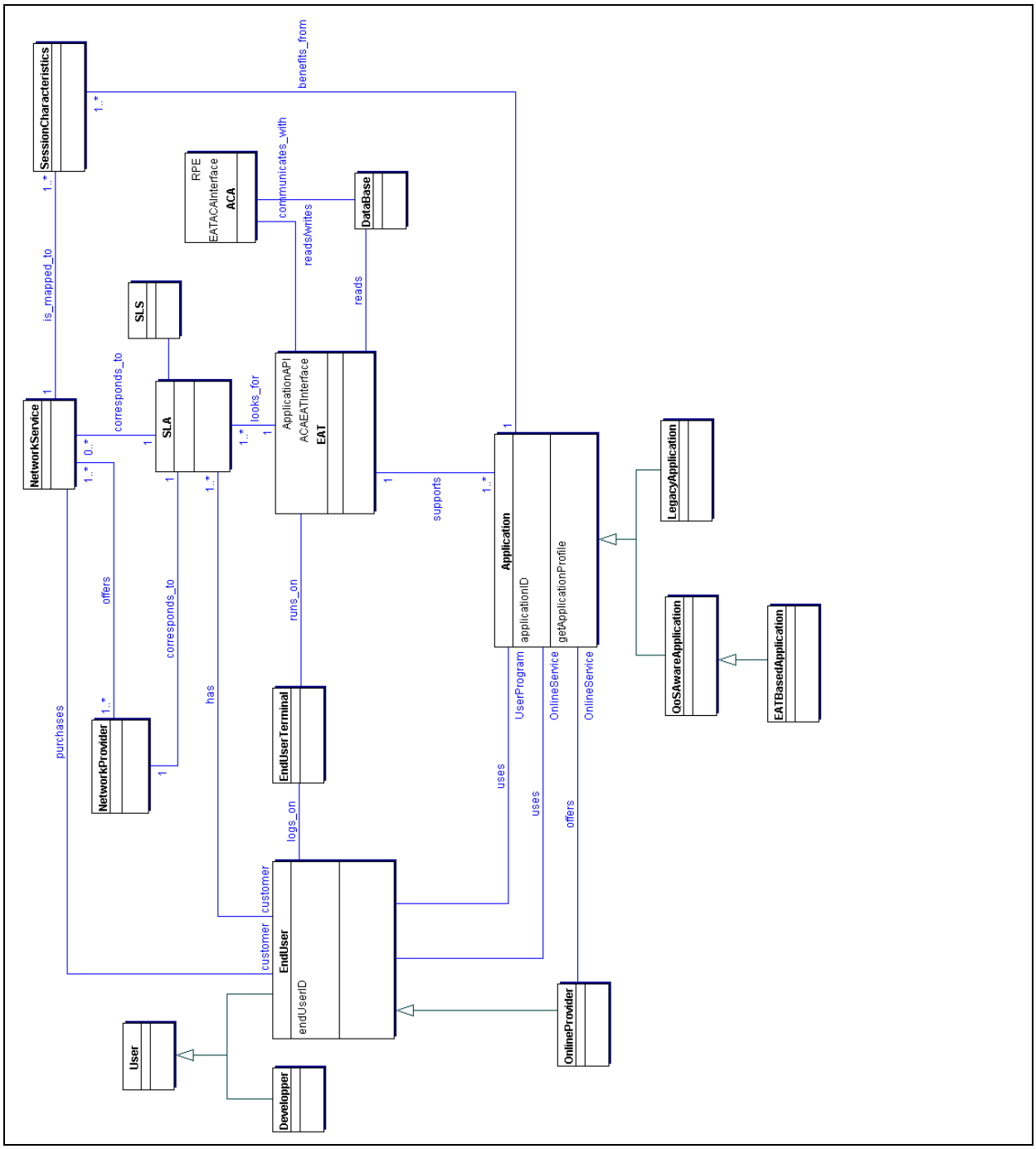


Figure 4-6: Class diagram for the EAT structure

4.4.5 EAT architecture draft

The following two diagrams (block diagram and class diagram) show the first elements composing the EAT.

At this stage of the work, we are able to identify some main components for the EAT. This architecture is a first draft.

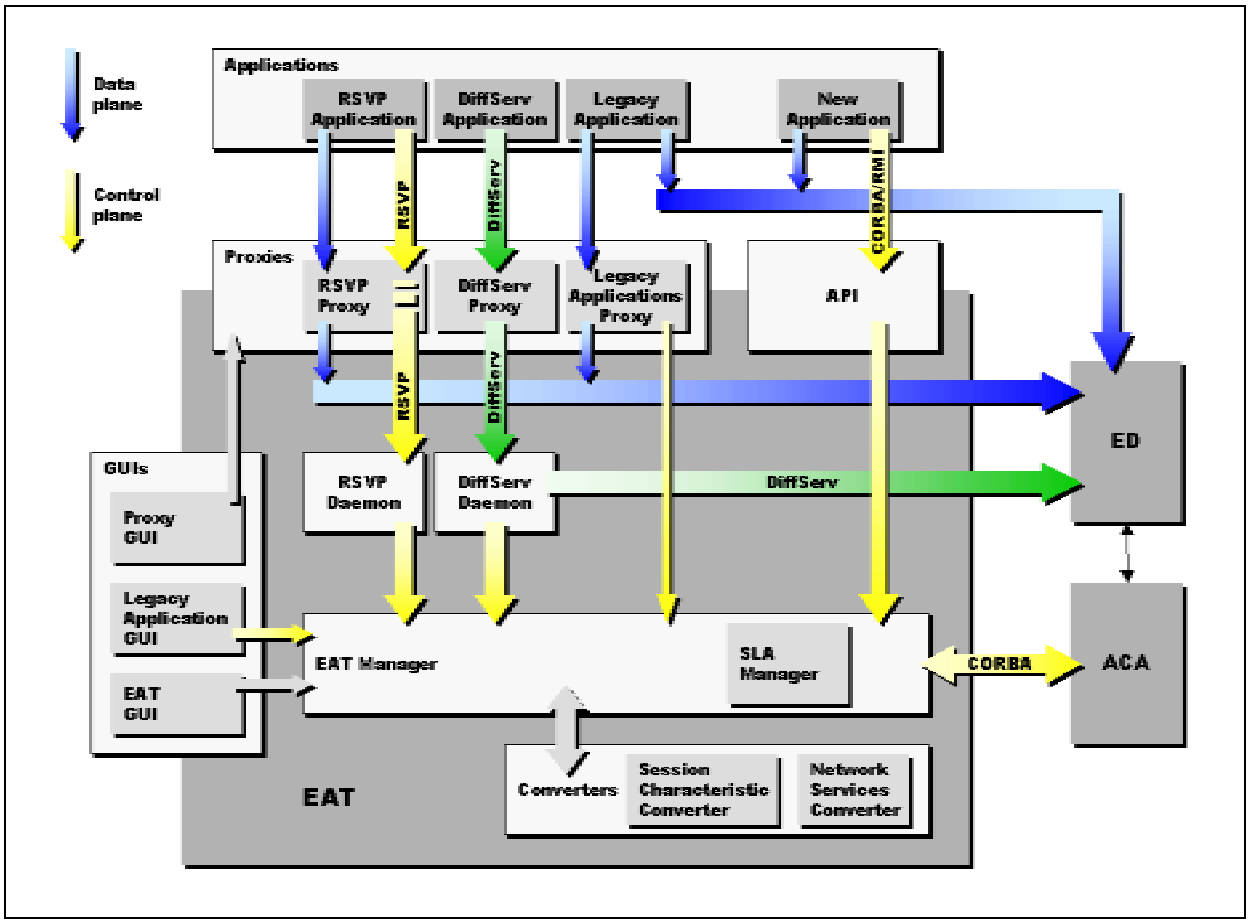


Figure 4-7: Draft block diagram of the EAT

The main components are:

- Three GUI components: one for the EAT in general (EAT GUI), one for the proxy functionality (Proxy GUI), and one for the legacy application (Legacy Application GUI).

  By the way of the GUI, the end user can for example choose his session characteristics, and in the case of legacy applications, the control plane data can be targeted and transmitted to the ACA.

- The EAT Manager: is the main part of the EAT and controls the whole process.

- The Proxies are for applications which are not based on the EAT's API. They enable the selective processing of the control plane information by forwarding some data to the EAT Manager or daemons.
  - RSVP Proxy: extract the control plane information to be processed by the RSVP Daemon.

- DiffServ Proxy.
- Legacy Applications Proxy: produce control plane information like begin-end of a session. The QoS control plane information are transmitted by the Legacy Application GUI.

- The API: enables the development of new applications that will directly use the AQUILA QoS architecture.

- The SLA Manager: look up for the SLA concluded by the end-user in order that the offered session characteristics are consistent with what the end-user paid for. (If the end-user paid for STD network services by his/her network provider, and if STD does not allow to get a video conference with a big picture size, the EAT will not propose such an offer to the end-user! The EAT will only propose the accessible session characteristics.)

- Two converter components: The main task of the EAT corresponds to the mapping of the (by the end-user concluded) network services offered by the ACA, into session characteristics (corresponding to the application in use).
- Network Services Converter.
- Session Characteristics Converter.

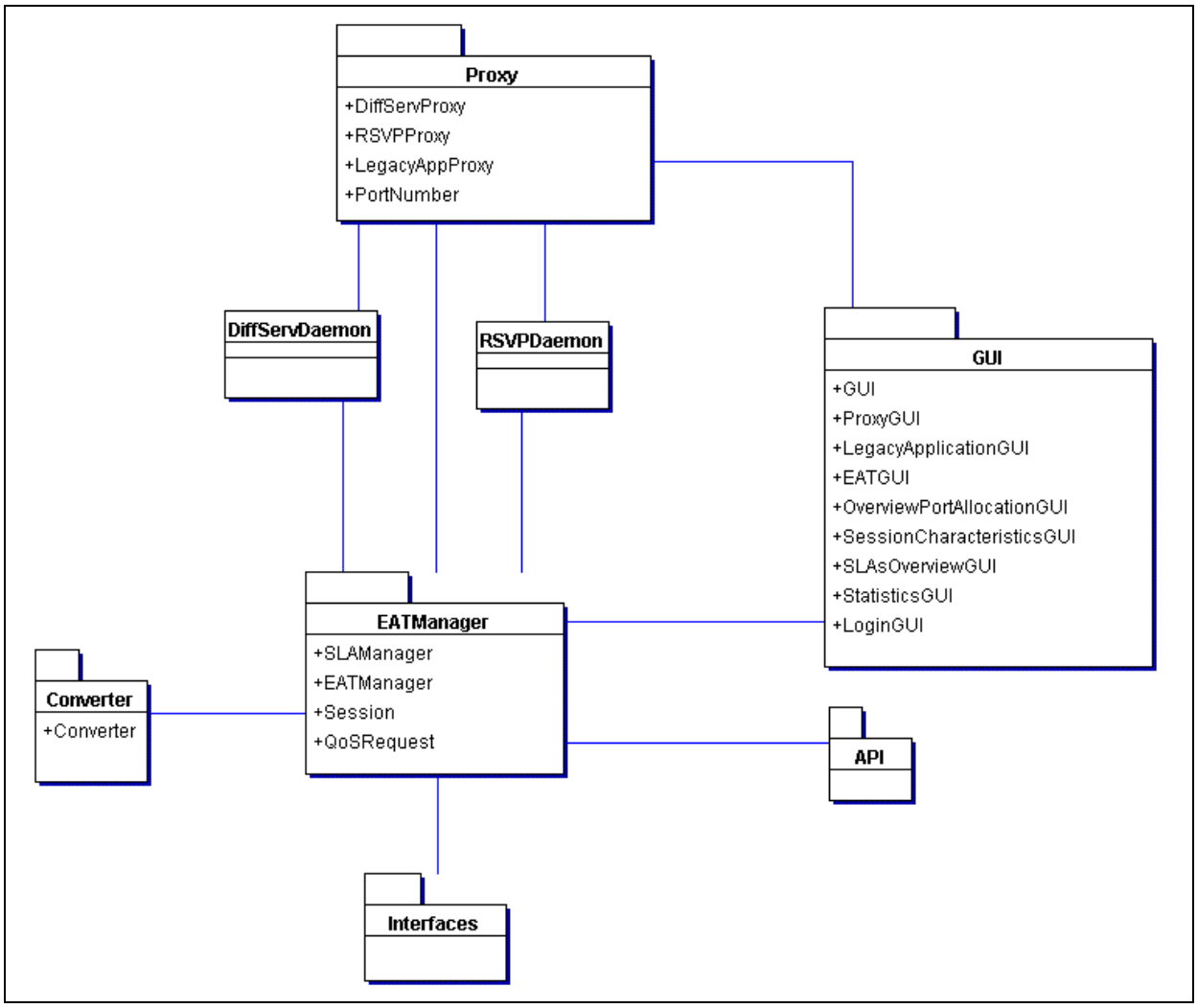


Figure 4-8: Draft class diagram of the EAT

4.4.6 Application profiles

The mapping of network services (the description of the reservation request interface towards the ACA) into session characteristics (the translation of the technical meaning of the network services into a set of terms highly expressive for the end-user, and corresponding to possible quality sets, they appear in the graphical user interface towards the end-users) is the result of many mappings depending on: the SLAs concluded by the end-users, the network services, the technical requirements of the applications, the application types, the applications.

An application is determined by its type (video conferencing, video streaming, etc.), and its technical realisation (codecs, for example). Therefore two kinds of *generic* profiles are needed:

Session characteristics profile = set (session characteristics qualifier)

Technical profile = set (technical parameters)

Furthermore, it could be useful to determine for each concrete application an application profile. An application profile, for a *concrete* application (for example for NetMeeting, version x.yz) would be the result of the combination of the qualifiers which correspond to the session characteristics profile of this application, and the parameters which correspond to the technical profile of this application.

Application profile = set (session characteristics qualifier) + set (technical parameters)

These parameters are the outcome of the generic profiles.

4.4.6.1 First sets of parameters: Generic parameters

These are:

- **Session characteristics profile:** Corresponds to a verbal description of the application. Is a set of terms, highly expressive for the end-user, representing the session characteristics features like: picture size, sound quality, etc. of an application type like: video streaming, video conferencing, etc.

  The session characteristics profiles must be created according to knowledge of respective experts (e.g. Bertelsmann AG, Q-Systems) in interdisciplinary domains: man-computer interaction and technical domains.

- **Technical profile:** Corresponds to the technical description of an application. Is a set of technical parameters describing various services like MPEG4 video, or G.711 audio etc., coding schemes or any other application-related technical requirements.

The parameters could be stored in a database accessible for the EAT. In order to support future applications, the sets of technical parameters and the database should be kept up to date. The new technical parameters (new codecs, for example) must be inserted in the database, and readily available for the EAT.

4.4.6.2 Second set of parameters: Application-specific parameters

To define an application a third set of parameters is needed and corresponds to a concrete application (e.g. Real player version x.yz) and combines one set of technical parameters and one set of session characteristics qualifier. This is an **application profile.**

These application profiles may also be developed by an experienced user (which brings to mind the JAIN initiative), or by the network provider (perhaps through the QMTool). They may be stored at distributed databases of the network provider and downloaded and used by the EAT's Session Characteristics Converter.

The EAT therefore could maintain a profile library that could be updated when a new application is installed on the system. It could also keep the profiles of its users containing their personalised settings with regard to a particular application.

Evidently, the large number of existing (and forthcoming) applications, as well as the multiplicity of end-user's notion of quality, may be prohibitive for the definition of a list with QoS characteristics for each application.

4.4.7 Utilisation of profiles

For legacy applications, the EAT contacts the database and – by means of an application identifier – finds the session characteristics profile corresponding to the kind of application required by the end-user (e.g. video). Then the EAT looks up in the database for the technical set of parameters corresponding to the concrete application (e.g. RealPlayer). After the mapping (between SLAs, network services, session characteristics and application profiles) the EAT will offer the various choices, via an intuitive GUI to the end-user, who will be able to perform a selection according to personal views as well as charging information.

In the case of EAT-based applications, they will be able to query for the profile and pass the profile descriptor in the request message to the ACA. In this case the EAT API could provide a kind of "profile mechanism", and when an end-user starts an API-based application, the application, by the way of this "profile mechanism" informs the EAT of its profile (using XML for example. XML can be used for the exchange of structured data, what we actually have).

5 Interfaces

This chapter describes all interfaces between the parts which make up the AQUILA architecture. Interface descriptions are either by IDL or by JAVA, depending of whether we use CORBA or RMI.

5.1 Internal Interfaces

5.1.1 Reservation Request

In the following, the interfaces between the EAT and the ACA as well as between the EAT and the service component are specified which are focused on reservation requests. The chapter is split into three sections:

In the first section, the EAT acts as the client whereas the ACA acts as the server. This interface mainly concerns reservation requests.

In the second section, the EAT acts as the client, too, but the server is the service component. This interface mainly concerns the network services provided for the reservations.

In the third section, the ACA acts as the client whereas the EAT acts as the server. This interface is mainly to allow notification of events.

ACA towards EAT

The ACA implements three interfaces for the EAT which have to do with the reservation request. These are:

- The SessionManager interface, which provides a login operation for a requester and acts as the factory[2] for an UserAgent. This interface will be implemented by a singleton[3] server object.
- The UserAgent interface, which represents the product of the SessionManager, provides a requestReservation operation, and acts as the factory for an UserReservation. Additionally, a logout operation for the requester is provided.
- The UserReservation interface, which represents the product of the UserAgent, and provides a getAccountingData operation as well as a release operation for the reservation. (Figure 5-1)

[2] **Factory.** An object that manufactures objects on demand.

[3] **Singleton.** An object that is the only instance of its class.

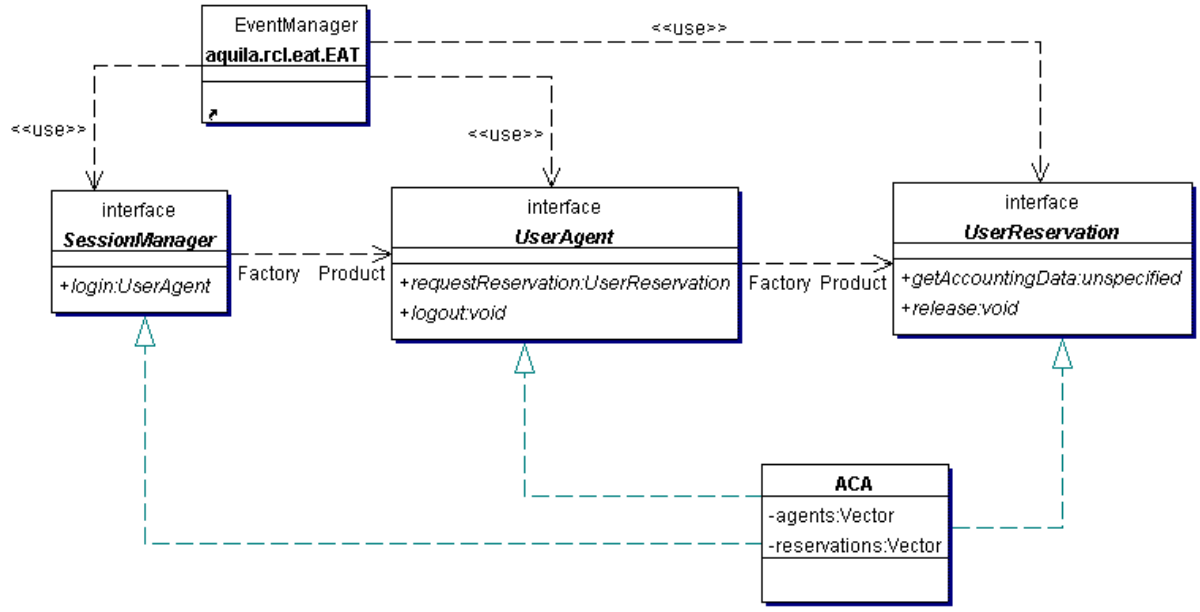


Figure 5-1: Interfaces ACA towards EAT

There are some more classes related to the reservation, for example TrafficModel, TrafficDescription, and ReservationTime. They are described in more detail (i.e. by an IDL specification) in the deliverable of workpackage 2.1.

Service component towards EAT

The service component/package provides one interface for the EAT which is the singleton ServiceManager. This interface provides operations on the NetworkService class which consists of several classes as shown in Figure 5-2.

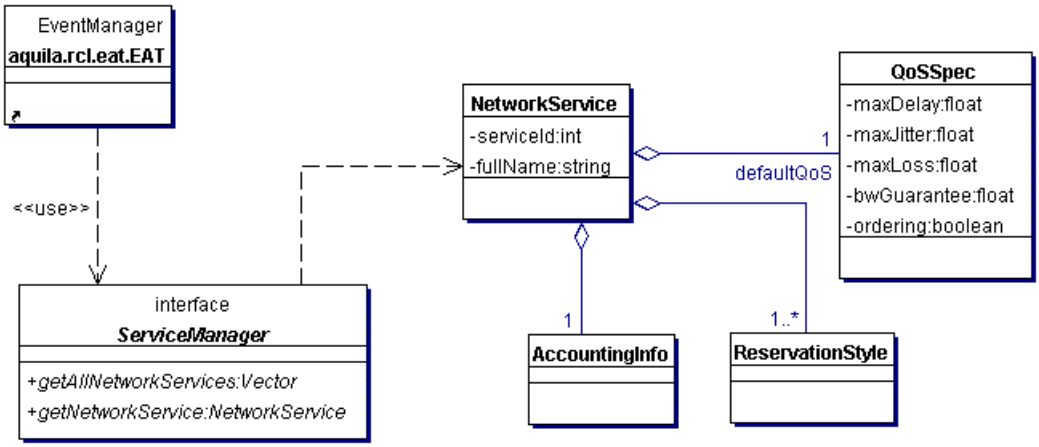


Figure 5-2: Interface Service towards EAT

EAT towards ACA

The EAT implements one interface for the ACA which is the EventManager interface. It provides a notify operation in order to allow the notification of the EAT by the ACA. This is for events e.g., when a new network service occurs.

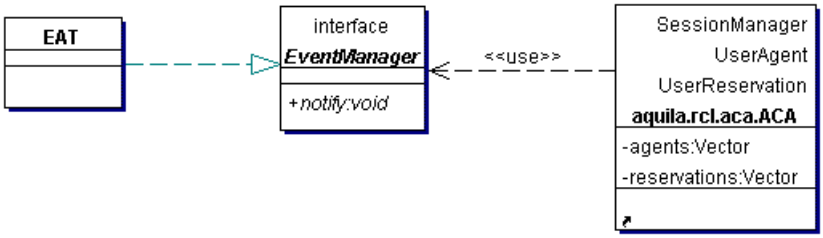


Figure 5-3: Interface EAT towards ACA

5.1.2 Interface between different instances of the ACA

(Note: Operations are described in this subchapter using Java syntax. This is to be regarded to be pseudo code, not the final operation description on implementation level.)

The interfaces used in the communication between different ACA instances, namely between manager ACA and sender ACA and between manager ACA and receiver ACA are covered by the ACA classes LocalAC and CommunicationSession.

The LocalAC class basically provides the following methods:

```
Resulttype reserve(CommunicationSession, boolean isIngress);
void release(CommunicationSession);
```

For ingress reservation, a UserAgent (running in some ACA, the manager ACA of the EAT that has made a reservation request) calls

```
reserve(cs, true);
```

at the LocalAC of the sender ACA (ingress LocalAC). The passed CommunicationSession bundles all the information needed for the ingress LocalAC to make a reservation and establish classifier/marker/policer state in its associated ED: TrafficClass, TrafficDescription, ReservationTime, and ClassifierInfo. The ingress LocalAC also uses the CommunicationSession object to store a reference to the IngressFlow object representing the established classifier/marker/policer state.

The reserve() method returns some result value indicating the result of the call.

Analogously, for egress reservation, the UserAgent calls

```
reserve(cs, false);
```

at the LocalAC of the sender ACA (ingress LocalAC).

The UserAgent uses the release method at both the ingress and egress LocalAC to cancel an existing reservation. Again, the passed CommunicationSession contains the information needed by the LocalAC to perform the release.

Furthermore, the class CommunicationSession provides a method

```
void addUsageData(float[]);
```

which the ingress LocalAC can use to pass usage data for the flow. (The LocalAC retrieves this information via the EDAdapter from the ED.) This method may be called in regular intervals and/or when releasing a reservation.

5.1.3 Interface ACA-RCA

The split of the RCL into ACA and RCA was motivated by having a clear separation between resource distribution mechanisms within a hierarchy of resource pools on the one hand and admission control for user requests on the other hand. This is reached by the approach described in the following.

Each ACA has a reference to some entity within the RCA that is responsible for assigning resources to this ACA. (see chapter 4.3). To allow an ACA to obtain its initial resources, this entity provides a method

```
ConsumableResourceShare[] initConsumableResourceShares();
```

ConsumableResourceShare is an interface defining the ACA's view of a resource assigned to it. Thus, an ACA's resources are made up from objects of a class implementing the interface ConsumableResourceShare. (These „resource objects" belong to the RCA.) Each ConsumableResourceShare is associated with a TrafficClass, and specifies, whether this is an ingress or an egress resource. Furthermore, it provides two methods:

```
Allocation alloc(TrafficDescription, ReservationTime);
void dealloc(Allocation);
```

where Allocation is a type for something like an „allocation handle", used when there is need to refer to an individual allocation (in particular: when de-allocating it). TrafficDescription and ReservationTime contain information about how much of the resource is needed at which time interval.

The following picture shows the classes relevant at the interface. The names of the RCA classes are yet to be finalised ("ResourceShare" and "ResourcePool" may still be changed, compare chapter 4.3).

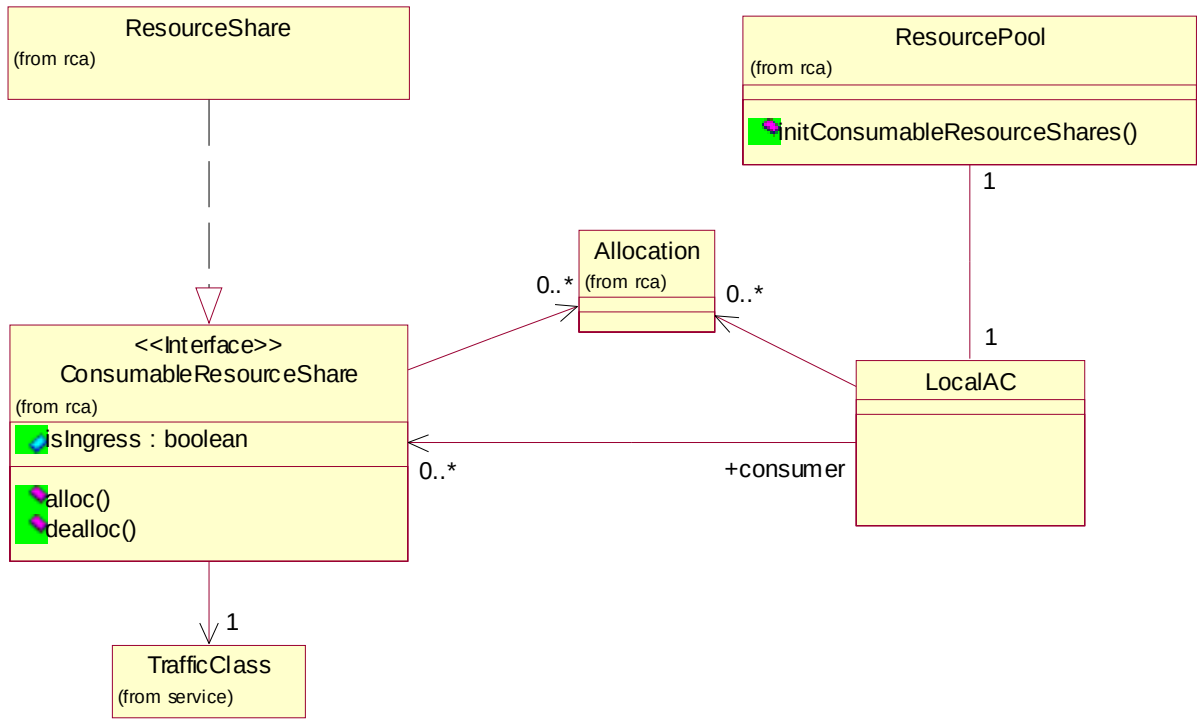


Figure 5-4: Interface ACA-RCA

Given this interface, the ACA only uses allocate and de-allocate. All the mechanisms for the dynamic modification of the resources assigned to an ACA, i.e. the resource distribution mechanisms, are handled by the class implementing the interface ConsumableResourceShare as well as by other classes belonging to the RCA. E.g., if there is some „high water mark“ mechanism, then on each call of alloc() the called object of the class implementing the interface would check whether the high water mark has been reached and would possibly try to gain additional bandwidth etc.

5.2 External Interfaces

5.2.1 Static configuration of edge devices

The configuration and management of the network devices is performed using one of three methods: CLI, SNMP or Web browser. The configuration of the most functional blocks of the ED is static during the runtime and is done during the start-up. The configuration of these blocks is not included in the RCL.

Most of the network administrator use the CLI. The CLI is manufacturer dependent. The network device is connect via a telnet connection and the CLI-commands are dealt online.

The Simple Network Management Protocol (SNMP) is an application-layer protocol designed to facilitate the exchange of management information between network devices. By using SNMP-transport data (such as packets per second and network error rates), network administrators can more easily manage network performance. Cisco routers have presently only proprietary MIBs for Differentiated Services. Objects of the MIBs are either for gathering statistics information or for configuration. For most of configuration objects the maximal level of access is defined as read-only.

The congestion management is one of these functionality's. In the Aquila project WFQ and PQ will be used. Following is an configuration example for WFQ:

```
Router(config-if)# fair-queue [congestive-discard-threshold [dynamic-queues
[reservable-queues]]]
```

This CLI Command configures an interface to use fair queuing

```
Router# show interfaces [interface] fair-queue
```

Displays information about an interface configured for WFQ and DWFQ.

```
Router# show queue interface-type interface-number
```

Displays the contents of packets inside a queue for a particular interface or VC.

```
Router# show queueing fair
```

Displays status of the fair queuing configuration.

Example:

```
Router(config)# interface Serial 3/0
Router(config-if)# ip unnumbered Ethernet 0/0
Router(config-if)# fair-queue 64 512 18
```

With DWFQ, packets are classified by flow. Packets with the same source IP address, destination IP address, source TCP or UDP port, destination TCP or UDP port, and protocol belong to the same flow.

DWFQ allocates an equal share of the bandwidth to each flow.

The following example enables DWFQ on the HSSI 0/0/0 interface: description 45Mbps to R2

```
Router(config)# interface Hssi0/0/0
Router(config-if)# description 45Mbps to R2
Router(config-if)# ip address 200.200.14.250 255.255.255.252
Router(config-if)# fair-queue
```

The following example configures QoS-group-based DWFQ. CAR policies are used to assign packets with an IP precedence of 2 to QoS group 2, and packets with IP precedence 6 are assigned to QoS group 6.

```
Router(config)# interface Hssi0/0/0
Router(config-if)# ip address 188.1.3.70 255.255.255.0
Router(config-if)# rate-limit output access-group rate-limit 6 155000000 2000000
8000000 conform-action set-qos-transmit 6 exceed-action drop
Router(config-if)# rate-limit output access-group rate-limit 2 155000000 2000000
8000000 conform-action set-qos-transmit 2 exceed-action drop
Router(config-if)# fair-queue qos-group
```

```
Router(config-if)# fair-queue qos-group 2 weight 10
Router(config-if)# fair-queue qos-group 2 limit 27
Router(config-if)# fair-queue qos-group 6 weight 30
Router(config-if)# fair-queue qos-group 6 limit 27
!
Router(config)# access-list rate-limit 2 2
Router(config)# access-list rate-limit 6 6
```

5.2.2 Dynamic control of edge devices

The EAT, ACA and RCA build the control plane of the Aquila network. The edge devices are combined to the control plane by the ACA. For each ED exists one ACA. The dynamic configuration of the classifier, marker and policer of the ED is included in the ACA and called EDAdapter. After the ACA has accepted the request for an ingress flow, the EDAdapter maps the information for the classifier, marker and policer provided by the ACA into CLI-, SNMP- or COPS-commands.

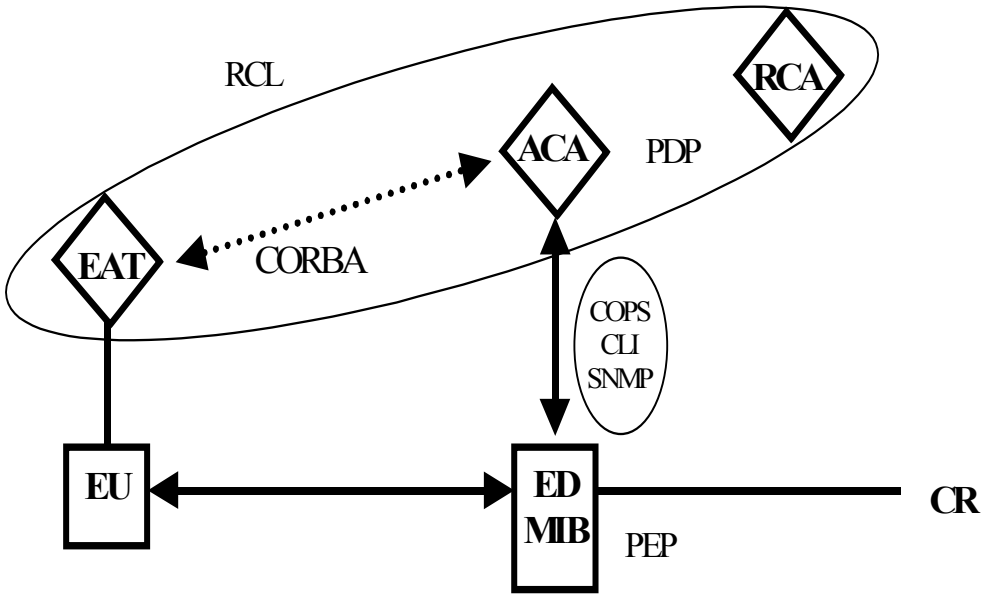


Figure 5-5: Router interfaces

The class EDAdapter basically provides the following methods (Java syntax):

```
IngressFlow establishIngressFlow(ClassifierInfo ci,
                                 PolicerInfo pi,
                                 MarkerInfo mi) {
```

EstablishIngressFlow is an the interface between the ACA and the ED. Class IngressFlow is instantiated once for each established ingress flow. The classes ClassifierInfo, PolicerInfo, MarkerInfo contains the information for the configuration of the ED, i.e. represents the classifier, policer and marker settings for a single reservation. The method builds the interface between ED and ACA and delivers a reference to IngressFlow object.

```
Float [] getUsageInfo(flowHandelID);
```

This method gives access to different MIBs. The ACA retrieves statistic information from the ED. The ACA needs this feedback information from the ED to avoid congestion by changing the marking of the packets.

To release the IngressFlow at the end of the session, the method

```
releaseIngressFlow(flowHandelID);
```

frees the allocated resources of a flow.

For the configuration of the classifier, marker and policer only CLI-command can be used. Most of the MIB objects are read-only defined for SNMP. The CLI of the Redstone Unisphere ERX is modelled after the Cisco CLI. Therefor the EDAdapter should based on CLI commands.

COPS is probably an option for the future but until now it is not supported for DiffServ provisioning.

5.2.3 Management of the RCL

In the first trial, there are no means for a sophisticated, dynamic management of the RCL. Instead, during operation, the RCL mainly uses pre-configured “management information”, i.e. information about the network topology, about traffic classes, about subscribers etc. This information is stored within a database. An administrator can use standard tools provided with the database implementation (to be selected) to enter the information into the database before the RCL is started. Part of the database contents may also be updated by an administrator during operation of the RCL, e.g. a new user may be entered.

The RCL uses the database also to store information that it gathers itself, e.g. service usage data for individual subscribers. Such information can be retrieved by an administrator from the database using standard tools provided with the database implementation.

6 Abbreviations

A

ACA	Admission Control Agent
AF	Assured Forwarding

B

BA	Behaviour Aggregate
BB	Bandwidth Broker
BE	Best Effort
BR	Border Router

C

CBR	Constant Bit Rate
CLI	Command Line Interface
COPS	Common Open Policy Service
CORBA	Common Object Request Broker Architecture
CPE	Customer Premises Equipment
CR	Core Router

D

DiffServ, DS	Differentiated Services
DSCP	DiffServ Code Point

E

EAT	End-user Application Toolkit
ED	Edge Device
EF	Expedited Forwarding
EGP	Exterior Gateway Protocol
ER	Edge Router

I

IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
IntServ	Integrated Services
ISDN	Integrated Services Digital Network
ISP	Internet Service Provider

L

LAN	Local Area Network

M

MF	Multi Field
MPLS	Multi Protocol Label Switching

O

OSPF	Open Shortest Path First

P

PDP	Policy Decision Point
PEP	Policy Enforcement Point
PHB	Per Hop Behaviour
PMC	Premium Mission Critical
PMM	Premium Multimedia
PoS	Packet over SONET/SDH
PVO	Premium Voice

Q

QMTool	QoS Management Tool
QoS	Quality of Service

R

RCA	Resource Control Agent
RCL	Resource Control Layer

RIP	Routing Information Protocol
RSVP	Resource Reservation Protocol

S

SNMP	Simple Network Management Protocol
SLA	Service Level Agreement
STD	Standard

T

TCB	Traffic Conditioning Block
TOS	Type Of Service

V

VBR	Variable Bit Rate
VPN	Virtual Private Network

W

WAN	Wide Area Network
WFQ	Weighted Fair Queuing

7 References

[Black99] Black, Darryl P.: Building Switched Networks – Multilayer Switching, QoS, IP Multicast, Network Policy, and Service Level Agreements, Addison Wesley, 1999

[Cisco99] Cisco IOS™ Software: Quality of Service Solutions, Whitepaper, http://www.cisco.com/warp/public/cc/cisco/mkt/ios/tech/tch/qosio_wp.htm

[Cisco00/1] Quality of Service Overview, http://www.cisco.com/univercd/cc/td/doc/product/software/ios121/121cgcr/qos_c/qcdintro.htm

[Cisco00/2] Cisco QoS Policy Manager, http://www.cisco.com/warp/public/cc/cisco/mkt/enm/cap/qospm/prodlit/qospm_ds.htm

[Cisco00/3] Classification Overview, http://www.cisco.com/univercd/cc/td/doc/product/software/ios121/121cgcr/qos_c/qcprt1/index.htm

[Cisco00/4] COPS for RSVP, http://www.cisco.com/univercd/cc/td/doc/product/software/ios121/121newft/121t/121t1/copsrsvp.htm

[CORBA] CORBA/IIOP 2.3.1 Specification, http://www.omg.org/corba/cichpter.html

[D1101] IST-1999-10077-WP1.1-DTA-1101-PU-R/b0, AQUILA Analysis and Requirements Report

[D1301] IST-1999-10077-WP1.3-COR-1301-PU-S, Specification of traffic handling for the first trial

[Ferg98] Ferguson, Paul; Huston, Geoff: Quality of Service – Delivering QoS on the Internet and in Cooperate Networks, Wiley Computer Publications, 1998

[GHJV95] E. Gamma, R. Helm, R. Johnson, J. Vlissidis: *Design Patterns – Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[RFC2475] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss: An Architecture for Differentiated Services, RFC 2475, December 1998, http://www.ietf.org/rfc/rfc2475.txt

[RFC2597] Heinanen, J.; et al.: Assured Forwarding PHB Group, RFC2597, June 1999, http://www.ietf.org/rfc/rfc2597.txt

[RFC2598] Jacobson, V.; et al.: An Expedited Forwarding PHB, RFC 2598, June 1999, http://www.ietf.org/rfc/rfc2598.txt

[RFC2638] Nichols, K.; et al.: A Two-bit Differentiated Services Architecture for the Internet, RFC2638, July 1999, http://www.ietf.org/rfc/rfc2638.txt

[RFC2748] The COPS (Common Open Policy Service) Protocol. J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, A. Sastry. January 2000. (Format: TXT=90906 bytes) (Status: PROPOSED STANDARD)

[RFC2749] COPS usage for RSVP. J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, A. Sastry. January 2000. (Format: TXT=33477 bytes) (Status: PROPOSED STANDARD)

[Star00] Stardust.com: A Quality of Service glossary of terms, http://stardust.com/qos/whitepapers/glossary.htm

[UML1.3] OMG Unified Modelling Language specification, version 1.3, June 1999. http://www.omg.org/cgi-bin/doc?ad/99-06-08

[Verma99] Verma Dinesh;– Supporting Service Level Agreements on IP Networks, Macmillan Technical Publishing, 1999