On Enabling Dynamically Adaptable Internet Applications

by

Saleem Noel Bhatti

A dissertation submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy
of the
University of London

Department of Computer Science
University College London
University of London

May 1998

Abstract

Network applications operating over a packet-switched network, such as the Internet, receive varying **quality of service (QoS)**. Fluctuations in QoS can be due to a combination of effects:

E1. variations in network behaviour due to network traffic from other sources
E2. variations in network paths due to the behaviour of routing functions
E3. the application resides on a mobile host

Fluctuations in QoS are observable over the lifetime of a single instance of an application, as well as between different instances of an application. Such fluctuations in QoS can lead to difficulties in operation for certain applications, particularly those with real-time media flows such as voice and video. We would like to offer these applications mechanisms that enable them to operate in environments where QoS can vary.

Research in progress within the Internet community considers schemes to provide resource reservation mechanisms. Resource reservation allows the application to make QoS requests to the network in terms of constraints defined by values of certain QoS parameters. However, it is unlikely that resource reservation will be ubiquitous in the short to medium term (next 2-5 years). Internet resource reservation using RSVP (Resource reSerVation Protocol) may not support sufficient receiver heterogeneity and is not robust to IP level outages. Additionally, there may be scenarios in which resource reservation may not be possible and/or practicable, e.g. mobile/wireless environments.

We argue that the network applications of the future will have to become **dynamically adaptable** in response to fluctuations in network QoS. Even where excellent end-to-end resource reservation capability is available to counter the effects of E1 and E2 above, applications will still need to be adaptable to cater for the effects of E3. In order to allow dynamic adaptability, there is a need for a general, flexible and practicable mechanism that allows an Internet application to make assessments of available QoS. In conjunction with user preferences and other application-specific information, QoS assessments should allow the application to make decisions about how it should adapt its flows in order to

match the available network QoS. We make two contributions to address dynamic adaptability for Internet applications: the **QoSSpace** and the **QoSEngine**.

The **QoSSpace** models a multi-dimensional space, with each of its dimensions defined by a **QoSParam**. A **QoSParam** is a value derived from a real measured QoS parameter value, such as data rate or jitter. An application's flow capabilities (flow-requirements) are used to specify regions of operation within QoSSpace, called **QoSRegions**. The network QoS is also expressed in terms of the same QoSParams and mapped into the QoSSpace. The QoSSpace forms part of the **QoSEngine**. The QoSEngine generates **QoSReports**. QoSReports contain QoS information summaries that are an indicator of the relative compatibility between an application's flow-requirements and the network QoS.

The QoSSpace can be seen as the QoSEngine "front-end". The "back-end" to the QoSEngine is a QoS information processing function that evaluates the network QoS in terms of QoSParams. The QoSEngine "back-end" maps the network QoS measurements into the QoSSpace. The QoSEngine then compares this mapping with the regions defined by QoSRegions to produce QoSReports. These consist of **region compatibility values (RCVs)** for each of the QoSRegions. A RCV for a QoSRegion is a measure of the compatibility between the network QoS and the QoS required to support that QoSRegion. The application uses the RCVs to help it make decisions about how to adapt its behaviour, i.e. change its flow-requirement.

The QoSEngine and QoSSpace are defined to be independent of any particular network technology. Additionally, we show that the QoSSpace and QoSEngine have the potential to be generally applicable to adaptable applications, and parameters other than network data (such as battery life, cost, host load, etc.) could also be used to derive QoSParams. We demonstrate the function and applicability of the QoSSpace and QoSEngine by the simulation of a dynamically adaptable audio tool.

Acknowledgements

Graham Knight has played the role of mentor and Devil's Advocate. During the time over which I have completed my work, he has always been extremely patient and reigned in my wilder thoughts with solid, common sense, as well as encouraging me to pursue my ideas. He has also been kind enough to give me the resources and time to do this work when I probably should have been doing something else. Graham has always been ready to give up his own time in order to discuss my work with me. I owe him a debt of gratitude that will be hard to repay.

Jon Crowcroft's oracular knowledge has kept me on the straight and narrow, while his holistic inputs have kept me on my toes! Whenever I have marched into his room for a "quick chat" he has always made time to talk to me, and I have yet to walk out of such meetings without having learnt something.

I would like to thank Graham Knight, Steve Wilbur, Stephen Hailes and Jon Crowcroft for their extremely helpful feedback on this work. I would also like to thank the anonymous reviewers who have made useful comments on the parts of this work that have found their way into papers.

I am grateful to Mark Handley at MIT (USA), Franco Sirovich at Systems Wizards s.p.a. (Italy), Kate Lance at connect.com.au Pty. Ltd (Australia), and Dieter Gantenbein at IBM Labs Zurich (Switzerland) for allowing let me use hosts at their sites in order to take network measurements.

There are many people within the Department, past and present, who have been kind enough to share their thoughts with me, and have provided me with valuable experience by giving me the chance to work with them.

My thanks to Prof. David Hutchison (Lancaster) and Prof. Chris Mitchell (Royal Holloway) for examining this work.

My parents, Daisy and Peter, my sisters, Pauline and Neena, and my BroBroBro, Andrew, have always encouraged me ever onwards. I thank them for their love and support.

Finally, for looking after me when I was holed-up in our box-room with a stack of papers and a couple of workstations; for putting up with many lost evenings and weekends over the past years; for her love, kindness and support; for making it all worthwhile, my thanks and love to my wife, Catriona.

Notes

References to IETF work-in-progress: IETF Working Groups produce Internet-Draft (I-D) documents that have no official status and should not be quoted other than as "work-in-progress". Abstracts of the current I-D documents are available in the file

```
1id-abstracts.txt
```

at the FTP locations shown below:

Africa:	`ftp.is.co.za/internet-drafts`
Europe:	`ftp.nordu.net/internet-drafts`
	`ftp.nis.garr.it/internet-drafts`
Pacific Rim:	`munnari.oz.au/internet-drafts`
US East Coast:	`ftp.ietf.org/internet-drafts`
US West Coast:	`ftp.isi.edu/internet-drafts`

Information about the IETF WGs can be found at http://www.ietf.org/

URLs: all URLs were checked to be correct at the time of writing but their longevity cannot be guaranteed.

Glossary: a collection of selected acronyms and definitions appears before the References.

Table of Contents

List of figures

Page

List of tables

1. Introduction

As communications and computing are integrated, and distributed applications become the norm rather than the exception, there is now a wide range of network technologies and a whole host of applications expected to work across those different network technologies. The Internet protocol (IP) [IPv4, IPv6] is a packet switched network layer protocol that can be supported by just about any network technology. This is useful as it provides a common interface to network connectivity and hides the specifics of the actual underlying network technology. However, what IP cannot hide is that each of the different network technologies has different capabilities: different data rates, different delay characteristics, different error characteristics. We say that each network technology offers a different **quality of service (QoS)**. Additionally, when IP is used for internetworking in wide area communication, there are many different networking technologies making up the underlying end-to-end network. In such situations, the QoS seen by an instance of an application at the communication end-points could vary dramatically during its lifetime, as well as between different instances of that application throughout the network.

1.1 Network aware applications

Many current applications that rely on network communication have one thing in common – they assume that the network resources that they require will be available to them throughout the course of their operation. In particular, these applications assume that the network will maintain a certain QoS guarantee while the application is operating. The QoS guarantee required by the application could be very weak (e.g. file transfer only requires best-effort) or it could be quite strong (e.g. real-time video communication may require guarantees on available data rate, delay bounds and jitter bounds). In today's Internet, it is rare for the QoS seen by an application instance to remain consistent [Pax97a, Pax97b], especially in the wide area, but this can also be true in the local area environment. Additionally, with the advent of mobile computing (portable hardware platforms), the physical connectivity that an application has to the network may be changing frequently. While this varying QoS may not be so important to applications such as file transfer that can make do with a best-effort service, interactive applications and real-time applications with QoS sensitive traffic flows[1] (such as audio and video) may suffer when there are QoS fluctuations. Consider the following two scenarios:

Scenario 1a: an audio conferencing tool using a packet switched network may experience large fluctuations in the delay, throughput, error rates and loss for the transmitted audio data. This will depend on the network load and state of the links both in the backbone network and the transit networks. This is the situation today for the Internet, and is depicted in Figure 1.1.

Scenario 2a: a user with a mobile host uses services located on an office LAN. The user can connect the mobile host using BR-ISDN (Basic Rate ISDN) from home, use a digital mobile phone link whilst travelling to work on the train, and then connect the host to the office LAN (directly) once at work. In each case, the change in connectivity means a change in the QoS. This is depicted in Figure 1.2.

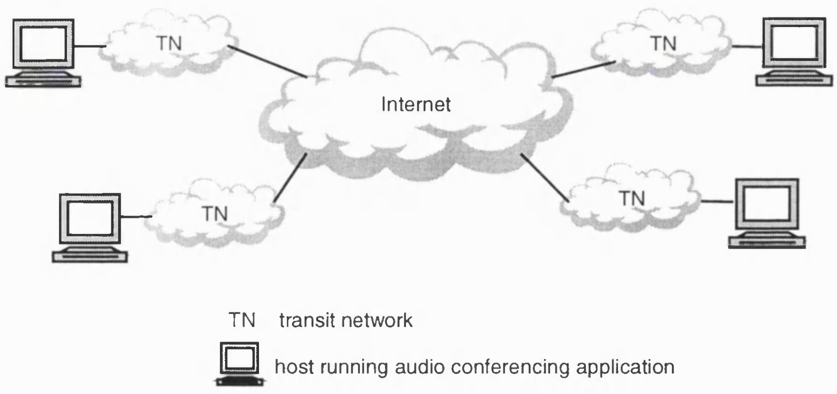


Figure 1.1: An audio conference scenario

[1] A flow is a sequence of packets that are semantically related to each other. Packets belonging to a flow may be identified by information such as their source address, destination address, port numbers or an application specific label, e.g. protocol indentifier.

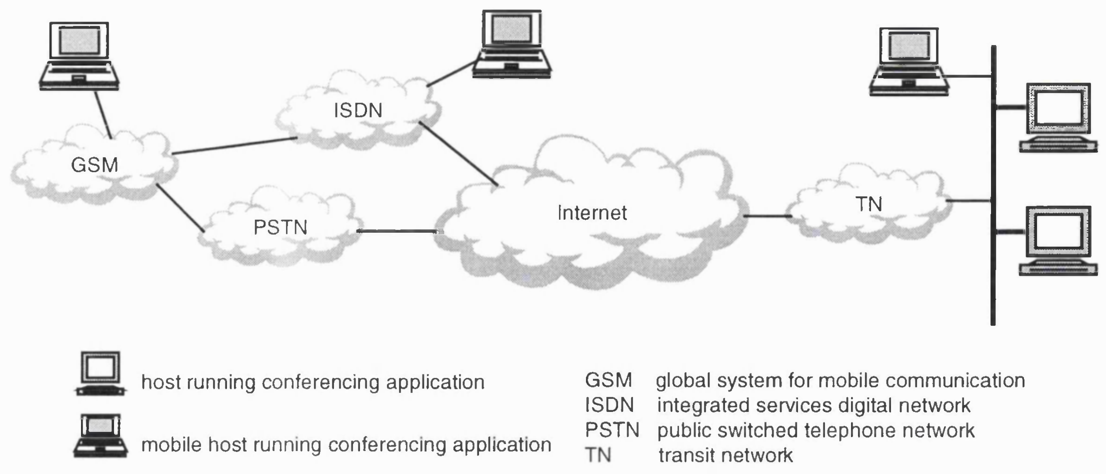

ISDN
GSM
PSTN
Internet
TN
host running conferencing application
mobile host running conferencing application
GSM global system for mobile communication
ISDN integrated services digital network
PSTN public switched telephone network
TN transit network

Figure 1.2: A mobile application scenario

So, an application may experience changes to its physical connectivity as well as its QoS. Any of the following may contribute to the QoS fluctuation:

E1. variations in network behaviour due to network traffic from other sources
E2. variations in network paths due to the behaviour of routing functions
E3. the application resides on a mobile host

In general, the more routing hops and network elements there are on the route between source and destination, the greater the QoS fluctuation that is experienced. Across the Internet, this is due primarily to the effects of E1 [Pax97a] and E2 [Pax97b]. However, the behaviour of the individual network elements will also play a part, e.g. buffering strategies, queuing/servicing mechanisms, routing-table look-ups, etc.

For both of our scenarios, IP hides the *connectivity* changes, and we can generalise that the applications experience *fluctuations in QoS*. We can quantify the QoS variations in terms of (instantaneous) measurements of QoS parameters, such as delay, jitter, available network capacity (data rate), etc. For instance, with our mobile user, the (maximum possible) throughput that the application has may be 128Kb/s on BR-ISDN, 9.6Kb/s on the digital mobile phone link and up to 100Mb/s on 100BaseT. This may drastically change the way in which the host is utilised, as applications may become unusable in certain conditions. It is possible for a user (with the appropriate expertise) to manually configure the host and applications for use in each connectivity scenario. However, not all users have the correct expertise or knowledge about the network, so ideally, we would like applications to be usable whenever possible by allowing them to **dynamically adapt** their operation in order that they can still offer a service to the user.

To appreciate the need for dynamic adaptability, consider again our two scenarios:

Scenario 1b: when the audio tool detects changes in the QoS that is being offered to its audio stream, it can adapt its behaviour to cope. For instance, current audio tools (such as *rat* [HSK98] and *vat* [vat]) adapt the size of their play-out buffers as they detect the delay variation of the received packets. However, if these applications could obtain information about other parameters, they may be able to adapt further. For example, *rat* and *vat* allow the use of several different audio encoding schemes, with differing packet sizes and data rates which, currently, must be configured manually. With appropriate information about throughput, loss and error rates, a particular *rat* or *vat* instance could dynamically adapt itself to pick the best audio coding scheme to use for the particular network conditions experienced at that time.

Scenario 2b: the user is involved in a multimedia conference that has flows for data, voice and video. When the user is at home (using BR-ISDN) only the audio and data flows are enabled (possibly the video is enabled but with a small-sized picture and low refresh-rate). When the user is on the train (using a mobile phone link) only the audio flow is enabled. When the user gets to work (using a LAN) then all three flows are enabled. In this way the user has remained part of the conference, but the conferencing application has adapted its service in order to cope with the change in the user's connectivity via the detected QoS.

Note that the examples in the scenarios consider only the **network QoS** issues, and not distributed architectural issues, such as resolution of receiver heterogeneity. For example, in a wide-area, multiparty audio conference, the many instances of the audio tool will each experience different QoS, and so may all decide to adapt differently, e.g. by each choosing a different audio encoding scheme. Some additional **application-level** mechanisms would be required in order for all instances of the audio tool to synchronise and/or locate proxy applications that are acting as audio transcoding or filtering gateways [KHC98, YGHS96].

So, in the future we would like to have the option of building **dynamically adaptable** applications that are network aware, and can change their **flow-requirements** to get the best from the offered communication conditions.

1.2 Overview

We use different descriptions for our view of the **application**, the **flow** and the **network**. An application may operate in different **modes**. Each of these modes may have one or more **flow-requirements**[2] for each of its flows, associated with that mode. The flow itself may operate at various **set-points**[3] within the same flow-requirement. The QoS experienced by a flow is expressed as the **network QoS** as seen by that flow. For example, an Internet audio application may have several modes in which it can operate, such as "high", "medium" or "low" quality modes. Each mode may uses one or more flow-requirements for the audio flow it generates, e.g. "high" quality may use PCM encoding, "medium" quality may use ADM6 or ADM4 encoding and "low" quality may use GSM or LPC encoding. Each flow-requirement is identified by a different name and has a different flow construction. While the flow is operating in a particular flow-requirement, it may also have a set-point within that flow-requirement. For example, an audio application may adjust the size of its audio play-out buffer to cope with jitter for the flow. This same adjustment may occur in any of the flow-requirements. The distinction between the flow-requirement and the set-point may be see as the distinction between a course-grained, discrete description of the flow construction and a finer-grained, intra-flow-requirement description.

Each mode may support the use of one or more flow-requirements. The choice of application mode will depend on user input (e.g. user preferences), information from the network (e.g. information about which flow-requirements the network can support) and other application-specific information. The application modes are application-specific. Our eventual goal is to allow the application to make assessments about how it should choose between a number of flow-requirements. We are not concerned here with the application modes (other than there may be one or more flow-requirements associated with an application mode) or the set-point (other than that the set-points exist within a flow-requirement).

This work considers how to enable dynamically adaptable Internet applications by providing then with a QoS summary of the compatibility between flows and the network. A QoS processing function has been developed that measures the relative compatibility

[2] A flow-requirement for a flow is a specific flow construction expressed in terms of QoS service requirements.

[3] A set-point for a flow is a well-defined, steady operating point for the flow.

between the network QoS and the application's flow requirements. The compatibility measurements can be used by the applictaion to make changes to its flow construction in order to make use of the available network QoS. The starting point for our approach is the model shown in Figure 1.3. We note three distinct areas of interest, each logically separated by a well-defined interface:

1. the interaction between the user and the application (via interface I_{ua})
2. the interaction between the application and the QoSEngine (via interface I_{aq})
3. the interaction between the QoSEngine and the network flows (via interface I_{qn})

Our work concerns only the part highlighted in the dashed box in Figure 1.3. We model the application as the generator and/or receiver of network **flows** (also called **traffic flows** or **media flows**). In this context, a flow is described as a sequence of packets that form a single unidirectional stream carrying information between a given source and given (unicast or multicast) destination [IPv6]. The granularity of a flow is application-specific e.g. it may be a single, homogeneous media stream such as audio, or the flow may be a logical channel carrying a variety of multiplexed traffic. The application sends flows over the network and may also receive flows. Figure 1.3 shows a single flow but there may be many flows being sent from or received by the application. We say that the application flows exist in **QoSSpace**.

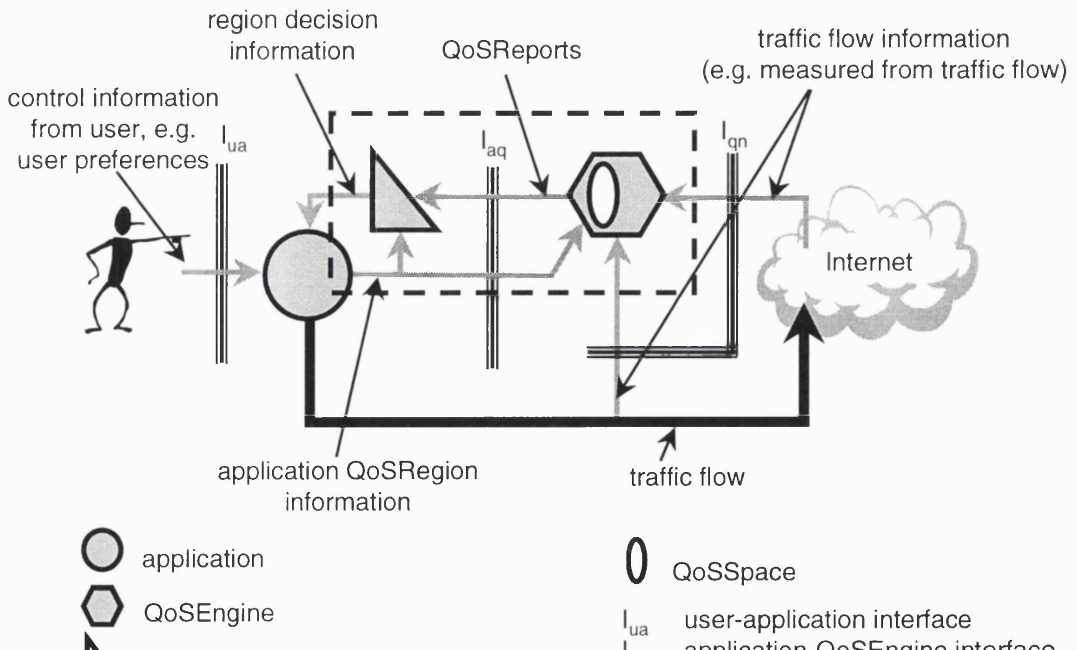

region decision information
QoSReports
traffic flow information (e.g. measured from traffic flow)
control information from user, e.g. user preferences
I_ua
I_aq
I_qn
Internet
application QoSRegion information
traffic flow
application
QoSEngine
application adaptation function AAF
QoSSpace
I_ua user-application interface
I_aq application-QoSEngine interface
I_qn QoSEngine-network interface

Figure 1.3: Interaction between the application and the network

The **QoSSpace** models a multi-dimensional space, with each of its dimensions defined by a **QoSParam**. A **QoSParam** is a value derived from a real measured QoS parameter value, such as data rate or jitter. An application's flow capabilities (flow-requirements) are used to specify regions of operation within QoSSpace, called **QoSRegions**. The network QoS is also expressed in terms of the same QoSParams and mapped into the QoSSpace. The QoSSpace forms part of the **QoSEngine**. The QoSEngine generates **QoSReports**. QoSReports contain QoS information summaries that are an indicator of the relative compatibility between an application's flow-requirements and the network QoS.

The QoSSpace can be seen as the QoSEngine "front-end". The "back-end" to the QoSEngine is a QoS information processing function that evaluates the network QoS in terms of QoSParams. The QoSEngine "back-end" maps the network QoS measurements into the QoSSpace. The QoSEngine then compares this mapping with the regions defined by QoSRegions to produce QoSReports. These consist of **region compatibility values (RCVs)** for each of the QoSRegions. A RCV for a QoSRegion is a measure of the compatibility between the network QoS and the QoS required to support that QoSRegion. The application uses the RCVs to help it make decisions about how to adapt its behaviour, i.e. change its flow-requirement.

The QoSEngine and QoSSpace are defined to be independent of any particular network technology. Additionally, we show that the QoSSpace and QoSEngine have the potential to be generally applicable to adaptable applications, and parameters other than network data (such as battery life, cost, host load, etc.) could also be used to derive QoSParams. We demonstrate the function and applicability of the QoSSpace and QoSEngine by the simulation of a dynamically adaptable audio tool.

The interface I_{ua} is likely to be application-specific and so we do not consider this in our work. Our aim with respect to this interface is to ensure that we do not impose any constraints on its design or operation. We assume that I_{ua} exists in some form. The interface I_{qn} is where the QoSEngine would receive measurements from the network. Again, this interfcae could be application specific and we must not constrain its operation. We also assume some measurement mechanism exists and can provide the raw QoS paramater values for a flow that are required by the QoSEngine.

1.3 Thesis

The QoSSpace and QoSEngine attempt to provide a general and flexible **network QoS model**. The application's interface to the model should be relatively simple and hide the complexities of the network and its traffic flows, as well as the operation of the QoSEngine [CAH96]. Also, the operation of the QoSEngine should not be restricted to any particular network technology or any particular application. The QoSEngine should be able to function in any environment in which IP can function. We offer the following thesis:

It is possible to design a general and practicable network QoS model that will enable Internet applications with QoS sensitive flows to make adaptation assessments dynamically, by monitoring the QoS being offered by the network to those flows.

1.4 Goals and approach

The goals of this work were to define:

G1. the functions of the QoSSpace and the QoSEngine

G2. an interface to allow interaction between the application and the QoSEngine (I_{aq})

Our main goal was G1, which has been investigated using simulation. The work has considered how information about the compatibility between the network QoS and the application flow-requiremnents can be represented. The interaction between the flow and the network has also been modelled. A specific engineering design for the information modelling and the interactions between the user, the applictaion and the network have been investigated. Specific processing functions have been defined that can combine measurements from the network with the application-derived flow requirements to produce QoS summaries indicating the suitability of the network to support particular flow-requirements.

G2 has been defined as an abstract interface, I_{aq}, in sufficient completeness only to allow any necessary testing of G1. The interface allows information to be passed between the application and the QoSEngine without constraining how the QoSEngine itself is engineered. The applictaion passes its flow-requrements to the QoSEngine, and the QoSEngine can pass QoS summaries back to the application across I_{aq}.

Specifically, the research has considered how G1 may be realised on a packet-switched network offering a connectionless, best-effort service, i.e. the Internet. Although some adaptability mechanisms may already exist in specific applications (e.g. elastic buffering to counter jitter in audio play-out), or for specific network technologies, our aim was to provide a mechanism that is general and flexible enough to be used on any network technology that can support IP.

We have developed a model of the network QoS (QoSSpace/QoSEngine) and applications flows (QoSRegions). We have simulated the operation of the QoSEngine using pre-defined input waveforms, as well as network data captured from hosts on the Internet.

Note that it was **not** the purpose of this research to define such functions as resource reservation or control of QoS within the network (e.g. RSVP [RSVP] and ST2+ [ST2+]). It was to provide a **network QoS model** that can assess the QoS being offered by the network to the application's flows, and to allow the generation of QoS summaries that the application can use to make adaptation decisions.

1.5 Outline of dissertation

We examine the context for the work, providing the motivation and the current state of the art in Chapters 2 and 3:

- **Chapter 2** examines the background for this research by considering the evolution in the use of the Internet and Internet applications. In particular, we consider the needs of QoS sensitive applications, such as those applications with real-time constraints and applications that make use of QoS sensitive data flows.
- **Chapter 3** considers related work, including other approaches to solving the network resource-usage problem. We list and discuss a set of requirements for enabling dynamic adaptability. Other adaptation mechanisms currently being researched are discussed.

Chapter 2 and Chapter 3 highlight design constraints or engineering constraints that we have applied to our work by examining existing work in related areas. We then go on to describe our contribution in Chapters 4, 5 and 6:

- **Chapter 4** presents the **QoSSpace**, our abstraction of the network in the **QoSEngine**, and the generation of **QoSReports**, our QoS information summary. We describe how we determine compatibility between the network QoS and the applications flow-requirements. Included here is a discussion of the application's interface to the QoSEngine. The interface to the QoSEngine is an abstract definition. This is in order to keep the model free from the constraints and biases of any particular programming language binding. However, the simple abstract types make it possible to realise a software interface easily.

- **Chapter 5** describes the QoSEngine back-end and the way it processes measured network QoS parameter values. The QoSEngine back-end uses a fuzzy logic processing system to counter the effects of noise in the measured QoS parameter values. A qualitative analysis of dynamics as well as a quantitative performance analysis is presented.
- **Chapter 6** discusses how the QoSEngine can be applied. We use an example audio application and investigate its behaviour in a simple scenario using real network data as well as constructed data. We show how the QoSReports from the QoSEngine can be used to allow the application to make dynamic adaptation decisions based on measured network data and user preferences through a simple **application adaptation function (AAF)**. We also highlight practical issues to be considered in designing an AAF.

We conclude in **Chapter 7** with a summary of the main contributions and limitations of this research, and identify the key aspects to be addressed in the future.

2. The evolution of Internet applications

Today's network users wish to use the Internet and Internet Protocols for a whole range of applications: from E-mail to World Wide Web (WWW) access to IP-telephony [Sch96], to name but a few of the ever increasing range of multimedia applications. All these applications may require different QoS guarantees to be provided by the underlying network. An E-mail application can make do with a best-effort network service. Interactive or real-time voice (and video) applications require (some or all of) delay, jitter, loss and throughput guarantees in order to function. Web access can make do with a best-effort service, but typically requires low delay, and may require high throughput depending on the content being accessed. As well as the application's functional requirements are the preferences and requirements of the people using those applications. Such user preferences may modify the operation of the application – some users may want "high" quality audio, some are happy with "low" quality[4]; some may be prepared to wait longer for their Web pages or E-mail to download than others[5]. In trying to satisfy the user, the application and the network must interact and co-operate to adapt operation of the application as required.

The Internet protocols were never designed to cope with such a sophisticated demand for services [Cla88]. Today's Internet is built upon many different underlying network technologies, of different age, capability and complexity. Most of these technologies are

[4] The definition of "high" and "low" will typically be application specific.

[5] Everyone will request the "highest" quality service unless there is a trade-off between quality and financial cost. Use of cost-based service provisioning, based on differentiated services for example, is currently of great interest within the Internet community.

unable to cope with such QoS demands. The explosive growth in the use of the Internet has resulted in much of the network being heavily loaded or overloaded. So there is a need to allow controlled use of resources.

In this chapter, we consider the way the use of Internet applications is evolving and what this means for the Internet applications of the future. We start with an overview of the use of the network, network protocols and network applications, then move on to application-level protocols and mechanisms. In particular, in considering the interaction between the network and the application, we look at these two areas:

- retrieving information about resource availability/utilisation from the network;
- giving resource requirement information to the network.

2.1 Towards integrated services networks

People today wish to use the Internet for many different applications. Some of these applications already exist on application specific networks, e.g. voice on POTS, data on X.25 [X.25], CATV networks for analogue broadcast television. As the ability to use a more diverse range of applications becomes available to an increasing number of people, there is a higher demand for these applications. To supply the demand and provide access to such a diverse range of applications, it would be impractical to maintain access to each of the application specific networks for each user. So, over the past two decades or so, there has been a move to provide a single integrated services network that can support the provision of any and all of these applications, e.g. N-ISDN (narrowband-ISDN), B-ISDN (broadband-ISDN). Although, in principle, such a network should be able to provide very good QoS guarantees, the notion of a single, ubiquitous network technology is not realistic (and in fact today's Internet services are provided across networks that consist of many different technologies). The Internet protocols are widely available, generally easy to use, have well-defined software APIs, and can operate on many network technologies. Consequently the Internet is being seen as a means for allowing access to integrated services [DT97]. However, the Internet and Internet protocols were not designed to support the wide range of QoS profiles required by the huge plethora of current (and future) applications. This deficiency is currently being addressed by the IETF INTSERV (Integrated Services) WG[6] [RFC1633].

[6] http://www.ietf.org/html.charters/intserv-charter.html

2.1.1 Integrated services in the Internet

In [CSZ92], the authors speak of the Internet evolving to an integrated services packet network (ISPN). Although they mainly address the importance of end-to-end delay for flows, they clearly favour the ability of applications to have functionality that performs flow adaptation in response to fluctuations in network QoS. The IETF INTSERV WG has proposed an architecture for evolving the Internet to an ISPN [RFC1633]. To support the architecture, INTSERV have produced a set of specifications for specific QoS service-levels based on a general network service specification template [RFC2216] and some general QoS parameters [RFC2215]. The template allows the definition of how network elements should treat traffic flows. With the present IP service enumerated as **best-effort**, currently, two service-level specifications are defined:

- **controlled-load** service [RFC2211]: the behaviour for a network element required to offer a service that approximates the QoS received from an unloaded, best-effort network
- **guaranteed** service [RFC2212]: the behaviour for a network element required to deliver guaranteed throughput and delay for a flow

Also specified is how to use a signalling protocol, RSVP [RSVP], to allow the use of these two services to be signalled through the network [RFC2210]. INTSERV also define SNMPv2 MIB extensions [RFC2213, RFC2214] to allow remote monitoring and management of network elements that support these network services. Part of the INTSERV work is the definition of an architecture for a QoS Manager (QM) entity that co-ordinates flow activities and resource usage at the end system [INTSERVQM].

Note that this architecture requires that the network elements and applications have semantic knowledge about the service-levels for the application flows, as specified in the service templates.

2.2 Mobile applications

As computing devices such as laptops, palmtops and PDAs (personal digital assistants) continue to increase in computing power, users will require similar applications to run on these mobile hosts as they currently have on desktop hosts. It will become increasingly impractical for applications to be re-engineered specifically for mobile platforms, so applications will have an increasing requirement to be designed for adaptable operation in order to cope with mobile environments.

The Internet community has recognised the requirement for mobility support and has produced specifications to allow nomadic IP hosts to maintain network connectivity [RFC2002, RFC2005]. Mobile IP hosts have a **home** location from which they may wander to a **foreign** network, i.e. a network that does not have the same IP network address. The mobile host locates a **foreign agent** within the foreign network and informs its **home agent** about its location with respect to the foreign agent (by way of a **care-of-address**). The home and foreign agents then ensure that packets destined for the mobile host are re-routed to the mobile host by the use of a tunnelling mechanism between them [RFC2002]. The reverse path (from mobile host to sender) will typically not traverse the tunnel, and will be delivered through normal routing mechanisms, but work is in progress that will allow **binding updates** from the mobile host (e.g. in IPv6). These binding updates contain information about the new location of the mobile host to allow a more direct IP communication path. This deals with mobility at the macro-level, providing end-to-end routing of IP packets and movement across IP networks during communication, but does not consider micro-level mobility issues (e.g. link-layer hand-off between cells), relying on link-layer or physical layer mechanisms for such capability. This is in keeping with a general goal of IP to be usable over any network technology. There are also specifications for encapsulation of packets for tunnelling [RFC2003, RFC2004] as well as a SNMPv2 MIB [RFC2006] to allow remote management of devices using IP mobility.

Indeed the scheme presented above typically sets up asymmetric communication paths, and even when binding updates allow symmetric paths, the fluctuations in the end-to-end IP path topology can have adverse affects where higher level protocols rely on feedback from the receiver [BPK97]. Although IPv6 has much better mobility support than IPv4, QoS issues have yet to be fully addressed. Mobile applications that have connectivity via wireless links may also experience QoS fluctuations due to environmental conditions. Transport-level protocols may experience increased errors in received packets, and reliable transport protocols, such as TCP, behave adversely when there is a burst of packet loss (more than one packet) during hand-off between cells [CI93]. Mobile computing is an area that has great need for adaptable applications in order to enable ubiquitous and continuous access [Kat94].

2.3 Network layer and transport layer protocols

A key feature in the design of the Internet protocol (IPv4 [IPv4]) was that it should be simple. From [IPv4]:

The internet protocol is specifically limited in scope to provide the functions necessary to deliver a package of bits (an internet datagram) from a source to a destination over an interconnected system of networks. There are no mechanisms to augment end-to-end data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols.

IPv4 provides a simple packet delivery service that gives no guarantees on QoS. Although IPv4 has some simple flags and a precedence indicator available for use in the **type of service (ToS)** field [RFC1349], support for the ToS is not well deployed. Also, when IPv4 was designed, it was required to work on many network technologies, so there are no QoS features in the design that were specific to any particular network technology. Furthermore, not all network technologies and network paths can support even the limited form of QoS requests made in the ToS field. Consequently, the ToS field has rarely been used for QoS control purposes, but there is now work in progress to revive the use of this field [DIFFSERV1]. IPv6 introduces (potentially) better awareness of QoS into IP by providing an 8-bit **traffic-class** field and a 20-bit **flow-label** in the packet header, allowing IP traffic to be marked for special handling by routers. The exact mechanisms and procedures to process QoS in IPv6 packets have yet to be agreed. INTSERV propose a flow-based scheme with *applications* signalling their requirements to the network (we discuss resource reservation later in Sections 2.6.3 and 2.6.4). DIFFSERV intend the packets to be marked within the *network* (e.g. by ingress routers), and that the IPv4 ToS field and the IPv6 traffic-class field should have the same syntax and semantics.

TCP (transmission control protocol) [TCP] was designed to provide reliable end-to-end (point-to-point) communication on top of IP by setting up a logical connection that uses a scheme of acknowledgements and re-transmissions. TCP also provides congestion control, flow control and recovery mechanisms [RFC2001, Jac88, Jac90], but the behaviour of these mechanisms is not designed to be controllable by the human user or by the application. Additionally, retransmission for *end-to-end* reliability as used in TCP is typically unsuitable for QoS sensitive flows (especially real-time interactive flows such as those for conferencing), as the delay introduced with an end-to-end retransmission scheme can be too large[7].

[7] This does not preclude the use of re-transmission on *individual links* that make up the end-to-end path.

For transporting real-time or QoS sensitive flows, retrieving resource usage information from the network, and signalling the applications' (and users') QoS requirements to the network, additional protocols and mechanisms are required.

2.3.1 Protocols for transporting QoS sensitive flows

With QoS sensitive flows, we see two major concerns:

1. making sure that the packets in the flow are matched to the QoS available in the network
2. making sure that the handling of the packets end-to-end supports the semantics of the flow

For the first of these points, we can use some combination of:

- tell the network what is required for a particular flow so that the QoS is fixed for the duration of the flow, i.e. make a resource reservation for the flow. Typically, this may involve a negotiation between the application and the network, where several different reservations may be attempted and when one is successful, the application can use an appropriate flow-requirement
- adapt to the (possibly fluctuating) QoS offered by the network by changing the construction of the flow in some application specific way, e.g. lower the transmission rate, decrease packet size, etc

For the second point (packet handling), only the application really knows how to deal with the flow as, in general, only the application knows the full semantics of the flow – the meaning of the ADU (application data unit). The information syntax of the ADU can (should) be de-coupled from the information semantics in that the network does not need to handle the information other than to transport it between end-systems [CT90, Laz92]. It is also argued in [CT90] and [Laz92] that any protocol control processing/signalling regarding the ADU is kept out-of-band, i.e. the application-level control signalling is not synchronised with the ADU processing/transmission in order allow greater flexibility, especially with respect to engineering options. This is typified in many communication reference models that show a separate *control-plane* and *user plane*, e.g. B-ISDN. Our interpretation of this is that the semantic information about a flow that is passed to the network should be kept to a minimum, reflecting common application requirements, and that the application should use additional application-level signalling procedures to *control* or *manage* the transport of any specific flow[8].

The Internet community has adopted this approach. The main protocol used for QoS sensitive data flows is RTP (real-time transport protocol) [RTP]. RTP is a general protocol providing an end-to-end delivery system for QoS sensitive flows. RTP itself is not a protocol for negotiating and guaranteeing QoS; rather it provides a framework in which to build QoS sensitive applications. RTP has an associated simple signalling protocol, RTCP (real-time transport control protocol), that allows QoS information such as delay, jitter, loss, etc. to be sent between senders and receivers. The relevant features of RTP are:

- it is an extensible framework
- it uses IP and UDP allowing unicast and multicast communication
- it supports the notion of a communication session, which may consist of many flows
- it supports identification of a session and participants in a session
- it supports dynamic group membership (members can leave and join a session)
- it allows synchronisation and timing of packets within a single flow

The features reflect general requirements for a communication protocol and RTP is used as the basis for transporting many different flow types including various audio and video formats [RFC2205, RFC2198, RFC2190, RFC2035, RFC2032, RFC2029, RFC1890]. Note that RTP is not designed to provide any mechanisms for inter-flow synchronisation or the description of session semantics (e.g. description of flows) – this is considered to be an application-specific issue. However, recent work shows that such session control is generally needed in many multimedia delivery scenarios (especially in conferencing systems) and other work addresses this [RTSP, SDP, Shu97, HWC95].

Retrieval of information from the network about the QoS experienced by a flow is not supported directly by IP. We can split the problem into two: taking a measurement and distributing that measurement information. All QoS architectures support some method of distributing measurement information [CAH96]. RTP supports time-stamping

[8] This reflects the Internet philosophy of keeping the state in the network to a minimum, thereby keeping the network as simple as possible.

mechanisms and also allows transport of other (application-specific) QoS parameter measurements. We say more about the measurement process itself, later.

2.3.2 Congestion control and flow control

The first attempts at QoS control for the Internet were designed to makes flows adaptive to combat congestion (congestion control) within the network due to high volumes of data (compared to the network capacity), and to try and allow the end-to-end delivery rate of data to be controlled (flow control). It is useful to discuss the properties of such mechanisms in the context of adaptable systems. Initially, such mechanisms were often built into network protocol and transport protocol operation, and were not designed to be controllable by the user. More recently, schemes have been proposed that are built into the application and allow some form of adaptability (e.g. [KHC98, VRC98]). There are some interesting and relevant issues:

- detecting network QoS variations in the network
- the behaviour of protocol operation when there are QoS fluctuations
- adaptation mechanisms

There are various schemes for congestion control and flow control in TCP (e.g. [Jac88, Jac90, WC91]), as well as for QoS sensitive packet flows (e.g. [Kes91, KMR93, BTW94, BV96]). Many of the schemes (especially TCP schemes) function in similar ways because they [YR95][9]:

1. model a closed-loop feedback control system with the feedback coming from the network (e.g. the remote host) to adjust the **set-point** with respect to the traffic flow
2. rely on the fact that all users run the same algorithm, are willing to co-operate and will receive a fair-share of the network resources
3. work to be "kind" to the network as well as provide a "good" service to the user
4. use end-to-end delay as a measure of network congestion
5. do not use the raw delay values but typically apply some sort of smoothing mechanism to form an estimate of the current delay (e.g. TCP is based on the use of [KP87])

The first point shows that these mechanisms are trying to be adaptive in response to changing network conditions. The second point is less true today than in the past

[9] This small list extracts general items of interest; see [YR95] for a full taxonomy and overview.

[LLSZ96], and, certainly within an integrated services network, some applications may require more resources than others may. "Fair" does not necessarily imply "equal" [DKS90], as some users may be willing to pay a premium for a "superior" service that allows access to more resources. Additionally, where there is no single mechanism or algorithm for adaptability, two different users with applications of equivalent functionality, the same user preferences and the same network QoS may get different adaptation behaviour. The (desired) property in the third point is found to be poorly achieved in some cases, and the use of acknowledgement packets as timing/clocking devices is found to be questionable [ZSC91]. The last of these points is of particular interest as whatever smoothing function these mechanisms apply must be accurate enough to give a true reflection of the current state of the network. We discuss this further in Section 2.5.

The adaptation action is to change the transmission rate. In TCP, the mechanism used is a conservative back-off policy (multiplicative decrease, additive increase), that has the granularity of a byte. This adaptation mechanism is not accessible for manipulation by the application. In an application with QoS sensitive data flows, however, such a policy and such small granularity of data transfer may not be suitable. For example, audio applications typically send ADUs contain a "time-slice" of audio (e.g. 20ms, 40ms or 80ms ADU "size" are possible in *vat* [vat] and *rat* [HSK98]). These applications require a different model which reflects the ADU flow granularity and allows access to the adaptation process, i.e. based on adaptation of flow-requirement (as well as set-point).

We note that the mechanisms above all rely on information about the flow measured from the network in order to adapt the flow-requirement and set-point (and perhaps the application mode), and that the adaptation decision depends on having a good estimate of some QoS parameter for that flow. Also, very importantly, the adaptation behaviour is application-specific.

2.4 Getting information from the user of the application

User preferences affect the behaviour of an application by selecting a mode of operation for the application. User preferences are often concerned with presentation of information at run-time and may be quantitative (e.g. font size, picture size, frame rate, etc.) or qualitative (e.g. "large" text, "high" quality picture). Typically, the latter are preferred, as they require less technical expertise from the user. For example, it is easier to give the non-technical user an enumerated choice of "high", "medium" and "low" quality for a voice application than to ask the user to pick a particular audio encoding scheme given information about its QoS parameters such as data rate, error tolerance, etc. There are many mechanisms for soliciting and obtaining user preferences and we do not consider these in our work. However, what is of importance is that the user is not necessarily concerned with the details of operation of the application but simply wants the application to perform reasonably well given the user's preferences and the application's operating conditions.

As shown in Figure 2.1, a decision function must take information from the user and network resource information (e.g. QoS measurements for a flow) as well as information that is application specific (dynamic runtime information as well as static information about capabilities), and use this to make an adaptation decision. The adaptation behaviour might involve changing the flow-requirement and application mode. Typically, an application relies on a user to implement the decision function by selecting the correct user preferences. For example, if a user preference indicates transmission of high quality video even when connected using a low-speed link, the application will try to service this request and it is up to the user to figure out why things do not work. To assess whether or not the user preferences are feasible, the application needs information from the network about the network resources available to it.

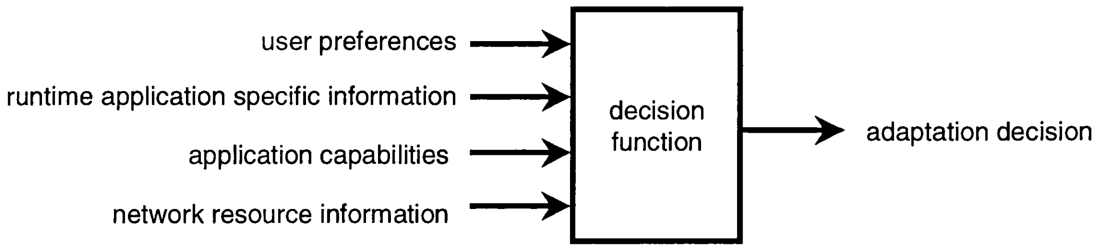


Figure 2.1: Interactions between the user, the application, and the network for adaptation

We would like to be able to move as much of the decision function into the application as possible. This does not necessarily mean that the decision function is totally automated, but may be semi-automated (or even totally under user control), with the application presenting a user-friendly QoS summary to the user for them to make an informed selection of the user preferences.

2.5 Getting information from the network

Information about network QoS is typically expressed as values of QoS parameters such as delay, jitter, available capacity, loss, etc. This information may be obtained in a number of ways:

- via local mechanisms, e.g. from the communication stack on the host
- via application-specific mechanisms, e.g. via proprietary signalling
- as control messages from the remote receivers, e.g. using RTP/RTCP [RTP]
- from network management tools, e.g. the RTFM MIB [RFC2063, RFC2064, RTFM1, RTFM2, RTFM3], using SNMP [SNMPv2]
- from a QoS/resource manager that receives information from the above mechanisms

We must appreciate that there is likely to be no single best, ubiquitous solution for getting information about the resources within the network. The best mechanism may depend on the network environment or the application's function or both, but only the application (application designer) is in a position to make that assessment [SRC84]. So, any network model that requires information about QoS parameters should not put any constraints on where that information should come from [CAH96]. However, any measurement process we use, as well as having its own behaviour and errors, will also be affected by noise and delay.

2.5.1 Timeliness of network measurements

If we have a system that allows us to receive measurements from the network, not only are these measurements noisy, but they are also always out of date due to network delays. So, when we need to use the values of some parameter, we require an estimate of its likely current value, rather than use the measured value directly. This immediately poses a problem – in order to predict the current value we need to:

1. remove noise from the measurement
2. have a model of the measured parameter that allows us to make a prediction

2.5.2 Noise in network measurements

It is not always possible to separate the noise from the normal system perturbation. The following treatment of measurements is from [KK92]. Consider a QoS parameter, *P*. In general, we have no knowledge about the distribution of the values of *P*, so we treat it as a random variable. We can consider values of *P* as a sequence $\{p_1, p_2, \ldots, p_k, p_{k+1}\}$, and we represent this as:

$$p_{k+1} = p_k + w_k \qquad \textbf{(2.1)}$$

where:

p_k	the value of the parameter *P* at time *k*
w_k	system perturbation

The sequence of measured values { $\tilde{p}_k$ } is represented as:

$$\tilde{p}_k = p_k + u_k \tag{2.2}$$

where:

$\tilde{p}_k$	the measured value of the parameter *P* at time *k*
p_k	the value of the parameter *P* at time *k*
u_k	observation noise

We also, in general, have no knowledge about the values of observation noise, u_k, and so it too appears as values from a random variable, *U*. As both *P* and *U* are random variables, we cannot predict their values accurately. We can see the difficulties by considering the scenario depicted in Figure 2.2.

Figure 2.2 shows a real scenario of some hosts connected by several different network technologies using IPv4. We use a simple, lightweight scheme of probes, which we will call **RDJ (rate, delay and jitter) probes** (see Appendix A for details), to take measurements of "available capacity" (data rate) along the network path between the hosts.

Consider first the path between the hosts `theakston` and `darhu`. We have a network route that contains 3 router hops. The LANs are both 10baseT and the ISDN connection between `adnams` and `ascend` uses a single B-Channel (i.e. maximum of 64Kb/s). Figure 2.3 shows a `traceroute` [traceroute] for the path between `darhu` and `theakston`, clearly showing that the hop between `ascend` to `adnams` (hop 2 to hop 3) is the delay bottleneck.

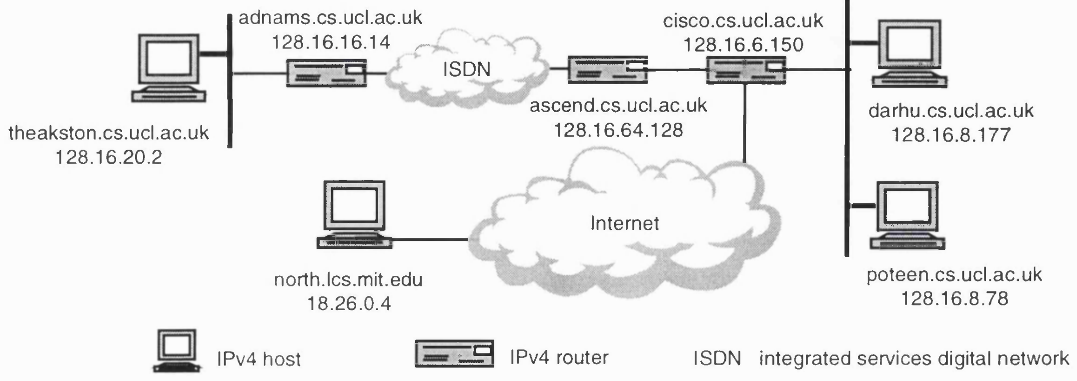


Figure 2.2: A connectivity scenario between UCL and MIT

During the time that the measurements were being taken, no other traffic other than the RDJ probes crossed the ISDN line. The graph in Figure 2.3 shows the available capacity on this path as measured by the RDJ probes. We note that:

- because there is no other traffic being sent, we know that the system perturbation is zero with respect to the B-Channel, so all the fluctuations are due to noise
- the amount of noise on the graph appears to be large, with many spikes (outliers) in the data
- as we know the network configuration, that the overall path has a very light load[10] and that there is no other traffic on the ISDN line, we could look at the graph and make a reasonable intuitive guess that the available capacity on the path is 57Kb/s[11]

```
darhu.cs.ucl.ac.uk %traceroute -n theakston
traceroute to theakston.cs.ucl.ac.uk (128.16.20.2)
 30 hops max, 40 byte packets
 1  128.16.6.150  3 ms  3 ms  2 ms
 2  128.16.64.28  3 ms  3 ms  5 ms
 3  128.16.16.14  31 ms  33 ms  30 ms
 4  128.16.20.2  32 ms  30 ms  31 ms
darhu.cs.ucl.ac.uk %
```

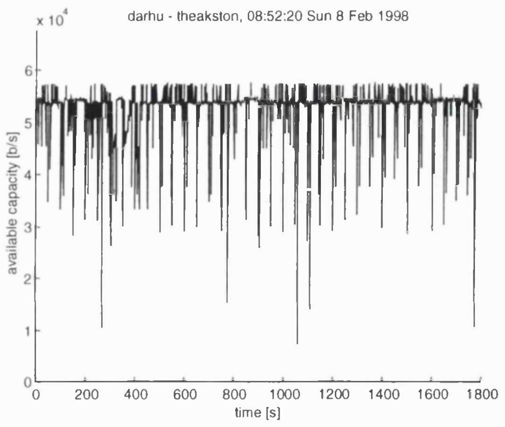


Figure 2.3: RDJ probes estimate of capacity between `darhu` and `theakston` (see Figure 2.2)

If we examine Figure 2.4, the path between `poteen` and `north`, we do not have the same level of *a priori* information about the network. The `traceroute` in Figure 2.4 reveals delay bottlenecks (e.g. hop 5 to hop 6) but tells us nothing about the nature of the network or the presence (or otherwise) of other network traffic. Indeed, the graph seems to indicate the behaviour of the measured value of R is modal (seems to fluctuate – almost periodically – between values of ~30Kb/s and ~250Kbs), and we can not determine why without additional information. It is difficult to make an assessment of the available capacity on this path simply by looking at the graph, let alone on a measurement by measurement basis in real-time.

[10] The measurements were taken between 7.00am and 11.00am (UK time) on Sundays, a period when the cs.ucl.ac.uk network traffic is very light.

[11] 57Kb/s is about what one can expect, allowing for protocol overhead (PPP, IP, TCP) some inefficiency and any error/bias in the measurement process itself. Large ftp transfers from theakston (a Windows NT v4.0 Intel-based machine) measures similar figures.

```
poteen.cs.ucl.ac.uk %traceroute -n north.lcs.mit.edu
traceroute to north.lcs.mit.edu (18.26.0.4)
 30 hops max, 40 byte packets
 1  128.16.6.150  2 ms  2 ms  2 ms
 2  128.40.14.245  2 ms  2 ms  2 ms
 3  128.40.20.254  2 ms  2 ms  2 ms
 4  194.83.100.62  3 ms  3 ms  3 ms
 5  193.63.94.85  3 ms  5 ms  3 ms
 6  207.45.206.241  87 ms  85 ms  85 ms
 7  207.45.199.233  103 ms  103 ms  103 ms
 8  144.228.164.61  106 ms  107 ms  106 ms
 9  144.232.5.93  106 ms  105 ms  106 ms
10  144.232.5.6  106 ms  107 ms  106 ms
11  144.228.180.10  107 ms  108 ms  108 ms
12  4.0.2.22  113 ms  112 ms  112 ms
13  4.0.1.202  129 ms  129 ms  128 ms
14  192.233.33.3  131 ms  141 ms  129 ms
15  18.168.0.14  128 ms  127 ms  132 ms
16  18.10.0.1  131 ms  131 ms  134 ms
17  18.24.11.1  131 ms  156 ms  147 ms
18  18.26.0.4  133 ms  *  137 ms
poteen.cs.ucl.ac.uk %
```

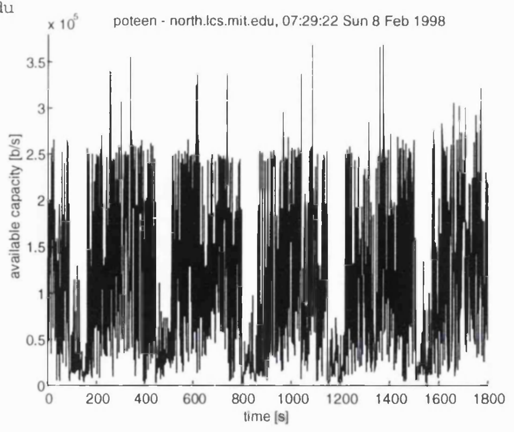


Figure 2.4 RDJ probes estimate of capacity between `poteen` and `north.lcs.mit.edu` (see Figure 2.2)

2.6 Giving information to the network

If the application knows the user preferences, can map them on to its own functional capability and knows the network capability, it can determine the flow-requirement it should use. It can then use the network in two ways with respect to the network:

- **actively:** it can ask the network to support its chosen flow-requirement by giving the network a description of the QoS it requires for that flow-requirement. This may, in general, involve a negotiation during which this initial QoS request may fail and the application must make a QoS request for another of its flow-requirements
- **passively:** it chooses an initial flow-requirement in which to operate and then changes its flow-requirement, as required, depending on the QoS it detects is being offered to it by the network

Let us consider the active case in this section. We will assume that a QoS mapping of the user preferences is available. The application now needs:

- a mechanism for specifying its own functional capabilities (its flow-requirements) in terms of the requirements of its flows (**QoS specification**)
- a mechanism for mapping those flow requirements into something the network can understand, i.e. a statement of its resource requirements (**QoS mapping**)
- to signal its network resource requirements to the network (which the network needs to propagate to the network elements along the communication path for that flow) and notify the application of success or failure, as required, i.e. it needs to make a **resource reservation**

We look at each of these in turn.

2.6.1 Application capability

The simplest, general, way for an application to express its requirements to the network is by describing the requirements of the application-level flows. Typically, these will be performance bounds defined in terms of quality of service parameters and traffic characteristic specification using parameters such as those in Table 2.1.

The traffic characteristic is commonly defined in terms of a token bucket filter, (R_B, B_B) where R_B is the normal transmission speed (units bytes/s) and B_B is the bucket depth (units bytes). B_B might be chosen to be a multiple of the (maximum or mean) ADU size. This method is used in [RFC2216] by INTSERV, which defines a *FlowSpec* to consist of a *RSpec* (the flow requirements) and a *TSpec* (the traffic characteristics). The idea of a *FlowSpec* was first proposed in [RFC1363]. The exact number and type of parameters required for the *RSpec* and *TSpec* will vary, depending on the service level requested [RFC2211, RFC2212]. The *RSpec* and *TSpec* must be defined by the application designer. This requires the designer to have detailed knowledge and understanding of the characteristics of the traffic flow for the application and the way these characteristics should be described using such a specification[12].

Parameter name	Description	Units
throughput, R	the minimum data rate required for this flow	bits/second [b/s] or bytes/s [B/s]
delay, D	the end-to-end delay for the data flow	seconds [s]
jitter, J	the inter-packet delay variation	seconds [s]
ADU (packet) error rate, E_{ADU}	the ratio of ADUs containing bit errors to correctly delivered ADUs	-
ADU (packet) loss rate, L_A	the ratio of lost ADUs to ADUs delivered without errors	-

Table 2.1: An example of some application-derived QoS parameters

[12] Our experience shows it is not always true that the designer is aware of the exact traffic characteristics of the applications, as this may, in general, depend on the particular usage of any instance of the application.

2.6.2 Network capability

There must also be a specification of what the network is capable of. In the INTSERV model, the description of the network element capabilities is defined in terms of one of the INTSERV templates [RFC2216].

However, recall that in the INTSERV architecture we noted that there is a guaranteed service and a controlled-load service. These abstractions are each a **QoS service-level**, and reflect the way in which the QoS specification is honoured by the network. Given the same *TSpec*, the guaranteed service has a greater probability of meeting the *TSpec* than the controlled load service, which in turn will have a greater probability of meeting the *TSpec* than the best-effort service[13].

In terms of our discussion, this allows specification of the requirements for one or more flow-requirements.

2.6.3 Resource reservation

Our application designer now knows how to define what is required in terms of QoS for an application's flow-requirements and there are mechanisms available that can gather measurements from the network. We must now try to make sure that the network (i.e. the network path that the data will traverse) will support the requested QoS. This is the job of a signalling protocol such as RSVP [RSVP] or ST2+ [ST2+]. The INTSERV architecture favours RSVP, so we confine our discussion to RSVP. A comparison of RSVP and ST2+ is presented in [MESL94] and [Bas96]. A more detailed description of RSVP is given in [WC97], and only a simple outline is presented here.

RSVP allows the application and routers in the network to signal each other about the resources that need to be allocated for a communication session. It can be used for multicast or unicast communication. RSVP identifies communication sessions by use of the destination IP address and port number, and the transport protocol identifier (as carried in the IP protocol field). All packets that form part of a session are given the same QoS. There may be many sessions between any receivers and senders. We can equate an RSVP session with our current notion of a flow, but RSVP allows many RSVP-flows per session. Senders initiate resource reservation by sending a *Path* message to the

[13] An analogy might be that of registered, first class and second class post, respectively; each will get the letter there but give you different levels of service (and cost different amounts).

receiver(s). The routers that see the *Path* message hold some resources and make a note of the route of the message. The routers then wait until a *Resv* message is received along the reverse path from a receiver, or the sender transmits a *PathTear* message, or the *Path* request times out. If the *Resv* is received, the held resources are allocated. If either the *PathTear* is seen or the *Path* request times out, then held resources are released. This last mechanism is a distinguishing feature of RSVP – it uses **soft-state** (which must be periodically refreshed) held in routers so that it does not need support from any particular network technology, and end-system failures do not tie up resources. At the end of a session (or when a receiver leaves a multicast group) the sender transmits a *PathTear* to release resources (or a receiver sends a *ResvTear*). RSVP can effectively provide a per-flow reservation capability.

Note that the use of RSVP requires the application to be aware of the protocol interaction for RSVP so that it can control how the session uses resources. This awareness could be passed on to the user to allow the user to make more subjective decisions, e.g. on cost versus QoS service-level. For example, the application makes RSVP requests for guaranteed service and controlled-load service and the network could reply with an indication as to the cost of both services. The user could then make a decision to request a controlled-load service, which may be less costly.

2.6.4 Reservations about reservations[14]

Why do we need dynamic adaptability if we have resource reservation? We have mentioned that resource reservation with RSVP is a useful mechanism for applications with QoS sensitive data flows. However, as IP cannot rely on any particular network technology-specific mechanisms, RSVP uses a soft-state technique with a two-pass protocol. We summarise the main problems with RSVP below [SB95, WGS97]:

1. during reservation establishment if the first pass of each of two separate reservation requests are sent through the same network element, where one request is a super-set of the other, the lesser one may be rejected (depending on the resources available), even if the greater one eventually fails to complete (of course it is possible to re-try)
2. if the first pass does succeed, the router must then hold a considerable amount of state for each receiver that wants to join the flow (e.g. in a multicast conference)

[14] This title taken from a Panel Session at the Fifth International Workshop on Quality of Service (IWQOS'97), 21-23 May 1997, New York, New York, USA.

3. the routers must communicate with receivers to refresh soft-state, generating extra traffic, otherwise the reservation will time out
4. complete heterogeneity is not supported, i.e. in a conference everyone must share the same service-level (e.g. guaranteed or controlled-load), though heterogeneity within the service-level is supported
5. if there are router failures along the path of the reservation, this results in IP route changes, so the RSVP reservation fails and the communication carries on at best-effort service, with the other routers still holding the original reservation until an explicit tear-down or the reservation times out

Resource reservation could be expensive on router resources and adaptation capability is still required within the application to cope with reservation failures or lack of end-to-end resource reservation capability. Indeed the shortcomings of RSVP, especially with respect to scaleability, have been acknowledged by the Internet community and it is now recommended for use only in restricted network environments [RFC2208]. Such concerns about resource reservation have directed the Internet community to consider alternatives; specifically differentiated services [DIFFSERV1]. Without resource reservation, we require a more flexible and dynamic adaptation capability within the application.

So, good dynamic adaptation capability in the application has value even where resource reservation might be an option.

2.6.5 Differentiated services

The IETF DIFFSERV (Differentiated Services) WG[15] takes a different view of using network resources. This work is still at very early stages, so there are several schemes being discussed [DIFFSERV2, DIFFSERV3, DIFFSERV4, DIFFSERV5, DIFFSERV6, DIFFSERV7]. The general model is to define a class-based system where packets are effectively marked with a well-known label in the IPv4 ToS field or IPv6 traffic-class field [DIFFSERV1]. This label identifies the service-level the packet will receive, much like a letter can be marked as registered, first class or second class delivery. This is a much coarser granularity of service, but reflects a well understood service model used in other commercial areas. The DIFFSERV model is different to RSVP. A key distinction of the DIFFSERV model is that it is geared to a business model of operation, based on administrative bounds, with services allocated to users or user groups. Whereas RSVP can act on a per-flow basis, the DIFFSERV classes may be used by many flows. Any packets within the same class must share resources with all other packets in that class, e.g. a particular organisation could request a Premium (low delay) quality with an Assured (low loss) service-level at a given data rate from their provider. The packets are treated on a per-hop basis by *traffic conditioners*, routers that determine the way a packet should be treated based on a policy that is selected by examining the value of the ToS/class-field. The policy could be applied to all the traffic from a single user (or user group), and could be set up when subscription to the service is requested [DIFFSERV3], or on a configurable profile basis [DIFFSERV2, DIFFSERV7]. The DIFFSERV mechanisms would typically be implemented within the network itself, without requiring runtime interaction from the end-system or the user, so are particularly attractive as a quick means of setting up tiered services, each with a different price to the customer.

The RSVP mechanism seeks to introduce well-defined, end-to-end, per-flow QoS guarantees by use of a sophisticated signalling procedure. The DIFFSERV work seeks to provide a "virtual pipe" with given properties in which the user still requires adaptation capability if there are multiple flows competing for the same "virtual pipe" capacity. Additionally, the DIFFSERV architecture means that different instances of the same application throughout the Internet could receive different QoS, so the application needs to be adaptable. Indeed, in [DHT95], the authors conclude that coarse-grained differentiation based on service-classes with application adaptation should be sufficient for multimedia applications:

> *If we accept the idea that all members of the same class should be treated equally, we find very little use for reservation. In fact we see two [uses for] the reservation procedure: a revenue opportunity for network operators and a prioritization tool for network recipients.*

2.7 Application adaptation and heterogeneity

In Section 2.5, we talked about getting QoS information from the network, and in Section 2.6 we talked about the need for passive applications that do not use resource reservation, but adapt to network changes based on this QoS information. Once the QoS information has been received, the application must interpret it to make a decision about whether or not it should change its flow-requirement. Such a decision must be made not

---

[15] http://www.ietf.org/html.charters/diffserv-charter.html

only in an application-specific manner, but will be subject to user preferences. However, we can see that the application must somehow determine which of its flow-requirements can be supported by the network given the network QoS information it has, and only then can it make a sensible decision about how to act with respect to the user preferences. When the network QoS is seen to be fluctuating rapidly, the application must be able to make decisions that do not result in "state-flapping" (instability) – any decision making function must be stable with respect to the functionality of the application and the user preferences.

Consider again Figure 2.2. It is unrealistic to suppose that the kind of detailed *a priori* information we have about the `darhu-theakston` path (network configuration, link capacity, knowledge of network traffic) will be available to a user in all circumstances, at all points in the network, and at all times. Certainly this is not true in the case of the `poteen-north` scenario. Furthermore, let us consider that the information about the available capacity is to be used by an application to make a decision about how it should behave and operate. Even if such information was readily available *a priori* for all possible network paths for a user, that user would then need either to make an adaptation decision based on that knowledge or pass that knowledge to the application (somehow) and trust that the application could make a suitable decision. There are some problems here:

- the user would need to be very knowledgeable to make a sensible decision based on the kind of QoS information that we have presented thus far. Not only would the user require detailed information about the network, but also detailed information about the operation of the application
- even if this first point was true, in a more complex (i.e. realistic) scenario, the user would need to share the network resources on the paths with many other traffic flows. So, there is a strong possibly that there would be many, rapid variations in QoS. The user would need to make many adaptation decisions during the use of the application. Such a process may require much of the user's time and attention, so we need support for an automatic QoS assessment process that would ease the user's task or allow the application to make a decision
- there is currently no general model by which the application can come to a decision as such a decision in a particular connectivity scenario can only be achieved by the result of the interaction between user, application and network

The last point is crucial: how can the application make an adaptation decision? One could argue that the application can make the decision based solely on network information as in Scenario 1b (Section 1.1). However, consider the case of Scenario 1b or Scenario 2b when different instances of the application are running over different networks, but are now using different resource reservation schemes or different service-levels, and are controlled by different user preferences. The amount of information required to make the adaptation decision is large and complex; it is becoming increasingly difficult for the application to make the decision. This complexity is evident in the following anecdotal scenario.

When the multimedia research group at UCL are to have a wide-area conference across the Internet using tools such as *rat* [HSK98] and *vic* [MJ95], they typically spend some time before the conference assessing the network quality and decide which audio encoding to use and the rate of the video. During the course of the conference, there will be someone at many (sometimes each) of the conference sites monitoring the conference quality. These people will communicate out-of-band in order to make co-ordinated adjustments to the operation of *rat* and *vic* as the quality changes (e.g. change the audio encoding or drop the video so that the audio can continue at reasonable quality).

There are three key points in this scenario:

1. the decision as to how to adapt is made through the expert knowledge and experience of the users
2. the decision making process requires all participating sites to come to the same decision, i.e. they have the same expertise and knowledge ("same algorithm")
3. the tools themselves rely on the humans to make the major decisions about the state of the applications (user preferences)

In this chapter, we have noted that there are mechanisms that may aid the application in obtaining information about the QoS offered by the network for a particular flow, as well as allowing the application to request QoS guarantees from the network. However, in the face of heterogeneity, it is still very hard to enable the application to make dynamic and automatic **QoS assessments**, or allow the user to make an informed change of user preferences. There is no capability that allows assessment of all the QoS information available to the application. We need to offer the application a mechanism that provides a suitable **QoS information summary** that allows it to *dynamically* make an assessment of how it should adapt. It may then be possible for it to interact with the user in order to change flow-requirement or mode, or even make an adaptation decision automatically.

2.8 Chapter summary

People now wish to use the Internet to provide access to integrated services. This includes the use of applications with QoS sensitive data flows, which require different QoS services from the network. The Internet protocols were never designed for such sophisticated use, but there is now work in progress to address these issues within the IETF INTSERV WG. INTSERV has produced a **QoS architecture** and defined two **QoS service-levels**, namely **guaranteed** service and **controlled-load** service. These can be requested by applications using RSVP. (Section 2.1.)

Additionally, applications have to run on mobile hosts, in environments where the QoS can vary. The Internet community now has specifications to ensure end-to-end connectivity for mobile hosts, but issues concerning QoS have yet to be addressed. This means that applications running on mobile hosts will have to be adaptable to the QoS available in their particular network environment. (Section 2.2.)

TCP and UDP alone cannot support applications with QoS sensitive data flows. Mechanisms such as flow control and congestion control in TCP are not controllable by the application. The application cannot control exactly the way it sends data using TCP, so application specific transport/session mechanisms must be built on top of UDP. Such a protocol is RTP, which allows some application level signalling. Information about QoS parameters such as throughput, delay, and jitter can be passed between application instances using RTCP. However, RTP is not designed to get information from, or send information to, the network elements. (Section 2.3.)

In order to be adaptable, the application needs a **QoS mapping** to allow it to interpret the user preferences in terms of its own capabilities and the network's capabilities (Section 2.4). It also needs to get **QoS information** about flows from the network, i.e. measurements of **QoS parameters**. However, measurements from the network are always out of date and contaminated with noise, so there is a need to find a suitable estimation and error-filtering mechanism. (Section 2.5.)

If the application is operating actively, using resource reservation, it also needs to send control signals to the network, e.g. to request QoS services. This is done using a resource reservation protocol such as RSVP. In order to specify the QoS it requires for its flows, the application must define **QoS flow performance specifications** in terms of **QoS parameters**. A particular network path may not always support per-flow reservations as that provided by end-to-end mechanisms such as RSVP, or RSVP could fail during the lifetime of a flow. Other QoS mechanisms, which do not necessarily act on a per-flow basis (such as differentiated services), may be in operation. So, the application must still be prepared to adapt. (Section 2.6.)

Even if excellent resource reservation capability is available, the application is faced with heterogeneity due to user preferences, different network conditions across the Internet, differing resource reservation regimes and QoS mechanisms, as well as host mobility. This means that the application needs to have adaptation capability. However, with such a diverse and heterogeneous range of connectivity and QoS possibilities, it is unreasonable to expect the user to be knowledgeable enough to set user preferences to select the correct application mode and flow-requirements for a given QoS/connectivity scenario. So, we would like to offer application mechanisms that offer a **QoS information summary** that can be used to make adaptation decisions dynamically, or that help the user to make informed choices of user preferences. (Section 2.7.)

3. Adaptability in Internet applications

In Chapter 2, we looked at the way the use of the Internet is changing, with users now expecting enhanced services. From the discussion, we draw two main conclusions:

1. applications with QoS sensitive flows need to be adaptive
2. there is a need for mechanisms to help those applications make adaptation decisions

In this chapter, we build on these conclusions by examining some of the issues raised in Chapter 2, and also look at the way that adaptability is being addressed in current research.

We start by giving a clearer description of adaptability and listing a set of requirements for supporting dynamic adaptability in end-systems. We then formalise the discussion on QoS presented in Chapter 2 by introducing the notion of a **QoS framework**. We discuss how general application QoS requirements fit in with enabling adaptability for Internet applications. Our aim in this chapter is to emphasise the importance of adaptability and especially the reason for choosing the main subject of our work. So, we examine the issues in current research into adaptability and show that the item of work on which we have chosen to concentrate is not currently being addressed adequately elsewhere.

3.1 Description of adaptation capability

We have argued for the need for dynamic adaptability in applications, but we first need to clarify our description of adaptability. We can think in terms of two classes of adaptability[16]:

- **dynamic adaptability:** during operation the application monitors network resources during its operation and changes its flow-requirement according to the QoS offered by the network to the flow
- **static adaptability:** the application evaluates the network resource availability when it begins operation and then assumes that the QoS will be available during its period of operation, so continues to use the same flow-requirement

It is clear that either Scenario 1b or Scenario 2b (Section 1.1) could operate under either class of adaptability if the users so wished. Applications that have static adaptability may need the support of resource reservation for ensuring end-to-end QoS. The limiting case is that the application has static adaptation capability with only a single mode or flow-requirement, i.e. it is actually a non-adaptive application and requires an end-to-end QoS guarantee. We have argued (Section 2.7) that dynamic adaptability is desirable in a best-effort network as the QoS can fluctuate during the operation of the application. However, there is another distinction between static and dynamic adaptability. We can consider that static adaptability may be provided using existing network management techniques and resource reservation mechanisms. For example, if our mobile host in Scenario 2b (Section 1.1) has a SNMP agent running, then it can detect which interface (ISDN, GSM or Ethernet) is active and discover its capability. It could then make a resource reservation based on this information (using RSVP), and then assume that the QoS will be constant throughout this period of operation[17]. Additionally, resources could be renegotiated during the lifetime of the flow. However, if resource reservation is not practicable or possible, and the QoS fluctuates during the period of operation of the flow, then additional mechanisms for adaptation are required. This example suggests that the availability of both static and dynamic adaptability, together, appears to be useful.

[16] [Cam97] also notes *static* and *adaptive* response to QoS fluctuations.

[17] Even if using RSVP within the Internet, this assumption may not hold, as explained in Section 2.6.4

3.2 Requirements for dynamic adaptability

We assume that the application can support different flow-requirements, each of which requires a different quality of service from the network. We offer the points listed R1 to R8 (below) as necessary and sufficient requirements for enabling dynamically adaptable applications, based on our discussion in Chapter 2. R1 to R8 help us to examine related work as well as to determine the design for our suggested solution.

R1. **QoS integration interfaces:** the provision of an adaptability mechanism requires interaction between the user, the application and the network[18], so we need to be able to offer integration of functional elements at each of the interfaces between these respective entities (see Figure 1.3)

R2. **QoS mapping:** the abstractions for expressing QoS at the user-application interface and at the application-network interfaces could be very different, so a mapping mechanism is required, which should be kept as simple as possible

R3. **QoS specification:** in order to allow adaptability, there must be stable flow-requirements defined for the application flows, so we need a mechanism for specifying the flow-requirements, which should be kept as simple as possible

R4. **QoS compatibility assessment mechanism:** an application adaptation needs to know when to change flow-requirement, so we need a mechanism to assess the network QoS and indicate which flow-requirements can be supported by the network

R5. **QoS information separation:** the behaviour of the application may be controlled not only by network QoS information, but additional application-specific information or user preferences, so the use of the QoS information about flows should not unduly constrain the design or operation of the application

R6. **QoS mechanism separation:** any adaptation decision is an application-level issue, so the QoS information made available from the network should be easily useable and accessible by the application and the mechanisms for its provision should not unduly constrain the design or operation of the application

R7. **QoS encapsulation:** the representation of network QoS information should offer an abstraction that is not confined by any particular network technology, but seeks to hide the complexity and details of the network technology, i.e. presents an application-level viewpoint

[18] In fact, we also need the co-operation and support of the end-system, the host on which the application instance is currently executing, but we say more about this later. Also, recall that our work considers only the application-network interaction.

R8. **QoS heterogeneity:** the representation of QoS information should be consistent in the face of heterogeneity in network service provision, e.g. if the same data rate is available at different service-levels (say guaranteed and controlled-load) a flow-requirement that uses that data rate should still be achievable

We have chosen to model our application as consisting of flows that have well-defined flow-requirements. The application's use of any one of the flow-requirements is dependent on the QoS within the network. The flow-requirements and the network QoS can be expressed using QoS parameters. We can state a very simple overall requirement: the application needs to have information that allows it to assess when it should change flow-requirement.

3.3 QoS architecture

In Section 2.8, we highlighted the following items in describing adaptable applications:

Q1. QoS architecture
Q2. QoS information
Q3. QoS service-levels
Q4. QoS flow performance specifications
Q5. QoS parameters

We have seen that in fact the need for an application to be adaptable is because it experiences fluctuations to its network QoS. So we need a framework in which to talk about QoS issues that is suitable for addressing R1 to R8 in terms of Q1 to Q5.

A description of the elements for a comprehensive **generalised QoS framework** for providing end-to-end QoS in distributed multimedia systems are formulated in [CAH96]. [CAH96] pulls together a large body of current work and also includes a review of some recent approaches to QoS architectures from various different data communication, distributed systems and telecommunication communities[19]. [CAH96] suggests a generalised framework that is based on the following:

- **QoS principles:** five principles governing the design of the framework
- **QoS specification:** for allowing the definition of QoS services and QoS requirements

[19] [CAH96] is a comprehensive survey and review, so we do not produce a full survey of QoS architectures, and choose only to examine points of relevance and interest.

- **QoS mechanisms:** for describing the elements that allow the provisioning and control of QoS services

The remainder of this section discusses the dynamic adaptation requirements R1 to R8 in terms of this generalised framework. We draw on work not only from the Internet community, but also from related data communications and telecommunications areas. This helps us to understand the role of dynamic adaptability in terms of the larger goal of QoS awareness for Internet applications.

3.3.1 QoS principles

Our interpretations of the five QoS principles in [CAH96] are given in Table 2.1.

Principle	Description
integration	QoS must be configurable, predictable and maintainable over all architectural layers to meet end-to-end QoS (across end-system modules, protocol stacks and network components)
separation	media transfer, control and management are functionally distinct architectural activities, and will be application-specific
transparency	applications should not be required to deal with the complexities of the underlying QoS specification, QoS management, and QoS maintenance mechanisms and procedures
asynchronous resource management	the timelines governing the management of resources (end-system or network elements) should be de-coupled from the timelines for normal operation of the application
performance	keep things as simple and efficient as possible, leaving the application (or application manager) with the final say

Table 3.1: Interpretation of the five QoS principles from [CAH96]

Our main concern is the first principle, that of integration. We make the following distinction between the provisioning of end-to-end QoS and enabling dynamic adaptability:

For the provisioning of end-to-end QoS, the user/application is telling the network what is required and asks that the network should adapt/configure itself to comply to the user's/application's requirements, i.e. static adaptation. In enabling dynamic adaptability, the application receives information about the QoS that the network can offer, and changes its flow-requirement to comply with the network capabilities.

With this in mind, the principle of integration does not have to apply so strongly for adaptable applications. Our integration needs, R1 are somewhat weaker, as we do not need QoS to be "configurable, predictable and maintainable over all architectural layers". What we require is a well-defined interface between the application and network to receive an application-oriented view of the network capabilities; i.e. something that is easily usable and understandable at the application level.

Our weaker integration requirement is both an advantage and a disadvantage. It is an advantage in that it requires looser coupling between the various architectural components giving greater freedom in system design, and this is consistent with R5 and R6. In fact, this looser integration is a requirement for Internet applications, as IP is not biased towards to any particular network technology. QoS architectures seeking to offer end-to-end QoS guarantees show tight coupling between the application-level QoS parameters and the QoS parameters required for a particular network technology [Cam97, HILY96, CCH94]. This is not to say that such tight coupling does not adhere to the separation principle, but such lower-layer dependencies are not in keeping with the IP philosophy.

The disadvantage with the lack of integration with any particular network technology is that it means we need a very general network abstraction to support R7 and R8. Also, we need mechanisms for mapping QoS parameters between application and network abstractions, and we cannot (directly) exploit any network specific support mechanisms for QoS that might exist in a particular network technology, e.g. such as in ATM[20].

The other principles listed in Table 3.1 match our requirements fairly well. The separation principle and the asynchronous management principle both support R5 and R6. The transparency principle supports R6 and R7. Performance transparency is defined in a

[20] Some see this "lower-layer blindness" as one of the major disadvantages of IP-based applications in general, and some see it as the reason why IP has been so successful.

more prescriptive manner than the other principles in [CAH96], but points to [SRD84] and [CT90] (amongst others) as guidelines to dealing with design and engineering issues related to performance. This mainly concerns the use of:

- **application layer framing (ALF):** designing ADUs as the unit of processing with an end-to-end, application-level view, e.g. sending audio packets as audio time-slices that may allow the flow as a whole to tolerate some loss or re-ordering
- **integrated layer processing (ILP):** trying to avoid serial/pipeline operations on ADUs where integrated operations may be possible, e.g. perhaps combined compression and error control

3.3.2 QoS specification

Our interpretation of the QoS specification from [CAH96] is given in Table 3.2.

Specification	Description
flow synchronisation specification	the degree of synchronisation between related flows, e.g. when separate video and audio streams require synchronisation
flow performance specification	a description of flow requirements expressed in terms of QoS parameters, e.g. minimum throughput, maximum jitter, etc.
level-of-service (service-level)	the way in which the flow specification is honoured, e.g. guaranteed service, controlled-load service
QoS management policy	a description of the way that non-conformance to the end-to-end reservation should be handled, e.g. adaptation required by application and media flows
cost of service	the price that the user is willing to pay for certain end-to-end QoS

Table 3.2: Interpretation of QoS specification from [CAH96]

It is clear from the discussion presented already that defining flow performance specification in terms of QoS parameters is important for describing the flow-requirements. Additionally, the service-level required may be important to the individual users and may determine the cost of the service, e.g. guaranteed-load is a "better" service than controlled-load so would cost more. However, we argue that the service-level should not be specified by the application (in support of R8). The application should be prepared to be more flexible in its adaptation capability [DHT95], leaving service-level selection to the user. Three reasons for this are:

1. the service-level may determine the cost of the service [Kel97] and users usually wish to control how much they pay
2. network heterogeneity, lack of resource reservation or network element failure may mean that a particular service-level is not available at a given time at a given point in the network
3. new, additional service-level definitions may be introduced that are more suitable (in terms of functionality or cost) for use in a given situation, e.g. **adaptive** service-level [LLB97, CCH96]

Additionally, if cost-based feedback is available from the network, then cost may not have to be specified separately and could be treated as a QoS parameter. Although not related to the performance of the flow it acts as a defining constraint in the same way as, say, defining minimum data rate or maximum jitter for a flow. However, the value of making this explicit as in [CAH96] is that it highlights the importance of cost as a feedback control mechanism in future services [SCEH96].

When guaranteeing end-to-end QoS, there may be requirements for flow synchronisation between different media that have temporal relationships, e.g. audio that accompanies a video stream. However, the way in which the synchronisation is handled may not only be application-specific, but also depend on the users. For example, video and audio streams that are part of a remote teaching activity may require different synchronisation depending on the subject being taught and the type of teaching. If the subject being taught is a foreign language, say French, then good synchronisation and quality for both audio and video is required so that lip synchronisation with sound can be seen by the student [HS97]. When the lecture subject is History of Art, the requirements may be for sufficient data rate to transfer slides containing full-colour, high quality art images relatively quickly (e.g. in the order of a few seconds), but with telephone quality audio, and looser synchronisation between audio and video. So, while the formulation of descriptions for flow synchronisation are valuable, the way that they affect adaptability of an application will depend on application-specific requirements and user preferences. There is also a possibility that these requirements change during the operation of a particular application instance. For example, while the remote instructor delivers the History of Art lecture, higher delay and jitter is tolerable on the audio and video than during the question and answer session at the end of the lecture. For our adaptability requirements we do not need to have mechanisms to directly support synchronisation, though any mechanisms we do propose should not constrain the application from applying synchronisation constraints.

The argument presented for flow synchronisation specification (above) also applies to the QoS management policy. This will be subject to user preferences and application specific behaviour. [CAH96] suggests that the specification of QoS management policy should be made clear before the application starts operating. This would certainly be of value to the network for controlled allocation of resources, and makes sense in the context of trying to *assure* end-to-end to QoS. However, in our consideration of *dynamic adaptability*, the use of the application typically requires interaction from the user in order to determine its adaptation requirements. The user preferences may not be known until after the application is running, or may change during operation of the application (viz. the History of Art lecture example). [CAH96] states that QoS management policy:

> *... captures the degree of QoS adaptation (continuous or discrete) that the flow can tolerate and the scaling actions to be taken in the event of violations to the contracted QoS [CCH96]*

In our work, we chose to make a separation between what "*the flow can tolerate*" and the "*scaling actions to be taken*". We argue that the former is a property of the media and the latter is an application-specific requirement that includes interaction with the user. Flow performance specifications can be used to indicate the flow-requirements that are possible for a flow and can be determined by the application designer. The action to be taken on fluctuations ("*violations*") of QoS is a dynamic adaptation decision and cannot be determined by the application designer *a priori*. It is the difference between the application designer saying, "I know what is sensible for the flow" and the user saying, "I know what is sensible for the application to do for me". Ultimately the application's functional constraints have the final say on which flow-requirement(s) is (are) functionally *possible*, but this should not dictate *how* the user would like the application to behave, i.e. how adaptation should take place. For example, the different requirements in the remote teaching scenario for the audio and video flows during the main part of the lecture and then during the question and answer session at the end of the lecture.

So, the flow performance specification, service-level specification and cost of service specification support R3, but the way in which the QoS management policy is finally determined cannot be specified exactly before run-time.

3.3.3 QoS mechanisms

In [CAH96], QoS mechanisms are listed in three categories:

1. **QoS provisioning mechanisms:** providing QoS mapping of QoS flow specifications at different system levels; admission testing to help determine the ability of a system to accept a flow; resource reservation protocols for signalling end-system and network system elements to arrange allocation of the resources required for the flow
2. **QoS control mechanisms:** deal with control of the transmitted media flow and involve the functions of flow shaping, flow scheduling, flow policing, end-to-end flow control and flow synchronisation
3. **QoS management mechanisms:** allow monitoring of system QoS parameter values to provide information for the QoS maintenance mechanism; QoS maintenance to ensure that the required QoS is achieved throughout operation; QoS degradation notifications in case of lower-level network failures; QoS signalling to allow users to specify sampling and notification intervals; QoS scalability including flow filtering and flow adaptation

Many of these mechanisms are for setting up and maintaining end-to-end QoS, so we extract only the ones of interest to our dynamic adaptation considerations.

QoS mapping is required to support R1 and R2. The representation of information should lend itself to a sensible mapping that hides the complexity of the underlying model from the user as well as allowing representation of the required information about the application flows (in support of R7). QoS admission testing (admission control) is effectively a mechanism to allow the application to assess if the network can support a particular QoS requirement for a flow in a static adaptation scenario. This again is mainly a resource reservation issue, and does not concern us for our run-time, dynamic adaptation considerations. However, success or failure notifications from an admission test act as control information for any static adaptation or may offer a starting point (the first flow-requirement) for our dynamically adaptable application. (Indeed, the value of using admission control for an Integrated Services Internet has been questioned [She95].) Example schemes for admission control are presented in [BFMM94] and [JDSZ95].

Control of the flow in our dynamic adaptation scenario covers only aspects of flow shaping, flow policing and end-to-end flow control. If we consider that the application mode change results in the change of the flow-requirement, this may be either as a result of QoS changes in the network or QoS changes at the receiver. Where the QoS changes are in the network, the change of flow-requirement acts as a form of self-policing that is shaping[21] (changing) the flow in order to match the flow characteristics to those in the network. Where the QoS change is at the receiver, the change in flow-requirement is effectively an end-to-end flow control. The application *may* need to know and care which of these two is actually happening, but the flow itself does not – it simply responds to the application's flow-requirement change decision.

The situation is complicated further when multicast is used. Application-level signalling may be required in order for the decision process to be finalised, as senders and receivers may need to synchronise flow-requirement changes and/or application mode changes [KHC98].

Ultimately, the application's behaviour dictates how these mechanisms are used and interact with each other. We have already discussed how flow synchronisation is considered to be an application-level issue (Section 3.3.2). Part of the adaptation decision may be to assess the synchronisation requirements between any new flow-requirements for currently active flows, which can only be done on a per-application instance basis. Flow scheduling and flow policing may be achieved through co-operation between the application, end-system and the network. Scheduling mechanisms in the end-system are important to ensure that the correct end-system resources are available to allow flow ADUs to be transmitted at the correct intervals (e.g. [LMM93]). If the packet scheduling is also maintained within the network (say as part of a resource reservation mechanism or a class-based scheme, e.g. [FJ95, WGCJF95]), then the flow is likely to remain fairly stable. In the absence of any such mechanisms, the QoS experienced by the flow fluctuates and adaptation is required. So, for dynamic adaptation purposes, we are not concerned with the existence of any scheduling and policing mechanisms in particular, just the observable effects of their presence or lack of presence. For example, there may be a cost associated with the adaptation process in the end-system, but in many cases this is only a significant issue to server end-systems and client end-systems with high-end use [SW97, L+96]. In situations where evaluation of such a cost is significant, general

---

[21] This is not actually the normal sense of how shaping and policing functions operate. They are normally considered to be functions within the network rather than the end-system.

modelling techniques are still maturing [MK97]. Indeed, [YL95] and [FPM95] suggest that the application could be engineered to be adaptable to the host load itself[22].

QoS monitoring and QoS notifications are closely linked for supporting dynamic adaptation. We have said that we can think of the flows as consisting of well-defined flow-requirements that are a function of the QoS parameters for the flow(s) of the application. A QoS evaluation function is required to sample the values of the QoS parameters for the flows at (application-defined) intervals and issue a report of how a flow's requirements are being met. [CAH96] talks of a QoS *degradation* notification only, but we will generalise (to support R4) and say that the QoS evaluation function issues reports that notify QoS *changes*, indicating the suitability of the network QoS to support a particular application flow-requirement. A QoS change report generated by a QoS evaluation function must be in a simple yet informative construction in order to support R1, R2 and R7.

3.4 Flow-requirement modelling

We have argued that an essential part of an adaptation mechanism is the ability to map QoS requirements (between user, application and network), and to allow monitoring of QoS parameter values. We now examine each of these in turn.

3.4.1 QoS mapping

QoS mapping allows the mapping of QoS requirements from user to application and from application to the network (and end-system). The QoS requirements from the user are expressed as user preferences. The mapping of these is an application-specific issue (see Section 3.3.2) and is not considered in detail in this work. We are concerned with the mapping between the application and the network (in both directions).

The application must specify the flow performance requirements. These are typically defined in terms of QoS parameters, such as those presented in Table 2.1. The mapping should be as simple as possible but without losing any context and keeping the amount of semantic information to a minimum. The generally accepted technique is to specify per-flow traffic characteristics plus a service-level. The service-level is an enumeration, each value of which express well-known semantics, so the mapping is simple as long as the

[22] Host load can often be used as a rough indicator for competition for end-system resources, and its value is usually accessible on general use end-systems.

network is aware of the service-level semantics. For traffic characteristics, the form of specification in general use is the token bucket [RFC1363, RFC2215]. This is because it was believed that such a traffic characteristic presents an easy modelling technique, which, with an on-off traffic source, provides a worst-case scenario for a burst of traffic from that source. An analysis of such traffic sources in [Oec97] states that this belief[23] is not true. However, [Oec97] points out that it is not yet possible to determine the actual worst-case traffic pattern.

So, while the token bucket still remains in common use and has been adopted by the INTSERV WG, any mapping of flow specification that we use should not attempt to rely heavily on that model.

3.4.2 QoS parameter value monitoring

In Section 2.5, we looked at the problems associated with taking measurements from the network in terms of noise and timeliness. We said that we need a way to estimate accurately the value of a QoS parameter.

We will need to apply our estimation mechanism to many QoS parameter values for each of many flows for a single application, and many applications may exist on a single end-system. So, for our estimation mechanism we must choose methods that:

- are computationally inexpensive to implement in software, so they can be used on hardware platforms containing general purpose processors
- keep the amount of state required to a minimum (e.g. number of previous measurements)

Techniques with such properties for estimation based on the use of an **exponentially weighted moving average (EMWA)** estimator are widely used with communication protocols (e.g. [Jac98, Kes91, KMR93, SEFJ97]). EWMA is computationally inexpensive and requires little state information. A modification of the EMWA allows practicable adaptive estimation using a heuristic method [KMR93] or by the use of fuzzy logic techniques [Kes91, KK92]. The adaptive mechanisms add fixed computational overhead to the basic EWMA, but make the EMWA more reactive to system perturbation, whilst retaining the ability to filter noise.

[23] ... or "myth" as stated in the title of [Oec97].

3.5 Supporting adaptability in applications

We chose to split the current research in supporting dynamic adaptability for applications into four main areas:

1. operating system support
2. distributed systems support
3. media flow scaling and filtering
4. support for dynamic QoS assessment for flows

We examine each of these in turn.

3.5.1 Operating system support

As we have already stated, we do not consider the specifics of the end-system or operating system within our work. However this remains an important issue for server systems and clients that have high-end requirements [SW97, L+96]. There are arguments that the application should be dynamically adaptable to CPU load [KH97, YL95]. End-system considerations are certainly of value where hosts have limited resources, e.g. limited power capability on mobile hosts. (We present a simple example of how our proposed solution could be integrated with such support in Section 6.5.)

3.5.2 Distributed systems support

Much of the activity in addressing QoS deals with support for end-to-end QoS guarantees [CAH96]. As such, the distributed systems support for adaptability mainly addresses static adaptation (perhaps allowing dynamic re-negotiation of QoS during operation). Other works consider adaptive ability in networks to cope with mobility, e.g. [Cam97, BCDRF97, DWFB97]. In our context, the dynamic adaptability decision is made within each application instance. So our need in a distributed system (in the first instance) is simply to transport the QoS information used for the decision-making (and perhaps the adaptation decision information itself) to other parts of the application if required. In a general distributed application scenario, there may be additional requirements to co-ordinate the adaptation activities with other mechanisms such as resource reservation, media scaling and filtering, and other application-specific activities.

There is currently much research into information distribution and application co-ordination mechanisms for distributed systems, for example, [HCBO98, GHMY96, BCDRF97, DWFB97]. [HCBO98] is an architecture proposed by the IETF MMUSIC (Multiparty Multimedia Session Control) WG[24] as a framework for multimedia conferencing applications. It does not explicitly address adaptability, considering it to be an application-level issue. However, it does incorporate the elements of the INTSERV WG, allowing both loosely-coupled (no floor control as in [BTW94]) and tightly-coupled (strict floor-control) conferences. [GHMY96] also has an architecture based around the Internet protocols but is geared more towards tightly-coupled co-ordination with static adaptation. [BCDRF97] and [DWFB97] use middleware support (ANSA [ANSA] and CORBA [CORBA], respectively) for an integrated approach based on static adaptation. CORBA is also used in [Cam97], which seeks to provide a QoS control architecture. [Cam97] also supports the notion of static and active adaptation but the algorithms for the decision making are executed at the network level and the transport level, and not by the application, the application having submitted its adaptation capability (effectively its possible flow-requirements) to the network.

Our dynamic adaptation requirements are for per-instance adaptation. So, our adaptation QoS summarisation function must be capable of working in (logical) isolation to produce **QoS summaries** (**QoSReports** in our model) for a particular instance of an application on an end-system. This means that the **QoS summarisation function** (the **QoSEngine** in our model) does not necessarily require distributed systems support. However, the **adaptation decision function** (the **AAF** in our model) might need such support in order to gather additional (application-specific) information to make an informed decision on how it should actually change mode and/or flow-requirement. Effectively, our requirements are for QoS summaries that reflects a local, per-flow view of QoS, delivered in a form that the application can easily use in a more distributed manner, if required.

3.5.3 Media flow scaling and filtering

We examine the way in which it is currently possible for flows to be adapted. The kind of adaptation possible can be categorised as:

- **rate adaptive:** the flow (and application) can change its construction in order to vary the throughput required for the flow, i.e. ADU delivery rate

[24] http://www.ietf.org/html.charters/mmusic-charter.html

- **delay adaptive:** the flow (and application) has a degree of flexibility in its tolerable end-to-end delay, i.e. delayed ADU delivery
- **jitter adaptive:** the flow (and application) has a degree of flexibility in the variation of its end-to-end delay, i.e. variable ADU inter-arrival time (some effects may be due to ADU mis-sequencing in transit)
- **error adaptive:** the flow has the capability of tolerating different amount of errors in ADUs, i.e. lost ADUs or ADUs received with non-correctable errors

Rate adaptation is possible through use of multiple streams of different rates, media-scaling [D+93] (continuous [BTW94] or discrete using hierarchical or layered coding [Sha92]), filtering [YGHS96] or transcoding [AMZ96]. Using multiple streams is the easiest approach to engineering in the transmitter. However, this requires the flow to be available in several different encodings, is not very efficient and may require complex synchronisation if there are many flows and users involved. Also, if two users require different rates but share the same path, then both streams must traverse that path.

In the hierarchical or layered coding approach a single media flow is split into several individual sub-flows, each of which adds additional detail or quality to a base sub-flow (e.g. [AMV96, CCH96, MJV96]). This has better scaling properties for multicast. For example, audio or video can have higher frequency components in separate sub-flows: an application must read the base sub-flow, but can select the amount of quality (number of additional sub-flows) that is required. In fact, each of the streams can be treated as a separate flow with different QoS in the network [CCH96]. Such an approach is fairly flexible and efficient with resources but may require additional flow synchronisation, and/or better dynamic delay, jitter and error adaptability in the application, especially if the separate sub-flows do not have the same QoS. Another possibility may to be provide resource reservation for each sub-flow [CCH96]. For our dynamic adaptation considerations, synchronisation is not a concern and each sub-flow can be treated either as an individual flow or as flow-requirements for a single flow.

Filtering techniques transform the bit-stream of the flow in some way. In fact, filtering is a general term describing flow transformation. Filters can be used to perform hierarchical coding, frame-dropping, transcoding (codec filter) and splitting/mixing of streams [YGHD96]. One common method of filtering is transcoding [AMZ95, YGHS96], changing the stream encoding to that of another format. Other types of filtering are possible. The activation of a filter might be handled by the network, or an application-level gateway, or at the source – again an application-level decision.

Errors can be handled through forward error correction (FEC) or retransmission. However, FEC can be computationally expensive, and end-to-end re-transmission is not always suitable for real-time media. So, error adaptability is often required at the application level. Schemes also exist for coping with loss by using redundant encoding [RFC2198, BV96, HSHW95]. The exact nature of the redundancy used depends on the amount of lost or erroneous ADUs. A flow may also change its construction to have smaller ADU sizes to cope with lost ADUs.

Delay and jitter adaptability is typically handled by use of elastic buffering. However the delay and jitter constraints must be within acceptable bounds for the application and the flow. Note that rate, delay and jitter adaptability are marked (at the beginning of this sub-section) as being supportable by *both* the media flow *and* the application. Remember that even if the flow construction supports adaptation capability, it must make sense for the application and its user (e.g. the two different lecture scenarios in Section 3.3.2). We note that [BV96] proposes an encoding scheme incorporating redundancy that allows combined rate, error and jitter control using distinct and well-defined flow-requirements for the audio flow.

So, we see that flows and applications can be adaptive, but co-ordination is needed between the application and the network. QoS information from the network is required in order to make decisions about dynamic adaptability. However, the application can only co-ordinate dynamic adaptation if it has a sensible way of assessing network QoS.

3.5.4 Support for dynamic QoS assessment for flows

Dynamic adaptability, generally, is not well supported in applications. As noted in [PHKS97]:

> *... a user can configure the audio tool to match the characteristics of a particular network. Often however, the user does not have the necessary knowledge to perform this task well ...*

Although the comments are specifically for an audio tool, they are generally applicable. Applications show static adaptability and have some tolerance to rate, delay and jitter variations, but, in general, rely on the user for correct tuning to operational conditions.

In the Section 3.5.2, we said that a dynamic QoS assessment function works in isolation per end-system, per application instance. The fact that the mechanism can work in isolation should not really be a surprise if we consider flow control and congestion control algorithms for TCP, for example. Although each instance of the algorithm works in isolation, the design of the system and the way it acts on the information it receives ensures a distributed effect [MSMO97]. (Such TCP-like behaviour is also possible for multicast [VRC98].) However, unlike TCP, the final decision regarding a flow-requirement change is left to the application (in co-operation with the user) not to the protocol software entity. So, the stability of the application (and perhaps at least part of the network) with respect to the application flow-requirement change is ultimately in the hands of the application instance itself – "the application knows best". The trade-off here is between flexibility with respect to the user and application, against the possible risk of application or (partial) network instability.

Applications use information from the network (e.g. in congestion control), or from remote receivers (e.g. in flow control) to change their behaviour, relying on a feedback-based control system in order to gain stability. The purpose of the QoS summary (QoSReports in our model) should be to provide the application with feedback about the compatibility of the network QoS with the application's flow-requirements. Indeed, we have seen that there may be other factors observable at the host (such as CPU load), not just network QoS, that determines flow-requirement changes. The application must take the flow-specific QoS information along with other application-specific input to make the final decision about changing its flow-requirement.

To the best of our knowledge, there is currently no general, automatic mechanism publicly documented that allows assessment of the network QoS from per-flow QoS parameter measurements, to support dynamic adaptation in the way we have described.

3.6 Chapter summary

Applications with QoS sensitive data flows need to be dynamically adaptive and require supporting mechanisms to enable such adaptability.

There are two classes of adaptability, **static** and **dynamic**. Applications that are statically adaptable have more that one mode in which they can operate. Such applications choose a mode of operation (or the user chooses a mode of operation for the application), and assume that the network will support the appropriate flow-requirements for the period of operation. Dynamically adaptable applications may use similar mechanisms to those of statically adaptive applications in order to start operation, but during operation they can change flow-requirement as required to deal with QoS fluctuations in the network. (Section 3.1.)

The requirements for supporting dynamically adaptive applications are; QoS interface integration enabling communication between network and application; QoS mapping between application and network; a QoS specification to describe flow requirements; the ability to make QoS compatibility assessment based on measured network QoS data; QoS information separation and QoS mechanism separation so that the application is not unduly constrained in design or operation; QoS encapsulation to hide the complexity of any network technology or QoS information mechanisms; and support for QoS heterogeneity. (Section 3.2)

There has been much consideration of QoS architectures for *ensuring* end-to-end QoS. Support for dynamic adaptability (as described here) is not fully considered in such architectures. Although we find that a general QoS architecture provides support for many of the QoS requirements listed above, we note that there are some key differences. These are mainly with respect to the principle of integration, support for heterogeneity in QoS service-levels and QoS management policy, in particular the separation of concerns between which functions are left to the application and which are left to the QoS architecture. We note that support for making dynamic adaptation decisions based on QoS assessments is not supported. (Section 3.3.)

A key aspect of enabling dynamic adaptation is modelling the flow-requirements. The mapping should allow per-flow descriptions, based on QoS parameter values, but should not be constrained by any particular traffic characteristics. We also need estimation mechanisms that can remove noise from QoS parameter measurements. Any mechanisms we use must be computationally inexpensive, and cope with the wide heterogeneity in the Internet traffic characteristics. (Section 3.4.)

A dynamic *QoS assessment* mechanism does not require specific support from the operating system or a distributed systems platform, but system resources (such as host load) may need to be taken into account in making decisions for changing flow-requirement. So, the application as a whole may require specific support from either or both of the operating system or a distributed systems platform for making adaptation decisions. Techniques exist for adapting media flows, but no mechanism exists for supporting dynamic QoS assessments and dynamic adaptation in the way we have described. (Section 3.5.)

4. Dynamic QoS assessments

In Section 3.6, we ended with the following conclusion:

> *... no mechanism exists for supporting dynamic QoS assessments and dynamic adaptation in the way we have described.*

In this Chapter, we make our first contribution to addressing this problem. We define the **QoSSpace** and **QoSReports**. The QoSSpace is a model to allow interaction between the network and the application. It has a well-defined interface and takes simple descriptions of the flow-requirements – **QoSRegions** – which describe the requirements of the flow. The QoSSpace and QoSRegions are defined in terms of **QoSParam** values. QoSParams have a name and value. The flows and the network are said to exist in QoSSpace. QoSReports are issued by the QoSEngine and quantify how well the QoSRegions match the current network QoS.

We start by first clarifying the problem, based on the discussion in Chapter 2 and Chapter 3, and our requirements R1 to R8 listed in Section 3.2. This is followed by a description of the QoSSpace and how QoSRegions (the model of the application's flow-requirements) are defined. We then describe the mapping of the QoSRegions and the network QoS, **NetQoSRegion**, into the QoSSpace and look at the dynamics of our model. We discuss the QoS related information required by both the QoSSpace and by the application, and define the interface between them.

4.1 The problem

A mechanism is required that can give an indication of the ability of the network to support any of a number of flow-requirements that the application might use. The application modes may be functions of network QoS and other application-specific information, so our mechanism cannot *make* the decision for the change of flow-requirement for the application, but offers summaries of QoS information – **QoSReports** – which represents the compatibility between flow-requirement and network QoS. In general we are not aware of the following application-specific details:

- the way in which information from the current QoSReport will be used with information from previous QoSReports in order to make a flow-requirement change decision
- the way in which QoSReports must be evaluated with other application-specific information or user preferences in order to make a flow-requirement change decision

For example, there may be a statistical or temporal sense in which the information in our QoSReports has meaning to the application with respect to its current mode and flow-requirement(s). This can only be assessed by the application. So, we chose to separate the flow information from other application information (in support of R5) and say that the application flow-requirements will be expressed separately as **QoSRegions**; information that is specific only to the QoS requirements and constraints of the individual application traffic flows. For example, in an audio conference, each audio application *instance* is concerned only with QoS experienced by its flow. However, the conferencing application as a *whole* may require all the separate instances of the audio application to synchronise on the highest quality audio encoding that is useable by all the conference participants. As we do not know how the application will use the QoSReports, the QoSEngine will simply issue QoSReports that are based on an evaluation of the *instantaneous* state of the network – a snapshot in time[25].

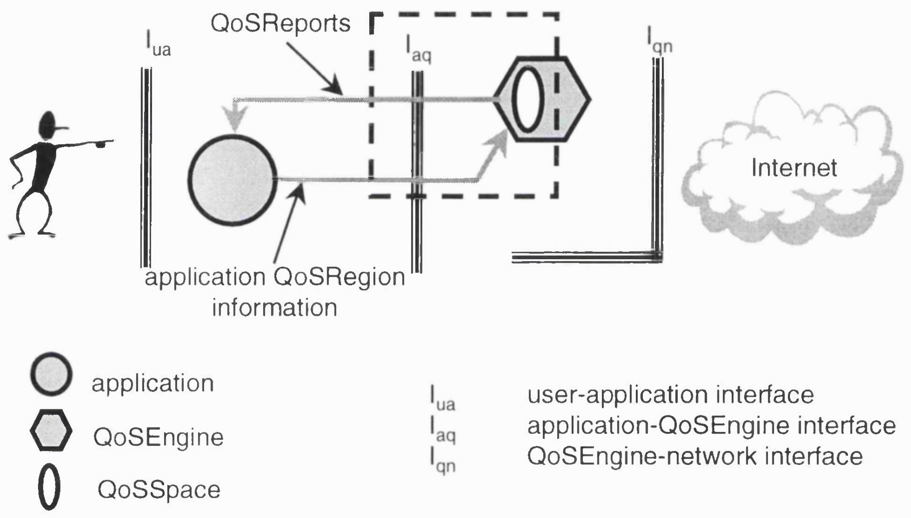
Our problem, then, is to devise a model that will report the suitability of the network to support QoSRegions defined for an application's flows. We have devised such a model that represents a multi-dimensional space in which the application flows conceptually exist and into which the network QoS is mapped. This is the **QoSSpace**. In Figure 4.1, we

[25] We see later when we discuss the **application adaptation function (AAF)** in Chapter 6, that in fact we may need to perform some "application-level smoothing in time" in order to control adaptation behaviour.

show a simplified version of Figure 1.3, highlighting the area of work considered in this Chapter (dashed box). So, in this Chapter, we describe:

- the QoSSpace and QoSParams
- the QoSRegions for a flow
- the way in which the QoSEngine produces QoSReports
- the interface I_{aq} (in both directions)

Note that the AAF (application adaptation function) has been removed – we return to this in Chapter 6.



I_{ua}
QoSReports
I_{aq}
I_{qn}
Internet
application QoSRegion information
application
QoSEngine
QoSSpace
I_{ua} user-application interface
I_{aq} application-QoSEngine interface
I_{qn} QoSEngine-network interface

Figure 4.1: The QoSSpace and the Interface I_{aq}

We start by clarifying the nature of the operation of our model.

4.2 Control systems and adaptation

Our model (Figure 1.3) has some similarities to that of a traditional control system process. However, it has significant differences, both in its aims and its function, which mean that the use of traditional control theory analysis and evaluation are not suitable. Many adaptive mechanisms (e.g. congestion control) rely on the use of a control systems approach to adjust their flows. A general control system is depicted in Figure 4.2.

In a traditional control system, the aim of the *controller* is to generate *control actions* that keep the *system* at a set-point (a defined operating point for the *system*). The *transfer function* uses *measurements* of specific parameters from the *output* of the *system* to produce *feedback* for the *controller*. The *controller* uses the *feedback* information with other *inputs* to decide the *control action* it should take in order to maintain the required *system* behaviour, i.e. maintain the *set-point.*

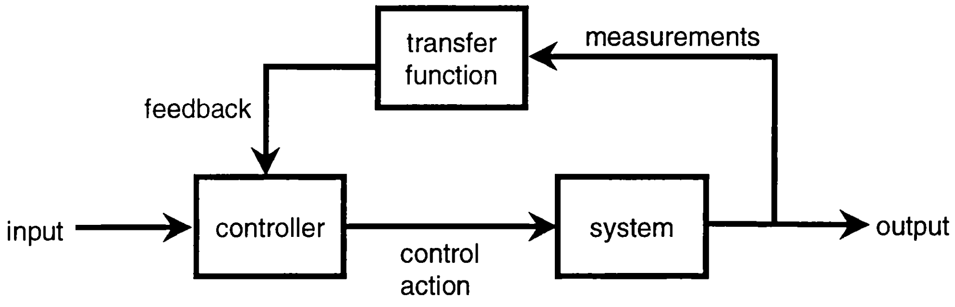


Figure 4.2: A schematic diagram of a general control system

If we compare Figure 4.2 with our model in Figure 1.3, we can identify similarities:

- the *system* is the application and the *output* is the flow as it traverses the network
- the *measurements* from the *output* are a set of QoS parameter measurements for the flow
- the *transfer function* is analogous to the QoSEngine which generates the QoSReports as a summary of *system* behaviour
- the *controller* is analogous to the AAF (application adaptation function) and the *control action* will be to select a suitable QoSRegion, which may affect the *system* (the application) and the *output* (the flow)

In a traditional control system, the *transfer function* tries to model the input/output characteristics of the *system*, and the *feedback* is compared with a reference set-point (typically an *input* to the controller) to determine the *control action*. However, our dynamic adaptation scenario differs from the traditional control system in three important ways:

1. the QoSEngine does not model the *system* input/output but provides a summary of the relative **compatibility** of the network QoS offered to the flow and the possible QoSRegions
2. the generation of the QoSReports can be controlled through manipulation of the definition of the QoSRegions, which is analogous to modifying the *transfer function*
3. the behaviour of the AAF may be modified by user preferences or inputs from other application specific sources, i.e. the *controller* is adaptable

So, overall, while we do have a feedback control process, the *measurements* are processed in a different manner to that of a traditional control system, and so the *feedback* is of a different nature. A traditional *transfer function* may be identified by system analysis or by empirical methods, but the QoSSpace is always the same abstraction (modified only by the QoSRegions and QoSParams defined by each application), and the QoSEngine always performs the same function (a compatibility test).

Essentially, the aim of our model is not the same as that for a traditional control system; instead of attempting to **maintain** a set-point, our model seeks to allow the application to **select** a QoSRegion.

4.3 QoSSpace and QoSParams

The **QoSSpace** is a multi-dimensional space in which flows conceptually exist, and into which the network QoS is mapped. Flow-requirements are represented by **QoSRegions**, where each flow may exist in one of a set of QoSRegions. The dimensions of the QoSSpace are a set of **QoSParams**. This is depicted in Figure 4.4, which shows only three QoSParams, but any number of QoSParams are possible.

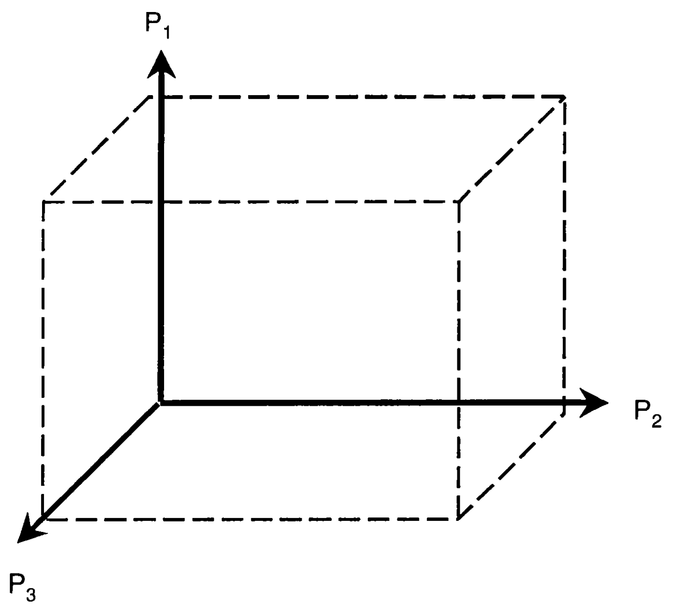


Figure 4.3: A QoSSpace defined in terms of three QoSParams

The QoSParams are variables that are representations of real QoS parameters, such as throughput, delay, jitter, etc, and have the same units as the QoS parameter they represent. The QoSParam is an estimate of the current value of the QoS parameter, based on measurements of that QoS parameter. QoSParams are chosen to suit the application, i.e. the dimensions of the QoSSpace are application-specific (in support of R3). We consider the timescales of operation to be application-specific (in support of R5 and R6), and the QoSSpace issues a QoSReport which is effectively a snapshot in time. However, the application must ensure that measurements of QoS parameters are being taken as least as often as QoSReports need to be generated. The interval over which measurements are taken and QoSReports are generated will be application-specific. The network QoS is evaluated and mapped into the QoSParam space by some mechanism (we describe a suitable mechanism in Chapter 5). We choose to separate the abstraction of the QoSSpace from any mechanism for performing the mapping of the network QoS in support of R5 and R6. We describe the application interface to the QoSSpace in Section 4.7. The simplicity of the QoSSpace supports R7.

When we talk about a QoSParam we identify it simply by the QoS parameter it represents, e.g. delay. The QoSSpace abstraction does not require knowledge of the semantics of the QoSParams, other than:

- that they are REAL numbers
- the QoSParam values are on a linear scale
- that relational operators (=, <, >) make sense, semantically, for the QoSParam values

These three simple rules are necessary and sufficient. They are necessary in order to allow the comparison of two values of a QoSParam in a meaningful way; and they are sufficient as any other QoSParam semantics (e.g. "high values for a parameter are better than low values") are application specific. Use of REAL numbers rather than INTEGER values allows a more convenient notation when using large numbers or high order units for QoSParam values, e.g. 64.1Kb/s instead of 64100Kb/s. The QoSSpace is not concerned with the units of any QoSParam but the application is. The definition and number of QoSParams in each QoSSpace is application-specific.

4.4 QoSRegions

We have chosen to think of the flows as having flow-requirements. These flow-requirements are defined in terms of QoSParams by specifying the operating/performance limits required for a particular flow-requirement. Each of these flow-requirements is called a **QoSRegion** for the flow, and is a region in QoSSpace in which the flow can operate. As an example, we may define a QoSRegion for a fictitious flow in terms of the QoSParams P_1, P_2 and P_3. We use simple boundaries, which we call ***_lo*** and ***_hi***:

$$qr1 = \{\langle P_1, p_1\_lo, p_1\_hi\rangle, \langle P_2, p_2\_lo, p_2\_hi\rangle, \langle P_3, p_3\_lo, p_3\_hi\rangle\}$$

This statement identifies a QoSRegion called *qr1*, which is defined in terms of the QoSParams P_1, P_2 and P_3. *qr1* consists of a set of tuples, each of which has the structure:

$$\langle id, \_lo, \_hi\rangle$$

where:

id	is a name identifying the QoSParam
_lo	is the low threshold value of the QoSParam
_hi	is the high threshold value of the QoSParam

For simplicity, we have chosen to make QoSRegion boundaries rectilinear. *qr1* is depicted in Figure 4.4.

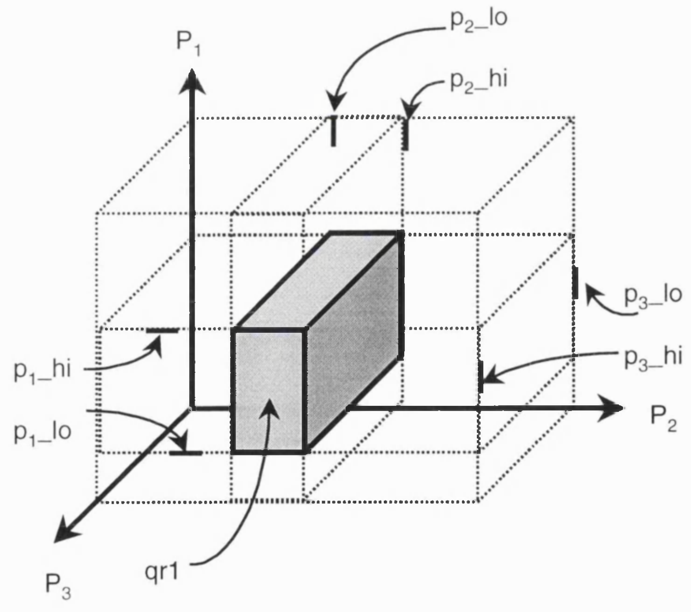


Figure 4.4: An example of a general QoSRegion definition with three QoSParams

In general, a flow may have many QoSRegions, one for each of its flow-requirements. For any set of QoSRegions for a flow:

- the number of QoSParam tuples specified need not be the same for all QoSRegions
- for any QoSParam tuple in the QoSRegion definition, either one of the *_hi* or *_lo* thresholds may be left undefined, as required, but at least one of them must be present
- QoSRegions may overlap
- some points in the QoSSpace may not belong to any QoSRegion

For N QoSParams, the QoSRegion is defined simply as:

$$qr = \{\langle P_1, p_1\_lo, p_1\_hi\rangle, \langle P_2, p_2\_lo, p_2\_hi\rangle, \ldots \langle P_N, p_N\_lo, p_N\_hi\rangle\} \qquad \textbf{(4.1)}$$

As an example, consider the use of throughput, delay and jitter in place of P_1, P_2 and P_3, as required for an audio flow. One QoSRegion may be defined as:

$$pcm = \{\langle throughput, 64, -\rangle, \langle delay, -, 500\rangle, \langle jitter, -, 500\rangle\}$$

This says that the QoSRegion *pcm*, requires a minimum of 64Kb/s throughput, and can tolerate a maximum of 500ms delay and a maximum of 500ms jitter. The QoSRegion *pcm* is depicted in Figure 4.5.

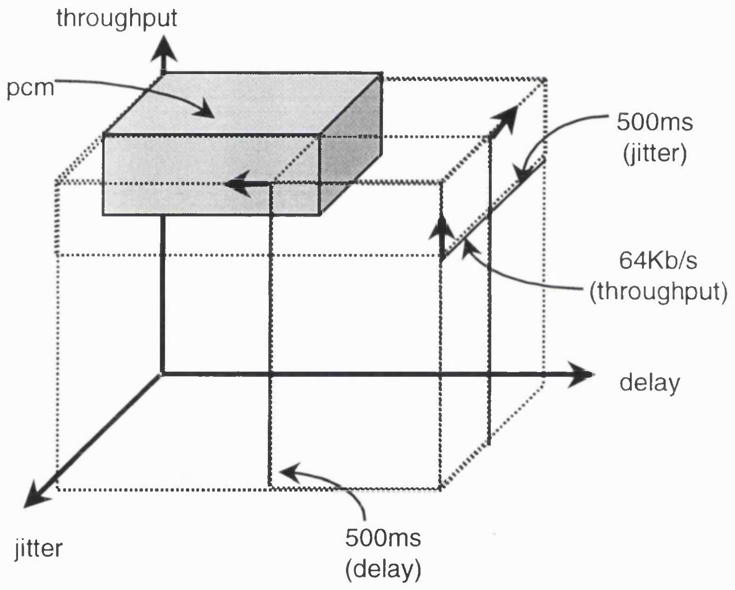


Figure 4.5: An example QoSRegion definition for an audio flow

For this same audio flow, other encoding schemes could be possible, and so other QoSRegions would be defined. The definition of the QoSRegions for a flow acts as a QoS specification for the application (in support of R3), and also serves as a QoS mapping from application view to a network view through the QoSSpace (in support of R2).

Note that the model is not concerned with the semantics of the QoSParams or any relationships between them – this is left for the application to control. Also, different QoSParams may exhibit strong correlation from their underlying QoS parameters (e.g. on some networks, delay and throughput may be related – as delay goes up throughput goes down), and so it may be possible to reduce the number of dimensions. This, again, is an application-level issue. The application may choose the complexity of the definition of the

QoSRegions, as required. However, the QoSSpace must be defined by the set of QoSParams given by the union of all the QoSParam tuples for all the QoSRegions for that flow. The simplicity of the QoSRegion definition supports R7 and R8.

4.5 NetQoSRegion

We have a mechanism for describing the application flow-requirements – QoSRegions. We now need a mapping of the network QoS into the QoSSpace. We achieve this by using QoSParam values to also describe the network QoS, **NetQoSRegion**. NetQoSRegion is a *region* in QoSSpace indicating the network QoS offered to a flow. The NetQoSRegion supports R2, mapping network QoS to the QoSSpace.

The NetQoSRegion will have the same dimensions as the QoSSpace. Note that this mapping is not *necessarily* the network QoS as seen at a *particular host interface* – it is the QoS that reflects the network resources available to a *particular flow*. However, the application may decide the granularity of the flow. For example, the application may decide that it has only one flow with respect to the QoSSpace, and this is a measure of the network resources available to the application as a whole via a particular host interface. Within that application flow, it may choose to send many traffic streams. There may be other constraints that decide the granularity of a flow, e.g. IPv6 and RSVP have slightly different definitions for a flow. The flow definition is application specific.

A simple interpretation of the NetQoSSpace would be to measure values of the QoS parameters for a flow, and use these directly as values of QoSParams. This would translate to a *point* within the QoSSpace:

$$r = \{q_n\}, \quad \forall\, n = 1 \ldots N, q_n \in P_n$$

i.e. $\{q_n\}$ are the values of the set of QoSParams $\{P_n\}$ that define the QoSSpace. However, we noted in Section 2.5 that delay and noise effects mean that we must transform our measured values of QoS parameters to try and estimate the true state of the network. Associated with this data transformation is an *uncertainty* in our estimation. As the network QoS fluctuates we may have different amounts of uncertainty. If the network is in a steady state, we may have a lesser degree of uncertainty than if the network is currently showing *fluctuations* in the QoS offered. Measures such as standard deviation are normally used to indicate such uncertainty. However the standard deviation may only have meaning and use in a statistical sense when we have some knowledge of a consistent model of our traffic and/or the network. What we actually want is an estimate of the current *variability*, *v_p*, of the QoSParam, i.e. some instantaneous estimate that indicates how much the value of the QoSParam is currently *fluctuating*. We discuss how we estimate a value for *v_p* in Section 4.6.1. In terms of our QoSSpace, we choose the mapping of the NetQoSRegion to be expressed as a region:

$$r = \{\langle P_1, q_1\_lo, q_1\_hi\rangle, \langle P_2, q_2\_lo, q_2\_hi\rangle, \ldots \langle P_N, q_N\_lo, q_N\_hi\rangle\} \qquad \textbf{(4.2)}$$

where the *_lo* and *_hi* thresholds in (4.2) indicate the limits of our estimate of variability for the QoSParam. Note that this is the same format as our expression for the QoSRegions in (4.1). However, the NetQoSRegion in (4.2), must have both a *_lo* and *_hi* component for each QoSParam tuple.

4.6 QoSReports and the region compatibility value (RCV)

We have mechanisms for representing the flow-requirements and the network QoS, the QoSRegion and NetQoSRegion, respectively. We now need a mechanism that can issue QoSReports that contain values that indicate the relative compatibility of the QoSRegions and the NetQoSRegion. In fact, a description of our task is relatively straightforward: we need to find when the region defined by the NetQoSRegion intersects with a region defined by a QoSRegion. If we can measure this intersection, we can offer the application a **region compatibility value (RCV)**, a measure of the how well the current network QoS might support a particular QoSRegion. This compatibility value is a unitless number that is easy to use in other parts of the application. Such a simple mechanism is in support of R1, R2, R3 and R7, as well as R4.

Note that the definition of the QoSRegion in (4.1) and that for the NetQoSRegion in (4.2) suggest that we may be able to treat these state definitions as hyper-volumes. For example, we may choose to use the ratio:

$$\frac{\text{volume of overlap of NetQoSRegion and QoSRegion}}{\text{volume of NetQoSRegion}}$$

to evaluate a RCV for each QoSRegion. However, we choose not to do this. In definitions of QoSRegions, our model allows use of different numbers of QoSParam tuples in defining QoSRegions for the same flow, resulting in different shaped volumes (see Section 4.4). In the evaluation of a volume, relative scaling by multiplication of values of *N* QoSParams may lead to a distortion when some values are particularly high or particularly low. Indeed, we need to process each QoSParam individually, and then offer some sensible summary to the application. So, we must first consider how we process individual QoSParam values.

4.6.1 Parameter compatibility value function (PCVF)

For each QoSParam, we can derive a **parameter compatibility value (PCV)**, that expresses the amount by which the value of a certain QoSParam from the NetQoSRegion falls within the operating region given by a corresponding tuple for a given QoSRegion. So, for a given tuple, T_p, from a QoSRegion, and the corresponding tuple, T_s, from the NetQoSRegion, for the same QoSParam, P:

$$\begin{aligned} T_p &= \langle P, p\_lo, p\_hi \rangle \\ T_s &= \langle P, q\_lo, q\_hi \rangle \\ PCV &= \mathrm{PCVF}(T_s, T_p) \end{aligned} \qquad \textbf{(4.3)}$$

where **PCVF** is the **parameter compatibility value function**. The operation of this function is to assess the following statement:

$$\mathrm{PCVF}(T_s, T_p) = T_s \ \mathrm{WITHIN}\, T_p \qquad \textbf{(4.4)}$$

where WITHIN is a function that evaluates to a single number that quantifies the ratio:

$$\frac{\text{length of intersection of } T_s \text{ and } T_p}{\text{length of } T_s}$$

The description of WITHIN is explained with the help of Figure 4.6. This shows the possible scenarios for evaluating WITHIN when T_s and T_p overlap. I is the length of the intersection of T_s with T_p. It is clear that the omission of either *p_lo* or *p_hi* from T_p (from the QoSRegion) poses no problem.

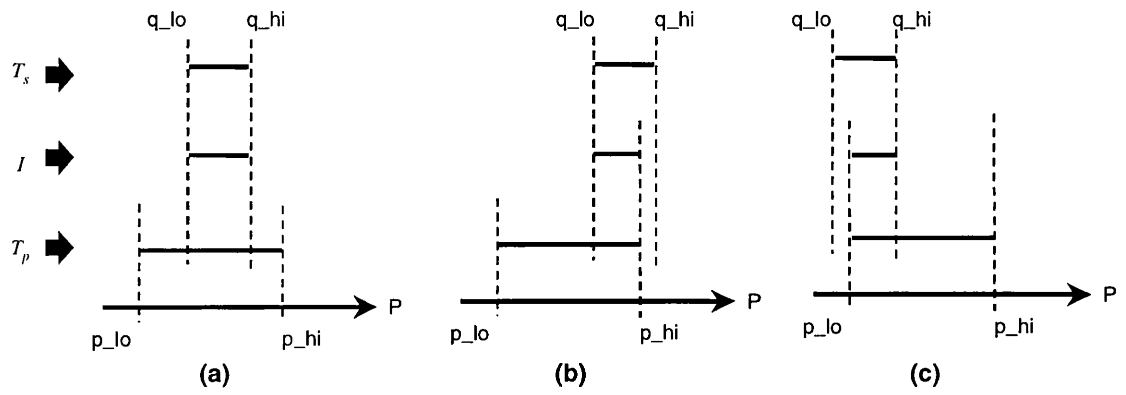


Figure 4.6: Scenarios for the evaluation of WITHIN in the PCVF

The PCVF has a simple algorithm:

$$\begin{aligned} L_q &= q\_hi - q\_lo \\ I &= \text{MIN}(q\_hi, p\_hi) - \text{MAX}(q\_lo, p\_lo) \\ PCV &= \text{MAX}(0, I / L_q) \end{aligned} \qquad \textbf{(4.5)}$$

The MIN and MAX functions in line 2 of (4.5) perform their usual operations, except that if either *p_hi* or *p_lo* are not defined, then *q_hi* or *q_lo* are used, respectively, as required. I takes the range $[-\infty, L_q]$. When there is no intersection, I is negative. We choose that PCV = 0, indicates "no compatibility" between QoSRegion and NetQoSRegion, while PCV = 1 indicates "full compatibility". So, the final line of (4.5) ensures that the range of the PCV is [0, 1]. This normalised value is a uniform, consistent and scaleable way of representing PCVs, supporting R2 and R7. The algorithm for the PCVF is also computationally simple. We see that as the variability of the QoSParam (i.e. the QoS fluctuation) increases, so L_q increases. Unless the fluctuations are completely contained within the region defined by *p_lo* and *p_hi* tuple thresholds (Figure 4.6(a)), as L_q increases, we have decreasing compatibility between the QoSRegion and NetQoSRegion with respect to that particular QoSParam (Figure 4.6(b) and (c)).

We choose to use a simple instantaneous estimate of variability in our QoSParam values. We define *v_p* as the absolute value of the difference between the current and previous values of P, and use this to evaluate *q_lo* and *q_hi*:

$$\begin{aligned} v\_p &= \text{ABS}(p_t - p_{t-1}) \\ q\_lo &= p_t - (v\_p / 2) \\ q\_hi &= p_t + (v\_p / 2) \end{aligned} \qquad \textbf{(4.6)}$$

In *v_p*, we try to embody the measurement of the current *fluctuations* that are present in the measured parameter values. We are uncertain of the statistical properties of the parameter. In general, different QoS parameters exhibit very different statistical properties and the properties for even a single parameter may change rapidly over time. We also need to ensure that *v_p* is simple to evalute and gives a timely reflection of the fluctuations in the measured values. Hence we choose a simple heuristic measure that requires little state and is computationally simple to evaluate.

4.6.2 Examples showing the dynamics of the PCVF

It is important to assess when QoSRegion boundaries are crossed, i.e. when there is a change in PCV which may result in a change of QoSRegion. We examine three scenarios where the value of P is close to a QoSRegion boundary, and so there is a change in the PCV. The three scenarios are:

1. rapid oscillations in P near a QoSRegion boundary
2. a slow change in P across a QoSRegion boundary
3. a large, sudden change in P across a QoSRegion boundary

We do not know the nature of the application and so cannot say if the detection of any of these three is important for the operation of the application. We must offer the application the ability to detect all of them.

In Figure 4.7(a) and (d), we define a simple QoSRegion, *qr3*, with a single tuple, $qr3 = \{\langle P,10,-\rangle\}$ (the region above and including the horizontal line at $P = 10$ in Figure 4.7(a) and (b)). In Figure 4.7(a), let us assume we observe a set of measurements of P that could, for example, represent some instability in P. In Figure 4.7(b), we use a simple threshold test to generate a Boolean indicator if the instantaneous value of P is within the region defined by *qr3*. We see the added value from the PCVF in Figure 4.7(c). The application has a more informative assessment about the possible instability in the values of P due to the PCV degrading from 1.0 over time ($t = 40$ to $t = 60$). Also, the PCV recognises that although there are large fluctuations in P, this does not necessarily mean that *qr3* cannot be supported, but that there is less compatibility between the network QoS and that QoSRegion, and so the PCV does not reach zero.

We see in Figure 4.7(d) that the values of P degrade such that the variability between successive measurements is smaller than in the excited fluctuations in Figure 4.7(a). (Figure 4.7(d) is actually the lower part of the "envelope" for the measurements of P in Figure 4.7(a).) In this case, the behaviour of the Boolean threshold and the PCV in Figure 4.7(e) and Figure 4.7(f), respectively, are very similar. In particular, although we see a general downward trend in the data from the graph in Figure 4.7(d), we do not receive the same degree of information from the PCV as when there are more rapid fluctuations in P. This is a consequence of the method we have chosen to represent variability for our model, in (4.6).

We need to be able to offer the application indications when the QoSRegion is being supported close to a boundary so that the application has forewarning of a possible QoSRegion change. We demonstrate in Section 4.6.4 how the use of another set of boundaries that define a **QoSiRegion** – a QoS intermediate region – allows us to evaluate when the QoSParam is close to a QoSRegion boundary.

Notice the spike at $t = 50$ in the PCV graph Figure 4.7(f). This may seem to be "incorrect" behaviour as we can see that value of P at that time is within the operating region for *qr3*. However, we only know it is "correct" because we can see what happens at $t = 51$. At $t = 50$, all that we can see is that there is a downwards change in the value of P very close to the boundary for the region of operation for *qr3*. We do not know at $t = 50$ what will happen at $t = 51$ so we assume the downwards trend in the values of P will continue. There is no reason why we cannot assume that the value at $t = 50$ will be the *same* as at $t = 51$, but we choose to side with inertia and offer the counter example in Figure 4.8 (discussed below). The application may wish to smooth such spikes if they are reported in the RCV and we discuss this when we describe an example AAF in Chapter 6.

The added value of the PCVF is shown further if we consider Figure 4.8. In Figure 4.8(a) we see a different scenario which shows a large and sudden change in *P*. As well as *qr3*, we have two additional QoSRegions, $qr2 = \{\langle P,6,-\rangle\}$ and $qr1 = \{\langle P,4,-\rangle\}$, marked as horizontal lines at $P = 6$ and $P = 4$ respectively, in Figure 4.8(a). The area of interest is between times $t = 3$ and $t = 5$. Here we see a sudden drop in *P*, from a point where it can support all three QoSRegions to a point where it can support only *qr1*. In the Boolean thresholds shown in Figure 4.8(b), we see that we have high compatibility for *qr2* at $t = 4$ even though we know that the variability of *P* is high at that point. In contrast, at $t = 4$, the PCV for *qr2* is not high, as shown in Figure 4.8(c). At $t = 5$, both the Boolean threshold and the PCV converge, but we see that the use of the PCV may help the application to avoid state-flapping (unnecessarily going into a transient state).

So, the PCVF appears to offer useful information for assessing QoSRegion changes for flows compared to a simple Boolean threshold.

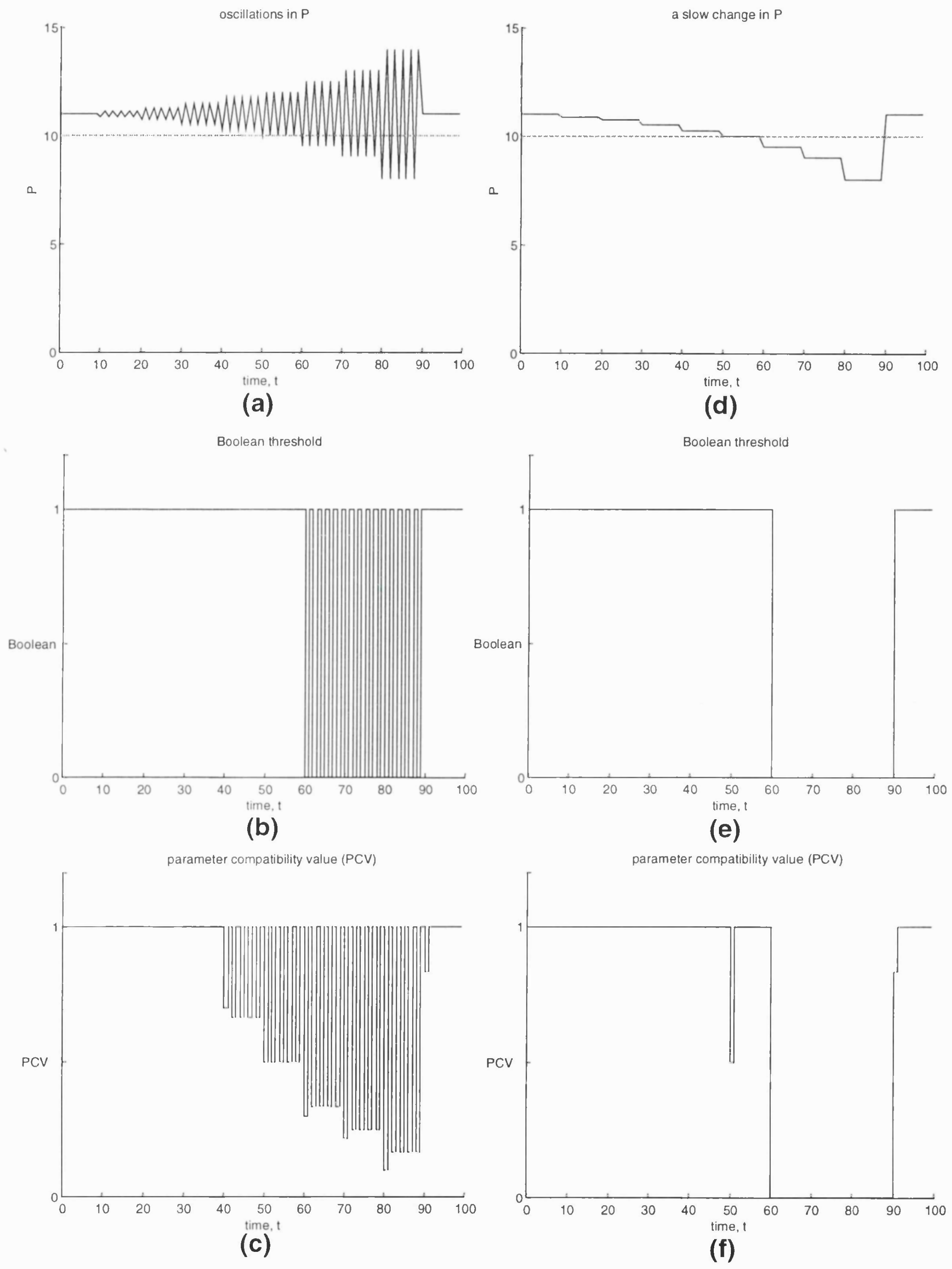


Figure 4.7: Two scenarios for changing values of *P*

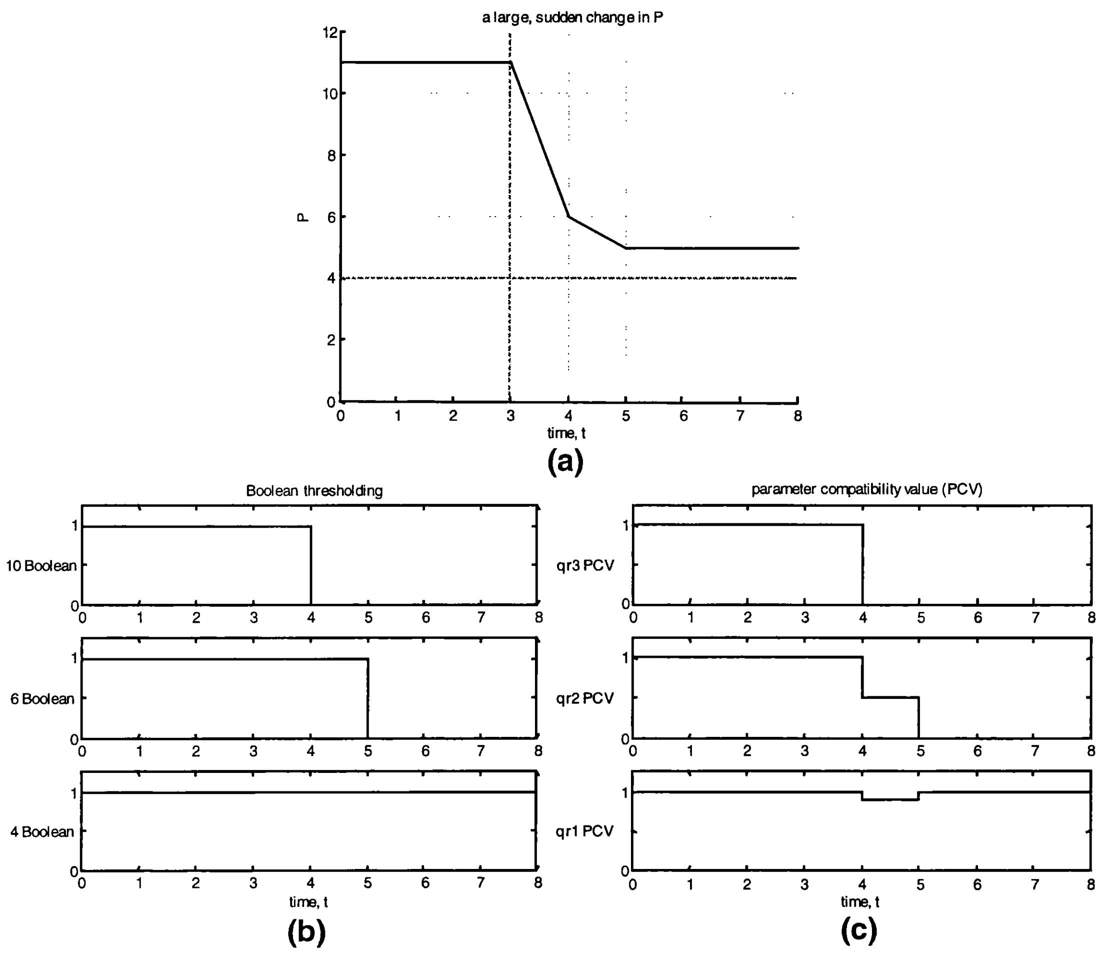


Figure 4.8: Response to large, sudden changes in *P*

4.6.3 Region compatibility value function (RCVF)

The **region compatibility value function (RCVF)** must take the PCVs for all the QoSParams in the QoSRegion and transform them into a RCV using by way of a summarisation. Any of the usual arithmetic summarisation functions that combine the values (such as a mean) could provide an incorrect summary of PCVs due to relative scaling. For example, consider five QoSParams, that give rise to the set of PCVs, $S_A = \{1.0, 1.0, 1.0, 1.0, 0.0\}$. The application may decide that it is likely to take a RCV of 0.8 to consider that a QoSRegion is useable. This may seem reasonable, but we can see that the mean of S_A is 0.8, yet clearly one of the parameter conditions cannot be supported (hence the PCV of 0.0). The application would make an incorrect decision and this could lead to application and/or network instability.

Looking at it another way, we have seen, in (4.4) that the PCVF evaluates the function WITHIN. For N QoSParams and a QoSRegion with tuples T_{pn} and NetQoSRegion with tuples T_{sn} ($\forall\ n = 1 \ldots N$), we base a RCVF algorithm on the following statement from (4.4):

$$\begin{array}{llll} \text{if} & T_{s1} \ \text{WITHIN} \ T_{p1} & \textbf{and} & \\ & T_{s2} \ \text{WITHIN} \ T_{p2} & \textbf{and} \ldots & \\ & T_{sN} \ \text{WITHIN} \ T_{pN} & & \\ \text{then} & RCV \ \text{is} \ \text{HIGH} & & \end{array} \qquad \textbf{(4.7)}$$

This reasoning makes linguistic sense. We see that if the QoSParam values (the NetQoSRegion) *all* fall WITHIN the thresholds defined in the tuples for the corresponding QoSRegions, then the degree to which the T_{sn} tuples are WITHIN their corresponding T_{pn} tuples is the degree to which the RCV is HIGH. The key to this assessment is how to evaluate **and** in (4.7) and so generate a value for HIGH.

Fuzzy logic provides a suitable interpretation of **and** as the MIN function. So, we can modify our statement to say:

$$\begin{array}{ll} \text{RCV} = & \text{PCVF}(T_{s1}, T_{p1}) \ \textbf{and}_\textbf{F} \\ & \text{PCVF}(T_{s2}, T_{p2}) \ \textbf{and}_\textbf{F} \ldots \\ & \text{PCVF}(T_{sN}, T_{pN}) \end{array} \qquad \textbf{(4.8)}$$

where **and**$_\textbf{F}$ is the fuzzy AND operator (MIN). We see from our example for the set of RCVs, S_A, (above) that we would now have the correct behaviour. The use of **and**$_\textbf{F}$ means that the RCV is in the range [0, 1], and is also computationally simple to evaluate.

Where a QoSRegion does not have a tuple defined for a particular QoSParam, then for the purposes of the RCVF, this is ignored. (Equivalently, the PCVF can evaluate a PCV = 1.0 for that QoSParam. This is also correct because it means that the QoSRegion is not dependent on that particular QoSParam, so it always has maximum compatibility with respect to that QoSParam.)

Testing of an implementation of the QoSEngine and generation of RCVs (as part of a simulation) is presented in Chapter 6.

4.6.4 QoSiRegions – QoS intermediate regions

In Section 4.6.2, we noted that the dynamics of the PCVF are such that the application may not be able to detect in advance a more gradual change in QoSParam values. In fact, we can be more precise about this: the application may like to have an indication when a QoSParam value in the NetQoSRegion is nearing a corresponding QoSRegion tuple *_lo* or *_hi* threshold. This is implemented through the use of **QoS intermediate regions** or **QoSiRegions**. These are quasi-flow-requirements that do not represent a flow-requirement as a QoSRegion does, but are an indication of the proximity of the QoSParam value to the *_lo* or *_hi* threshold of a QoSRegion. The QoSiRegion only exists within a QoSRegion, and is an optional part of the QoSRegion definition. A QoSiRegion is also specified by use of a boundaries, *_qlo* or *_qhi*. The relationship of the QoSiRegion to the QoSRegion with respect to a single QoSParam, *P*, is depicted in Figure 4.9. It is clear that a QoSiRegion can only exist if it has a corresponding *_lo* or *_hi* defined in the QoSRegion.

There are parameter compatibility values, PCV_hi and PCV_lo, associated with the *_hi* and *_lo* QoSiRegions, respectively, for each QoSParam tuple. These are evaluated in exactly the same the way as a PCV for the QoSParam in (4.5) but using the QoSiRegion tuple, $T_q = \langle p\_qhi, p\_hi \rangle$ or $T_q = \langle p\_lo, p\_qlo \rangle$ in place of the QoSRegion tuple, $T_p = \langle p\_lo, p\_hi \rangle$, as required. The *_qhi* and *_qlo* boundaries are specified by the application, and can be left undefined.

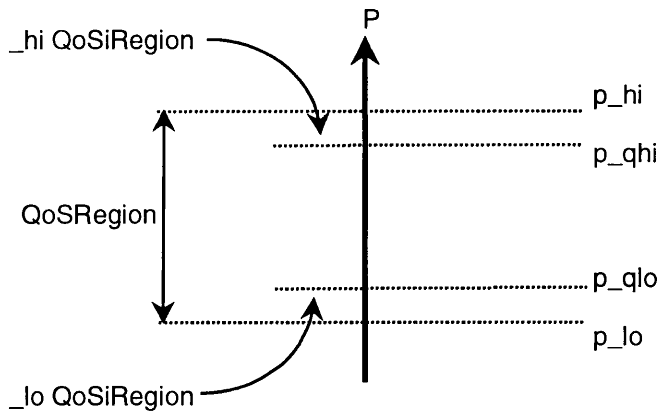


Figure 4.9: QoS intermediate regions – QoSiRegions

The QoSiRegions also have a region compatibility value, RCV_I, which is evaluated in a different manner to that of the RCV for the QoSRegion. Consider a QoSRegion defined using *N* QoSParams. If *any* of the *N* QoSParam tuples in the NetQoSRegion, T_{sn}, suggest that the corresponding QoSParam value might be WITHIN a QoSiRegion tuple, T_{qn} (either a *_lo* or *_hi* QoSiRegion) then we know that the whole QoSRegion is operating close to one of its boundaries. (The application may conclude from this that the QoSRegion may soon not be supportable, but this is an application-level decision.) So, our reasoning for the RCV_I is:

$$\begin{array}{llll} \text{if} & T_{s1} & \text{WITHIN} & T_{q1} \ \ \textbf{or} \\ & T_{s2} & \text{WITHIN} & T_{q2} \ \ \textbf{or}\ldots \\ & T_{sN} & \text{WITHIN} & T_{qN} \\ \text{then} & RCV\_I & \text{is} & \text{HIGH} \end{array} \qquad \textbf{(4.9)}$$

As with (4.7), we see that (4.9) makes linguistic sense. We need an interpretation for **or** that lets us assign a value to RCV_I. Again, fuzzy logic offers us a suitable interpretation of **or** as the MAX function. So, using (4.4) and (4.9), we have:

$$\begin{array}{ll} \text{RCV\_I} = & \text{PCVF}(T_{s1},T_{q1}) \ \ \mathbf{or_F} \\ & \text{PCVF}(T_{s2},T_{q2}) \ \ \mathbf{or_F}\ldots \\ & \text{PCVF}(T_{sN},T_{qN}) \end{array} \qquad \textbf{(4.10)}$$

where $\mathbf{or_F}$ is the fuzzy OR operator (MAX). The $\mathbf{or_F}$ operator makes sense: it only needs one QoSParam to enter a QoSiRegion to indicate that the QoSRegion is operating near a boundary. In Figure 4.10, we consider again the example presented in Figure 4.7(d) but use a *_lo* QoSiRegion for *qr3* with *p_qlo* = 10.5 (marked as a horizontal line at $P = 10.5$ in Figure 4.10(a)). We record the PCV_lo values in Figure 4.10(b). We can see in the lower graph of Figure 4.10(b) how the value of PCV_lo shows that *qr3* is in the region of the QoSiRegion, forewarning of a possible QoSRegion change.

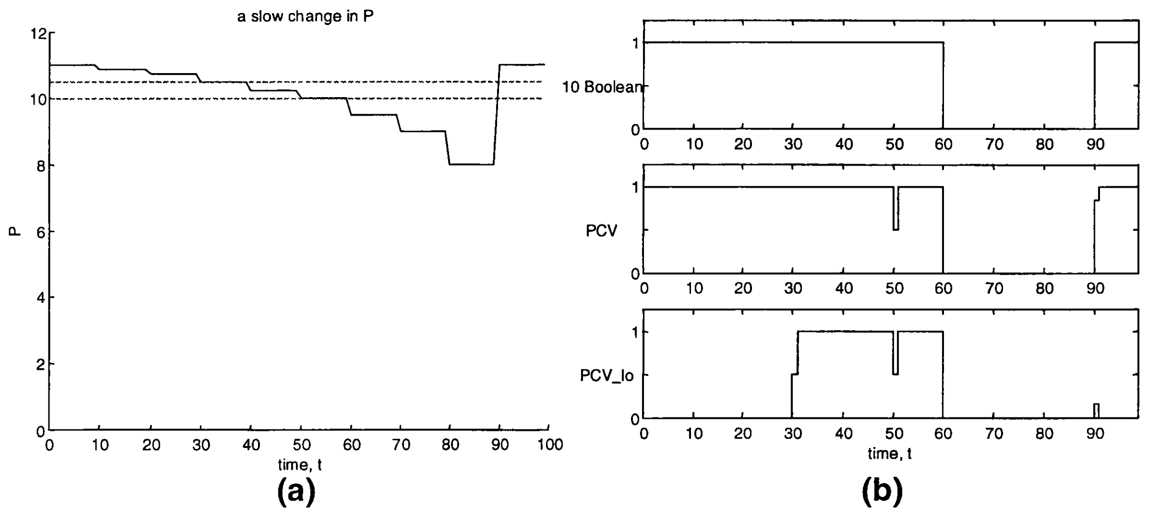
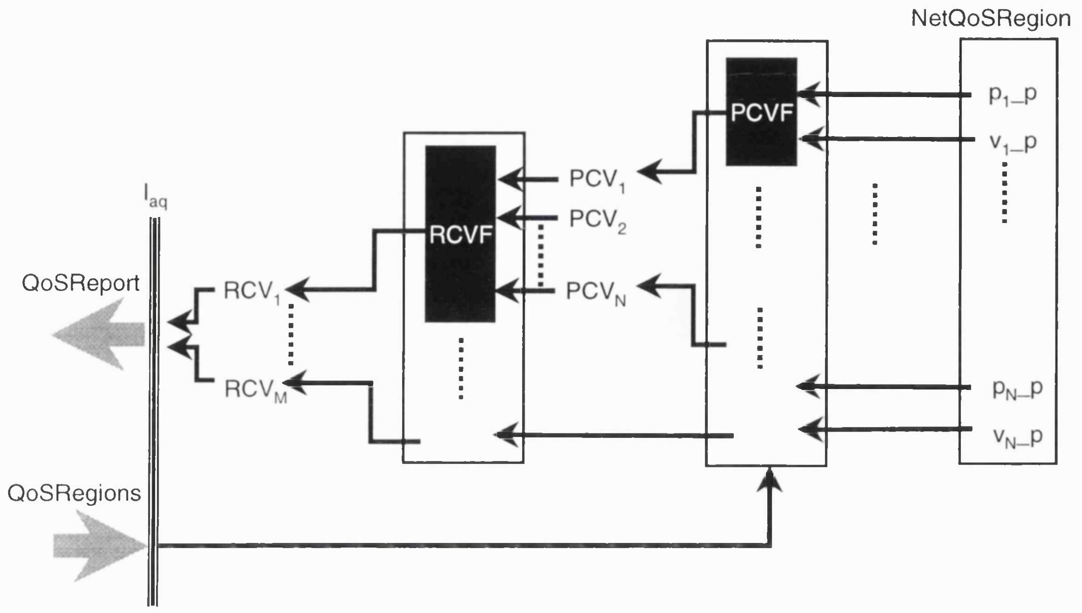


Figure 4.10: Example use of a QoSiRegion

4.7 The interface between the application and the QoSEngine, I_{aq}

The relationship between the RCVF and the PCVF components, and the interface I_{aq} are shown in Figure 4.11. Note that the process of taking measurements of QoSParam values is not the responsibility of the QoSEngine, in support of R6 and R7.



I_{aq}	application-QoSEngine interface
p_n_p	measured value of QoSParam n
PCV	parameter compatibility value (including _lo and _hi)
PCVF	PCV function
RCV	region compatibility value (including _lo and _hi)
RCVF	RCV function
v_n_p	variability of QoSParam n

Figure 4.11: A schematic diagram of the internal functions of the QoSEngine

A suitable mechanism for estimating p_n_p and generating v_n_p is described in Chapter 5.

We consider here the abstract interface between the application and the QoSEngine. The inputs to the QoSEngine are the definitions of the QoSRegions. The output of the QoSEngine is a QoSReport containing RCVs for each QoSRegion. The definition of the QoSRegion is given in (4.11).

$$\begin{aligned} QoSTuple &= \langle qt\_id, p\_lo, p\_hi, p\_qlo, p\_qhi \rangle \\ QoSRegion &= \langle qr\_id, QoSTuple_1, QoSTuple_2, \ldots QoSTuple_N \rangle \end{aligned} \qquad \textbf{(4.11)}$$

where:

qt_id	a value that can be used to unambiguously identify (in the context of the QoSEngine and the application instance[26]) the QoSParam to QoS parameter mapping for the flow.
qr_id	a value that can be used to unambiguously identify (in context of the

[26] In the context of the application instance, this will typically be a single "flow", in whatever manner the application defines "flow".

	QoSEngine and the application instance[26]) the QoSRegion to which the tuples apply
p_lo	a numeric value indicating the lower threshold value of the QoSParam. If *p_hi* is defined then *p_lo* must be less than *p_hi.*
p_hi	a numeric value indicating the lower threshold value of the QoSParam. If *p_lo* is defined then *p_hi* must be greater than *p_lo.*
p_qlo	a numeric value greater than *p_lo* indicating the upper threshold value of the QoSParam for the *_hi* QoSiRegion. *p_qlo* can only be defined if *p_lo* is also defined. If either *p_qhi* or *p_hi* is defined then *p_qlo* must be less than MIN(*p_qhi*, *p_hi*).
p_qhi	a numeric value less than *p_hi* indicating the lower threshold value of the QoSParam for the *_lo* QoSiRegion. *p_qhi* can only be defined if *p_hi* is also defined. If either *p_lo* or *p_qlo* is defined then *p_qhi* must be greater than MAX(*p_lo*, *p_qlo*).

At least one of *p_lo* or *p_hi* must be defined for the QoSRegion. *p_qlo* and *p_qhi* are optional.

The QoSEngine returns a per-flow QoSReport as follows:

$$\begin{aligned} QoSRCV &= \langle qr\_id, rcv, rcv\_i \rangle \\ QoSReport &= \{QoSRCV_1, QoSRCV_2, ... QoSRCV_M\} \end{aligned} \qquad \textbf{(4.12)}$$

where:

qr_id	as defined for the corresponding QoSRegion.
rcv	a numeric value in the range [0, 1] indicating the compatibility between the NetQoSRegion and the QoSRegion identified by *qr_id.*
rcv_i	the RCV indicating that the QoSRegion is being supported within the for the *_lo* or *_hi* QoSiRegion, if either QoSiRegion is defined.

The implementation of the interface should be asynchronous, in general, to support R6. The way the application and the QoSEngine interact can be entirely implementation-specific. For example, the application instance could "register" the QoSRegion information with a QoSEngine instance and be notified by an event carrying the QoSReport when RCVs change value. Alternatively, the application could receive QoSReports periodically, regardless of whether there are changes in the RCVs or not. The abstract interface presented here does not require or constrain any particular model of use.

4.8 Discussion

The key to the use of the QoSEngine is the definition of the QoSRegions (including QoSiRegions). The QoSRegions represent the flow-requirements of the application flow. The dynamics, semantics and relationships of QoSRegions (inter-flow and intra-flow) are known only to the application. The QoSEngine needs very little semantic knowledge of the QoSRegions.

Although in the examples, we have used common QoS parameters (such as throughput, delay and jitter) the QoSSpace can be applied in a more diverse manner. For example, parameters such as battery life, host load and cost could also be used.

The QoSEngine is presented as an abstraction that requires only two pieces of information per QoSParam per flow: an estimate of the current value, *p_p*, and an estimate of the current variability in that value, *v_p*. Our simple definition of *v_p* means that the QoSEngine needs to hold very little historic information for a flow. Additionally, the QoSEngine is separated from the mechanism that is used to provide the values of *p_p* and *v_p*. This means that the implementation of the QoSEngine is not constrained. It could, for example, be tightly coupled with the application (embedded), implemented as a kernel module or daemon on a host, or implemented as part of a distributed system using middleware.

Note that in fact what the QoSEngine actually needs for its operation is the values of *q_lo* and *q_hi* for each of the NetQoSRegion tuples. We have decided to generate these values by taking an estimate of the current value of a QoS parameter, p_p_t, from measured values and generating *q_lo* and *q_hi* by using the absolute value of the difference between p_p_t and p_p_{t-1}. (We describe a way of generating estimates for parameter values in the next chapter.) Other methods of generating *q_lo* and *q_hi* could be used. However, we choose to use the approach based on direct parameter estimation from measurements because:

- parameter value measurement and estimation is widely used within communication applications and protocols and is well supported by a number of measurement mechanisms
- there are computationally inexpensive methods of generating parameter estimates from measured values
- generation of parameter estimates may be required in any case, in order to fine tune the operation of the application

The simple feedback to the application, using only RCVs which are unitless and have values in the range [0, 1], means that applications should find it easy to use the RCV information in making decisions. The PCV and RCV give reasonable scaling properties. The computational cost for each QoSReport is approximately of the order:

$$M + N + \sum_{m=1}^{M} Q_m \qquad \textbf{(4.13)}$$

where:

M	number of QoSRegions for a flow (evaluation of RCVF)
N	number of QoSParams (evaluation of NetQoSRegion)
Q_m	number of QoSTuples defined in QoSRegion *m* (evaluation of PCVF)

If all QoSRegions have the same number of QoSTuples, with a QoSTuple for each QoSParam, (4.13) becomes:

$$M(N+1)+N \qquad \textbf{(4.14)}$$

So, in an application with a fixed number of QoSParams, the computational cost of the use of the QoSEngine scales linearly with the number of QoSRegions for the flow, i.e. the evaluation of the RCV for QoSRegions and the RCV_I for QoSiRegions. Evaluation of both the RCV and RCV_I involves only simple operations, so the added cost for extra QoSRegions and QoSiRegions is relatively low. It can be seen from (4.14) that there is also an identical linear relationship for N with fixed M. However, adding extra QoSParams may require additional network resources to measure and distribute QoS parameter values, and so is likely to have a greater overall cost than increasing M. This additional cost will depend greatly on the (application-specific) measurement and distribution mechanism used.

The QoSEngine treats all QoSParams as orthogonal. Each QoSParam is evaluated individually, in the PCVF, and only then is the RCVF for each QoSRegion evaluated. So, effectively, the treatment of any one of the N QoSParams is identical to the treatment of just a single QoSParam. Although in some cases there may be correlation between QoSParams, this is for the application designer to resolve and define QoSRegions appropriately, if required.

Applications may use elastic buffering techniques that require accurate measurements of QoS parameters like throughput, delay or jitter to fine-tune the flow play-out set-point. These applications can still make use of the QoSEngine, but may also require access to absolute QoS parameter values. In this case, the *QoSReport* definition of (4.12) can be easily extended to include the values *p_p* and *v_p*, which are passed through from the NetQoSRegion:

$$\begin{aligned} QoSRCV &= \langle qr\_id, rcv, rcv\_i \rangle \\ NetQoSTuple &= \langle qt\_id, p\_p, v\_p \rangle \\ QoSReport &= \langle \{QoSRCV_1, ...QoSRCV_M\}, \{NetQoSTuple_1, ...NetQoSTuple_N\} \rangle \end{aligned}$$

The specification of QoSRegions and NetQoSRegions makes no mention of traffic characteristics or QoS service-levels. There are no assumptions made about the support that will be provided by the network to the QoSEngine or the behaviour of the network. The formulation of the RCVs is based on simple, logical reasoning, and is computationally inexpensive.

The QoSEngine fulfils the requirements R1 to R8, as defined in Section 3.2. In particular, we believe it fulfils the requirement R4.

4.9 Chapter summary

Adaptable applications lack a mechanism that allows them to make sensible assessments of network QoS and so make decisions about dynamic adaptation in the face of QoS fluctuations. We do not know the time scales of operation of the application or the other information required by the application in order for it to make a flow-requirement change decision. So, our proposed model should provide reports – snapshots in time – about the current state of the network and its ability to support application flow-requirements. We need a model of the network QoS and the application flow-requirements. (Section 4.1.)

We define a network QoS model as a multi-dimensional space, **QoSSpace**. Each of the QoSSpace dimensions is defined by a **QoSParam**, derived from the QoS parameters that they model, e.g. throughput, delay, jitter, etc. (Section 4.3.)

The application flow-requirements are modelled as a set of **QoSRegion** definitions. Each QoSRegion represents a region of operation for the flow within the QoSSpace by use of *_hi* and *_lo* boundaries that are effectively threshold values of QoSParams for that QoSRegion. The flows are said to exist in QoSSpace as independent QoSRegions, and only the application knows the inter-QoSRegion (intra-flow) and inter-flow relationships. The application defines the QoSRegions and passes them to the QoSEngine. (Section 4.4).

The network QoS, **NetQoSRegion**, is also modelled in a similar way to QoSRegions, using values of QoSParams to indicate the current operating region of the network QoS with respect to a flow. NetQoSRegion is not a point in QoSSpace, but a region, expressing the uncertainty we have of the exact value of the QoS parameters. (Section 4.5.)

The QoSEngine generates **QoSReports** containing **RCVs (region compatibility values)**. These are values in the range [0, 1] for each QoSRegion, and express the compatibility between that QoSRegion and the network QoS, i.e. the RCV is a measure of the network's current ability to support that QoSRegion. A value of zero indicates no compatibility and a value of one indicates full compatibility. The RCVs allow the detection of rapid fluctuations directly, and **QoSiRegions** allow detection of when a QoSiRegion is operating close to one of its boundaries. The evaluation of RCVs is computationally simple and has reasonable scaling properties. (Section 4.6.)

We have defined an abstract interface, I_{aq}, between the application and the QoSEngine. This is used by the application to pass QoSRegion definitions to the QoSSpace, and allows the QoSEngine to pass QoSReports containing RCVs back to the application. The interface can be realised as an asynchronous interface, and the implementation can be loosely or tightly coupled. (Section 4.7.)

We find that the QoSEngine offers a potentially useful QoS summary in its QoSReports with reasonable scaling properties. The application may require access to the QoS parameter measurements (QoSParams) for other application specific mechanisms. If so, then the QoSParam values can be passed in the QoSReport also. We believe that the QoSEngine fulfils requirements R1 to R8. (Section 4.8.)

5. Processing QoS parameter measurements

In Chapter 4, we described the QoSSpace, an abstraction that allows the modelling of the network QoS. An interface to the QoSEngine allows:

- the application flow-requirements, QoSRegions, to be incorporated into the QoSSpace
- the QoSEngine to issue QoSReports containing RCVs (region compatibility values) for each QoSRegion as an indicator of the networks' ability to support that QoSRegion

This is the "front-end" of the QoSEngine. The QoSEngine "back-end" needs information about the network QoS in for the flows in the QoSSpace, i.e. it needs values of the QoS parameters for the flows to allow the QoSEngine to produce PCVs (parameter compatibility values). In Figure 4.11, we see that the back-end of the QoSEngine requires two values:

1. ***p_p*:** the **QoSParam** value, an estimate of the current value of the QoS parameter
2. ***v_p*:** an estimate of the current variability of the QoSParam

We have already chosen a simple definition of *v_p* from (4.6):

$$\begin{aligned} v\_p &= \mathrm{ABS}(p\_p_t - p\_p_{t-1}) \\ p\_lo &= p\_p_t - (v\_p/2) \\ p\_hi &= p\_p_t + (v\_p/2) \end{aligned} \qquad \textbf{(5.1)}$$

i.e. the absolute value of the difference between the current and previous QoSParam values. It remains for us to find a suitable mechanism for generating *p_p*.

In this chapter, we present our second contribution to addressing the dynamic adaptability problem. We describe the QoSEngine back-end, a method of providing QoSParam values and generating NetQoSRegion. This provides the network input to the PCVFs.

5.1 The problem

We must provide a mechanism to generate QoSParam values, *p_p,* from raw QoS parameter measurements. In Section 2.5, we discussed mechanisms for getting information from the network, and the need to provide an estimate of the current value of the QoS parameter value in the face of delay in the measurement process and noise in the measured values. We need a mechanism that is general, robust, accurate, and has good immunity to noise. We need to show that this mechanism is a suitable back-end to the QoSSpace.

In Figure 5.1, we show a simplified version of Figure 1.3, highlighting the area of work considered in this Chapter (dashed box). So, in this Chapter, we consider:

- the estimation mechanism (how the QoSEngine turns raw QoS parameter measurements into QoSParam values)
- the dynamics and performance of the estimation mechanism
- the integration of this estimation technique as the back-end of the QoSEngine

We test our estimation mechanism with pre-defined waveform measurements and show how it deals with real network measurements.

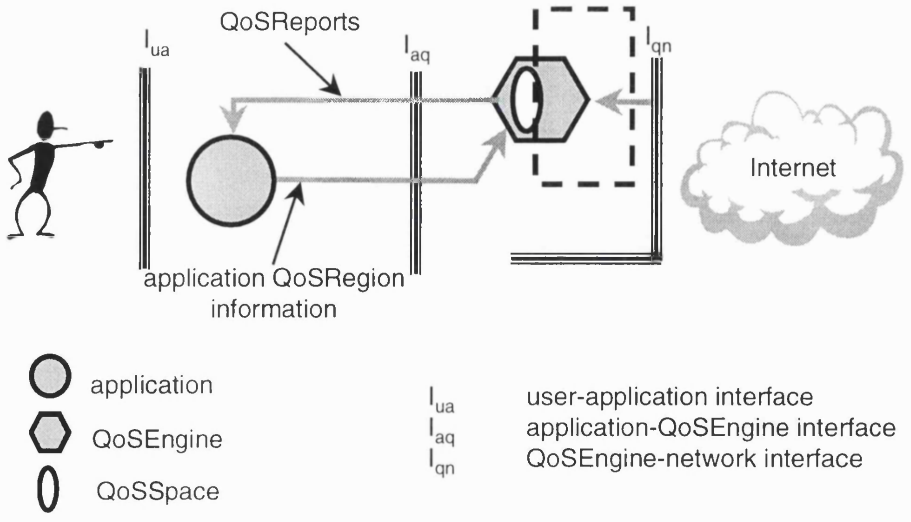

QoSReports
I_{ua}
I_{aq}
I_{qn}
Internet
application QoSRegion information
application
QoSEngine
QoSSpace
I_{ua} user-application interface
I_{aq} application-QoSEngine interface
I_{qn} QoSEngine-network interface

Figure 5.1: The back-end for the QoSEngine

5.2 Fuzzy adaptive prediction (FAP)

We need an estimation technique that is adaptive so that it can cope with the variety of QoS and connectivity conditions. We also require the estimation technique to be computationally simple, as it may be required to generate many estimates for many application flows, in software running on a general-purpose processor. A common estimation (and noise filtering) technique with such properties is **exponential weighted moving averaging (EWMA)**:

$$\hat{p}_{t+1} = \beta \hat{p}_t + (1-\beta)\tilde{p}_t \qquad \textbf{(5.2)}$$

where:

$\hat{p}$	estimated value of parameter *P*
$\tilde{p}$	measured value of parameter *P*
β	control parameter, $0 < \beta < 1$

This technique is known for its robustness, stability and ease of use, so finds applications in many situations, e.g. [KP87, Jac88, Kes91, KMR93, SEFJ97]. The key to this technique is the value of β: setting the value too large leads to a lack of responsiveness and setting the value too small leads to "noisy" estimations. Consider the graphs in Figure 5.2. In Figure 5.2(a) we see a square wave that has mean SNR = 19.9dB (10.1% noise) including large spikes. We treat the noisy trace as our measured readings for *P*.

We see in Figure 5.2(b) the effects of using (5.2) with $\beta = 0.9$, and in Figure 5.2(c) $\beta = 0.1$. We see that the former offers poor response to changes in *P*, but can cope with noise, whilst the latter provides virtually no smoothing of noise but good response. One way to deal with such circumstances is to try and adapt the value of β as required. Mechanisms for this are proposed in [Kes91] and [KMR93]. [KMR93] uses a heuristic based approach for the control of video stream traffic, while [Kes91] proposes a more general mechanism which is described in detail in [KK92]. Both [Kes91] and [KMR93] use an adaptive EWMA to produce estimates of flow parameters from network measurements, and both use the technique for QoS sensitive flows. [KK92] treats the stream of measured values as a time series of observations from a random variable, as shown in (2.1) and (2.2) (Section 2.5.2). [KK92] uses a fuzzy logic feedback control system[27] to adapt the value of β, and we will call this the **fuzzy adaptive predictor (FAP)**. The results of using FAP with the noisy trace in Figure 5.2(a) is given in Figure 5.2(d).

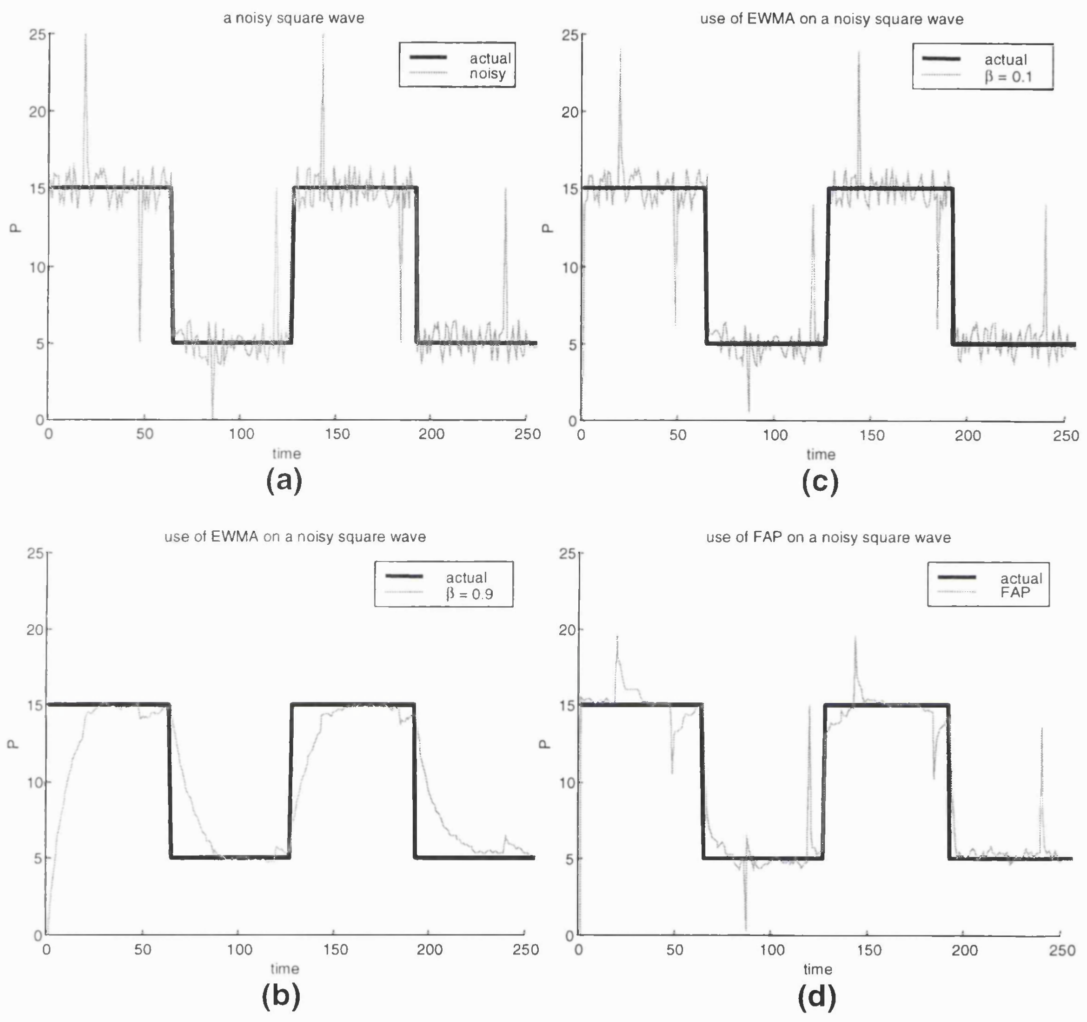


Figure 5.2: Using EWMA: the effects of changing β

When comparing Figure 5.2(d) with Figure 5.2(b), we find that FAP has fairly good immunity to most of the noise as well as being responsive. Although the FAP is designed to have some immunity to spikes, comparing Figure 5.2(a) and Figure 5.2(d), we find that the FAP is not immune to large spikes in the measured values. This is demonstrated clearly in the example of Figure 5.3. Figure 5.3(a) is the input to the FAP, a steady value of 10 with four spikes, and Figure 5.3(b) shows the output from the FAP. In the next

[27] The following information, not documented in [KK92] is a result of private communication with S. Keshav: all the fuzzy rules (assertions) are evaluated using the *min-max rule of inference* with a *centroid (composite moment) de-fuzzification*, and the final consequent fuzzy region is then normalised to produce the mapping functions. Full details can be found in [KK92]. Note that [KK92] has an incorrect diagram for the FAP in Figure 2 and the correct version appears in [Kes91] in Figure 4.

section, we describe how these large spikes can be removed[28], if required, by using a pre-filter to the FAP.

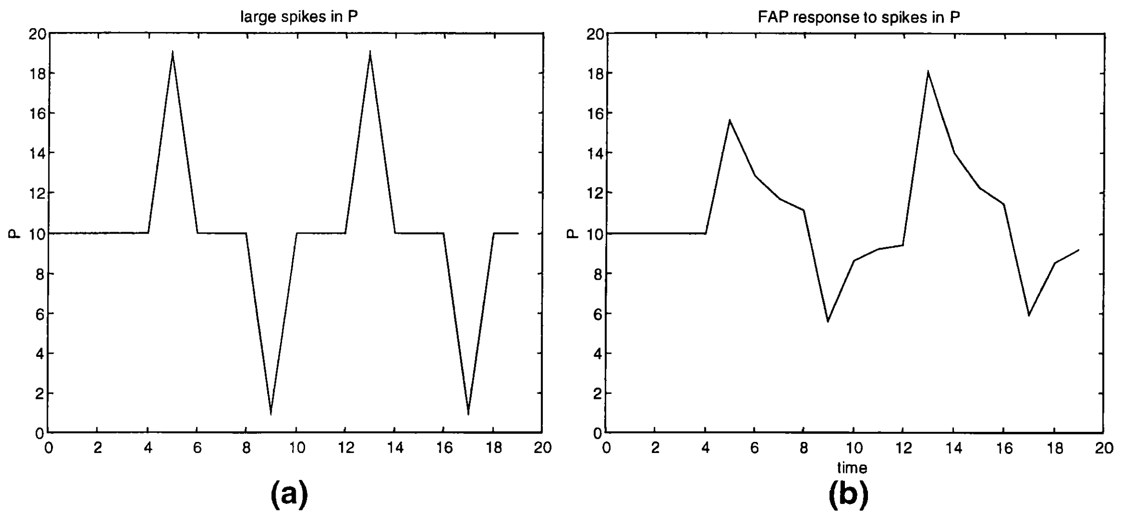


Figure 5.3: The response of FAP to large spikes in values of *P*

5.3 Fuzzy adaptive smoothing (FAS)

We present a smoothing mechanism based on the work in [KK92]. This mechanism, which we call the **fuzzy adaptive smoother (FAS),** can be used as a pre-filter to the FAP (or any other mechanism). The purpose of this pre-filter is to "de-spike" noisy measurements. The FAS is based on (5.1):

$$\hat{p}_t = \gamma \hat{p}_{t-1} + (1-\gamma)\tilde{p}_t \tag{5.3}$$

The FAS uses a fuzzy logic mechanism to provide values for γ . The fuzzy reasoning for computing values of γ is as follows:

$$\begin{array}{llll} \text{if} & \text{CIV is LARGE} & \text{then} & \gamma \text{ is LARGE} \\ \text{if} & \text{CIV is SMALL} & \text{then} & \gamma \text{ is SMALL} \end{array} \tag{5.4}$$

where **CIV** is the **change-in-value**:

$$\text{CIV} = \text{MIN}\left(1, \frac{p_1}{p_2} - 1\right), \quad p_1 = \text{MAX}(\tilde{p}_t, \tilde{p}_{t-K}), p_2 = \text{MIN}(\tilde{p}_t, \tilde{p}_{t-K}) \tag{5.5}$$

[28] As FAP is not immune to large spikes, we use our own smoothing mechanism, FAS, which can cope with large spikes. So, in our use of FAP, we choose to set the FAP "spike-smoothing" in the estimation of "change-in-error" to 1 rather than 3 as suggested in [KK92] and [Kes91].

K is an integer that is a measure of the "spike-duration" that we wish to filter. Note that for (5.5) we define:

CIV = 0 if $p_1 = 0, p_2 = 0$

CIV = 1 if $p_1 \neq 0, p_2 = 0$

CIV has the range [0, 1], and assumes that measured values of p_t and p_{t-K} are always positive. To explain the use of CIV, we assume that we are in the steady state. The use of (5.3) with the if-then assertions in (5.4) are saying:

1. if there is a LARGE change-in-value (CIV), this is probably due to a spike, rather than a change in P, so give precedence to the previous estimate, i.e. ignore the spike
2. if there is a SMALL change in value (CIV), this is probably due to changes in the value of P so give precedence to the current measured value

In the definition of CIV from (5.5), we see that we have chosen to define a spike as a change of factor 2 or greater in the measured values. This is depicted in Figure 5.4.

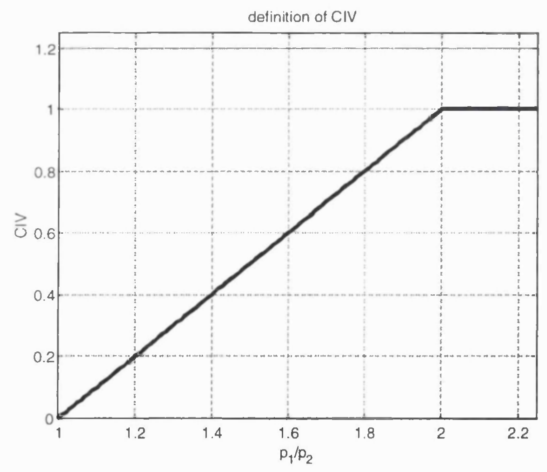


Figure 5.4: Definition of CIV (change-in-value)

The effects of increasing the value of K are shown in Figure 5.5. The input to the FAS is shown in Figure 5.5(a) and has four spikes of duration 1, 2, 3 and 4. The effect of using the FAS with $K = 1$, $K = 2$ and $K = 3$ is shown in parts (b), (c) and (d) of Figure 5.5, respectively. Notice the delay in the output of K time units compared to the input. So the FAS imposes a trade-off between filtering of spikes and timeliness of information.

LARGE and SMALL are fuzzy linguistic variables that are defined as shown in Figure 5.6(a). The overall mapping of CIV to γ is shown in Figure 5.6(b). The mapping in (5.4) from CIV to γ uses a min-max rule of inference with a centroid (composite moment) de-fuzzification process, and then the final consequent fuzzy set is normalised.

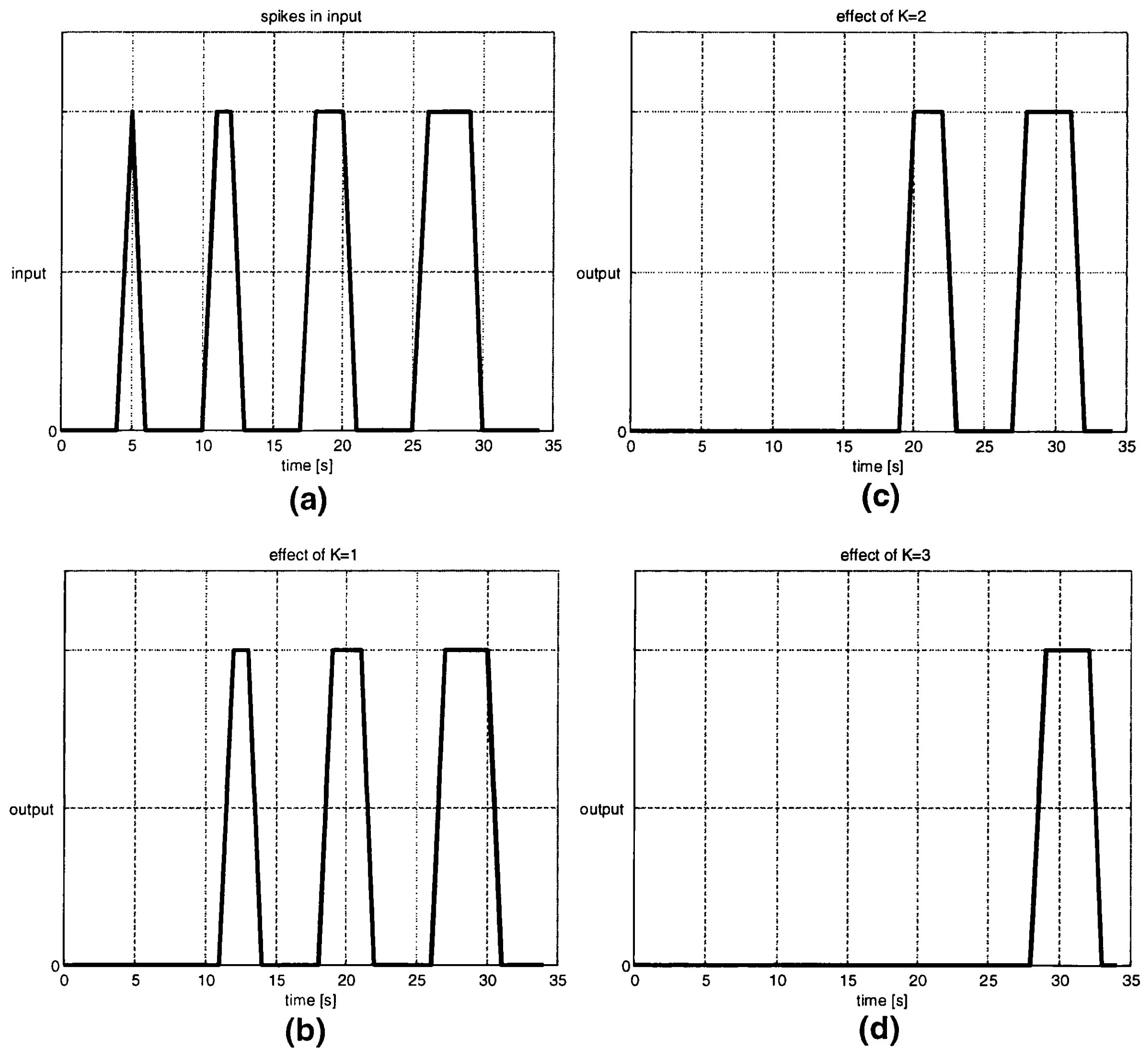


Figure 5.5: Using fuzzy adaptive smoothing (FAS) to remove spikes of duration *K* from the input

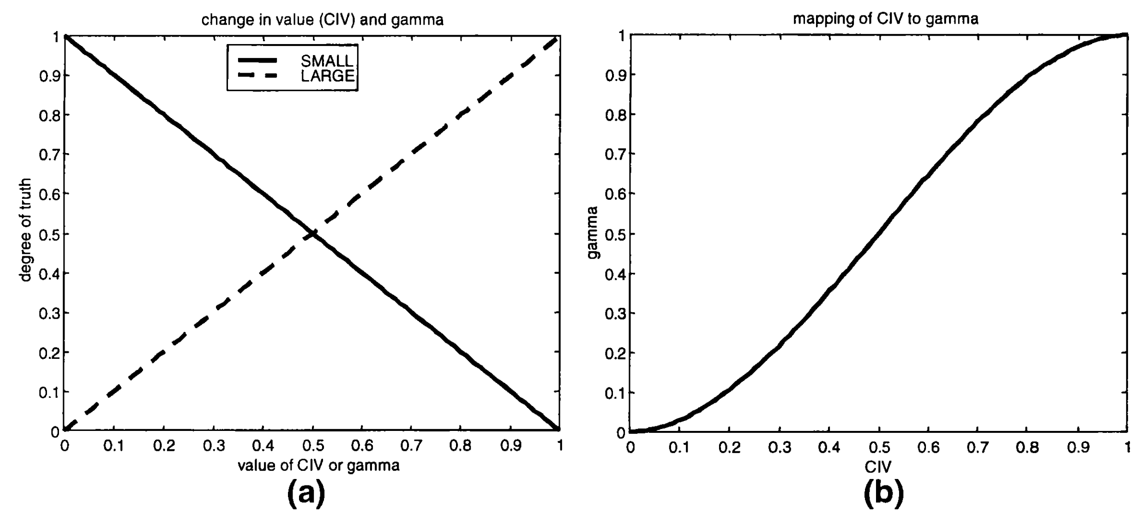


Figure 5.6: (a) defintion of LARGE and SMALL; (b) mapping for change-in-value (CIV) to γ

The mapping in Figure 5.6(b) is an evaluation of the assertions in (5.4). The value of γ can be computed from CIV efficiently by use of tables for Figure 5.6(b)[29]. This removes the need to evaluate the consequent fuzzy sets in (5.4) dynamically. The process of computing a value for $\hat{p}_t$ involves the following steps:

1. evaluate CIV using (5.5)
2. map CIV to γ using tables for Figure 5.6(b)
3. evaluate $\hat{p}_t$ using (5.3)

Notice that Figure 5.6(b) shows that the mapping of CIV to γ is very close to being linear. Using the linear approximation CIV = γ would eliminate step 2, simplifying the evaluation of $\hat{p}_t$ even further. The amount of state required is simply the table for Figure 5.6(b) (which is static) and the previous K measurements of P for evaluating the CIV in (5.5). Similar methods are used for the FAP, which has two EWMA adaptive filters, but the FAP only needs to store one previous value for each of them. So, both the FAS and FAP mechanisms remain relatively inexpensive in terms of computational cost.

The FAS is used to pre-filter the input to the FAP (FAS-FAP), as shown in Figure 5.7.

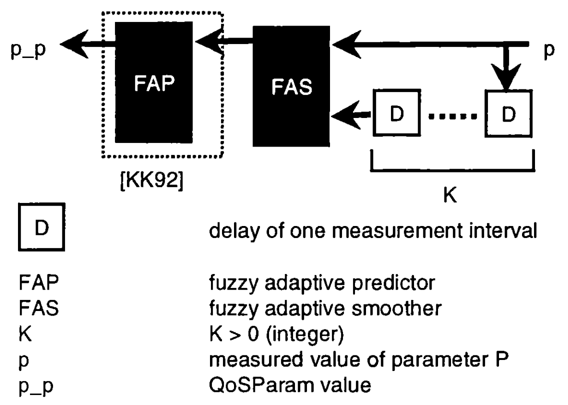


Figure 5.7: A schematic diagram of the FAS-FAP estimator for producing QoSParam values

The effects of using FAS-FAP ($K = 1$) is shown in Figure 5.8. Figure 5.8(a) shows a generated input of steady state value P = 10 with (uniform, random, zero mean) noise, and spikes of duration 1. The input has a mean SNR = 19.7dB (10.4% noise). Figure 5.8(b) shows the effects of using only the FAP and then using the FAS and FAP together.

[29] In all our simulations, we have used tables of size 100.

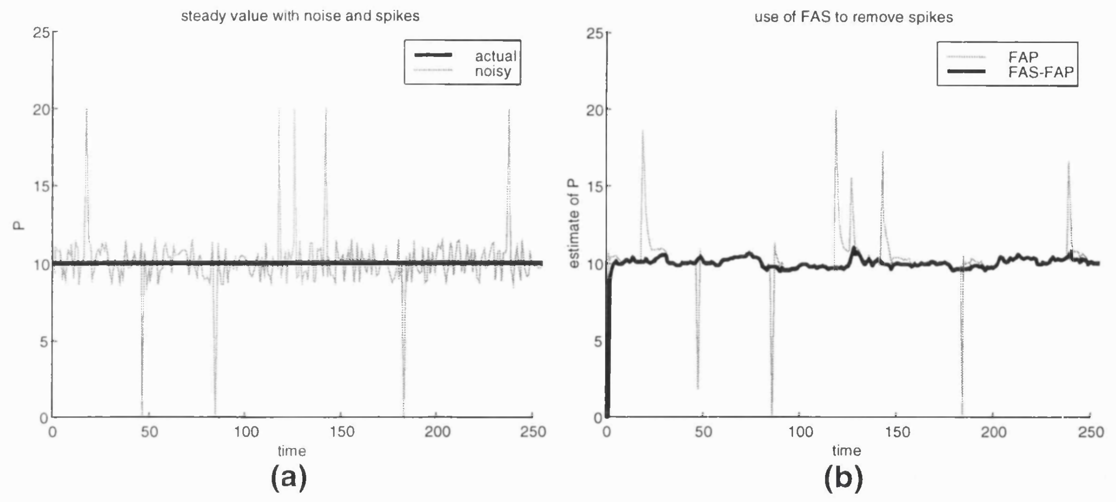


Figure 5.8: Use of fuzzy adaptive smoothing (FAS) and fuzzy adaptive prediction (FAP)

For now we see that the FAS-FAP appears to provide suitable removal of noise and spikes. Next, we examine the FAP in more detail and assess its suitability for use in the QoSEngine back-end.

5.4 Performance of the FAP

Here we evaluate the performance of the FAP. Our metric is the signal-to-noise ratio (SNR) that we find in the output of the FAP compared with a set of measurements of pre-defined waveforms that have been modulated with random, uniformly distributed, zero-mean noise.

In our evaluation, we compare the FAP system with another method of adaptive EMWA, which we will call KMR (after its authors in [KMR93]). KMR uses the same form as (5.1), but with a different algorithm for evaluating β:

$$\hat{p}_{t+1} = \beta_{KMR}\hat{p}_t + (1-\beta_{KMR})\tilde{p}_t \qquad \textbf{(5.6)}$$

$$\begin{aligned} E &= \tilde{p}_t - \hat{p}_t \\ \sigma_{t+1} &= 0.25E^2 + 0.75\sigma_t \\ \beta_{KMR} &= 0.25E^2 / \sigma_{t+1} \end{aligned} \qquad \textbf{(5.7)}$$

We have three groups of tests, comparing the system output of both the FAP and KMR systems with an input of:

1. various pre-defined wave forms, each with (approximately) the same mean SNR
2. steady state P with decreasing mean SNR (increasing noise)
3. randomly generated waveforms with random noise and varying mean SNR

We use the following definitions in the results below:

SNR_S	signal to noise ratio of input
SNR_{IFAP}	SNR improvement in FAP output
SNR_{IKMR}	SNR improvement in KMR output

5.4.1 FAP response to pre-defined waveforms

Here, we measure the response of the FAP estimator to certain pre-defined waveforms that have been subjected to noise in order to assess its ability to detect system perturbations in the presence of noise. The waveforms and the response of the FAP and the KMR mechanisms are shown in Figure 5.9 to Figure 5.14. The mean SNR is evaluated across the entire set of values. A comparison of the SNR improvement in the output of the FAP and KMR mechanisms is given in Table 5.1 and Figure 5.15.

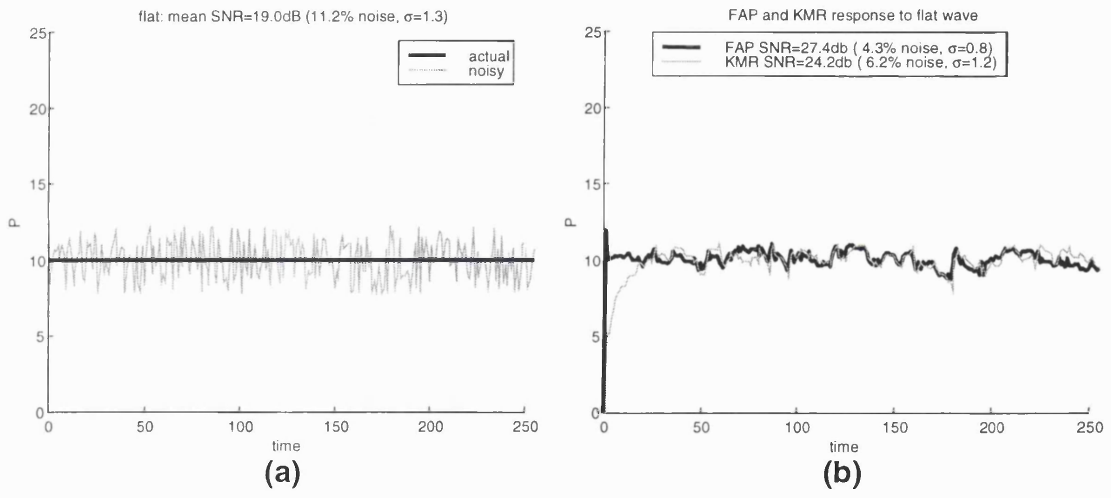


Figure 5.9: FAP and KMR response to noise in steady state

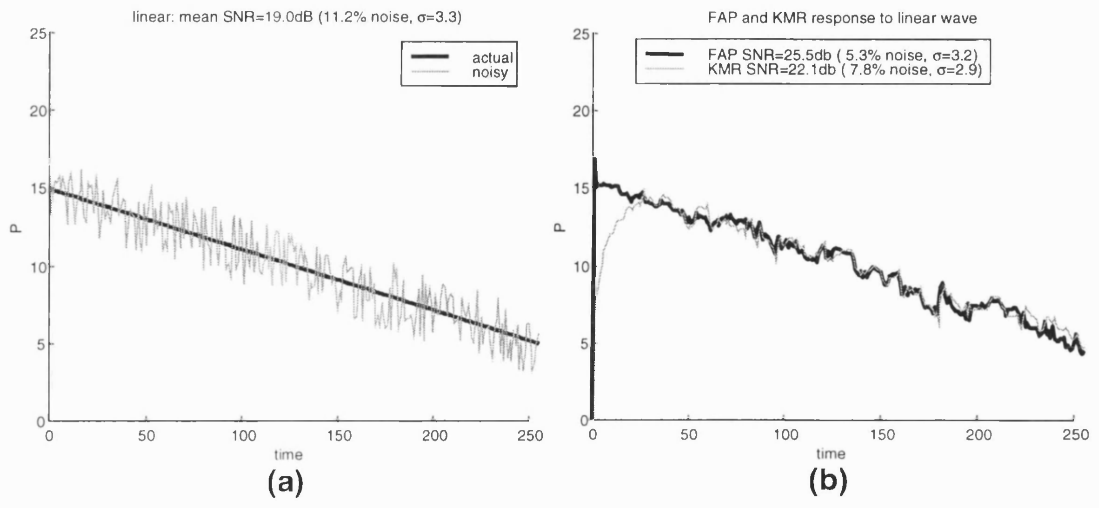


Figure 5.10: FAP and KMR response to linear change

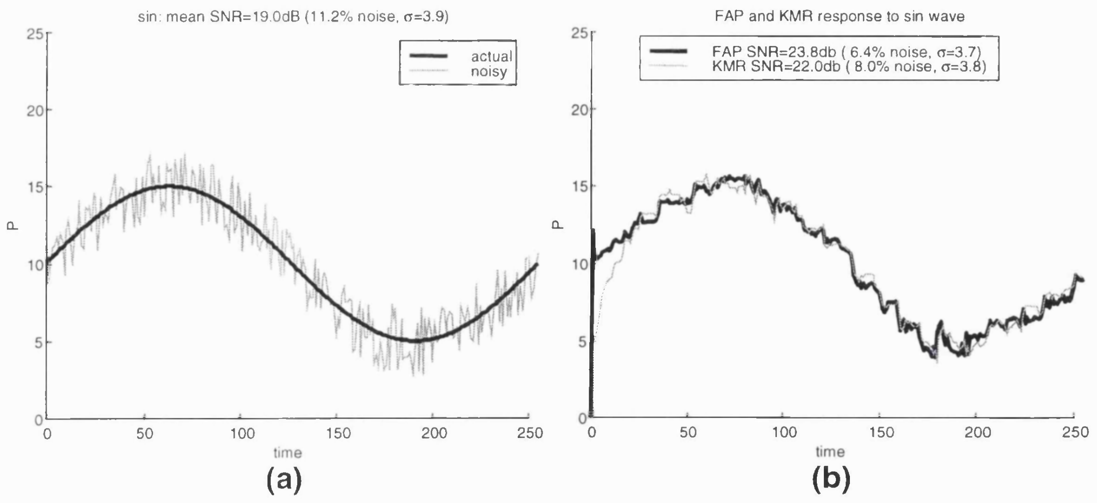


Figure 5.11: FAP and KMR response to sine wave

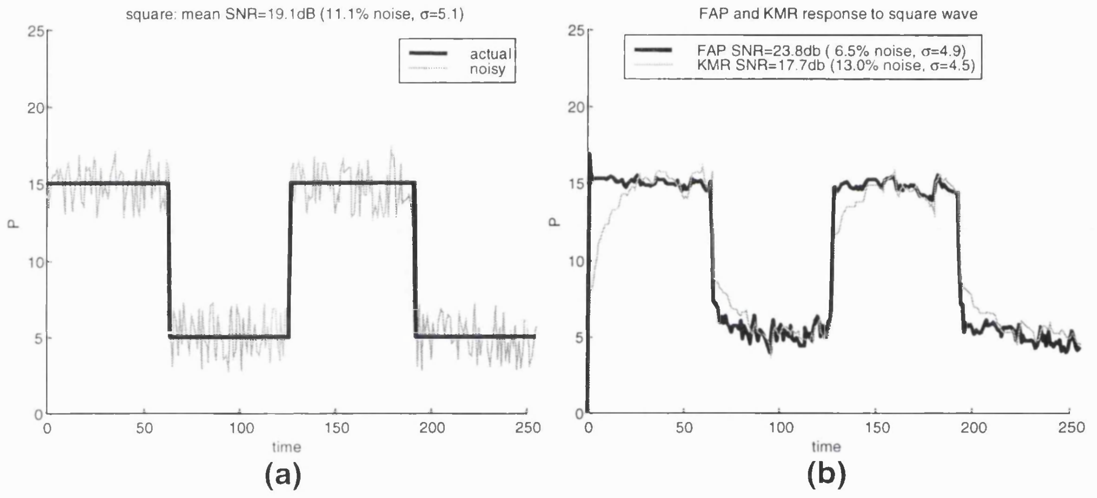


Figure 5.12: FAP and KMR response to a square wave

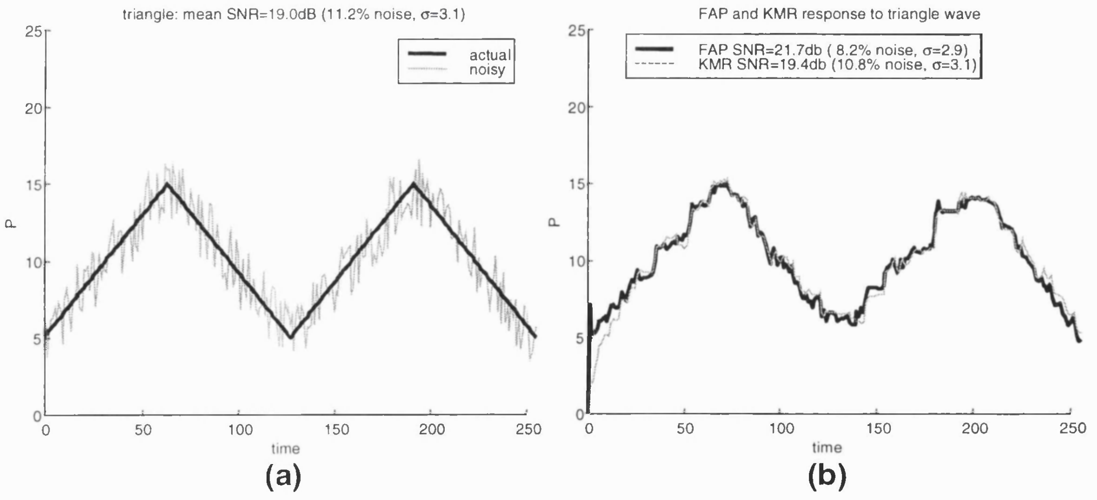


Figure 5.13: FAP and KMR response to triangle wave

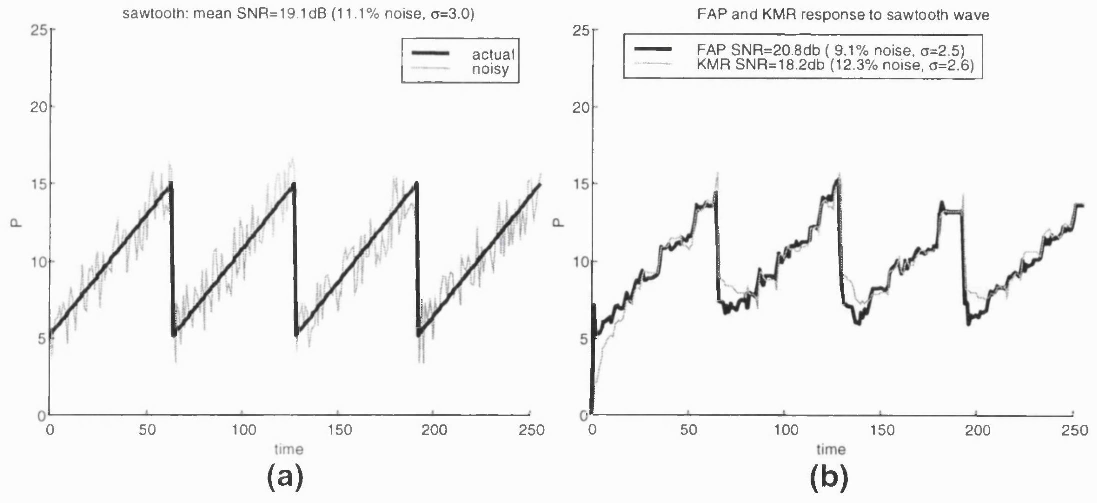


Figure 5.14: FAP and KMR response to sawtooth wave

waveform	SNR_S [dB]	SNR_{IFAP} [dB]	SNR_{IKMR} [dB]
flat	19.0	8.4	5.2
linear	19.0	6.5	6.1
sin	19.0	4.8	3.0
square	19.1	4.7	-1.4
triangle	19.0	2.7	0.3
sawtooth	19.1	1.7	-0.9

Table 5.1: Comparison of FAP and KMR response: different waveforms (see Figure 5.15).

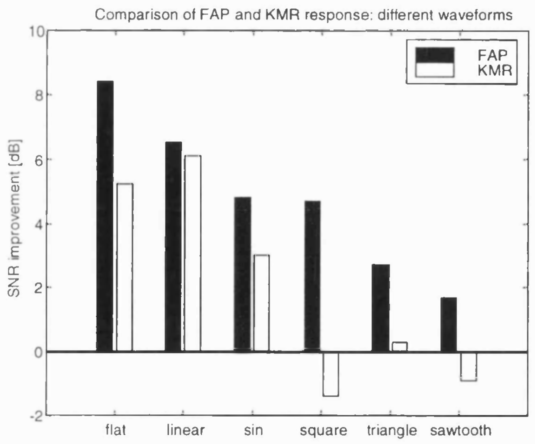


Figure 5.15: Comparison of FAP and KMR response: different waveforms (see Table 5.1)

These pre-defined waveforms were chosen to represent a range of forms for the system perturbations. We see that both the FAP and KMR produce an increased SNR in the steady state (flat, Figure 5.9). Both the FAP and KMR systems also deal well with slow, gradual changes in value (linear, Figure 5.10, and sin, Figure 5.11). Additionally, FAP responds well to sharp changes from one steady state to another (square, Figure 5.12), as might be experienced in modal behaviour. The FAP system also gives some improvement in SNR when faced with faster, gradual changes in value (triangle, Figure 5.13, and sawtooth, Figure 5.14) but in such environments, sharp changes are harder for the FAP to handle. The FAP system performs well, and shows an ability to cope with the variety of behaviour in the test inputs. FAP performs better than KMR in all the tested cases (Table 5.1 and Figure 5.15).

5.4.2 FAP response to noise in the steady state

We examine the ability of the FAP estimator to cope with different levels of noise. We use the steady state as in Figure 5.9, but with different levels of noise. Again, we use the KMR system for comparison. The results are presented in Table 5.2 and Figure 5.16.

SNR_S [dB] (% noise)	SNR_{IFAP} [dB] (% noise)	SNR_{IKMR} [dB] (% noise)
31.9 (2.6)	11.8 (1.9)	0.8 (0.2)
25.8 (5.1)	10.4 (3.6)	3.5 (1.7)
22.3 (7.7)	9.3 (5.0)	4.6 (3.2)
19.8 (10.2)	8.6 (6.4)	5.2 (4.6)
17.9 (12.8)	8.2 (7.8)	5.7 (6.1)
16.3 (15.3)	7.8 (9.0)	5.9 (7.6)
15.0 (17.9)	7.4 (10.3)	6.1 (9.1)
13.8 (20.4)	7.1 (11.4)	6.3 (10.5)
12.8 (23.0)	6.8 (12.4)	6.4 (12.0)
11.9 (25.5)	6.4 (13.3)	6.5 (13.4)
11.0 (28.1)	6.1 (14.1)	6.5 (14.8)
10.3 (30.6)	5.8 (14.9)	6.5 (16.2)
9.6 (33.2)	5.5 (15.6)	6.6 (17.6)
8.9 (35.7)	5.2 (16.2)	6.6 (19.0)
8.3 (38.3)	5.0 (16.7)	6.6 (20.4)
7.8 (40.8)	4.8 (17.4)	6.6 (21.7)
7.3 (43.4)	4.7 (18.1)	6.6 (23.1)
6.8 (45.9)	4.4 (18.4)	6.6 (24.5)
6.3 (48.5)	4.2 (18.6)	6.6 (25.9)
5.8 (51.0)	3.8 (18.2)	6.6 (27.3)

Table 5.2: Comparison of FAP and KMR response: different noise levels (see Figure 5.16)

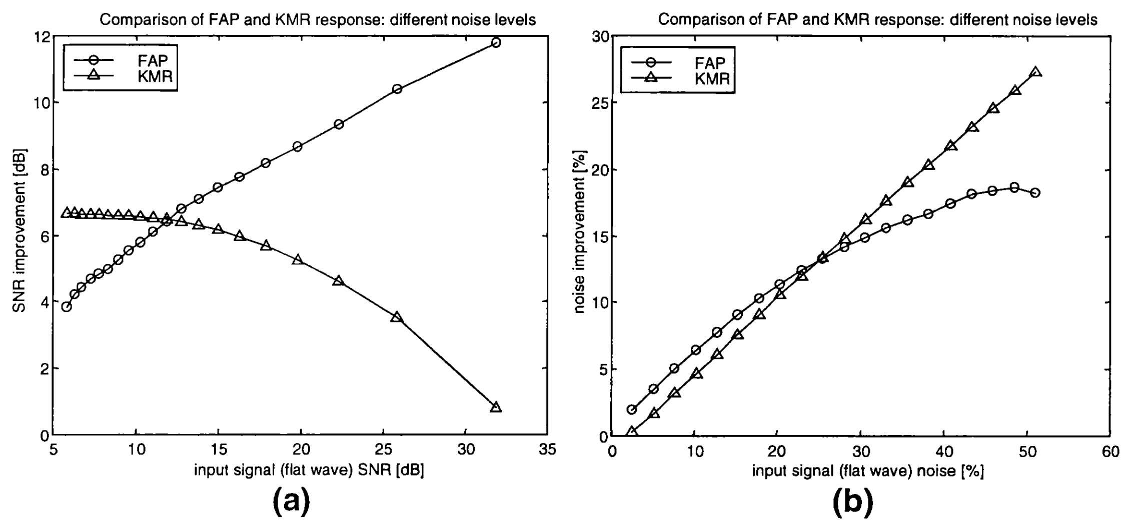


Figure 5.16: Comparison of FAP and KMR response: different noise levels (see Table 5.2)

Figure 5.16(a) shows the noise improvement in dB and Figure 5.16(b) shows the noise improvement as % noise. We see that both the FAP and the KMR systems give improved SNR under increasingly noisy conditions. At around 12dB (~25% noise in the input) in the steady state measurements, the tested cases suggest that the FAP system performs worse than the KMR system, although it continues to provide a noise improvement.

5.4.3 FAP response to random signals with random noise

We have so far examined the performance of the FAP system using a fairly limited number of noise and system perturbation conditions. We have seen that, compared to the KMR, the FAP seems to perform better across different types of system perturbation, while the KMR seems to respond better to conditions of increasing noise (below 12dB SNR) in the steady state input. However, in (2.1) and (2.2) (Section 2.5.2), we have said that we must treat our measurements as values from a random variable modulated with noise from another random variable, so we have tested the FAP with randomly generated inputs.

We have performed a paired-difference test using values of the SNR improvement in the output from both the FAP (SNR_{IFAP}) and KMR (SNR_{IKMR}) systems given the same input (SNR_S). We state the null hypothesis and alternative hypothesis:

$$\begin{aligned} H_0 &: \mu_d > 0 \\ H_1 &: \mu_d \le 0 \end{aligned} \qquad \textbf{(5.8)}$$

where the subscript d denotes the sample paired-difference between SNR_{IFAP} and SNR_{IKMR} results. We choose a 99% confidence level. We need to perform n tests:

$$n = \frac{z^2 \sigma_d^2}{E^2}$$

and with $z = 2.33$ (from tables), $\sigma_d = 0.03$ (estimated from preliminary trials) and $E = 0.01$ (99% confidence), we have $n = 49$. However, we choose $n = 1000$ in order to examine other properties. We assess the performance of the FAP system in response to some randomly generated waveforms that have been subjected to noise. A summary of the waveform details is as follows:

T1. 1000 waveforms each consisting of 512 values
T2. each waveform constructed from randomly selected flat segments of values in the range [0, 10000], and linear segments of gradients in the range [0, 500], each waveform consisting of at least 32 segments (i.e., each segment has a duration of no more than 16 values)
T3. all values in the range [0, 22010]
T4. all waveforms modulated with uniform, random, zero-mean noise with values in the range [0, 1000]
T5. standard deviation values for the waveforms are in the range [2820.2, 6549.7]
T6. mean SNR of the waveforms varies between 12.3dB and 21.1dB, with mean SNR of 16.7dB across all the waveforms

Note that the waveforms have relatively low SNR, near the lower range of the useful operational limits of the FAP and KMR (according to our simple analysis so far).

Our paired-difference evaluation yields $\mu_d = 5.09$, $\sigma_d = 0.02$, giving a z-score test statistic of 254.44. So we accept H_0 and believe the FAP to be more suitable for use with the QoSEngine than the KMR. The distribution of SNR_S values for our input waveforms is shown in Figure 5.17(a) (bin size = 0.2dB). The SNR improvements given by KMR and FAP are shown in Figure 5.17(b) and Figure 5.17(c) respectively (bin size = 0.2dB). We notice that the KMR gives worse SNR for 99.8% of the waveforms in our test data. This does not mean that the KMR system will always give worse SNR than its input, as we have already noted that the KMR system performs better than the FAP in some cases. However, it would appear that while KMR does gives good noise filtering in the steady state, is not as reactive as FAP to large, sudden system perturbations (this can be seen clearly in Figure 5.12(b), Figure 5.14(b) and Figure 5.15 for the noisy square wave and sawtooth inputs). So, overall, the FAP has better performance.

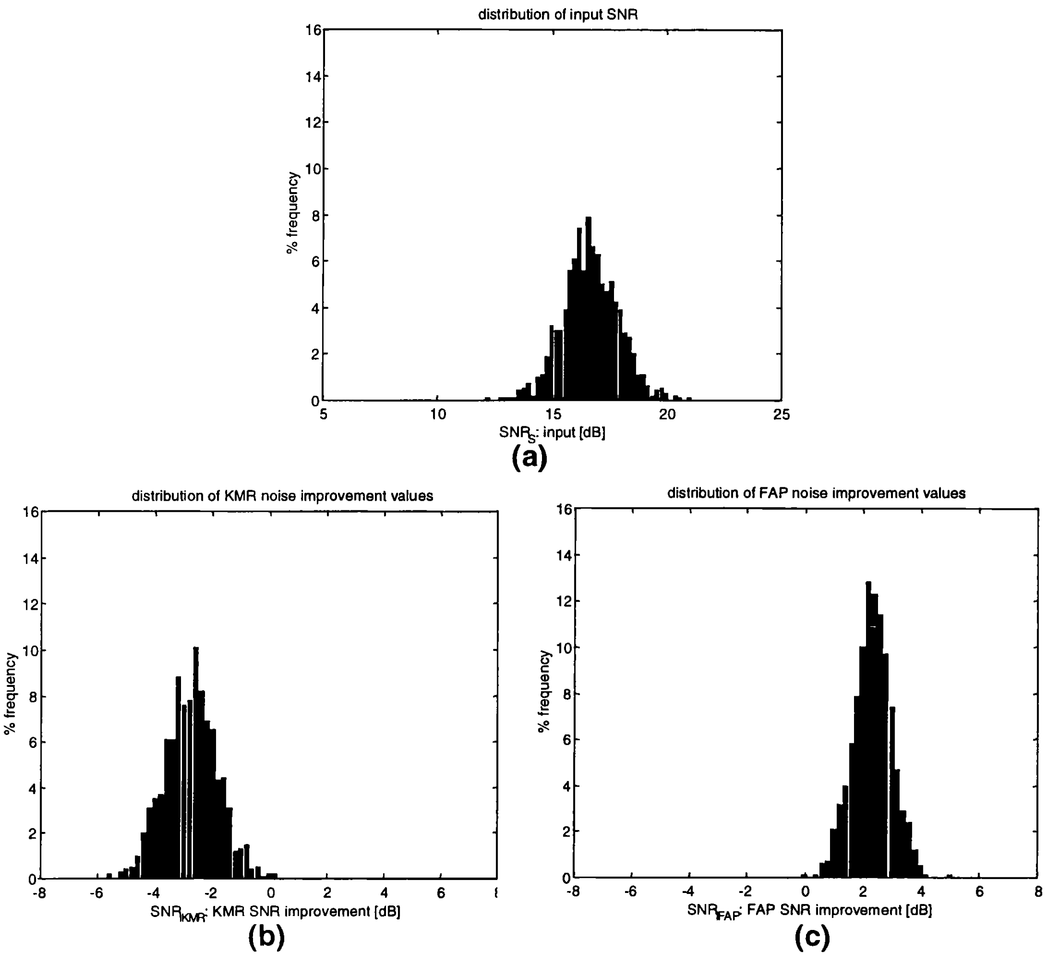


Figure 5.17: (a) input SNR figures; (b) SNR improvement for KMR; (c) SNR improvement for FAP

5.5 Producing NetQoSRegion values

The purpose of the QoSEngine back-end is to provide the NetQoSRegion (see Figure 4.11). NetQoSRegion is essentially two values per QoSParam: *p_p*, the current estimate of the QoSParam and *v_p* an estimate of its variability. We have described a mechanism, the FAP, for deriving *p_p* from raw QoS parameter measurements and we have our definition for *v_p* in (5.2), so a schematic of the QoSEngine back-end is as given in Figure 5.18, which we call the **FAS-FAP system**. The variability estimation function (VEF) in Figure 5.18 implements (5.2).

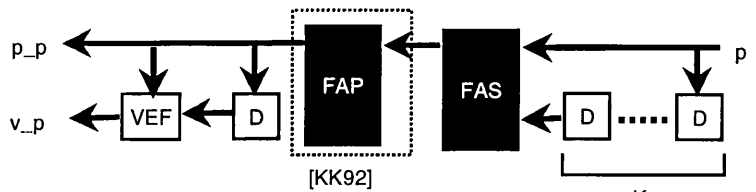
The NetQoSRegion is defined in the same abstract way as QoSRegion:

$$\begin{aligned} &NetQoSTuple = \{qt\_id, p\_p, v\_p\} \\ &NetQoSState = \{NetQoSTuple_1, NetQoSTuple_2, \ldots NetQoSTuple_N\} \end{aligned} \qquad (5.9)$$

qt_id a value that can be used to unambiguously identify (in the context of the QoSSpace and the application instance) the QoSParam to QoS parameter mapping for the flow. This identifier should map to the

	corresponding QoSTuple.*qt_id* specified in (4.11), but the mapping does not require QoSTuple.*qt_id* and NetQoSTuple.*qt_id* to be identical
p_p	estimate of QoSParam value
v_p	estimate of variability of QoSParam value

The mapping of *qt_id* is application specific. Typically, *qt_id* will identify a parameter from a flow. The mechanism for obtaining the raw QoS parameter measurements, *p*, in Figure 5.18 is application-specific.



D	delay of one measurement interval
FAP	fuzzy adaptive predictor
FAS	fuzzy adaptive smoother
K	K > 0 (integer)
p	measured value of parameter P
p_p	QoSParam value
v_p	estimate of variability of p_p
VEF	variability estimation function

Figure 5.18: A schematic diagram of the QoSEngine back-end

We have examined the FAS-FAP system using some real network measurements of path data rate estimates. These measurements were performed using **RDJ probes** to various sites across the Internet (see Appendix A). In Figure 5.19 to Figure 5.24, we see the use of the RDJ probes (the raw values in (a)) and the FAS-FAP (the processed values in (b)). Although we have no reference as in our tests so far, from our analysis in Section 5.4, we believe the output of the FAS-FAP to provide a reasonable estimate for the path data rate in the tested cases. Indeed, Figure 5.19(a) is identical to Figure 2.3(a) (the scenario in Figure 2.2) and we can see form the graph that we have a reasonable estimate of the achievable data rate across the BR-ISDN line. However, we notice that there is some residual error in the output of the FAS-FAP. Note that the mechanism used for the RDJ probes produces particularly noisy and inaccurate measurements, but we have chosen such a method deliberately in order to see the effectiveness of the FAS-FAP. Information about the hosts and their location for these tests is given in Appendix A.

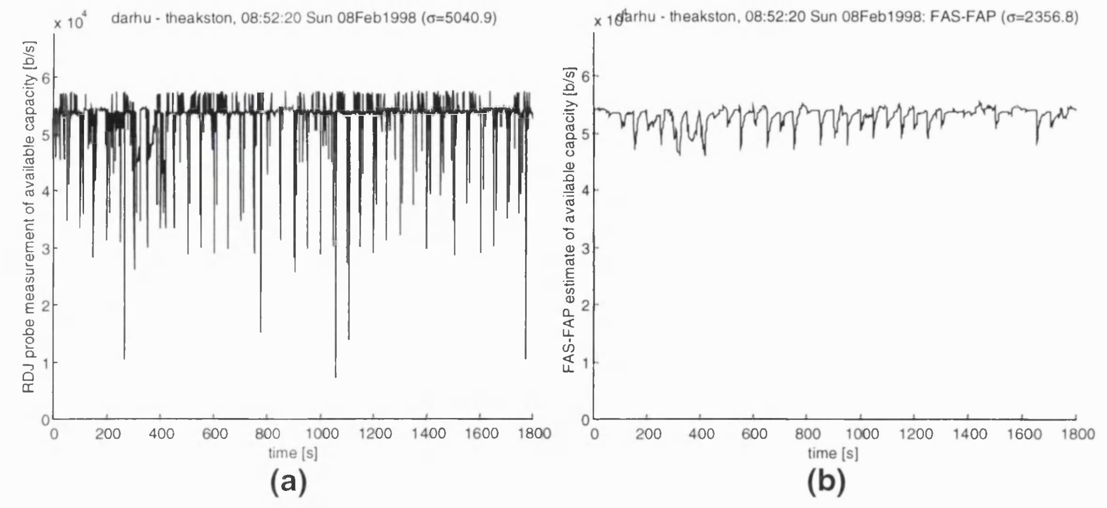


Figure 5.19: FAS-FAP (K = 2) with RDJ probes: `darhu - theakston` (see Figure 2.2)

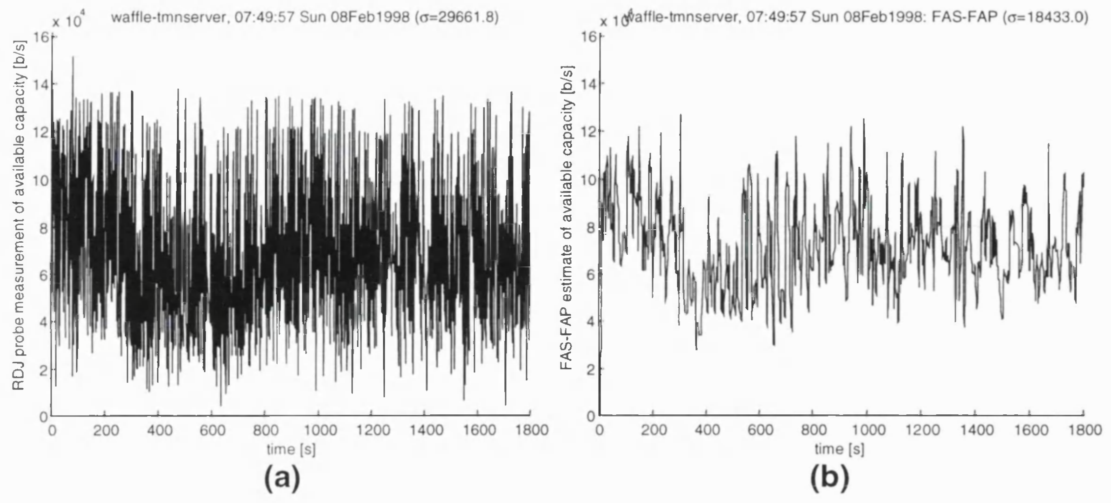


Figure 5.20: FAS-FAP (K = 2) with RDJ probes: `waffle - tmnserver`

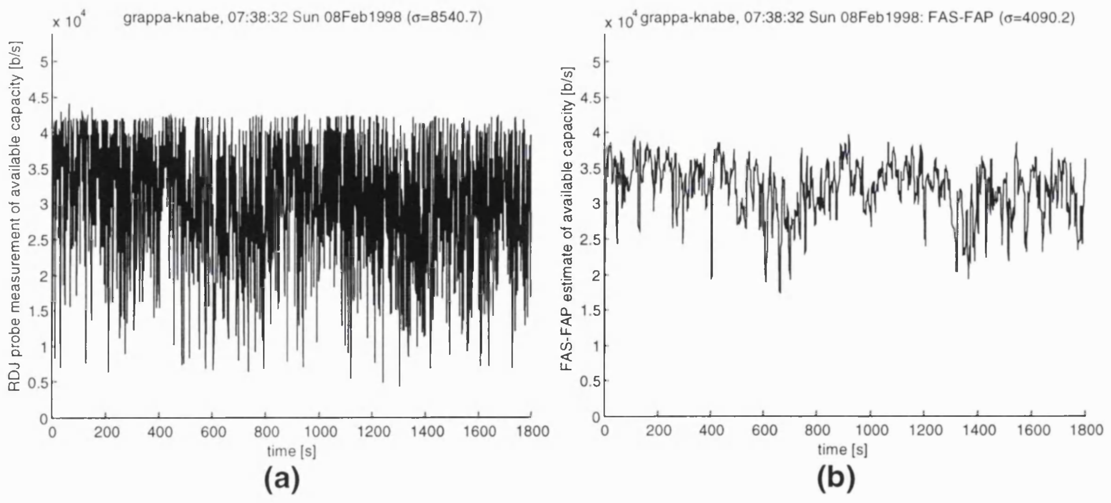


Figure 5.21: FAS-FAP (K = 2) with RDJ probes: `grappa - knabe`

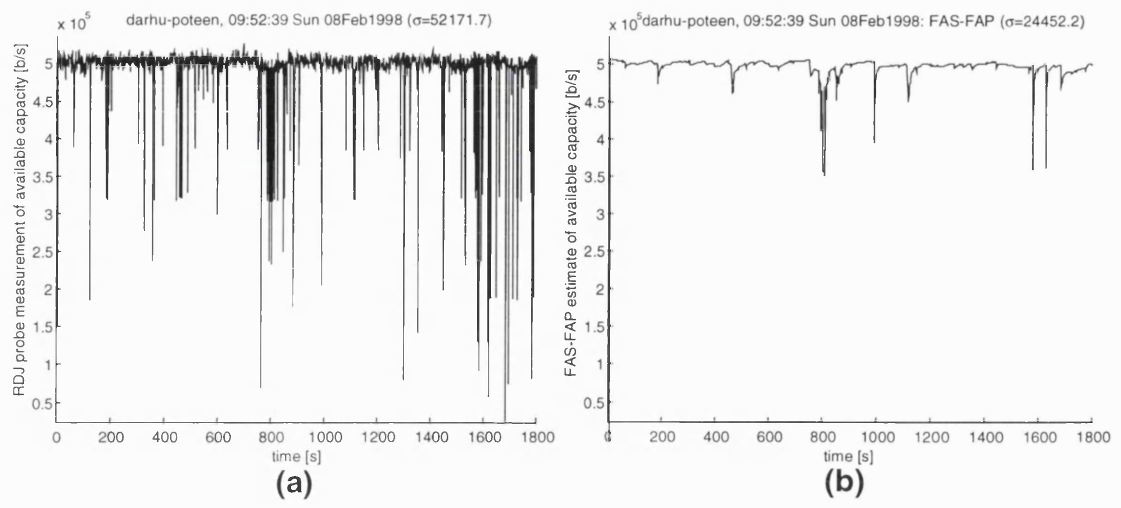


Figure 5.22: FAS-FAP ($K = 2$) with RDJ probes: `darhu - poteen`[30]

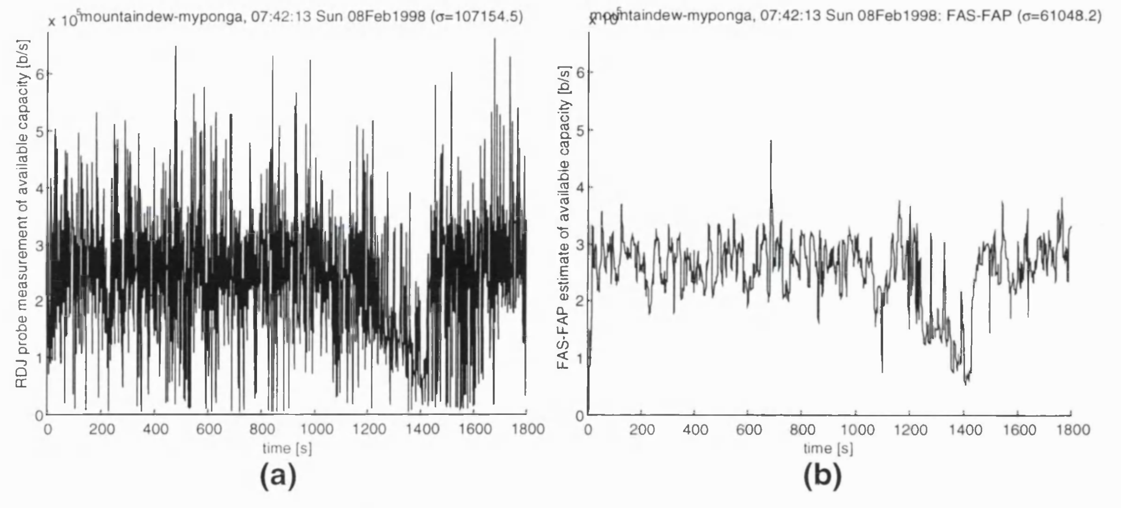


Figure 5.23: FAS-FAP ($K = 2$) with RDJ probes: `mountaindew - myponga`

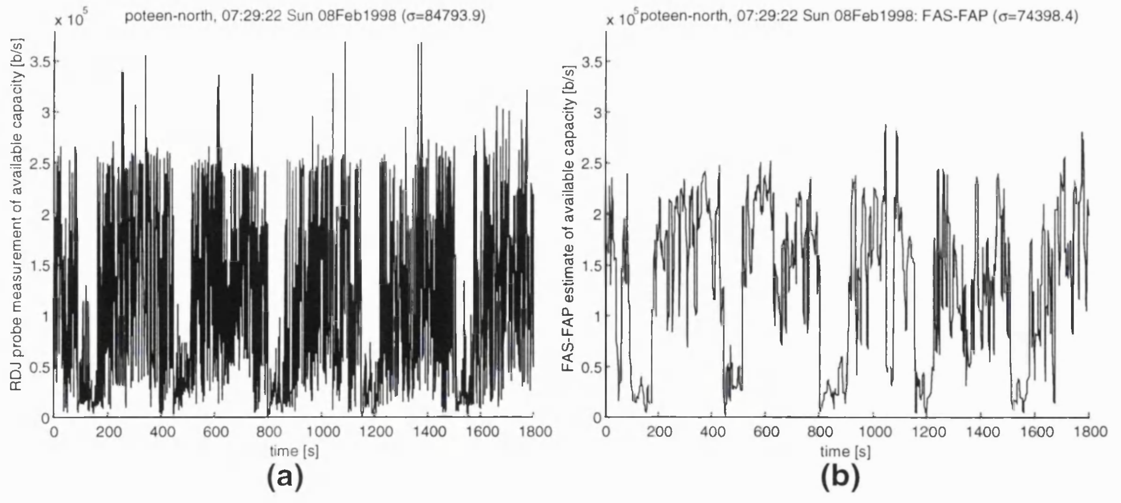


Figure 5.24: FAS-FAP ($K = 2$) with RDJ probes: `poteen - north`

[30] Note that this shows a maximum of ~0.5Mb/s for 10BaseT! This is due to the clock granularity of the host on which the measurements were taken – see Appendix A for more details.

5.6 Discussion

There is a cost associated with use of the FAS; the FAS-FAP estimator is delayed by K measurement intervals (time units) when responding to large, sudden changes. We do not see this as a significant problem for the following reasons:

- we expect K will typically be small, e.g. $1 \leq K \leq 5$, and we have found $K = 2$ appears to give reasonable results with the noisy RDJ probe measurements (some examples are shown in Figure 5.19 to Figure 5.24 – see Appendix A for further information). The absolute time for the FAS-FAP to respond will be Kt_I, where t_I is the interval between measurements, and t_I will be chosen to suit the nature of the application.
- we expect that the adaptation process will occur over a different (longer) time frame to that of the measurements of the QoS parameters for the flow
- if there is a large, sudden change that is not a spike, it will still be true after K measurement intervals, so the change *will* be detected within a useful time frame

In response to a change in QoS, as well as adjusting its flow-requirements, the application may have to perform other tasks. These might include interacting with the user in order to confirm an adaptation decision or (for a distributed application) application-level synchronisation or signalling to perform a mode change. These actions will typically take longer than the Kt_I. Additionally, rapid, flow-requirement changes or mode changes may be undesirable for the user of an application, even if the application itself is functionally capable of them. So flow-requirement changes may be relatively infrequent, perhaps occurring in the order of several 10s of seconds or several minutes, and not in the order of a few seconds (or less) like the QoSParam value changes (we examine this further in Chapter 6.).

The limitations of the FAP are discussed in [KK92]. The three main drawbacks of FAS-FAP are:

1. distortion of estimation when $\tilde{p}_t$ is close to zero (FAP)
2. it is assumed that network noise is of a higher frequency than the network perturbation, i.e. change of network QoS (FAP)
3. we restrict QoSParam values to be greater than or equal to 0 (FAS)

Although we have said that the FAS-FAP estimator has relatively good immunity to noise and have shown that it seems to have good dynamics, typically we cannot quantify the noise (amplitude and frequency) that will exist in any measurements of QoS parameter values. We have shown that FAP has reasonable performance down to a signal SNR of about 12dB (~25% noise) in the steady state, and so we must assume that the noise in the measurements will not cross this threshold. We have presented an analysis in Section 5.4 that tests the FAP with different pre-defined and randomly generated waveforms, being representations of "possible" system perturbation forms. The results give us reasonable confidence that the FAS-FAP estimator will be robust and relatively accurate, but we cannot state this with absolute certainty in all cases.

The large variation in Internet data rate values is visible in our RDJ probe measurements that have been used as input to the FAS-FAP experiments. (More details can be found in Appendix A.) The RDJ probes produce particularly noisy measurements, but use of such data was intentional, allowing us to test the robustness of the FAS-FAP system. RDJ probe measurements were used because they are easy to produce (as they use ICMP ECHO packets) and they can be seen as representing a worst-case measurment process. Typically, an application will not be expected to use RDJ probes and the measurement process used by a particular application is likely to be more accurate.

The QoSSpace is designed to be modular in support of R5 and R6. The FAS-FAP estimator can be replaced by another estimator, as long as the NetQoSRegion values can still be generated. This will not alter the abstraction seen by the application via the interface I_{aq}. We require our estimation mechanism to be lightweight and so we do not use a more complex (and perhaps more accurate) estimation/modelling mechanism.

Indeed, the modelling of traffic is currently of great concern within the data communications and telecommunications community, as there are currently some difficulties in obtaining accurate estimation and modelling. Traffic estimation and modelling techniques have traditionally relied on the Poisson model, but this has been shown to be highly inaccurate [PF95]. It is now suggested that traffic has a self-similar pattern [WTSW95], and can have long range dependencies (LRDs) that are hard to model [GB96]. In [WP97] (a discussion paper for [Res97]), the authors note the following:

- there is a *"prevalence of the infinite variance phenomenon"* in measurements of network-related traffic i.e. measurements typically show heavy-tailed distributions and exhibit a long range dependency

- *"... the traditional theoretical framework based on Poisson assumptions ... lies in shambles"*
- current understanding of LRD models are only a *"good approximation if nothing more"*

Considering Internet end-to-end packet dynamics, in [Pax97a] the author lists among his conclusions:

- *"We find wide ranges of behaviour, such that we must exercise great caution in regarding any aspect of packet dynamics as 'typical.' "*
- *"Some common assumptions such as in-order packet delivery, FIFO bottleneck queuing, independent loss events, single congestion time scales and path symmetries are violated, sometimes frequently."*

A general overview of the difficulties in estimation and simulation of traffic behaviour in the Internet is given in [PF97]. So, as well as continuing to develop better statistical models, the research community has been actively seeking measurement-based methods using alternative models to overcome such difficulties. Much of the relevant work has been focused around connection admission control (CAC), and is based on empirical techniques using adaptive estimation and/or measurements, e.g. [INTSERV] and [C+97]. Other approaches to measurement-based estimation in communication systems use fuzzy logic, e.g. [CKL95, BG94, CC94]. Fuzzy logic allows semantic tags – **fuzzy variables** – to be attached to well-defined regions of the parameter space and solution space. Fuzzy assertions test the compatibility of measurements with those regions. The fuzzy variables are chosen to reflect regions of interest in the parameter values or the solution space. The interactions of the fuzzy regions with a set of assertions effectively defines the characteristic function of the overall system using linguistic constructs without losing precision (see Appendix B).

So, we have chosen to use a well-known and widely used, measurement-based estimation mechanism, the EWMA, as more complex models may add significant additional computational cost. However, we have used measurement-based adaptation techniques to adjust the EWMA behaviour dynamically.

5.7 Chapter summary

The QoSEngine requires a back-end to generate NetQoSRegion values, *p_p* and *v_p*. The QoSEngine back-end should be an adaptive, general, and robust mechanism in order to deal with a wide range of QoS parameter value behaviour. (Section 5.1.)

A fuzzy adaptive estimator (FAP) is described in [KK92]. This is an adaptive and general mechanism based on the use of exponentially weighted moving average (EWMA), so is simple and robust. However, it is susceptible to large errors if large spikes are introduced to its input. (Section 5.2.)

A fuzzy adaptive smoother (FAS) can be used as a pre-filter to the FAP. The FAS can de-spike the input. The FAS mechanism is tuneable to spikes of any duration, K, at the cost of lack of response of K measurement intervals (time units) to large, sudden, changes in the measured values. The FAS system can be implemented efficiently (using tables) and is relatively simple in terms of computational complexity. (Section 5.3.)

The FAP system has reasonable adaptation dynamics. It performs well in fairly noisy conditions and can cope with random fluctuations in the system perturbation as well as with random noise. It appears to show good adaptive behaviour when faced with a set of measurements from pre-defined waveforms as well as with randomly generated waveforms in the presence of noise. It is able to tolerate relatively high levels of noise in the measurements and still provide reasonably accurate output. However, there may be some residual noise, even with the FAS pre-filter. (Section 5.4.)

The FAS-FAP can easily be used as a QoSEngine back-end. (Section 5.5.)

The FAS-FAP with $K = 2$, seems to provide a usable, general, robust and practicable mechanism for estimation. From our qualitative assessment using RDJ probes (which produce particularly noisy measurements), we see that the FAS-FAP system seems to perform well. We expect that the delay in the FAS-FAP response due to K will not be a problem in real applications as long as the time interval between measurements, t_I, remains greater than the frequency desired for adaptations in the application flow-requirements and (anticipated) frequency of changes in network QoS. Although more accurate estimation mechanisms may be available they are likely to be much more computationally expensive. (Section 5.6.)

6. Enabling dynamically adaptable applications

In Chapter 4, we defined the QoSSpace, a general network abstraction, and the part of the QoSEngine that can generate QoSReports. In Chapter 5, we described the back-end to the QoSEngine, an adaptive, robust and general parameter value processing function that can be used to provide the input necessary for the QoSEngine PCVF (parameter compatibility value function). In this chapter, we describe how the QoSEngine is used in an example application in order to allow dynamic adaptation.

We introduce a (fictitious) **dynamically adaptable audio tool (*daat*)** to show how the QoSEngine can be used. *daat* is based heavily on the existing capabilities in audio tools such as *vat* [vat] and *rat* [HSK98]. We construct an example **application adaptation function (AAF)** based on simple rules from user preferences. We assume that the *daat* is operating over a best-effort service as offered by a network running IP.

6.1 The problem

We have an abstraction of the network QoS, the QoSSpace, in which application flow-requirements (QoSRegions) and network QoS (NetQoSRegion) can interact. The QoSEngine abstraction provides QoSReports containing RCVs (region compatibility values) as an indication of the network's ability to support a particular QoSRegion for a flow. In this chapter, we answer the following two questions:

1. How are RCVs used by the application in order to allow dynamic adaptation?
2. What interactions might exist with the user to enable dynamic adaptation?

In Figure 6.1, we show a simplified version of Figure 1.3, highlighting the area of work considered in this chapter (dashed box).

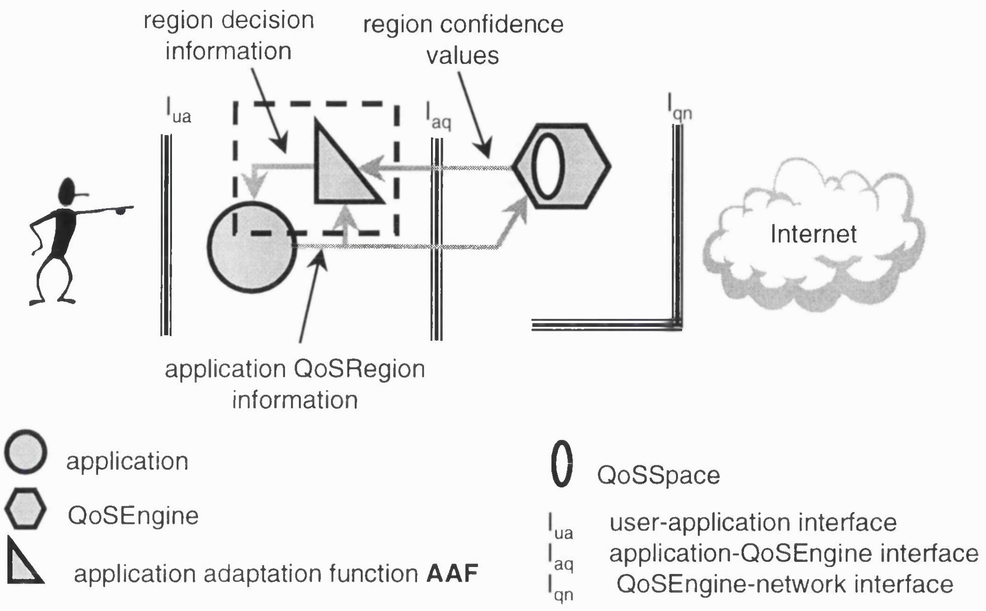

region decision information
region confidence values
I_{ua}
I_{aq}
I_{qn}
Internet
application QoSRegion information
application
QoSEngine
application adaptation function **AAF**
QoSSpace
I_{ua} user-application interface
I_{aq} application-QoSEngine interface
I_{qn} QoSEngine-network interface

Figure 6.1: The application adaptation function (AAF)

We consider the **application adaptation function (AAF)** and the way it is used to indicate possible QoSRegion (i.e. flow-requirement) changes to an application. Note that in the simple diagram of Figure 6.1, the AAF is concerned only with RCV information and no other information that might be related to flow-requirement changes or changes in the application mode; we assume that a QoSRegion has a 1:1 mapping with an application mode. However, this does not have to be the case and the QoSEngine and the AAF do not impose such restrictions in general. We show in our examples how simple user preferences can also be incorporated into the AAF.

6.2 An example application – *daat* (dynamically adaptable audio tool)

To demonstrate the use of the QoSSpace, we describe an audio tool that can adapt its audio flow data rate in response to information about data rate availability for that flow. It does this by changing the audio encoding it uses. (RTP extensions for enabling multiple audio encodings within audio flows are given in [RFC2198].) We will refer to our example audio tool as ***daat*** **(dynamically adaptable audio tool)**. *daat* is modelled on information presented in [BV96] for an audio tool developed at the Department of Computer Science, University College London [HSK98]. [HSHW95] shows that mixing audio encodings in an audio flow provides usable quality audio streams for Internet-wide use. *daat* is capable of the voice encoding schemes shown in Table 6.1, taken from [BV96].

encoding name	Data rate of flow [Kb/s]
PCM	64.0
ADM6	48.0
ADM4	32.0
ADM2	16.0
GSM	13.0
LPC	4.8

Table 6.1: ***daat*** **audio encoding schemes**

In Table 6.1, the second column is the minimum data rate requirement to allow the audio flow for the encoding scheme in the first column[31]. We can immediately define QoSRegions for *daat* using (4.11) and a single QoSParam for data rate, *R* (units Kb/s):

$$\begin{array}{ll} \langle pcm, & \langle R,64.0,-,-,-\rangle\rangle \\ \langle adm6, & \langle R,48.0,-,-,-\rangle\rangle \\ \langle adm4, & \langle R,32.0,-,-,-\rangle\rangle \\ \langle adm2, & \langle R,16.0,-,-,-\rangle\rangle \\ \langle gsm, & \langle R,13.0,-,-,-\rangle\rangle \\ \langle lpc, & \langle R,4.8,-,-,-\rangle\rangle \end{array} \qquad \textbf{(6.1)}$$

6.3 User preferences: adaptation policy

In this work, we do not investigate the interaction with the user in detail, but we introduce user preferences for *daat* in order to demonstrate the use of the QoSSpace. The user preferences are based on three simple rules to control the adaptation of the application:

P1. always use the "best" quality voice encoding possible

P2. do not change to a "better quality" voice encoding unless there is at least an average of 80% compatibility with the network QoS

P3. changes to a "better" quality encoding are to occur no more frequently than once a minute

These simple rules represent the minimum requirement for a dynamic adaptation capability based on the QoSEngine. However, they may not all need to be specified by the user, as we shall discuss. The rules cover two aspects of the operation of the application:

[31] For the purposes of the examples in this Chapter, we chose to ignore any protocol overhead due to link-layer framing, IP, UDP or RTP, etc.

1. **QoS:** P1 is a qualitative expression of the QoS required and P2 denotes how "strict" the application should be in interpreting "best" in P1
2. **stability:** P3 places a stabilising constraint on the frequency of QoSRegion changes that may occur for the application instance

The rules P1, P2 and P3 can be seen as a user-to-application QoS mapping, and comprise the user control mechanism for the adaptation decision-making process. We interpret "best" in rule P1 as meaning "with the highest flow data rate possible", i.e. requiring more network resources. The QoSEngine has no knowledge of the inter-QoSRegion relationships or the semantics of the flow, so cannot judge "best" or "better" – this must be done by the application. Another method of assessing "best" might be by the application designer simply enumerating the application modes and the user assigning simple priority values to each mode (e.g. a number between 1 and 10; 1 low priority, 10 high priority), so denoting user preferences.

The interpretation of rules P2 and P3 is how tolerant the user might be to QoSRegion or application mode changes and so a stability period of one minute is specified. Note that the intention of this constraint is actually to control the rate at which an application might move to a "better" quality QoSRegion, as degradation in QoS should be supported by an immediate move to a "lower" quality QoSRegion that can be supported. The compatibility level expressed in P2 is important – it represents the assurance that the user and/or application requires for QoSRegion changes, but can be mapped to the same numeric range as RCVs, acting as a threshold for RCVs in decision-making.

Effectively, P1 and P2 govern *how* adaptation is to occur and P2 and P3 govern *when* adaptation is to occur.

6.4 An application adaptation function (AAF) for *daat*

We will assume that the *daat* application is capable of changing its QoSRegion once every second (the same rate at which we have taken measurements of R). The user does not want the application to change QoSRegion every second, so must control how adaptation takes place through the user preferences. The user preferences must be translated to a QoS-based adaptation policy by the application.

So, for the *daat*, we define an **application adaptation function (AAF)** which will provide a value of the QoSRegion identifier (*qr_id*) to indicate the QoSRegion that the audio flow should be in at any time interval. The algorithm for the AAF is based on the three rules P1, P2 and P3. For P1, we number the QoSRegions using the simple mapping {*qr_id* ⇔ number} as follows:

{lpc ⇔ 1}	{gsm ⇔ 2}	{adm2 ⇔ 3}
{adm4 ⇔ 4}	{adm6 ⇔ 5}	{pcm ⇔ 6}

i.e. with a higher number used for identifying a "better" quality QoSRegion. This is effectively a priority assignment for the application modes based on the user preferences, i.e. a QoS mapping from user to application. We also use the following definitions:

q_compatibility	from rule P2; value 0.8
q_time	from rule P3; the QoSRegion stability time [seconds], value 60s
Q_SCORE(s, n, H)	evaluation of (6.2) for QoSRegion s, where $s = 1 \ldots 6$

$$\frac{1}{H}\sum_{h=0}^{H-1}\text{BOOLEAN}(RCV^{s}_{n-h} \geq q\_compatibility) \qquad \textbf{(6.2)}$$

	BOOLEAN(X) returns 1 if X is true and 0 if X is false; RCV^{s}_{n-h} is the *RCV* for QoSRegion s at time (n-h)
n	the nth time unit
scv(s, n)	RCV for QoSRegion s at time n
sdi(n)	the RDI (QoSRegion decision information) is a value of s identifying the QoSRegion that the application is in at time n
q_n_time	the value INTEGER(*q_time*/t_i), where t_i is the interval at which measurements are taken [seconds]
q_epoch	the previous value of n at which there was a QoSRegion change

The AAF algorithm is described in pseudo-code in Figure 6.2 and a schematic diagram of the QoS information flows is shown in Figure 6.3.

For the *daat*, the algorithm performs the following function:

1. if the application has just started ($n < q\_n\_time$), then choose the highest quality (highest numbered) QoSRegion with the highest Q_SCORE
2. if the application has been underway for sometime ($n \geq q\_n\_time$):
   a. if we are within one minute (*q_n_time*) of the last QoSRegion change (*q_epoch*), do not change QoSRegion unless the current QoSRegion is no longer usable

b. if we are over one minute (*q_n_time*) since the last QoSRegion change (*q_epoch*), check if it is possible to move to another QoSRegion

We use the *daat* QoSRegions with real network data gathered using RDJ probes. We will use the measurements shown in Figure 5.19(b) and Figure 5.20(b) as values for the capacity available to the audio flow, i.e. the values for the QoSParam *R*. When we use the *daat* QoSRegions in (6.1) with these measurements, we get the RCVs shown in Figure 6.4(a) and Figure 6.5(a). The results of applying the AAF to these are shown in Figure 6.4(b) and Figure 6.5(b), respectively, which record the QoSRegion that the *daat* takes. The graphs in Figure 6.4(c) and Figure 6.5(c) have been marked with the flow rate from the application based on these QoSRegions, using the data rate values from Table 6.1. The RCVs show the variability of QoS in the network. (We have chosen one set of network measurements with low variability, `darhu – theakston`, and one set of measurements with high variability, `waffle - tmnserver`).

Note that the Q_SCORE in (6.2) is a mean and so helps to smooth small disturbances in the RCV (e.g. residual noise from the QoS parameter measurements that has passed through the PCVF). For example, some of the spikes in the RCV for *adm6* in Figure 6.4(a) and Figure 6.5(a) are ignored by the AAF as shown in Figure 6.4(b) and Figure 6.5(b). However, a much more cautious version of the AAF might use the following definition of Q_SCORE in place of (6.2):

$$\text{BOOLEAN}(\text{MIN}(RCV^{s}_{n},\ldots,RCV^{s}_{n-(H-1)}) \geq q\_compatibility) \qquad \textbf{(6.3)}$$

This would greatly amplify the presence of a single low RCV. In both (6.2) and (6.3), as the value of *q_compatibility* gets smaller, so there may be larger Q_SCOREs for more QoSRegions, even if they have relatively low compatibility. In our simulations, we have found through observation that $q\_compatibility = 0.8$ provides reasonable results, reflecting an acceptable threshold between stability with respect to the number of QoSRegion changes and utilisation of the available network resources. We have said in Section 6.3 that the *q_compatibility* value might be provided by the user. However, we shall see that the value of *q_compatibility* could change operation of the application quite significantly, so, for certain applications, there may be a strong case for moving the value of *q_compatibility* (rule P2) out of the user's control and into the application. This may depend, for example, on the QoS sensitivity of the media flow, e.g. high quality video would require better QoS compatibility than the relatively low quality audio offered by *daat*, because human perception of quality is more tolerant of low quality in audio than in video.

```
if n < q_n_time                        // just started ..

  q_score_hi = 0,  q_s = 0
  for s = 1 to S                       // higher regions override lower ones
    q_s = Q_SCORE(s,n,n)
    if (q_s >= q_score_hi) &
        (q_s >= q_compatibility)   // so use region with highest score
      rdi(n)      = s
      q_score_hi = q_s
      q_epoch    = n
    endif
  endfor

else                                   // has been going for some time ..

  if rdi(n-1) > 0                      // and was previously in a region
    q_rcv = Q_SCORE(rdi(n-1),n,q_n_time)
  else
    q_rcv = 0;
  endif

  q_n = n - q_epoch
  if (q_n <= q_n_time) &               // if within epoch and ..
      (q _rcv >= q_compatibility)  // region is still usable ..
    rdi(n) = rdi(n-1)                  // stay in present region

  else                                 // else check for region change
    for s = 1 to S                     // higher regions override lower ones
      q_rcv = Q_SCORE(s,n,q_n_time)
      if q_rcv >= q_compatibility      // Q_SCORE OK for this region so ..
        rdi(n) = s                     // use this region
      endif
    endfor

    if rdi(n) <> rdi(n-1)              // region changed so mark new epoch
      q_epoch = n
    endif

  endif

endif
```

Figure 6.2: Simple AAF algorithm for *daat* in pseudo-code

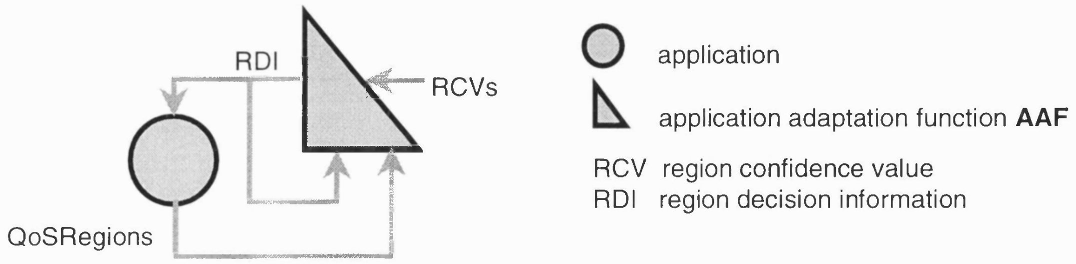

RDI
RCVs
QoSRegions
application
application adaptation function AAF
RCV region confidence value
RDI region decision information

Figure 6.3: QoS information flows between the simple AAF and the *daat* application

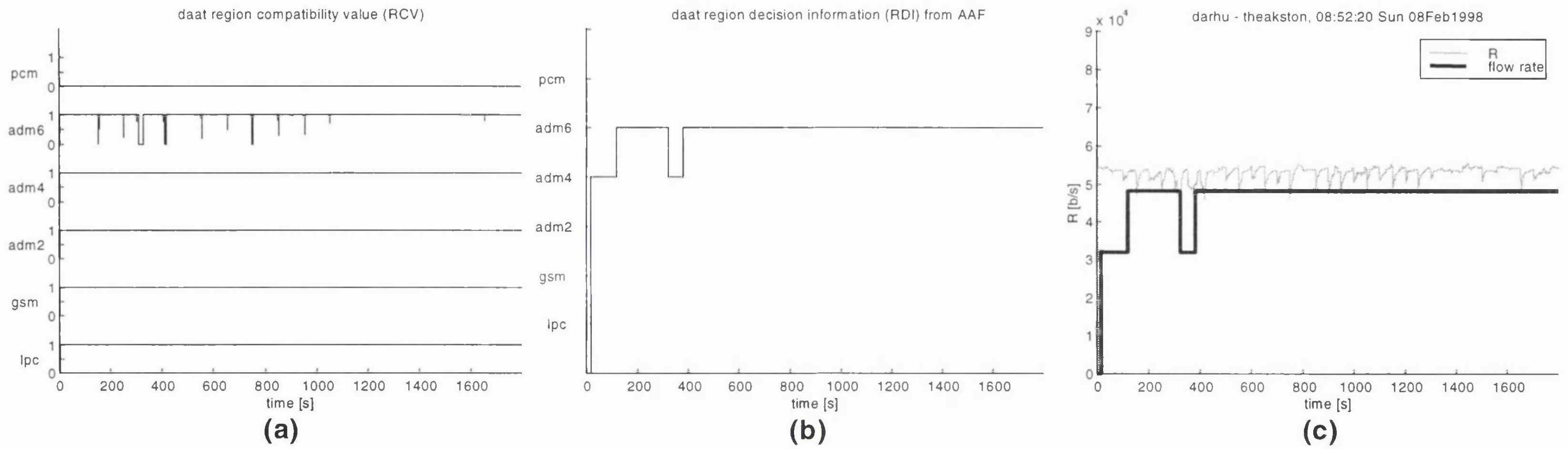


Figure 6.4: *daat* **with AAF using** `darhu - theakston` **data,** *q_compatibility* **= 0.8,** *q_time* **= 60s**

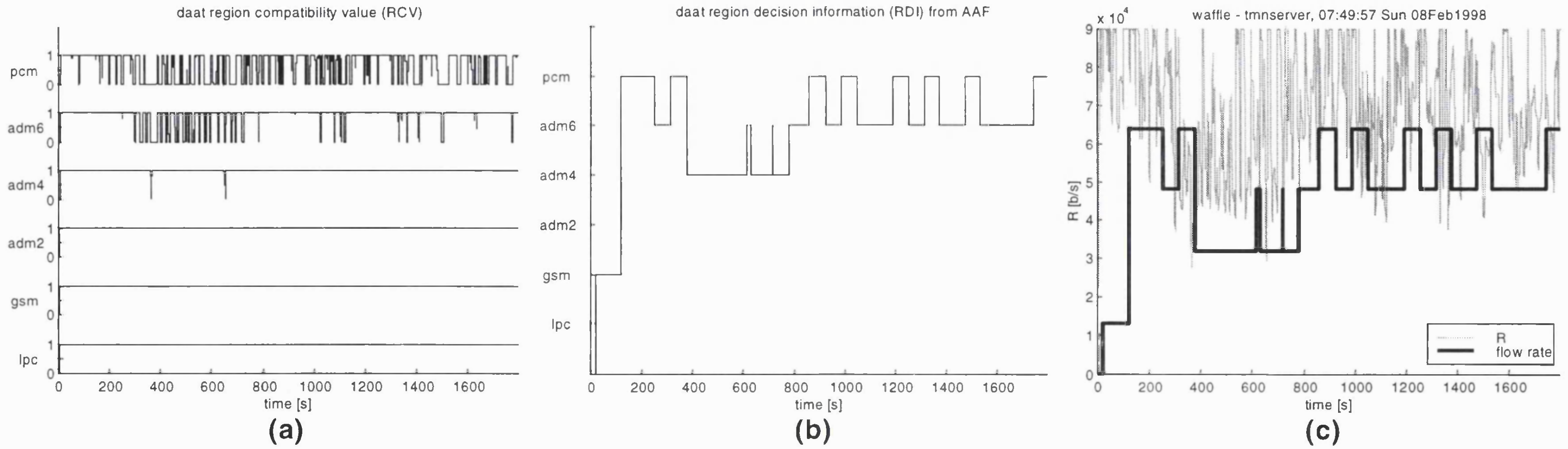


Figure 6.5: *daat* **with AAF using** `waffle - tmnserver` **data,** *q_compatibility* **= 0.8,** *q_time* **= 60s**

6.4.1 Effects of varying values for q_time and q_compatibility

As we change the values of *q_time* and *q_compatibility*, we observe the following effects:

- varying *q_time* results in greater stability of the application for the user (with respect to QoSRegion changes), but the application may:
    - stay longer in "lower" quality QoSRegions, not using available resources
    - the smoothing offered by larger *q_time* values may mask real deviations from QoS in the RCV, resulting in delayed response to QoS changes, which may be particularly critical if the QoS is degraded
- increasing the value of *q_compatibility* results in a harsher discrimination of a QoSRegion's suitability for use, and may result in under utilisation of available network resources

We see from Figure 6.5(c) ($600 < t < 800$), how changes to better quality QoSRegions are controlled to the *q_time* = 60s time period, but how changes to lower quality QoSRegions occur as soon as an inadequate QoS provision is detected. The effects of using *q_time* = 90s, *q_time* = 120s and *q_time* = 180s for R in Figure 6.4(a) and Figure 6.5(a) are shown in Figure 6.6 and Figure 6.8, respectively. We can see how the application achieves greater stability (with respect to QoSRegion changes) in the face of network QoS variability as *q_time* is increased, notably in Figure 6.8. The respective flow rates for Figure 6.6 and Figure 6.8 are shown in Figure 6.7 and Figure 6.9. The *q_time* value could be adjusted dynamically by the user (or automatically by the application), as required, and acts as a form of application-level smoothing for QoSRegion changes.

As we increase the value of *q_compatibility*, we find that the application takes longer to enter better quality QoSRegions in the face of variability in the value of R. This is shown in Figure 6.10, Figure 6.11, Figure 6.12 and Figure 6.13 for our two data sets, with *q_compatibility* = 0.85, *q_compatibility* = 0.9 and *q_compatibility* = 0.95. This is to be expected: as we approach a value of 1.0 for *q_compatibility*, we are asking for the RCVs to be totally stable with respect to the QoSRegion. We know from (4.5) that this is only possible if *p_p* is within the QoSRegion region and either the variability, *v_p*, of the QoSParam values is zero, or the variability is completely confined within the QoSRegion region. We can see from Figure 6.4 and Figure 6.5 that a value of *q_compatibility* = 0.8 seems to offer an adaptation capability that tracks the value of R well.

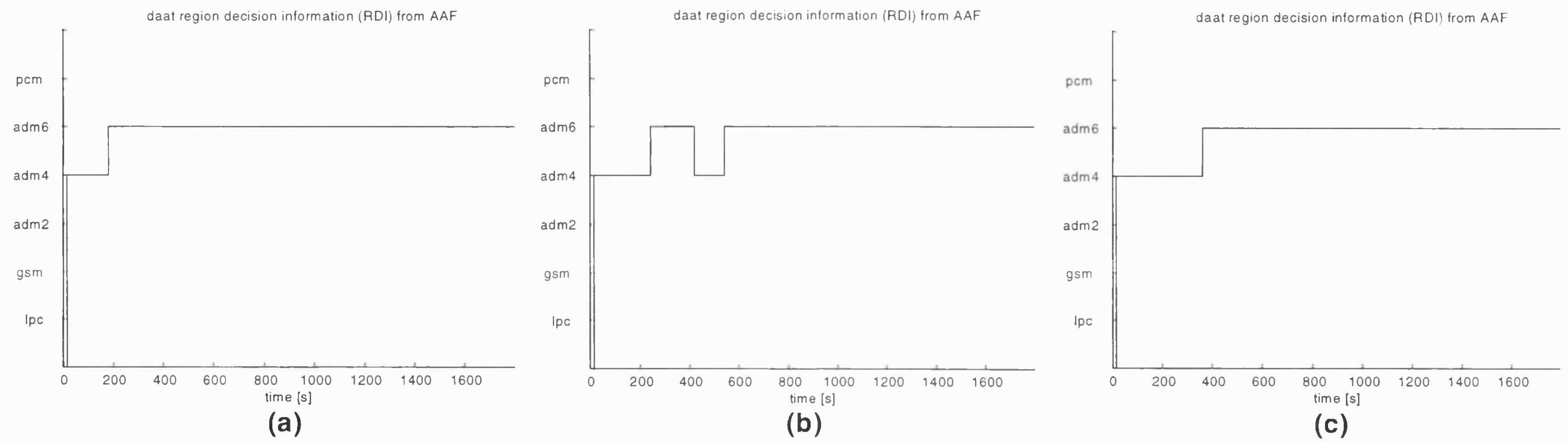


Figure 6.6: RDI for *daat* with AAF using `darhu - theakston` data with *q_compatibility* = 0.8; (a) *q_time* = 90s; (b) *q_time* = 120s; (c) *q_time* = 180s

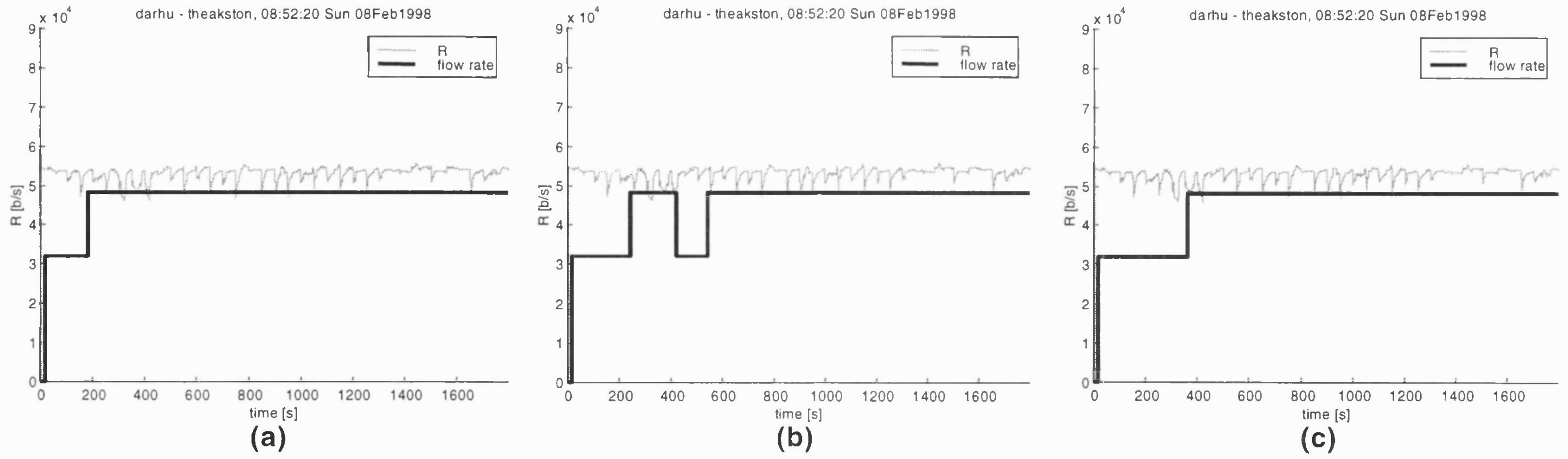


Figure 6.7: Flow rate for *daat* with AAF using `darhu - theakston` data with *q_compatibility* = 0.8; (a) *q_time* = 90s; (b) *q_time* = 120s; (c) *q_time* = 180s

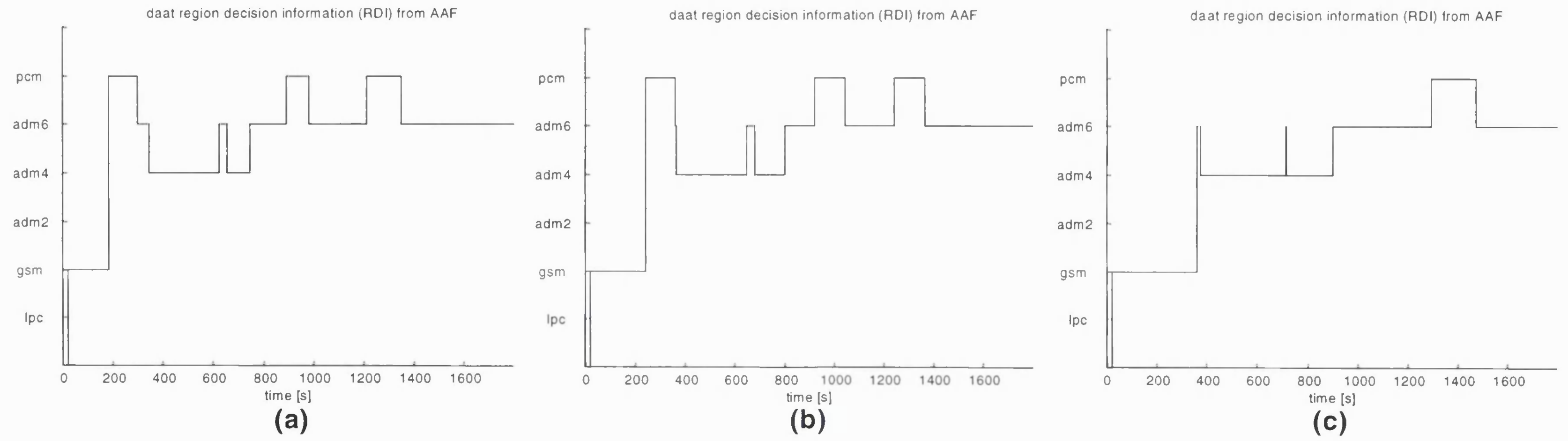


Figure 6.8: RDI for *daat* with AAF using `waffle` - `tmnserver` data with *q_compatibility* = 0.8; (a) *q_time* = 90s; (b) *q_time* = 120s; (c) *q_time* = 180s

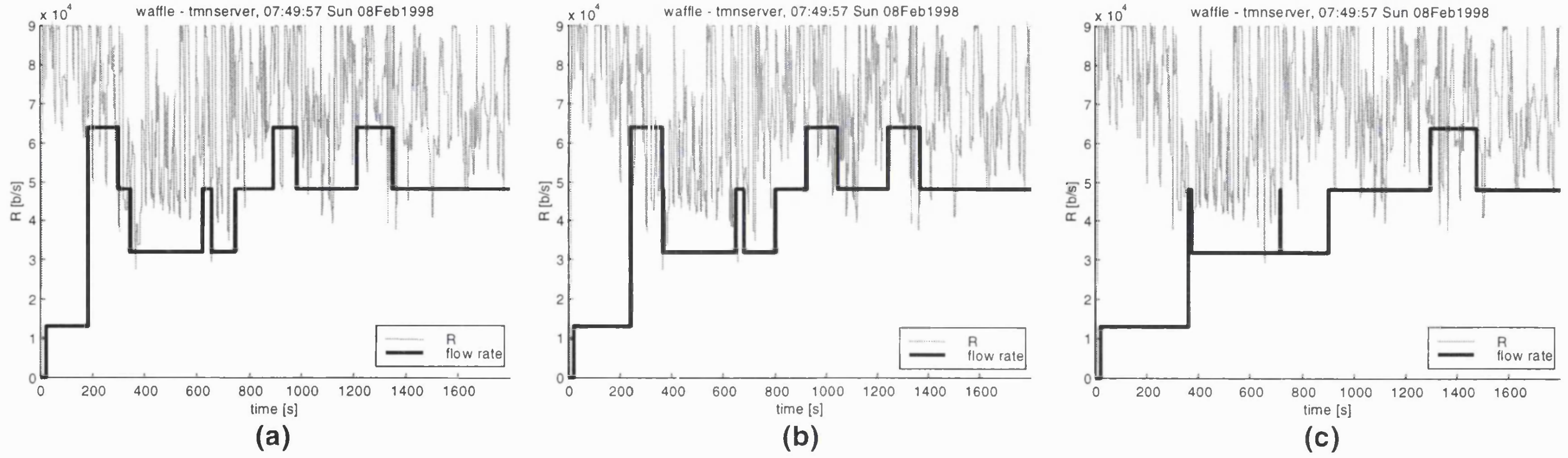


Figure 6.9: Flow rate for *daat* with AAF using `waffle` - `tmnserver` data with *q_compatibility* = 0.8; (a) *q_time* = 90s; (b) *q_time* = 120s; (c) *q_time* = 180s

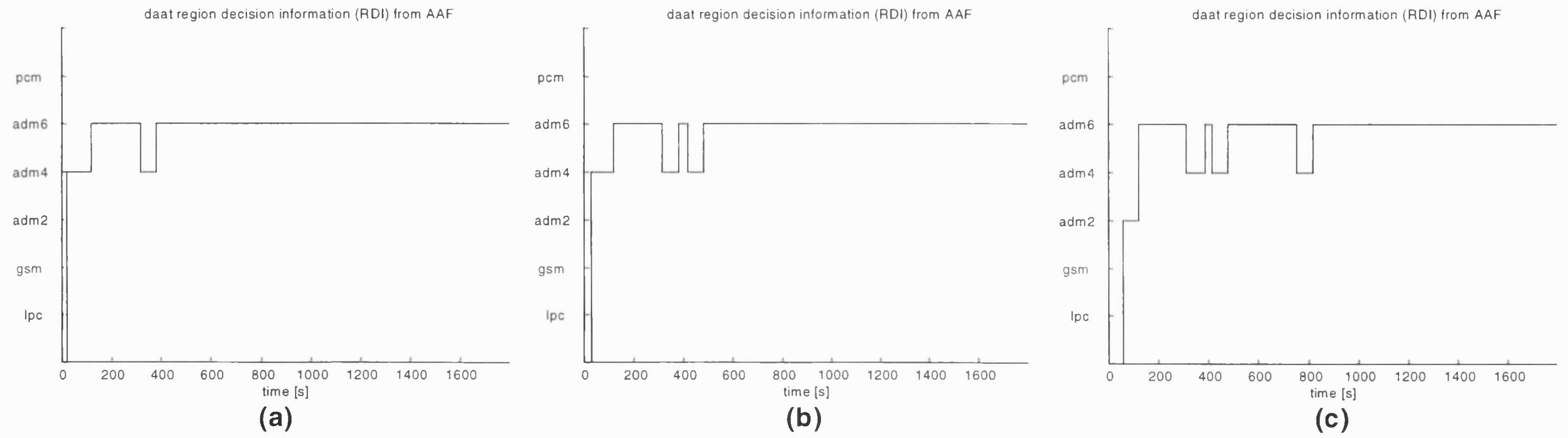


Figure 6.10: RDI for *daat* with AAF using `darhu - theakston` data with *q_time* = 60; (a) *q_compatibility* = 0.85; (b) *q_compatibility* = 0.9; (c) *q_compatibility* = 0.95s

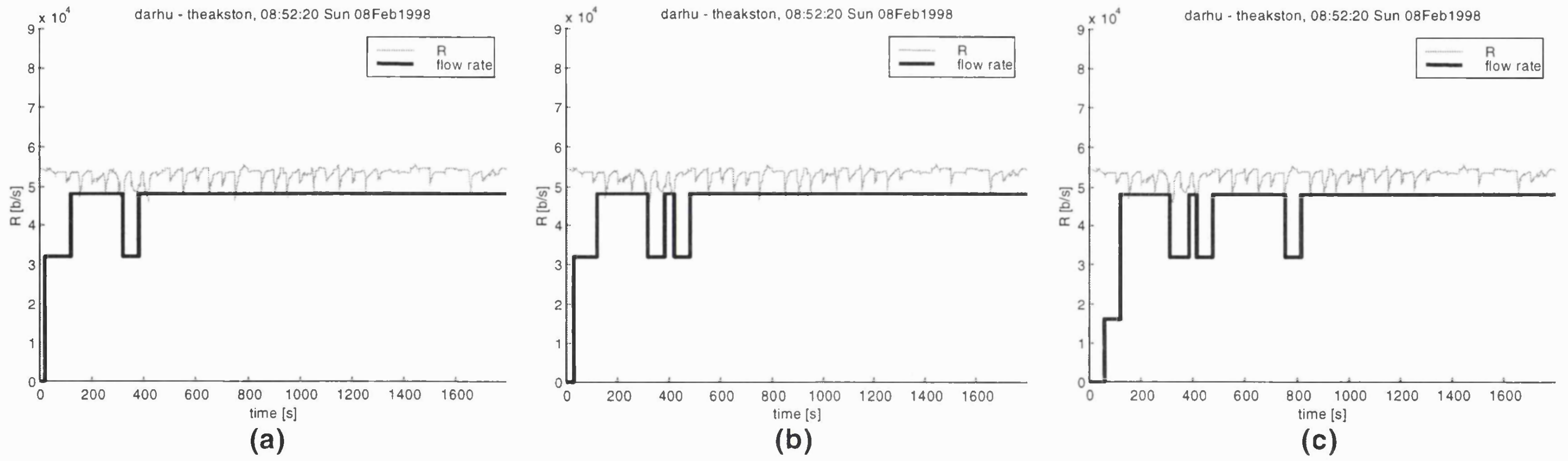


Figure 6.11: Flow rate for *daat* with AAF using `darhu - theakston` data with *q_time* = 60; (a) *q_compatibility* = 0.85; (b) *q_compatibility* = 0.9; (c) *q_compatibility* = 0.95s

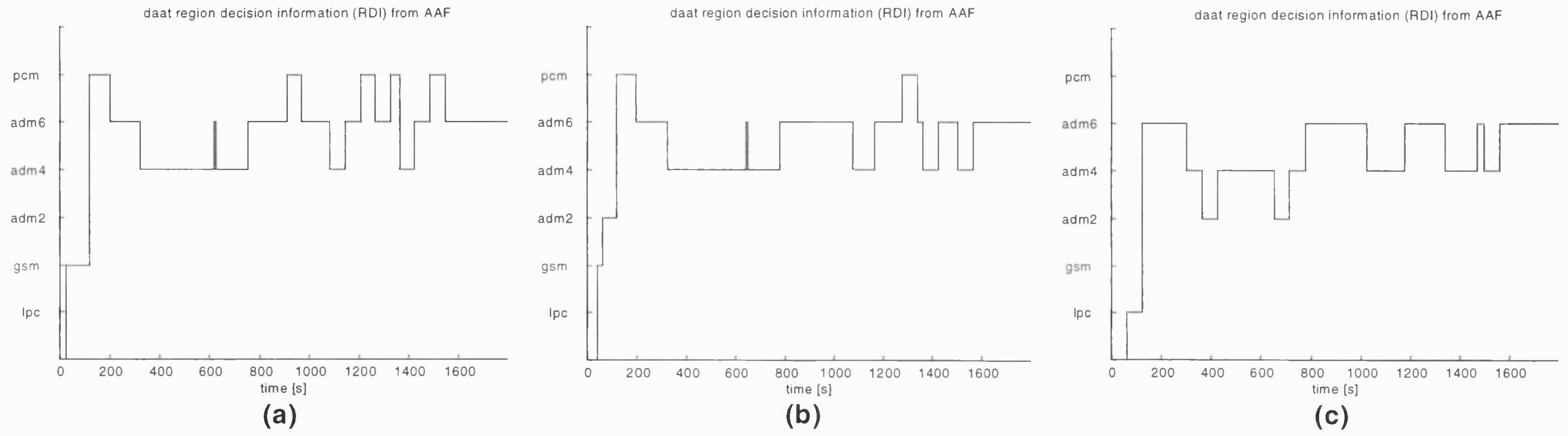


Figure 6.12: RDI for *daat* with AAF using `waffle` - `tmnserver` data with *q_time* = 60; (a) *q_compatibility* = 0.85; (b) *q_compatibility* = 0.9; (c) *q_compatibility* = 0.95

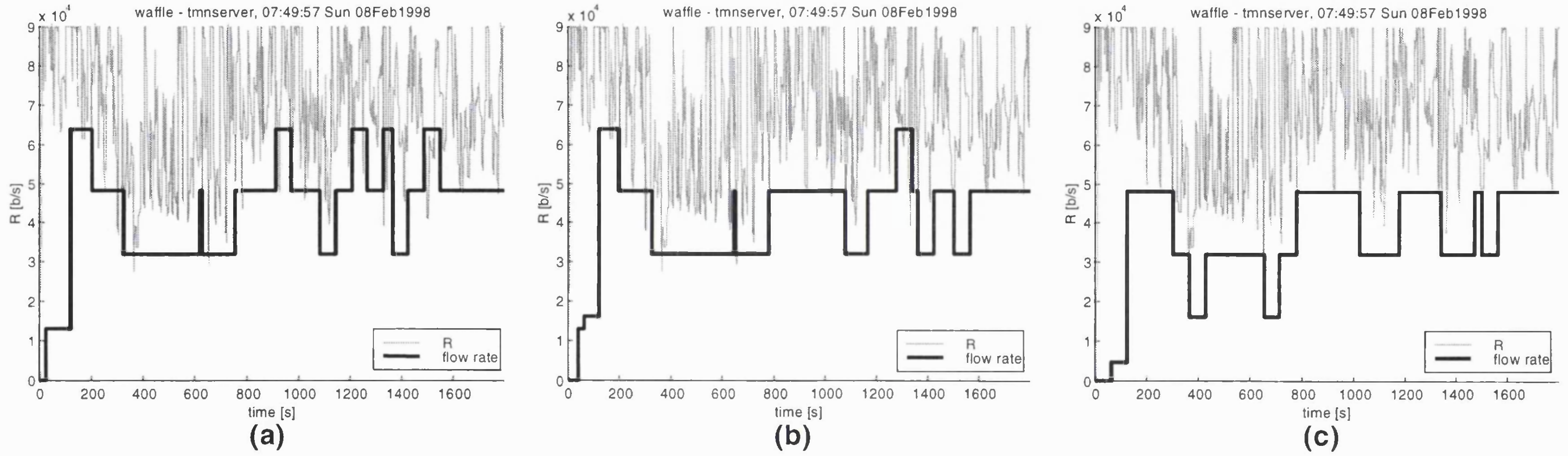


Figure 6.13: Flow rate for *daat* with AAF using `waffle` - `tmnserver` data with *q_time* = 60; (a) *q_compatibility* = 0.85; (b) *q_compatibility* = 0.9; (c) *q_compatibility* = 0.95

The judicious use of *q_time* and *q_compatibility* allow the user and the application to have considerable control over the adaptation capability of the *daat*, within the capabilities of the current network QoS.

6.5 A mobile scenario of the *daat*: power conservation demonstration

In Section 6.4, we demonstrated the use of the QoSSpace and showed the dynamics of an example AAF in order to control adaptation. We now consider a scenario with two QoSParams, in order to demonstrate the generality and flexibility that is possible with the QoSSpace for dynamic adaptation. For this, we assume our *daat* is executing on a mobile host. Here we consider that the mobile host is powered by a battery until reaching a place where mains operation is possible. During battery operation, we wish to conserve power. So, we introduce a new QoSParam, batter power, *B*, which we use to modify our QoSRegion definitions for the *daat*. The QoSParam *B* has values in the range [0, 1], 0 indicating the battery is totally spent and 1 indicating the battery is fully charged (or that the mobile host is running on mains power). We now modify Table 6.1 to generate some (artificial) power consumption figures for each of the audio encoding schemes to give Table 6.2.

encoding name	data rate of flow [Kb/s]	relative CPU cost	power cost [1/n][32]
PCM	64.0	1	0.00001
ADM6	48.0	13	0.00014
ADM4	32.0	11	0.00011
ADM2	16.0	9	0.00009
GSM	13.0	1200	0.01250
LPC	4.8	110	0.00114

Table 6.2: ***daat*** **audio encoding schemes and cost for CPU and battery**

The third column for Table 6.2 is taken from [BV96]. The fourth column is generated from the third column by dividing by 96000[33], and represents power consumption per

[32] This scenario assumes that the mobile host is in fact a palmtop PC or personal digital assistant and has no disc and a low power LCD screen. If the mobile host is a laptop computer, the most significant power consumption will be for powering the hard disc and the screen.

[33] This value is chosen simply to generate some artificial numbers suitable for our simulation.

time unit, *n*. We also use the third column of the table to generate some *p_lo* thresholds for *B* by dividing by 2400[34] and modify (6.1) to give the following QoSRegions for *daat*:

$$\begin{array}{ll}
\langle pcm, & \langle R,64.0,–,–,–\rangle\rangle \\
\langle adm6, & \langle R,48.0,–,–,–\rangle,\langle B,0.0054,–,–,–\rangle\rangle \\
\langle adm4, & \langle R,32.0,–,–,–\rangle,\langle B,0.0046,–,–,–\rangle\rangle \\
\langle adm2, & \langle R,16.0,–,–,–\rangle,\langle B,0.0038,–,–,–\rangle\rangle \\
\langle gsm, & \langle R,13.0,–,–,–\rangle,\langle B,0.5000,–,–,–\rangle\rangle \\
\langle lpc, & \langle R,4.8,–,–,–\rangle\rangle
\end{array} \qquad \textbf{(6.4)}$$

In (6.4), we deliberately choose to have the QoSRegions *lpc* and *pcm* defined only in terms of *R*, in order to demonstrate the ability of the QoSSpace and AAF to handle heterogeneity in the number of parameters used to define QoSRegions for the same flow.

We use a naïve scheme in which the *p_lo* thresholds for each QoSRegion in (6.4) mark the lowest battery power charge that is allowed before that audio encoding can be used[35]. The simulation of this scenario is shown in Figure 6.14. For clarity, in the values of *R* and *B*, we have not simulated any noise, but we have already shown that the QoSEngine can deal with noise in Chapter 5 and in our scenario in Section 6.4.

The mobile host user moves from home to work. At home our user connects via ISDN (64Kb/s, time < 20), then by mobile phone (13Kb/s, 20 ≤ time < 120) on the way to work, and finally by Ethernet at work. The user starts with a fully charged battery (time = 0) and is not connected to the mains power until time > 200. We see in Figure 6.14(a) and Figure 6.14(c) how the *daat* switches from the GSM encoding to the LPC encoding (even though the GSM rate is still achievable) when the battery power, shown in Figure 6.14(b), goes down to 0.5 (time = 80). We have used *q_time* = 1 (to allow fast adaptability to better quality QoSRegions), *q_compatibility* = 0.8 and *K* = 2.

This example shows that the battery power takes precedence in the adaptation policy only when its role becomes significant, i.e. when a QoSRegion for a boundary defined by QoSParam *B* is reached.

[34] This value is chosen simply to generate some artificial numbers suitable for our simulation.

[35] A better scheme might be for the application to perform a calibration when it is first started by using each encoding scheme to encode a buffer of data to measure the rate at which each encoding drains the battery power and then set these readings as *p_lo* thresholds for a QoSParam that is the *rate of change* in *B* rather than the absolute value of *B* itself. We chose our naïve approach in order to demonstrate better the dynamics of the AAF.

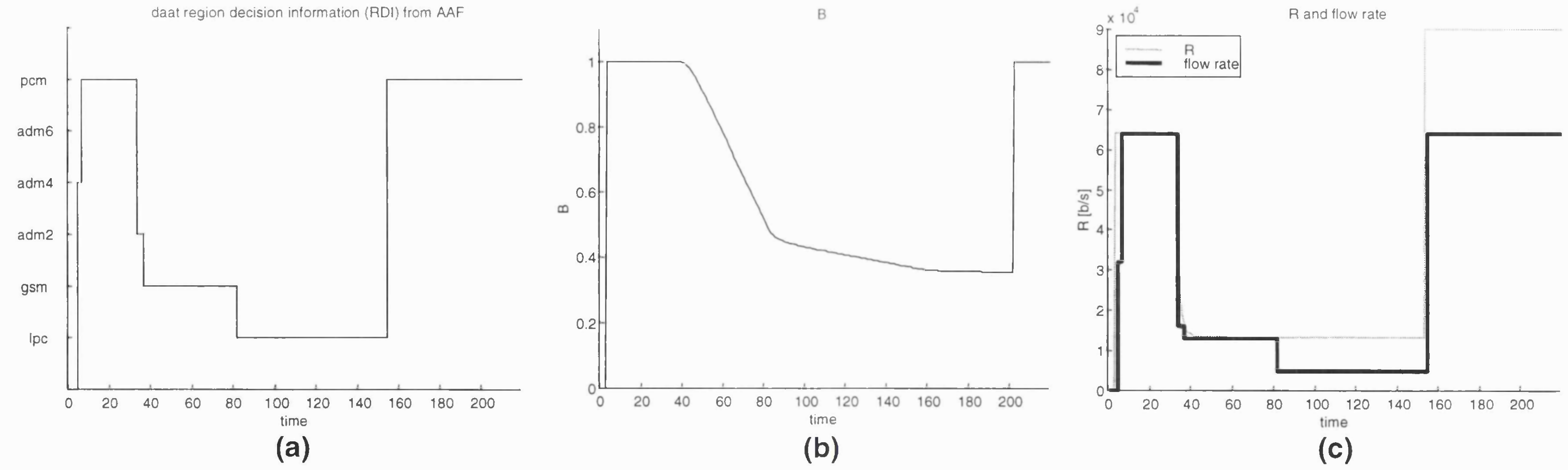


Figure 6.14: ***daat*** **power consumption in the mobile host**

6.6 Use of QoSiRegions

In (6.1) and the AAF in Figure 6.2, we have not considered the use of QoSiRegions. In using QoSiRegions we must remember that they are of a different nature than the QoSRegions which they inhabit. The QoSiRegions are effectively indicators that a QoSRegion is operating very close to one of its boundaries, so they may be interpreted as indicators of "negative compatibility". (However their exact use and interpretation will be application specific.)

In the same way as we can define a Q_SCORE for QoSRegions, we can also define a Q_ISCORE for QoSiRegions. We may use (6.2) for evaluating a Q_ISCORE with RCV_Is in place of RCVs. Indeed this would appear to be the most sensible definition for Q_ISCORE for *daat*, as we are concerned with trying to measure the suitability of a particular QoSRegion to be supportable over a given period of time, *q_time*. Consider if (6.3) were used for Q_ISCORE. This would mean that a long sequence of high RCV_I values would be ignored in the presence of a single low value, and we would lose information about the true extent of the presence of QoSParam values in the QoSiRegion. However, if we were to substitute MAX in place of MIN in (6.3) this would give precedence to a single high RCV_I (which might be due to residual noise from the QoS parameter measurements), giving a pessimistic assessment.

We show highlighted in Figure 6.15 how our AAF algorithm is modified to incorporate use of QoSiRegions. In the modified AAF of Figure 6.15, we chose to employ a policy where if the NetQoSRegion is such that the QoSRegion would be operating very close to its boundaries, we chose not to use that QoSRegion. For the *daat*, this would help to avoid state-flapping when one of the QoSParams in the QoSRegion was wavering around a *p_lo* boundary of one of the QoSRegions. This makes sense for the *daat* as all its QoSRegions have boundaries that (effectively) border on each other so it is possible to use a lower quality QoSRegion to avoid state-flapping.

We show the use of the same *q_compatibility* and *q_time* values for the Q_ISCORE as that for Q_SCORE, but different values could be used. Setting lower values for the *q_compatibility* and *q_time* for Q_ISCORE would make the AAF more sensitive to operation of a QoSRegion close to its boundary. This difference between thresholds and timescales between Q_SCORE and Q_ISCORE could, for example, be used to allow fast detection of operation close to QoSRegion boundaries, while still maintain flow stability when operation is well within QoSRegion boundaries.

```
if (n * t) < q_time

   q_score_hi = 0,  q_s = 0
   for s = 1 to S
      q_s = Q_SCORE(s,n,n)
      if (q_s >= q_score_hi) &
            (q_s >= q_compatibility)
         rdi(n)      = s
         q_score_hi = q_s
         q_epoch    = n
      endif
   endfor

else

   q_n = n - q_epoch

   if rdi(n-1) > 0
      q_rcv   = Q_SCORE(rdi(n-1),n,q_n_time)
      q_rcv_i = Q_ISCORE(rdi(n-1),n,q_n_time)
   else
      q_rcv   = 0;
      q_rcv_i = 0
   end

   if (q_n <= q_n_time) &
         (q _rcv >= q_compatibility) & (q_rcv_i < q_compatibility)
      rdi(n) = rdi(n-1)

   else
      q_n = n - q_n_time
      for s = 1 to S
         q_rcv   = Q_SCORE(s,n,q_n_time)
         q_rcv_i = Q_ISCORE(s,n,q_n_time)

         if (q_rcv >= q_compatibility) & (q_rcv_i < q_compatibility)
            rdi(n) = s
         endif
      endfor

      if rdi(n) <> rdi(n-1)
         q_epoch = n
      endif

   endif

endif
```

Figure 6.15: Simple modifications to the AAF for *daat* to use QoSiRegions (see also Figure 6.2)

We can contrive a scenario for our *daat* application that shows the modified AAF of Figure 6.15 at work. First, we generate a scenario where there is state-flapping, as shown in Figure 6.16 (*q_time* = 1 for rapid adaptability response, *q_compatibility* = 0.8, *K* = 2,). We see in Figure 6.16(a) and Figure 6.16(c) how the value of *R* wavers around the *adm6* boundary causing the *daat* to oscillate between *adm4* and *adm6*. (Again, for clarity we choose not to simulate noise for *R*.) We can introduce *_lo* QoSiRegions for our

QoSRegions of (6.1) in order to overcome this. We use a *p_qlo* value that is the value of *p_lo* +10%, giving us the following QoSRegion definitions:

$\langle pcm,$	$\langle R, 64.0, -, -, -\rangle\rangle$
$\langle adm6,$	$\langle R, 48.0, -, 52.8, -\rangle\rangle$
$\langle adm4,$	$\langle R, 32.0, -, 35.2, -\rangle\rangle$
$\langle adm2,$	$\langle R, 16.0, -, 17.6, -\rangle\rangle$
$\langle gsm,$	$\langle R, 13.0, -, 14.3, -\rangle\rangle$
$\langle lpc,$	$\langle R, 4.8, -, -, -\rangle\rangle$

(6.5)

Again, in (6.5), we deliberately choose to have the QoSRegions *lpc* and *pcm* defined without QoSiRegions, in order to demonstrate the ability of the QoSSpace and AAF to handle heterogeneity in QoSRegions that are defined using the same QoSParams.

The results of using the definitions in (6.5) and the modified AAF of Figure 6.15 are shown in Figure 6.17. We can see how the use of the QoSiRegion and the modified AAF has removed the state-flapping seen in Figure 6.16. Figure 6.16(a) shows the values of the RCVs, which are identical when used with (6.1) and (6.5), while Figure 6.17(a) shows the additional *_lo* QoSiRegion RCV_Is which are used only with (6.5). Figure 6.16(b) and Figure 6.17(b) shows the QoSRegions selected by the AAF, while Figure 6.16(c) and Figure 6.17(c) show the flow rate achieved.

Similar application-level smoothing may be achievable by using a large enough value for *q_time*, but this would then result in lack of responsiveness in adaptability.

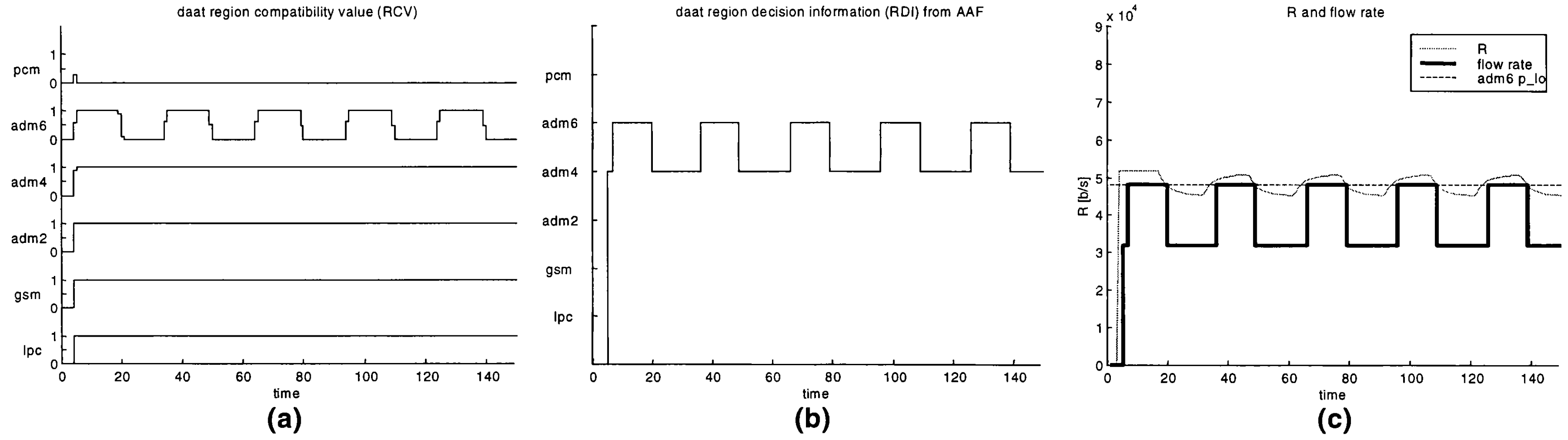


Figure 6.16: State-flapping for *daat* without QoSiRegions

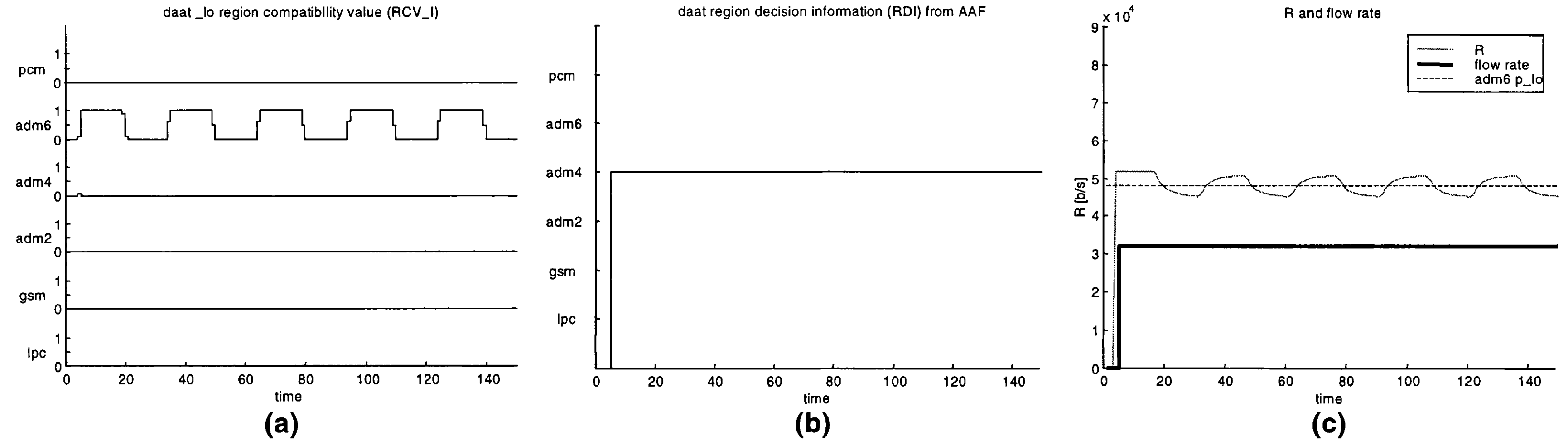


Figure 6.17: State-flapping removed using QoSiRegions and the modified AAF

6.7 Discussion

We have contrived a dynamically adaptable audio application (*daat*) that can adjust its audio encoding to change its flow rate. The *daat* is based on media scaling functionality that exists in real tools (e.g. *rat* [HSK98]), representing a realistic scenario. The QoS assessment capability offered by the QoSSpace allows *daat* to adapt to fluctuating network QoS. The decision is made by the AAF and the *daat* application *automatically*, but includes (static) user preferences. Another application may be more interactive, letting the user make the decision *manually* but present the user with a list of options based on the Q_SCORE/Q_ISCORE values or RCVs, allowing the user to make an informed decision.

The AAF is a simple algorithm. The main control issue is the interpretation of the user's wishes, via the user preferences. We have already seen that the QoSSpace has very little semantic knowledge of the flows. Notice that the AAF also has very simple semantic knowledge of the flows in *daat*. The QoS mapping from the user is simple; better quality QoSRegions have higher numbers than lower quality QoSRegions. The suitability of use of any particular QoSRegion is evaluated with Q_SCORES and Q_ISCORES for which "high" and "low" also have meaning. These are the only semantics that the AAF is aware of, making it a simple and easily implemented algorithm. Such simple QoS mapping between user, application and network, coupled with the simple nature of RCVs makes for easy decision-making and easy programmability in real applications. The main aim of this chapter was to show that the RCVs provided in the QoSReport ease the decision-making process. If simple relationships can be found between user preferences and application-modes and QoSRegions (as in our *daat* examples), the AAF has quite a simple task to perform.

The exact use of the adaptation capability will ultimately depend on the user and the application. We have modelled the *daat* to automatically adjust its flow rate by changing the flow encoding and this is reasonable because studies show that such behaviour in an audio flow does not adversely affect users' perception of quality [HS97]. This may not be true for all media types and for all people, even if the media is scaleable (e.g. video). In our examples, we have let the user choose *how* and *when* adaptation occurs, mapping values from the user directly to *q_time* and *q_compatibility* (P1, P2 and P3, Section 6.3). However, this does not preclude these values from being determined by the application through a different interaction with the user.

Stability and resource utilisation are also issues that must be left to the application. The QoSEngine tries to ensure some stability in the RCVs through the use of FAS, and allows the application to spot when the NetQoSRegion is near QoSRegion boundaries (by using the QoSiRegions). If the user chooses a very small or very large value for either *q_time* or *q_compatibility* value, must the application honour this? We have already seen (Section 6.4) that there may be a strong case to move at least the control of the *q_compatibility* value to the application where the nature of the flow demands, e.g. with video.

The QoSSpace does not attempt to deal with distributed application issues. Decision-making algorithms in a distributed environment could be centralised or distributed. We have shown a simple AAF algorithm that bases decisions about adaptability on information seen by a single *daat* instance, i.e. local information. If *daat* were used in a conferencing scenario, there may be a need to build in application-level signalling into the AAF. However, there is much heterogeneity (network QoS and user preferences) in multicast scenarios (e.g. Scenario 1a, Section 1.1). So, even if the adaptation decisions are not made on a local, per-instance basis, there will need to be feedback of local information from the application sites to any centralised/distributed decision-making mechanism. This may have effects on the value of *q_time* (to account for time required for application-level signalling), and so the application may also wish to have some control over its value.

So, both *q_time* and *q_compatbility* could be totally under application control. However, it could also be argued that allowing user dissatisfaction to be expressed as an input to the application (to adjust *q_time* and *q_compatbility* values) could also result in a similar effect but under user control [LSD98].

In a multicast scenario, the decision-making process may make use of localised mechanisms, allowing closely-located receivers to make adaptation decisions by exchanging QoSReports. Such self-organised, receiver-driven schemes are currently of great importance for scaleable Internet multicast [KHC98, MJV96], and one key element of their success is being able to share information about the QoS that the application instances experience.

The RCVs from the QoSSpace offer a simple and scaleable mechanism for conveying information about QoS experienced by an application in a user-friendly manner. For our *daat* example with six QoSRegions, if we were to map our RCVs from the range [0, 1] to an integer value in the range [0, 31] we could represent the whole QoSReport for the *daat* in six bytes (one byte per QoSRegion). This is regardless of the number of QoSParams used to define each QoSRegion. If we consider a distributed conferencing application, with separate voice, video and data flows, and use RCVs scaled to integer values in the range [0, 100], we can still represent the information for over 100 QoSRegions per-flow in 16 bits per QoSRegion. The QoS assessment process itself is scaleable – multiple QoSParams per QoSRegion do not require changes in the AAF and QoSSpace – because of the $\mathbf{and}_\mathbf{F}$ and $\mathbf{or}_\mathbf{F}$ operators. So the QoSSpace and RCVs offer a reasonably scaleable way of representing per-instance application QoS summaries.

The AAF used in this chapter is only an example to show the use of the RCVs. We would expect that application specific adaptation functions would be developed as required. The *daat* simulations indicate it is possible for a single flow to adapt to changes in the QoS available to the flow. Where multiple flows share resources with other flows in a best-effort network, we must be conscious of how the flows affect the network and the application behaviour. We must emphasise that the QoSSpace is a system for providing QoS information summaries to *aid* the process of making adaptation decisions. The AAF must select a QoSRegion and is responsible for *implementing* the adaptation. So, we would expect the AAF to include mechanisms for ensuring:

- **fairness:** with respect to the resources available to other flows
- **stability:** how the adaptation policy affects the network

A multicast application might achieve fair-share and congestion control by use of schemes such as [VRC98] that allow multicast traffic to share capacity fairly with TCP traffic. Again, such mechanisms need information about flow and network compatibility, which may be provided by the QoSSpace. Such mechanisms would also need to cater for congestion due to synchronisation effects in QoSRegion changes (e.g. the "9.00am effect") by use of heuristic mechanisms such as slow-start. In our *daat* AAF, this might be by insisting that all *daat* instances must use the *lpc* QoSRegion while $n < q\_time$. The general properties of macroscopic behaviour for multiple flows is currently under study [MSMO97] but suggests that some random behaviour in the network's treatment of packets may help to reduce congestion effects, especially those due to synchronisation. Examples of current mechanisms used for congestion control include the mechanisms in TCP [Jac88, Jac90] and various schemes for multicast [BTW94, MJV96, VRC98]. The difference between these schemes and a scheme based on the use of RCVs is that the QoSSpace can provide a summary of compatibility with many different QoS parameters and not just delay and/or loss.

We noted in Section 3.6 that there is currently no mechanism that supports QoS assessment for dynamic adaptation. We have demonstrated in this chapter that the QoSSpace appears to provide sufficient information for the decision-making process, based on the measured network QoS and can be easily incorporated with simple user preferences.

6.8 Chapter summary

We need to show how the QoSSpace can be used in an application to make dynamic adaptation decisions. (Section 6.1.)

We use an example application, the dynamically adaptable audio tool (*daat*), that can scale its audio flow data rate by using different audio encoding schemes. (Section 6.2.)

The *daat* needs some control information on *how* and *when* it should adapt. So we generate some user preferences that can be interpreted as simple rules for an adaptation policy. (Section 6.3.)

We then define an application adaptation function (AAF) for *daat* that incorporates the user's rule-based preferences. We determine values for *q_time*, the maximum adaptation frequency to "better" quality QoSRegions, and *q_compatibility*, a measure of the assurance the application must have of the network QoS before it can use a "better" quality QoSRegion. We show how the values of *q_time* and *q_compatibility* affect adaptation. (Section 6.4.)

In our extended example of the *daat*, we demonstrate the potential flexibility and generality of *daat* by showing how it could be used in a mobile application scenario for power conservation. We show operation with more than one QoSParam and also that QoSParams need not be restricted to the usual QoS parameters like delay, data rate etc. Additionally, we show how heterogeneity in the definition of QoSRegions is easily supported. (Section 6.5.)

The QoSEngine offers stability mechanisms to the application when it is operating near QoSRegion boundaries through the use of QoSiRegions. It is also possible to have heterogeneity in the definition of QoSiRegions. (Section 6.6.)

The QoSEngine appears to provide a flexible, general and practicable QoS assessment function for enabling decision-making for dynamic adaptability. The RCV generation process and the RCVs themselves are reasonably scaleable ways of generating per-instance, per-flow, application-level QoS summaries. The application is free to choose the nature of the interaction with the user. The application is not unduly constrained in its operation by the use of the QoSEngine. AAFs will be application-specific and will have to contain mechanisms to ensure stability and fairness. (Section 6.7.)

7. Summary and conclusions

Our main objective has been to provide functionality that informs the application of the suitability of the network's current QoS to support the application's operation. In a network offering best-effort service, such as the Internet, the **quality of service (QoS)** is hard to guarantee and typically fluctuates during the lifetime of an application instance. Additionally, different instances of the same application may see different QoS. Our primary motivation has been that applications currently lack functionality that allows them to make **dynamic QoS assessments** allowing them to become dynamically adaptable. As a result of this research we can conclude that:

It is possible to offer Internet applications a general and practicable network QoS model that will enable them to make per-flow QoS assessments dynamically, and allow dynamic adaptation, by monitoring the QoS being offered by the network.

In this work we have proposed a particular QoS information processing model, the **QoSEngine**, that can take application flow-requirements and per-flow QoS parameter measurements, and generate per-flow QoS summaries of compatibility between the network QoS and the applications requirements for operation of a flow. We have shown not only that dynamic adaptation is possible at the application level, we have also presented a particular engineering solution to enable the detection of QoS changes to enable adaptation decisions to be made. We have demonstrated the use of this solution with the simulation of an example application, a dynamically adaptable audio tool, using real measurements captured form various hosts on the Internet. Even where dynamic adaptation is not required, the QoSEngine could be used to detect per-flow QoS violations.

There are three main issues concerning the QoSEngine in practical use:

- the QoSEngine must still rely on the stability, robustness and correct behaviour of the underlying estimation mechanism (the QoSEngine back-end) and the measurement process it uses
- the application is ultimately responsible for the adaptation decision and so may need to further summarise the information from the QoSEngine, as system perturbations may generally be of a higher frequency than a user's adaptation requirements for an application
- the application needs to include mechanisms to ensure network stability and fairness

7.1 Integrated services via the Internet

The Internet seems an attractive access point for providing Integrated services:

- Internet access is readily available, at relatively low cost
- general purpose hardware platforms (such as PCs) with a wide range of applications can provide traditional (telephony) services as well as data communication services using Internet applications
- Internet protocols and software are easily accessible for application developers

However, the Internet was never designed for the **quality of service (QoS)** demands of real-time multimedia applications. The Internet offers only a best-effort service, which may not be sufficient for interactive applications or applications with real-time flows such as voice and video. Such applications can try to force the network to comply with their needs using **resource reservation** (static adaptation) or to try to adapt themselves so that they can operate in whatever network conditions they find (dynamic adaptation). It should also be possible to use some combination of static and dynamic adaptation.

Resource reservation using RSVP could be used to maintain a predetermined quality of service for Internet applications. However, for resource reservation to be truly useful end-to-end, it must be fully deployed *and* there must sufficient resources in the network to honour reservation for the majority of reservation requests.

A likely scenario for Internet services in the near future is **differentiated services**, where different users of the same application may subscribe to different QoS service-levels. Hence the same application may be required to work under different QoS conditions. This means that the application needs some form of adaptive capability.

In today's Internet, resource reservation and differentiated services do not exist and so the application must rely on the user to select the correct operating state for the application. Today's Internet applications lack mechanisms to enable them to make QoS assessments dynamically in order to aid the application or user in this task [BK98b].

7.2 Adaptability: an essential part of the QoS framework

We have produced a definition and requirements for dynamic adaptability for Internet applications. We have examined a **general QoS framework** and shown how our requirements fit into this framework. The general QoS framework is based around the provision of assured QoS and we find our dynamic adaptation requirements have strong agreements in many cases. However, we have also shown that our requirements do not agree with the all the general QoS framework requirements, notably on points concerning QoS flow-specification, service-level specification and the final determination of a QoS management policy for the application [BK98b].

We have discussed that RSVP has drawbacks and is not robust to failures along a reserved path. We have also seen how user requirements in terms of service-levels, as well as the user preferences for particular uses of applications, may introduce much heterogeneity during the operation of an application instance. Furthermore, mobility introduces much heterogeneity and requires applications to be adaptable to changes in QoS as the user moves location from one site to another, but also to cater for varying QoS during operation. So, even where resource reservation may be an option, there is a requirement for dynamic adaptability in applications.

There has been much work in the research community in order to allow media flows to be scaleable by use of novel encoding schemes, filtering, mixing and use of application-level gateways. However, this may still require either:

- a knowledgeable user to set and maintain correct user preferences and configuration
- mechanisms in the network to ensure that adaptability mechanisms are located at the proper locations within the network

We cannot always assume that the user has sufficient technical knowledge to configure an application and the Internet architecture and philosophy dictates that the application must not rely on any network technology-specific mechanisms, i.e. the application should take responsibility for making adaptation decisions. We have argued that, given the heterogeneity of user preferences and network service-levels, the application is the best place to make the adaptation decisions. However, the application lacks a general way of assessing QoS that will allow adaptation decision to be made.

7.3 Contributions of this work

The contributions we have made in this work are [BK98a, BK98b]:

- **to show that dynamic application adaptability is an important part of an application's QoS framework:** the application must be able to adapt in order to cater for variations in the offered network QoS. For the Internet, resource reservation is not ubiquitous, and RSVP cannot be wholly relied upon, so the application must have other functionality in order to maintain an adequate level of operation. There are also circumstances (such as mobility) where resource reservation may not be available, or user preferences and network service-levels present great heterogeneity, so dynamic adaptability is required
- **the QoSSpace:** a model of the network that allows application flows and network QoS to interact. **QoSRegions** are descriptions of the application's flow-requirement requirements using **QoSParam** boundaries, providing a simple QoS mapping between the application and the network. QoSRegions conceptually exist in QoSSpace and the network QoS, **NetQoSRegion**, can also be mapped into QoSSpace
- **the QoSEngine:** a system comprising the QoSSpace and a back-end QoS parameter processing function that is robust, adaptive and general. The QoSEngine can produce QoSReports that give an assessment of the network's current ability to support an application's flow-requirements. The QoSEngine is general and robust. The QoSReports contain **region compatibility values (RCVs)** which are scaleable, simple in nature and easy to understand and manipulate. The QoSEngine maps the network QoS, **NetQoSRegion**, as a region given by QoSParam value ranges within the QoSSpace. The QoSEngine is not constrained by any particular network model or traffic model. It can be used in general circumstances to provide estimates of QoSParam values given measured QoS parameter values. In most cases, the QoSEngine has good noise immunity and produces sensible values for QoSParams. The QoSEngine's performance degrades where there is extremely poor SNR in the

measurements (~13dB or below) or extreme variability in the measurements. The QoSEngine can be used to presents a particular engineering solution that can be used within the general QoS framework to detect QoS violations. The QoSEngine does not unduly constrain the design, construction or operation of the application

- **dynamic adaptation is possible and appears practicable:** we have used simulations to show how the RCVs from the QoSEngine can be used. Simple algorithms are used to incorporate user preferences and application requirements into an automatically controlled adaptation policy that controls the operation of a fictitious dynamically adaptable audio tool (*daat*). The *daat* adaptation algorithm uses a RCV threshold, *q_compatibility*, and a stability time, *q_time*, to produce an evaluation, the Q_SCORE/Q_ISCORE, of the suitability of a QoSRegion for use

The QoSEngine can easily be integrated into applications. The QoSEngine incurs relatively low additional computational cost, as it uses simple operations (MIN, MAX, plus a few simple arithmetic operations), and the EWMA filters used in the QoSEngine back-end require only a small amount of historic information per-flow. The QoSRegions specify an application-level flow specification but the definition and granularity of a flow is not constrained, remaining application-specific.

7.4 Limitations and future work

Our intention in this work was to define an abstraction that would enable applications to make QoS assessments, and so make decisions about flow-requirement changes based on measured QoS. We have achieved this goal, but there are a number of limitations. Those that need to be addressed first include:

- **refinement of the QoSEngine back-end:** we have the FAS-FAP estimator for the QoSEngine back-end to generate values for *p_p*. We have selected this mechanism as it is computationally inexpensive and the EWMA is widely used and is known to be robust and general. However, it can not deal with very large levels of noise or variability in the input. We note that any improvements to the estimation system must still produce a computationally inexpensive mechanism as the estimation system may need to provide many estimations for each of many flows and will be implemented in software on a general hardware platform
- **choice of variability parameter definition:** our intention with the variability parameter, *v_p*, is to try and measure the current fluctuations in the values of *p_p* in order generate values of *q_lo* and *q_hi* for the NetQoSRegion. This is a key part of the

QoSEngine's evaluation mechanism and directly influences the value of the RCV. Further investigation is needed in order to prove that our current definition of *v_p* is indeed applicable in general. There may be better mechanisms for generating *v_p* in certain circumstances, or for evaluating *q_lo* and *q_hi* directly from values of raw QoS parameter measurements. In our work, we have chosen to use a heuristic definition for *v_p* (based on our experience and observations) that it is computationally simple, produces a timely estimate and requires little state information

- **use of the QoSSpace within the *rat* audio tool:** in our examples in Chapter 6, we have referred to our fictitious tool, *daat*. In fact, *daat* is modelled around the current media-scaling functionality of *rat* [HSK98]. It should be possible to incorporate the QoSEngine into *rat* and show it working with another of *rat's* media adaptive abilities, namely redundant encoding to cope for errors and packet loss, in a conference scenario (e.g. Scenario 1b, Section 1.1). We would also like to integrate the QoSSpace into a video tool. We intend that such work will help to strengthen our claim for the generality and flexibility of our approach
- **adaptability in mobile applications:** in one of our examples in Chapter 6, we looked at *daat* operating on a mobile host. This is one area that we would be interesting and useful to investigate, as mobile applications typically require dynamic adaptability to cope with varying QoS as users move location (e.g. Scenario 2b, Section 1.1)
- **integrated QoS architecture for Internet applications:** we would like to investigate how dynamic adaptability fits into an integrated QoS architecture with other Internet QoS mechanisms, e.g. the INTSERV work using RSVP and the use of RTFM through SNMP. We are particularly interested in examining how adaptable applications would function within the DIFFSERV network architecture. This would involve an investigation of the interface I_{qn} (Figure 1.3) that we have not considered in detail in this work. Our aim here would be to show integration between adaptation mechanisms, QoS assurance mechanisms and network management (both static and dynamic adaptation)

In the longer term, we see that application-level interactions need to be addressed, including:

- **interaction with the user:** as we have noted in this dissertation, consideration of the user preferences and requirements is important. We need to investigate how the use of dynamic adaptability changes the way the user and the application must interact. Is it possible to have totally *automatic* dynamically adaptable applications or must there be some interaction with the user during the execution of an application, rather than just the use of static user preferences? This would involve an investigation of the interface I_{aq} (Figure 1.3) that we have not considered in detail in this work

- **distributed applications:** the following issues need to be addressed:
  - when an application *does* have the ability to make adaptation decisions, what are the interactions between instances that affect application mode changes and flow-requirement changes?
  - how does the local interaction with the user affect the decision-making process in distributed instances?
  - what is the interaction of the architectural components to support receiver heterogeneity in the decision making-process? (For example, how might the decision making process interact with the filtering mechanisms in the application level gateways or in the network?)

Glossary

A collection of selected acronyms and definitions used in this dissertation.

ADM	adaptive delta modulation
ADU	application data unit
delay	some measure of elapsed time since an epoch such as the transmission of a packet to its reception, e.g. end-to-end delay is the time taken for a packet to go travel between two-hosts; units seconds
DIFFSERV	Differential Services Working Group of the IETF
dynamic adaptation	an application is dynamically adaptable if it can monitor the network QoS it is receiving and automatically adapt its flow-requirement to operate within the offered network QoS
end-system	see: host
EWMA	exponential weighted moving averaging
FAP	fuzzy adaptive predictor
FAS	fuzzy adaptive smoother
flow	a sequence of packets that form a single unidirectional stream carrying information between a given source and given (unicast or multicast) destination
flow-requirement	a description of a particular flow-construction in terms of QoS service requirements, e.g. data rate, minimum delay, etc.
GSM	global system for mobile communications
host	a workstation or computing device that forms the hardware platform for the execution of an application or process that

	generates and/or receives flows
ICMP	Internet control message protocol
IETF	Internet Engineering Task Force
INTSERV	Integrated Services Working Group of the IETF
IP	Internet protocol
IPv4	IP version 4
IPv6	IP version 6 (note we use the specification in [IPV6] and not RFC1883)
ISPN	integrated services packet network
jitter	the variation in successive delay measurements; units seconds
LPC	linear predictive coding
media flow	see: flow
MIB	management information base
mode	a well-defined state of operation for an application
NetQoSRegion	network QoS defined in terms of QoSParam range values that show the variability of the QoSParams. The network has only one NetQoSRegion with respect to a particular flow at any point in time. NetQoSRegion must be defined by the same number of QoSParams that define the QoSRegions for a flow
operating point	see: set-point
PCM	pulse code modulation
PCV	parameter compatibility value: a number in the range [0, 1] that expresses the degree to which the QoSSpace assesses that a certain QoSParam lies within a region defined for that QoSParam within a QoSRegion for a flow
PCVF	PCV function: a part of the QoSSpace that evaluates PCVs
POTS	plain old telephone service

QoS	quality of service
QoSEngine	a QoS information processing function that takes raw measurements of QoS parameter values from the network and derives values of RCVs for QoSReports
QoSiRegion	an intermediate (quasi) region that is (an optional) part of the QoSRegion definition; it is not a true flow-requirement but represents a region within the QoSRegion that is in the proximity of the boundary of the QoSRegion region
QoSParam	values derived from the corresponding measured values of QoS parameters, generated by the QoSEngine back-end
QoSReport	a per-flow QoS summary containing RCVs for each flow-requirement
QoSSpace	a multi-dimensional space model, with its dimensions defined by QoSParams, in which the application flows conceptually exist and into which the network QoS can be mapped
QoSRegion	the flow-requirement specifed in terms of QoSParam boundary values that define the operating region of that flow-requirement within QoSSpace. A flow can have many QoSRegions. A QoSRegion can have many different QoSParams that define it
RSVP	Resource reSerVation Protocol
RTCP	Real-time Transport Control Protocol
RTFM	Real-time Traffic Flow Monitoring Working Group of the IETF
RTP	Real-time Transport Protocol
RCV	region compatibility value: a number in the range [0, 1] that expresses the degree to which the QoSSpace assesses that a certain QoSRegion can be supported by the network
RCVF	RCV function: a part of the QoSSpace that evaluates RCVs

set-point	a steady operating point for a flow
SNMP	simple network management protocol
SNR	signal to noise ratio
state-flapping	when an application oscillates between QoSRegions (more often than the user would typically prefer); for example due to a QoSParam value wavering around a QoSRegion boundary
static adaptation	an application is statically adaptive if it interacts with the network in order to assure some sort of reservation of resources so that it may operate in a fixed mode and/or with a fixed flow-requirement
traffic flow	see: flow
VEF	variability estimation function

References

[AMV96] E. Amir, S. McCanne, M. Vetterli, "A Layered DCT Coder for Internet Video", Proc. ICIP'96, Lausanne, Switzerland, Sep 1996.

http://http.cs.berkeley.edu/~elan/pubs/papers/ldct.ps

[AMZ95] E. Amir, S. McCanne, H. Zhang, "An Application Level Video Gateway", Proc. ACM Multimedia'95, San Francisco, CA, USA, Nov 1995

http://http.cs.berkeley.edu/~elan/pubs/papers/vgw.ps

[ANSA] APM Ltd, "ANSA: An Engineers Introduction to the Architecture", Technical Document Release TR.03.02, APM Cambridge Ltd, Poseidon House, Castle Park, Cambridge, CB3 0RD, UK, 1989

[Bas96] T. Bass, "A Functional Comparison of STII and RSVP in a Real-Time Heterogenous Client-Server Environment", 3 Apr 1996.

http://www.silkroad.com/working/stii-rsvp.ps

[BCDRF97] G. Blair, G. Coulson, N. Davies, P. Robin, T. Fitzpatrick, "Adaptive Middleware for Mobile Multimedia Applications", Proc. Network and Operating System Support for Digital Audio and Video (NOSSDAV'97), St Louis, USA 1997.

http://www.comp.lancs.ac.uk/computing/research/mpg/most/reports/nossdav97.ps.gz

[BFMM94] A. Banerjea, D. Ferrari, B. A. Mah, M. Moran, "The tenet real-time protocol suite: Design, implementation, and experiences", Technical Report TR-94-059, University of California at Berkeley, Berkeley, California, Nov. 1994.

ftp://tenet.berkeley.edu/pub/tenet/Papers/BaFeMaMoVeZh94.ps

[BG94] A. R. Bonde, Jr., S. Ghosh, "A Comparitive Study of Fuzzy Versus "Fixed" Thresholds for Robust Queue Management in Cell-Switch Networks", IEEE/ACM Transactions on Networking, vol. 2, no. 4, pp337-343, Aug 1994.

[BK98a] S. N. Bhatti, G. Knight, "Notes on a QoS information model for making adaptation decisions", Proc. 4th International Workshop on High Performance Protocol Architectures (HIPPARCH'98), University College London, London, UK, 15-16 Jun 1998.

[BK98b] S. N. Bhatti, G. Knight, "QoS Assurance vs. Dynamic Adaptability for Applications", Proc. 8th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'98), New Hall, Cambridge University, Cambridge, UK, 8-10 Jul 1998.

[BPK97] H. Balakrishnan, V. N. Padmanabhan, R. H. Katz, "The Effects of Asymmetry on TCP Performance", Proc. ACM/IEEE MOBICOM'97, Budapest, Hungary, Sep 1997.

ftp://daedalus.cs.berkeley.edu/pub/papers/tcpasym-mobicom97.ps.gz

[BTW94] J.-C. Bolot, T. Turletti, I. Wakeman, "Scalable Feedback Control for Multicast Video Distribution in the Internet", Proc. ACM SIGCOMM'94, London, UK, pp58-67, Oct 1994.

[BV96] J.-C. Bolot, A. Vega-Garcia, "Control Mechanisms for Packet Video in the Internet", Proc. IEEE INFOCOM'96, pp232-239, San Francisco, California, USA, Mar 1996.

http://www.cs.columbia.edu/~hgs/papers/Bolo9603_Control.ps.gz

[C+97] S. Crosby, I. Leslie, J. T. Lewis, R. Russel, F. Toomey, B. McGurk, "Practical Connection Admission Control for ATM Networks Based on On-line Measurements", Proc. IEEEATM'97, Lisbon , Jun 1997.

[CAH96] A. Campbell, C. Aurrecoechea, L. Hauw, "A Review of QoS Architectures", Proc. of 4th IFIP International Workshop on Quality of Service (IWQoS'96), Paris, France, March, 1996.

ftp://ftp.ctr.columbia.edu/CTR-Research/comet/public/papers/96/CAM96a.ps.gz

[Cam97] A. T. Campbell, "Mobiware: QoS-Aware Middleware for Mobile Multimedia Networking", Proc. IFIP 7th International Conference on High Performance Networking, White Plains, New York, Apr 1997.

http://comet.ctr.columbia.edu/~mobiware/wireless/PUB/hpn97.ps.Z

[CC94] C.-J. Chang, R-G. Cheng, "Traffic Control in an ATM Network Using Fuzzy Set Theory", Proc. IEEE INFOCOM'94 pp1200-1207, 1994.

[CCH94] A. Campbell, G. Coulson, D. Hutchison, "A Quality of Service Architecture", ACM Computer Communications Review, vol. 24, no. 2, 1994, pp6-27.

ftp://ftp.comp.iancs.ac.uk/pub/mpg/MPG-94-08.ps.Z

[CCH96] A. Campbell, G. Coulson, D. Hutchison, "Supporting Adaptive Flows in Quality of Service Architecture", ACM Multimedia Systems Journal, May 1996.

ftp://ftp.ctr.columbia.edu/CTR-Research/comet/public/papers/96/CAM96c.ps.gz

[CI93] R. Cacares, L. Iftode, "The effects of mobility on reliable transport protocols", Technical Report MITL-TR-73-93, Matsushita Information Technology Laboratory, Princeton, New Jersey, Nov. 1993.

ftp://miti.research.panasonic.com/pub/tr73-93.Z

[CKL94] P. Chemouil, J. Khalfet, M. Lebourges, "A Fuzzy Control Approach for Adaptive Traffic Routing", IEEE Communications Magazine, pp70-76, Jul 1995.

[Cla88] D. D. Clark, "The Design Philosophy of the DARPA Internet Protocols", Proc. ACM SIGCOMM'88, pp106-114, Aug 1988.

[CORBA96] CORBA v2.0 Specification, Object Management Group (OMG)

http://www.omg.org/

[Cox94] E. Cox, "The Fuzzy Systems Handbook", [AP Professional] 1994

[CSZ92] D. D. Clark, S. Shenker, L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism", Proc. ACM SIGCOMM'92, pp14-26, Aug 1992

[CT90] D. D. Clark, D. L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols", Proc. ACM SIGCOMM'90, pp 200-208. Sep 1990.

[D+93] L. Delgrossi, C. Halstrick, D. Hehmann, R. G. Herrtwich, O. Krone, J. Sandvoss, C. Vogt, "Media scaling for Audiovisual Communication with the Heidelberg Transport System", Proc. ACM Multimedia, pp99-104, Jun 1993.

[DHT95] C. Diot, C. Huitema, T. Turletti, "Multimedia Application should be Adaptive," Proc. High Performance Computing Symposium, (Mystic, Connecticut), Aug. 1995.
ftp://www.inria.fr/rodeo/ivs/papers/hpcs95.ps.gz

[DIFFSERV1] K. Nichols, S. Blake (Eds), "Differentiated Services Operational Model and Definitions", IETF DIFFSERV WG, work-in-progress, Feb 1998.

[DIFFSERV2] D. Clark, J. Wroclawski, "An Approach To Service Allocation in the Internet", IETF DIFFSERV WG work-in-progress, Jul 1997.

[DIFFSERV3] Z. Wang, "User-Share Differentiation (USD) Scaleable bandwidth allocation for differentiated services", IETF DIFFSERV WG work-in-progress, Nov 1997.

[DIFFSERV4] J. Heinanen, "Use of the IPv4 ToS Octet to Support Differential Services", IETF DIFFSERV WG work-in-progress, Nov 1997.

[DIFFSERV5] S. Blake, "Some Issues and Applications of Packet Marking for Differentied Services", IETF DIFFSERV WG work-in-progress, Dec 1998.

[DIFFSERV6] K. Nichols, V. Jacobson, L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet", IETF DIFFSERV WG work-in-progress, Dec 1997.

[DIFFSERV7] D. Clark, W. Fang, "Explicit Allocation of Best Effort Packet Delivery Service", IETF DIFFSERV WG work-in-progress, Nov 1997.

[DKS90] A. Demers, S. Keshav, S. Shenker, "Analysis and simulation of a fair queuing algorithm", Internetworking Research and Experience, vol. 1, pp3-26, Jan 1990.

[DT97] M. Decina, V. Trecordi, "Convergence of Telecommunications and Computing to Networking Models for Integrated Services and Applications", Proceedings of the IEEE, vol. 85 no. 12, Dec 1997.

[DWFB97] N. Davies, S. P. Wade, A. Friday and G. S. Blair, "Limbo: A Tuple Space Based Platform for Adaptive Mobile Applications", Proc. International Conference on Open Distributed Processing/-Distributed Platforms (ICODP/ICDP '97), Toronto, Canada, 27-30 May 1997.

http://www.comp.lancs.ac.uk/computing/research/mpg/most/reports/icodp97.ps.gz

[FJ95] S. Floyd, V. Jacobson, "Link-sharing and Resource Management Models for Packet Networks", IEEE/ACM Transactions on Networking, Vol. 3 No. 4, pp365-386, Aug 1995.

ftp://ftp.ee.lbl.gov/papers/link.ps.Z

[FPM95] K. Fall, J. Pasquale, S. McCanne, "Workstation Video Playback Performance with Competitive Process Load", Proc. 5th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'95), Durham, New Hampshire, pp179-182, Apr 18-21, 1995.

http://hulk.bu.edu/nossdav95/papers/KevinFall.ps

[GB96] M. Grossglauser, J.-C. Bolot, "On the Relevance of Long Range Dependency in Network Traffic", Proc. ACM SIGCOMM'96, pp15-24, Aug 1996.

[GHMY96] F. Garcia, D. Hutchison, A. Mauthe, N. Yeadon, "QoS Support for Distributed Multimedia Communications" Proc. 1st International Conference on Distributed Platforms, Dresden, Germany, 27 Feb-1 Mar 1996.

ftp://ftp.comp.lancs.ac.uk/pub/mpg/MPG-96-08.ps.Z

[HCBO98] M. Handley, J. Crowcroft, C. Borman, J. Ott, "Very Large Conferences on the Internet: the Internet Multimedia Conferencing Architecture", to appear in Computer Networks and ISDN Systems, Special Issue on Internet Telephony, 1998

http://north.east.isi.edu/~mjh/cnis.ps

[HILY96] J.-F. Huard, I. Inoue, A. A. Lazar, H. Yamanaka, "Meeting QOS Guarantees by end-to-End QOS Monitoring and Adaptation", Workshop on Multimedia and Collaborative Environments of the 5th IEEE International Symposium On High Performance Distributed Computing (HPDC-5), Syracuse NY, Aug. 1996.

ftp://ftp.ctr.columbia.edu/CTR-Research/comet/public/papers/96/HUA96a.ps.gz

[HS97] J. Hughes, M.-A. Sasse, "Internet Multimedia Conferencing - Results from the ReLaTe Project", Proceedings of the ICDE World Conference, Pennsylvania State University, 2-6 Jun, 1997

http://www.cs.ucl.ac.uk/staff/A.Sasse/icde.ps

[HSHW95] V. Hardman, M. A. Sasse, M. Handley & A. Watson, "Reliable Audio for Use over the Internet", Proc. INET'95, pp171-178, Hawaii, USA, 27-30 Jun 1995.

http://www.cs.ucl.ac.uk/staff/A.Sasse/inet95audio.ps

[HSK98] V. Hardman, A. Sasse, I. Kouvelas, "Successful Multi-party Audio Communication over the Internet", University College London, to appear in Communications of the ACM, May 1998.

[HWC95] M. Handley, I. Wakeman, J. Crowcroft, "The Conference Control Channel Protocol: A scaleable base for building conference control applications", Proc. ACM SIGCOMM'95, pp275-287, Sep 1995.

[ICMPv4] J. Postel, "Internet Control Message Protocol", RFC792, 1 Sep. 1981.

[ICMPv4] J. Postel, "Internet Control Message Protocol", RFC792, 1 Sep 1981.

[INTSERV] S. Jamin, C. Jin, L. Breslau, "A Measurement Based Algorithm for Controlled Load Service with a Reference Implementation Framework", INTSERV WG, work-in-progress, Oct. 1997.

[INTSERVQM] D. Clark, “The Quality Management Interface”, slides presented at the 31st IETF meeting, Jan 1995.

ftp://mercury.lcs.mit.edu/pub/intserv/clark-qos-manager.ps

[IPv4] J. Postel (Ed), “Internet Protocol DARPA Internet Program Protocol Specification”, RFC791, Sep 1981.

[IPv6] S. Deering, R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification”, IETF IPNG WG, work-in-progress, Nov 1997.

[Jac88] V. Jacobson, “Congestion Avoidance and Control”, ACM Computer Communication Review, vol. 18, no. 4, pp314-329, Aug 1988.

ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z

[Jac90] V. Jacobson, “Modified TCP Congestion Avoidance Algorithm”, end2end-interest mailing list, 30 Apr 1990.

ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail

[JDSZ95] S. Jamin, P. Danzig, S. Shenker, L. Zhang, “A Measurement-based Admission Control Algorithm for Integrated Services Packet Networks”, Proc. ACM SIGCOMM’95, pp2-13, Sep 1995.

[Kat94] R. H. Katz, “Adaptation and Mobility in Wireless Information Systems”, IEEE Personal Communications, vol. 1 no. 1, 1994.

[Kel97] F. P. Kelly, “Charging and rate control for elastic traffic”, European Transactions on Telecommunications, vol 8, pp33-37, 1997.

http://www.statslab.cam.ac.uk/~frank/elastic.ps

[Kes91] S. Keshav, “A Control-Theoretic Approach to Flow Control”, Proc. ACM SIGCOMM’91, Aug 1991.

http://simon.cs.cornell.edu/Info/People/skeshav/tenet/Keshav91a.ps

[KH97] I. Kouvelas, V. Hardman, “Overcoming Workstation Scheduling Problems in a Real-Time Audio Tool”, Proc. Usenix Annual Technical Conference, Anaheim, California, USA, Jan 1997.

http://www-mice.cs.ucl.ac.uk/staff/I.Kouvelas/publications/rat_usenix.ps.gz

[KHC98] I. Kouvelas, V. Hardman, J. Crowcroft, "Network Adaptive Continuous-Media Applications Through Self Organised Transcoding", to appear Proc. 8th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'98), New Hall, Cambridge University, Cambridge, UK, 8-10 Jul 1998.

[KK92] P. S. Khedkar, S. Keshav, "Fuzzy Prediction of Timeseries", Proc. IEEE Conference on Fuzzy Systems, Mar 1992.

http://simon.cs.cornell.edu/Info/People/skeshav/tenet/KhKe92.ps

[KMR93] H. Kanakia, P. P. Mishra, A. Reibman, "An Adaptive Congestion Control Scheme for Real-Time Packet Video Transport", Proc. ACM SIGCOMM'93, pp20-31, Oct 1993.

[Kos97] B. Kosko, "Fuzzy Engineering", [Prentice Hall], 1997

[KP87] P. Karn, C. Partridge, "Improving Round-Trip Estimates in Reliable Transport Protocols", Proc. SIGCOMM'87, pp2-7, Aug 1987

[L+96] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, E. Hyden, "The Design and Implementation of an Operating System to Support Distributed Multimedia Applications", IEEE Journal of Selected Areas in Communication, vol. 14, no. 7, Sep 1996.

http://www.cl.cam.ac.uk/Research/SRG/pegasus/papers/jsac-jun97.ps.gz

[Laz92] A. Lazar, "A real-time management, control and information transport architecture for broadband networks", Proc. 1992 International Zurich Seminar on Digital Communications, (Zurich, Switzerland), Mar 1992.

ftp://ftp.ctr.columbia.edu/CTR-Research/comet/public/papers/92/LAZ92b.ps.gz

[LLB97] S. Lu, K.-W. Lee, V. Bharghavan, "Adaptive Service in Mobile Computing Environments", in Building QoS into Distributed Systems, (A. Campbell, K. Nahrstedt, Eds), pp25-36, [Chapman & Hall] 1997

[LLSZ96] C. Lefelhocz, B. Lyles, S. Shenker, L. Zhang, "Congestion Control for Best Effort Service: Why We Need a New Paradigm", IEEE Network, vol. 10 no. 1, Jan/Feb 1996.

http://www.ieee.org/comsoc/lefelhocz.html

[LMM93] I. M. Leslie, D. R. McAuley, S. J. Mullender, "Pegasus - Operating System Support for Distributed Multimedia Systems", ACM Operating Systems Review, vol. 27, pp69-78, Jan. 1993.

[LSD98] B. Landfeldt, A. Seneviratne, C. Diot, "User Services Assistant: An End-to-End Reactive QoS Architecture", to appear Proc. 4th International Workshop on High Performance Protocol Architectures (HIPPARCH'98), University College London, London, UK, 15-16 Jun 1998.

[MESL94] D. Mitzel, D. Estrin, S. Shenker, L. Zhang, "An architectural comparison of ST-II and RSVP", Proc. IEEE INFOCOM'94, Toronto, Canada, Jun 1994.

ftp://caldera.usc.edu/pub/mitzel/Infocom94/infocom94.ps

[MJ95] S. McCanne, V. Jacobson, "vic: A flexible framework for packet video" Proc. of ACM Multimedia'95, Nov. 1995.

ftp://ftp.ee.lbl.gov/papers/vic-mm95.ps.Z

[MJV96] S. McCanne, V. Jacobson, M. Vetterli, "Receiver Driven Layered Multicast", ACM SIGCOMM'96, California, USA, pp117-130, Aug 1996.

[MK97] P. Moghe, A. Kalavade, "Terminal QoS of Adaptive Applications and its Analytical Computation", in Building QoS into Distributed Systems, (A. Campbell, K. Nahrstedt, Eds), pp367-378, [Chapman & Hall] 1997

[MSMO97] M. Mathis, J. Semke, J. Mahdavi, T. Ott, "The Macroscopic Behaviour of the TCP Congestion Control Algorithm", Acm Computer Communications Review, vol. 27 no. 3, Jul 1997.

[Oec97] P. Oechslin, "Worst case arrivals of leaky bucket constrained sources: the myth of the on-off source", in Building QoS into Distributed Systems, (A. Campbell, K. Nahrstedt, Eds), pp67-76, [Chapman & Hall] 1997.

[Pax97a] V. Paxson , "End-to-End Internet Packet Dynamics", Proc. ACM SIGCOMM'97, pp139-152, Sep 1997.

ftp://ftp.ee.lbl.gov/papers/vp-pkt-dyn-sigcomm97.ps.Z

[Pax97b] V. Paxson, "End-to-End Routing Behavior in the Internet", IEEE/ACM Transactions on Networking, vol. 5 no. 5, pp601-615, Oct. 1997

ftp://ftp.ee.lbl.gov/papers/vp-routing-TON.ps.Z

[PF95] V. Paxson, S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling", IEEE/ACM Transactions on Networking, vol. 3 no. 3, pp226-244, June 1995.

ftp://ftp.ee.lbl.gov/papers/WAN-poisson.ps.Z

[PF97] V. Paxson, S. Floyd, "Why we don't know how to simulate the Internet", Proc. 1997 Winter Simulation Conference

ftp://ftp.ee.lbl.gov/papers/wsc97.ps

[PHKS97] C. Perkins, V. Hardman, I. Kouvelas, & M. A. Sasse, "Multicast Audio: The Next Generation", Proc. INET'97, Putra World Trade Centre, Kuala Lumpur, Malaysia, Jun 1997.

http://www.cs.ucl.ac.uk/staff/c.perkins/papers/INET97.html

[Res97] S. L. Resnick, "Heavy Tail Modelling and Teletraffic Data", The Annals of Statistics, vol. 25, no. 5, 1997

[RFC1349] P. Almquist, "Type of Service in the Internet Protocol Suite", RFC1349, Jul 1992

[RFC1363] C. Partridge, "A Proposed Flow Specification", RFC1363, Sep 1992.

[RFC1633] B. Braden, D. Clark, S. Shenker, "Integrated Services in the Internet Architecture: An Overview", RFC1633, Jun 1994.

[RFC1809] C. Partridge, "Using the Flow Label Field in IPv6", RFC1809, Jun 1995.

[RFC1890] H. Schulzrinne, "RTP Profile for Audio and Video Conferences with Minimal Control", RFC1890, Jan 1996.

[RFC2001] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC2001, Jan 1997.

[RFC2002] C. Perkins, "IP Mobility Support", RFC2002, Oct 1996

[RFC2003] C. Perkins, "IP Encapsulation within IP", RFC2003, Oct 1996.

[RFC2004] C. Perkins, "Minimal Encapsulation within IP", RFC2004, Oct 1996.

[RFC2005] J. Solomon, "Applicability Statement for IP Mobility Support" RFC2005, Oct 1996

[RFC2006] D. Cong, M. Hamlen, C. Perkins, "The Definitions of Managed Objects for IP Mobility Support using SMIv2", RFC2006, Oct 1996

[RFC2029] M. Speer, D. Hoffman, "RTP Payload Format of Sun's CellB Video Encoding", RFC2029, Oct 1996.

[RFC2032] T. Turletti, C. Huitema, "RTP Payload Format for H.261 Video Streams", RFC2032, Oct 1996.

[RFC2035] L. Berc, W. Fenner, R. Frederick, S. McCanne, "RTP Payload Format for JPEG-compressed Video", RFC2035, Oct 1996.

[RFC2063] N. Brownlee, C. Mills, G. Ruth, "Traffic Flow Measurement: Architecture", RFC2063, Jan 1997.

[RFC2064] N. Brownlee, "Traffic Flow Measurement: Meter MIB", RFC2064, Jan 1997.

[RFC2190] C. Zhu, "RTP Payload Format for H.263 Video Streams", RFC2190, Sep 1997.

[RFC2198] C. Perkins, I. Kouvelas, O. Hodson, V. Hardman, M. Handley, J. C. Bolot, A. Vega-Garcia, S. Fosse-Parisis, "RTP Payload for Redundant Audio Data", RFC2198, Sep 1997.

[RFC2205] N. Freed, S. Kille, "RTP Payload Format for MPEG1/MPEG2 Video", RFC2205 Jan 1998.

[RFC2208]	A. Mankin, F. Baker, B. Braden, S. Bradner, M. O'Dell, A. Romanow, A. Weinrib, L. Zhang, "Resource ReSerVation Protocol (RSVP) – Version 1 Applicability Statement Some Guidelines on Deployment", RFC2208, Sep 1997.
[RFC2210]	J. Wroclawski, "The Use of RSVP with IETF Integrated Services", RFC2210, Sep 1997.
[RFC2211]	J. Wroclawski, "Specification of the Controlled-Load Network Element Service", RFC2211, Sep 1997.
[RFC2212]	S. Shenker, C. Partridge, R. Guerin, "Specification of Guaranteed Quality of Service", RFC2212, Sep 1997.
[RFC2213]	F. Baker, J. Krawczyk, A. Sastry, "Integrated Services Management Information Base using SMIv2", RFC2213 Sep 1997.
[RFC2214]	F. Baker, J. Krawczyk, A. Sastry, "Integrated Services Management Information Base Guaranteed Service Extensions using SMIv2", RFC2214, Sep 1997.
[RFC2215]	S. Shenker, J. Wroclawski, "General Characterization Parameters for Integrated Service Network Elements", RFC2215, Sep 1997.
[RFC2216]	S. Shenker, J. Wroclawski, "Network Element Service Specification Template", RFC2216, Sep 1997.
[RSVP]	R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification", RFC2205, Sep 1997.
[RTFM1]	N. Brownlee, C. Mills, G. Ruth, "Traffic Flow Measurement: Architecture", IETF RTFM WG work-in-progress, Dec 1997.
[RTFM2]	N. Brownlee, "Traffic Flow Measurement: Meter MIB", IETF RTFM WG work-in-progress, Dec 1997.
[RTFM3]	S. W. Handleman, N. Brownlee, G. Ruth, "IETF RTFM Working Group – New Attributes for Traffic Flow Measurements", IETF RTFM WG work-in-progress, Jan 1998

[RTP] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP A Transport Protocol for Real-Time Applications". RFC1889, Jan 1996.

[RTSP] H. Schulzrinne, A. Rao, R. Lanphier, "Real Time Streaming Protocol (RTSP)", IETF MMUSIC WG, work-in-progress, Dec 1997.

[SB95] S. Shenker, L. Breslau, "Two Issues in reservation Establishment", Proc ACM SIGCOMM'95, pp14-26, Sep 1995.

[SCEH96] S. Shenkar, D. Clark, D. Estrin, S. Herzog, "Pricing in Computer Network: Reshaping the Research Agenda", ACM Computer Communications Review, vol. 26. No. 2, pp19-43, Apr 1996

ftp://ftp.parc.xerox.com/pub/net-research/picn.ps

[Sch96] H. Schulzrinne, "Internet telephony - towards the integrated services internet", Proc. IEEE Workshop on Internet Telephony, Utrecht, The Netherlands, Feb. 1996.

http://www.cs.columbia.edu/~hgs/papers/Schu9602_Internet.ps.gz

[Sch97] H. Schulzrinne, "A comprehensive multimedia control architecture for the Internet", Proc. 7th International Workshop on Operating System Support for Digital Audio and Video (NOSSDAV'97), May 1997.

[SDP] M. Handley, V. Jacobson, "SDP: session description protocol", IETF MMUSIC WG, work-in-progress, Dec 1997

[SEFJ97] P. Sharma, D. Estrin, S. Floyd, V. Jacobson, "Scalable Timers for Soft State Protocols", Proc. IEEE INFOCOM'97, Apr 1997.

ftp://catarina.usc.edu/pub/puneetsh/papers/infocom97.ps

[Sha92] N. Shacham, "Multipoint communication by hierarchically encoded data", Proc. IEEE INFOCOM'92, vol. 3, pp2107-2114 (9A.4), Florence, Italy, May 1992.

[She95] S. Shenker, "Fundamental Design Issues for the Future Internet", IEEE Journal of Selected areas in Communication, no.13, pp1141-1149, 1995

http://ana-www.lcs.mit.edu/anaweb/pdf-papers/shenker.pdf

[SNMPv2] J. Case, K. McCloghrie, M. Rose, S. Waldbusser, "Introduction to Community-based SNMPv2", RFC1901, Jan 1996.

[SRC84] J. H. Saltzer, D. P. Reed, D. Clark, "End-To-End Arguments In System Design", ACM Transactions on Computer Systems, vol. 2, no. 4, pp277-288, Nov 1984

[ST2+] L. Delgrossi & L. Berger, Editors, "Internet Stream Protocol Version 2 (ST2) Protocol Specification - Version ST2+", RFC1819, Aug 1995.

[SW97] R. Steinmetz, L. C. Wolf, "Quality of Service: Where are We?", in Building QoS into Distributed Systems, (A. Campbell, K. Nahrstedt, Eds), pp210-221, [Chapman & Hall] 1997

[TCP] J. Postel (Ed), "Transmission Control Protocol DARPA Internet Program Protocol Specicifcation", RFC793, Sep 1981.

[traceroute] "Traceroute", Lawrence Berkeley Laboratory Network Research Group, USA

ftp://ftp.ee.lbl.gov/traceroute.tar.Z

[UDP] J. Postel, "User Datagram Protocol", RFC768, 28 Aug 1980.

[vat] "Visual Audio Tool", Lawrence Berkeley Laboratory Network Research Group, USA

http://www-nrg.ee.lbl.gov/vat/

[VRC98] L. Vicisano, L. Rizzo, J. Crowcroft, "TCP-like congestion control for layered multicast data transfer", to appear Proc. IEEE INFOCOM'98, San Francisco, USA, 29 Mar – 2 Apr 1998.

[WC91] Z. Wang, J. Crowcroft, "A New Congestion Control Scheme: Slow-Start and Search: Tri-S", ACM Computer Communications Review, pp32-43, Jan 1991.

[WC97] P. P. White, J. Crowcroft, "The Integrated Services in the Internet: State of the Art", Proceedings of the IEEE, vol. 82 no. 12, pp1934-1946, Dec 1997.

[WGCJF95] I. Wakeman, A. Ghosh, J. Crowcroft, V. Jacobson, S. Floyd, "Implementing Real Time Packet Forwarding Policies using Streams", Proc. USENIX'95, New Orleans, Louisiana, USA pp71-82, Jan 1995.

[WGS97] L. Wolf, C. Gridwodz, R. Steinmetz, "Multimedia Communication", Proceedings of the IEEE, vol. 85 no. 12, pp-1915-1933, Dec 1997.

[WTSW95] W. Willinger, M. S. Taqqu, R. Sherman, D. V. Wilson, "Self-Similarity Through High Variability: Statistical analysis of Ethernet LAN Traffic at the Source Level", Proc. ACM SIGCOMM'95, pp100-113, Sep 1995.

[WW97] W. Willinger, V. Paxson, "Discussion of 'Heavy Tail Modelling and Teletraffic Data' by S. R. Resnick", The Annals of Statistics, vol. 25, no. 5, 1997

[YGHS96] N. Yeadon, F. Garcia, D. Hutchison, D. Shepherd, "Filters: QoS Support Mechanisms for Multipeer Communications", IEEE Journal of Selected Areas in Communication, vol. 14, no. 7, pp1245-1262, Sep 1996.

[YL95] R. Yavatkar, K. Lakshman, "A CPU Scheduling Algorithm for Continuous Media Applications", Proc. 5th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'95), Durham, New Hampshire, pp223-226, Apr 18-21, 1995.

http://hulk.bu.edu/nossdav95/papers/yavatkar.ps

[YR95] C-Q Yang, A. V. S. Reddy, "A Taxonomy for Congestion Algorithms in Packet Switched Networks", IEEE Network, vol. 9 no. 5 Jul/Aug 1995

http://www.ieee.org/comsoc/yang.html

[Zad73] L. A. Zadeh, "Outline of a New Approach to the Analysis of Complex systems and Decision Processes", IEEE Transactions on Systems, Man and Cybernetics, vol. SMC-3, no. 1, 1973.

[ZSC91] L. Zhang, S. Shenker, D. D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic", Proc. ACM SIGCOMM'91, pp133-147, Sep 1991.

Appendix A: RDJ probes

Estimates of throughput (rate), delay and jitter using ICMP ECHO packets

We describe here the mechanism used to measure the available data rate along a network path in the Internet for this work. Note that it was not the intention of this mechanism to provide accurate or noise-free measurements. Indeed, we need noisy measurements in order to test estimation mechanism that forms the QoSEngine back-end. Our aim was to produce a simple and quick mechanism to generate some data rather than for accuracy.

We use IPV4 ICMP ECHO requests [ICMPv4] to estimate the rate, delay and jitter (RDJ) on an Internet path. ICMP ECHO is the same mechanism used by the `ping` program available on a variety of platforms, and so our mechanism suffers from the same drawbacks. ICMP ECHO packets:

- can be lost in the network, especially in the wide area
- are often treated with very low priority by routers under congestion, i.e. they are not treated FIFO (first-in first-out) so the delays for ICMP packets can be highly variable
- are not handled consistently by all routers in the network, so may not reflect the true characteristics of the network path

We refer to each ICMP ECHO request/response pair as a **RDJ probe** or just **probe**. Our measurement mechanism is simple. We first send 16 probes of size 24 bytes to a host IP address and use the minimum delay obtained from this set of 16 probes as the calibration time, T_C. Our measurements then consist of N probes, once per second, of size 128 bytes. We use the values:

$$D = t_n$$
$$R = (t_n - T_C)/(128\text{-}24)$$
$$J = \mathrm{ABS}(t_n - t_{n-1})$$

where D is the delay of the probe, R is the achievable data rate and J the jitter ($n = 1 \ldots N$). If a probe is lost, we use the measurements of the previous probe. With our knowledge of the geographical location of the remote host sites and the network path to the remote host (using `traceoute`), we have been careful to take calibrations and measurements at times when we expect the network load on the path to be light.

We have taken measurements from hosts at UCL, on network `cs.ucl.ac.uk` (128.16.0.0/16) to the hosts shown in Table A.1.

Each set of measurements involved the probes running for 1800 seconds, sending one probe a second. The names of the hosts involved at UCL were `theakston`, `poteen`, `grappa`, `darhu`, `mountaindew`, and `waffle`. `theakston` is connected to UCL via BR-ISDN (see Figure 2.2) while the other UCL hosts are all interconnected using 10BaseT. The RDJ probes were used to measure estimates of available capacity, i.e. the achievable data rate between the UCL hosts and the external hosts.

Host name	Short name	Geographical location
`north.lcs.mit.edu`	`north`	Laboratory for Computer Science, MIT, USA.
`knabe.syswiz.it`	`knabe`	System Wizards s.p.a, Italy
`myponga.connect.com.au`	`myponga`	connect.com.au Pty. Ltd., Australia
`tmnserver.ibm.ch`	`tmnserver`	IBM Laboratories Zurich, Switzerland.

Table A.1: Hosts involved in RDJ probe experiments

In the main text of the dissertation, we refer to data set from a set of probes as:

```
probe_sender_host – probe_reply_host
```

for example:

```
darhu – theakston
```

In our examples in the main text, we have used only the *R* values from the probes.

Note that the resolution of the system clock effects the measurements drastically when the RTT (round trip time) is close to the resolution of the clock. All our measurements were taken on Sun4 hosts running SunOS 4.1.3, with a clock resolution of 1.6ms.

Appendix B: A short fuzzy logic primer

In this section, we cover just enough of the principles of fuzzy logic to allow understanding of its use in our work. For a fuller discussion, an extremely readable and practical approach is presented in [Cox94]. A more thorough, mathematical presentation is given in [Kos97]. Other examples of the use of fuzzy systems are given in [BG94, CC94, CKL95].

Why fuzzy?

In our work, we have attempted to seek a model that allows us to construct a QoS assessment process for decision-making. In mathematics, decisions are generally based on Boolean logic (true or false) or on a statistical analysis. Boolean logic can be too restrictive in its single-bit granularity, and statistical analysis relies on having some well-defined model of the probability distribution of the system being measured. In [Zad73], the author notes:

> *... as the complexity of a system increases, our ability to make precise yet significant statements about its behaviour diminishes until a threshold is reached beyond which precision and significance (or relevance) become mutually exclusive characteristics.*

From [Cox94]:

> *Fuzzy logic is a calculus of compatibility. Unlike probability, which is based on frequency distribution in a random population, fuzzy logic deals with the characteristics of properties. Fuzzy logic describes properties that have continuously varying values by associating portions of these values with a semantic label. Much of the descriptive power of fuzzy logic comes from the fact that these semantic partitions can overlap. This overlap corresponds to transition from one state to the next. These transitions arise from the naturally occurring ambiguity associated with the intermediate states of the semantic labels.*

We would like to assess the ability of the network to support a particular application flow-requirement. Both of these can be complex to model accurately using mathematical techniques, as the research community has shown. We choose to use fuzzy logic in order to let us deal with the flow-requirement and network QoS to assess their relative *compatibility*, rather than try to assess the equality (or otherwise) of their absolute values.

Fuzzy sets

A **linguistic variable** is used to name a **fuzzy set** that is defined across a region of a **domain** set. Consider Figure B.1(a). We assess the length of a piece of string. Here:

- the length of the string, P, forms the **domain** set
- LONG is a **linguistic variable**, the name of a **fuzzy set**
- the fuzzy set is the mapping of the values in the range into the fuzzy set identified by the linguistic variable LONG, i.e. μ

The values of LONG, μ(p), (p is a member of set P) are in the range [0, 1]. The value μ(p) is:

the degree of membership that p has in set LONG

A more intuitive interpretation is:

the amount of truth there is in the assertion that "p is LONG"

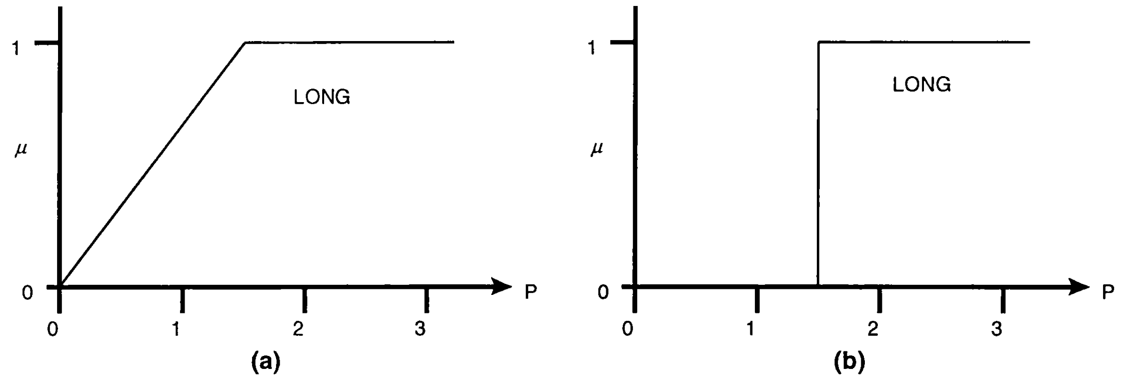


Figure B.1: How long is a piece of string? (a) fuzzy (b) Boolean

We can see the value of the fuzzy definition of LONG if we compare it with a Boolean definition in Figure B.1(b). Here, the string is defined to be LONG if the value of P ≥ 2.5m. That is to say, a piece of string that is of length 1.499m is NOT LONG (would have a truth value of zero), but a piece of string that is of length 1.501m is LONG (having a truth value of 1). The difference between the former and latter is about the width of this full stop.

We observe the following:

- the linguistic variable acts as a semantic tag

- the fuzzy set definition is application or user specific, e.g. LONG may mean different things to a 5 year old child (P in metres) than to a manufacturer and supplier of string (P in thousands of metres)
- the fuzzy set (although expressing an infinite range of values for the notion of LONG) is well-defined in the domain

Fuzzy sets offer the ability to apply semantic encapsulation of a part of the domain set using application specific tags, offering flexibility in definitions without loss of precision in the domain set. The added value that they offer is the ability to include and account for stages of transition in the domain value. Such a transition might be in the change of network QoS with respect to a flow-requirement.

Fuzzy operators

We refrain from a full treatment but note the following mappings for operators in fuzzy logic:

• AND	is the MIN function
• OR	is the MAX function
• NOT	is the complement of a value $(1 - \mu)$

We offer here only a hand-waving proof in that if truly Boolean values are used with our fuzzy definitions of AND ≡ MIN, OR ≡ MAX and NOT ≡ complement, they still behave correctly.

In our work, we have defined a new binary operator, WITHIN (as explained in Chapter 4) whose output is the amount of truth that:

the current value of the network parameter for a flow is within given boundaries

Fuzzy assertions

An **assertion** is a proposition that is assessed for the amount truth it contains. In a Boolean assertion, the result is either zero or one (true or false). In a fuzzy assertion, the truth can be in the range [0, 1]. In a **conditional assertion**, the predicate determines the solution space of the result, e.g. for a seller of string trying to determine a suitable pricing policy that includes a bulk discount:

if piece_of_string is LONG then COST is LESS

The truth of "piece_of_string is LONG" determines how much truth there is in "LESS" and so determines "COST" per metre. If many assertions apply to the same domain, they must be evaluated together, e.g.:

if piece_of_string is LONG then COST is LESS
if piece_of_string is THICK then COST is MORE

The amount by which the piece_of_string string is THICK indicates how much MORE it will cost per metre. We note that the reasoning process now takes on a linguistic form that retains ease of interpretation without losing precision. We look at how to assess multiple rules below.

A simple fuzzy system

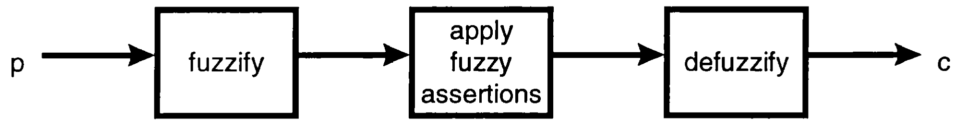


Figure B.2: A schematic diagram of a simple fuzzy system

A simple fuzzy system is depicted in Figure B.2. We have looked at how to fuzzify information and seen how assertions are used. We now look at how assertions are evaluated to a **consequent fuzzy set** (a solution fuzzy set resulting from an evaluation of the assertions) and then converted back to a real value in a solution domain. Note that the input and output of the fuzzy system can be from different domains.

We use P for the LENGTH, Q for how THICK the string is and C for its COST per metre. In Figure B.3, we see how the truth of P (LENGTH) and Q (THICK) are mapped onto linguistics variable MORE and LESS for C (COST). The MINimum values of predicate truth for each assertion on C produces a partial solution space (effectively a set intersection). Each of the partial solution spaces is then combined by taking the MAXimum values across the partial solutions spaces occupying the same domain region (effectively a set union). This method of composing the consequent fuzzy set by evaluating the partial solution spaces from each assertion is called **min-max composition**.

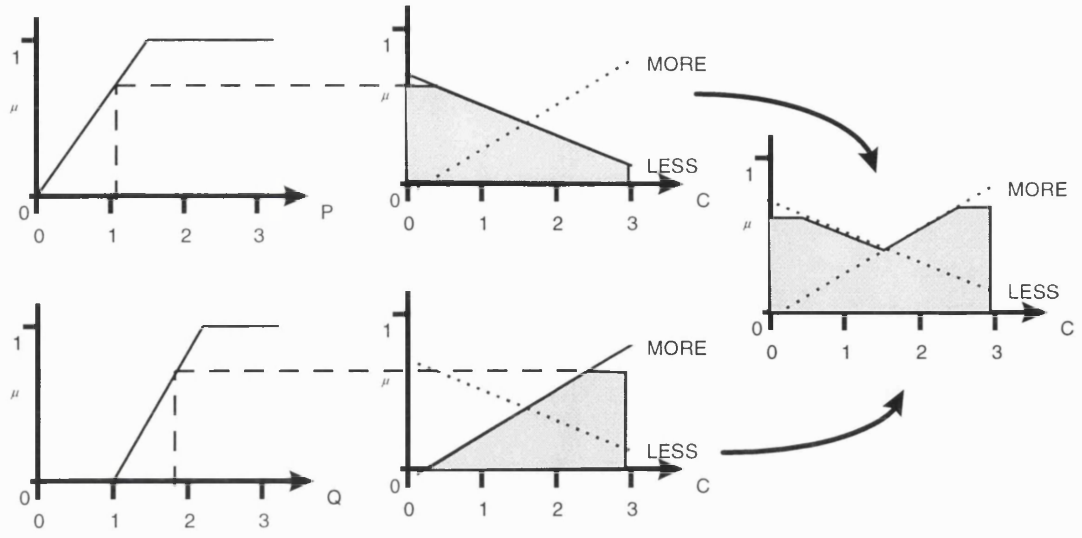


Figure B.3: min-max composition of assertions

A real value is then produced from the final consequent fuzzy set using by **defuzzification**. The most common method of deffuzification is to evaluate the **composite moment** or **centroid** of the final consequent fuzzy set:

$$c = \frac{\sum_{i=1}^{N} c_i \mu(i)}{\sum_{i=1}^{N} \mu(i)}$$

where:

c	value in domain C
c_i	is the i^{th} domain value in C
$\mu(i)$	is the truth of the i^{th} domain value in C in the final consequent fuzzy set

The centroid is a weighted mean of the consequent fuzzy set.