ADAPTIVE BEHAVIOR OF QUALITY OF SERVICE IN DISTRIBUTED MULTIMEDIA SYSTEMS

BY

BAOCHUN LI

B.Engr., Tsinghua University, 1995

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1997

Urbana, Illinois

Abstract

Current distributed multimedia applications demand Quality of Service from the system to facilitate effective services to the end users. However, within the range of QoS demands appropriately specified by the application, lower level transport facilities may not be able to constantly provide Quality of Service without perturbations. We introduce facilities in the middleware level to perform Quality of Service adaptations on a specific Quality of Service metric that is measured from the raw Quality of Service provided by the underlying layers, and refer to the component performing the task as *adaptors*. We are able to approach Quality of Service adaptations from a different perspective with a simplified but precise model for Quality of Service metrics, adaptations, and the adaptation behavior. Utilizing theories and techniques from the digital control theories and digital signal processing, we are able to model adaptation stability and agility of the performed adaptation behavior, and to introduce methods to analyze and configure the transformations according to the demands specified by the distributed multimedia applications.

TO MY PARENTS

Acknowledgments

First and foremost, I would like to thank my thesis advisor, Professor Klara Nahrstedt, for her invaluable directions and support throughout my research efforts towards this thesis. Her insights and suggestions to the problems presented in this thesis enlightened me in various detailed aspects through the work.

I would also like to acknowledge the constant assistance and encouragements from my best friends. Among them, special thanks goes to Dongyan Xu, who was also conducting research in the area of Quality of Service, and who has been generously taking time to discuss research possibilities with me, as well as to proofread my thesis at its completion.

Last but not least, my family deserves particular recognition for their unconditional emotional support during the past years. I am very grateful for my parents and my wife, Fang, for their love, dedication and belief in my ongoing graduate studies, even though we are so far away.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

In recent years, the field of distributed multimedia systems has shown extraordinary growth as witnessed by the tremendous amount of interest shown by academia, and by the wide spectrum of interactive distributed multimedia applications that rapidly emerged and reshaped our model of computing. A typical *multimedia application* conveys information to the end user via all possible presentation media, including synchronized video and audio, images, hyperlinks, graphics and animations, in addition to traditional text found in legacy applications. These promising new possibilities of presenting information have extended the capabilities of the application, thus made it possible to build complex frameworks such as digital libraries, video conferencing and high-performance video-on-demand distribution servers. These applications are only limited by the ingenuity of human imagination.

Along with these promising prospects for potential benefits of the unfolding multimedia revolution, we are also faced with technological challenges pushing the limits of the currently available hardware and software infrastructure. While providing extended capabilities of presentation, multimedia applications are also critically demanding for system processing resources in all levels and categories. To realize actual working systems, extensive research and development efforts are needed in various areas of general purpose distributed multimedia information systems.

Distributed multimedia applications are uniquely sensitive to the performance behavior of the components on the delivery path, while in contrast, traditional network applications are more tolerant of dynamic fluctuations of performance. Examples include multimedia conference appli-

cations that demand guaranteed end-to-end latency, video applications that demand guaranteed bandwidth, and speech applications that are sensitive to excessive jitter of delivery delay.

It is therefore necessary to assure that the *Quality of Service (QoS)* performance delivered by the coordination of all components on the delivery path matches the requirements of multimedia applications. While *Quality of Service* performance criteria traditionally included parameters with regards to network transmission, such as end-to-end latency, minimum allocated bandwidth, loss rate and delay jitter, it does not limit to this domain. As we shall notice, the notion of *Quality of Service* can be extended to much more diverse dimensions. As an example, we may extend the notion to the quality control of timing interdependencies, where an upper bound on the synchronization skew among different transmission streams may be desired. We may also extend Quality of Service to the quality of the result of a query to a digital library, in which case certain degrees of accuracy, timeliness or completeness of retrieved information may be demanded.

Due to the fact that the relative sensitivity to Quality of Service of multimedia applications usually exceed traditional applications by several orders of magnitude, guaranteeing the satisfaction of the expected Quality of Service over the course of delivery is not trivial, especially when utilizing currently adopted networking infrastructure to provide such guarantees. For some applications, we need *Guaranteed Quality of Service*, in which case once the delivered Quality of Service is promised and the connection is admitted, it should never be violated. However, *Adaptive Quality of Service* is sufficient for most other less demanding applications, where the need to dynamically adapt to variations of the actual Quality of Service delivered by the network may arise from time to time. These adaptations may be conducted at various levels, from the switching mechanism in the underlying network to the application itself.

Recent studies of Quality of Service delivery have focused on the design of network mechanisms to assure the guaranteed QoS. These studies range from the design of *Asynchronous Transfer Mode (ATM)* [20] protocols, resource allocation and reservation techniques such as the *Resource ReSerVation Protocol (RSVP)* [24], to the design of multimedia transport protocols. Mostly, these mechanisms have focused on regulating competition for network resources among traffic sources at the network level. They involve resource allocation and reservation, as well as flow and admission control techniques used by packet, cell, or transport layers.

On the contrary, this thesis will focus on the design of application level mechanisms to support effective adaptation to and control of Quality of Service delivery. Specifically, it focuses on implementing the adaptation policies in the middleware level, serving as a intermediate mediator between the multimedia applications that utilize Quality of Service delivery and the transport and network level mechanisms that provide Quality of Service. Apparently, any adaptive behavior and policies that the middleware addresses are only meaningful in the scenario of *Adaptive Quality of Service*, where the actual Quality of Service delivery varies in the time domain.

1.2 Motivation

As elaborated above, in order to provide stable and effective services to the user, distributed multimedia applications demand distinct Quality of Service guarantees from the underlying system and transport facilities. These demands, in turn, give rise to a range of new problems with regards to resource allocation, reservation, and manipulation techniques. A primary requirement of all the applications in the class is the provision of end-to-end real-time processing and service guarantees, and these guarantees ensure that continuous and discrete information can be achieved within a limited upper time bound.

The open problems of resource allocation, reservation and manipulation in the system and transport layers to provide Quality of Service have been addressed by various research efforts in the past years. However, these research efforts mainly focused in two broad categories. The first category, hereby referred to as the one-dimensional category, included the research of policies and mechanisms that provides guaranteed quality to one media stream, satisfying its requirements on various Quality of Service criteria such as end-to-end delay, bandwidth, and local processing resources. The second category, referred to as the two-dimensional category, included the cases of preserving timing interdependencies between two or multiple interdependent media streams, satisfying their demands on the upper bound of various interdependent behavior such as synchronization skew.

The research in the third category, referred to as the three-dimensional category, is still in its infancy. This direction of research focuses on the proper response by the underlying system and transport layers to the adaptive behavior of continuous media streams. In this category, the de-

sired quality of service of an distributed application may change after the negotiated connection has been set up, therefore demanding modified Quality of Service levels by the underlying layers to accommodate its needs. The system allows the demand in Quality of Service to change in predictable or unpredictable ways. In the case of predictable Quality of Service changes, the application may change its demands in a time deterministic manner or a event deterministic manner, which is driven by preset time instances or event types. In the case of unpredictable quality of service changes, the demands may change in unpredictable manners that are not specified off-line in advance. In both cases, the underlying system must be able to provide the adaptations promptly and in a stable manner, so that the response time of adaptation has its upper time bound, which, along with other time variant adaptation parameters, constitutes another main category of Quality of Service criteria that may be observed and demanded by the application.

In addition to the three-dimensional category problems elaborated above, another major class of adaptation possibilities should also be brought into consideration. As stated before, *Adaptive Quality of Service*, rather than *Guaranteed Quality of Service*, suffices to satisfy the QoS concerns and requirements of most of the applications. During the course of delivering *Adaptive Quality of Service*, the delivered QoS may change over time. For example, while the major previous research efforts tend to assume that the underlying system capabilities would stay stable without dramatic configuration and bandwidth changes, it may not be the case in the emerging mobile environment, where the mobile agents are constantly on the move and may lead to sudden and unexpected bandwidth and configuration changes in the wireless communication channels, especially when during the handoff process between heterogeneous wireless networks. In this scenario, the application and the system need to coordinate to provide graceful adaptation to dynamic Quality of Service variations. With the rapid evolution of commercially available mobile computing environments witnessed in recent years, the problems of adaptability emerge as a focus of recent research efforts.

The major focus of this thesis will be a close examination on the adaptive behavior and various associated parameters of the provided Quality of Service to regular distributed multimedia applications. The purpose of the study will be two-fold: one is to examine the problems occurred in the adaptation capabilities of underlying systems support, while the demanded Quality of Service

from the application changes over time in a predictable or unpredictable way after a connection with guaranteed service was set up; another purpose is to examine and experiment with the adaptive behavior of application Quality of Service demands while the provided Quality of Service by the underlying system changes over time, particularly in the scenario of mobile computing over heterogeneous networks.

1.3 Proposed Approach

Our proposed approach to solve the open problems elaborated above is to introduce the *middleware* level between the distributed multimedia applications and the transport facilities. This level will operate above the kernel space, and need to be portable over heterogeneous environments that is typical for distributed multimedia systems. Two objectives of introducing the middleware level will be addressed in this thesis. First, the middleware level is to encapsulate any minor or major changes in the raw Quality of Service that is delivered by the transport facilities, in order to make them transparent for the applications. Thus, the applications will only be aware of the filtered Quality of Service after intervened by the middleware, and the behavior of adaptations by the middleware can be specified and configured by the applications. Second, the middleware also needs to encapsulate the deterministic or indeterministic changes in Quality of Service requirements made by the applications, so that the transport facilities do not need to go through inadequate renegotiation at excessive frequency, and in the case of a multicast environment, the Quality of Service provided to other clients will not be violated or forced into modifications.

In order to realize the capabilities expected in the middleware level, a modular generic framework is being proposed that emphasizes the delivery of high availability, predictability, reliability and timeliness. The framework is proposed to be flexible for encompassing future developments, and to accommodate the demands of a wider range of distributed applications. Unsolved research issues will be investigated in the specification, classification and identification of Quality of Service. A generic model for the framework need to be presented to provide the maximum overall quality of the applications, under circumstances of limited input quality and available system resources. This requirement justifies the introduction of *reward profiles* into the middleware, which specify output quality as a multidimensional function of input quality and resource availability.

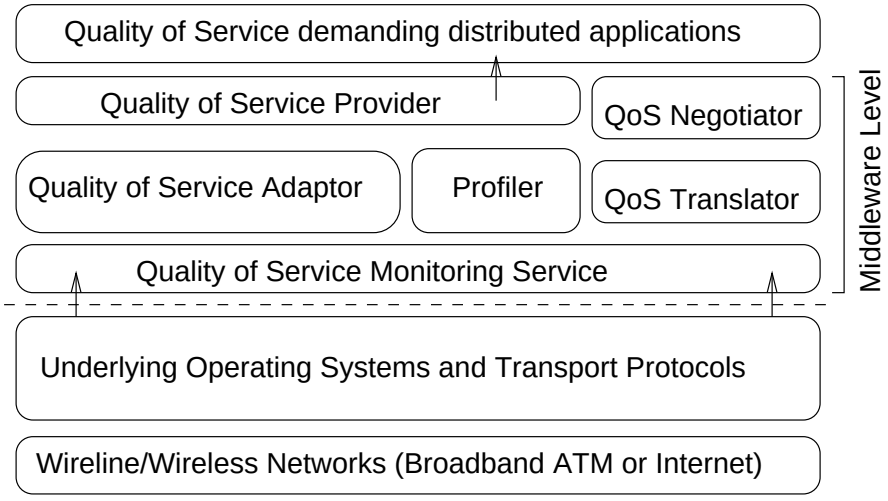

Figure 1.1: A generic framework for QoS-aware distributed applications

It also demand local or global optimization mechanisms in order to optimize overall system resource utilization, without jeopardizing the Quality of Service provided to individual tasks. In order to provide the above mechanisms, various services are necessary to be introduced into the middleware level, such as *adaptors* for adaptation purposes, *profilers* for QoS profiling, *translators* for QoS translations and mappings among different categories, and *negotiators* for QoS negotiations and re-negotiations. Figure 1.1 shows the generic architecture of the proposed framework for QoS-aware distributed applications.

As a start, this thesis will focus on the control and delivery of adaptation behavior that is tentatively implemented in the component referred to as *adaptors* in the above mentioned framework. These adaptors run in the application space and should be measured and configured according to their adaptive capabilities. In the case of a mobile environment, where distributed multimedia applications may suffer from sudden changes in the delivered Quality of Service, the need for adaptations becomes crucially important to maintain an acceptable behavior of the applications. There are two extremes with regards to adaptive capabilities of an adaptor. One is to avoid any attempts for adaptations, and pass the raw Quality of Service provided by the transport facilities directly to the applications. The other extreme is to provide an application-transparent adaptation, and places the entire responsibilities for adaptation on the middleware level. We need some specific metrics to measure the adaptive capabilities of the adaptor between the two extremes.

We propose the metric referred to as *adaptation agility* or *sensitivity* of the adaptor, which repre-

sents the ability and extent of an adaptor to promptly respond to perturbations in the raw Quality of Service from the lower levels. We will quantitatively measure the *agility* of the adaptors in this thesis, and propose methods to configure the adaptor behavior to a specified agility that is desired by the application.

An adaptor that is highly agile may suffer from instability. Such an adaptor consumes almost all its resources reacting to minor perturbations, hence taking excessive computational resources from the system. This behavior is apparently not desired. We note that stability is another important issue in the configuration process of an adaptor. We will introduce methods to verify the stability of the adaptors, and to guarantee the stability of a newly configured adaptor.

In addition to the adaptors, we also need *monitors* in the middleware level so that we can monitor the provided Quality of Service performance in from the underlying levels and detect changes in the time domain. Monitors should be capable of detecting variations for any specified QoS metrics, typically end-to-end delay, transmission rate and delay jitter. The output of monitors will be fed into adaptors to activate adaptations to Quality of Service fluctuations.

There are also certain occasions for the middleware level to adapt to requested changes of Quality of Service requirements made by the application. The need for such adaptations will arise when the requests were made frequently and the overhead for re-negotiation and re-establishment of new resource reservation arrangements in the lower layers is significant. Such adaptations are also desired in the case of multicast environments, where a sudden increase in the Quality of Service demands of one application may severely affect the provided Quality of Service to other applications, due to the resource limitations and resource sharing characteristics at the multicast server.

In addition to the metrics mentioned above for an adaptor, we propose an extended Quality of Service specification mechanism to specify the *Adaptive Quality of Service* desired by the application. We introduce the *Imprecise Computational Model* [4], that suits the QoS requirements of *flexible tasks* in *flexible applications*. It is possible to design a wide range of multimedia real-time applications so that their tasks are flexible. The output quality of these applications improves while the provided Quality of Service improves. For example, allocating a tracking application more processor time to complete a better but more complex tracking algorithm leads to a lower probability

of producing false tracks. Research on *Imprecise Computation* in areas as diverse as digital control systems [5] and artificial intelligence [12] have shown it to be a feasible and effective approach.

Under the model of Imprecise Computation, we logically divide the requirement of each flexible application into two parts: a *mandatory* part and an *optional* part. this division is done in such a way that the system still performs acceptably as long as all the mandatory parts of the requirement are satisfied. In other words, the mandatory part of the requirement is the least tolerable Quality of Service that the system must provide in order to maintain acceptable behavior of the application. In contrast, providing the optional part of the requirement improves the quality of the application's output, but it is not required for minimum performance of the application.

Once we have the extended specification mechanisms for Quality of Service, we can make decisions in the middleware adaptors according to the extent of changes in the QoS requirements made by the application, and the extent of violations for the mandatory part of the provided QoS to other applications in a multicasting environment. In the case of limited shared resources in the multicasting server, we can propose adequate policies as a response to the increasing or decreasing requests for Quality of Service.

In summary, in order to preserve the distributed multimedia system from disturbance of sudden perturbations of provided Quality of Service performance or the requests from other applications that share limited resources, we need to introduce adaptors and monitors in the middleware level to screen raw Quality of Service provided by the underlying transport facilities. The metrics, behavior and policies adopted in the middleware level are the major research concerns of this thesis.

1.4 Related Work

Currently there are many ongoing active research projects that focus on the open issues in Quality of Service management in end systems. Many of them focus on the transport or operating system levels in the end system, or on a generic framework for Quality of Service architecture. There are also research efforts that address the problems related to adaptations or graceful degradations of Quality of Service.

The *Quality of Service Architecture (QoS-A)* developed at Lancaster University [2] targets a lay-

ered architecture of services and mechanisms for QoS management and control of media flows in multiservice networks. Among others, the QoS specification supports the notion of QoS scaling policy to identify the QoS adaptation [23] similar to the notion of adaptors developed in this thesis.

The *OMEGA End-Point Architecture* [17] [18] developed at University of Pennsylvania introduces the notion of *QoS Broker* in order to provide end-to-end Quality of Service guarantees for distributed applications. QoS parameters are translated between application and network requirements by the QoS Broker, thus integrating media and network QoS management into a single entity. The work focuses on mapping and translation of QoS parameters. Not much attention is paid to adaptation issues.

The *Quality Assurance Language (QuAL)* developed at Columbia University [7] presents a new specification language referred to as *QuAL*. QuAL abstractions allow the specification of QoS constraints expected from the underlying computing and communication environment. These specifications are compiled into run-time components that monitor the actual Quality of Service delivered from the underlying layers. Upon QoS violations, application provided *exception handlers* are signaled to act upon the faulty events. QuAL also generates *profiles*, referred to as *Management Information Bases* in the work, that contain QoS statistics per application. Such MIBs, derived from the SNMP protocol, may be used to integrate application level QoS management mechanisms into standard network management frameworks.

Recent research interests in mobile computing also address the issues in Quality of Service adaptations for mobile transmissions over heterogeneous wireless networks [1] [21]. These research efforts focus on graceful adaptations to dynamic Quality of Service variations in the environment of mobile computing. Current state-of-the-art mobile environments must deal with scarce and dynamically varying resources, in particular, the network Quality of Service. Applications which execute in such environments need to adapt to the dynamic operating conditions in order to preserve the illusion of seamlessness for the end-user as far as possible. The research and development of the *Prayer* mobile computing environment [14] [15], for example, proposes a framework for adaptation which provides applications with runtime support for Quality of Service negotiation, monitoring and notification services. Utilizing the framework, applications only

need to specify the policy of adaptation at a high level, and are shielded from the mechanics of adaptation behavior. The generic objectives of this work are similar to the work presented in this thesis, though the approaches are from a different perspective.

Open problems in the area of adaptive playout control mechanisms in destination end systems have also been explored in previous research efforts. The *Adaptive Playout Mechanism* for packetized audio applications over wide area networks, developed at University of Massachusetts, focuses on the elimination of end-to-end delay jitter for audio data transmission over the Internet. Various algorithms for adaptively adjusting the playout delay of audio packets in an interactive packet-audio terminal application have been addressed in the work. The algorithms are able to explicitly adjust to the sharp, spike-like increases in packet delay. Since audio data playback is extremely sensitive to delay jitter, delay adaptation in the face of varying networking delays is crucial in preserving the audio quality at the destination end systems, especially in the cases of Internet transmissions. The goal of this work is also similar with the work presented in this thesis, though the former only focuses on delay jitter adaptations during audio transmissions. Similar research work also includes the work presented in [11], which also focuses on jitter control of playing back continuous media streams.

While the above mentioned work deals only with audio transmissions, various playout algorithms for video transmissions have also been discussed in previous research efforts, especially in the context of Video-On-Demand applications. The work at University of Pennsylvania [16], for example, mainly focuses on the playout requirements in destination end systems, assuming constant rate transmission of video data in Video-On-Demand applications over the ATM network. The protocol assumes the establishment of constant bit rate (CBR) virtual channel between the video provider and the viewer's set-top box, and it needs a certain number of cells be built up in the set-top box buffer space before the commencement of playback. The *build up*, cell transmission rate and set-top buffer size must be chosen so that there is no starvation or overflow at the set-top box. This work is analogous to the presented work in this thesis in the sense that we also deal with problems with the playback buffer space in the end systems, though our adaptation algorithm will be more complicated than the work presented in [16].

Previous research in the area of flow shaping and control in the source end system has led to

numerous noteworthy results in the past few years. In the work at AT&T Bell Laboratories [19], for example, has examined the performance of the well-known *Leaky Bucket* control scheme for both intraframe and mixed intra/interframe variable bit rate (VBR) MPEG video sources. The work investigates the effect of varying the leaky bucket parameters, and presents a new technique for VBR video transmission using multiple leaky buckets. In essence, source shaping and flow control are to some degree similar to the adaptation behavior that we address in this thesis. Some of the research results in this area are proved to be valuable and helpful to the research in our thesis.

The research efforts on switch-based flow control algorithms in the transport and network layers also provide valuable basis for the research presented in this thesis. Among them, we take the work at University of Texas as an example [8]. The work presents an adaptive network layer protocol for variable bit rate (VBR) video transport, which minimizes buffer requirement in the network while guaranteeing that packets of VBR encoded video flows will not be lost. It also works to minimize the end-to-end delay and jitter of video frames. To achieve these objectives, it utilizes a protocol referred to as receiver-oriented adaptive credit-based flow control algorithm. Research work for flow control protocols in the transport and network layers appears to be a major research concern in the past few years. Numerous research results are presented, which deserve close attention in the ongoing research for adaptation.

In more practical aspects, previous research efforts also focus on the preservation of Quality of Service, particularly in the case of audio and video transmissions, over best effort environments such as the Internet. Outstanding examples include the work at University of Illinois [3], in which a real time protocol referred to as the *Video Datagram Protocol (VDP)* for transmitting video and audio data over best effort environments was developed. VDP reduces inter-frame jitter and dynamically adapted to the client CPU load and network congestion. Best efforts environments such as the Internet shift all the workload of adaptation requirements to the end systems, thus require the protocols utilized to be stable and robust, while minimizing the resources required at the end systems. Previous work in this direction also serves as a basis for the ongoing research in this thesis.

To summarize, the adaptation problems faced in the ongoing research efforts presented this thesis are not new. Numerous previous research work related to these problems often presented

fruitful and thought provoking results. It is obviously necessary to conduct ongoing research efforts based on these results and experiences, while presenting new protocols and algorithms to meet more stringent requirements and research objectives.

Chapter 2

A Model for Adaptive Quality of Service

2.1 Overview

In order to provide certain adaptive behavior on the raw Quality of Service performance provided by the underlying transport facilities, we have introduced the notion of *adaptors* and *monitors* in the middleware level, that serves as an intervening medium between the application that receives Quality of Service and the underlying transport mechanisms that provides Quality of Service. Apparently, in order to deliver the desired adaptabilities, we should be able to measure and configure the behavior of the adaptors and monitors. In order to achieve this configurability, we need to formalize the concept of various Quality of Service metrics and adaptation behavior parameters, such as *adaptation agility* or *stability*. We address these problems in this chapter.

2.2 A Model for Quality of Service Metrics

2.2.1 Definition of QoS Metrics

A *Quality of Service Metric*, in its restrictive sense, measures the delivery performance of a stream of *entities*. In its broader sense, Quality of Service metrics can denote any kinds of quality that is delivered to the application transparently, for instance consistency, completeness or accuracy in a typical query to a digital library. We will only address the restrictive sense of Quality of Service in this thesis, and leave other criteria to later research efforts. However, we believe that the conclusions drawn from examinations on the restrictive sense of Quality of Service may also apply to any broader contents of Quality of Service.

We represent a *stream* as an ordered sequence of $S = \{e_k\}_{k \in I}$, where e_k denotes a distinct *entity*, which may in reality a *frame* of video data, a *packet* of data transferred over packet switching networks, or a protocol independent *message* carried by any transport facilities. I is the set of entity indices, while a stream S is an ordered sequence of entities. Also, we define T as the domain of time instants, which are normally represented by real numbers, and R as the domain of real numbers.

For each e_k, we assign a distinct *Quality of Service Signature* π_k, where $k \in I$:

$$\pi_k = \langle s_k, r_k \rangle (s_k < r_k,\ s_k \in T,\ r_k \in T) \tag{2.1}$$

where s_k and r_k represent the sending time at the original source of transmission and receiving time at the destination of the entity e_k, respectively.

Once the above notions are defined, we are able to define the ordered sequence of QoS signatures. We formally define the ordered sequence $\Pi_S = \{\pi_k\}_{e_k \in S}$ as the *Quality of Service Profile* of sequence *S*. Furthermore, the predicates $first(\Pi_S)$ and $last(\Pi_S)$ denote the indices of the first and last QoS signatures in Π_S respectively. For example, the expression $r_{first(\Pi_S)}$ represents the receiving time of the first entity in the stream S, while the expression $s_{last(\Pi_S)}$ denotes the sending time of the last entity in the stream S. We will also use the expression $\Pi_S[C]$ to represent the subset of QoS signatures in Π_S that satisfy the condition *C*, where *C* may be any unconditional or conditional equations.

Given the above definitions, a *Quality of Service Metric* consists of a measure computed from the *Quality of Service Profiles* of one or multiple streams. Formally, it represents a mapping function:

$$QoS : \Phi^n \mapsto R \tag{2.2}$$

where Φ is the domain of QoS profiles, *n* represents the *dimension* of the QoS metric, and *R* is the domain of real numbers. For example, in the one-dimensional category, the value of $QoS(\Pi_S)$ is the result of computing the QoS metric *QoS* on the Quality of Service Profile Π_S. If in reality we are interested in the Quality of Service *delay*, we will compute $delay(\Pi_S)$ based on the QoS profile Π_S of the stream S.

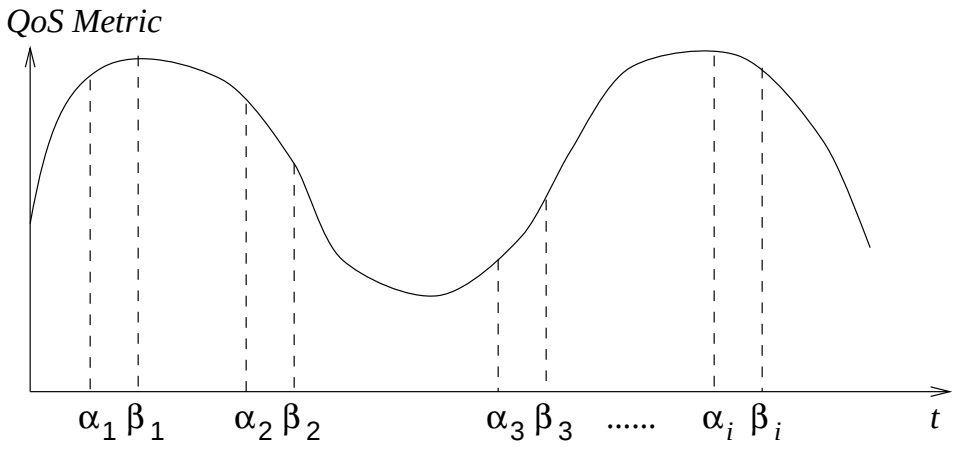

Figure 2.1: Sampling time intervals for evaluating Approximated QoS Metrics

Even though the Quality of Service metrics can be defined as such, it is generally difficult to obtain these metrics strictly as the definitions, constrained by practical measurement limitations. We thus introduce the notion of *Approximated Quality of Service Metrics*, whose definitions make it practical to measure and monitor QoS metrics in various implementations. Though these definitions can only approximate formally presented metrics shown above, they are normally sufficient for most cases.

For *Approximated QoS Metrics*, we define a mapping function:

$$QoS^* : T \times T \times \Phi^n \mapsto R \tag{2.3}$$

where again, Φ is the domain of QoS profiles, n is the dimension of the metric, R is the domain of real numbers, and T represents the time domain. To be more specific, if we define α_i and β_i to be *time instants* so that

$$\alpha_i < \beta_i,\ \alpha_i \in T,\ \beta_i \in T \tag{2.4}$$

where i is an index number. We can thus define a *time interval* $[\alpha_i, \beta_i]$ so that

$$t \in [\alpha_i, \beta_i] \text{ if and only if } \alpha_i \leq t \leq \beta_i \tag{2.5}$$

As an example, in the one-dimensional category, the value of $QoS^*(\alpha_1, \beta_1, \Pi_S)$ is the result of

Notation	*Definition*	*Example*
e_k	an entity of index k	a data packet or message
I	the set of entity indices	integers
S	a stream of entities: $S = \{e_k\}_{k \in I}$	a regular data stream
s_k	sending time of an entity e_k	sending at time 1
r_k	receiving time of an entity e_k	receiving at time 2
π_k	Quality of Service Signature: $\pi_k = \langle s_k, r_k \rangle (s_k, r_k \in T)$	$\langle 1, 2 \rangle$
Π_S	Quality of Service Profile: $\Pi_S = \{\pi_k\}_{e_k \in S}$	
$first(\Pi_S)$	the index of the first entity in stream S	
$last(\Pi_S)$	the index of the last entity in stream S	
$\Pi_S[C]$	the subset of QoS signatures in Π_S that satisfies C	$\Pi_S[r_k \leq 2]$
$QoS(\Pi_S)$	*QoS metric* computed on Π_S	$delay(\Pi_S)$
$QoS^*(t_1, t_2, \Pi_S)$	*Approximated QoS metric* computed on Π_S	$delay^*(t_1, t_2, \Pi_S)$

Table 2.1: A simplified model for Quality of Service metrics

computing the approximated QoS metric QoS^* on QoS profile Π_S, over the time interval $[\alpha_1, \beta_1]$, where $\alpha_1, \beta_1 \in T$. In reality if we are interested in the QoS metric *delay*, we use $delay^*(\alpha_i, \beta_i, \Pi_S)$ to compute the average delay of the QoS Profile Π_S in the time interval $[\alpha_i, \beta_i]$. We may establish multiple time intervals referred to as *sampling intervals* during the course of QoS delivery, so that we can evaluate an ordered sequence of Quality of Service metric values during each of these time intervals, as illustrated in Figure 2.1. We will discuss these problems in later sections.

For easier references, we summarize the above notations and definitions in table 2.1.

Given above, we are able to define the common Quality of Service metrics such as *system rate*, *generating rate*, *delay*, *loss* and *jitter*. These metrics are addressed in the following sections.

2.2.2 System Rate

System rate is generally referred to as *transfer rate*, *leaky bucket rate*, *bandwidth* or *throughput* in a network transmission context. However, the definition of *rate* should not be limited to this context, and we will refer to this as *system rate*. It should denote a QoS metric that measures the delivery speed or efficiency of any QoS quantitative entities. Given the definitions for QoS metrics in the previous section, *system rate* measures the mean number of entities per time unit *received* during the duration of a stream. For this purpose, it is useful to define the function $quantity : \Pi_S[C] \mapsto N$ that counts the quantity of entities in a QoS profile $\Pi_S[C]$ under the condition C, or, equivalently, a subset of the stream S. For example, $quantity(\Pi_S[t_1 \leq a \leq t_2])$ is the number of entities in S that are received within the time interval $[t_1, t_2]$. The formal definition of *system rate* is:

$$rate_s : \Phi \mapsto R$$

$$rate_s(\Pi_S) = \frac{quantity(\Pi_S[C_r])}{r_{last(\Pi_S)} - r_{first(\Pi_S)}} \tag{2.6}$$

where condition C_r can be defined as:

$$C_r = r_{first(\Pi_S)} \leq r_k \leq r_{last(\Pi_S)},\ r_k \in T \tag{2.7}$$

As stated in the previous section, it is normally not convenient to practically measure the *system rate* defined above. We may define the *approximated system rate* so that it can be measured practically. The proposed solution is to use an *interval* of time during which the rates are computed. Once the time interval starts, all computations are reset to their initial status. When the interval finishes, the metrics are computed and analyzed. Immediately following the computations of the previous interval, the statistics are initialized again as the new interval starts. The process should be repeated for the duration of the stream. The same mechanism will be used for most other QoS metrics. Given a time interval $[\alpha_i, \beta_i]$ defined in equation (2.4) and 2.5, the definition of *approximated system rate* is as follows:

$$rate^*_s : T \times T \times \Phi \mapsto R$$

$$rate^*_s(\alpha_i, \beta_i, \Pi_S) = \frac{quantity(\Pi_S[C^*_r])}{\beta_i - \alpha_i} \tag{2.8}$$

where condition C^*_r can be defined as:

$$C^*_r = \alpha_i \leq r_k \leq \beta_i \tag{2.9}$$

2.2.3 Generating Rate

The *generating rate* QoS metric measures the rate in which entities are generated or inserted into the stream. Its computation is very similar to that of *system rate* and defined by:

$$rate_g : \Phi \mapsto R$$

$$rate_g(\Pi_S) = \frac{quantity(\Pi_S[C_s])}{s_{last(\Pi_S)} - s_{first(\Pi_S)}} \tag{2.10}$$

where condition C_s can be defined as:

$$C_s = s_{first(\Pi_S)} \leq s_k \leq s_{last(\Pi_S)} \tag{2.11}$$

Similarly, *approximated generating rate* is defined by:

$$rate_g^* : T \times T \times \Phi \mapsto R$$

$$rate_g^*(\alpha_i, \beta_i, \Pi_S) = \frac{quantity(\Pi_S[C_s^*])}{\beta_i - \alpha_i} \tag{2.12}$$

where condition C_s^* can be defined as:

$$C_s^* = \alpha_i \leq s_k \leq \beta_i \tag{2.13}$$

2.2.4 Delay

Delay is usually referred to as *end-to-end delay* in the network transmission context, it measures the average difference between the sending time s_k and the receiving time r_k of all or a subset of QoS signature in a QoS profile Π_S. Formally, we define:

$$delay : \Phi \mapsto R$$

$$delay(\Pi_S) = \frac{\sum_{\forall \pi_k \in \Pi_S[C_r]}(r_k - s_k)}{quantity(\Pi_S[C_r])} \tag{2.14}$$

where C_r was defined in equation (2.7). Similarly, *approximated delay* is defined by:

$$delay^* : T \times T \times \Phi \mapsto R$$

$$delay^*(\alpha_i, \beta_i, \Pi_S) = \frac{\sum_{\forall \pi_k \in \Pi_S[C_r^*]}(r_k - s_k)}{quantity(\Pi_S[C_r^*])} \tag{2.15}$$

where C_r^* was defined in equation (2.9).

2.2.5 Loss

An QoS entity is considered to be *lost* if and only if it is received more than a given timeout t after its sending time. We can define the metric *loss* based on this assumption:

$$loss_t : \Phi \mapsto R$$

$$loss_t(\Pi_S) = \frac{quantity(\Pi_S[(r_k - s_k) > t \wedge C_s])}{quantity(\Pi_S[C_s])} \tag{2.16}$$

where C_s was defined in equation (2.11). Similarly, *approximated loss* is defined by:

$$loss_t^* : T \times T \times \Phi \mapsto R$$

$$loss_t^*(\Pi_S) = \frac{quantity(\Pi_S[(r_k - s_k) > t \wedge C_s^*])}{quantity(\Pi_S[C_s^*])} \tag{2.17}$$

where C_s^* was defined in equation (2.13).

For example, assume a stream S with $quantity(\Pi_S) = 20$, and $quantity(\Pi_S[r_k - s_k > t]) = 5$, we will conclude that the loss rate $loss_t$ is 25%.

2.2.6 Jitter

The QoS metric *jitter*, also referred to as *delay jitter*, measures how far apart consecutive entities are received. It can be measured as the *standard deviation* among individual entity delays and the average delay of the stream. The definition of standard deviation is:

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n - 1}} \tag{2.18}$$

where μ is the mean of all data, i is the index, n is the total number of data points, and x_i is a data point of index i.

Following the definition of *standard deviation* above, we can therefore define *jitter* as:

$$jitter : \Phi \mapsto R$$

$$jitter(\Pi_S) = \sqrt{\frac{\sum_{\forall \pi_k \in \Pi_S[C_r]}[(r_k - s_k) - delay(\Pi_S[C_r])]^2}{quantity(\Pi_S[C_r]) - 1}} \tag{2.19}$$

where C_r was defined in equation (2.7). As usual, *approximated jitter* is defined as:

$$jitter^* : T \times T \times \Phi \mapsto R$$

$$jitter^*(\alpha_i, \beta_i, \Pi_S) = \sqrt{\frac{\sum_{\forall \pi_k \in \Pi_S[C_r^*]}[(r_k - s_k) - delay^*(\alpha_i, \beta_i, \Pi_S)]^2}{quantity(\Pi_S[C_r^*]) - 1}} \tag{2.20}$$

where C_r^* was defined in equation (2.9).

2.2.7 Conclusion

As is shown in the previous sections, our model for *Quality of Service Metrics* suits well for defining the most common QoS metrics such as *system rate*, *generating rate*, *delay*, *loss* and *jitter*. This model uses a simplified approach in modeling these metrics, so that they can be monitored, configured and manipulated in QoS adaptors in later sections. Despite the simplified characteristics of the model, it can well represent a wide range of other metrics not mentioned in previous sections, such as *synchronization skew*, *peak delay* and *permutation*. However, its capabilities are limited in simple quantitative metrics, and are unlikely to be extended to more complex, qualitative metrics. The solutions to these deficiencies are left to future research efforts.

2.3 Quality of Service Specification

2.3.1 QoS Demands

In addition to the Quality of Service metrics elaborated and formally defined in the previous sections, we also need to specify the application demands, or requirements, for a specific Quality of Service level. Adaptive Quality of Service can only be made possible if the specified Quality of Service allows flexible fluctuations in a reasonable range, thus allows adaptations to occur within

authorized limits. We refer to these applications as *flexible applications*. These applications may similarly be composed of a series of *flexible tasks*, which carry different parts of the responsibilities of the application, and may have their own specific Quality of Service demands. In order to model the QoS demands of these flexible applications or tasks, we apply the *Imprecise Computation Model* [4] [12] [5] [13], which originated from the research of hard real-time scheduling for computational tasks.

The essence of the *Imprecise Computation Model* is to make the scheduling of real-time systems easier by dividing the processing of each task and its result in the system into two parts: a *mandatory* part and an *optional* part. The mandatory processing time of each task must be satisfied so that the task provides results of minimum quality requirement. If the optional processing can also be performed, the quality of the result produced by that task is improved, thus improving the quality of the output from the system as a whole. The system schedules the mandatory and optional parts so as to optimize the overall quality of its output.

The concept of the *Imprecise Computation* model was motivated by the observation that for many real-time applications, we may prefer to have approximated results of a poorer but acceptable quality on a timely basis, other than late results of the desired quality. Applying this concept to the context of multimedia applications, for example, it is often more acceptable to tolerate poor quality images or voices in multimedia transmissions than late frames and long silences. It is also observed that good approximate results can often be produced with much less processor time or system resources than results of desired quality. By trading the quality of the results for the amount of time and resource required to produce them, a system based on this technique tries to make approximate results of acceptable quality available whenever it cannot produce results of the desired quality in time. Such system is often referred to as an *adaptive system*. The approach is most frequently applied to cases when the system is overloaded.

Even though the original concept of *Imprecise Computation Model* is for the purpose of hard real-time scheduling of flexible tasks, it is natural to extend the concept to adaptive Quality of Service management, graceful degradations and adaptations of real-time systems. All systems that require Quality of Service support are in some sense real-time systems, and applications executing on these systems indeed have their implicit or explicit deadlines. Some of these deadlines are

presented as Quality of Service metrics that are specified in the Quality of Service demands, and require satisfaction in the provided Quality of Service. This scenario is naturally very similar to that of task scheduling in real-time systems.

We thus propose the Quality of Service specification mechanism be extended to accommodate minor fluctuations or adaptations, within the range of specified QoS demands. In the case that an underlying system cannot meet both the mandatory and the optional part of the Quality of Service demands of the flexible applications or tasks, the application or individual tasks will yield *imprecise results*, whose precision depends on the QoS that the system can provide. While the system will make every effort trying to meet the demands of both parts, in case of overloading or other exceptions, the application agrees to tolerate if the system decides to trade quality with time.

In the original *Imprecise Computation* model, each task T_i is logically decomposed into two tasks, the *mandatory* task M_i followed by the optional task O_i, independent of the method used to implement it. Let τ_i, m_i and o_i denote the processing times of T_i, M_i and O_i, respectively. Clearly, $m_i + o_i = \tau_i$. The classical deterministic model is a special case of this imprecise computation model where all tasks are mandatory, that is, $o_i = 0$ for all i. Similarly, a sieve or an anytime computation is a task that is entirely optional.

Similarly, in the specification of any of the Quality of Service metrics discussed in the previous sections, we can logically decompose the Quality of Service demand for a particular metric T_{qos} or T_{qos^*} by the application into two partial demands, the *mandatory* demand M_{qos} and the *optional* demand O_{qos}. Let τ_{qos}, m_{qos} and o_{qos} denote the quantitative measures of T_{qos}, M_{qos} and O_{qos}, respectively. Clearly, $m_{qos} + o_{qos} = \tau_{qos}$.

In addition to extensions to the Quality of Service specification mechanism, we can also apply the original Imprecision Computation Model to the representation of *data*. In this scenario, the tasks will be defined as manipulations or operations on *data*, and the mandatory and optional part of the tasks will naturally represent a division of data into different sections. For example, in progressive coding schemes, some spatial or temporal parts of the image or video will be mandatory, while the refinements and enhancements will be optional. Corresponding to different requirements and divisions, different number or types of tasks will need to be performed in order to

render the desired quality of images or video streams.

2.3.2 Quality of Service Violations

A *Quality of Service violation* occurs when a QoS metric of the provided Quality of Service is not within the tolerable range that specified *a priori* by the application. All QoS violations are detected by a *Quality of Service monitor* that is integrated in the implementation of the middleware level. A QoS violation function defines in reality a class of functions, where each function measures violation of a specific metric. Given a QoS metric *qos*, where *qos* can be any Quality of Service metric such as *system rate* or *jitter*, $violation_{qos}$ indicates whether a stream violates the metric *qos*. When $m_{qos} \leq qos(\Pi_S) \leq \tau_{qos}$ does not hold to be true, we can determine that a QoS violation occurs. Formally, we define:

$$violation_{qos} : \Phi \times R \times R \mapsto \{True, False\}$$

$$violation_{qos}(\Pi_S, m_{qos}, o_{qos}) = \begin{cases} False, & \text{if } m_{qos} \leq qos(\Pi_S) \leq (m_{qos} + o_{qos}) \\ True, & \text{otherwise} \end{cases} \tag{2.21}$$

Similar to the study of *Quality of Service metrics*, violations can be approximated and checked over a time interval $[\alpha_i, \beta_i]$. Formally:

$$violation^*_{qos^*} : T \times T \times \Phi \times R \times R \mapsto \{True, False\}$$

$$violation^*_{qos^*}(\alpha_i, \beta_i, \Pi_S, m_{qos^*}, o_{qos^*}) = \begin{cases} False, & \text{if } m_{qos^*} \leq qos^*(\alpha_i, \beta_i, \Pi_S) \leq \tau_{qos^*} \\ True, & \text{otherwise} \end{cases} \tag{2.22}$$

Consider, for example, the monitoring of a QoS metric *system rate* on a Quality of Service profile. $violation^*_{rate^*_s}(\alpha_i, \beta_i, \Pi_S, m_{rate^*_s}, o_{rate^*_s}) = False$ if and only if it holds that the average system rate delivered by the QoS underlying layers in the time interval $[\alpha_i, \beta_i]$ is within the interval $[m_{rate^*_s}, m_{rate^*_s} + o_{rate^*_s}]$, which is the same as $[m_{rate^*_s}, \tau_{rate^*_s}]$. Here, $m_{rate^*_s}$, $o_{rate^*_s}$ and $\tau_{rate^*_s}$ are all

determined by the application, in its specified Quality of Service demands, using the proposed extended mechanism for Quality of Service specifications.

2.4 Quality of Service Adaptations

2.4.1 Modeling Generic Adaptors

The discussions proposed in previous sections provide sufficient grounds for the elaboration of a model suitable for modeling the behavior of *Quality of Service Adaptations* commonly found in the *Quality of Service adaptors* in the middleware level. These adaptors control the raw Quality of Service metrics measured by the monitors upon the QoS provided by underlying layers, according to a predefined configuration. In this section we formally define the behavior model of the adaptor.

Generally, an *adaptor* is an *operator* or *filter* on a particular stream that controls and modifies its Quality of Service profile. Formally, an *adaptor* is defined as:

$$\varphi : \Phi \mapsto \Phi$$

where $\varphi(\Pi_S)$ is any transformation performed on Π_S.

As a simple example, $\varphi(\{\langle s_k, r_k \rangle\}) = \{\langle s_k, r_k + const \rangle\}$ is an adaptor that delays the receiving time of all entities in a stream by a constant *const*. This adaptor can be understood as a *delaying adaptor*, whose only transformation applied on the QoS profile is to delay the receiving time by a specified amount.

Modeling and configuring the behavior of a generic adaptor described above is not a trivial task. To the extent of this thesis, we will only address a specific subset of these generic adaptors, namely, those adaptors whose behavior only transforms a specific *QoS metric* of a particular QoS profile. In addition, since the adaptor will be implemented on the receiver side, it is only able to affect the receiving time of each QoS signature, the sending time remains unaffected. Formally:

$$\varphi : \Phi \mapsto \Phi$$

$$\Pi'_S = \varphi(\Pi_S) = \{\langle s_k, r'_k \rangle\},\ for\ \forall \langle s_k, r_k \rangle \in \Pi_S \tag{2.23}$$

Apparently, the behavior of these adaptors is to transform the receiving time r_k to r'_k in all QoS signatures of a QoS profile according to a predefined set of transformation rules. However, we still need a precise model for the transformation rules, so that we can specify and configure these rules to perform the desired transformations. This problem is addressed in the next section.

2.4.2 Modeling Transformation Rules

Based on the modeling for adaptors elaborated above, we can model the behavior of adaptors as a transformation from original values of r_k to some new values r'_k. Since we are more interested in adapting to the fluctuations in specific Quality of Service metrics, the transformation should not be applied to the receiving times in signatures directly, as the case of the simple delaying adaptor illustrated in the previous section. Instead, the transformation will be applied to a specific QoS metric, such as *system rate*, and the response of the transformation will then be values based on the measuring unit of this QoS metric. By utilizing the definitions that we developed for QoS metrics or approximated QoS metrics, we can easily compute values for these metrics based on a sequence of QoS signatures in a specific QoS profile. Furthermore, it is obvious to notice that, based on the definitions, we can also develop mechanisms to reversely convert the known values of QoS metrics to QoS signatures of a QoS profile, in which the known values of QoS metrics are the response of the transformation made by the adaptor.

To be more concrete, we take the QoS metric *system rate* as an example. We already learned that:

$$rate^*_s(\alpha_i, \beta_i, \Pi_S) = \frac{quantity(\Pi_S[\alpha_i \leq r_k \leq \beta_i])}{\beta_i - \alpha_i} \tag{2.24}$$

Once applied to an ordered sequence of *time intervals* α_i and β_i, as illustrated previously in Figure 2.1, this definition of *system rate* can be utilized to compute a discrete-time ordered sequence of values $rate^*_s[i]$ based on the QoS profile Π_S. For the most frequently used case of consequent time intervals where

$$\beta_i = \alpha_{i+1} \tag{2.25}$$

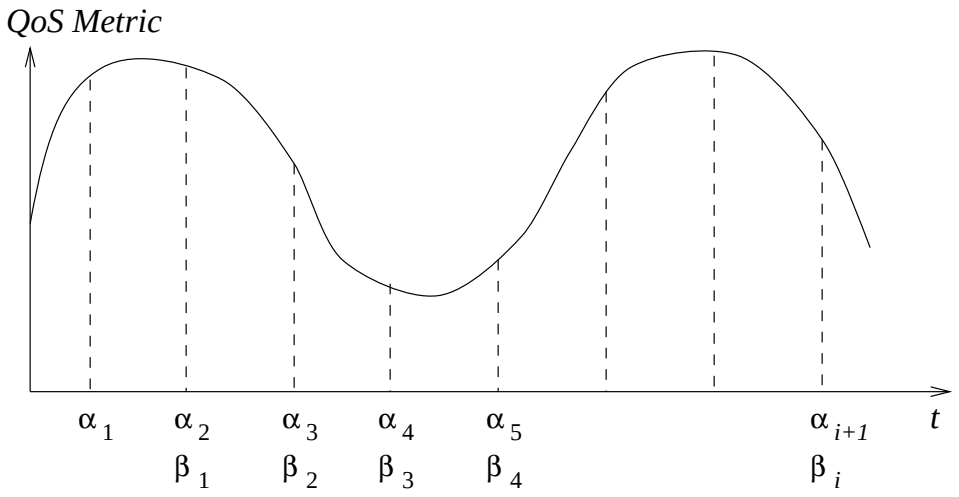

Figure 2.2: A frequently used special case for sampling time intervals

as shown in Figure 2.2. we have:

$$0 \leq i \leq \frac{r_{last(\Pi_S)} - r_{first(\Pi_S)}}{\beta_i - \alpha_i} \tag{2.26}$$

and the computation of the ordered discrete-time sequence $rate_s^*[i]$:

$$rate_s^*[i](\alpha_i, \beta_i, \Pi_S) = \frac{quantity(\Pi_S[\alpha_i \leq r_k \leq \beta_i])}{\beta_i - \alpha_i} \tag{2.27}$$

Reversely, we can take a discrete-time sequence of $rate_s^*[i]$ and convert it to a Quality of Service profile Π_S whose *system rate* is identical or approximated by the sequence $rate_s^*[i]$. Formally:

$$quantity(\Pi_S[\alpha_1 \leq r_k \leq \alpha_i]) = \sum_{k=1}^{i} (\beta_k - \alpha_k) rate_s^*[k](\alpha_k, \beta_k, \Pi_S) \tag{2.28}$$

In any Quality of Service signature $\langle s_k, r'_k \rangle$ in the Quality of Service Profile $\Pi'_S[\alpha_1 \leq r_k \leq \beta_n]$ defined in equation (2.23), we have:

$$r'_1 = r_1$$

$$r'_k = r'_{k-1} + \frac{\beta_i - \alpha_i}{quantity(\Pi'_S[\alpha_i \leq r'_k \leq \beta_i])} = r'_{k-1} + \frac{1}{rate_s^*[i](\alpha_i, \beta_i, \Pi'_S)},\ if\ \alpha_i \leq r'_k \leq \beta_i \tag{2.29}$$

In later discussions, we take *system rate* as an example of the Quality of Service metrics on

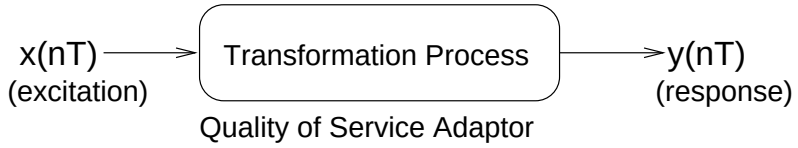

Figure 2.3: Applying adaptation process to the raw QoS metrics

which the adaptor applies adaptations. Other common QoS metrics as discussed in previous sections are similar in forms of treatment.

Because of the limitations in measurements, the original values $rate_s^*[i]$ are a series of discrete-time values, or *signals*, rather than continuous-time signals. Formally, the original values of QoS metrics $rate_s^*[i]$ can be represented by a function $x(nT)$, where T is a constant and n is an integer in the range (n_1, n_2) such that $-\infty \leq n_1$ and $n_2 \leq \infty$. Alternatively, a discrete-time signal can be represented by $x(n)$ or x_n. In this thesis we will be using the first two notations, namely, $x(nT)$ and $x(n)$.

The transformation process can be represented by the block diagram of Figure 2.3. Input $x(nT)$ and output $y(nT)$ are the *excitation* and *response* of the transformation, respectively. In their original senses, input $x(nT)$ is the original ordered sequence of $rate_s^*[i]$ of a specific QoS profile, and output $y(nT)$ is the transformed response series $rate_s'^*[i]$, corresponding to the transformed QoS profile Π'_S according to equation (2.29).

Obviously, the response is related to the excitation by some rule of correspondence. We can indicate this fact notationally as:

$$y(nT) = \Theta x(nT) \tag{2.30}$$

where Θ is an operator. Equation 2.30 can be treated as a generic definition for the Quality of Service adaptors that we will discuss in later chapters.

Like other digital systems [10], adaptors can be classified with respect to *time-invariance*, *causality*, *linearity* and *recursiveness*.

Time Invariance

An adaptor is said to be *time-invariant* if its response to an arbitrary excitation does not depend on the time of application of the excitation. As in other types of systems, the response of an

adaptor depends on a number of internal system parameters. In a time-invariant adaptor, these parameters do not change with time. Formally, an adaptor with excitation $x(nT)$ and response $y(nT)$, such that $x(nT) = y(nT) = 0$ for $n < 0$, is said to be time-invariant if and only if

$$\Theta x(nT - kT) = y(nT - kT) \tag{2.31}$$

holds for all possible excitations $x(nT)$ and all integers k. In other words, in a time-invariant adaptor, the response produced if the excitation $x(nT)$ is delayed by a period kT is numerically equal to the original response $y(nT)$ delayed by a period kT.

For example, $y(nT) = \Theta x(nT) = 2nTx(nT)$ is not a time-invariant adaptor, since $\Theta x(nT - kT) = 2nTx(nT - kT) \neq y(nT - kT) = 2(nT - kT)x(nT - kT)$. On the other hand, $y(nT) = \Theta x(nT) = 2x[(n-1)T] + x[(n-2)T]$ is time-invariant, since $\Theta x(nT - kT) = 2x[(n-k)T - T] + x[(n-k)T - 2T] = y(nT - kT)$.

Causality

A *causal* adaptor is one whose response at a specific instant is independent of subsequent values of the excitation. More precisely, an adaptor in which $x(nT) = y(nT) = 0$ for $n < 0$ is said to be causal if and only if

$$\Theta x_1(nT) = \Theta x_2(nT) \ for \ n \leq k \tag{2.32}$$

for all possible distinct excitations $x_1(nT)$ and $x_2(nT)$ such that

$$x_1(nT) = x_2(nT) \ for \ n \leq k \tag{2.33}$$

Conversely, if

$$\Theta x_1(nT) \neq \Theta x_2(nT) \ for \ n \leq k \tag{2.34}$$

for at least one pair of distinct excitations $x_1(nT)$ and $x_2(nT)$ such that

$$x_1(nT) = x_2(nT) \ for \ n \leq k \tag{2.35}$$

then the adaptor is not causal.

The above causality definition can be easily justified. If all possible pairs of excitations $x_1(nT)$ and $x_2(nT)$ that satisfy $x_1(nT) = x_2(nT)$ produce responses that are equal at instants $nT \leq kT$, then the adaptor response must depend only on values of the excitation at instants prior to nT, where $x_1(nT)$ and $x_2(nT)$ are assumed to be equal, and the adaptor is *causal*. Conversely, if at least two *distinct* excitations $x_1(nT)$ and $x_2(nT)$ that satisfy $x_1(nT) = x_2(nT)$ produce responses that are not equal at instants $nT \leq kT$, then the adaptor response must depend on values of the excitation at instants subsequent to nT, since the differences between $x_1(nT)$ and $x_2(nT)$ occur after nT, and the adaptor is not causal.

For example, $y(nT) = \Theta x(nT) = 3x[(n-2)T] + 3x[(n+2)T]$ is not causal. On the other hand, $y(nT) = \Theta x(nT) = 3x[(n-1)T] - 3x[(n-2)T]$ is causal.

Linearity

An adaptor is *linear* if and only if it satisfies the conditions

$$\Theta \alpha x(nT) = \alpha \Theta x(nT) \tag{2.36}$$

$$\Theta[x_1(nT) + x_2(nT)] = \Theta x_1(nT) + \Theta x_2(nT) \tag{2.37}$$

for all possible values of α and all possible excitations $x_1(nT)$ and $x_2(nT)$. These conditions are also referred to as the *homogeneity* and *additivity* conditions.

The response of a linear adaptor to an excitation $\alpha x_1(nT) + \beta x_2(nT)$, where α and β are arbitrary constants, can be expressed as

$$y(nT) = \Theta[\alpha x_1(nT) + \beta x_2(nT)] = \Theta \alpha x_1(nT) + \Theta \beta x_2(nT) = \alpha \Theta x_1(nT) + \beta \Theta x_2(nT) \tag{2.38}$$

Therefore, the above two conditions can be combined into one as

$$\Theta[\alpha x_1(nT) + \beta x_2(nT)] = \alpha \Theta x_1(nT) + \beta \Theta x_2(nT) \tag{2.39}$$

If this condition is violated for any pair of excitations or any constant α or β, then the adaptor is *nonlinear*. For example, while $y(nT) = \Theta x(nT) = 3x(nT - 2T)$ is linear, obviously $y(nT) = \Theta x(nT) = x^2(nT - T)$ is nonlinear.

Recursiveness

In this thesis, we only address the analysis and configuration of the subset of adaptors that are time-invariant, causal and linear. Similar to the analog control systems in the control theory [22] that are characterized in terms of *differential equations*, we characterize this subset of adaptors in terms of *difference equations*. There are two types of adaptors that can be identified, *non-recursive* and *recursive* adaptors.

The response of a generic nonrecursive adaptor at instant nT is of the form

$$y(nT) = f\{\ldots, x(nT - T), x(nT), x(nT + T), \ldots\} \tag{2.40}$$

If we assume linearity and time invariance, $y(nT)$ can be expressed as

$$y(nT) = \sum_{i=-\infty}^{\infty} a_i x(nT - iT) \tag{2.41}$$

where a_i represents constants. Now on assuming that the adaptor is causal and noting that $x(nT + T)$, $x(nT + 2T)$, ... are subsequent values of the excitation with respect to instant nT, we must have

$$a_i = 0 \ for \ i \leq -1 \tag{2.42}$$

and so

$$y(nT) = \sum_{i=0}^{\infty} a_i x(nT - iT) \tag{2.43}$$

If, in addition, $x(nT) = 0$ for $n < 0$ and $a_i = 0$ for $i > N$, we have

$$y(nT) = \sum_{i=0}^{N} a_i x(nT - iT) \tag{2.44}$$

Therefore, a linear, time-invariant, causal and nonrecursive adaptor can be represented by an

Nth-order linear difference equation. N is referred to as the *order* of the adaptor.

Similarly, the response of a recursive adaptor is a function of elements in the excitation as well as the response sequence. In the case of a linear, time-invariant, causal adaptor, we have

$$y(nT) = \sum_{i=0}^{M} a_i x(nT - iT) - \sum_{i=1}^{N} b_i y(nT - iT) \tag{2.45}$$

i.e. if instant nT is taken to be the present, the present response is a function of the present and past M values of the excitation as well as the past N values of the response. Note that if we let $b_i = 0$, nonrecursive adaptors is obviously a special case of the recursive adaptors.

For simplicity, if expressed in the form of $x(n)$ instead of $x(nT)$ and $y(n)$ instead of $y(nT)$, and in a more concise form, we have

$$\sum_{i=0}^{N} a_i y(n - i) = \sum_{i=0}^{M} b_i x(n - i) \tag{2.46}$$

2.5 Conclusion

From the analysis discussed in this chapter, we developed a suitable simplified but unified model for Quality of Service metrics, Quality of Service specification mechanisms, and Quality of Service adaptations. Once we can utilize this model to illustrate the complex dynamic behavior of adaptive Quality of Service delivery, we can focus in the later chapters on discussions addressing only this abstract model, with the assumption that it will model without loss of generality a wide range of basic Quality of Service adaptive behaviors that we study. We reasonably ignore the details in the adaptive behaviors that the model cannot describe. We will come back to these details in our discussion of implementation issues.

Chapter 3

Configuring Quality of Service Adaptors

3.1 Overview

In previous chapters, we decided to model Quality of Service adaptors in the middleware level using *difference equations*, and only to address a subset of adaptors that are time-invariant, causal and linear. The function of the Quality of Service adaptors is to apply adaptations on a particular QoS metric, such as *system rate*, so that to provide the applications with adapted Quality of Service, such as graceful service degradation if the services of raw QoS degrades severely and rapidly. The adaptations are in essence some form of transformations, such as the transformation of one series of QoS metric values to another. These series of QoS metric values are modeled by discrete-time values, or *signals*, and the transformation can be modeled by difference equations provided its behavior is time-invariant, causal and linear. The general form of the difference equations that we address can be expressed as equation (2.46) in the previous chapter:

$$\sum_{i=0}^{N} a_i y(n-i) = \sum_{i=0}^{M} b_i x(n-i) \tag{3.1}$$

In this chapter, we will introduce theories in digital control systems [22] [10] to analyze and configure the adaptors modeled by the above difference equations. Specifically, we will utilize *z-transform*, an important mathematical transformation method introduced in digital systems control theory, and analyze the *frequency domain* responses of the adaptive behavior of an adaptor. The analysis for the adaptor will be focused on the *stability* of the adaptive behavior, and the configuration will be based on a specified *adaptation agility* or *sensitivity* in the application Quality of

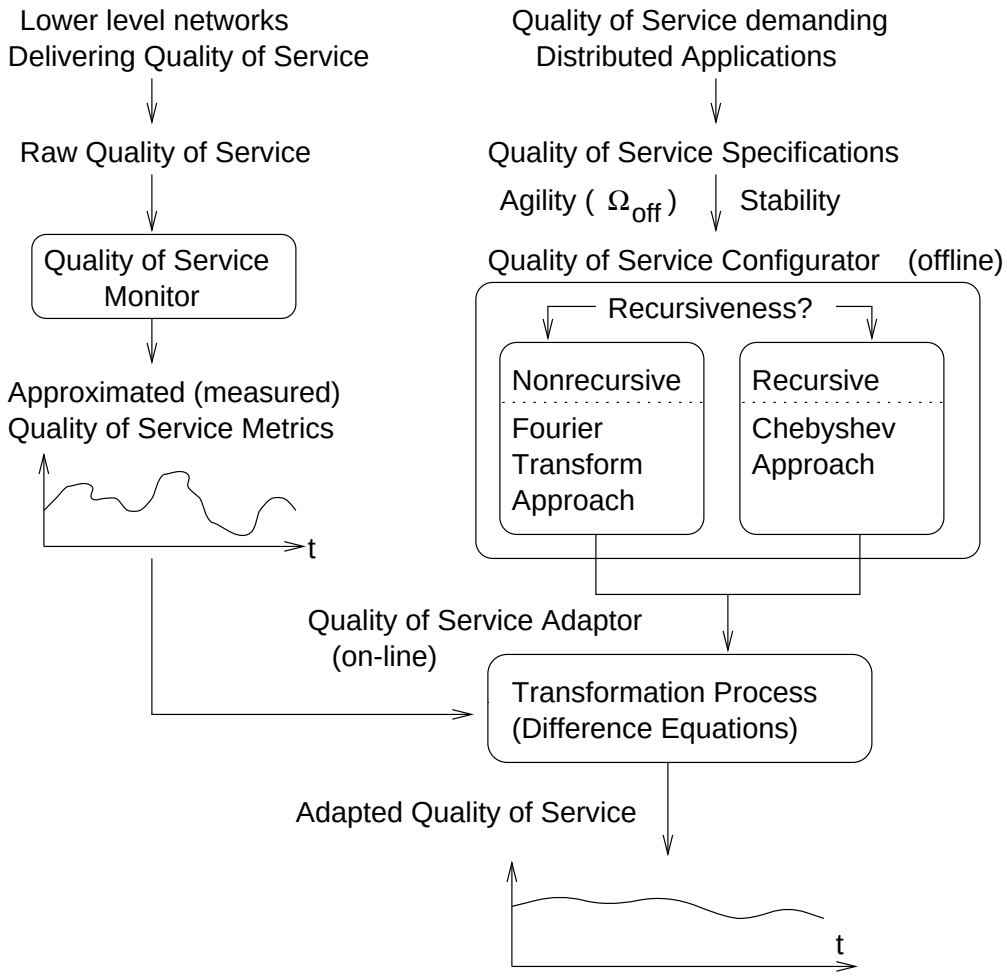

Figure 3.1: Configuring Quality of Service adaptors: an overview

Service specification. As a basis of further discussions, time-domain properties of the adaptor will also be addressed in this chapter.

We present an overview on the configuration process described above in figure 3.1. The configuration starts with the raw Quality of Service that is provided by the underlying transport facilities, as well as the Quality of Service specifications demanded by distributed applications. Applications specify a desired *adaptation agility* and *stability*, which we will further describe in later sections, and the Quality of Service configurator will configure adaptors according to a specific agility demanded the applications. As the focus of this chapter, we will present two approaches to configure both non-recursive or recursive adaptors. One of the common characteristics of these approaches is that they both operate in the frequency domain, thus requiring the result of the configuration be converted back to the time domain in the form of difference equations. These

difference equations will precisely model the transformation process made by the adaptors, generating the final *adapted Quality of Service* that is delivered to the applications.

3.2 Time Domain Analysis and the Convolution Summation Property

3.2.1 Overview

In order to analyze the time-domain properties of the difference equations that describe the behavior of the adaptors defined in equation (2.46), methods in *time domain* analysis of traditional analog control systems [22] use several *elementary functions* such as the *unit impulse* function, the *unit step* function, etc.

Similar functions also apply to the discrete-time difference equations. The discrete-time *unit step*, *unit ramp*, *exponential* and *sinusoid* functions are generated by letting $t = nT$ in the corresponding continuous-time functions. The discrete-time *unit impulse* $\delta(nT)$, however, is generated by letting $t = nT$ in the pulse function $p_{t_0}(t)$ given by

$$p_{t_0}(t) = \begin{cases} 1 & \text{for } |t| \leq t_0 < T \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Now we can introduce the definition to the discrete-time *unit impulse* $\delta(n)$ as follows:

$$\delta(n) = \begin{cases} 0 & n \neq 0 \\ \delta(n) = 1 & n = 0 \end{cases} \tag{3.3}$$

The role of $\delta(n)$ in discrete-time systems is analogous to the role of the continuous-time unit impulse $\delta(t)$ in analog systems. Nevertheless, the two functions are defined differently, and as a consequence $\delta(t)$ cannot be generated by letting $t = nT$ in $\delta(t)$. Given the above explanations, some frequently used discrete-time functions are illustrated in figure 3.2 [1]. The impulse function shown in the figure is $\delta(n - 0.1)$.

[1] The *x* axis of the graph denotes *time*, in the unit of *second*. Same applies to all similar graphs in this chapter. The *y* axis is normalized to the range of $[0, 1]$.

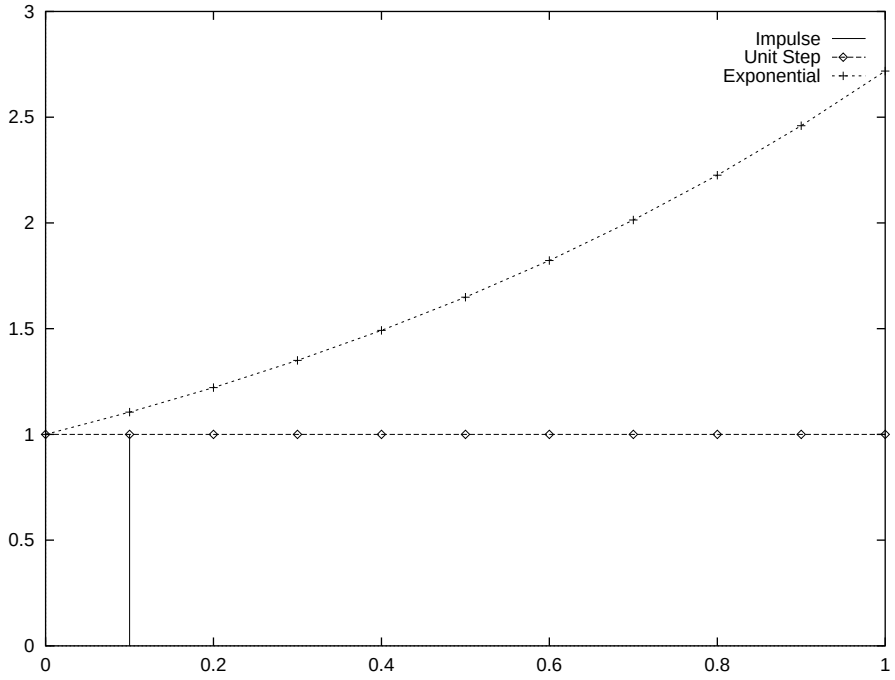

Figure 3.2: Elementary excitation functions: the *impulse*, *unit step*, and *exponential* function

We will frequently refer to the concept of *impulse response* of an adaptor. It is defined as the response of an adaptor when the excitation signal is a simple impulse function $\delta(n)$. As is shown in later sections, the impulse response is able to represent all the time domain properties of an adaptor and its difference equation. The impulse response function will be formally defined in equation 3.7.

As an example, assume we have an adaptor whose difference equation is recursive and can be expressed as:

$$y(n) = 1.5y(n-1) - 0.85y(n-2) + x(n) \tag{3.4}$$

It is easy to calculate the impulse response of this adaptor, which is illustrated in Figure 3.3.

Evidently, the time domain response of simple adaptors can be determined by solving the difference equation directly using *induction*. However, this approach is somewhat primitive. By applying the *Convolution Summation Property*, introduced in the next section, we are able to derive the response to any excitation signals by calculating convolution with the impulse response of the adaptor.

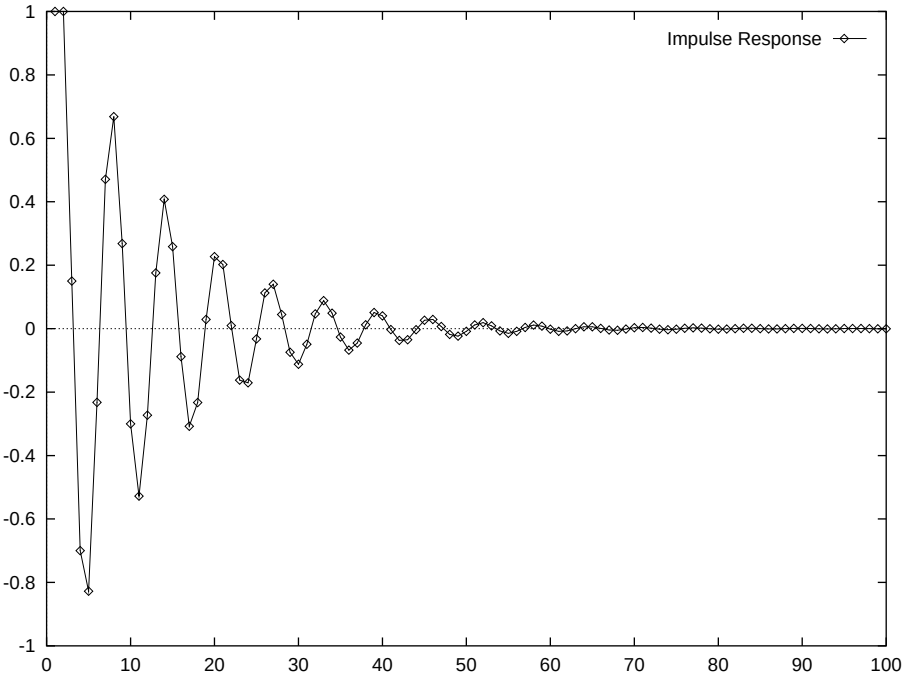

Figure 3.3: The *impulse response*: an example

3.2.2 Convolution Summation Property

It can be proved in control theory [22] that, the response of an adaptor to an arbitrary excitation can be expressed in terms of the impulse response of the adaptor. An excitation $x(n)$ can be written as

$$x(n) = \sum_{k=-\infty}^{\infty} x_k(n) \tag{3.5}$$

where

$$x_k(n) = \begin{cases} x(k) & \text{for } n = k \\ 0 & \text{otherwise} \end{cases}$$

Alternatively

$$x_k(n) = x(k)\delta(n-k)$$

and hence

$$x(n) = \sum_{k=-\infty}^{\infty} x(k)\delta(n-k) \tag{3.6}$$

We further formally define *impulse response function* $h(n)$, which was informally explained in the previous section:

$$h(n) = \Theta\delta(n) \tag{3.7}$$

where Θ is previously defined in Chapter 2, equation (2.30), which we repeat as below:

$$y(n) = \Theta x(n) \tag{3.8}$$

Substituting equation (3.6) in equation (3.8), we have

$$\begin{aligned} y(n) &= \Theta \sum_{k=-\infty}^{\infty} x(k)\delta(n-k) \\ &= \sum_{k=-\infty}^{\infty} x(k)\Theta\delta(n-k) \\ &= \sum_{k=-\infty}^{\infty} x(k)h(n-k) \\ &= \sum_{k=-\infty}^{\infty} h(k)x(n-k) \end{aligned} \tag{3.9}$$

where the second line is deduced by a simple change of variable, and the last line is deduced based on the proved [9] *commutative* property of convolution:

$$x(n) \bigotimes y(n) = y(n) \bigotimes x(n) \tag{3.10}$$

where the convolution operator $\bigotimes$ is defined by:

$$x(n) \bigotimes y(n) = \sum_{k=-\infty}^{\infty} x(k)y(n-k) \tag{3.11}$$

The relation shown in 3.9, known as the *convolution summation* property, is of considerable importance in the characterization as well as analysis of adaptors and is an important part in the time domain analysis in the control theory [22]. Given this property, we can derive the response

of an adaptor to any excitation signals, once we know the impulse response $h(n)$ of the adaptor. This demonstrates that the impulse response $h(n)$ fully represents all the time domain properties of the transformation process of an adaptor.

As an example, consider a simple excitation signal with two different frequencies of sinusoids, namely:

$$x(n) = \sin(\frac{\pi n}{30}) + \sin(\frac{\pi n}{5}),\ 60 \leq n \leq 320 \tag{3.12}$$

and a simple moving-average adaptor, whose recursive difference function is expressed by [2]:

$$y(n) = y(n-1) + \frac{1}{10}[x(n) - x(n-10)] \tag{3.13}$$

Its *impulse response* function $h(n)$ is obviously:

$$h(n) = \begin{cases} \frac{1}{10} & \text{for } 0 \leq n \leq 9 \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

By applying the *convolution summation* property expressed in equation 3.9, we can calculate the response, as illustrated in Figure 3.4. As illustrated, the adaptor eliminates the high frequency perturbations in the excitation signal, and generates response to represent the low frequency long term trends of the excitation.

3.3 Frequency Domain Analysis and Adaptation Agility

3.3.1 Frequency Spectrum of the Excitation

It is understood that the excitation to an adaptor can be modeled as a discrete-time signal series, and the transformations performed by the adaptor can be modeled as difference equations. As stated in the previous section, it is relatively straightforward to analyze the functionalities of the adaptor in the time domain, i.e., to model the response as a discrete-time function of time. To produce such a function, we only need to apply the difference equation to the current and previous

[2] This adaptor can also be expressed in non-recursive difference function as: $y(n) = 0.1[x(n)+x(n-1)+\ldots+x(n-9)]$.

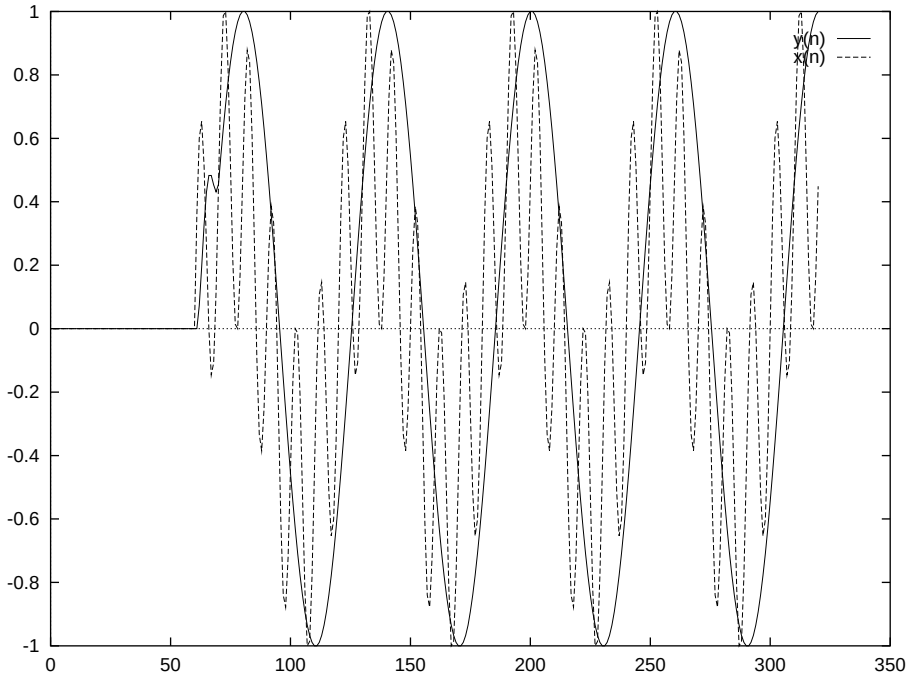

Figure 3.4: The Convolution Summation Property: an example

values of excitations and responses, thus yield the current value of response. Alternatively, we can also apply the *convolution summation* property if we know the impulse response function $h(n)$ of the transformation.

However, the analysis of the behavior of an adaptor in the time domain can not reveal the full details of characteristics of the transformation that the adaptor performs. One of the most important characteristics that cannot normally be described is the *frequency spectrum* of the excitation signal and the *frequency response* of the transformation.

Most excitation signals can be characterized as *aperiodic*, which is the case for the excitation input to the adaptors. This type of excitation signals can normally be analyzed by applying *Fourier Transform* to the excitation signal and thus generate its frequency spectrum. Since the *Fourier Transform* is normally used for the frequency analysis for *periodic* and *continuous-time* signals, we need to modify it to cope with aperiodic discrete-time signals.

The *analysis equation* of the *Discrete Fourier Series* can be defined as:

$$c_k = \frac{1}{N} \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi kn}{N}} \tag{3.15}$$

in which c_k represents the kth spectral component, and N is the number of sample values in each period of the signal.

Conversely, if we know the coefficients c_k, we may regenerate $x(n)$ using the *synthesis equation* defined as:

$$x(n) = \sum_{k=0}^{N} c_k e^{\frac{j2\pi kn}{N}} \tag{3.16}$$

The above analysis and synthesis equations apply, apparently, to a strictly periodic signal with period N. However, it is possible to extend the equations to apply to aperiodic signals. We could make $N \to \infty$, thus yield a continuous, rather than discrete, distribution of spectral energy. Although each spectral coefficient becomes vanishingly small as $N \to \infty$, the product Nc_k remains finite. Let $X = Nc_k$, $\Omega = \frac{2\pi k}{N}$, and think of them as continuous frequency variables, we have

$$X(\Omega) = Nc_k = \sum_{n=-\infty}^{\infty} x(n)e^{-j\Omega n} \tag{3.17}$$

This analysis equation defines the Fourier Transform $X(\Omega)$ of the aperiodic signal $x(n)$. Using similar arguments and substitutions, we can develop the *inverse transform* from the synthesis equation previously shown, as:

$$x(n) = \frac{1}{2\pi} \int_{2\pi} X(\Omega)e^{j\Omega n} d\Omega \tag{3.18}$$

Having defined the analysis and synthesis equations for aperiodic discrete-time signals, we will be able to fully generate and analyze the *frequency spectrum* of an excitation signal.

3.3.2 Frequency Response of the Transformation

We have discussed the analysis and synthesis of the frequency spectrum of an aperiodic signal, the problem left to be addressed is the application of *Fourier Transform* to analyze the frequency domain performance of the transformations performed by the adaptor.

By applying the analysis equation addressed in the previous section, we can derive the frequency spectrum of an unit impulse $\delta(n)$ defined in equation 3.3 at $n = 0$:

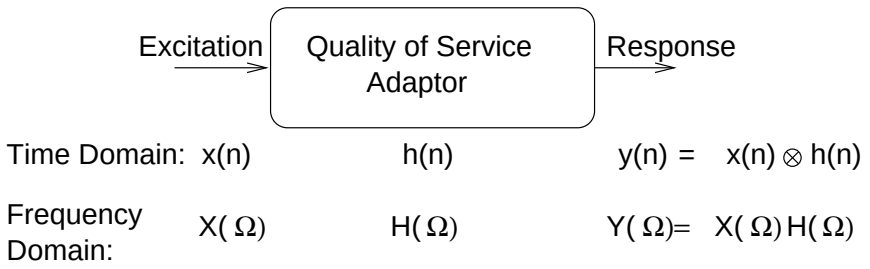

Figure 3.5: A comparison between time and frequency domain analysis

$$X(\Omega) = \sum_{n=-\infty}^{n=\infty} \delta(n) e^{-j\Omega n} = e^{-j\Omega n}|_{n=0} = 1 \tag{3.19}$$

This shows that $\delta(n)$ contains an equal amount of all frequencies in its frequency spectrum. It could be synthesized from an infinite set of cosines, all of vanishingly small, but equal, amplitudes.

We also note that unlike the case in the time domain, where the excitation signal $x(n)$ is *convolved* with the time domain response of the transformation $h(n)$ to produce the output signal $y(n)$ as demonstrated in equation (3.9), the equivalent frequency domain process must be *multiplication*. The frequency spectrum of the response signal $Y(\Omega)$ can be expressed as

$$Y(\Omega) = X(\Omega)H(\Omega) \tag{3.20}$$

where $H(\Omega)$ is the *frequency response* of the transformation. Figure 3.5 demonstrates a comparison between time domain analysis and frequency domain analysis on an adaptor. Since we demonstrated that the frequency spectrum of the unit impulse is unity, that is, $X(\Omega) = 1$, we then have

$$Y(\Omega) = X(\Omega)H(\Omega) = H(\Omega) \tag{3.21}$$

This shows that the *frequency response* of a transformation is the Fourier Transform of the output response signal $y(n)$ after applying the transformation on an excitation signal equivalent to the *unit impulse*. It is easy to see the fact that since an unit impulse contains an equal amount of all frequencies, when used as an input signal, it simultaneously probes the system's response to all possible input frequencies.

As an example, we take equation 3.4 as the recursive difference equation of the adaptor and

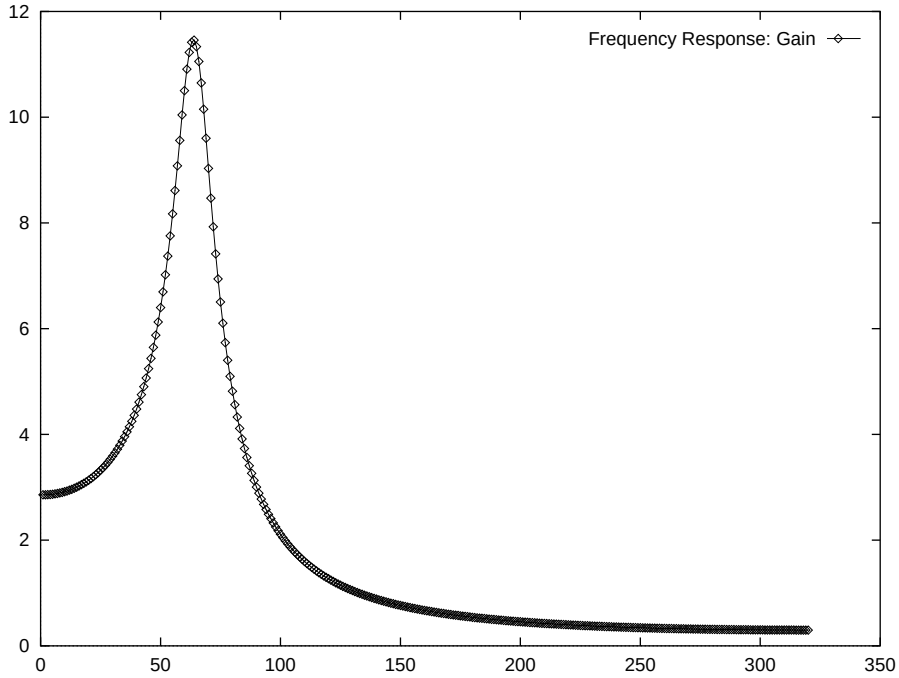

Figure 3.6: The gain function of the *frequency response*: an example

analyze its frequency response, compared with its time domain impulse response illustrated in figure 3.3. Using equation (3.20) we will have:

$$\begin{aligned} H(\Omega) &= \frac{Y(\Omega)}{X(\Omega)} \\ &= \frac{\sum_{k=0}^{N} a_k e^{-jk\Omega}}{\sum_{k=0}^{M} b_k e^{-jk\Omega}} \\ &= \frac{1}{1 - 1.5e^{-j\Omega} + 0.85e^{-j2\Omega}} \\ &= \frac{1}{(1 - 1.5\cos\Omega + 0.85\cos 2\Omega) + j(1.5\sin\Omega - 0.85\sin 2\Omega)} \end{aligned} \tag{3.22}$$

The gain and phase functions are therefore:

$$|H(\Omega)| = \frac{1}{\{(1 - 1.5\cos\Omega + 0.85\cos 2\Omega)^2 + (1.5\sin\Omega - 0.85\sin 2\Omega)^2\}^{\frac{1}{2}}} \tag{3.23}$$

$$\Phi_H(\Omega) = \arctan\left(\frac{1.5\sin\Omega - 0.85\sin 2\Omega}{1 - 1.5\cos\Omega + 0.85\cos 2\Omega}\right) \tag{3.24}$$

The result from equation (3.23) is visually illustrated in figure 3.6. This figure demonstrates the

frequency domain impulse response $H(\Omega)$, which is obviously different from the impulse response $h(n)$ in the time domain illustrated in figure 3.3, and fully represents the properties of an adaptor in the frequency domain.

3.3.3 Adaptation Agility

As addressed in the first chapter, for distributed multimedia applications in need of Quality of Service support, we need adaptors serving in the middleware level to be fully configurable by the application that demands Quality of Service. In order to compare the adaptive capabilities of two adaptors and to configure them, one of the most important metrics is *adaptation agility*, or *sensitivity*, which represents the ability and extent of an adaptor to promptly respond to perturbations in the raw Quality of Service from the lower layers.

An adaptor that is too *agile* may suffer from instability. Such an adaptor consumes almost all its resources reacting to minor perturbations, hence taking excessive computational resources from the system. This behavior is apparently not desired. An adaptor that is not *agile* enough may consume too much temporary resources such as buffer space to transparently adapt to most of the fluctuations in the excitation signals and keep the response undisturbed. It is important for the application to specify a suitable value for the agility of the adaptor. Based on the model that we developed in previous chapters, we are able to quantitatively model the transformation performed by an adaptor, thus makes it possible to precisely define the *adaptation agility* of the adaptor on which configuration may be based on.

It is understood that the basic functionality that the adaptor should provide is to filter the rapid perturbations and high frequency short-term fluctuations from the long-term trends of changes in the excitation signal. An example of this was illustrated in section 3.2.2 and figure 3.4. In the frequency domain that we discussed in the previous section, this may be expressed as a *low pass adaptor*, the transformation of which screens the incoming frequency spectrum and only passes low frequency components, eliminating all high frequency components of the signal. The time-domain effects of this transformation is that all the high frequency short-term perturbations are eliminated to stress the long-term low frequency changes, which is precisely the desired adaptation effects for the adaptor.

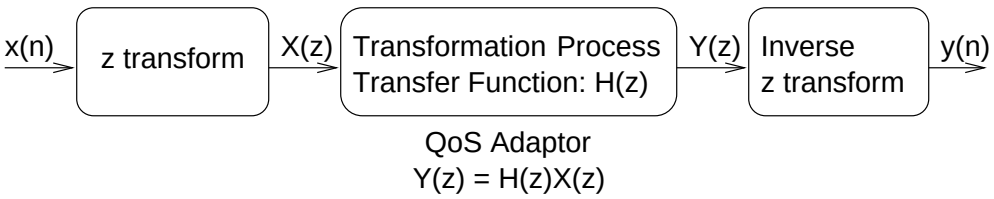

Figure 3.7: Modeling the transformation process using z transform

Given above, the *adaptation agility*, or *sensitivity* of an adaptor can be defined as the cut-off frequency Ω_{off} of the *frequency response* of a transformation applied by the adaptor. The cut-off frequency Ω_{off} is illustrated in figure 3.9 and will be discussed later in section 3.6. Defined as such, we can therefore configure the adaptor to provide a specific agility required and specified by the application. The result of configuration should be an adaptor showing the agility, or cut-off frequency, required by the application. The configuration process will be addressed in details in later sections.

However, the configuration process is not trivial. We need to utilize the knowledge of *z Transform* for this purpose.

3.4 Frequency Domain Analysis: the *z* Transform

3.4.1 Overview

In order to configure the adaptive behavior to conform to the required agility from an application, we need to introduce the z transform as a valuable set of techniques for the frequency analysis on excitation signals and transformations themselves. The z transform and the Fourier Transform, which was introduced in previous sections, are closely related, and should be regarded as complementary to one another. They all model the analysis of the frequency domain behavior. However, the z transform is inherently concerned with discrete-time signals, whereas digital Fourier Transform was derived from the continuous-time analysis techniques.

The z transform, similar to the case for the Fourier transform, is useful because it has an inverse z transform. The application of the z-transform to a discrete-time signal $x(n)$ yields a representation of the signal in terms of a rational function $X(z)$ where z is a complex variable. if the signal is to be processed by an adaptor, then the required processing can be carried out in the z do-

main through algebraic manipulations. In this way, a transformed version of $X(z)$, say $Y(z)$, is obtained. Consequently, by applying the inverse z transform to $Y(z)$, the time-domain response $y(n)$ is obtained. Figure 3.7 illustrates the transformation process using the z transform.

3.4.2 Definitions

The z *transform* of a discrete-time function $x(n)$ is defined as

$$X(z) = \mathcal{Z}x(n) = \sum_{n=-\infty}^{\infty} x(n)z^{-n} \tag{3.25}$$

for all z for which $X(z)$ converges. Conversely, $x(n)$ can be uniquely determined as

$$x(n) = \mathcal{Z}^{-1}X(z) = \frac{1}{2\pi j}\oint_{\Gamma} X(z)z^{n-1}dz \tag{3.26}$$

where Γ is defined as a contour in the counterclockwise sense enclosing all the *singularities*, also referred to as *poles*, of $X(z)z^{n-1}$. Equation (3.26) is referred to as the *inverse z transform* of $X(z)$.

3.4.3 Applications

Originally, the transformation performed by an adaptor was modeled by difference equations, which is inconvenient for the analysis and synthesis of major transformation characteristics such as *stability*. However, through the use of the z transform, the transformation of an adaptor can be characterized by a discrete-time *transfer function*, which profoundly facilitates time domain and frequency domain analysis on the transformation behavior.

The *transfer function* of an adaptor is defined as the ratio of the z transform of the response to the z transform of the excitation. Consider a linear, time-invariant adaptor, and let $x(n)$, $y(n)$ and $h(n)$ be the excitation, response, and impulse response, respectively. It is understood that in frequency domain analysis the response is obtained by *multiplication* of the excitation signal and the frequency response of the adaptor, the property also holds for z transform:

$$Y(z) = H(z)X(z) \tag{3.27}$$

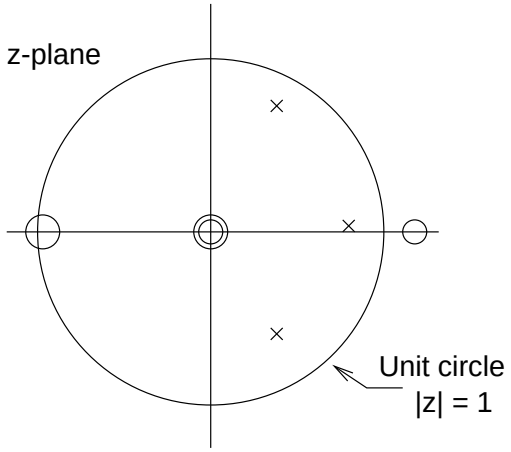

Figure 3.8: Poles and zeros on the z-plane: an example

In effect, the *transfer function* of an adaptor is the z transform of the impulse response $H(z)$. It can be put in the form

$$H(z) = \frac{Y(z)}{X(z)} = \frac{N(z)}{D(z)} = \frac{K\Pi_{i=1}^{N}(z - z_i)}{\Pi_{i=1}^{M}(z - p_i)} \tag{3.28}$$

where z_1, z_2, ..., z_N are referred to as the *zeros* and p_1, p_2, ..., p_N are referred to as the *poles* of $H(z)$, and m_i is the order of pole p_i.

A very useful representation of a z transform is to obtained by plotting its poles and zeros in the complex plane. The plane is then referred to as the z-plane. The z-plane plot is very useful for determining the *adaptation stability* which will be discussed in the next section.

For example, assume an adaptor has a *transfer function* as follows:

$$H(z) = \frac{z^2(z - 1.2)(z + 1)}{(z - 0.5 + j0.7)(z - 0.5 - j0.7)(z - 0.8)} \tag{3.29}$$

From the transfer function we learn that the adaptor has 4 zeros at 0, 0, 1.2 and -1, respectively. It also has 3 poles at $0.5 \pm j0.7$ and 0.8, respectively. We can thus plot these points on the z-plane as illustrated in figure 3.8.

3.5 Adaptation Stability

Intuitively, all the adaptations made by an adaptor should be *stable*, that is, any reasonable excitation signals should not yield an unbounded response that will keep fluctuate forever. This

is a basic requirement that must be imposed on the configuration of any adaptors.

3.5.1 Definition

An adaptor is said to be *stable* if and only if any bounded excitation results in a bounded response, i.e., if

$$|x(n)| < \infty \; for \; all \; n \tag{3.30}$$

implies that

$$|y(n)| < \infty \; for \; all \; n \tag{3.31}$$

For a linear, time-invariant adaptor, the time domain convolution summation property of equation (3.9) concludes that

$$y(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k) \tag{3.32}$$

we then have

$$|y(n)| \leq \sum_{k=-\infty}^{\infty} |h(k)||x(n-k)| \tag{3.33}$$

and if

$$|x(n)| \leq M \leq \infty \; for \; all \; n \tag{3.34}$$

we have

$$|y(n)| \leq M \sum_{k=-\infty}^{\infty} |h(k)| \tag{3.35}$$

Clearly if

$$\sum_{k=-\infty}^{\infty} |h(k)| < \infty \tag{3.36}$$

then

$$|y(n)| < \infty \; for \; all \; n \tag{3.37}$$

and therefore, equation (3.36) constitutes a sufficient condition for stability.

An adaptor can be classified as stable only if its response is bounded for all possible bounded

excitations. Consider the bounded excitation

$$x(n-k) = \begin{cases} M & \text{if } h(k) \geq 0 \\ -M & \text{if } h(k) < 0 \end{cases} \tag{3.38}$$

where *M* is a positive constant. From equation (3.9) we have

$$y(n) = |y(n)| = \sum_{k=-\infty}^{\infty} M|h(k)| \tag{3.39}$$

or

$$|y(nT)| = M \sum_{k=-\infty}^{\infty} |h(k)| \tag{3.40}$$

Evidently, the response will be bounded if and only if equation (3.36) holds and, therefore, equation (3.36) constitutes a necessary and sufficient condition for stability.

3.5.2 Judging Criteria

Consider an adaptor characterized by the transfer function of equation (3.28). The time domain impulse response $h(n)$ of such an adaptor is given by equation (3.26) as

$$h(n) = \mathcal{Z}^{-1}H(z) = \frac{1}{2\pi j}\oint_{\Gamma} H(z)z^{n-1}dz \tag{3.41}$$

We may express all the *poles* of $H(z)$ defined in equation (3.28) in polar forms:

$$p_i = r_i e^{j\phi_i},\ for\ i = 1, 2, \ldots, N \tag{3.42}$$

where r_i is the radius and ϕ_i is the angle.

we can prove that [10]:

$$r_i < 1\ for\ i = 1, 2, \ldots, N \tag{3.43}$$

is the necessary and sufficient condition for deciding that the adaptor is stable.

If we plot all the poles in the z-plane as described in section 3.4.3 and figure 3.8, the condition in equation (3.43) satisfies if and only if all poles p_i is located inside the unit circle $|z| = 1$. For

example, the transfer function in equation (3.29) satisfies the stability condition (3.43), thus can be determined as a stable adaptor. As illustrated in figure 3.8, all the poles locate inside the unit circle $|z| = 1$. The necessary and sufficient condition in equation (3.43) is proved to be a very simple but useful judging criteria to judge the stability of an adaptor.

3.6 Configuring Adaptation Agility: the Fourier Transform Approach

3.6.1 Overview

As noted in section 3.3.1, *Discrete Fourier Transform* is widely used to analyze the frequency spectrum of a series of excitation signals or the frequency response of a transformation. In fact, the Discrete Fourier Transform, when combined with its inverse transform, can not only be applied to frequency domain analysis, but also to the configuration of adaptation agility of an non-recursive adaptor.

The general form of difference equation for a causal, linear and time-invariant adaptor was first introduced in equation (2.44) as:

$$y(n) = \sum_{k=0}^{N} a_k x(n-k) \tag{3.44}$$

Such non-recursive adaptors has the advantageous property that it implements the convolution sum directly, and the coefficients a_k are simply equal to successive terms in its impulse response $h(n)$.

According to equation (3.25) and equation (3.17), the transfer function $H(z)$ and frequency response $H(\Omega)$ corresponding to a difference equation of equation (3.44) are, respectively:

$$H(z) = \sum_{k=0}^{N} h(k) z^{-k} = \sum_{k=0}^{N} a_k z^{-k} \tag{3.45}$$

and

$$H(\Omega) = \sum_{k=0}^{N} h(k) e^{-jk\Omega} = \sum_{k=0}^{N} a_k e^{-jk\Omega} \tag{3.46}$$

The approach is conceptually straightforward. If we start with a desired adaptation agility,

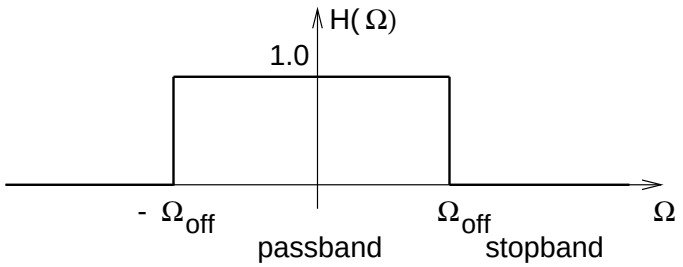

Figure 3.9: An ideal low-pass adaptor with cut-off frequency Ω_{off}

which is defined as a cut-off frequency Ω_{off} [3] as in section 3.3.3, we will be able to deduce a desired frequency response $H(\Omega)$. Equation (3.18) defines the inverse Discrete Fourier Transform, which is able to derive the corresponding impulse response $h(n)$ from the desired frequency response $H(\Omega)$:

$$x(n) = \frac{1}{2\pi} \int_{2\pi} X(\Omega) e^{j\Omega n} d\Omega \tag{3.47}$$

When rewriting it to describe the frequency response of a transformation made by an adaptor, rather than an excitation signal, we have:

$$h(n) = \frac{1}{2\pi} \int_{2\pi} H(\Omega) e^{j\Omega n} d\Omega \tag{3.48}$$

If we start with a desired frequency response $H(\Omega)$, equation (3.48) shows how to derive the corresponding impulse response $h(n)$. It is explained previously that the sample values of $h(n)$ are identical with the required multiplier coefficients a_k for the difference equation representing the non-recursive adaptor.

There are two major difficulties in this approach. First, if $H(\Omega)$ has a complicated form, the integral in equation (3.48) is not always easy to solve. This problem can be addressed by concentrating on simple, idealized magnitude characteristics of the frequency response, shown in figure 3.9, and will assume linear phase responses.

The second difficulty concerns the number of terms in $h(n)$. Our choice of $H(\Omega)$ as shown in figure 3.9 may result in an impulse response with infinite or a large number of terms, making it not realistic and economic. We must clearly have some way of limiting the number of coefficients,

[3] Ω_{off} is also visually illustrated in the ideal frequency response in figure 3.9.

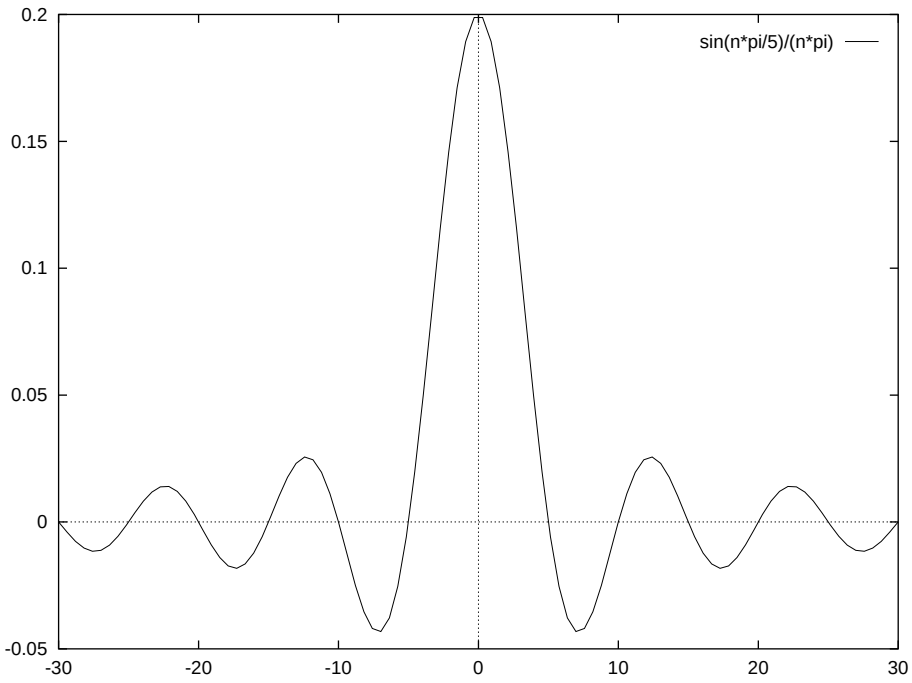

Figure 3.10: The *frequency response* to ideal low-pass adaptors: an example

and settling for a compromise between time domain and frequency domain performance.

Starting with the ideal low pass characteristics of figure 3.9, we can assume a zero-phase system for which $H(\Omega)$ is real, and assume unity gain between frequencies $\pm\Omega_{off}$. Equation (3.48) specifies integration over any convenient 2π interval, we defined $H(\Omega)$ over the range $\Omega = -\pi$ to π, rather then 0 to 2π, to simplify the integral in equation (3.48). Thus:

$$\begin{aligned} h(n) &= \frac{1}{2\pi}\int_{-\pi}^{\pi} H(\Omega)e^{j\Omega n}d\Omega \\ &= \frac{1}{2\pi}\int_{-\Omega_{off}}^{\Omega_{off}} 1\cdot e^{j\Omega n}d\Omega = \frac{1}{2\pi}\left[\frac{e^{j\Omega n}}{jn}\right]_{-\Omega_{off}}^{\Omega_{off}} \\ &= \frac{1}{2\pi jn}\{e^{j\Omega_{off}n} - e^{-j\Omega_{off}n}\} \\ &= \frac{1}{n\pi}\sin(n\Omega_{off}) \end{aligned} \tag{3.49}$$

For example, if we specify Ω_{off} to be $\frac{\pi}{5}$, using equation (3.49) we have

$$h(n) = \frac{1}{n\pi}\sin(\frac{n\pi}{5}) \tag{3.50}$$

We illustrate the result of equation (3.50) in figure 3.10.

Although the impulse response shown in figure 3.10 decay to either side of n=0, they theoretically continue infinitely in both directions. This reflects the general antithesis between band limitation and time limitation: since we have chosen a frequency response $H(\Omega)$ with an infinitely sharp cut-off curve, the time-domain response continues infinitely. To realize such an adaptor we need to *truncate* the impulse response using a *windowing function*. The simplest windowing function is to ignore the infinitely small sample values toward both ends. After truncating, we can then shift $h(n)$ to begin at $n = 0$, yielding a causal, linear-phase, adaptor. Obviously, the more terms remained in the final difference equation, the more accurate its actual frequency response approximates the ideal magnitude characteristics in figure 3.9.

3.6.2 The Windowing Functions

When we truncate an infinite-length impulse response, the process is equivalent to multiplying it by a finite-length *windowing function*. In other words, the final difference equation coefficients are derived by time-domain multiplications. The modulation property [9] of the Fourier Transform shows that time-domain multiplication is equivalent to frequency-domain convolution. Therefore, the truncating process in the time domain will produce an *actual* frequency response $H_A(\Omega)$ which is the convolution of the *desired* frequency response $H_D(\Omega)$ with the frequency spectrum of the window $W(\Omega)$, namely

$$H_A(\Omega) = H_D(\Omega) \otimes W(\Omega) \tag{3.51}$$

If we use a simple windowing function, the *rectangular* window, defined as:

$$w(n) = \begin{cases} 1 & |n| \leq M \\ 0 & |n| > M \end{cases} \tag{3.52}$$

As we have shown in equation (3.49), the spectrum of the rectangular windowing function tends to the $\frac{\sin x}{x}$ form, thus its convolution with $H_D(\Omega)$ gives an approximation to the desired frequency response containing a number of fluctuations which we normally referred to as *ripples*. These ripples will distort the shape of the passband response of the frequencies, and produce

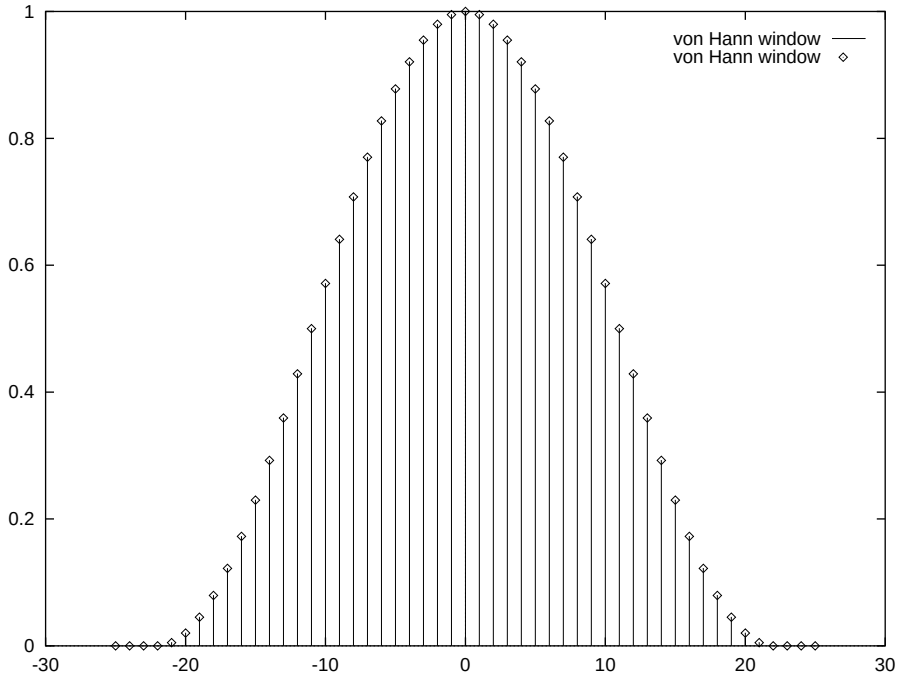

Figure 3.11: A 51-term *von Hann* windowing function

unwanted sidelobes. [4]

The main reason for the poor performance of the rectangular windowing function stems from the trade-off antithesis between the time domain and the frequency domain. As known in the control theory [22], an unit impulse in the frequency domain maps to an infinitely long time windowing function in the time domain, and vice versa. Intuitively, if the windowing function suddenly chops off in the time domain, it tends to spread out in the frequency domain. Consequently, we may expect that a windowing function with more gentle curves in the time domain will introduce less ripples and sidelobes in the frequency domain.

In theories related to digital signal processing [9], two widely accepted windowing functions are referred to as the *von Hann* window and the *Hamming* window. The definition for the *von Hann* window is:

$$w(n) = \begin{cases} \frac{1}{2} + \frac{1}{2}\cos\frac{n\pi}{M+1}, & -M \leq n \leq M \\ 0 & \text{elsewhere.} \end{cases} \tag{3.53}$$

[4]The term *passband*, as well as *stopband* which will be referred later, denote the range of frequencies being passed or eliminated in the frequency response.

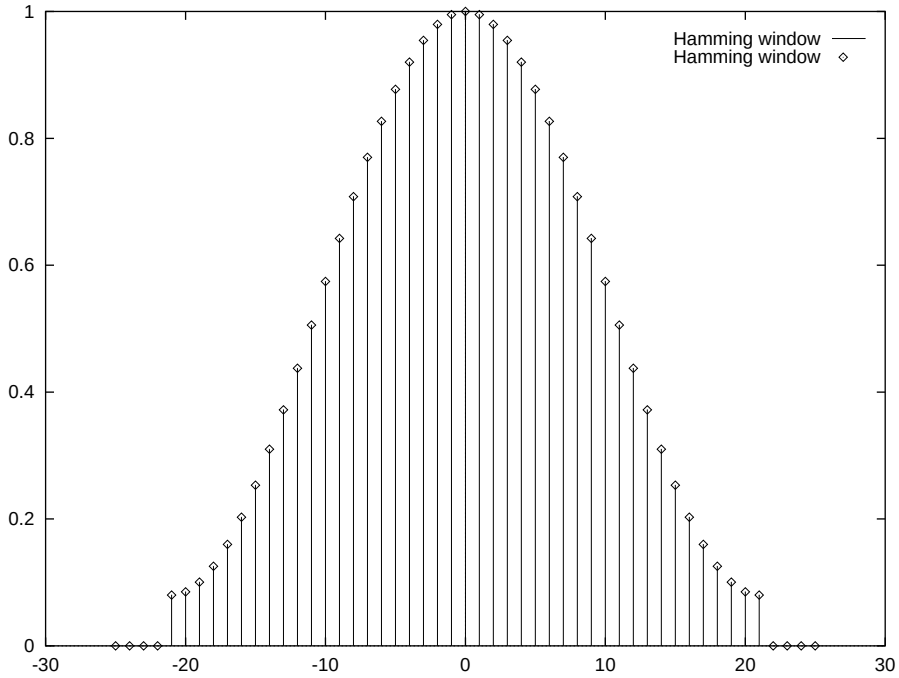

Figure 3.12: A 51-term *Hamming* windowing function

it consists of one period of a sampled cosine, and a constant level which makes all the sample values positive. Apparently the cosine shape gives a smoother tapering action than the rectangular window. Figure 3.11 illustrates the *von Hann* windowing function.

Similarly, the definition for the *Hamming* window is:

$$w(n) = \begin{cases} 0.54 + 0.46 \cos \frac{n\pi}{M+1}, & -M \leq n \leq M \\ 0 & \text{elsewhere.} \end{cases} \tag{3.54}$$

Figure 3.12 illustrates the *Hamming* window.

The corresponding spectrum function of both the *von Hann* window and *Hamming* window is given by:

$$W(\Omega) = w(0) + 2 \sum_{k=1}^{M} w(k) cos(k\Omega) \tag{3.55}$$

Experiences [10] showed that the *Hamming* window performs better, thus it will be chosen in lieu of the rectangular window in the configuration of adaptation agility of adaptors. We will show an example of configuring non-recursive adaptors using the Fourier Transform approach in

section 4.4.

3.7 Configuring Adaptation Agility: The Chebyshev Approach

3.7.1 Overview

Configurations using the Fourier Transform approach will result in a non-recursive difference equation that may include a considerable number of terms in order to meet the required accuracy to approximate the desired frequency response. To reduce the complexity of the difference equations, thus reducing the complexity of the computation needed for adaptation, we could configure the difference equations to be recursive equations, as defined in equation (2.46).

One of the suitable approaches for configuration is to transform the available continuous-time transfer functions into the desired discrete-time difference equations. The theories related to the study of analog signals and analog control systems [22] have been evolved for years and are readily available for the purpose of discrete-time adaptation design. As a comparison, while in the discrete-time cases we used z transform, the *Laplace transform* plays a similar role in the continuous-time analog design. For example, an analog control system can always be described by a frequency-domain transfer function of the general form:

$$H(s) = \frac{K(s-z_1)(s-z_2)(s-z_3)\dots}{(s-p_1)(s-p_2)(s-p_3)\dots} \tag{3.56}$$

where s is the *Laplace variable* and K is a constant, or *gain*, factor. Apart from this factor, as in the case of difference equations with z transform and *z-plane*, the transfer function is also characterized by its poles $(p_1, p_2, p_3, \dots)$ and zeros $(z_1, z_2, z_3, \dots)$, which can be plotted in the complex *s-plane*.

Although the form of equation (3.56) is identical to that describing the transfer function *H(z)* of an transformation that we addressed in section 3.4.3 and equation (3.28), the variable s has different connotations from the variable z. While the frequency response of a discrete-time transfer function as in equation (3.28) can be obtained by substituting z with $e^{j\Omega}$, the equivalent substitution in the analog case is $s \rightarrow j\omega$, where ω is the angular frequency in radians per second. It follows that the imaginary axis in the s-plane ($s = j\omega$) corresponds to the unit circle $|z| = 1$ in the z-plane, and that the interpretation of pole/zero locations is different in the two cases. Another

essential difference is that the frequency response of an analog control system is not a periodic function, any conversion from an continuous-time design into an discrete-time adaptation design must clearly take these factors into account.

To summarize, we need to be able to convert a transfer function $H(s)$ into a transfer function $H(z)$, so that the frequency response of the transformation made by an adaptor over the range of $0 < \Omega < \pi$ approximates, in an acceptable manner, that of the frequency response of an analog transfer function over the range of $0 < \omega < \pi$. Our approach is to use an effective way referred to as the *bilinear transformation*, and by the application of bilinear transformation, which will be defined in equation (3.60), on the widely used *Chebyshev* analog design, we could achieve acceptable configurations of an recursive difference equation.

3.7.2 the Chebyshev Approach

Our approach is to convert an analog design into a discrete-time equivalent configuration by means of *bilinear transformation*, defined in equation (3.60). In this section we apply the method to the best know *Chebyshev* design, resulting in an acceptable configuration of recursive difference equations.

The *Chebyshev* design offers an approximation of the ideal rectangular response characteristic illustrated in figure 3.9. The ideal response has unity transmission in the passband, and zero transmission in the stopband, while a Chebyshev approximation gives an *equiripple* performance in the passband, and the response oscillates between 1.0 and $(1 + \epsilon^2)^{-\frac{1}{2}}$, where ϵ is a configurable ripple parameter. The number of passband ripples increases with the order of the approximation. The magnitude functions of the frequency response is given by:

$$|H(\omega)| = \frac{1}{\left\{1 + \epsilon^2 C_n^2 \left(\frac{\omega}{\omega_{off}}\right)\right\}^{\frac{1}{2}}} \tag{3.57}$$

As an illustration, a typical magnitude frequency response for the Chebyshev analog approximation is given in figure 3.13. This graph is generated by *Mathematica*, with $\epsilon = 0.3$ and $\omega_{off} = 1.2$. The generating function is obtained from equation (3.57):

```
Chebyshev = Plot[(1/Sqrt[1 + 0.3^2 * ChebyshevT[5, x] * (x / 1.2)],
```

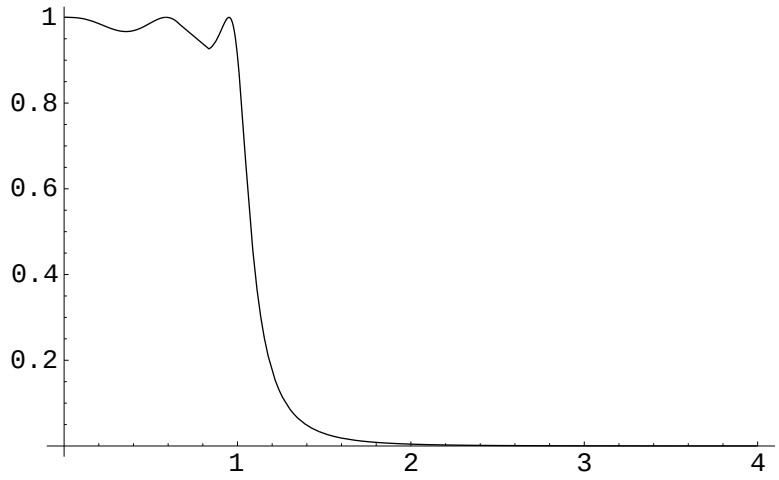

Figure 3.13: Typical frequency response of Chebyshev approximation

```
(x, 0, 3)]
```

where n is the order and ω_{off} is the cut-off frequency. C_n is referred to as the *Chebyshev polynomial* of nth order. It oscillates between 0 and 1 in the passband(for any value of $n > 0$), rising to large values in the stopband. The amount of passband ripple δ is related to the parameter ϵ by the expression:

$$\delta = 1 - (1 + \epsilon^2)^{-\frac{1}{2}} \tag{3.58}$$

The *Chebyshev polynomial* C_n is recursively given by:

$$C_0(x) = 1$$

$$C_1(x) = x$$

$$C_n(x) = 2xC_{n-1}(x) - C_{n-2}(x) \tag{3.59}$$

Although the magnitude characteristics of the Chebyshev approximation is good, its phase responses are less impressive. It departs considerably from the ideal linear-phase characteristic, especially towards the cut-off frequency. It also displays extra unwanted ripples. In this respect the Chebyshev approximation is inferior to the non-recursive Fourier Transform approach discussed in section 3.6.

We now introduce the bilinear transformation. Consider the function:

$$F(z) = \frac{z-1}{z+1} \tag{3.60}$$

It is *bilinear* in the sense that its numerator and denominator are both linear in z. To illustrate its application in the present context, we need to obtain its frequency spectrum. We have:

$$\begin{aligned} F(\Omega) &= \frac{e^{j\Omega}-1}{e^{j\Omega}+1} \\ &= \frac{e^{\frac{j\Omega}{2}}\left\{e^{\frac{j\Omega}{2}} - e^{\frac{-j\Omega}{2}}\right\}}{e^{\frac{j\Omega}{2}}\left\{e^{\frac{j\Omega}{2}} + e^{\frac{-j\Omega}{2}}\right\}} \\ &= \frac{2j\sin(\frac{\Omega}{2})}{2\cos(\frac{\Omega}{2})} \\ &= j\tan\left(\frac{\Omega}{2}\right) \end{aligned} \tag{3.61}$$

Evidently, $F(\Omega)$ is purely imaginary and periodic. Its magnitude varies between 0 and ∞ as Ω varies between 0 and π. If we have the transfer function of the desired analog approximation as equation (3.56), its frequency response can be determined by substituting $j\omega$ for s:

$$H(\omega) = H(s)\,|_{s=j\omega} = \frac{K(j\omega - z_1)(j\omega - z_2)(j\omega - z_3)\ldots}{(j\omega - p_1)(j\omega - p_2)(j\omega - p_3)\ldots} \tag{3.62}$$

The complete response is clearly generated as ω varies from 0 to ∞. If we substitute $F(\Omega) = j\tan(\frac{\Omega}{2})$ for $j\omega$, exactly same values must be produced as Ω varies between 0 and π. In other words we obtain a function $H(\Omega)$ in which the complete frequency response of the analog approximation is compressed into the range of $0 < \Omega < \pi$.

As a conclusion, the bilinear transformation gives a discrete-time transformation whose response over the range of $0 < \Omega < \pi$ reproduces that of the analog approximation over the range of $0 < \omega < \infty$. However, the resulting compression of the frequency scale is non-linear. The shape of the $\tan$ function decides that the compression, or *warping*, effect is very small near $\Omega = 0$; but it increases dramatically as we approach $\Omega = \pi$.

By applying the bilinear transformation, given the transfer function $H(s)$ of the Chebyshev approximation, we can readily generate the transfer function $H(z)$ of its discrete-time counter-

part. However, the transfer function of the Chebyshev approximation is relatively complex. Fortunately, the bilinear transformation has already been done [9]. We describe the result of the transformation by giving formulae for the z-plane pole-zero locations.

A Chebyshev approximation of nth order has n poles in the z-plane, and an nth-order real zero at $z = -1$. The corresponding formulae are:

$$re(p_m) = \frac{2\{1 - \alpha \tan(\frac{\Omega_{off}}{2}) \cos\phi\}}{\psi - 1}$$

$$im(p_m) = 2\beta \tan(\frac{\Omega_{off}}{2}) \sin\phi \tag{3.63}$$

where

$$\psi = \{1 - \alpha \tan(\frac{\Omega_{off}}{2}) \cos\phi\}^2 + \beta^2 \tan^2(\frac{\Omega_{off}}{2}) \sin^2\phi$$

$$\alpha = \frac{1}{2}(\gamma^{\frac{1}{n}} - \gamma^{-\frac{1}{n}})$$

$$\beta = \frac{1}{2}(\gamma^{\frac{1}{n}} + \gamma^{-\frac{1}{n}})$$

$$\gamma = (1 + \epsilon^{-1} + \epsilon^{-2})^{\frac{1}{2}}$$

$$\phi = \frac{m\pi}{n}$$

$$m = 0, 1, \ldots, 2n - 1$$

With the above formulae, once we have the order n of the Chebyshev approximation and the desired adaptation agility Ω_{off}, we can calculate the pole-zero positions of the transfer function in the z-plane, and thus have the final recursive difference equation as the result of the configuration. An comprehensive example will be demonstrated in section 4.4.

3.8 Conclusion

In this chapter, we introduced models derived from developed theories in digital control systems and digital signal processing systems [9] [10] to analyze and configure the adaptors modeled by non-recursive or recursive difference equations. We addressed the analysis of both the exci-

tation signals and the adaptive transformation itself in the time domain and frequency domain, and utilized the *Discrete Fourier Transform* and *z-transform* for analysis and configuration purposes. Based on the methods discussed, we were able to analyze the adaptor in terms of its *stability* of adaptive behavior, and to configure the adaptation according to a pre-determined *adaptation agility* or *sensitivity* in the application Quality of Service specification.

Chapter 4

A Prototype Adaptor for Video-On-Demand Applications

4.1 Overview

4.1.1 General Infrastructure

In Chapter 2, we proposed a simplified formal model for measurements of Quality of Service metrics as well as adaptation behavior parameters such as *adaptation agility* or *stability*. In Chapter 3, we addressed the problems related to the analysis and configuration of the adaptation behavior of a typical Quality of Service adaptor, utilizing developed theories in digital control systems. In order to verify the validity and performance issues of the proposed approach given by the previous chapters, we developed prototype adaptors based on the proposed methods, and analyzed the performance and adaptation behavior of these prototype adaptors in the context of Video-On-Demand applications, one of the widely known distributed multimedia applications in the area.

The infrastructure of the system including the Video-On-Demand applications and prototype adaptors is illustrated in figure 4.1. In the Video-On-Demand clients, we developed the middleware level between the multimedia applications and the underlying operating system and transport levels. In the current context, the middleware level consists of two components. One component is a Quality of Service *monitor* which monitors current status and changing trends of multiple specific Quality of Service metrics from the *raw Quality of Service* provided by underlying transport facilities. The other component is a Quality of Service *adaptor* which performs required adaptation on the Quality of Service metric that is being monitored. After performing the adaptation, the

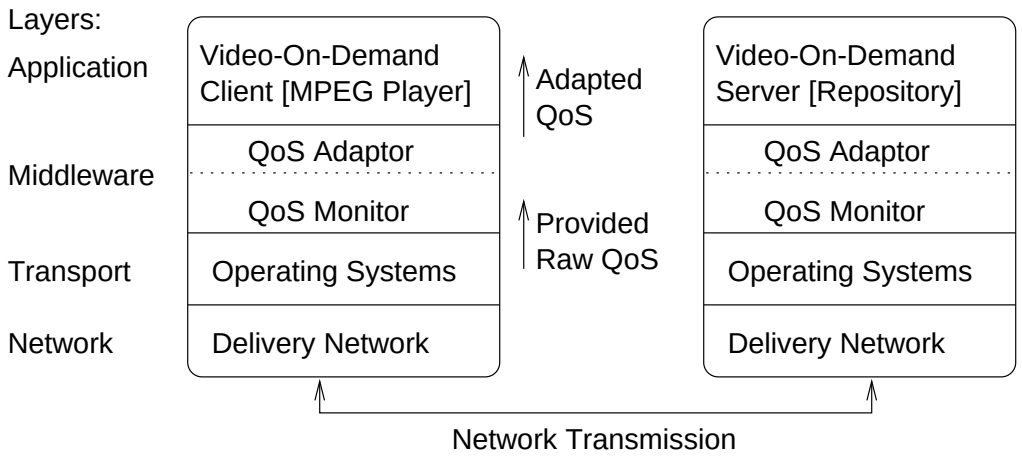

Figure 4.1: The infrastructure of Video-On-Demand application with adaptors

adaptor will deliver the *adapted Quality of Service* to the application layer.

The Video-On-Demand application is adopted as a testbed for our experimental prototype adaptor in the middleware level. It is designed to fit in the client-server model, and includes a Video-On-Demand server and a Video-On-Demand client. The server serves as a central video repository that satisfies the requests made from the possibly remote clients, and the client is designed as a simple video player that is capable of video playback based on the information retrieved from the server. This relationship between the client and the server is also illustrated in figure 4.1.

The performance of the system described above will be enhanced once the adaptors in the middleware level are introduced. There are occasions, especially in the heterogeneous mobile environment when the client is constantly on the move from one wireless base station to another, the bandwidth of the connection between the client and the server ports may change considerably over time. In these occasions, the application and the system need to coordinate to provide graceful adaptation to dynamic Quality of Service variations. This includes graceful degradation of provided Quality of Service to the application when the raw Quality of Service from the network level degrades significantly, and also includes pessimistic recovery speed when the connection quality recovers quickly. The adaptors described and formally modeled in the previous chapters will suit this purpose, after proper configuration with a suitable *adaptation agility*.

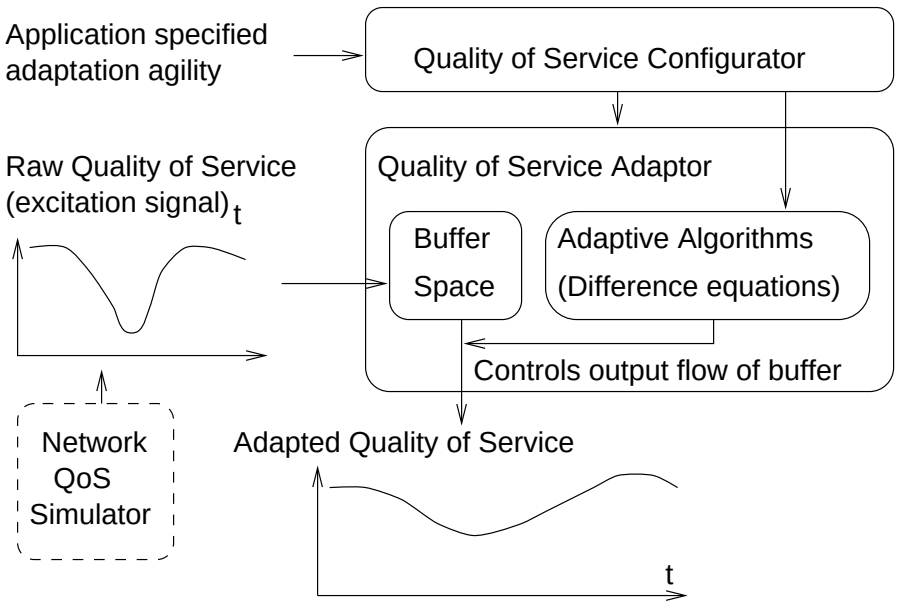

Figure 4.2: System design of the prototype Quality of Service adaptor

4.1.2 System Design

In order to achieve acceptable adaptation behavior of Quality of Service, we need to design the structure of various system components and their coordination and interconnections appropriately. We have discussed the general framework of the system briefly in chapter 3 and figure 3.1. Evidently, besides the actual Quality of Service adaptor that will be operating on line, we also need the Quality of Service configurator, which will generate adaptor configurations according to application needs. A more detailed design of the system is illustrated in figure 4.2.

For the purpose of fast prototyping and achieving capabilities to execute on heterogeneous environments, we used the Java language [6] for the implementation of the prototype adaptor. The Video-On-Demand application testbed, though, was implemented in a combination of C and Java languages, where the MPEG decompression engine was implemented in C due to the nature of MPEG-2 decompression complexity. The whole testbed was implemented in the Windows NT environment.

In order to simulate the fluctuations in the performance of the networking environment, we implemented a simulator that simulates the vibrations of the bandwidth factor over the connection. Due to the bandwidth fluctuations, the QoS metric *system rate* that can be measured by the monitors will also change accordingly. The initial experiments with adaptation behavior were

applied on these fluctuations with regards to the *system rate* of data arrivals.

The adaptors are implemented fully in the Java language as standalone applications. We implemented the configuration and analysis methods addressed in Chapter 3 in these adaptors, as well as all the related mathematical calculations and transformations, such as the *Discrete Fourier Transform* and its inverse form, the *Chebyshev* approximation, and the *Hamming* windowing functions. As a result, we were able to configure these adaptors to conform to prescribed adaptation agility, and to analyze the adaptation behavior of these adaptors. The adaptors are also responsible for delivering the Quality of Service metric, initially the *system rate* in our experimental prototype, to the application.

The monitors are implemented again in the Java language, and provide the functionalities of measuring a specific Quality of Service metric of the current instant according to the proposed model elaborated in Chapter 2. For instance, in our initial experiments *system rate* was selected as the Quality of Service metric being adapted, so we need to refer to equation (2.6), or alternatively and more realistically in terms of implementation, equation (2.8), to implement the measurements in the monitors:

$$rate_s^* : T \times T \times \Phi \mapsto R$$

$$rate_s^*(\alpha_i, \beta_i, \Pi_S) = \frac{quantity(\Pi_S[\alpha_i \leq a_k \leq \beta_i])}{\beta_i - \alpha_i} \tag{4.1}$$

Using this model it is relatively straightforward to implement the Quality of Service monitor, as well as to activate it periodically to monitor the raw Quality of Service provided from the transport facilities.

4.2 Designing the Prototype Adaptor

4.2.1 Quality of Service Monitors

Before designing the overall structure and implementation details of the prototype adaptors, we first need to prepare appropriate inputs that are required by the adaptor. One of the most important input is the sequence of Quality of Service metric values on which we need to apply

adaptation. This sequence of values on a pre-specified Quality of Service metric can be measured by the monitor. The monitor will interact with the underlying transport levels and will be fully aware of the raw Quality of Service provided by the transport facilities.

Based on the specific QoS metric that is intended to be measured and adapted, the detailed implementation of the measurement algorithms in the monitor may be significantly different. This makes it unlikely to construct a general-purpose monitor that is suitable for all monitoring tasks, individual monitors may need to be devised according to the characteristics of the specific QoS metric.

It it worth to mention that not all Quality of Service metrics can be adapted. While we can easily monitor and adapt to some commonly known metrics such as *system rate* and *delay*, some of the QoS metrics are inherently impossible to be adapted based on our current adaptation definition and algorithms. These metrics include *error rate* and *compression ratio*, whose common characteristic is that they are all direct reflections of the inherent quality of network transmissions or media streams, and cannot be easily modulated without substantial revision of the transmission or media coding schemes. We thus divide the Quality of Service metrics into two subsets, the first subset includes all the metrics that can be adapted using the current adaptation scheme, the other subset includes other metrics that are harder to harness without more substantial renovation.

For the prototype adaptors and monitors, we focus on a specific metric *system rate* for our experiments. Based on these assumptions, the QoS monitors can be defined in the Java language implementation as below:

```
public abstract class MMonitor extends Object {
  protected DatagramSocket inputChannel;
  protected MSequence metricSequence;
  private int metricId;

  public MMonitor(DatagramSocket inputChannel, int metric);
  public setChannel(DatagramSocket channel);
  public MSequence getMetricSequence();
}
```

4.2.2 Quality of Service Adaptors

Once we have the monitors available, we can apply appropriate adaptations on the sequence of metric values that we obtained by the monitors. For the preliminary prototype, we were able to configure the adaptor according to a predefined adaptation agility or sensitivity, and a specified precision, or order, of the adaptation difference equation. Obviously, the higher the order of the difference equation, the more precision it can possibly achieve the ideal frequency response. However, when the order of the difference equation is too high, it can introduce excessive amount of calculation overhead and may not be desired at the middleware level. Moreover, The order of a recursive difference equation tends to be the number of pole/zero positions in the transfer function, and the order of a non-recursive difference equation is the number of terms that appear on the right side of the equation.

The adaptor should also be capable of analyzing the stability of the current configuration of adaptation behavior, according to the stability judging criteria that was addressed in section 3.5.2. The stability of the adaptation behavior is very important in order to ensure that the system will be stable even after adaptation is introduced in the middleware level.

The class definition shown below in the Java language specification shows the important interfaces and functionalities that an adaptor should provide:

```
public class MAdaptor extends Object {
  protected MSequence excitation;
  protected MSequence response;
  private boolean recursive;
  private float agility;
  private int order;
  private MDiffEquation adaptEquation;
  private MBuffer internalBuffer;

  public MAdaptor(float agility, boolean recursive);
  public final boolean isRecursive();
  public setInputSequence(excitation);
  public isStable();
  public MSequence responseSequence();
}
```

Evidently, some basic class definitions need to be defined before the definition of `MAdaptor`. `MSequence` denotes the sequence of values of a specific Quality of Service metric, and should be

readily produced by the monitors. `MDiffEquation` defines the difference equation and transfer function of the adaptation, which fully represent and describe the adaptation behavior of the adaptor. Method `isStable()` judges the stability of the current adaptation status, and method `responseSequence()` retrieves the result sequence of the transformation performed by the adaptation.

4.3 Adjusting Buffer Allocation

In the process of performing adaptations on the specified Quality of Service metrics, in order to maintain graceful degradation when the raw Quality of Service provided by the underlying levels degrades significantly over a short period of time, we need to maintain buffer spaces and associate them with the adaptor itself. Before graceful degradation can be performed according to the current configuration, we need to allow a considerable amount of preprocessing in order to activate the adaptor buffer with initial metric values. This is normally achieved by prefetching metric values from the sequence produced by the monitors to fill the adaptor buffers, before feeding the adapted Quality of Service to the application. The tradeoff being made in the preprocessing step is that end-to-end delay may be sacrificed if the requirement is so stringent that the prefetching time makes a significant difference. Due to the relatively fast processing time of the processor as compared to the transmission time of the transport facilities, the above mentioned scenario normally will not be the case in regular distributed multimedia systems.

However, another problem may arise during the initialization phase of the adaptors. The initialization need to allocate a predetermined space for the adaptor buffers, so that to minimize later attempts to reallocate buffers to acquire more space. We thus need an appropriate approximation of the buffer space, so that the approximate value will not be too large to extend the prefetching time, and that the value should not be too small so that enlargement requests for the buffer space are made too frequently.

It is impossible to determine the required buffer space according to inherent characteristics of the adaptor itself. This space requirement will also depend on the fluctuations in the excitation sequence, as it can be expressed as the integral of the difference between the response sequence and the excitation sequence over a specific period of time. If we have

$$\sum_{i=0}^{N} a_i y(n-i) = \sum_{i=0}^{M} b_i x(n-i) \tag{4.2}$$

as the difference equation of the adaptation behavior. The excitation signals are represented by $x(i)$, and the response signals are represented by $y(i)$. If we interpolate these discrete-time signals to generate continuous-time functions $x(t)$ and $y(t)$, we will have

$$s_b|_{t_1}^{t_2} = \int_{t_1}^{t_2} y(t) - x(t) \tag{4.3}$$

where s_b denotes the required buffer space from time t_1 to t_2.

Obviously, the required buffer space s_b cannot be determined without the knowledge of excitation signal $x(t)$. This makes it difficult to determine s_b at the initialization phase before performing the adaptations. In the prototype adaptors that we implemented, we choose the value of s_b according to the desired adaptation agility Ω_{off}, using

$$s_b = \frac{k}{\Omega_{off}} \tag{4.4}$$

where k is a pre-determined constant according to experiments or experiences of previous rounds of execution. The adaptor buffer is implemented as a regular ring buffer.

4.4 Experimental Results

4.4.1 Configuration of Non-Recursive Adaptors

In the initial prototype of our adaptors we experimented with the configuration of some non-recursive low-pass adaptors, using the Fourier Transform approach addressed in section 3.6. As an example, we present the experiment results in this section with typical configuration parameter sets, and apply the configured adaptor to excitation signals that the network QoS simulator generated.

Using the Fourier Transform configurator, we successfully configured a 101-term low-pass adaptor with an adaptation agility of 0.2π, which is defined as cut-off frequency Ω_{off} in 3.3.3. For better configuration results, we chose the *Hamming* windowing function defined in section

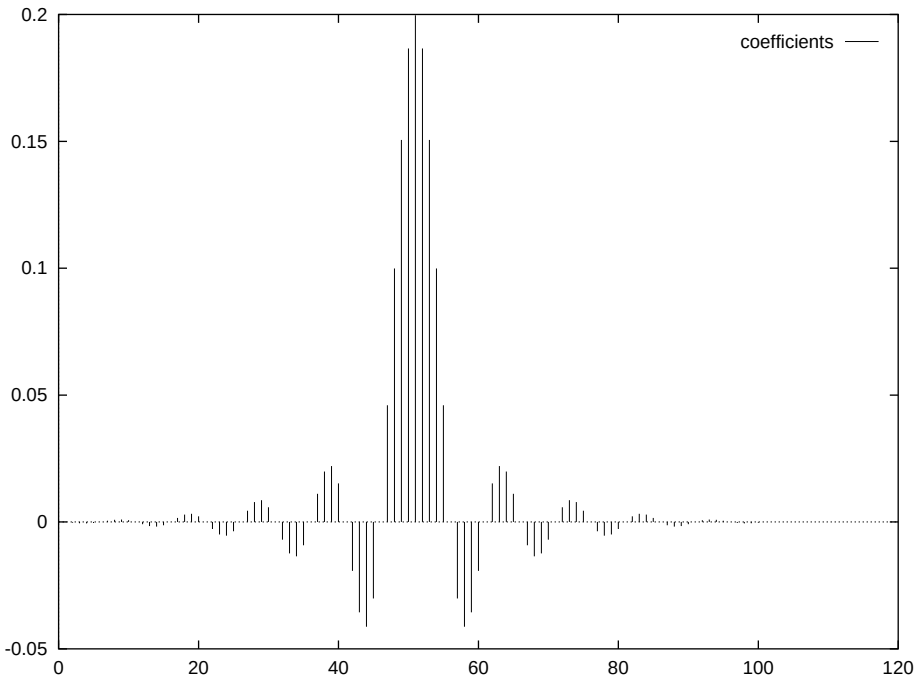

Figure 4.3: The coefficients in the non-recursive difference equation of the configured adaptor

3.6.2 to truncate the original result of Inverse Fourier Transform shown in equation (3.48) and equation (3.49). The approach discussed in section 3.6 was utilized in the configuration. As it is easily realized, the number of terms in the non-recursive adaptor determines the precision of the approximation of the actual frequency response to the ideal frequency response illustrated in figure 3.9. We chose to configure a 101-term low-pass adaptor to achieve an acceptable precision of approximation, while not sacrificing the adaptation overhead.

The result of a configured adaptor is expressed in the form of difference equations. Non-recursive difference equations were defined in equation (2.44), and we only need to present the coefficients a_i to fully represent the difference equation. Due to the volume of coefficients we need to present here, we chose to illustrate them in figure 4.3 [1].

For better visibility of passband ripples and sidelobes in the frequency response, we chose to plot the frequency response function to logarithmic scales rather than linear scales. A widely used logarithmic measure of spectral magnitude (or gain) is the *decibel*. If we have a function $H(\Omega)$ whose magnitude at some frequency is G, then the equivalent value in decibels (dB) is $20log_{10}G$.

[1]The x axis in figure 4.3 denotes *time* in the unit of *second*. The y axis denotes a real number as a coefficient in the difference equation.

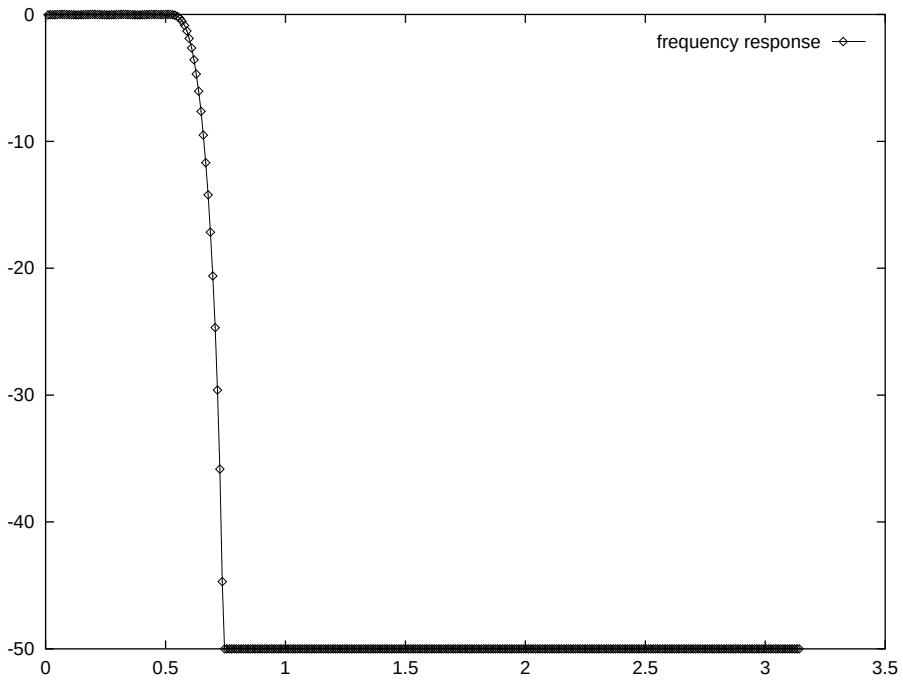

Figure 4.4: Spectral magnitude in the frequency response of configured non-recursive adaptor

Furthermore, it is often convenient to normalize the frequency response function to unity, namely in the range of $[0, 1]$, giving a logarithmic plot with a maximum of 0 dB in the passband. We illustrate the frequency response plotted to the logarithmic scale in figure 4.4 [2].

We now apply this adaptor to actual QoS metrics generated by the network QoS simulator illustrated in figure 4.2. The simulator simulates the actual performance of the network, utilizing traces obtained from the Unix ping command between the client and the server. Figure 4.5 [3] illustrates one sequence of the simulator output, where system rate values are illustrated in the unit of kilobits per second. Figure 4.6 illustrates the generated adaptation result of the configured adaptor.

As shown in figure 4.6, the results of the adaptation behavior are as we expected and encouraging. Using the jitter evaluation component that is included in the network QoS simulator and the adaptors, we can calculate the standard deviation, or *jitter*, of the excitation and response signals using equation (2.18), which is repeated in equation (4.5):

[2]The x axis in figure 4.4 denotes *frequency* in the unit of *radians*. The y axis denotes *spectral magnitude* in the unit of *decibels*.

[3]The x axis in figure 4.5 is *time* in the unit of *second*, the y axis is *system rate* in the unit of *Kbps*. Same applies to figure 4.6.

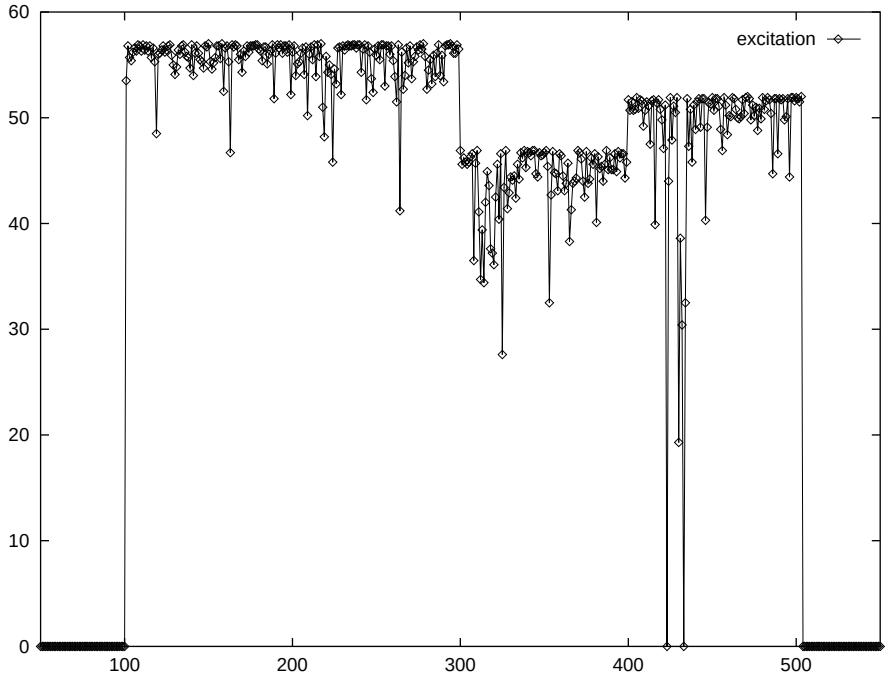

Figure 4.5: Excitation signal for the non-recursive adaptor: generated by the simulator

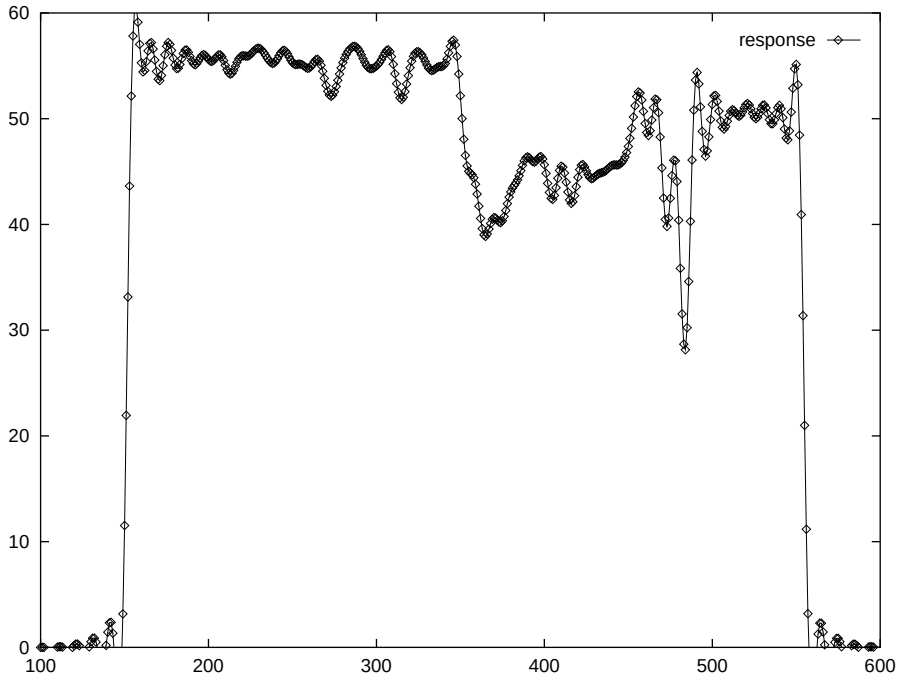

Figure 4.6: Response generated by the non-recursive adaptor

Evaluation metric	*Excitation*	*Response*
Jitter (Standard Deviation)	7.252160 Kbps	5.838508 Kbps

Table 4.1: An evaluation of the adaptation effectiveness for non-recursive adaptors

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n-1}} \tag{4.5}$$

where μ is the mean of all data, i is the index, n is the total number of data points, and x_i is a data point of index i.

The calculation results were shown in table 4.1.

4.4.2 Configuration of Recursive Adaptors

As an alternative, we also conducted experiments using our prototype adaptors with the configuration of recursive low-pass adaptors, using the Chebyshev approach addressed in section 3.7. As an another example, we present the experiment results in this section with typical configuration parameter sets for the recursive adaptors, and again, apply the configured adaptor to the excitation signals that the network QoS simulator generated. For the purpose of comparison, The trace we used was the same as the last section.

Using the Chebyshev configurator, we configured a 5th order low-pass adaptor with an adaptation agility of 0.2π, and a passband fractional ripple of 0.3. The approach discussed in section 3.7 was utilized in the configuration. Obviously, the order of the low-pass adaptor in the recursive adaptor, rather than the number of terms in the difference equation for non-recursive adaptors, determines the precision of the approximation of the actual frequency response to the ideal frequency response illustrated in figure 3.9. Similar to the configuration of non-recursive adaptors, We chose to configure a 5th order low-pass adaptor to achieve an acceptable precision of approximation, while not sacrificing the adaptation overhead.

The ultimate result of a configured adaptor should also be expressed in the form of difference equations. The recursive form of difference equations was defined in equation (2.46). However, the configuration process using the Chebyshev approach makes extensive use of pole/zero positions to specify the difference equations. As we will later demonstrate, the conversion from pole/zero position specifications to regular difference equations is convenient and straightforward.

By starting the Chebyshev configurator using our selected parameters, we found that a 5th order adaptor with a cut-off frequency of 0.2π has a real zero of order 5, at $z = -1$. The pole

radius r	*angle* θ
0.892786	0.000000 degrees
0.968629	34.840079 degrees
0.915411	21.986345 degrees

Table 4.2: Pole positions for the recursive adaptor

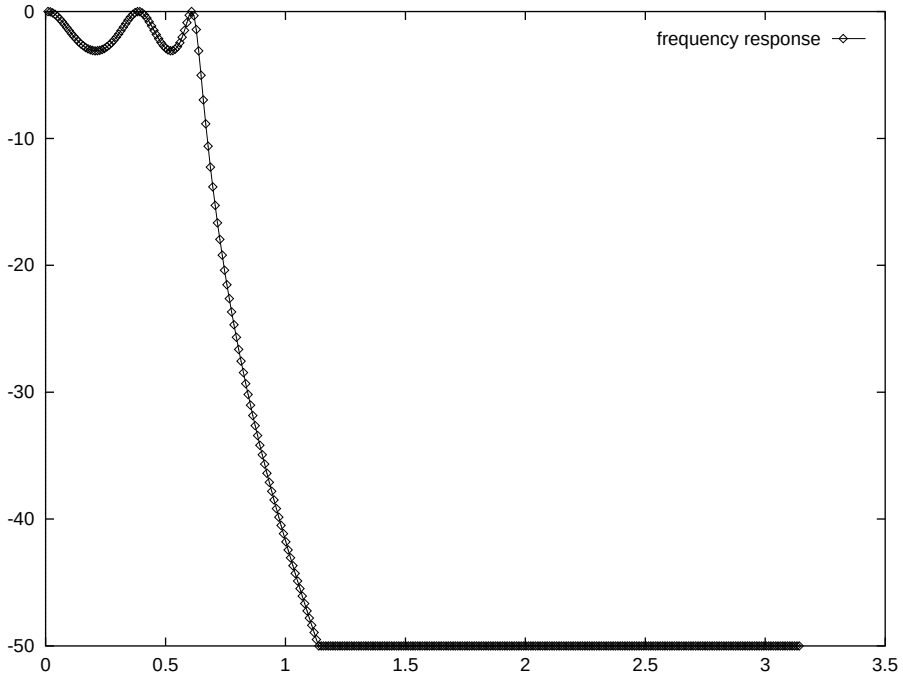

Figure 4.7: Spectral magnitude in the frequency response of configured recursive adaptor

locations, given in terms of radius r and angle θ, defined in equation (3.42), are shown in table 4.2.

Obviously, the adaptor contains one real pole, and two complex-conjugate pole-pairs. The configurator also calculates the maximum gain of the adaptor to be 6108.689453.

We then are able to illustrate the frequency response of the adaptor that we just configured. Similar to the last section, for better visibility of passband ripples and sidelobes, we still chose to plot the frequency response function to logarithmic scales rather than linear scales. We also normalized the frequency response function to the range of $[0, 1]$, giving a logarithmic plot with a maximum of 0 dB in the passband. We illustrated the frequency response plotted to the logarithmic scale in figure 4.7 [4].

Before applying the configured adaptor to the actual excitation signals, we first need to derive

[4]The x axis in figure 4.7 denotes *frequency* in the unit of *radians*. The y axis denotes *spectral magnitude* in the unit of *decibels*.

the difference equation of the adaptor from the specification of pole/zero locations. Obviously, knowing the poles and zeros, we could multiply the factors of $H(z)$ to obtain a numerator and denominator polynomial, and hence derive the difference equation of the adaptor. However this would involve a considerable amount of coefficient multiplication, which causes a problem that worsens as the adaptor order increases. A convenient alternative is to treat the overall adaptor as a *cascaded* set of first and second-order subadaptors, as shown in figure 4.8. The first-order subadaptor has a single real pole and zero; each second-order subadaptor comprises a complex pole-pair and a second-order zero. In this way the total complement of five poles and a 5th-order zero is built up. Note that the intermediate outputs are labeled $v(n)$ and $w(n)$.

The transfer function of the first-order subadaptor takes the form:

$$\frac{V(z)}{X(z)} = \frac{z+1}{z-\alpha} \tag{4.6}$$

giving the difference equation:

$$v(n) = \alpha v(n-1) + x(n) + x(n-1) \tag{4.7}$$

Each second-order subadaptor has a transfer function of the form:

$$\begin{aligned} \frac{W(z)}{V(z)} &= \frac{(z+1)^2}{[z-re^{j\theta}][z-re^{-j\theta}]} \\ &= \frac{z^2+2z+1}{z^2-2r\cos\theta z + r^2} \end{aligned} \tag{4.8}$$

yielding a difference equation:

$$w(n) = 2r\cos\theta w(n-1) - r^2 w(n-2) + v(n) + 2v(n-1) + v(n-2) \tag{4.9}$$

Inserting the pole values found in table 4.2, we obtain the following set of difference equations:

$$v(n) = 0.892786v(n-1) + x(n) + x(n-1)$$

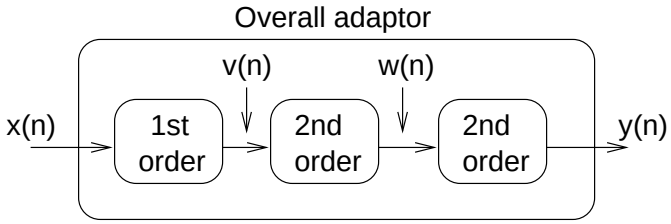

Figure 4.8: Cascading relationship among subadaptors

Evaluation metric	*Excitation*	*Response*
Jitter (Standard Deviation)	7.252160 Kbps	5.562410 Kbps

Table 4.3: An evaluation of the adaptation effectiveness for recursive adaptors

$$w(n) = 1.590004w(n-1) - 0.93824w(n-2) + v(n) + 2v(n-1) + v(n-2)$$

$$y(n) = 1.697672y(n-1) - 0.837977y(n-2) + w(n) + 2w(n-1) + w(n-2)$$

We now apply this adaptor to actual QoS metrics generated by the network QoS simulator illustrated in figure 4.2. Figure 4.9 [5] illustrates one sequence of the simulator output, where system rate values are illustrated in the unit of kilobits per second. Figure 4.10 illustrates the generated adaptation result of the configured adaptor.

As shown in figure 4.10, the results of the adaptation behavior for recursive adaptors are also positive. Using the jitter evaluation component that was included in the network QoS simulator and the adaptors, we calculated the standard deviation, or *jitter*, of the excitation and response signals using equation (2.18), the results are shown in table 4.3.

4.5 Conclusion

Preliminary experiments of the adaptation behaviors using our prototype system proved that the adaptations were effective in delivering adapted Quality of Service to the distributed multimedia application, in our case, the Video-On-Demand application. While sacrificing an insignificant amount of end-to-end delay time in order to prefetch QoS metric values into the adaptor buffer, the adaptors were able to deliver a much smoother and graceful degradation to the application

[5] The *x* axis in figure 4.9 is *time* in the unit of *second*, the *y* axis is *system rate* in the unit of *Kbps*. Same applies to figure 4.10.

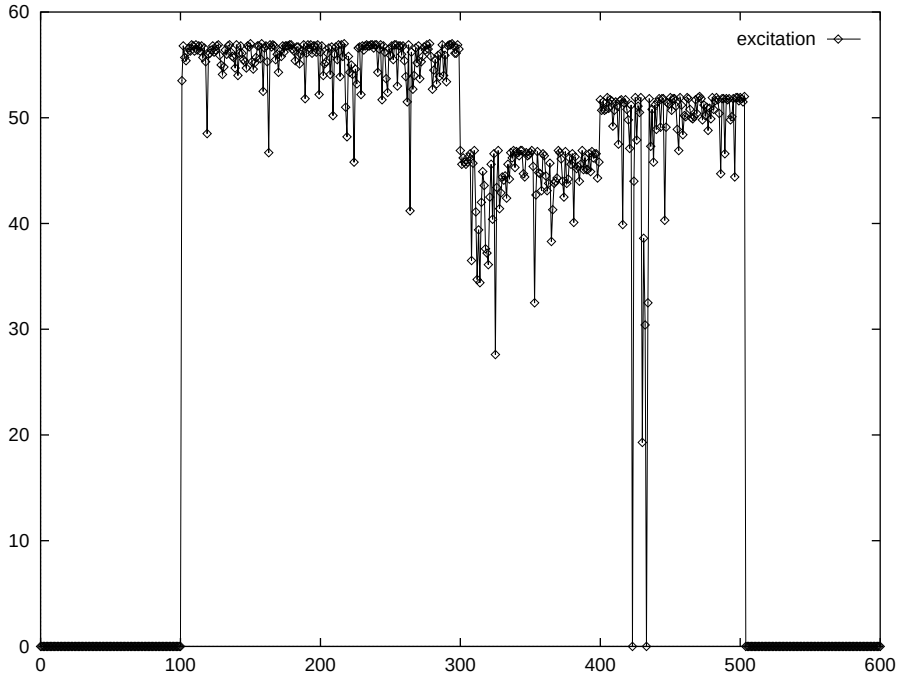

Figure 4.9: Excitation signal for the recursive adaptor: generated by the simulator

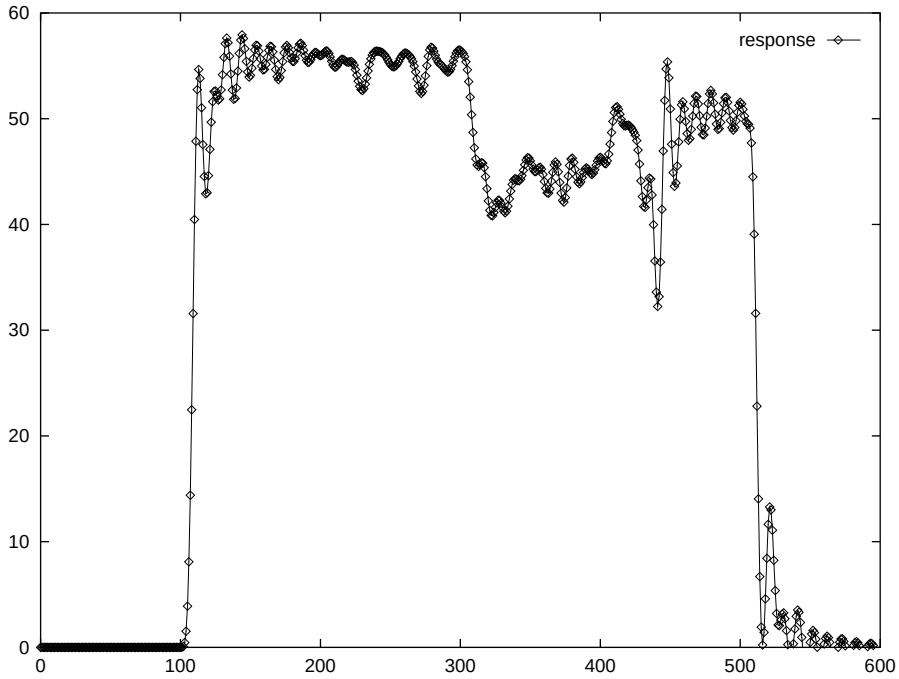

Figure 4.10: Response generated by the recursive adaptor

in the event that sudden and unexpected Quality of Service degradations occur significantly in a short period of time. This is demonstrated by the slowly effected frame rate that is being displayed at the Video-On-Demand client, while the simulator that simulates the bandwidth vibration produced significant jitter and burstiness over the connection between the server and the client.

Chapter 5

Conclusion

5.1 Conclusion

In order to provide stable and smooth Quality of Service to a wide range of distributed multimedia applications, and in the scenario that the applications do not demand guaranteed Quality of Service delivery, we are able to deliver *adaptive Quality of Service* to the application.

There are cases, especially in the heterogeneous wireless networks with mobile users constantly on the move, when it may be possible for the transport facilities to deliver severely fluctuated raw Quality of Service. The sudden and unexpected variations in the raw Quality of Service are obviously not desired by the application, and it can considerably affect the quality that the application can deliver to the end user. Our approach was to design and incorporate a specific component, referred to as *adaptors*, into the *middleware* level that is plugged in between the application and the transport layers. The adaptors, with the assistance of monitors, are responsible to monitor the raw Quality of Service provided by underlying levels, and perform adaptations to a specific Quality of Service metric, such as *system rate* of the flow, before delivering the adapted Quality of Service to the application. This is made possible by adopting existing theories in the area of digital control systems and digital signal processing.

We defined and analyzed various properties of the transformation that the adaptation may perform on the specific Quality of Service metric. An important inherent property among them is *adaptation stability*. If the adaptation behavior is not stable, the adaptor may yield an unbounded response that will keep fluctuating forever, on input of a reasonably bounded excitation signal. This is certainly not desired by the application. We also defined *adaptation agility* or *sensitivity*

which represents the ability and extent of an adaptor to promptly respond to perturbations in the raw Quality of Service from lower layers. We were able to model adaptation agility using the cut-off frequency of the desired frequency response, and to configure the adaptation behavior in the frequency domain with a desired adaptation agility requested by the Quality of Service specification.

Our preliminary experiments of a prototype adaptor showed promising results for Video-On-Demand applications using a client-server model. The results proved that the adaptations was effective in delivering adapted Quality of Service of a much smoother jitter level to the distributed multimedia applications. While sacrificing an insignificant amount of end-to-end delay time in order to prefetch QoS metric values into the adaptor buffer, the adaptation was able to deliver graceful degradation to the application in the event that sudden and unexpected Quality of Service degradations occur significantly in a very short period of time, typically in the environment of mobile wireless communications, when the mobile user is moving from one base station to the next.

To conclude, we are able to approach Quality of Service adaptations from a different perspective, and to precisely model Quality of Service metrics and the adaptation behavior using techniques from the digital control theory. Our initial results were promising with respect to the adaptation behavior on a specific Quality of Service metric, utilizing Video-On-Demand applications as our testbed platform.

5.2 Future Work

While the initial results of the adaptation behavior delivered by the middleware level are acceptable, adaptations, in a broader sense, can be renovated and designed to be a team of much more complicated tasks that is able to react to the various perturbations from the transport facilities and deliver adapted Quality of Service with a significantly improved performance of multiple QoS metrics.

As a part of future work, we are considering the possibilities to integrate the adaptors in the middleware level with other components in the same or lower levels, such as Quality of Service negotiation services, system resource allocation services, Quality of Service profile managers, as

well as Quality of Service translators between different levels and categories of Quality of Service metrics in the specification. By appropriate integration of these components in different levels with the adaptors, we will be able to deliver adaptations to those QoS metrics that are not possible to be adapted by middleware adaptors alone.

For example, the adaptors could coordinate with the system resource allocation components in the underlying layers to dynamically allocate available system resources according to the adaptation requirements. The adaptors could also coordinate with the Quality of Service negotiation component to renegotiate the various parameters in the transport facilities to better facilitate the adaptations performed. To conclude, we are actively seeking the theoretical and practical model of a generic framework that integrates all the components in the middleware level, so that this framework could deliver significantly improved Quality of Service to the applications, possibly with the assistance of mechanisms in the transport facilities.

In the process of performing adaptations on a specific Quality of Service metric, the preservation of various other Quality of Service metrics in possibly different levels or categories also proved to be a complex problem. For example, while adapting to the QoS metric *system rate* of a single media stream, it may be complicated to preserve acceptable QoS metric *synchronization skew* that represents the synchronization qualities among multiple media streams. With the integration of other components in the middleware level, we may need to devise a significantly complex theoretical model to deliver improved Quality of Service with regards to the overall quality that the application demands, rather than improving a portion of the quality while sacrificing the rest. A combination of local or global optimization methods may need to be employed at this stage of research.

Towards the objective of constructing an integrated framework for application level Quality of Service management and services, we are currently conducting research on various other components in the middleware level, such as the Quality of Service translation and mapping services between different categories of QoS metrics, the Quality of Service profiling services and its capabilities in assisting adaptations and translations, and Quality of Service trading and cost functions that facilitates the improvement of the overall Quality of Service that is delivered to the various distributed applications and end users.

References

[1] V. Bharghavan. Challenges and Solutions to Adaptive Computing and Seamless Mobility over Heterogeneous Wireless Networks. *to appear in the International Journal on Wireless Personal Communications: Special Issue on Mobile and Wireless Networking*, 1997.

[2] A. Campbell, G. Coulson, F. Garcia, D. Hutchison, and H. Leopold. Integrated Quality of Service for Multimedia Communications. *Proceedings of the IEEE INFOCOM '93*, 2, 1993.

[3] Z. Chen, S. Tan, R. Campbell, and Y. Li. Real Time Video and Audio in the World Wide Web. *Fourth International World Wide Web Conference*, 1995.

[4] J.Y. Chung, J. W.S. Liu, and K. J. Lin. Scheduling Real-Time, Periodic Jobs Using Imprecise Results. *IEEE Transactions on Computers*, September 1990.

[5] W. Feng and J. W.S. Liu. An Extended Imprecise Computation Model for Time-Constrained Speech Processing and Generation. *Proceedings of the IEEE Workshop on Real-Time Applications*, May 1993.

[6] D. Flanagan. *Java in a Nutshell*. O'Reilly & Associates, Inc., 1996.

[7] P. Florissi. Qual: Quality Assurance Language. *Technical Report CUCS-007-94, Department of Computer Science, Columbia University*, 1994.

[8] P. Goyal, H. Vin, C. Shen, and P. Shenoy. A Reliable, Adaptive Network Protocol for Video Transport. *IEEE INFOCOM '96*, 3, 1996.

[9] D. Manolakis J. Proakis. *Digital signal processing, principles, algorithms, and applications*. Prentice Hall, 1996.

[10] R. Jacquot. *Model Digital Control Systems*. Marcel Dekker, Inc., 2 edition, 1995.

[11] S. Jha and M. Fry. Continuous Media Playback and Jitter Control. *Proceedings of IEEE International Conference on Multimedia Computing and Systems (Multimedia '96)*, 1996.

[12] C. Liu and J. W.S. Liu. Effects of Imprecise Computation in Time-Invariant Control Systems. *Proceedings of the Twenty-Ninth Annual Conference on Information Sciences and Systems*, March 1995.

[13] J. W.S. Liu, K. Lin, R. Bettati, D. Hull, and A. Yu. Use of Imprecise Computation to Enhance Dependability of Real-Time Systems. In *Foundations of Dependable Computing: Paradigms for Dependable Applications*. Kluwer Academic Publishers, 1994.

[14] S. Lu and V. Bharghavan. Adaptive Resource Management Algorithms for Indoor Mobile Computing Environments. *ACM SIGCOMM '96*, 1996.

[15] S. Lu and V. Bharghavan. Adaptive Resource Reservation for Indoor Wireless LANs. *IEEE GLOBECOM '96*, 1996.

[16] J. M. McManus and K. W. Ross. Video on Demand over ATM: Constant-Rate Transmission and Transport. *IEEE Journal on Selected Areas in Communications*, June 1996.

[17] K. Nahrstedt and J. M. Smith. The QoS Broker. *IEEE Multimedia*, 1995.

[18] K. Nahrstedt and J. M. Smith. Design, Implementation, and Experiences of the OMEGA End-Point Architecture. *IEEE Journal on Selected Areas in Communications*, August 1996.

[19] P. Pancha and M. Zarki. Leaky Bucket Access Control for VBR MPEG Video. *IEEE INFOCOM '95*, 2, 1995.

[20] M. De Prycker. *Asynchronous Transfer Mode: Solution for Broadband ISDN*. Ellis Horwood Limited, 1993.

[21] M. Satyanarayanan, B. Noble, P. Kumar, and M. Price. Application-Aware Adaptation for Mobile Computing. *Operating Systems Review*, 29, 1995.

[22] S. Shinners. *Modern Control System Theory and Design*. John Wiley and Sons, Inc., 1992.

[23] N. Yeadon, F. Garcia, A. Campbell, and D. Hutchison. QoS Adaptation and Flow Filtering in ATM Networks. *Proceedings of the Second International Workshop on Multimedia: Advanced Teleservices and High Speed Communication Architectures*, 1994.

[24] L. Zhang, S. Deering, S. Shenker D. Estrin, and D. Zappala. RSVP: A New Resource ReSer-Vation Protocol. *IEEE Network Magazine*, September 1993.