An Open Task Control Model for Quality of Service Adaptation

Baochun Li, Klara Nahrstedt[*]
Department of Computer Science
University of Illinois at Urbana-Champaign
b-li@cs.uiuc.edu, klara@cs.uiuc.edu

Abstract

Current distributed multimedia applications demand Quality of Service (QoS) from the supporting system to facilitate effective services to the end users. However, within the range of QoS demands specified by the application, lower level transport facility may not be able to constantly provide the required QoS without perturbations, especially in the case of wireless communications. In this scenario, we propose a task control model that leverages existing digital processing and control theories to introduce *adaptation tasks* that perform QoS adaptations on a specific QoS metric. We are also able to configure these adaptation tasks according to a desired *adaptation agility* of their adaptation behavior. We show the viability of the approach by some preliminary experiments.

1 Introduction

Emerging state-of-the-art distributed applications pose increasing requirements for the underlying system to provide high availability, predictability, reliability and timeliness. This technology push justifies the need to study more dynamic real-time multimedia systems in which system components are heterogeneous and highly distributed, or even constantly on the move, in the case of wireless communications.

Due to the fact that the relative sensitivity to Quality of Service (QoS) 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, such as the *Internet*, to provide such guarantees. Even with the assistance of currently available techniques of providing QoS guarantees along the transmission path, such as suitable scheduling techniques in intermediate switches, the performance behavior along the transmission path in these networks cannot be easily guaranteed to be stable, especially if wireless communication links are involved in the path. This observation calls for the need of proper adaptation mechanisms in the end systems, so that applications adapt to the dynamics of the underlying environment. This is most suitable for the type of flexible applications that can tolerate a certain degree of variations in the provided QoS. Our objective is to develop a QoS architecture that lies in end-to-end system middleware layers and supports graceful adaptation in heterogeneous and distributed computing environments. We also note that the adaptation not only handles dynamic changes in resource availability along the transmission path, but also applies to dynamic modifications in user requirements, which may be specified interactively.

The traditional approach was that the adaptation behavior is integrated within the applications. This approach does not need to radically modify the existing protocols already implemented and running in current networks, so that the QoS delivery could be implemented with least modifications. However, there are also some disadvantages to this approach. First of all, since adaptation capabilities are within individual multimedia applications, different applications running on the same system may have very different adaptive behavior when QoS variations occur. Some of them may consume a considerable amount of system resources to perform their desired adaptation behavior, while others may not perform any adaptations at all. If system resources are limited, some applications may not be able to perform their

[*]This research was supported by the Air Force Grant under contract number F30602-97-2-0121, and National Science Foundation Career Grant under contract number NSF CCR 96-23867.

desired adaptations due to insufficient resources. It is therefore desired that there should be a central allocation facility to control the adaptation behavior of each application, as well as arbitrate and balance the resources required during the adaptation. Optimally, this central management facility should be located in between the underlying transport protocols and the applications demanding QoS. Furthermore, the adaptation component integrated into the application is not generic and reusable, which makes it a burden on the application developer with regards to implementation. Finally, since the application can only blindly apply its predetermined adaptation policy to all incoming traffic without any knowledge of the underlying transport layer activities, the adaptation performed may not be optimal for the situation and may not be modified on the fly.

In this paper, we propose an approach to perform the adaptation behavior in the middleware level which is located and operated between the transport facilities and applications. The adaptation behavior is configured off-line by the application itself, so that only the desired degree of adaptation activities is performed. The middleware adaptation facilities will monitor QoS delivered by underlying transport protocols, perform the desired adaptation behavior, and deliver the adapted QoS to the specific application. Utilizing this framework, applications only need to specify the policy of adaptation at a high level, and are shielded from the mechanics of adaptation behavior itself.

A major advantage to this approach is that by creating a software component to solely control the adaptation behavior for the entire end system, we could avoid unbalanced or conflicting demands for system resources, which is inherently a problem if we integrate adaptation behavior into each application. Another advantage by centralizing adaptation controls is that we could optimize global resource management and allocations by allowing different applications to share the same resource pool, especially when only limited resources are available, or when future adaptation activities are predictable. Finally, the approach also enables the middleware level as a whole to interact with underlying transport protocols, so that the adaptation behavior could be optimized by on-the-fly measurements of QoS delivery, and be able to react according to the monitored perturbations, providing capabilities of *active adaptations*.

We will consider the middleware level as a set of adaptation tasks which will be modeled according to the task control model introduced later. These adaptation tasks interact with the applications and the underlying layers, and react to perturbations in the provided QoS, so that graceful degradation can be achieved in the case of severe and unexpected Quality of Service changes.

This paper is structured as follows. Section 2 discusses relevant related work in the area of end system Quality of Service management and adaptations. Section 3 presents a task model for modeling QoS control in end systems, and leverages existing theories in digital control systems to model the adaptation behavior. Section 4 discusses in greater details constructing and configuring passive adaptation tasks. In Section 5 we describe the design and implementation choices for a proof-of-concept prototype of the adaptation tasks, as well as evaluations for their performance. Section 6 concludes the paper.

2 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, while there also exist research efforts that address the problems related to adaptations or graceful degradations of Quality of Service.

Recent research interests in mobile computing are addressing the issues in QoS adaptations for mobile transmissions over heterogeneous wireless networks [1] [14]. These research efforts focus on graceful adaptations to dynamic QoS variations in a mobile environment. Current state-of-the-art mobile environments must deal with scarce and dynamically varying resources in the end systems or delivery paths. 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. The research and development of the *Prayer* mobile computing environment [9] [10], for example, proposes a framework for adaptation which provides applications with runtime support for QoS negotiation, monitoring and notification services. The work presented in this paper has similar objectives, but leverages existing theories in digital control systems so that the adaptation mechanism in the middleware level can be more generic and not limited to mobile environments.

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 [12], focuses on the elimination of end-to-end delay jitter for audio data transmission over the Internet. Various algorithms proposed in the work 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 audio quality at the destination end systems. The goal of this work is also similar with the work presented in this paper, though the former only focuses on delay jitter adaptations during audio transmissions, and our work proposes a generic approach for modeling adaptation behavior. Similar research work is presented in [7], 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 [11], 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. Our work is analogous in the sense that we also need to consider proper allocation and utilization of system resources consumed by the end system adaptations. However, our adaptation algorithm focuses on the elimination of high frequency perturbations using theories for control systems, while still preserves long term trends of input QoS changes. Though theories in control systems lead to more complex designs for the adaptation algorithms, it proves to be occupying an insignificant amount of computation.

Open research problems for smoothing bandwidth requirements along the transmission path in the context of Video-On-Demand applications have also been explored. The work at Ohio State University [2] [3] focuses on *bandwidth smoothing algorithms* operated on the server based on *a priori* knowledge of the frame sizes in the prerecorded compressed video. According to different bandwidth optimization needs, various algorithms are applied at the server, so that the server can reduce burstiness in the stream by prefetching video frames in advance of each burst. The work operates optimally if frame sizes for the entire video stream are known *a priori*, even though later results [13] extend the model to live video streams by applying the algorithms to previous hopping or sliding windows. Our work mainly focuses on generic adaptation solutions on the client side, and will not be limited to only Video-On-Demand applications and the delivery of Variable Bit Rate (VBR) video streams.

3 The Task Control Model: A Model for QoS Adaptation in End Systems

3.1 The Task Flow Model for QoS Systems

We propose to use a *Task Control Model* to model any system delivering QoS, including the transport subsystems and the end systems, in a generic framework. By *task*, we refer to an application, a software component, or a module in an application. that executes so it can deliver a service or result to other system components, namely, other tasks [5]. With the definition of *task* in this context, if we consider a set of inter-related tasks that form a larger component, we are able to model the entire system as a task flow graph, which is, by its nature, a directed acyclic graph. Each node in the graph represents a task, and each task has a certain set of inputs, denoted by incoming links to the task, and a set of outputs, denoted by outgoing links. In addition, an edge from node T_i to note T_k indicates that the task T_k uses the result produced by task T_i as its input. We believe that this task model is generic in nature and can be applied to any systems that involve QoS delivery. Figure 1 illustrates a generic framework of the task control model and its mapping to traditional view of the end systems delivering QoS.

For each task, there will be quality parameters and values associated with both input and output of the task. In addition, at any particular instant, the task will consume a specific amount of resources; and for a task with regular behavior, the more resources allocated to the task, the better the output quality.

As an example, we consider a distributed video tracking system that acquires live video from a remote site (inconvenient for human interaction), transmits the video stream to local site in real time, identifies at the local site image objects with a certain shape in the video stream, and tracks the movements of the objects based on shape identification. Figure 2 shows the task flow graph of the system. We notice that the surveillance video camera is the only source task. Its output quality is fixed and characterized by QoS parameters such as image resolution, size, luminance, color depth, and frame rate. The frame acquisition task is responsible for grabbing frames from the camera, and converting the signal into a digital format. The output of the frame acquisition task (i.e. the values of its QoS parameters) depends on the amount of available resources such as processing capabilities and memory allocated. The same applies for the Network Transmission task, where the available resources, such as bandwidth, significantly affects its output quality.

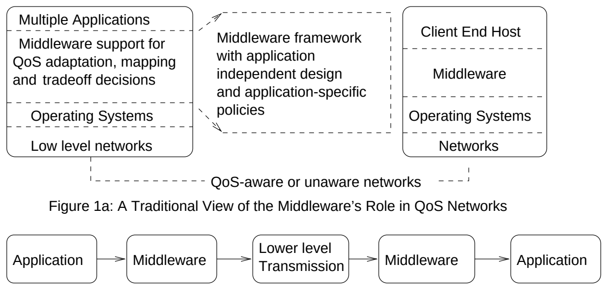

Multiple Applications
Middleware support for QoS adaptation, mapping and tradeoff decisions
Operating Systems
Low level networks
Middleware framework with application independent design and application-specific policies
Client End Host
Middleware
Operating Systems
Networks
QoS-aware or unaware networks
Figure 1a: A Traditional View of the Middleware's Role in QoS Networks
Application
Middleware
Lower level Transmission
Middleware
Application
Figure 1b: Viewing the Middleware's Role from a Task Flow Graph

Figure 1: A generic framework for the task control model

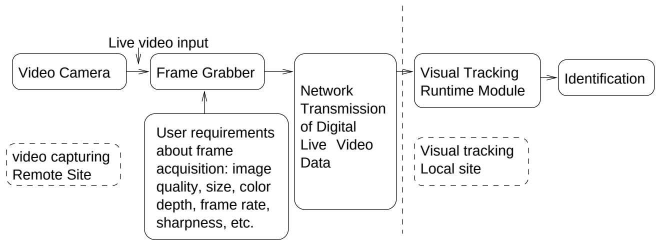

Live video input
Video Camera
Frame Grabber
Network Transmission of Digital Live Video Data
Visual Tracking Runtime Module
Identification
User requirements about frame acquisition: image quality, size, color depth, frame rate, sharpness, etc.
video capturing Remote Site
Visual tracking Local site

Figure 2: The visual tracking system: an example of the task control model

3.2 The Task Control Model for QoS Systems

The tasks introduced in the above model are in most cases *flexible tasks*, in the sense that if sufficient resources are not provided the tasks can still generate a specific output, even though the quality of the output is degraded due to lack of available resources. These flexible tasks justify the need to gracefully respond to unpredictable changes and oscillations in dynamic QoS requirements and resource availabilities for each task. This calls for the need of adaptation.

Under the task flow model introduced above, we propose to leverage existing theories in digital control systems to model the behavior of adaptation for the flexible tasks. Given the objective to improve the adaptability of individual tasks or the entire system, we can introduce specific *adaptation tasks* into the system, which integrates adaptation algorithms into additional tasks. Unlike ordinary tasks, these tasks are solely responsible for adaptation.

We can characterize adaptive behavior of these tasks into two major categories. One category, which we refer to as *passive adaptation tasks*, includes adaptive algorithms that only change certain QoS parameters of the input quality, so that the adapted quality will be improved by the adaptation. These passive adaptation tasks can be best illustrated as a simple additional task whose only responsibility is to provide a filter in between two consecutive original tasks in the system, and to improve the input quality of the next major task. Since the adaptation tasks have to contend with original tasks for limited resources, it is a tradeoff between improved adaptability and resource availability. The role of these adaptation tasks is illustrated in Figure 3.

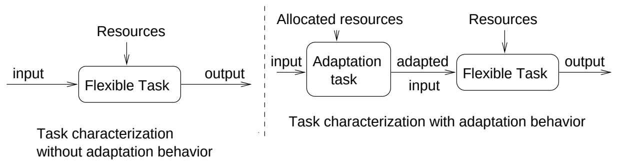


Figure 3: Applying passive adaptation tasks to tasks in need of adaptation

Examples of passive adaptation include various playout algorithms for video playback in the end systems. These playout algorithms only improve QoS of the incoming video streams in terms of specific QoS parameters such as delay jitter in frame arrival times, thus they actually serve as additional tasks utilizing preallocated resources such as buffer spaces to control the QoS parameters of the input quality to the main task.

Applying passive adaptation may prove to be sufficient in some general cases to adapt to changes in input quality. However in the cases of extremes such as long, persistent lack of resources or presence of extremely low input qualities, theoretically we need to provide an infinite amount of resources to the passive adaptation task in order to provide acceptable input quality for the main task. This calls for the need of the other category, which is referred to as *active adaptation*.

Active adaptation represents the class of various more aggressive adaptation attempts, in which the individual task that desires adaptation will coordinate with other flexible tasks in the task flow graph, in order to adapt to the changes that affect the entire system. Examples include the utilization of QoS tradeoff algorithms, feedback mechanisms, QoS renegotiation algorithms, and modification of resource allocation regulations in order to keep acceptable QoS for the individual tasks. In the example of video playback tasks, active adaptation tasks could coordinate with other related tasks in the system to change the compression ratio or various quantization parameters in the video streams, so that less resources are needed for the execution of the task.

Naturally, we can model the active adaptation problem in a task flow graph as a digital control problem. In this context, the individual task that adaptation is targeted can be treated as the plant to be controlled, and the active adaptation task can be treated as the controller. The mapping is illustrated in Figure 4. This mapping makes it possible to leverage existing theories in digital control systems to analyze the stability and performance of the adaptation tasks and the entire system.

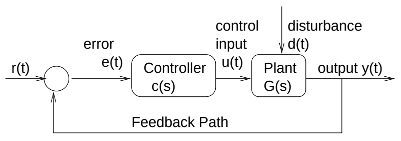


Figure 4a: A block diagram in a classic control system

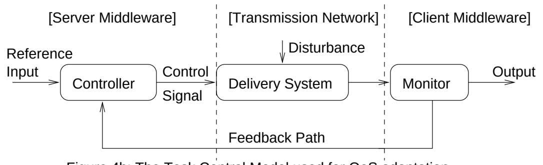


Figure 4b: The Task Control Model used for QoS adaptation

Figure 4: A mapping between block diagrams in control systems and the task control model

By introducing categories of adaptation, it is also important to solve the problem that in what circumstances we need to activate passive adaptation and in what else we need active adaptation. Apparently, if QoS delivery is stable, we do not need any adaptation at all. On the other extreme, if QoS delivery is in a severe situation we need to activate multiple active adaptation techniques concurrently. In situations in between the two extremes, we need to decide between passive and active adaptation and choose between different active adaptation schemes. This calls for determining and dividing QoS into different *QoS regions*, where in each region one specific adaptation scheme is applied.

A major advantage of the task control model is that it is an open model suitable of modeling a wide variety of QoS systems, and not limited to a specific type. This is because that any systems consist of functional components, and the task of each component is to operate on some input, and to generate some output, while consuming a specific amount of available resources. The task control model itself generalizes this observation, and provides a context for the application of existing control theories.

This paper focuses on the modeling of passive adaptation tasks. For this purpose, we quantify various QoS parameters of input quality, and model them as digital signals which are able to fully express the variations of QoS parameters in the time domain. The advantage of modeling QoS metrics into digital signals that vary in the time domain is that we can thus leverage existing theories in digital control systems to precisely model the passive adaptation task as a digital filter to the main task. We could therefore analyze and configure the adaptation task in the frequency domain, which proves to be more expressive of properties such as *adaptation agility*. Work for active adaptation will be presented in other papers.

4 Modeling Passive Adaptation Tasks

4.1 A Simplified Model for Passive Adaptation

As stated in the previous section, in order to provide the functionality of passive adaptation, our solution is to introduce components referred to as *Passive Adaptation Tasks*. These components reside in the end system middleware level, and serve as an intervening medium between the application that receives QoS and the underlying transport mechanisms that provides QoS. In order to accurately model the behavior of the passive adaptation tasks, we need to be able to quantitatively measure various QoS metrics that adaptation will be based on. Furthermore, in order to precisely model the configurable properties of the adaptation tasks, we need to formalize the concept of *adaptation agility*. We address these issues in this section.

4.1.1 Modeling Quality of Service Metrics

Before we address the issue of modeling passive adaptation and configurable parameters, we first need to present a simplified model for *Quality of Service metrics*. The objective of the model is to measure various QoS metrics in a quantitative way, so that each metric can be monitored in real time by an assisting task referred to as *monitor tasks*.

A *Quality of Service metric*, in its restrictive sense, measures the delivery performance of a continuous stream of *data units*. In its broader sense, Quality of Service metrics can denote any kind of quality parameters in any measurement units, for instance consistency, completeness or accuracy in a typical query to a digital library. We will only address the restrictive sense of QoS metrics in this paper. However, we believe that our conclusions drawn from examinations on the restrictive sense of QoS metrics are generic and may also apply to any broader contents of Quality of Service.

We represent a *stream* as an ordered sequence of $S = \{d_k\}_{k \in I}$, where d_k denotes a distinct *data unit*, which may in reality be a *frame* of video data, a *packet* of data transferred over packet switching networks, or a protocol independent *message* carried by any transport facility. I is the set of data unit indices, while a stream S is an ordered sequence of data units. 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 d_k, we assign a distinct *Quality of Service Signature* π_k, where $k \in I$:

$$\pi_k = \langle s_k, r_k \rangle (s_k < r_k,\ s_k \in T,\ r_k \in T) \tag{1}$$

where s_k and r_k represent the sending time at the original source of transmission and receiving time at the destination of the data unit d_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 $\Pi_S = \{\pi_k\}_{d_k \in S}$ 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 data unit in the stream S, while the expression $s_{last(\Pi_S)}$ denotes the sending time of the last data unit 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 conditional equation. 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:

$$QoS : \Phi^n \mapsto R \tag{2}$$

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.

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

For *approximated QoS metrics*, we define a mapping function:

$$QoS^* : T \times T \times \Phi^n \mapsto R \tag{3}$$

where 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

Notation	*Definition*	*Example*
d_k	an data unit of index k	a data packet or message
I	the set of data unit indices	integers
S	a stream of data units: $S = \{d_k\}_{k \in I}$	a regular data stream
s_k	sending time of an data unit d_k	sending at time 1
r_k	receiving time of an data unit d_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: $\Pi_S = \{\pi_k\}_{d_k \in S}$	
$first(\Pi_S)$	the index of the first data unit in stream S	
$last(\Pi_S)$	the index of the last data unit 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 1: A simplified model for Quality of Service metrics

$$\alpha_i < \beta_i,\ \alpha_i \in T,\ \beta_i \in T \tag{4}$$

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

$$t \in [\alpha_i, \beta_i] \text{ if and only if } \alpha_i \leq t \leq \beta_i \tag{5}$$

As an example, in the one-dimensional category, the value of $QoS^*(\alpha_1, \beta_1, \Pi_S)$ is the result of 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. We will discuss these problems in later sections.

For easier references, we summarize the above notations and definitions in Table 1.

Given above, we are able to define the common Quality of Service metrics such as *system rate*, *generating rate*, *delay*, *loss* and *jitter* [8]. As an example, we give a definition for the QoS metric *System Rate*.

System rate is generally referred to as *bandwidth* or *throughput* in a network transmission context. It denotes a QoS metric that measures the delivery speed or efficiency of any QoS quantitative data units. Given the definitions for QoS metrics in the previous section, *system rate* measures the mean number of data units 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 data units 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 data units in *S* that are received within the time interval $[t_1, t_2]$. The formal definition of *system rate* is:

$$rate_s : \Phi \mapsto R$$
$$rate_s(\Pi_S) = \frac{quantity(\Pi_S[C_r])}{r_{last(\Pi_S)} - r_{first(\Pi_S)}} \tag{6}$$

where condition C_r can be defined as:

$$C_r = r_{first(\Pi_S)} \leq r_k \leq r_{last(\Pi_S)},\ r_k \in T \tag{7}$$

As stated previously, it is generally not convenient to measure *system rate* defined above. We may then define *approximated system rate* so that it can be measured practically. Given a time interval $[\alpha_i, \beta_i]$ defined in Equation (4) and (5), the definition of *approximated system rate* can be easily derived as follows:

$$rate_s^* : T \times T \times \Phi \mapsto R$$
$$rate_s^*(\alpha_i, \beta_i, \Pi_S) = \frac{quantity(\Pi_S[C_r^*])}{\beta_i - \alpha_i} \tag{8}$$

where condition C_r^* can be defined as:

$$C_r^* = \alpha_i \leq r_k \leq \beta_i \tag{9}$$

4.1.2 Modeling Passive Adaptation

The discussions proposed previously provide sufficient grounds for the elaboration of a model suitable for modeling the behavior of passive adaptation.

As illustrated in Figure 5, monitor tasks measure the raw QoS metrics of an input stream of data units, while passive adaptation tasks perform the actual adaptation and produce a series of adapted values in a specific QoS metric, which can then be converted reversely and fed into the main task.

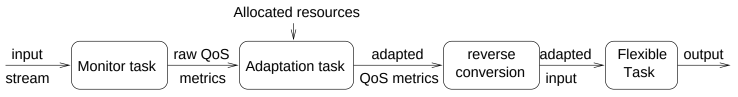


Figure 5: The role of a passive adaptation task

Generally, a *passive adaptation task* is an *operator* or *filter* on a particular stream that controls and modifies its QoS profile. Formally, it 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 task that delays the receiving time of all data units in a stream by a constant *const*. This task can be understood as a *delaying task*, whose only transformation applied on the QoS profile is to delay the receiving time by a specified amount.

To the extent of this paper, we will only address a specific subset of the passive adaptation tasks defined above, namely, those tasks whose behavior only transforms a specific QoS metric, such as system rate, of a particular QoS profile. The result of the transformation is a series of values based on the measuring unit of this metric. By utilizing the definitions 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, based on the definitions, we can also develop mechanisms to reversely convert the known values of QoS metrics to QoS signatures, in which the known values of QoS metrics are the result of the transformation made by the adaptation task.

Since the adaptation tasks are implemented in the end systems, after any transformation on the QoS metrics, they are only able to modify the receiving time of each QoS signature, leaving the sending time unaffected. The behavior of these tasks is thus defined 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*. Formally:

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

$$\Pi'_S = \varphi(\Pi_S) = \{\langle s_k, r'_k\rangle\},\ for\ \forall\langle s_k, r_k\rangle \in \Pi_S \tag{10}$$

Taking the metric *approximated system rate* as an example, if we apply Equation (8) to an ordered sequence of *time intervals* α_i and β_i, a series of discrete-time ordered sequence of values $rate^*_s[i]$ can be computed based on the QoS profile Π_S. For the most frequently used case of consequent time intervals where $\beta_i = \alpha_{i+1}$, we can take a sequence of $rate^*_s[i]$ and convert it to a QoS profile Π_S whose system rate is identical or approximated by the sequence $rate^*_s[i]$. Formally:

$$quantity(\Pi_S[\alpha_1 \le r_k \le \alpha_i]) = \sum_{k=1}^{i}(\beta_k - \alpha_k)rate^*_s[k](\alpha_k, \beta_k, \Pi_S) \tag{11}$$

In any Quality of Service signature $\langle s_k, r'_k\rangle$ in the Quality of Service Profile $\Pi'_S[\alpha_1 \le r_k \le \beta_n]$ defined in Equation (10), we have:

$$r'_1 = r_1$$

$$r'_k = r'_{k-1} + \frac{\beta_i - \alpha_i}{quantity(\Pi'_S[\alpha_i \le r'_k \le \beta_i])} = r'_{k-1} + \frac{1}{rate^*_s[i](\alpha_i, \beta_i, \Pi'_S)},\ if\ \alpha_i \le r'_k \le \beta_i \tag{12}$$

We now have sufficient grounds to apply theories in digital control systems to model the transformation process. 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 sampling interval and *n* is an integer in the range (n_1, n_2) such that $-\infty \leq n_1$ and $n_2 \leq \infty$.

The transformation process defined above is illustrated by the block diagram in Figure 6. Input $x(nT)$ and output $y(nT)$ are generally referred to as the *excitation* and *response* signals 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 (12).

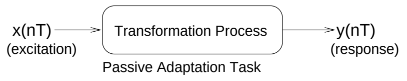


Figure 6: Applying adaptation process to the raw QoS metrics

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

$$y(nT) = \Theta x(nT) \tag{13}$$

where Θ is an operator. Equation (13) can be treated as a generic definition for passive adaptation tasks.

Like other digital systems, passive adaptation tasks can be classified with respect to *time-invariance*, *causality*, *linearity* and *recursiveness* [15]. In this paper, we only address the analysis and configuration of a subset of tasks that are time-invariant, causal, linear, and non-recursive. We characterize this subset with *difference equations* [6]. Assuming the above, we can express Equation (13) as a Nth-order linear difference equation as Equation (14), where $x(nT)$ is represented alternatively as $x(n)$:

$$y(n) = \sum_{i=0}^{N} a_i x(n-i) \tag{14}$$

4.2 Configurable Passive Adaptation

4.2.1 Modeling Adaptation Agility

Passive adaptations, as formally presented in the previous section, are in essence some form of transformations, so that the quality of a specific metric may improve. The series of QoS metric values are modeled by discrete-time values, or *signals*, and the transformation can be modeled by difference equations as in Equation (14).

One of the important factors that should be configurable for adaptation tasks is *adaptation agility*. First introduced in [14], it is defined as the ability of adaptation tasks to promptly respond to sudden and unexpected perturbations in the time domain. If the agility of an adaptation task is high, it means that the task frequently and promptly reacts to short term fluctuations. On the other hand, a system shows low agility if it only reacts to long term moving trends in the changes of a specific metric, and eliminates most of the short term fluctuations. Obviously, systems of low agility are often desired for most applications, as short term fluctuations naturally need to be eliminated. However, a system that has very low agility may consume too much temporary resources such as buffer spaces in order to transparently adapt to most of the fluctuations in the excitation signals and keep the response undisturbed, the amount of temporary resources may not be readily available on the end system. This tradeoff decision should be left to the application as an important part of the QoS specification. An illustration for the concept of adaptation agility is shown in Figure 7.

Utilizing theories in control systems, It is easy to quantitatively express the *frequency spectrum* of the excitation signal and the *frequency response* of the transformation by analyzing the adaptation process in the frequency domain, and by applying a *Discrete Fourier Transform* without energy loss of the signal.

Once we are able to model signals and transformations in the frequency domain, we can model the adaptation functionality that filters the rapid perturbations and high frequency short-term fluctuations from the long-term trends of changes in the excitation signal. This filtering capability can be best expressed in the frequency domain as a low pass filter, 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 passive adaptation tasks.

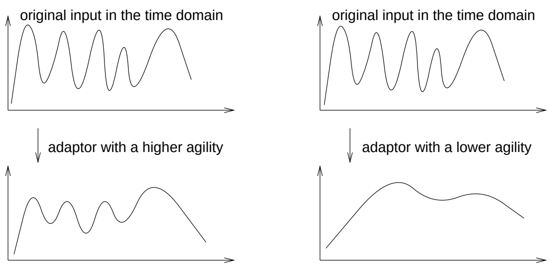


Figure 7: Adaptation agility for configurable passive adaptation tasks

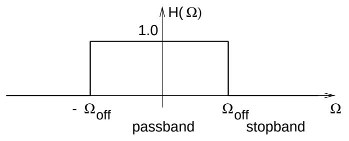


Figure 8: An ideal low-pass passive adaptation task with cut-off frequency Ω_{off}

Given above, the adaptation agility of an passive adaptation task can be defined as the cut-off frequency Ω_{off} of the frequency response of the transformation, illustrated in Figure 8. Defined as such, we can therefore configure the adaptation task to provide a specific agility specified by the application.

4.2.2 Configuring Adaptation Agility

Once the theories in digital control systems are applied, there are numerous ways to configure and design the passive adaptation tasks so that they conform to a certain agility requirement. One of the simplest approach that works on non-recursive difference equations is the application of Discrete Fourier Transform and its inverse transform, which may be found in some references on digital filters [6]. In order to improve the quality of the generated result after Inverse Fourier Transform, we may need to use more complex windowing functions such as the well known *von Hann* window or the *Hamming* window [6]. Since the configuration process operate in the frequency domain, we need to convert the result back to time domain in the form of difference equations. After configuring the passive adaptation tasks, they can be activated to perform the specified adaptation agility.

5 Experiments on Passive Adaptation Tasks

5.1 The Design of Experimental Framework

In order to explore the validity and performance of the proposed passive adaptation tasks, we developed a proof-of-concept prototype, and analyzed the adaptation performance of the prototype in the context of Video-On-Demand (VOD) applications. The framework is illustrated in Figure 9.

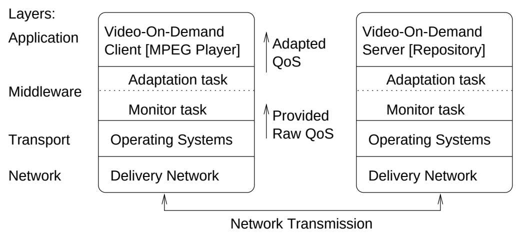


Figure 9: The proof-of-concept prototype framework

In VOD clients, we integrated the passive adaptation tasks and monitor tasks between the VOD applications and the underlying transport protocols. In the current context. The VOD 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. we also integrated a QoS configurator, which will configure the passive adaptation tasks according to a specific agility requirement.

For the purpose of fast prototyping and achieving capabilities to execute on heterogeneous environments, we used the Java language [4] for the implementation of the adaptation tasks. The Video-On-Demand application testbed, though, was implemented in a combination of C and Java languages, in which the MPEG-2 decompression engine was implemented in ANSI C due to the nature of MPEG-2 decompression complexity, and the display controls were implemented in Java. The whole testbed was implemented in the Windows NT environment, but can also be compiled to run in Unix environments thanks to the portability of the ANSI C codec source and the interpretive property of the Java language.

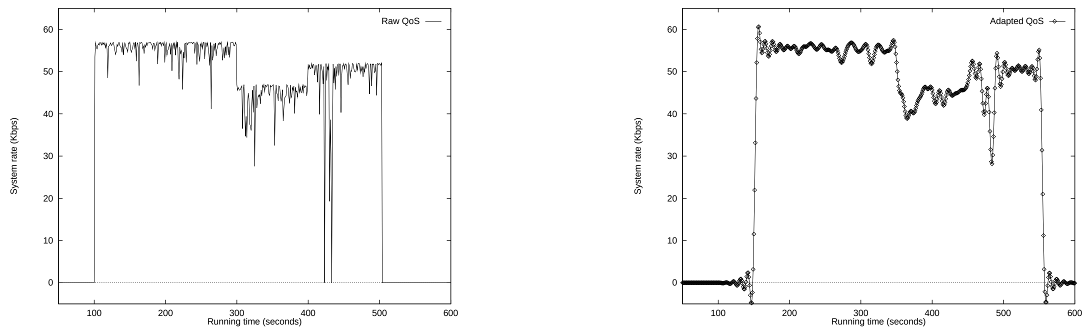
In order to simulate the fluctuations in the performance of the networking environment, we implemented a simulator that simulates the vibrations of the bandwidth factor over the connection. Due to the bandwidth fluctuations, the QoS metric *system rate* that can be measured by the monitor tasks will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

5.2 Experimental Results

We experimented with the configuration of some non-recursive low-pass adaptation tasks, using the Fourier Transform approach for configuration. As an example, we present the experiment results in this section with typical configuration parameter sets, and apply the configured adaptation task to excitation signals that the monitor task generated based on network performance generated by the network simulator.

Using the Fourier Transform configurator, we successfully configured a 101-term low-pass passive adaptation with an adaptation agility of 0.2π, which is defined as cut-off frequency Ω_{off} in 4.2.2. For better configuration results, we chose the Hamming windowing function to truncate the original result of Inverse Fourier Transform. The number of terms in the non-recursive adaptation task determines the precision that the actual frequency response approximates the ideal case illustrated in Figure 8. We chose to configure a 101-term low-pass adaptation task to achieve an acceptable precision of approximation, while not sacrificing the adaptation overhead. The result of a configured adaptation task is expressed in the form of difference equations.

We now apply the configured adaptation task to actual QoS metrics generated by the network QoS simulator. Figure 10(a) illustrates one sequence of the simulator output for a total length of 600 seconds, where system rate values are illustrated in the unit of kilobits per second. Figure 10(b) illustrates the generated adaptation result of the configured passive adaptation task.



(a) Raw QoS metric sequence generated by the monitor task

(b) Adapted QoS metric sequence generated by the passive adaptation task

Figure 10: Adaptation behavior of the 101-term non-recursive passive adaptation task

As shown in Figure 10(b), the results of the adaptation behavior are as we expected and encouraging. We can calculate the standard deviation of the excitation and response signals, shown in Table 2.

We are also interested in the buffer space requirements of the configured non-recursive adaptation task. Naturally, the actual requirement will not be predictable *a priori*, and will be decided by both the difference equation of the task

Evaluation metric	*Excitation*	*Response*
Standard Deviation	7.252160 Kbps	5.838508 Kbps

Table 2: An evaluation of the adaptation effectiveness for non-recursive passive adaptation tasks

and the raw QoS sequence monitored. In our preliminary experiment, the buffer space requirements as a function of time is illustrated in kilobytes in Figure 11. As shown in the figure, the peak buffer space requirement in our experimental context is around 350KB, which is affordable for most current systems.

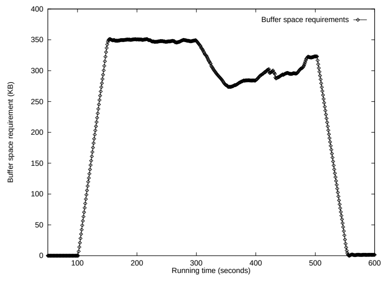


Figure 11: Buffer space requirements for the configured passive adaptation task

With regards to computational overhead for the configuration and adaptation processes, In spite of the slower speed of the interpreted Java program that we experimented with, we are able to record a typical computational overhead of 2 milliseconds for each round of adaptation, and a maximum overhead of no more than 3 milliseconds, tested on a light loaded Pentium 200Mhz processor. While adaptations are only needed intermittently and the time interval is not likely to be less than one second, the computational overhead for the adaptation process can be safely ignored. The configuration process is not necessary to meet real time needs, however it also occupies an insignificant computational overhead. As we recorded, a typical configuration process for our passive adaptation tasks takes 161 milliseconds to complete, which can also be safely ignored in most situations.

6 Conclusion

In order to provide stable and smooth Quality of Service to a wide range of distributed applications, and in the scenario that severe short term fluctuations that frequently occur in the QoS provided by the transport facility, especially in the heterogeneous wireless networks with mobile users constantly on the move, we propose to introduce an open task control model so that it is straightforward to map the existing theories in digital control systems, and to incorporate *adaptation tasks* into the task control model, which are responsible for performing adaptations on a specific QoS metric measured by monitor tasks.

One of the major contribution of the task control model is its openness and generic nature. Since the model is not specific to a particular system or platform, adaptation functionalities can be integrated into a wide variety of end systems, eliminating the need of designing a specific adaptation technique for each type of end systems. It is also possible to apply digital control theories to these systems, treating the task to be controlled as the *plant*, and the adaptation task as a controller.

We also believe that the adaptation tasks should be configurable. One of the configurable parameters is *adaptation agility*, which represents the ability and extent of an adaptation task to promptly respond to perturbations in the raw QoS 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 tasks in the frequency domain with a desired adaptation agility.

Preliminary experiments of the adaptation tasks using our prototype system proved that the adaptations were effective in delivering adapted QoS to the distributed applications, in our case, a Video-On-Demand application. While sacrificing an insignificant amount of end-to-end delay time in order to accommodate the phase shift between response and excitation signals, the adaptation tasks were able to deliver a smoother and graceful degradation to the application in the event that sudden and unexpected Quality of Service degradations occurred significantly in a short period of time. The adaptation was accomplished at an insignificant cost of computational overhead and an affordable cost of buffer spaces, especially when taking into consideration the advantage of possible sharing of a common pool of elastic buffers.

To conclude, we are able to approach QoS adaptations from a different perspective, and to precisely model QoS metrics and the adaptation tasks in the context of a task control model and digital control theory. Our initial results are promising with respect to the adaptation performance on a specific QoS metric, with affordable costs and minor overhead.

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] W. Feng. Rate-constrained bandwidth smoothing for the sdelivery of stored video. *IS&T-SPIE Multimedia Computing and Networking 1997*, 1997.

[3] W. Feng and J. Rexford. A Comparison of Bandwidth Smoothing Techniques for the Transmission of Prerecorded Compressed Video. *IEEE INFOCOM 97*, 1997.

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

[5] D. Hull, A. Shankar, K. Nahrstedt, and J. Liu. An end-to-end qos model and management architecture. *Proceedings of IEEE Workshop on Middleware for Distributed Real-time Systems and Services*, December 1997.

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

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

[8] B. Li. Adaptive Behavior of Quality of Service in Distributed Multimedia Systems. *Master's thesis, University of Illinois at Urbana-Champaign*, 1997.

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

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

[11] 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.

[12] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne. Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks. *IEEE INFOCOM '94*, 2, 1994.

[13] J. Rexford, S. Sen, J. Dey, W. Feng, J. Kurose, J. Stankovic, and D. Towsley. Online Smoothing of Live, Variable-Bit-Rate Video. *the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 1997.

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

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