End-to-End QoS Support for Adaptive Applications Over the Internet

Baochun Li, Dongyan Xu, Klara Nahrstedt, Jane W.-S. Liu *

Department of Computer Science
University of Illinois at Urbana-Champaign

ABSTRACT

In current end systems, multiple flexible, complex and distributed applications concurrently share and compete both end systems resources and transmission bandwidth of heterogeneous multi-protocol networks, especially the Internet. Our objective is to enable adaptation awareness in these applications to fully cope with the dynamics in resource availability over the heterogeneous Internet, as well as fluctuations in QoS requirements of the applications themselves. In this paper, we present the theoretical and practical aspects of a Task Control Model implemented in the middleware layer, which applies control theoretical approaches to utilize measurement-based samples monitored in the network traffic, as well as resource and QoS demand dynamics observed in the end systems.

In our Task Control Model, we introduce Adaptation Tasks for controlling the adaptive behavior of applications, and Observation Tasks for measurements of traffic and end systems resources. We have made several contributions based on the Task Control Model. First, we are able to quantitatively analyze the stability and responsiveness properties of adaptive algorithms, as well as maintaining fairness properties among all applications. Secondly, we present specific translation mechanisms to theoretical output of Adaptation Tasks to semantic adaptive behavior within the applications. Thirdly, we explore necessary enabling service platforms in the middleware level to embrace the Task Control Model. Finally, the theoretical analysis in the paper is carried out in the context of a distributed visual tracking application, in order to demonstrate the effects of adaptation in real systems.

Keywords: QoS Adaptation, Middleware

1. INTRODUCTION

In heterogeneous multi-protocol networks such as the Internet, there exist a scenario where a variety of distributed multimedia applications share and compete for resources, both in the end systems and transmission networks. These applications may demand Quality of Service (QoS), which is provided by the underlying supporting platform. Frequently desired QoS factors include bandwidth, timeliness and reliability (loss probability). However, applications may not be able to receive constant quality assurance from the system. There are two main reasons. First, there may be physical limitations in a distributed environment, for example transmission paths that include wireless links may be error-prone with high bandwidth variations. Second, dynamic activation and deactivation of multiple concurrent threads or applications may cause variations in QoS provided to a specific application, in an environment without proper reservation and assurance mechanisms. For example, in an end system without real-time prioritized scheduling and reservation of CPU resources, CPU-intensive applications may not be able to receive constant QoS with respect to its timeliness requirements in application execution.

It is therefore necessary for the application to be adaptive to the variations in its executing environment, both in the end system and on the transmission path. There are two categories of applications. First, if the applications have strict mission-critical real-time requirements, once the provided QoS is violated and no longer constant, the application may fail. For these real-time applications, QoS guarantees are important rather than adaptations. However, in the second category where distributed applications demand a specific QoS level but are *flexible*, adaptations play an important role when constant guarantees are either not possible, when physical limitations are present, or not cost-effective, when it is impossible to predict the maximum QoS requirements to be reserved for interactive applications. These *flexible* applications present the following properties. First, they can accept and tolerate resource scarcity to a certain minimum bound, and can improve its performance if given a larger share of resources. Second, they are willing to sacrifice the performance of some quality parameters in order to preserve the quality of critical parameters. In the context of variations in resource availability, it is thus desirable to trade off less critical quality parameters for preserving quality assurance for critical parameters.

Our objective is to support the second category of flexible applications with a middleware layer of software components. The support provided by the middleware layer include two parts. First, it enhances the adaptation awareness of flexible applications being supported. These flexible applications may not be designed to be aware or fully cope with the dynamics in resource availability over heterogeneous Internet, even though they are indeed flexible, and thus have potential to be adaptive to these dynamics. Second, it makes decisions to control the adaptive behavior of the applications. These decisions include when, how and to what extent adaptation is carried out in the application.

There are several important advantages of implementing a middleware layer for QoS adaptations. First, it provides global control of multiple concurrent applications, so that the applications may coordinate with each other with respect to adaptive behavior and conflicts in resource requirements can be solved. This is not possible if implementing the adaptation choices within the applications themselves. Second, the adaptation decisions can be made following guidance of theoretical algorithms with strictly proved stability, fairness and adaptation agility properties. Thirdly, with a centralized control within the end system, re-implementations of adaptive mechanisms of possibly conflicting manners in different applications can be avoided.

In order to adapt to external dynamics, the internal adjustments within an application are decided according to the observable parameters within the dynamics. This mechanism corresponds identically to a control system, where input is determined by a *controller* and based on the observed states of the control target. Our previous work[1] takes advantage of control theory to model adaptation behavior, based on its analogy to the control system. In the process of analyzing the analogy and developing suitable modeling techniques to exploit this analogy, the Task Control Model was developed and theoretical results were given to prove stability and fairness properties in the model. The goal of the control policies developed in the Task Control Model, however, is to adjust the internal semantics and behavior of distributed multimedia applications. While the control policies are generated by the middleware components in the context of a Task Control Model, the actual mechanisms (e.g., buffering, prefetching, parameter tuning or reconfigurations) are carried out within the applications themselves, and controlled by the middleware components. The separation of control policies and mechanisms makes it possible for a single control policy to control applications via different generic or application-specific mechanisms.

The major contribution of this paper is the following. (1) The Task Control Model introduced in our previous work[1] only investigated stability, fairness and adaptation agility properties of control algorithms, without further mentioning of the translation and mapping between theoretical values generated by control algorithms and the actual decisions made to control real applications on-the-fly. This issue is examined and completed in this paper. (2) We extend possible adaptation mechanisms from parameter and data-level scaling to functional reconfigurations, and we develop mechanisms to reconfigure the application while still utilizing the same control algorithms for quantitative parameter tuning. (3) We analyze theoretical results given by our previous work in the context of a distributed visual tracking application, and present experimental results using middleware components controlling the application.

In the context of the distributed visual tracking application, the rest of this paper is organized as follows. In Section 2, we discuss existing work related to adaptive QoS. In Section 3, we review our Task Control Model proposed in our previous work,[1] in the context of a distributed visual tracking application. In Section 4, we emphasize the translation mechanisms between theoretical output from control algorithms and actual control actions in terms of parameter tuning and reconfiguration options. In Section 5 we present preliminary results with the distributed visual tracking application. Section 6 concludes the paper and discusses future work.

2. RELATED WORK

The application of control theories has been explored in recent work in the area of QoS adaptation. In one paper,[2] the application of control theory is suggested as a future research direction to analyze adaptation behavior in wireless environments. In another,[3] a control model is proposed for adaptive QoS specification in an end-to-end scenario. In the third example,[4] the time variations along the transmission path of a telerobotics system are modeled as disturbances in the proposed perturbed plant model, in which the mobile robot is the target to be controlled. In our previous work,[1] theoretical proofs are given for various properties applying control theory to model QoS adaptation.

It is a widely accepted fact that flexible distributed applications need to be adaptive in a certain QoS range between $[QoS_{min}, QoS_{max}]$,[5] in order to provide a graceful reaction to dynamic resource availability in a distributed environment. Various schemes[5][6][7][8] have been proposed. Our work propose a different insight in that it concentrates on determining *adaptive policies*, instead of *adaptive mechanisms*. We use control theory to quantitatively determine the states of the adaptive system, and based on these states we activate control algorithms to adapt to the dynamics of the system. The output of the control algorithms will then be translated on-the-fly to parameter-tuning and reconfiguration choices within an application.

Our work is also closely related to and utilizes the knowledge of dynamic resource allocations. In one example[9] the work focuses on maximizing the utility functions, while keeping QoS received by each application within a feasible range. In the second example[10] it focuses on a multi-machine environment running a single complex application, and the objective is to dynamically change the configuration of the application to adapt to the environment. In comparison, our work focuses on the analysis of the actual adaptation dynamics, rather than individual or overall utility factors. We also focus on an environment with multiple applications competing for a pool of shared resources, which we believe is a common scenario easily found in many actual systems.

A critical character of a closed-loop control system is its feedback path. Various related work similarly utilize feedback information for adaptation purposes. For example, the work from OGI[11] uses software feedback mechanisms that enhance system adaptation awareness by adjusting video sending rate according to on-the-fly network variations. However, the algorithms used in most of the previous work are heuristic in nature, and various adaptation properties such as stability, steady-state fairness and adaptation agility of the algorithms are not addressed.

3. CONTROLLING MULTIMEDIA APPLICATIONS IN MIDDLEWARE COMPONENTS

In order to control the adaptive behavior of distributed applications according to resource availability, we integrated the Task Control Model as proposed in our previous work[1] into middleware components. We review the model briefly in the context of a real world application, the distributed visual tracking. In the next section we further extend our previous work and emphasize the translations between theoretical output from control algorithms and actual control actions in terms of parameter tuning and reconfiguration options.

3.1. The Task Control Model for Middleware Components

Before proceeding to analyze quantitative properties of the adaptive behavior, we need a model to map execution environment of flexible distributed applications to the paradigm of traditional control systems. For this purpose, we consider each distributed application as an ensemble of functional components, which we refer to as *tasks*. Tasks are execution units that perform certain actions to deliver results to other tasks or the end user. All tasks in an application can be presented as directed acyclic graph, which illustrates the producer-consumer dependency among tasks. A directed edge from task T_i to T_j indicates that T_j uses the output produced by T_i. Each task can be uniquely characterized by its *input quality*, *output quality* and *utilized resources*.[12]

The Task Control Model focuses on a single task in the application, referred to as a *Target Task*. The Target Task is the task to be controlled by the middleware components. Based on the analogy with control systems, we also need to introduce an *Adaptation Task*, which enforces the control policy, as well as an *Observation Task*, which observes or estimates the states of the *Target Task* and feeds them back to the *Adaptation Task*. Both tasks are implemented in respective functional middleware components. By utilizing a translation component that translates the theoretical output of the Adaptation Task, The combination of these two tasks effectively controls the Target Task in the application, and assures that the output quality of critical quality parameters is preserved within the desired QoS level, regardless of variations in resource availability.

In the scope of this paper, we present a distributed visual tracking application that exemplifies our approach. For demonstrating capabilities in middleware components, we have developed a distributed visual tracking application based on the XVision[13] project. The basic operations of the application are described as follows. A tracking server grabs live video frames from a video camera, and sends the video feed over the transmission network to the tracking client. The client executes a complicated kernel tracking algorithm, which identifies and tracks the object of interest to the user. The tracking result is presented to the user visually showing coordinates of the tracked object. Obviously, the critical quality parameter in this application is tracking precision.

The distributed visual tracking application is flexible in the following manner. First, since network bandwidth between the tracking server and the tracking client may fluctuate, the image quality and timeliness may be affected, thus reducing tracking precision. The application may downgrade to lower image resolution or smaller image size and still be able to preserve tracking precision. Second, since the tracking algorithm can track multiple objects simultaneously, when the reserved or actual CPU cycles are not sufficient for all objects, it may downgrade by tracking only the most important objects. Thirdly, the application may wish to reconfigure itself and add compression or security modules in order to take advantage of excessive CPU cycles and release the burden on transmission bandwidth. The application shows its superior flexibility by the above reconfiguration and parameter tuning choices.

Given these reconfigure options within the application, our objective is to adequately model the Target Task, and utilize the Task Control Model to control adaptive behavior of the application in middleware components. An example of the Task

Control Model presented in the form of a directed acyclic graph is shown in Figure 1(a), and an example for the distributed visual tracking application is shown in Figure 1(b).

(a) Task Control Model in the form of a Directed Acyclic Graph

(b) A distributed visual tracking application

Figure 1. The Task Control Model in the context of a distributed visual tracking application

3.2. A Linear Model for the Target Task

In order to develop the control algorithms in the Adaptation Tasks, we need to have a precise analytical model to characterize the internal dynamics in the Target Task. We refer to the independent parameters in this model as *task states*. If $\mathbf{x}$ denotes task states, $\mathbf{u}$ denotes controlled input, $\mathbf{y}$ denotes system output, $\mathbf{w}$ denotes system noise, $\mathbf{z}$ denotes observation, and $\mathbf{v}$ denotes observation error, we confine the scope of this paper to linear and discrete-time models described by the following form:

$$\mathbf{x}(k) = \mathbf{\Phi}\mathbf{x}(k-1) + \mathbf{\Gamma}\mathbf{u}(k-1) + \mathbf{w}(k-1) \tag{1}$$

$$\mathbf{y}(k) = \mathbf{H}\mathbf{x}(k) \tag{2}$$

$$\mathbf{z}(k) = \mathbf{y}(k) + \mathbf{v}(k) \tag{3}$$

where $k = 1$ to k_{max} with k_{max} as the maximum possible time instant, and $\mathbf{\Phi}$, $\mathbf{\Gamma}$, and $\mathbf{H}$ are known matrices without error. We assume in later examples that the *Target Task* can be characterized by discrete-time and linear difference equations as Equations (1), (2), and (3).

Now the work left to be done is to give a scenario in which a model in the form of Equations (1), (2) and (3) can be established and further utilized for designing control algorithms and translation mechanisms in middleware components.

3.2.1. A Scenario of Modeling Target Tasks

To show a concrete example of a Task Control Model, we consider the following scenario. Let us assume multiple tasks competing for a shared resource pool with the capacity C_{max}. Each task T_i makes *new requests* r_i for resources in order to perform their actions on inputs and produce outputs. These requests may be *granted* or *outstanding*. If a request is granted, resources are allocated immediately. Otherwise, the request waits with an outstanding status until it is granted. The system grants requests from multiple tasks with a constant *request granting rate* y.

For different types of resources, the notation *resource requests* is interpreted differently. For *temporal* resources, such as communication bandwidth and CPU, where the resources are shared in a temporal fashion, outstanding resource requests are mapped to data in the waiting queue, and granted requests are mapped to allocated temporal resources, such as transmission bandwidth. For example, for transmission tasks, the request granting rate y denotes the total physical bandwidth of the communication channel, while the *granted requests* denote data that has completed transmission over the channel, and the *outstanding requests* denote data that is in flight in the channel or in the waiting queue.

Figure 2 illustrates the above scenario. As the figure shows, the *new request* rate needs to be throttled by the *Adaptation Task*, referred to as *throttled request rate*.

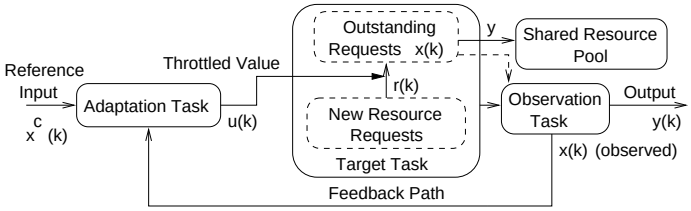


Figure 2. A Scenario of Modeling Target Tasks

3.2.2. A Linear Model for Target Tasks in the Scenario

In the above scenario, we can develop a linear model for the Target Task. In the case that complete system states can be observed, for example in a Target Task that shares CPU or memory resources on the same end system with other tasks, it is possible to design control algorithms in the Adaptation Task to promote fairness among all tasks sharing resources, according to their respective importance weight factor. Our previous work[1] analyzed this case theoretically, and presented analytical results when using PID control as the control algorithm in the Adaptation Task. In the context of a distributed visual tracking application, the next section extend previous work and present translation mechanisms implemented in a standalone component, which maps theoretical values from Adaptation Tasks to parameter tuning or reconfigure choices within the application.

We define the following notions for *Target Task* T_i:

1. t_c is a constant sampling time interval, which is the time elapsed in interval $[k, k+1]$, k being time instants satisfying $k = 1$ to k_{max};
2. $y_i(k)$ is the number of granted requests for T_i in interval [k, k+1], and the observation of which is $z_i(k)$ with an error of $v_i(k)$;
3. y is the total number of granted requests for all tasks, which we assume to be a constant;
4. $u_i(k)$ is the number of throttled resource requests in $[k, k+1]$ controlled by *Adaptation Task* for T_i;
5. $x_i(k)$ is the number of outstanding resource requests made by T_i;
6. $x(k)$ is the *total number* of *outstanding resource requests* made by *all tasks* at time k;

With these notations, we may define the following to model the Target Task:

$$\dot{x} = x(k) - x(k-1) = \sum_{i=0}^{M(k-1)} u_i(k-1) - y \tag{4}$$

where $M(k)$ is the total number of active tasks competing for resources in the system in $[k, k+1]$.

The difference equation (4) depict the internal dynamics of the Target Task. Intuitively, the difference between outstanding resource requests of two adjacent time instants should be equivalent to the difference the input and the granted (output) quantities in terms of resource requests. In our scenario, with the presence of the Adaptation Task, the input resource request rate is the same as *throttled* resource requests $u_i(k)$.

3.3. Control Algorithms in Adaptation Tasks

In our previous work we developed a standard proportional-integral- derivative (PID) control[14] † is used, then $u_i(k)$ obeys the equation

$$u_i(k) = u_i(k-1) + \alpha[x^c(k) - x(k)] + \beta\{[x^c(k) - x(k)] - [x^c(k-1) - x(k-1)]\} \tag{5}$$

where α and β are configurable scaling factors. We were able to prove the following properties given priority weights for each task:

† PID control is a classic control algorithm where the control signal is a linear combination of the error, the time integral of the error, and the rate of change of the error.

- *Equilibrium:* The number of outstanding resource requests x in the system, established by Equation (4) and (5), will converge to an equilibrium value which equals to the reference value x^c.
- *Fairness:* The system fairly shares resources among competing tasks according to the weighted max-min fairness property, based on their priority weights.
- *Stability:* There exist appropriate values of parameters α and β so that all tasks in the system are asymptotically stable around a local neighborhood, for any pre-determined priority weight for task T_i.

4. DESIGN OF THE TRANSLATION MECHANISMS

Within the scope of middleware components, translation mechanisms need to be introduced on an application-specific basis in order to map theoretical adaptation values to actual control actions. For this purpose, we introduce a translation component in the middleware. The major advantage of the component is that it offers a common architecture for all application tasks yet still fully parameterized and configurable. We take the distributed visual tracking application as an example to illustrate our translation schemes.

4.1. Parameterized Linear Translation

In the first case, one or more performance or application-specific parameters are *linearly* related to the abstract notion *resource requests* in our scenario where we developed the model for Target Tasks. As a start, we assume that the parameter is related to a single type of resource. In this case, it is sufficient to design a linear translation scheme to transform theoretical values linearly with coefficients specified by the application task.

If we denote p_i as the parameter related to resource requests in the Target Task T_i, which means that by tuning parameter p_i, the corresponding new resource request rate r_i are decided according to a linear function with known coefficient γ_i:

$$r_i(k) = \gamma_i p_i(k) \tag{6}$$

It follows that the controlled value p_i', which is the control decision made by the Adaptation Task for T_i, is decided by the equation:

$$p_i'(k) = \frac{1}{\gamma_i} u_i(k) = \frac{p_i(k)}{r_i(k)} u_i(k) \tag{7}$$

In our example of distributed visual tracking, the critical quality parameter, *tracking precision*, is linearly related to the network bandwidth in the following manner. First, tracking precision is linearly related to the frame rate; Second, the frame rate is linearly related to network bandwidth between client and server, since the application uses uncompressed RGB image format. In our experiments, one of the parameters that are controlled from the middleware components is the *frame size* for each image sent from the tracking server. By adjusting frame size, tracking precision can be controlled regardless of the bandwidth variations. Following Equation (6), if $r_i(k)$ denotes bandwidth requested, $fs_i(k)$ denotes frame size, then $\gamma_i = fr^s * bpp$, where fr^s is the sustained frame rate and bpp is bytes used per pixel in the uncompressed image. Following Equation (7) we have $fs_i'(k) = u_i(k)/(fr^s * bpp)$, where $u_i(k)$ is derived from Equation (5). In our experiments, $bpp = 4$.

Note that the sustained frame rate fr^s is linearly related to *tracking precision* that we sought. It is irrelevant with $x^c(k)$, which is the reference for the control algorithm in the Adaptation Task. This implies that the desired sustained level for quality parameters in the application is expressed within the translation component, rather than the Adaptation Task. Normally, $x_c(k)$ should be set to the setpoint desired for the queue of residual outstanding requests, which can be 0, or a positive value in order to increase utilization when resource granting rate y increases, showing an system with abundant resources.

4.2. Configurable Nonlinear Translation: A Fuzzy Logic Approach

In the second case, the performance or application-specific parameters to be controlled are not linearly related to the abstract notion of *resource requests*, or it may be the case that rather than parameter tuning, reconfiguration options are to be adopted. In this case, we propose a fuzzy logic approach to associate the theoretical output from Adaptation Tasks with the decision of control actions.

We adopt a fuzzy logic approach in the translation mechanisms. There are two justifications. First, in the case of a quantitatively tunable parameter, The control decision needs to balance between the precision of control actions to follow theoretical $u_i(k)$ values and the frequency of control. Second, this tradeoff decision needs to be configurable according to the requirements of specific applications. The simple approach of engaging threshold values are not sufficient. Third, in the case of reconfiguration options, the choices among different reconfiguration option depend on the current $u_i(k)$ values from the Adaptation Tasks and the resource requirements for each reconfiguration option. The optimal decision on which reconfiguration option to use is a decision making process with multiple objectives, each objective reflecting the *throttled request rate* values $u_i(k)$ for a different type of resources. These complications warrant a fuzzy logic approach to make optimal translations.

We assume that there are three Adaptation Tasks in the middleware components for Target Task T_i, each one corresponding to a type of resource shared by task T_i, being CPU, memory and transmission bandwidth, respectively. Associated with T_i, we have three respective *throttled request rate* values generated by three Adaptation Tasks for the three types of resources: $u_i^c(k)$, $u_i^m(k)$ and $u_i^b(k)$, respectively, at time k. Similarly, we also have three values corresponding to the actual new requests made by T_i, namely, $r_i^c(k)$, $r_i^m(k)$ and $r_i^b(k)$.

4.2.1. Translation for Parameter Tuning Actions

We assume that all quantitatively tunable parameters are associated with a single task T_i. We propose to utilize fuzzy logic and fuzzy set theories to translate *throttled request rate* $u_i(k)$ values to parameter tuning actions as follows.

Without loss of generality, we take CPU resource type and its corresponding $u_i^c(k)$ and $r_i^c(k)$ as an example. Assume task T_i defines a pair of parameter tuning actions: a function F_u that can increase CPU utilization in T_i, and a function F_d that can decrease CPU utilization in T_i. We also define *control error* values $e_i^c(k)$ as follows:

$$e_i^c(k) = r_i^c(k) - u_i^c(k) \tag{8}$$

Obviously, the *universe of discourse* U in the fuzzy set theory[15] corresponds to the set of all possible values of $e_i^c(k)$. We can thus define the membership function *PositiveError* of a fuzzy set *PositiveSet* as a function of:

$$\textit{PositiveError} : U^+ \rightarrow [0, 1]. \tag{9}$$

Where U^+ is the positive half of U. So, every element $e_i^c(k)$ from U^+ has a membership degree *PositiveError* ($e_i^c(k)$) $\in [0, 1]$, where *PositiveSet* is completely determined by the set of tuples

$$\textit{PositiveSet} = \{(e_i^c(k), \textit{PositiveError}(e_i^c(k))) \mid e_i^c(k) \in U\} \tag{10}$$

Similarly, we can define membership functions *NegativeError* and *Correct* for fuzzy sets *NegativeSet* and *CorrectSet*. The relationships of these membership functions and $e_i^c(k)$ values are illustrated in Figure 3, where a simple linear membership function is shown.

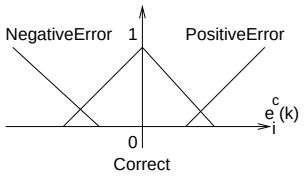


Figure 3. The Membership Functions and Fuzzy Sets related to $e_i^c(k)$

With definitions shown above, we design the following algorithm to decide appropriate control actions:

switch max{*NegativeError*($e_i^c(k)$), *Correct*($e_i^c(k)$), *PositiveError*($e_i^c(k)$)}

begin

case *NegativeError*($e_i^c(k)$): ActivateControlAction(F_u, *NegativeError*($e_i^c(k)$)).

case *PositiveError*($e_i^c(k)$): ActivateControlAction(F_d, *PositiveError*($e_i^c(k)$)).

case *Correct*($e_i^c(k)$): No changes.

end.

where **ActivateControlAction** calls the function F_u or F_d with the second parameter as a scaling factor, which determines the adjustment extent with regards to F_u or F_d.

Our approach is fully configurable according to the application's needs, by changing membership functions *NegativeError*, *Correct* and *PositiveError*. For example, if the application wishes to adapt only in extreme cases, both functions *NegativeError* and *PositiveError* can be specified to cover only those values at extreme ends.

4.2.2. Translation For Reconfiguration Control Actions

Reconfiguration needs to be performed only in extreme cases where the current configuration does not match the environment. We extend our approach in the previous section for parameter tuning, in order to determine appropriate activation timing for the reconfiguration process.

If there are multiple reconfiguration choices, a selection algorithm is needed to choose the optimal selection of one reconfiguration alternative. We assume a universe of reconfiguration options (alternatives) A and a collection $\{O\}$ of selection objectives, each objective corresponding to a type of resource. The goal is to evaluate how well each alternative satisfies each objective, and to combine the weighted objectives into an overall decision function in some plausible way.

We define a universe of n alternatives, $A = \{a_1, a_2, \ldots, a_n\}$ and a set of j objectives, $O = \{O_1, O_2, \ldots, O_j\}$. Let O_i indicate the ith objective. The degree of membership of alternative a in O_i, denoted by $\mu_{O_i}(a)$ is the degree to which alternative a satisfies the criteria specified for this objective. We seek a decision function that simultaneously satisfies all of the decision objectives; hence, the decision function, D, is given by the intersection of all the objective sets,

$$D = O_1 \cap O_2 \cap \cdots \cap O_j \tag{11}$$

Therefore, the grade of membership that the decision function, D, has for each alternative a is given by

$$\mu_D(a) = \min\{\mu_{O_1}(a), \mu_{O_2}(a), \ldots, \mu_{O_j}(a)\} \tag{12}$$

The optimum decision, a^*, will then be the infimum alternative that satisfies

$$\mu_D(a^*) = \max_{a \in A}(\mu_D(a)) \tag{13}$$

We now define a set of values, $\{G\}$, which we will constrain to being linear and ordinal. These values are originally the *adapted request rate* $u_i(k)$ values established by the Adaptation Task, and then scaled to a pre-determined linearly ordered scale, e.g. $[-1, 1]$, $[1, 10]$, etc., according to the maximum and minimum values in the respective domain of resource types. These scaled values will be attached to each of the objectives to quantify the influence that each objective should have on the chosen alternative. Let the parameter, g_i, be contained on the set of scaled values $\{G\}$, where $i = 1, 2, \ldots, j$. Hence, we have for each objective a measure of how important it is to the decision.

The decision function, D, now takes on a more general form when each objective is associated with a weight expressing its importance to the decision. This function is represented as the intersection of j-tuples, denoted as a *decision measure*, $M(O_i, g_i)$, involving objectives and preferences,

$$D = M(O_1, g_1) \cap M(O_2, g_2) \cap \cdots \cap M(O_j, g_j) \tag{14}$$

A key question is what operation should relate each objective, O_i, and its guidance value, g_i, that preserves the linear ordering required of the guidance value set $\{G\}$, and at the same time relates the two quantities in a logical way where negation is also accommodated. it turns out that the classical implication operator satisfies all of these requirements. Hence, the decision measure for a particular alternative, a, can be replaced with a classical implication of the form,

$$M(O_i(a), g_i) = g_i \rightarrow O_i(a) = \neg g_i \vee O_i(a) \tag{15}$$

Therefore, a reasonable decision model will be the joint intersection of j decision measures,

$$D = \bigcap_{i=1}^{j} (\neg g_i \cup O_i) \tag{16}$$

and the optimum solution, a^*, is the alternative that maximizes D. If we define

$$C_i = \neg g_i \cup O_i \text{ hence } \mu_{C_i}(a) = \max[\mu_{\neg g_i}(a), \mu_{O_i}(a)] \tag{17}$$

the optimum solution, expressed in membership form, is given by

$$\mu_D(a^*) = \max_{a \in A}[\min\{\mu_{C_1}(a), \mu_{C_2}(a), \ldots, \mu_{C_r}(a)\}] \tag{18}$$

This model is intuitive in the following manner. As the ith objective becomes more important in the final decision, g_i increases, causing $\neg g_i$ to decrease, which in turn causes $C_i(a)$ to decrease, thereby increasing the likelihood that $C_i(a) = O_i(a)$, where now $O_i(a)$ will be the value of the decision function, D, representing alternative a. As we repeat this process for other alternatives, a, Equation (18) reveals that the largest value $O_i(a)$ for other alternatives will eventually result in the choice of the optimum solution, a^*. This is exactly how we would want the process to work.

5. PRELIMINARY EXPERIMENTAL RESULTS

Based on the control algorithms and translation mechanisms developed in previous sections, we have implemented a middleware framework to control the distributed visual tracking application, which is the example throughout the paper. Based on tracking algorithms in the XVision[13] project, we have successfully implemented this application on the Windows NT 4.0 platform in Visual C++, using Windows Sockets 2 layer for the network transmission.

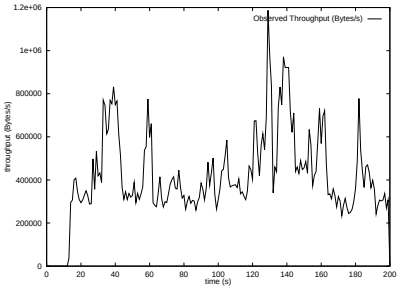
We implemented Adaptation Tasks, Observation Tasks and translation mechanisms as middleware components in C++ and Java, which interact among one another and with the application using service enabling platforms such as CORBA. We use ORBacus 2.0.4[16] as our CORBA implementation. In the case of visual tracking, the primary quality parameter is the tracking precision. For quality assurance of this parameter, we may sacrifice less crucial quality parameters such as frame size of the live video feed.

In order to simulate bandwidth fluctuations in a typical distributed environment over the Internet, we have also implemented a network simulator, which simulates packet delay through a transmission path of multiple network routers, each of them implementing the FIFO scheduling algorithm. Because of the bursty nature of cross traffic, throughput fluctuations may occur at various times over the connection.

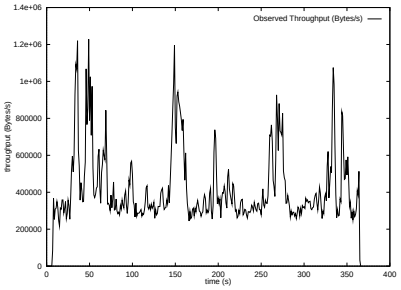
We evaluated the effects of the middleware components on one of the quantitative controllable parameters in the distributed Visual Tracking system: the tracking precision. The objective is to assure the quality of tracking precision, in spite of bandwidth fluctuations. By adopting linear translation mechanisms developed previously as middleware components, the frame size of the application is adjusted when necessary in order to maintain the tracking precision.

For the purpose of repeating the same set of experiments and for measurements of tracking precision, we use a computer generated image sequence, in which the object moves at fixed speed and path. For the experimental results shown in Figure 4, the moving speed of the rectangle is set at a constant 3 pixels per second continuously. In addition, we assume there are no other CPU intensive process running in the background on the same platform. This is for the purpose of simplifying the experiments and concentrating on a single type of resource.

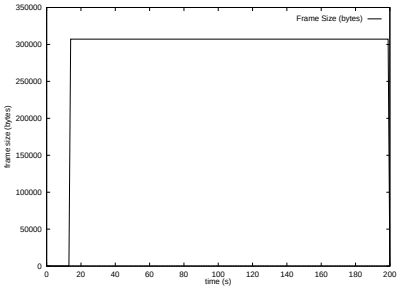
In Figure 4, the three graphs on the left shows the tracking results without any adaptation. Those on the right shows the case with adaptation support from the middleware framework, adopting the translation mechanisms developed in the previous



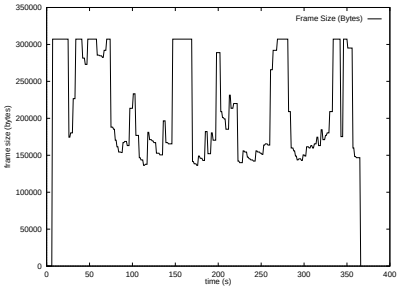
(a) Observed Throughput



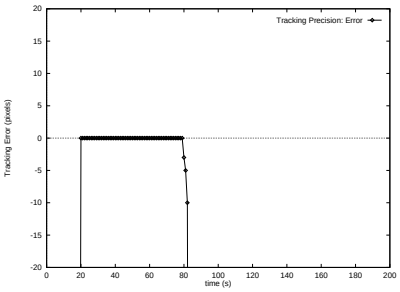
(b) Observed Throughput



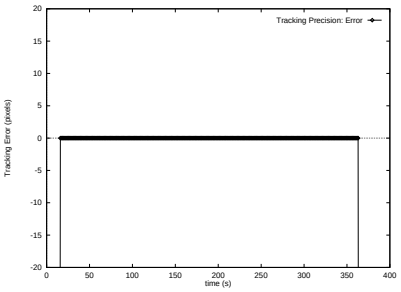
(c) Controllable Parameter: Frame Size



(d) Controllable Parameter: Frame Size



(e) Tracking Precision



(f) Tracking Precision

Figure 4 (Left): Experiments without adaptation support

Figure 4 (Right): Experiments with adaptation support

Figure 4. Experiments with the Distributed Visual Tracking Application

section. We can observe that by changing the frame size of the visual tracking application, the tracking precision will be preserved without any tracking error at all times during the connection. In contrast, without any adaptation, when the network throughput degrades to a certain degree, the tracking algorithm is not able to keep track of the object, the error accumulates rapidly verifying that the tracking algorithm loses the object. This prove-of-concept system proves that the approach we have taken is effective in preserving tracking precision in a distributed environment, such as the Internet, with fluctuating bandwidth between the tracking client and server.

6. CONCLUSIONS

In this work, we focus on the scenario that flexible distributed multimedia applications need to adapt their behavior to variations of the resource availability and assure quality of critical QoS parameters. Based on our previous work that developed theoretical control algorithms to control the application, this work completes the control process by the development of

translation mechanisms, in order to translate theoretical values to actual control actions of parameter tuning or reconfiguration choices. In our experiments, we show that our model successfully controls a distributed visual tracking application, where network bandwidth fluctuates as in a distributed environment over the Internet. Ongoing and future work involves application configurable adaptation that takes advantage of user preferences, as well as collaboration issues involving multiple Adaptation Tasks in a unicast or multicast environment.

REFERENCES

1. B. Li and K. Nahrstedt, "A Control Theoretical Model for Quality of Service Adaptations," *Proceedings of Sixth International Workshop on Quality of Service* , 1998.
2. M. Satyanarayanan, "Fundamental Challenges in Mobile Computing," *Proceedings of the ACM Symposium on Principles of Distributed Computing* , 1996.
3. J. DeMeer, "On the Specification of End-to-End QoS Control," *Proceedings of 5th International Workshop on Quality of Service '97* , May 1997.
4. F. Goktas, J. Smith, and R. Bajcsy, "Telerobotics over Communication Networks: Control and Networking Issues," *36th IEEE Conference on Decision and Control* , 1997.
5. S. Lu, K.-W. Lee, and V. Bharghavan, "Adaptive Service in Mobile Computing Environments," *Proceedings of 5th International Workshop on Quality of Service '97* , May 1997.
6. A. Campbell and G. Coulson, "QoS Adaptive Transports: Delivering Scalable Media to the Desk Top," *IEEE Network* , 1997.
7. Z. Chen, S. Tan, R. Campbell, and Y. Li, "Real Time Video and Audio in the World Wide Web," *Proceedings of Fourth International World Wide Web Conference* , 1995.
8. 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.
9. R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A Resource Allocation Model for QoS Management," *18th IEEE Real-Time System Symposium* , 1997.
10. D. Rosu, K. Schwan, S. Yalamanchili, and R. Jha, "On Adaptive Resource Allocation for Complex Real-Time Applications," *18th IEEE Real-Time System Symposium* , 1997.
11. S. Cen, C. Pu, R. Staehli, C. Cowan, and J. Walpole, "A Distributed Real-Time MPEG Video Audio Player," *Proceedings of the 5th International Workshop on Network and Operating System Support of Digital Audio and Video (NOSSDAV'95)* , Apr. 1995.
12. 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* , Dec. 1997.
13. G. Hager and K. Toyama, "The XVision System: A General-Purpose Substrate for Portable Real-Time Vision Applications," *Computer Vision and Image Understanding* , 1997.
14. G. Franklin and J. Powell, *Digital Control of Dynamic Systems*, Addison-Wesley, 1981.
15. B. Graham and A. Ollero, *An Introduction to Fuzzy Control*, Springer, 1996.
16. O. O. C. Inc., "ORBacus for C++ and Java," *ftp://ftp.ooc.com/pub/OB/3.1/OB-3.1b1.pdf* , 1998.