NAIST-IS-DD0361039

Doctoral Dissertation

Achieving Quality of Service in Distributed Multi-media Systems

Tao Sun

February 02, 2006

Department of Information Processing
Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Tao Sun

Thesis Committee:

Professor Minoru Ito	(Supervisor)
Professor Hiroyuki Seki	(Co-supervisor)
Associate Professor Keiichi Yasumoto	(Co-supervisor)

Achieving Quality of Service in Distributed Multi-media Systems*

Tao Sun

Abstract

In distributed multi-media systems, the QoS (quality of service) is restricted due to various factors such as bandwidth, network jitter, performance of servers/ clients, battery amount and so on. It is important how to achieve QoS which meets user's requirements within these restrictions.

With the development of mobile systems and wireless networks, various kinds of mobile terminals become part of distributed multimedia systems. However, the restrictions, especially the limitation of battery amount restricts the quality of video/audio playback, leading to the end user's dissatisfaction. So, we need a technique to adapt application-level QoS for end mobile users depending on battery amount. With the development of distributed multi-media systems, the performance of end terminals and user's requirements become manifold. In order to provide multiple users with multimedia streaming services, the multi-media systems should be highly functional, scalable and robust. For this purpose, the traditional server/client architecture is already obsolete. Instead, we need a new architecture including an efficient delivery network based on peer-to-peer overlay network and functional components distributed among multiple distant nodes for accommodating a large number of users and continuing multimedia services even with node/link failures. In order to guarantee the performance and/or appropriateness of QoS adaptation mechanisms implemented as a software system, it is important to test whether each mechanism is correctly implemented, without executing the whole system.

* Doctoral Dissertation, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0361039, February 02, 2006.

So a new method for testing QoS in multimedia systems is indispensable. This thesis provides the following three research topics.

First, in order to guarantee the correctness of QoS adaptation mechanisms, we propose a testing method for QoS functions in distributed multi-media systems. In the proposed test method, we use a statistical approach where test sequences take samplings of actual frame rates and/or time lags when an IUT (implementation under test) is executed, and report test results from ratio of samplings with low quality below a threshold in a normal distribution of all samplings.

Secondly, a QoS adaptation method for streaming video playback for portable computing devices where playback quality of each video segment is automatically adjusted from the remaining battery amount, desirable playback duration and the user's preference to each segment, is proposed. In this method, we assume that video segments are classified into several predefined categories. Each user specifies relative importance among categories and preferred video property such as motion speed and vividness for each category. From the information, playback quality and property of each category are determined so that the video playback can last for the specified duration within the battery amount.

Finally, a new video delivery method called *MTcast (Multiple Transcode based video multicast)* is proposed. It achieves efficient simultaneous video delivery to multiple users with different quality requirements by relying on user nodes to transcode and forward video to other user nodes. In MTcast, each user specifies a quality requirement for a video consisting of bitrate, picture size and frame rate based on the user's environmental resource limitation. All users can receive video with the specified quality (or near this quality) along a single delivery tree.

Some experimental results show that our proposed test method works effectively for QoS functional tests in a typical multimedia playback program, and our QoS adaptation method improves playback quality of important categories a few times better than flattening the playback quality. Through simulations, we show that our video delivery method can achieve much higher user satisfaction degree and robustness against node failure than the layered multicast method.

Keywords:

multimedia, QoS, distributed systems, peer-to-peer, transcode, test

Contents

List of Figures

List of Tables

1. Introduction

As broadband infrastructure is widespread in the Internet, the distributed multimedia systems have been deployed rapidly. The distributed multi-media system is the system where the multimedia objects such as a video and a voice are exchanged between multiple contents servers and user nodes in real time, and all of user nodes are connected to a distributed system such as the Internet. This revolution is transforming the way people live, work, and interact with one another, and is impacting businesses, government services, education and so on.

In general, the quality of service (QoS) is restricted due to various factors such as bandwidth, network jitter, performance of servers/clients, battery amount and so on. So QoS (Quality of Service) requirements vary considerably depending on types of networks and user terminals.

The current trend in distributed multimedia systems is to tackle adaptation to ubiquitous environments. With the development of mobile systems and wireless networks, various kinds of mobile terminals become part of distributed multimedia systems. However, the restrictions, especially the limitation of battery amount restricts the quality of video/audio playback, leading to the end user's dissatisfaction. This makes it difficult to integrate mobile terminals in distributed multi-media systems. So, we need a technique to adapt application-level QoS for end mobile users depending on battery amount. With the development of distributed multi-media systems, the performance of end terminals and requests of users become manifold. In order to provide multiple users with multimedia streaming services, the multi-media systems should be highly functional, scalable and robust. For this purpose, the traditional server/client architecture is already obsolete. Instead, we need a new architecture including an efficient delivery network based on peer-to-peer overlay network and functional components distributed among multiple distant nodes for accommodating a large number of users and continuing multimedia services even with node/link failures.

In order to guarantee the performance and/or appropriateness of QoS adaptation mechanisms implemented as a software system, it is important for distributed multi-media systems to provide multi-media services with certain QoS functions to end users satisfying user's requests. Among various QoS functions, control mechanisms for the frame rate (the number of frames displayed every second), size of picture, bit-rate and for the lip synchronization [8] among multiple concurrent media objects seem most important ones.

For development of multi-media systems with high reliability, it is desirable to establish a method for testing if those QoS functions are correctly implemented in a given IUT (Implementation Under Test). Traditional software testing methods which have succeeded in protocol engineering, focus mainly on the correctness of input/output correspondences[11]. Therefore those methods could not be directly applied to QoS testing such as video/audio playback timings. Essentially important test problems of multi-media systems are not only correspondence relations of input/output actions, but also time difference between an input and the corresponding output or the time duration in executing a sequence of actions. Then, even if all the correspondence relations of input/output actions hold, it is not necessarily guaranteed to pass a test due to the time difference between input and output actions.

Even if temporal relations among multi-media objects are specified in detail by a formalism such as Timed CTL [2] used in real time systems, it makes test sequences be explosively complicated and is not realistic to test multi-media systems. Suppose that a specification for a video playback system specifies that a video frame is drawn exactly every 33 msec plus/minus 5msec. In general, when an IUT does not satisfy such a specification a little (e.g., only a frame was delayed for 10msec), it may not be considered a problem as long as media objects are played back naturally. So, for QoS testing for the playback of media objects, it is desirable to statistically analyze the temporal relations and give test results based on statistically calculated information[14].

There exist some studies on multi-media testing such as testing of multi-media transmission systems [5], the quality of multi-media contents [6] and interoperability and performance testing of distributed systems [14]. However they do not deal with testing on temporal relations of input/output actions. A few researches, which deal with temporal relations in multi-media systems explicitly, have been reported [4, 12]. These studies propose a method to test binary temporal relations on the starting time and/or the ending time between two objects by using a statistical approach. However they do not deal with the quality during playback of an object. Moreover, [16] proposes a method for functional testing of media synchronization protocols using a concurrent timed I/O automata model, where a given IUT is tested by executing each input action within an appropriate time interval calculated in advance, and by observing whether execution timings of output actions are within the appropriate time interval. However, this approach does not consider statistical aspects of multi-media systems. As a different approach, [3] proposes a model checking method for media constraints such as lip synchronization among multiple media objects.

In Chapter 2, a new testing method based on statistical approach for QoS functions in distributed multi-media systems is introduced.

Another problem in multi-media system is to keep good playback quality for portable computing devices (such as PDAs and smart phones) within the limitation of battery amount. Owing to recent innovations in portable computing devices and wireless communication infrastructure (such as WLAN and wide-band CDMA), we can watch video contents using those devices every time and everywhere. However, the restrictions such as bandwidth and battery amount are so hard that the portable computing device users may not be satisfied with the quality of playback.

In such an on-demand video streaming and playback, however, the portable computing device consumes much more battery amount than other business applications, due mainly to video decoding and wireless communication. Consequently, there will be a demand to control battery life depending on user requirements or situations so that the battery is not exhausted until the video playback finishes or the specified amount of battery (e.g., 50%) is left after playing back the video content. In general, we can extend video playback duration by reducing video quality. However, it is difficult to estimate how long the battery lasts for each playback quality, since the battery consumption depends on not only video playback but also other factors such as OS, LCD, and so on. Besides it, these factors are device dependent. Moreover, when the remaining battery amount is small and the playback quality is reduced over the whole playback duration, users may get frustrated due to low playback quality. In order to mitigate such a situation, some fragments of a video important for a user should be played back with higher quality than others. Also, to each fragment, a user should be able to specify playback preferences such as balance of motion speed and vividness.

In Chapter 3, a QoS control mechanism which enables playback of multiple video segments with different playback quality based on the user's preference, is presented. When a user specifies relative importance among video segments and preferred video property (proportion between picture size and frame rate) for each segment, the proposed algorithm determines the playback quality of each video segment so that the video playback can last for the specified duration within the remaining battery amount.

Due to difference in environmental limitation, performance of terminals and users' preferences, the required quality for a video stream becomes different among users. On the other hand, as a consequence of recent innovation of information technology and widespread of the Internet, various types of computing devices such as PCs, PDAs, cell phones, car computers, and set-top boxes can be connected to the Internet via various types of communication infrastructures including ADSL, WLAN, W-CDMA, Bluetooth and so on.

Accordingly, there would be a demand for an efficient video delivery method for these heterogeneous user nodes which have different computation powers, display sizes and available bandwidths.

There are several approaches for simultaneously delivering video to multiple users with different quality requirements. In the multiversion technique [34, 35], multiple versions of a video with different bitrates are prepared in advance so that the best one can be delivered to each user, within resource limitation.

In the online transcoding method [36], an original video is transcoded at a server or an intermediate node (i.e. proxy) to videos with various quality, according to receivers' preferences, and forwarded to the receivers.

In the layered multicast method [37, 38, 39, 40, 41, 42], video is encoded with layered coding techniques such as [43] so that each user can decode the video by receiving arbitrary number of layers. Since each layer is delivered as an independent multicast stream, each user can receive as many layers as possible within his/her resource limitation. In this method, as the number of users increases, more layers are required in order to improve user satisfaction degree. However, decoding video from many layers consumes large processing power and buffers. In

[37], a method for optimizing bitrate of each layer to maximize user satisfaction degree is proposed. In the multiversion method, the control mechanism is simple, but not efficient in terms of server storage and network bandwidth usage. In the multiversion and layered multicast methods, there can be a large gap between the requested quality and the delivered quality if there are not enough number of versions or layers. It would also be difficult for these methods to treat user preference which requires large picture size and a small frame rate. On the other hand, the online transcoding method can satisfy all the above requirements since it can transcode original video to arbitrary quality video. But, large computation power required for transcoding can be a problem.

In the research of peer to peer network video streaming method, The Overlay Multicast Network Infrastructure (OMNI) [49] is proposed. It builds a multicast tree consisting of multicast service nodes which are connected by a set of clients and service to them. This distributed scheme adapts to the changing of the client distribution and network conditions. CoopNet [50] augments traditional client-server streaming with P2P streaming when the load of server exceeds it's limit. In CoopNet clients cache parts of streams, and deliver them through multiple diverse distribution trees to another client if the server is overwhelmed. Both of them do not consider the different requirements of users.

In Chapter 4, a video delivery method called *MTcast (Multiple Transcode based video multicast)* which achieves efficient simultaneous video delivery to multiple users with different quality requirements by relying on user nodes to transcode and forward video to other user nodes, is proposed.

Hereafter, in Chapter 2, a testing method for QoS functions in distributed multi-media systems using a statistical approach is introduced. In Chapter 3, a QoS adaptation method for streaming video playback for portable computing devices is presented. In Chapter 4, a new video delivery method, which achieves efficient simultaneous video delivery to multiple users with different quality requirements, is proposed. Finally, Chapter 5 concludes this thesis.

2. QoS Functional Testing for Multi-media Systems

2.1 Introduction

In this chapter, in typical distributed multi-media systems consisting of servers and clients connected via a network, a method for testing QoS of media playback functions with respect to frame rates and lip-synchronization for a client program (IUT), is proposed. In the proposed method, we specify scenarios for QoS functional tests for an IUT in timed EFSM (a hybrid model of EFSM and timed automata [1]), where we designate the input flow characteristics like jitter / packet loss ratio and the play-back quality of frames to be realized for the given traffic. Furthermore, we specify the quality of inter-media synchronization as a constraint to be satisfied among sub-scenarios corresponding to concurrent playbacks of different media objects (e.g., video and corresponding audio) by using the constraint oriented description style [13]. From those test scenarios, we generate test sequences to test whether a given IUT realizes the specified QoS.

In the proposed technique, the specifications of an input flow and the playback quality to be realized (e.g., the range of fluctuation of frame rates) on playback behaviors is given in timed EFSMs.

In the proposed test method, we use a statistical approach where test sequences take samplings of actual frame rates and/or time lags between the latest frames on multiple objects when an IUT is executed, and report test results from ratio of low quality samplings (e.g., frame rates less than a threshold) below a specified threshold in a normal distribution of all samplings.

We have implemented a test system for executing test sequences in real-time in Java language, and applied it to a sample video playback program. Our experimental results show that our method works effectively for QoS functional tests.

2.2 Outline of Proposed QoS Functional Testing

As a target, we suppose distributed multi-media systems where a server transmits a stream of a requested media object to each client. Here, we test whether the

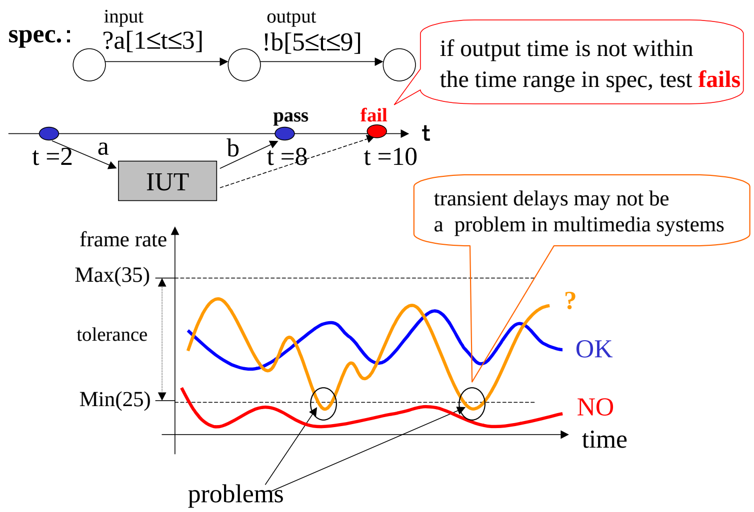


Figure 1. Existing real-time testing methods

playback quality of media objects at a client computer is feasible or not according to the characteristic of the given input flow.

In existing real-time testing methods as in [7, 10], tests are carried out by giving each input to the IUT at appropriate time, and by observing and testing if the corresponding output action is executed at appropriate time satisfying the constraints given in the specification (see Fig. 1). However, testing playback quality of media objects in multi-media systems should be different from those existing methods since some jitter in multi-media playback which may be caused by packet losses/delays is allowed to a certain extent. So, we need a new testing method for playback quality of multi-media objects.

In the proposed method, we adopt the statistical technique for testing playback quality of a single media object and of preciseness of inter-media synchronization among multiple object playbacks at a client computer of a client-server based system.

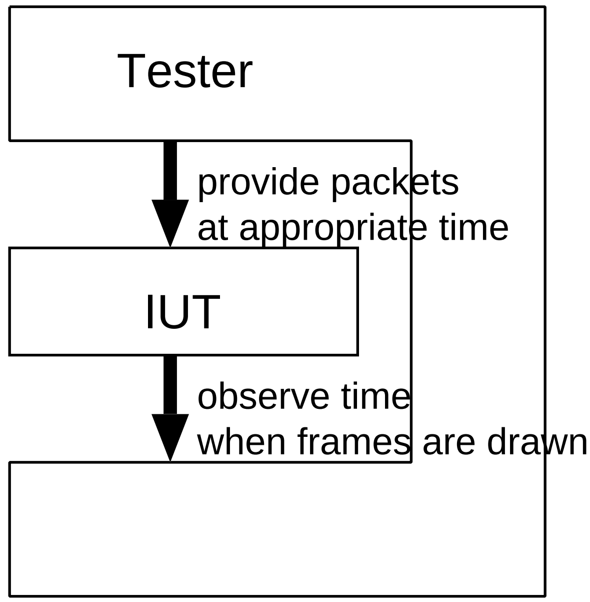


Figure 2. Environment for QoS Functional Testing

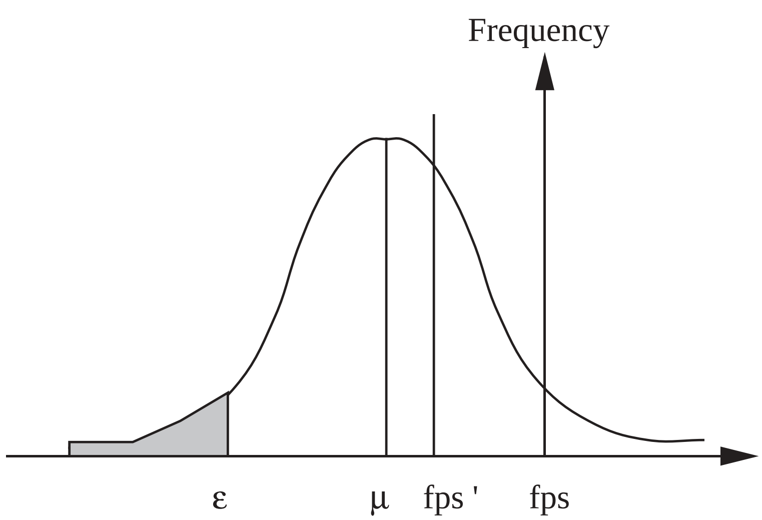


Figure 3. Distribution Graph of Frame Rates

2.2.1 Outline of Proposed Method

In the proposed method, we calculate the ideal playback quality of an object for a given data stream (flow), and test whether or not the IUT works satisfying the constraints in the specified test scenario by observing the time at which the object outputs each data unit (called *frame*. e.g., a video picture, a unit of audio data, etc) as shown in Fig. 2.

In general, when we measure the frame rate (the number of frames displayed every second) of an object in a relatively short time interval (denoted as SP), it can vary due to the characteristics of an input flow such as jitter in packet arrival time and packet losses. In the proposed method, we take a frame rate as a sampling every SP time interval for a sufficiently long time interval (denoted as LP), calculate statistic information from distribution of those samplings, and judge from the information whether or not the IUT correctly implements the control mechanism for the frame rate.

We compose a test scenario of the following two sub-scenarios: (1) *traffic testing scenario* which characterizes an input flow, and (2) *quality testing scenario* which represents behavior of the playback with certain quality. From those scenarios, we generate the following test cases (a set of test sequences).

- each test sequence transmits packets to the IUT at time within the allowable time range considering jitters and bursts specified in the traffic testing scenario.

- each test sequence measures actual packet loss ratio in the IUT for a time interval MP (such that $SP \leq MP \leq LP$), and calculates the ideal frame rate fps' which is defined in Sect. 2.2.2.

- each test sequence measures average frame rate every time interval SP by observing output from the IUT, and keeps it as a sampling. The test sequence collects samplings for time interval LP, and calculates the average value and the standard deviation from those samplings. The test sequence judges whether the IUT correctly implements the frame rate control mechanism with the specified QoS level, based on those calculated statistical values and *maximum tolerance acceptable* denoted by ϵ which the test examiner gives in advance (see Fig. 3).

For the sake of simplicity, we assume that the distribution of frame rates follows the normal distribution like in Fig. 3. Then the judgment process can be described as the following procedure.

1. calculate the average value μ and the standard deviation s derived from the samplings kept during time interval LP.

2. apply normalization expression $z = (x - \mu)/s$ to ϵ, and calculate area C that is an integral of $(-\infty, (\epsilon - \mu)/s]$ in the standard normal distribution (see Fig. 3).

3. when the value of C is smaller than the *reliance level* r which was given by the test examiner in advance, we conclude that the IUT passes the test.

In general, as shown in Fig. 3, we expect that the average frame rate μ measured during time interval LP may match neither the original frame rate fps nor the frame rate considering packet losses fps' explained in the next section, due to external and/or internal load factors.

2.2.2 Discussion about Playback Quality of Objects

The following factors are considered to give some influences to playback quality of objects.

- jitter in packet arrival time and packet loss ratio

- heavy load/low performance at a client computer

For example, suppose that a server transmits to a client a video file encoded in 30 frames/sec at a fixed transmission rate. A client computer receives packets from the server and tries to play back the video at an appropriate frame rate. In this case, the playback quality depends on the receiving rate, packet loss ratio, jitters in packet arrival time, and load of the client computer.

If the client receives packets at almost the same rate as the server transmits, if packet loss ratio is almost 0 %, and if its load is light enough, the frame rate to be achieved will be close to the originally encoded one (i.e., 30 frames/sec). On the other hand, if the packet loss ratio is high and/or the client's load is high, the frame rate will be less than 30 frames/sec. According to the above discussion, we define the ideally achievable frame rate as the following expression.

$$fps' = fps \cdot (1 - f(Loss)) \cdot \beta$$

Here, fps and $Loss$ denote the originally encoded frame rate and the packet loss ratio, respectively. $f(x)$ $(0 \leq f(x) \leq 1)$ denotes the function representing ratio of how much each packet's loss causes the frame loss. Here, $f(x) = x$ when each frame is transmitted by exactly one packet. When each frame is transmitted by several packets and/or there is inter-frame dependency like MPEG movies, $f(x)$ will be larger than x. β $(0 < \beta \leq 1)$ is another factor such as CPU load other than flow characteristics at a client computer. For the sake of simplicity, we suppose that $\beta = 1$ in this thesis.

2.3 Test Scenarios for Multi-Media Systems

As explained before, each test scenario for a multi-media system consists of a multiple sub-scenarios. We specify these scenarios in a **timed EFSM**. In timed EFSM, variables and guard expressions with those variables (i.e., execution condition of transitions) used in EFSM can be used in addition to the basic functions of timed automata[1]. Moreover, synchronization and constraints among multiple timed EFSMs are specified in the form of synchronized parallel execution of those timed EFSMs using the multi-way synchronization of LOTOS [9]. That is, we specify behavior of the whole system by making timed EFSMs execute synchronously and in parallel with the constraint oriented description style[13].

A timed EFSM is given as 6-tuple $M =< S, A, C, V, \delta, s_0 >$, where $S = \{s_0, s_1, \cdots, s_n\}$ is a finite set of states, A is a finite set of actions (events), C is a finite set of clock variables, V is a finite set of variables, $\delta : S \times A \times C \times V \rightarrow S \times V \times C$ is a transition function, and s_0 is an initial state. Let G be a set of gates which represent interaction points to an external environment and IO be a set of inputs/outputs from/to a gate. Here, $A \subseteq G \times IO$. $g?x$ represents an action which inputs a value from gate g and stores it in variable x, and $g!E$ represents an action which outputs the value of expression E to gate g, respectively. A transition function δ is represented by $s \xrightarrow[Def]{g?x[Guard]} s'$ where s and s' are the current state and the next state just after an action is executed at state s, respectively. A transition condition for an action, denoted by $Guard$, is represented by a logical conjunction of linear inequality with clock variables in C, variables in V and constants. Def is a set of value assignments to variables including reset of clock variables, which is represented as $\{x := x + 1, clock := 0\}$ for example, and a value assignment is executed when a state transition occurs.

We specify interaction and synchronization among timed EFSMs with the multi-way synchronization mechanism. The test scenario for the whole system S can be defined as follows.

$$S ::= S\ |[gate\_list]|\ S \quad | \quad S\ |||\ S \quad | \quad (S) \quad | \quad EFSM$$

Here, $EFSM$ is a name of a timed EFSM, and $|[gate\_list]|$ is the synchronous parallel operator where $gate\_list$ is a gate list of the events to be synchronized between its operator's both sides of timed EFSMs. The operator $|||$ is the asynchronous parallel operator, and it denotes that its operator's both sides of timed EFSMs can run in parallel without any synchronization. Those parallel operators can be used recursively.

2.3.1 Test Scenario for Object Playback Functions

We suppose an IUT which plays back media objects such as audio and video data. We specify a test scenario to test a playback function of the IUT with two timed EFSMs: S_T and S_Q, which correspond to the traffic testing scenario and the quality testing scenario, respectively. The whole test scenario $Player$ is given as follows.

$$Player := S_T|||S_Q$$

S_T specifies traffic characteristics which the IUT receives, and S_Q specifies quality constraints about frame displaying time which the IUT should satisfy. Note that these scenarios test given IUTs from external environments.

In S_T, we explicitly specify an allowable time interval between subsequent packets which the IUT receives. Here, for applicability to various network environments, we use four parameters: (1) transmission rate $AvRT$ of a media object, (2) maximum burst length $Burst$, (3) maximum packet loss ratio $Loss$, and (4) maximum jitter in packet arrival time JT. For the sake of simplicity, we assume that all packets have the same size and denote this by $PctSz$. We show example description of S_T in Fig. 4 where a double circle means the initial state.

In Fig. 4, clk and $TP(q)$ represent a clock variable and a time interval by which a server transmits each packet to realize playback quality q of a media object, respectively. Here, $TP(q) = PctSz/AvRT$. ln and pn denote the numbers of lost packets and transmitted packets during a time interval MP, respectively. There are four branches in S_T : (1) normal mode: each packet is transmitted to gate $n(n!pct)$ in a time range of maximum jitter JT, that is, $TP(q) - JT \leq clk \leq TP(q) + JT$; (2) burst transmission: when going into burst transmission mode ($burst\_start$), each packet is transmitted in smaller interval, that is, $clk < TP(q) - JT$ while the totally transmitted data size does not exceed $Burst$. When burst transmission finishes, the current state returns to a normal mode ($burst\_end$); (3) packet loss: each packet can be lost if packet loss ratio $((ln+1)/(pn+ln+1))$ measured during time interval MP is less than $Loss$; and (4) initialization: pn, ln and $burst$ are initialized to 0 ($reset$) every time interval MP. As we will explain in Sect.2.4, by executing these choices repeatedly in an appropriate probability we can test the IUT in the environment including jitter, loss and burst to receive packets.

Similarly, we give S_Q on playback quality of frames. For S_Q, we use two parameters FJT and MD, where FJT specifies the maximum value of fluctuation on the time interval of subsequent frames while a media object is played back, and MD specifies the maximum time skew representing how long the current frame can be delayed or preceding from the expected displaying time of the frame (e.g., 1000th frame in 30 fps video is expected to be displayed at 1000×33ms=33sec after the video started). We show an example description of S_Q in Fig. 5, where $TF(q)$ indicates the standard time interval between two subsequent frames with quality q, and vn and sn indicate the numbers of displayed frames and skipped frames measured during time interval MP, respectively.

There are three branches in S_Q of Fig. 5: (1) frame display: when the current time (clk) is within the appropriate time interval ($TF(q) - FJT \leq clk \leq TF(q) + FJT$) and the current frame ($sn + vn + 1$-th frame) is not too much delayed or preceding from the expected time ($(sn + vn + 1) \cdot TF(q) - MD \leq clk2 \leq$

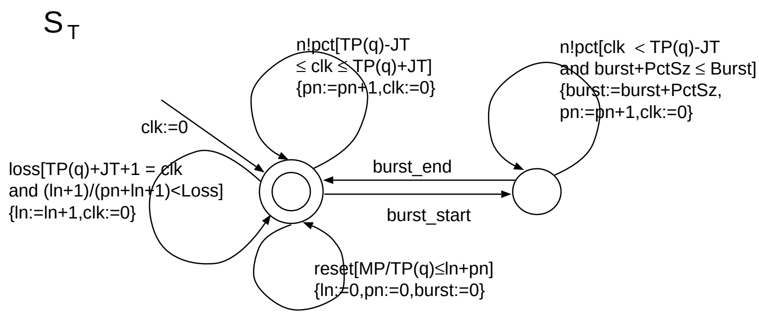


Figure 4. Traffic Testing Scenario

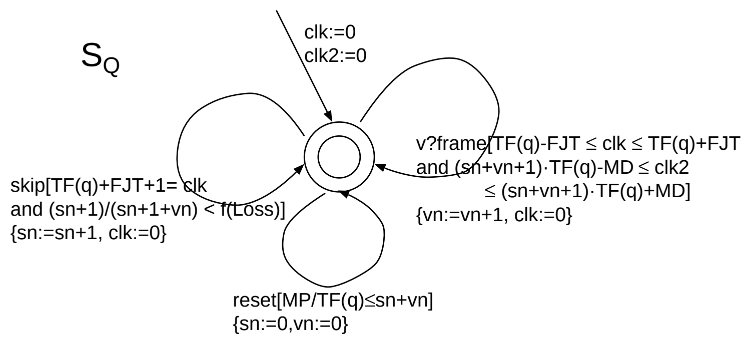


Figure 5. Quality Testing Scenario

$(sn + vn + 1) \cdot TF(q) + MD)$, a frame is allowed to be displayed $(v?frame)$; (2) frame skipping: when the current time exceeds the allowable time interval without displaying any frames and the frame skipping rate $(sn+1)/(sn+vn+1)$ measured during MP is less than $f(Loss)$ defined in Sect. 2.2.2, a frame is allowed to be skipped $(skip[TF(q) + FJT + 1 = clk])$; and (3) initialization: we initializes variables sn and vn to 0 every time interval MP.

2.3.2 Scenario for Testing Lip-Synchronization among Multiple Objects

We describe a sub-scenario for testing lip-synchronization among multiple media objects as a constraint among multiple quality testing scenarios for those objects,

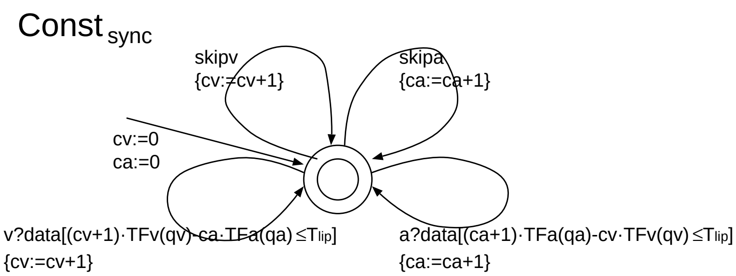


Figure 6. Constraint for Lip Synchronization

using the constraint oriented description style[13].

For example, let $Player[n_v, v, skip_v]$ and $Player[n_a, a, skip_a]$ be quality testing scenarios for video playback and for audio playback, respectively, and we assume that these two scenarios are executed independently in parallel. We also denote n_v, n_a and v, a to be input gate names and output gate names of video and audio, respectively. $skip$ behaviors for video and audio are distinguished by $skip_v$ and $skip_a$. Moreover, we let c_v and c_a denote the sequence numbers of frames of video and audio frames, and $TF_v(q_v)$ and $TF_a(q_a)$ denote the playback interval of video and audio frames, respectively. Here, we describe the sub-scenario $Const_{sync}$ such that the maximum time skew between video and audio must be within T_{lip}msec as shown in Fig. 6. In general, T_{lip} should be less than 80msec. In Fig. 6, it is explicitly described that each output of a video frame $v?$ (output of an audio frame $a?$) is allowed only if the time skew between the expected time of the current frame and that of audio (video) is kept less than T_{lip} msec.

The test scenario for the whole system including the lip-synchronization constraint is given by the following constraint oriented description.

$$(Player[n_v, v, skip_v]|||Player[n_a, a, skip_a])|[v, a, skip_v, skip_a]|Const_{sync}$$

2.4 Test Case Generation

We derive test sequences from the traffic testing scenario, the quality testing scenario, and the constraint for lip synchronization, explained in Sect. 3.2. Hereafter, we denote each test sequence as the following sequence $Tseq$.

$$Tseq := a.Tseq | Tseq +_p Tseq | (Tseq) | Tseq^{*K}$$

Here, $a.Tseq, Tseq +_p Tseq$ and $Tseq^{*K}$ denote sequential execution of actions, choice between two sequences and iterative execution of the sequence, respectively. $Tseq1 +_p Tseq2$ specifies that $Tseq1$ and $Tseq2$ are executed at probability $1 - p$ and p, respectively. In $Tseq^{*K}$, K denotes the number of iterations. If these values are not specified, default values such as $p = 0.5$ and $K = 100$ are used when executing test sequences.

2.4.1 Test Case Generation for a Single Object Playback

In the proposed method, the test system observes time at which each frame is displayed in the IUT, takes a sampling of a frame rate every time interval SP, and reports the test result by calculating ratio of samplings with low frame rate below a threshold in a normal distribution of all samplings as explained in Sect. 2.2.1.

Consequently, from test scenarios S_T and S_Q, we derive test sequences $Test_T$ and $Test_Q$ by adding new sequences for collecting a sampling every time interval SP and for test verdict computation when monitoring period LP expires, and by fixing the probability of choice "+" and the number of iteration "*" in the sequences. We show examples of $Test_T$ and $Test_Q$ in Table 1 and Table 2, respectively.

In Table 1, $Open()$ and $Read()$ denote some file operation primitives. $Packet()$ denotes a primitive to create a packet. In Table 2 $Sampling(x), CalcStatistics$ and $Judgesult$ are primitives which record x as a sampling, calculates statistical information, and calculates test verdict, respectively.

Derived test sequences $Test_T$ and $Test_Q$ must be executed in parallel for an IUT. Since each action in those sequences can be executed at any time instance within a specified time interval, the number of possible action sequences for their parallel composition will be so many. Therefore, we do not serialize those sequences. Instead, we make the test system capable of executing multiple sequences in parallel.

Table 1. Example Test Sequence $Test_T$ for Testing Input Traffic

$$
\begin{array}{l}
Test_T := \\
\quad \{fp := Open(file)\}. \\
\quad \{clk := 0, Loss := 0.0, ln := pn := burst := 0\}. \\
\quad (\{pct := Packet(Read(fp))\}. \\
\quad\; n!pct[TP(q) - JT \leq clk \leq TP(q) + JT]\{pn := pn + 1, clk := 0\} \\
\quad +[MP/TP(q) \leq pn + ln]\{pn := ln := 0\}. \\
\quad +([TP(q) + JT + 1 = clk \ and \ (ln + 1)/(pn + ln + 1) \leq Loss] \\
\qquad \{ln := ln + 1, clk := 0\} \\
\quad\; +(n!pct[clk < TP(q) - JT \ and \ burst + PctSz \leq Burst] \\
\qquad \{burst := burst + PctSz, pn := pn + 1, clk := 0\} \\
\qquad)^{*(Burst/PctSz)} \\
\quad\;) \\
\quad)^{*(LP/TP(q))}
\end{array}
$$

Table 2. Example of Test Sequence $Test_Q$ for Testing Playback Quality

$$
\begin{array}{l}
Test_Q := \\
\quad \{clk2 := 0\}. \\
\quad\; (\{clk := 0, vn := sn := 0\}. \\
\qquad (v?frame[TF(q) - FJT \leq clk < TF(q) + FJT \ and \\
\qquad\qquad (sn + vn + 1) \cdot TF(q) - MD \leq clk2 \leq (sn + vn + 1) \cdot TF(q) + MD] \\
\qquad \{vn := vn + 1, clk := 0\} \\
\qquad +skip[TF(q) + FJT + 1 = clk]\{sn := sn + 1, clk := 0\} \\
\qquad +[MP/TF(q) \leq sn + vn]\{sn := vn := 0\}. \\
\qquad)^{*(SP/TF(q))} \\
\qquad .Sampling(vn/SP)\{vn := 0, sn := 0\} \\
\quad\;)^{*(LP/SP)} \\
\quad\; .CalcStatistics.JudgeResult
\end{array}
$$

Table 3. Example Test Sequence $Test_{sync}$ for Lip Synchronization

$$
\begin{aligned}
&Test_{sync} := \\
&\quad \{c_a = c_v = 0\}. \\
&\quad (v?data\{c_v := c_v + 1\}.Sampling((c_v + 1) \cdot TF_v(q_v) - c_a \cdot TF_a(q_a)) \\
&\quad +skip_v\{c_v := c_v + 1\} \\
&\quad +a?data\{c_a := c_a + 1\}.Sampling((c_a + 1) \cdot TF_a(q_a) - c_v \cdot TF_v(q_v)) \\
&\quad +skip_a\{c_a := c_a + 1\} \\
&\quad)^{*(LP/Min(TF_v(q_v),TF_a(q_a))} \\
&\quad .CalcStatistics.JudgeResult
\end{aligned}
$$

2.4.2 Test Case Generation for Lip-synchronization among Multiple Objects Playback

In the proposed method, a test of the lip-synchronization among multiple playbacks of different media objects is similarly carried out using the statistical method stated in Sect. 2.2.1, where a time lag of the latest frames between two objects are collected as samplings. From test scenario $Const_{sync}$ in Sect. 2.3.2, we can derive a test sequence $Test_{sync}$ as shown in Table 3.

Let us denote $Test_v$ and $Test_a$ be test sequences for testing playbacks of video and audio objects, respectively. Here, $Test_v := (Test_{T_v} ||| Test_{Q_v})$. With $Test_v$, $Test_a$ and $Test_{sync}$, we finally obtain the following test sequence for testing a given IUT w.r.t. lip synchronization and playback qualities.

$$(Test_v ||| Test_a)|[v, a, skip_v, skip_a]|Test_{sync}$$

Since the above test sequence contains parallel and synchronization behaviors, we have to implement a test system which provides parallel and synchronous execution of multiple test sequences. Details of a test system are given in Sect. 2.5.

2.5 Implementation of Test System

A test system consists of a given IUT and a program which executes test sequences derived in Sect. 2.4. We call the program as a *tester*. We would like to treat a given IUT as a black box. So, we make the IUT and its tester run in parallel as different processes, and make them communicate with each other.

2.5.1 Implementation of Tester Program

Test sequences which we derived in Sect. 2.4 have the following characteristics.

(1) each action in a sequence specifies a time range during which it can be executed. The exact execution time is not specified.

(2) a probability is specified in each choice "+" between two sub-sequences. The number of iterative execution of a sub-sequence is also specified.

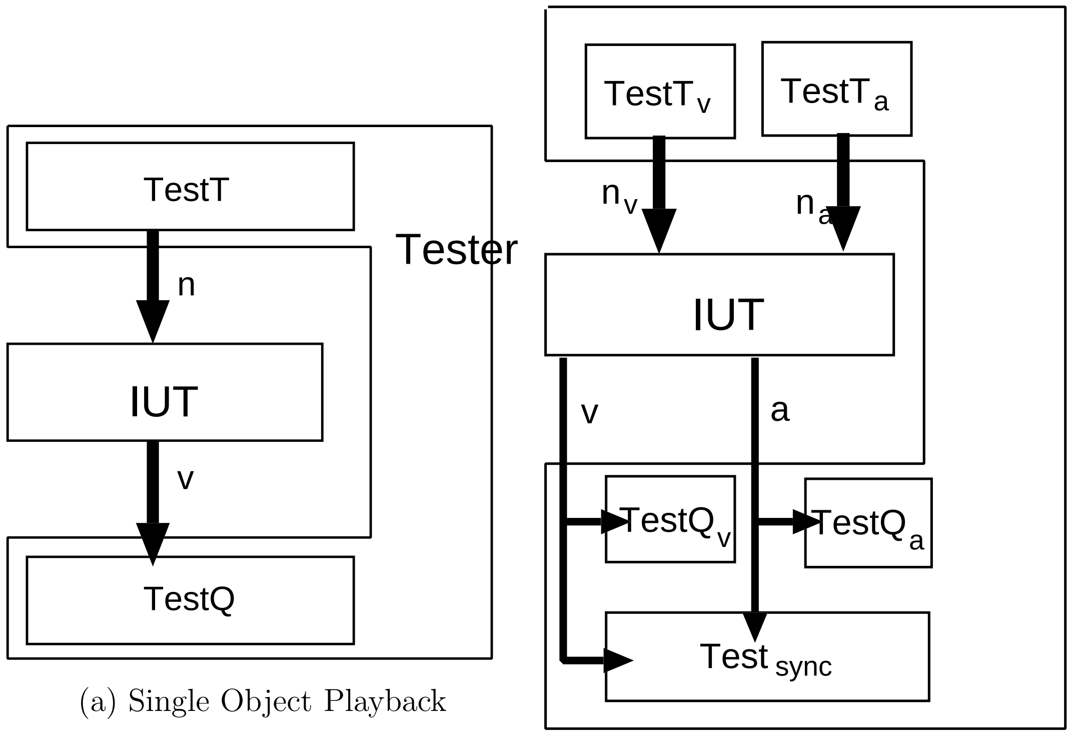
(3) for testing an object playback, two sequences are specified to be executed in parallel.

(4) for lip-synchronization among multiple object playbacks, a constraint sequence is also specified to be executed synchronously with multiple test sequences for those objects.

(5) statistical calculation primitives such as Sampling(), CalcStatistics(), JudgeResult() are contained in test sequences.

We have implemented a tester program which satisfies the above requirements in Java language. Since test sequences are given in the syntax defined in Sect. 2.4, first we have developed a parser program using JavaCC. Based on some techniques used in our real-time LOTOS compiler [17], we have implemented parallel execution and synchronization mechanisms for multiple test sequences.

For the above (1), it is desirable to test if the IUT works correctly for all time instances within the specified time range of each action in a given test sequence. However, it is impossible since combination of time instances in multiple actions will be infinite (in case of dense time). So, we have just implemented our tester only to select a time instance at random within the specified range if the next action in the test sequence is an output to the IUT. If the next action is an input from the IUT, the tester measures the clock value based on the current system time and checks whether or not the action has been executed within the specified time range. If so, the tester continues. Otherwise, it stops to report that the



(a) Single Object Playback

(b) Lip Synchronization between Two Objects

Figure 7. Test Environment

test has failed. For the above (2), the tester just selects one of branching sub-sequences based on random numbers, and repeats the specified sub-sequence the specified times, respectively. To improve the validity of the test result in (1), we can increase the number of iterations[1] . For the above (5), we have implemented statistical calculation primitives in Java, according to techniques explained in Sect. 2.2.1.

2.5.2 Construction of Test Environment

The tester must be able to observe the time when each frame is output in the IUT. Here, we assume that a certain event occurs and the time is notified to the tester via gate v when a frame is output in the IUT. The test sequence for a single

[1] In [4, 12], time instances near the borders of time ranges are intensively selected for tests. We can also adopt the similar technique to improve the test validity.

object playback derived in Sect. 2.4 contains parallel execution of two sub test sequences: $Test_T$ and $Test_Q$. In this case, the whole test system is executed as shown in Fig. 7 (a). The test sequence for lip synchronization between two objects contains four sub-test sequences (two for each) and one constraint $Test_{sync}$. These sequences are executed as shown in Fig. 7 (b). In order to reduce the influence coming from the test environment such as additional delay and jitter, we execute both the tester and the IUT in the same computer or in the separate computers in the LAN.

2.6 Experimental Results

With the test system explained in Sect. 2.5, we have executed test sequences and a given IUT and measured distribution of actual frame rates.

Our test purpose is to investigate whether a given IUT works satisfying QoS functions (specified by ideal frame rate fps', maximum tolerance acceptable ϵ, and reliance level r of area below ϵ in the standard normal distribution) when the IUT receives packets according to the specified traffic pattern (given by packet loss ratio $Loss$, maximum jitter JT in packet arrival time, and maximum burst length $Burst$).

Two kinds of programs have been used as IUT in our experiments: IUT_a which decodes and displays frames immediately after receiving packets (not regulate frame rates); and IUT_b which receives packets and stores them in a given buffer for absorbing jitters of the packet arrival time in order to display frames at the specified frame rate. We have implemented those IUTs in JMF2.1.1c(Java Media Framework) [18].

With the test system explained in Sect.2.5, by changing the above parameters given to IUT_a and IUT_b, we have examined the quality of playback mechanism in those IUTs (Exp1 to Exp 10). The list of parameters is shown in Table 4.

We have used a motion JPEG stream (320×240pixels, 25fps, the size of each frame is 5.88kbit) in all of our experiments and have applied SP=1s, MP=60s, LP=1800s and TP=40ms (i.e, each frame is transmitted by one packet) to the IUT. In the experiments we have measured the distribution of frame rates, by which we can decide whether the IUT passes the test or not.

First, we have executed Exp1 to Exp5 for the IUT_a. In the Exp1 and Exp2 we

Table 4. Parameters used in Experiments

	JT	$Loss$	$Burst$(bit)
Exp1	30ms	0.0	0
Exp2	10ms	0.0	0
Exp3	30ms	0.1	0
Exp4	30ms	0.1	58800
Exp5	30ms	0.2	58800
Exp6	30ms	0.0	0
Exp7	30ms	0.0	58800
Exp8	30ms	0.1	0
Exp9	30ms	0.2	0
Exp10	30ms	0.2	58800

Table 5. Test Results

		average	standard deviation	given r	given ϵ	ratio under ϵ	test result
	Exp1	24.6	2.65	2%	25.0	4.2%	failed
	Exp2	24.8	1.71	2%	25.0	0.0%	passed
IUT_a	Exp3	22.0	3.50	2%	22.5	12.3%	failed
	Exp4	19.6	4.36	2%	22.5	21.0%	failed
	Exp5	20.1	5.03	2%	20.0	20.3%	failed
	Exp6	25.0	—	2%	25.0	0.0%	passed
	Exp7	25.0	—	2%	22.5	0.0%	passed
IUT_b	Exp8	22.5	4.23	2%	18.0	1.8%	passed
	Exp9	20.0	4.86	2%	16.0	16.6%	failed
	Exp10	19.7	3.80	2%	16.0	20.3%	failed

have investigated the difference in the distribution of the frame rate by changing the value of JT. From Exp3 to Exp5 we have executed test sequences which contains jitters, packet losses (Exp3, Exp4) and burst transmission (Exp5) to the IUT_a. The distribution of samplings for these experiments are shown in Fig. 8 and Fig. 9. These figures show that samplings are distributed in the wide range where the center is around $fps' = 25 \times (1 - Loss)$, and the distribution range of samplings is influenced by each of jitters, packet losses, and burst transmission.

Then, we have executed Exp6 to Exp10 for the IUT_b which has a buffer and can control the frame rate. In Exp6 and Exp7, we have executed the test sequence only containing jitters, and the test sequence containing both jitters and burst transmission, respectively. In Exp8 and Exp9, we have executed the test sequence containing jitters and packet losses, where the value of $Loss$ is bigger in Exp9 than in Exp8. In the last experiment Exp10, the test sequence contain all factors: jitters, packet losses and burst transmission. The distribution of these experiments are shown in Fig. 10 and Fig. 11. In Fig. 10, we see that all of samplings concentrate in $25(fps')$ or $24(fps' - 1)$ because Jitters were absorbed. On the other hand, if we compare Exp8 with Exp9 in Fig. 11, we see that the distributed range of samplings is wider when the value of $Loss$ is higher. We think that this is because we specify $Loss$ as the maximum packet loss ratio for period MP=60s, but the actual packet loss ratio measured during short period (e.g., SP) may be more or less than the value of $Loss$. In Fig. 11 comparing Exp8 with Exp9, we see that the influence due to burst transmission is small.

Next, with the method explained in Sect.2.2.1 , we have decided whether tests pass or not. We calculated the average and standard deviation of samplings for each experiment by supposing their normal distribution. Here, for example we set *maximum tolerance acceptable* ϵ as $fps' \pm 20\%$ and *reliance level* r as 2%. The list of the average, the standard deviation, the ratio of area under ϵ and the test result for all experiments are shown in table 5. According to the table, we see that the IUT_a passes the test in the condition which JT was within 10ms and $Loss$ was close to 0 (Exp2). About IUT_b, it passes the test when JT was within 30ms and Loss was within 0.1 (Exp6, 7, 8).

In order to evaluate the validity of our assumption which the distribution of frame rates follow the normal distribution, we calculated the proportion of the

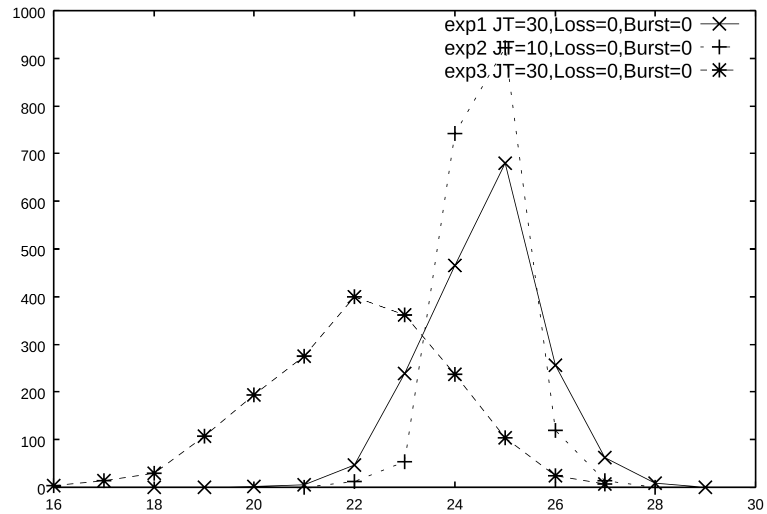


Figure 8. Distribution for Experiments 1,2,3

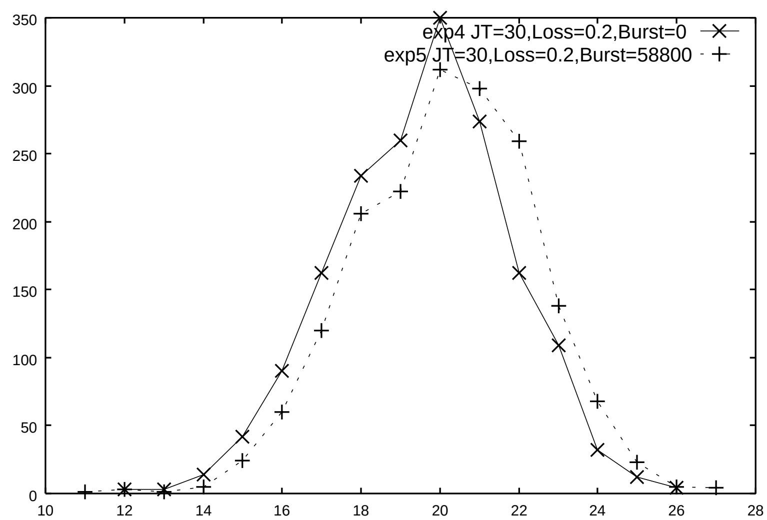


Figure 9. Distribution for Experiments 4,5

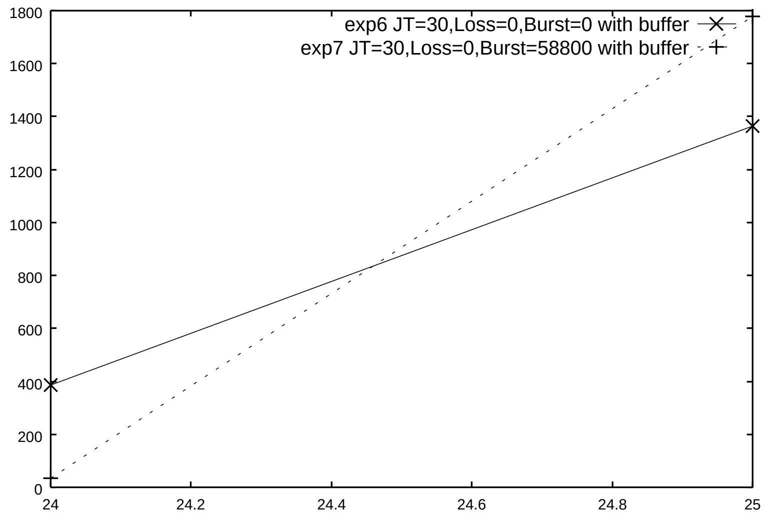


Figure 10. Distribution for Experiments 6,7

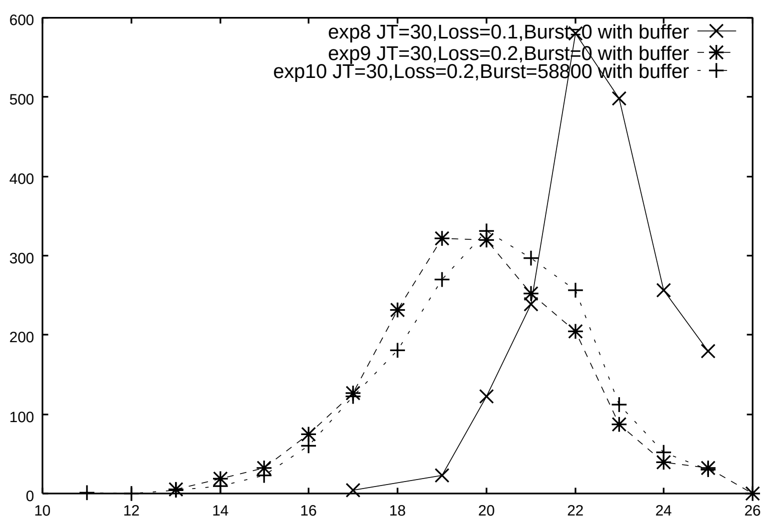


Figure 11. Distribution for Experiments 8,9,10

Table 6. Comparison between Normal Distribution and Distribution of Actual Samplings

	proportion below(beyond) ϵ	
frame rate	normal distribution	actual samplings
16	0.04182	0.002277
17	0.07493	0.010245
18	0.12302	0.026750
19	0.19215	0.088218
20	0.27759	0.198633
21	0.38209	0.355150
22	0.49601	0.582242
23	0.39358	0.417758
24	0.28774	0.211725
25	0.20045	0.076836
26	0.12924	0.018213
27	0.07927	0.003984

samplings under (or upper when $\epsilon > fps'$) ϵ to the whole of samplings and the proportion of C $[-\infty, (\epsilon - \mu)/s]$ to the whole area in the normal distribution. We show the result in Table 6. Comparing the normal distribution and the distribution of the actual samplings, we see that the proportion in the normal distribution is in most cases bigger than in the actual samplings for each ϵ. If the test passes on the proposed method, then the actual proportion under ϵ was not bigger than reliance level r. So we think our assumption is valid enough.

Test for Lip Synchronization

Next, we have executed Exp11 to Exp13 for testing lip synchronization when multiple object are played back in parallel.

In this experiment, the programs which can play back two videos in parallel have been used as IUT_c [2] . IUT_c has a buffer and can control the frame rate

[2] Althoughth we only observed the deviation of playback between two videos in experiment

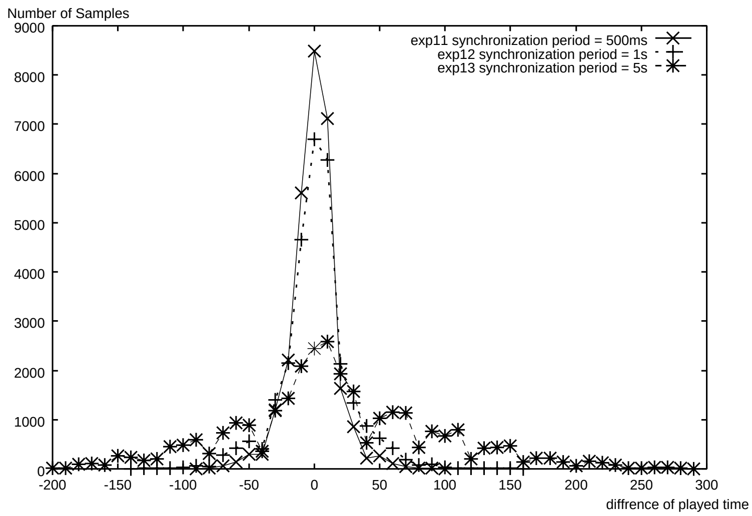


Figure 12. Distribution of Lip Synchronization for Experiments 11,12,13

Table 7. Lip Synchronization Test Results

	average	standard deviation	ϵ	ratio under ϵ	test result
Exp11(500ms)	-0.520	12.842	40	0.082%	passed
Exp12(1000ms)	1.279	16.141	40	0.84%	passed
Exp13(5000ms)	16.057	20.815	40	12.5%	failed

similarly to IUT_b, and can make two objects synchronize every SP ms. We have executed Exp11-13 using IUT_c which executes two test sequences in parallel and SP is changed among 500ms, 1000ms and 5000ms. The distribution of samplings corresponding to time skew between two objects in these experiments are shown in Fig. 12. In Fig. 12, we see that samplings are distributed in the wide range where the center is a round 0, and when the synchronization period is smaller, more sampling are near the center.

Next, using the method explained in Sect.2.2.1, we have decided whether tests pass or not. We calculated the average and standard deviation of samplings for each experiment by supposing their normal distribution. Here, for example we set ϵ as $40ms$ and *reliance level* r as 2%. The list of the average, the standard deviation, the ratio of area under ϵ and the test result for all experiments are shown in table 7. According to the table, we see that the IUT_c passes the test in the condition that SP is less than 1000 ms.

2.7 Conclusion

In this Chapter, we proposed a test method for QoS functions in distributed multi-media systems. In the proposed method, using timed EFSM model, we describe a test scenario which specifies input flow characteristics and play-back quality to be realized for the flow, and generate test sequences from the scenario. Using the generated test sequences, we can statistically test whether a given IUT realizes certain quality for a given input flow. Through experiments with our test system and sample IUTs, it is confirmed that the proposed test method can efficiently test given IUTs depending on target network environments and required playback quality.

---

for the sake of simplicity, the testing for deviation between audio can be executed as the same method.

3. Energy-aware Video Streaming System with QoS Adaptation Based on Inter-segment Importance

3.1 Introduction

In this Chapter, we propose a QoS adaptation method for streaming video playback for portable computing devices where playback quality of each video fragment is automatically adjusted from the remaining battery amount, desirable playback duration and the user's preference to each fragment. In our method, we assume that video segments (shots) of a video are classified into some predefined categories in advance. For example, video segments of a soccer game can be classified into categories: *shoot*, *normal-play*, *set-play*, *audience*, *other*, etc. These categories are described as meta information in MPEG-7 format. Classification can be done manually using annotation tools like [30], or done automatically using tools like [31]. Next, a user specifies priorities among categories. For each category, the user also specifies relative importance among playback parameters such as motion speed, vividness and sound.

From the information above, playback quality/property for each category are determined so that the video playback can last for the specified duration within the battery amount. In our previous work [33], we have proposed a method to determine fixed playback parameter values where a video can be played back for the specified duration with the fixed quality. In this thesis, we enhance this algorithm so that the battery amount can be allocated to categories according to the specified priority and the playback property of each category is determined based on the specified preference.

We have implemented a video streaming system consisting of a transcoder which converts a video stream from a contents server to a new stream with any specified parameters, and a video player which can be executed on PDAs. From some experiments using our system, we have confirmed that the playback quality of important categories can be improved a few times better than flattening the playback quality over the playback duration, and that the prediction error of the playback duration is within 5%.

In previous transcoding techniques which simply reduce the picture size, objects in each picture frame become too small and difficult to identify. [29] copes with this problem by specifying the user's interesting area in the picture with the MPEG-21 DIA framework so that only the area is trimmed off and transcoded. In [28], a video in MPEG-4 format is divided into objects of several categories such as foreground objects and background objects. Here, playback qualities of important objects are maintained while qualities of other objects are lowered.

The objectives of these existing researches are to satisfy restrictions of portable devices w.r.t. picture size and available bandwidth. However, we believe that the restriction w.r.t. the battery amount and the playback property of each fragment are also important. These points are new in our approach.

3.2 Describing Meta Data and Priorities

MPEG-7 has been standardized by ISO/IEC as a description method of meta information for multimedia contents encoded in MPEG-1, 2 and 4. In MPEG-7, meta data can be specified for any fragment of a video in order to facilitate users to search a specified fragment by its "feature data". These meta data of a video are described in a XML file.

3.2.1 Specifying feature data to each video segment

We use a keyword called *category* as feature data, and denote a set of categories by $C = \{c_1, \cdots, c_n\}$. Here, a category c_i is specified by a string. For example, for the video of a soccer game, we may use a set of categories $C = \{shoot, play, audience, other\}$. A fragment in a video taken by the same camera work is called a *shot* or *segment*. In this thesis, we suppose that a category $c_i \in C$ is assigned to each segment.

In general, MPEG files do not contain the boundary information of each segment. The tool named VideoAnnEx (IBM MPEG-7 Annotation Tool) [30] can read a MPEG1 file, identify each video segment automatically, assign a string to each segment, and output an MPEG-7 file as shown below.

```
<VideoSegment>
  <TextAnnotation>
```

```
      <FreeTextAnnotation> shoot
      </FreeTextAnnotation>
   </TextAnnotation>
   <MediaTime>
      <MediaTimePoint> T00:00:00:0F25
      </MediaTimePoint>
      <MediaIncrDuration mediaTimeUnit
         ="PT1N25F"> 78
      </MediaIncrDuration>
   </MediaTime>
</VideoSegment>
```

In the above file, string *shoot* is specified to a video segment as a category using tag `<TextAnnotation>`. Tag `<MediaTime>` describes the starting time and the duration of this segment.

3.2.2 Specifying importance among categories

It is desirable for users to be able to specify what part of a video will be played back with higher quality. So, we allow users to specify relative importance among categories as priority values. Let p_i denote the priority specified to category c_i where p_i is an integer number such that $p_i \geq 1$.

The *playback property* of a video is decided by the balance of its picture size, frame rate and bitrate. In general, users have different preferences for the playback property of each category. Also, there may be various combinations of properties which consume the same electric power. So, we allow users to specify a preference to the property of each category by relative importance among three factors: motion speed, vividness and sound. We denote these factors by spd_i, vid_i and snd_i for category c_i. An integer number between 0 and m (m is a constant such as 3) can be specified to each factor.

For example, in a video of a soccer game, suppose that sound is not very important in all categories, that both the motion speed and the vividness are very important in category *shoot*, that only the motion speed is somewhat important in category *play*, and that only the vividness is somewhat important in categories *audience* and *other*. In such a case, users give the following preference.

category	spd	vid	snd
shoot	3	3	1
play	2	1	1
audience	1	2	1
other	1	2	1

3.3 Algorithm for Deciding Playback Quality

3.3.1 Battery distribution among categories

Let us denote battery amount of a portable computing device by E_0, and the desirable playback duration of a video by T. We denote by w_0 the power consumed while no video is played back (i.e., the power consumed by the operating system, the back-light for LCD, and so on). Thus, the battery required for playback of a video with duration T is denoted by $E = E_0 - w_0 T$. Here, we can easily measure the actual value of w_0 for any device.

For each category $c_i \in C$, the product of its importance and playback duration is called the *virtual playback time* of c_i. We denote it by $T'_i(= p_i T_i)$. Also, the total sum of the virtual time of all categories is denoted by $T'(= \sum_{c_i \in C} T'_i)$.

In our algorithm, we distribute the remaining battery amount E among categories according to the proportion of the virtual time T'_i/T' of each category. That is, $E_i(= ET'_i/T')$ is allocated for playback of each category c_i.

The property of each video is represented by picture size r, frame rate f and bitrate b. We denote it by (r, f, b). We denote the properties of videos with the maximum quality and with the minimum quality by $(r_{max}, f_{max}, b_{max})$ and $(r_{min}, f_{min}, b_{min})$, respectively. Here, the video with the maximum quality might be the one with satisfactory quality or the maximum one which the device can play back without changing its property. The video with the minimum quality can similarly be defined.

In [33], we have confirmed that the battery amount E consumed by video playback is approximately proportional to the product of picture size r, frame rate f, bitrate b and playback duration T. That is, $E = \alpha r f b T$. Here, α is a device specific constant and can be measured for any device using our technique in [33].

Due to this fact, if $E_i > \alpha r_{max} f_{max} b_{max} T_i$, E_i is too much for playback of video segments in c_i. Similarly, if $E_i < \alpha r_{min} f_{min} b_{min} T_i$, E_i is too small for playing back segments in c_i. In either case, we fix $E_i = \alpha r_{max} f_{max} b_{max} T_i$ or $E_i = \alpha r_{min} f_{min} b_{min} T_i$, and distribute the remaining battery amount $E' (= E - E_i)$ among remaining categories $C - \{c_i\}$. Consequently, we can obtain battery amount E_i for playback of category c_i as a constant value.

3.3.2 Decision of each category's playback property

We would like to decide the playback property of each category c_i as picture size (i.e., number of pixels) r_i, frame rate f_i and bitrate b_i from battery E_i assigned for c_i, playback duration T_i and the user preference (spd_i, vid_i, snd_i) for playback property of c_i.

Here, it is considered that motion speed spd_i and vividness vid_i influence the frame rate and the picture size, respectively. On the other hand, bitrate b_i is influenced from all of spd_i, vid_i and snd_i. Therefore, the proportion of b_i will be $(vid_i + spd_i + snd_i)/3$. If we do not use sound (i.e., $snd_i = 0$), the proportion will be $(vid_i + spd_i)/2$. For the sake of simplicity, we suppose $snd_i = 0$, hereafter.

When playing back from storage

From user preference (spd_i, vid_i), we would like to decide playback property (r_i, f_i, b_i) such that $E_i = \alpha r_i f_i b_i T_i$. Since we cannot directly compare the ratio between the picture size, the frame rate and the bitrate, we use the ratio of each video parameter to the corresponding one of an original video (r_0, f_0, b_0) as follows.

$$\frac{r_i}{r_0} : \frac{f_i}{f_0} : \frac{b_i}{b_0} = vid_i : spd_i : \frac{vid_i + spd_i}{2}$$

From the above equation, we can derive $f_i = \frac{spd_i f_0}{vid_i r_0} r_i$ and $b_i = \frac{(spd_i + vid_i) b_0}{2 vid_i r_0} r_i$. When we assign these equations to formula $E_i = \alpha r_i f_i b_i T_i$,

$$E_i = \alpha \frac{spd_i (spd_i + vid_i) f_0 b_0}{2 vid_i^2 r_0^2} T_i r_i^3$$

is derived. Then, we can calculate the value of r_i as follows.

$$r_i = \sqrt[3]{\frac{2E_i vid_i^2 r_0^2}{\alpha T_i spd_i(spd_i + vid_i) f_0 b_0}}$$

Similarly, the values of f_i and b_i can be obtained as follows.

$$f_i = \sqrt[3]{\frac{2E_i spd_i^2 r_0^2}{\alpha T_i vid_i(spd_i + vid_i) r_0 b_0}} \quad b_i = \sqrt[3]{\frac{E_i(vid_i + spd_i)^2 b_0^2}{4\alpha T_i spd_i vid_i f_0 r_0}}$$

When playing back streaming video via wireless LAN

When playing back a video with bitrate b bps using IEEE 802.11b, the power consumed for communication by a portable computing device and a WNIC (wireless network interface card) can be approximated by the linear expression of bitrate b [33]. That is, $\beta + \gamma b$. Here, β and γ are device specific constants and can be measured for any portable devices and WNIC.

Our preliminary experiments using IEEE 802.11b have shown that β is much larger than γb. Owing to this fact, when available bandwidth is larger than b bps, we can vastly reduce battery consumption [33] by dividing a video (whose bitrate is b bps) to fragments with M bit, transmitting each fragment at B bps ($B > b$) every M/b seconds so that the portable device stores each received fragment in a local buffer, turns off its WNIC until the next transmission period comes, and plays back the fragment from the buffer. We call this scheme *buffered playback*. In the buffered playback, when transmitting each fragment at $k(= B/b)$ times of original bitrate b, the portable device can receive it in $1/k$ of the originally required time. So, the power supply to WNIC can be stopped during most of the playback time. However, actually, it takes a few seconds (denoted by $t_{on/off}$) to stop/resume WNIC during which some power (denoted by τ) is consumed.

In a video, there are some video segments (denoted by seg_i) which belong to category c_i. Total size of video segments in c_i is $b_i T_i$. When we divide it to M bit fragments, the total number of transmissions can be denoted by $\frac{b_i T_i}{M} + seg_i$ in the worst case. Practically, we can omit "$+seg_i$" from the expression.

Consequently, battery consumption for playing back c_i when using buffered playback, is represented by

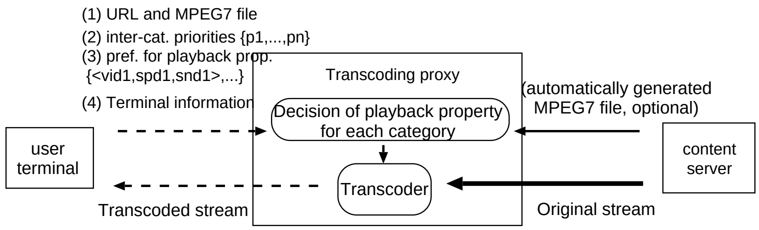


Figure 13. Energy-aware streaming system

$$E_i = \alpha r_i f_i b_i T_i + (\beta + \gamma B)\frac{b_i}{B}T_i + \frac{b_i T_i}{M}\tau t_{on/off}$$

By assigning equations $f_i = \frac{spd_i f_0}{vid_i r_0} r_i$ and $b_i = \frac{(spd_i + vid_i) b_0}{2 vid_i r_0} r_i$, we can get the following equation of r_i.

$$E_i = \alpha \frac{spd_i(spd_i + vid_i) f_0 b_0}{vid_i^2 r_0^2} T_i r_i^3$$
$$+ (\frac{\beta + \gamma B}{B} + \frac{\tau t_{on/off}}{M}) \frac{(spd_i + vid_i) b_0}{2 vid_i r_0} T_i r_i$$

We can obtain the value of r_i from the above equation, for example, using Newton's method.

If either of the calculated values of r_i, f_i and b_i is larger/smaller than the maximum/minimum threshold, we can fix the parameter value, and re-calculate the values of the other parameters. For example, the value of r_i is larger than portable device's screen size r_{max}, we fix r_i to r_{max} and re-calculate f_i and b_i using the algorithm recursively.

3.4 Streaming System

We have implemented a video streaming system consisting of a movie player and a transcoding proxy as shown in Fig.13. The transcoding proxy is supposed to be executed at a contents server or at an intermediate node on the network. Each user sends (1) a video's URL with the desirable playback duration and its MPEG-7 file, (2) priorities among categories, (3) a preference to the playback property for each category and (4) the device specific terminal information (values of E, α, β, γ, etc) to the transcoding proxy. The transcoding proxy transcodes a video stream transmitted from a contents server to a new stream with the playback quality and property calculated by the algorithm described in Sect. 4.3.3, and relays the stream to the portable computing device.

3.5 Experimental Results and Evaluation

Playback quality in important categories

Using the algorithm in Sect. 4.3.3, we have investigated to what extent the playback quality of important categories is improved and the quality of the other categories is degraded.

In the experiment, we assume that video segments in a video are classified into two categories: important category c_1 and less-important category c_2.

Let R denote the ratio of playback duration T_1 of c_1 to total playback duration $T = T_1 + T_2$ (i.e., $R \overset{def}{=} T_1/(T_1 + T_2)$). Let p_1 and p_2 denote the priorities for c_1 and c_2, respectively. Let M denote the ratio of p_1 to p_2 (i.e., $M \overset{def}{=} p_1/p_2$) and we assume that $0.05 \leq R \leq 0.5$ and $1.5 \leq M \leq 4$.

We have observed variation of the playback qualities of video segments in c_1 and c_2 by changing R from 0.05 to 0.5 by 0.05 step and M in 1.5, 2, 3 and 4, respectively. The resulting graphs are depicted in Fig. 14, where the horizontal axis and the vertical axis represent R and playback quality Q, respectively. Since quality Q is defined as $\sqrt[3]{r_i f_i b_i / r_0 f_0 b_0}$, Q varies between 0 and 1, where (r_0, f_0, b_0) is the property of an original video before transcoding. Since we mainly focus on the use of PDAs, we set (r_0, f_0, b_0) to $(320 \times 240, 30fps, 700Kbps)$ in this experiment. Q becomes 0.41, if all categories have the same priorities, that is, $p_1 = p_2$. Fig. 14 shows that while R is less than 0.2, the playback quality in important categories can be improved significantly by a small reduction of the playback quality of less-important categories. Even when R is high (around 0.3), we can improve the quality of important categories much with about 20 % quality degradation in less-important categories, by controlling M under 2.

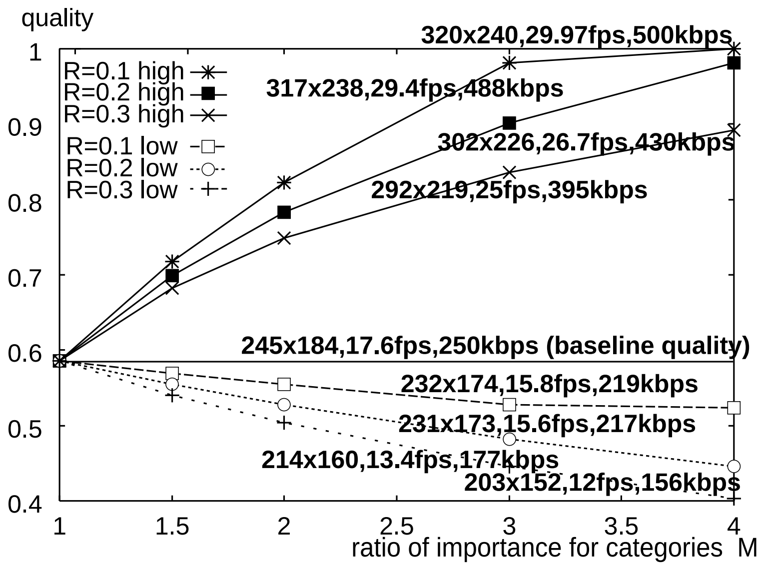


Figure 14. Quality improvement

Playback property among categories

In the proposed method, we can also change the playback property of each category according to the importance among categories. Here, using preferences of *pref1*, *pref2*, *pref3* and *pref4* in Table 8, playback properties of their categories were calculated using the algorithm in Sect. 4.3.3. The results are shown in Table 8.

The results show that the playback properties were changed by the ratio of importance among categories. For example, for $pref3$ and $pref4$, we set the same importance of category $c2$ as 2 and change the playback property of categories ($pref4$ spd_i: $vid_i = 2:1$, $pref3$ spd_i:$vid_i = 1:2$). We see that the frame rate of $pref4$ becomes larger than $pref3$ and the size becomes smaller.

Ratio of prediction error

We have measured actual playback durations of a video with 1800 sec. within the remaining battery using preferences in Table 8.

In the experiment, a PDA (SHARP, ZAURUS SL-C700) with an IEEE 802.11b

Table 8. Playback Qualities with property

	cat.	(T_i, p_i, spd_i, vid_i)	(r, f, b)
pref1	c_1	(678, 1, 1, 1)	$(245 \times 184, 17.60, 250K)$
	c_2	(662, 1, 1, 1)	$(245 \times 184, 17.60, 250K)$
	c_3	(460, 1, 1, 1)	$(245 \times 184, 17.60, 250K)$
pref2	c_1	(678, 1, 1, 1)	$(196 \times 147, 11.26, 143K)$
	c_2	(662, 2, 1, 1)	$(240 \times 180, 16.91, 238K)$
	c_3	(460, 4, 1, 1)	$(293 \times 219, 25.06, 395K)$
pref3	c_1	(678, 1, 1, 1)	$(196 \times 147, 11.26, 143K)$
	c_2	(662, 2, 1, 2)	$(292 \times 219, 12.47, 201K)$
	c_3	(460, 4, 2, 3)	$(320 \times 240, 21.61, 365K)$
pref4	c_1	(678, 1, 1, 1)	$(196 \times 147, 11.26, 143K)$
	c_2	(662, 2, 2, 1)	$(196 \times 147, 22.48, 288K)$
	c_3	(460, 4, 3, 2)	$(261 \times 196, 29.97, 439K)$

WLAN card (WN-B11/CF, I-O DATA Device, inc.) was used. A WLAN access point is connected to the LAN to which a PC executing a transcoding proxy is connected.

For *pref1* - *pref4* video segments in the video are played back with the playback qualities of their categories shown in Table 8. For *pref1*, *pref2*, *pref3* and *pref4*, actual playback durations were 1868sec,1850sec, 1820sec and 1849sec.

Evaluation

We have evaluated the impact of the proposed method by means of questionnaire. In the evaluation, we used a soccer video with 180 minutes and let four testers watch the video with dynamic QoS adaptation using *pref1* in Table 8 and that with the fixed quality (picture size of 230×172, $15.51fps$, 362Kbps) calculated by our previous algorithm in [33].

As a result, all of testers preferred the playback quality in important categories using the proposed method to the fixed playback quality using our previous method. Some of testers preferred larger picture size to larger frame rate. There are different opinions on the playback quality in less-important categories. Some said that the picture size is too small and the motion speed is too clumsy, others said no problem. Also, there is a comment that the sudden picture size change is a bit unnatural.

3.6 Conclusions

In this Chapter, we proposed an energy-aware QoS adaptation method for streaming video playback for portable computing devices, based on MPEG-7 meta information and priorities among segments in a video. We confirmed that on portable devices with limited battery amount, the user's feeling of satisfaction can be improved to some extent compared with flattening playback quality over the playback duration.

4. MTcast: A Multiple Transcoding Approach for Video Multicast

4.1 Introduction

In this chapter, we propose a new video delivery method called *MTcast (Multiple Transcode based video multicast)* which achieves efficient simultaneous video delivery to multiple heterogeneous users by relying on user nodes to transcode and forward video to other user nodes. In MTcast, each user specifies a quality requirement for a video consisting of bitrate, picture size and frame rate based on the user's environmental resource limitation. All users can receive video with specified quality (or near quality) along a single delivery tree. A different quality requirement can be specified to each time segment or to each video shot.

In order to achieve the proposed method, the followings should be considered: (1) *high scalability* for accommodating a large number of users, (2) *high user satisfaction degree* in the sense that the delivered quality is close to the required quality, (3) *small resource consumption* within available resource of each user node, (4) *short startup latency* to start playing back video quickly, (5) *reasonable number of transcoding times* for keeping good video quality as well as short delivery latency, and (6) *high robustness* for continuing video delivery service even with node/link failures.

In order to achieve the above (1) to (3), in the proposed method, we construct a delivery tree called *transcode tree* whose root is the sender of a video content. The transcode tree is constructed as a perfect n-ary tree, where user nodes with higher quality requirements are located near the root of the tree, and nodes with lower quality requirements are located far from the root. Nodes are placed in the appropriate places of the transcode tree according to their computation power, available downstream and upstream bandwidths. Each node in the tree receives a video stream, transcodes it to lower quality video in real time and forwards it to its children nodes. In order to achieve the above (4) to (6), nodes are grouped so that each group has k members with similar quality requirements. These groups are called *layers*. All nodes in a layer receives the video with the same quality from their parent nodes along the transcode tree. We let the representative node of each layer keep the complete information of the tree. This allows a new receiver to easily find the layer which has the closest quality to its own quality requirement and to quickly send a request to the node in the layer to start delivery of the video. In order to accommodate new receivers or to replace faulty nodes with normal ones, we made each layer to keep a certain amount of extra computation power and available upstream bandwidth (computed from the value of k). In general, if we use a large number for k, we can improve performance of the above (4) to (6). However, user satisfaction degree may be reduced since the received video quality is averaged over k members. So, in the proposed method, we adopted an approach to dynamically increase the value of k as the total number of receivers increases. When the number of receivers is sufficiently large, we can make both user satisfaction and system robustness high.

After certain time elapses, extra resources at a layer might have been exhausted. So, our method reconstructs the transcode tree periodically or at each time boundary between subsequent video shots. Video delivery requests and failures occurring after extra resources of a layer is exhausted, are processed during the next tree reconstruction.

We have investigated performance of MTcast by simulations using topologies generated by Inet3.0[44]. As a result, we have confirmed that MTcast can achieve both higher user satisfaction degree and higher robustness than the layered multicast method.

4.2 Basic Ideas

4.2.1 Target Environment and Goals

In this chapter, we deal with a method for simultaneously delivering a video content to multiple *heterogeneous users* who have different available bandwidth, different computation power, and different display resolutions. Here, we assume the following number of users, types of user terminals, types of communication infrastructures and target contents.

- user terminal: desktop PC, laptop PC, PDA, cellular phone, etc.

- communication infrastructure: Either fixed broadband (leased lines, ADSL, CATV, etc.) or wireless network (wireless LAN, W-CDMA, Bluetooth, GSM/PDC, etc).

- the total number of users: 500 to 100,000
- target contents: video (both recorded and live)

We assume a service which starts to transmit a video content to all receivers at the same starting time like TV broadcast. Even after the starting time of the video, users can start to receive the video anytime, but the video can be watched from the scene currently in transmission. The on-demand service in which users can watch favorite contents from beginning anytime is out of scope of this research.

Transmitting a video stream from a server to the several thousands of users directly is not realistic in terms of the required bandwidth and the server load. Here, we assume that user nodes are connected to each other through overlay links, and that each node uses overlay multicast to transmit/receive streams to/from the other node.

In order to achieve simultaneous video delivery to multiple heterogeneous users by the overlay multicast, it is necessary to build a multicast tree which satisfies the quality requirement and environmental restriction of each node. So, in the multicast tree, we let each user node except leaf nodes transcode a video stream and forward it to its children nodes as well as receive and play back the stream.

Although other existing methods such as the layered multicast method [39, 41], the multi-description coding [45, 46], and MPEG4-FGS[43] can change only bitrates, the transcode method can change multiple video parameters (picture size, bitrate, frame rate, encoding method, etc). Thus, it can treat user's various quality requirements such as keeping the picture size larger by reducing frame rate within the same bitrate.

From the above discussion, the main purpose would be to build and manage the multicast tree which satisfies criteria (1) to (6) in Sect. 4.1 and to devise the efficient video delivery method using the tree.

4.2.2 Basic Policy

In order to satisfy the criteria (1) to (6) in Sect. 4.1, the following four points are considered as basic policies in our method.

Firstly, playback quality of a node cannot exceed parent's quality even if the terminal has ability to play back the video in higher quality. Thus, if a terminal with poor computation power such as PDA is placed near the root of the multicast tree, video quality of all descendant nodes of the PDA becomes also poor. Accordingly, in order to satisfy as many users' quality requirements as possible, (1) we build the multicast tree so that the quality of video decreases gradually from the root node to the leaf nodes.

Secondly, video quality deteriorates if transcoding is repeatedly applied to a video. Thus, in order to reduce the number of transcoding times, (2) we build the multicast tree whose height is as short as possible.

Thirdly, since the required computation power for transcoding is relatively big, it is not realistic that a specific node transcodes multiple streams at the same time. So, (3) we control computation power required for transcoding within a certain threshold so that the service quality(i.e., playback quality) is not reduced by transcoding load.

Finally, we must consider node failure in overlay networks. So, (4) even if a node unexpectedly fails, we control the stream to be delivered smoothly to its children nodes by assigning an alternative parent node.

4.3 MTcast

4.3.1 Definitions

Let s denote a video server, and $U = \{u_1, ..., u_N\}$ denote a set of user nodes. We assume that for each $u_i \in U$, available upstream (i.e., node to network) bandwidth and downstream (i.e., network to node) bandwidth are known in advance. We denote them by $u_i.upper\_bw$ and $u_i.lower\_bw$, respectively. Let $u_i.q$ denote u_i's video quality requirement. In general, as $u_i.q$, multiple video parameters such as bitrate, picture size and frame rate are specified. Here, we assume that $u_i.q$

represents only bitrate of video[3] . Let $u_i.n_{trans}(q)$ denote the maximum number of simultaneous transcoding which can be executed by u_i for videos with quality q. Let $u_i.n_{link}(q)$ denote the maximum number of simultaneous forwarding of videos with quality q which can be performed by u_i. $u_i.n_{trans}(q)$ and $u_i.n_{link}(q)$ are calculated from computation power of u_i, $u_i.upper\_bw$ and video quality.

In the proposed method, we construct a multicast tree where s is a root node and user nodes in U are intermediate or a leaf nodes. Hereafter, this multicast tree is called the *transcode tree*. In a transcode tree, nodes with children nodes are called *internal nodes*, and nodes without children nodes are called *leaf nodes*.

4.3.2 Structure of Transcode Tree

Internal nodes in a transcode tree transmit a video stream to children nodes. It is not desirable for internal nodes to have too many children nodes since it would cause severe overload. In the proposed method, we assume that fanout (degree) of each node is basically a constant (denoted by n). As we will explain in Sect. 4.3.4, we decide the value of n depending on available resources of user nodes.

In order to reduce the delay time and the number of transcoding until each leaf node receives video, we construct the transcode tree as a slight modification of complete n-ary tree where degree of the root node is changed to k instead of n (k is a constant, and explained later). In a transcode tree, for each node $u_i \in U$ and each of its children nodes u_j, $u_i.q \geq u_j.q$ holds. That is, from the root node to each leaf node, nodes are ordered in decreasing order of quality requirements.

In order to tolerate node failures and to shorten startup delay to start video delivery, every k nodes in U are bunched up into one group. We call each group a *layer*, where k is a predetermined constant, as shown in Fig. 15. We let user nodes in the same layer receive video with the same quality. This quality is called the *layer quality*. A representative node is selected for each layer. Parent-child relationship among all layers on the transcode tree is called the *layer tree*. Since fanout of each node is n, each layer's fanout in the layer tree also becomes n.

An example of the transcode tree with $n = 2$ and $k = 6$ is shown in Fig. 15. Here, small circles and big ovals represent nodes and layers, respectively. Each

[3] By defining the total order among tuples of video parameters, the proposed method can be applied to multiple parameter cases.

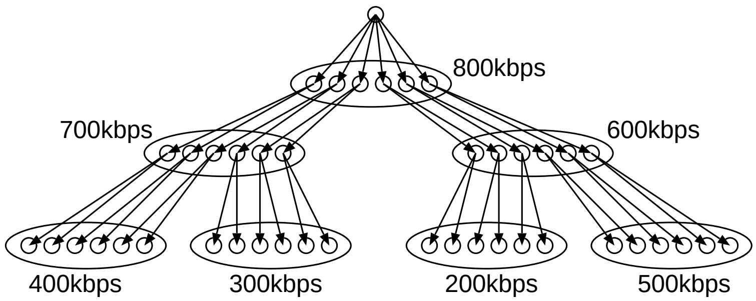


Figure 15. Example of Transcode Tree, where $n = 2, k = 6$

bitrate (e.g., 500kbps) represents the layer quality.

4.3.3 Construction of Transcode Tree

In our method, the transcode tree is calculated in a centralized way by a user node u_c or a server S. The way of deciding u_c is explained later. We assume that u_c has information of a video server s and user nodes $U' \subseteq U$ who have requested video. Our tree construction algorithm consists of the following three steps.

Let $A = \{s\} \cup U'$, denote the set of all nodes. In the first step, our algorithm divides A into the set of candidate internal nodes U_I and the set of leaf nodes U_L. We always put s into U_I.

$$u.n_{trans}(u.q) \geq 1 \tag{1}$$

$$u.n_{link}(u.q) \geq n + 1 \tag{2}$$

For each node $u \in A$, the algorithm checks if the above inequalities hold or not. If they hold for u, then u is put into U_I, otherwise put into U_L. The above inequalities (1) and (2) represent whether node u can perform transcoding of one or more videos and whether u can forward $n + 1$ video streams.

After that, if $|U_I| < \frac{1}{n}|A|$, quality requirements of $|U_L| - \frac{n-1}{n}|A|$ nodes in U_L with larger upstream bandwidths are reduced so that the inequalities (1) and (2) hold. Then, those nodes are moved to U_I. By the above procedure, $|U_I| \geq \frac{1}{n}|A|$ always holds.

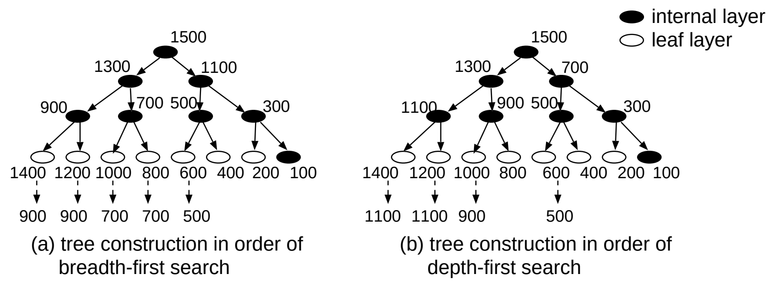


Figure 16. Layer-tree construction

In the second step, the algorithm assigns the set of all nodes A to layers. Elements of U_I are sorted in decreasing order of their quality requirements and every bunch of k elements is packed to an internal layer. Here, we select the first node of each layer as the representative node of the layer. The average value of quality requirements is assigned as the layer quality. For the set of leaf nodes U_L, elements are similarly packed to leaf layers.

In the last step, the transcode tree is constructed. The algorithm sorts internal layers in decreasing order of layer quality, and constructs the complete n-ary tree of those internal layers so that the layer quality of each layer does not exceeds that of its parent layer. Next, the algorithm attaches each leaf layer L to the internal layer whose layer quality is closest to L. If the layer quality of L exceeds that of L's parent layer, the layer quality of L is adjusted to that of L's parent.

The order of assigning internal layers to n-ary tree could be of many ways. For example, either order of breadth-first search or order of depth-first search can be used for this purpose. In Fig. 16, we show results when we assign to a complete binary tree 8 internal layers and 7 leaf layers in order of breadth-first search and order of depth-first search, respectively. Here, the layer quality of internal layers is $\{100, 300, ..., 1500\}$ and that of leaf layers is $\{200, 400, ..., 1400\}$. We assigned one internal layer (layer with quality 100) without children as a child of the internal node with the 2nd smallest layer quality.

From Fig. 16, we see that depth-first search based assignment can construct a better layer tree than breadth-first search based one. So, in our algorithm, we use the depth-first search based assignment.

Finally, the transcode tree is obtained by assigning internal nodes and leaf nodes to internal layers and leaf layers in decreasing order of their required quality, respectively.

4.3.4 Adaptation to available bandwidth between nodes

In our method, after constructing the layer tree, each node which belongs to the child layer selects an actual delivery node from k nodes in the parent layer. Whether each child node can receive the video with the requested quality or not depends on the available bandwidth on the path, that is, links on a physical network connecting the child node to the parent node. Below, we describe how to decide the parent nodes by taking into consideration of the physical topology of the network and available bandwidths on paths in the network. Here, we also consider the case that two or more overlay links share the same physical links and thus compete the available bandwidths on those links.

Let C and P be the sets of nodes which belong to a layer and its parent layer, respectively. We suppose that, for each pair of nodes between the child layer and the parent layer, the physical path and the available bandwidth can be obtained with tools such as *traceroute* and *pathload*[47], respectively. Let $bw(c,p)$ and $L(c,p)$ denote the available bandwidth measured with a tool like pathload (called *measured available bandwidth*, hereafter) and the set of links between $c \in C$ and $p \in P$ except for links connected to nodes c and p, respectively.

Next, we estimate the worst-case available bandwidth of each overlay link (called *estimated available bandwidth*, hereafter) by considering some of links are shared among multiple overlay links. Initially, for each pair of nodes $(c,p) \in C \times P$, the estimated available bandwidth of each link $l \in L(c,p)$ is set to $bw(c,p)$.

The estimation is done based on the *link stress* of each link (i.e., the number of overlay links which use the same physical link for the same data transmission) as follows. (1) The initial link stress is set to 0 for each physical link. (2) For each pair $(c,p) \in C \times P$ and for each link $l \in L(c,p)$, the link stress of l is incremented. However, once the link stress has been already incremented by node c, we do not let other paths including c increment the link stress of the same link to avoid duplicated counting.

Based on the measured available bandwidth and the link stress of each physical link, we decide the parent node of each child node as follows. (i) For each $c \in C$, the following step (ii) is examined in increasing order of node ID. (ii) For each $p \in P$, whether node p can deliver the video with the specified bitrate to node c or not is decided based on the estimated available bandwidth on path $L(c, p)$. If there is no parent node which has enough available bandwidth for the video delivery to node c, node c is moved to a lower quality layer. If only a node can deliver video to c with required bitrate, this node is selected as the parent node of c, and the following step (iv) is executed. If there are multiple nodes which can deliver video to node c with the required bitrate, the following step (iii) is applied to selecting the parent node of c. (iii) For each node $p \in P$ which can deliver video to c with the required bitrate, the new estimated available bandwidth for each link in $L(c, p)$ is calculated by dividing the current estimated bandwidth by the link stress. One node with the largest estimated available bandwidth is selected as the parent node of c. (iv) Once node p is selected as the parent of c, we re-calculate the link stress of each link $l \in L(c, p)$ without incrementing it by the paths including c and subtract the bitrate of the video from the estimated available bandwidth of l. If some bandwidth is still remaining in l, it can be used for another overlay links.

We will explain more about our algorithm with an example. We assume that $k = 3$, $C = \{c_1, c_2, c_3\}$, and $P = \{p_1, p_2, p_3\}$. We also assume that each node $c \in C$ requires the video with bitrate 800Kbps. We show measured available bandwidths and estimated available bandwidths in Fig. 17(a) and (b), respectively.

First, we decide the parent node of c_1. Link stresses on all links are shown in Fig. 17 (b). Here, all of p_1, p_2 and p_3 have enough available bandwidths for delivery of video with 800Kbps to node c_1. So, among them, one with the largest estimated available bandwidth is selected as the parent node of c_1. Since the estimated available bandwidths are 500Kbps, 400Kbps, and 400Kbps for links $(r_3, r_2), (r_3, r_6)$, and (r_3, r_6), respectively, node p_1 which has the largest estimated bandwidth is selected as the parent of c_1. Then the link stresses of all links included in the paths including p_1 are re-calculated for the topology without those paths, and the bitrate 800Kbps is subtracted from estimated available bandwidths of links on the path between c_1 and p_1. Similarly, when we apply the same algorithm to selection of the parent nodes of c_2 and c_3, p_3 and p_2 are selected as

	path	measured bandwidth
(c_1, p_1)	$(c_1, r_3, r_2, r_1, p_1)$	1000Kbps
(c_1, p_2)	$(c_1, r_3, r_6, r_5, r_4, p_2)$	800Kbps
(c_1, p_3)	$(c_1, r_3, r_6, r_5, r_8, r_7, p_3)$	800Kbps
(c_2, p_1)	$(c_2, r_6, r_3, r_2, r_1, p_1)$	800Kbps
(c_2, p_2)	$(c_2, r_6, r_5, r_4, p_2)$	900Kbps
(c_2, p_3)	$(c_2, r_6, r_5, r_8, r_7, p_3)$	2000Kbps
(c_3, p_1)	$(c_3, r_9, r_5, r_4, r_2, r_1, p_1)$	1500Kbps
(c_3, p_2)	$(c_3, r_9, r_5, r_4, p_2)$	1000Kbps
(c_3, p_3)	$(c_3, r_9, r_{10}, r_7, p_3)$	800Kbps

Table 9. Measured available bandwidths on physical paths

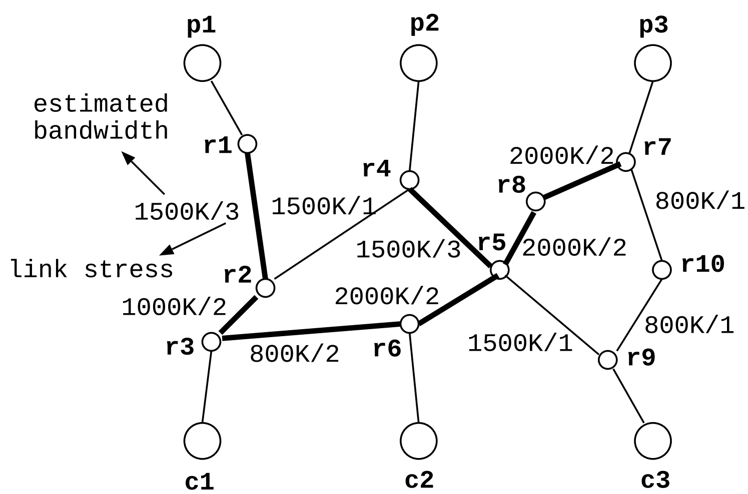


Figure 17. Physical topology between nodes with estimated available bandwidths and link stresses

their parent nodes, respectively.

How to decide appropriate values of n and k

In our method, the transcode tree is constructed as (a modified version of) a complete n-ary tree. So, as the value of n becomes large, the tree height (i.e., the number of transcoding) decreases. Since the required upstream bandwidth of each node increases in proportion of n's value, the value of n must be carefully decided considering upstream bandwidth limitation of each node. We can decide the maximum value of n so that the number of nodes satisfying inequality $u.n_{link}(q) \geq n+1$ is equal to $\frac{1}{n}|A|$. If f nodes can leave from a layer at the same time before the transcode tree is reconstructed, the remaining $k-f$ nodes in the current layer must transmit video streams to $n \cdot k$ children nodes. So, the following inequalities must hold in order to recover from f simultaneous failures in each layer.

$$\begin{aligned}(k-f)u.n_{link}(q) &\geq n \cdot k \\ (k-f)u.n_{trans}(q) &\geq \lceil \frac{k}{u.n_{link}(q)} \rceil n \end{aligned} \tag{3}$$

Thus, we can get the range of k from values of n and f. When we decide the value of k, if the value of k is not multiple of $n \times n_{trans}(q)$, nodes of the child layer can not be averagely distributed among nodes of the parent layer (for example, when $k = 5$, $n = 2$ and $n_{trans}(q) = 1$, in a layer, a parent node may have 3 child nodes, another parent node may have only 1 child node). On the other hand, if the k is not multiple of $n_{link}(q)$, the value of f will not change along with the increase of k. For example, in case of $n_{link}(q) = 3$, the value of f is always 2 even we change k from 6 to 8. So, the appropriate value of k can be selected so that the value of k is multiple of $n_{link}(q)$ and $n \times n_{trans}(q)$.

4.4 Behavior of MTcast

Startup Behavior

Let t denote the time when s starts video delivery. Each user who wants to receive video stream sends a video delivery request to the video server s before time $t - \delta$. At time $t - \delta$, s calculates the transcode tree with the algorithm explained in Sect. 4.3.3. Here, δ is the time to calculate the transcode tree and distribute the necessary information to all nodes. s also decides the node u_c which calculates the transcode tree next time. u_c is selected from representative nodes of layers which have sufficient downstream bandwidths. Next, s distributes the information which is necessary for video delivery to all nodes in T.

For information distribution, s composes data I which contains the complete information on T, its layer tree, representative nodes and quality of layers, and u_c. Then, it sends I to the representative node of the root layer. Then, the node forwards the information to its children layers' representative nodes. Data I is propagated until all leaf layers' representative nodes receive it. When each representative node receives the data I, it sends part of the information in I to member nodes of the same layer. We let each representative node keep (a) the whole layer tree with each layer's layer quality and representative node's address, and (b) its responsible layer and addresses of the layer's member nodes. We also let each node keep (1) addresses and layer quality of children nodes, (2) current layer's quality and responsible node's address, (3) parent node's layer and its responsible node's address, and (4) node u_c to calculate the transcode tree next time.

By the above steps, information of the transcode tree is shared among all nodes and video gets ready to be delivered.

How to cope with new delivery requests and node failures

As explained in Sect. 4.3.3, each node in an internal layer has an extra upstream bandwidth for forwarding one more video stream. A user node u_{new} who has requested video delivery after time t can use this extra bandwidth to receive a video stream. Here, the fanout of the forwarding node u_f which sends a stream to u_{new} is allowed to be $n + 1$ tentatively. The forwarding node u_f does not need to transcode a video stream for u_{new}, since u_f is already transmitting a video stream to n children nodes and it transmits the same stream to u_{new}.

If one or more nodes in a layer fail or suddenly leave from the transcode tree, all of their descendant nodes will not be able to receive video streams. Our method allows children nodes of the failure nodes to find alternative nodes in the same layer as those failure nodes and to ask them to forward video streams. Therefore those alternative nodes use their extra upstream bandwidths similar to the case of processing new delivery requests.

As we will explain later, the transcode tree is reconstructed periodically, the fanout of each stream is reduced to n or less than n and the consumed extra upstream bandwidth is regained after reconstruction.

If the representative node of a layer fails, children nodes of the representative node cannot find new parent nodes. Thus, one of other nodes in the layer becomes sub representative node, and nodes in children layers keep address of these nodes. When the representative node fails, one of children nodes of the representative node sends switch request to the sub representative node so that the sub representative node becomes the new representative node. If a sub representative node fails before the representative node fails, one of the other nodes become sub representative node.

Procedure for new delivery requests

We assume that a new user node u_{new} knows at least one node u^* in the transcode tree which is already receiving a video stream. u_{new} tries to find the best node in the transcode tree which can be u_{news}'s parent node in the following procedure. (1) u_{new} sends a query with its quality requirement $u_{new}.q$ and its address to u^*. (2) If u^* is not a responsible node of any layer, it forwards the received query to the responsible node u_r of u^*'s current layer. (3) When u_r receives the query, it sends the information of the layer tree to u_{new}. (4) When u_{new} receives the layer tree, it finds the layer which has the layer quality closest to $u_{new}.q$ and sends a video delivery request to the responsible node u'_r of the layer. (5) u'_r selects a node u' and forwards the request to u' which has the required extra upstream bandwidth. (6) Finally, u' starts to deliver a video stream to u_{new}.

Recovery from node failure

We let each node u monitor status of data receiving in real-time, and u thinks that node failure happened when it does not receive any data (or the average data reception rate is much less than the expected one) during a specified time period. When u detects failure of its parent node u_p, u sends a video forwarding request to the representative node of u_p's layer. Then, similarly to the case of a new video delivery request, the video stream is forwarded from an alternative node if it has an extra upstream bandwidth. At u, video can be played back seamlessly by buffering certain time of video data during the above switching process.

Reconstruction of Transcode Tree

User node u_c reconstructs the transcode tree in the following steps. We assume that all nodes know the time t_r when the reconstructed transcode tree is in effect.

Before time $t_r - \delta'$, each node u sends a new quality requirement which will be effective after t_r to the representative node of $u's$ current layer, if u wants to change video quality. Here, δ' is the time to gather quality requirements from all nodes, calculate the transcode tree and distribute the necessary information to (part of) nodes. When the representative node u_L of each layer L receives quality requirements from all members of L and those from representative nodes of L's children layers (if L has children layers), u_L sends the unified list of quality requirements to L's parent layer's representative node. Finally, the representative node of the root layer sends the received list of quality requirements to node u_c. Finally, u_c has quality requirements of all nodes which will be effective after time t_r.

Then, node u_c calculates the transcode tree with the algorithm in Sect. 4.3.3 and distribute to all nodes the information for the new transcode tree and the node u'_c which calculates the tree next time, as explained in Sect. 4.3.3.

At time t_r, all nodes stop receiving streams from current parent nodes and the nodes in the root layer of the new transcode tree starts to deliver video streams. Nodes in internal layers also forward video streams after receiving them. The video stream transmitted along the new transcode tree arrives after a certain time lag due to transcode and link latency. So, during the time lag, each node plays back video from its buffer to avoid blank screen.

For the next reconstruction of the transcode tree, the buffer of each node must be filled with video data of the above time lag. This process is done by transmitting the video stream slightly faster than its playback speed. This fast transmission requires more computation power for transcode and more bandwidth for forwarding video data. Let α denote the ratio of the above time lag over the time period between two subsequent tree reconstructions. Here, α is a real constant number between 0 and 1. Then this fast transmission requires computation power and upstream/downstream bandwidths $(1+\alpha)$ times as much as the normal transmission.

Reconstruction of the transcode tree may greatly change positions of nodes in the tree. For example, a node near the root node may change its position near the leaf node in the new transcode tree, or vice versa. Because the time lag until receiving the video stream since the video server sent it is different depending on positions in the transcoding tree, we should keep all nodes to play back the same scene in the video. To do so, we let nodes closer to the root node play back video with larger delay by buffering certain time of video data. Data amount to be buffered can be decided with statistic information calculated from received video streams.

Considerations

Decision method of quality requirement

In our method, being an internal node requires rather high computation power, since video decoding and transcoding are simultaneously performed. On the other hand, being a leaf node only requires computation power to decode a video stream. Also if a node doesn't have enough processing power to transcode received video, the node is likely assigned to a leaf layer. Thus, users may try to become a leaf nodes by specifying superfluously high video quality. In this section, we describe a way to avoid such a situation by defining the maximum quality requirement.

There are following restrictions of quality requirements. (1) If a node has enough network bandwidth, screen resolution and processing power, the node can request any quality requirement. (2) If a node doesn't have large screen, appropriate bitrate of video is calculated from the screen size, and the node can request bitrate less than that. Appropriate bitrate of video can be calculated from screen size and frame rate[48]. (3) If a node doesn't have either large network bandwidth or processing power, the node can request quality requirement under the condition that the node can become an internal node. Only if the maximum quality requirement calculated by this method is too low and useful information cannot be obtained from the video, the node is allowed to request quality requirement which doesn't meet the condition to become an internal node.

Validity of assumptions

As described in Sect.4.3.2, if there are not enough number of candidate internal nodes, part of leaf nodes which have larger upstream bandwidths are transformed into internal nodes by decreasing their quality requirements. This could be the largest factor of users' unsatisfaction. Thus, the proposed technique is especially effective if (i) there are many users who have larger bandwidths compared to quality requirements. Also, our technique is effective if (ii) users' quality requirements distribute widely, since the technique can flexibly adjust video quality by transcoding, compared to layered multicast techniques.

Hereafter, we give some typical environments where the condition (i) and (ii) hold. Under following three example environments, condition (i) holds. (1) A video delivery system in which users pay fee according to video quality. (2) An environment where user's available network bandwidth is much larger than bitrate of video. (3) An environment where video quality is restricted by display resolution. Regarding (1), even if a user has large available network bandwidth, the user may want to keep video quality low, especially when the video is not very important. Regarding (2), if a user is connecting to the Internet through the optical fibre network, available network bandwidth is usually much larger than bitrate of video, and thus there can be many users with large unused network bandwidth if such a network becomes popular. Regarding (3), it is possible that a user watches video using a portable game console or a PDA. These devices normally have screens with resolutions smaller than VGA, and it is quite unlikely that users of these devices request larger resolution than that, even if there is plenty of network bandwidth.

Next, we give three examples under which condition (ii) holds. (1) Watching multiple videos simultaneously on a single screen. (2) Recording video under restriction of disk space. (3) Watching multi-object video with adjusted quality of objects according to importance of each object. Regarding (1), contents such as news and stock prices are displayed on PC screen, and watched when the user is doing other jobs on another window. Users set window size according to their interests, and thus there would be various quality requirements. Regarding (2), users may want to record received video in bitrate according to importance of the video. In this case, quality requirement varies depending on user's interest and the size of disk space. Regarding (3), multi-object video is played back under constraints of network bandwidth. Users may want to watch important objects in higher bitrate. Quality requirements of objects would vary depending on importance of objects.

Assuming environments described above, we investigate user's satisfaction by our technique under conditions where the number of transformed leaf nodes is varied, and distributions of quality requirements are varied.

4.5 Evaluation

In order to show usefulness of MTcast, we have conducted several experiments. We have measured (i) required computation power for transcoding, (ii) video quality deterioration by multiple transcoding, (iii) overhead of tree reconstruction, (iv) the user satisfaction degree with MTcast, and (v) physical transmission path length in overlay links.

4.5.1 Required Computation Power for Transcoding

In our method, since transcoding is processed on user nodes, the load of transcoding should not influence the playback of video. So, we examined the load of transcoding while playing back a video using a desktop PC, a laptop PC, and a PDA. In the experiment, we measured maximum processing speed of transcoding (in fps) while playing back a video and compared it with the actual playback speed of the video. If the maximum processing speed is sufficiently larger than the actual playback speed, it can be said that the load of the transcoding doesn't influence playback of the video. We measured the maximum processing speed by changing the transcoding degree (i.e., the number of simultaneous transcode processing) from 1 to 3. In the experiment, we used `mpeg2dec 0.4.0b` as the decoder and `ffmpeg 0.4.9-pre1` as the encoder. The experimental parameters and results are shown in Table 10. The specifications of the devices in Table 10 are as follows: desktop PC (CPU: Pentium4 2.4GHz, 256MB RAM, Linux2.6.10), laptop PC (CPU: Celeron 1GHz, 384MB RAM, Linux 2.4.29), and PDA (SHARP

Zaurus SL-700, CPU: XScale PXA250 400MHz, 32MB RAM, Linux 2.4.28). The original frame rate of all videos is 24 fps.

Table 10. Maximum processing speed while playing back a video

device	original video		transcoded video		transcoding degree		
	picture size	bit rate (kbps)	picture size	bit rate(kbps)	1	2	3
Desktop PC	640x480	3000	480x360	2000	35.66fps	20.03fps	14.84fps
Desktop PC	480x360	2000	352x288	1500	61.60fps	36.40fps	25.89fps
Laptop PC	352x288	1500	320x240	1000	49.90fps	30.65fps	21.84fps
PDA	320x240	1000	208x176	384	10.12fps	6.04fps	4.33fps

Table 10 shows that common desktop PCs and laptop PCs have enough computation power to simultaneously transcode one or more videos with 3000Kbps (640x480 pixels) and with 1500 Kbps (352x288 pixels) in real-time, respectively. In MTcast, each internal node needs computation power more than one transcoding degree. Table 10 shows that this requirement is not so hard. However, PDA's maximum processing speed is 10.12 fps even if the transcoding degree is 1. It shows that PDAs and smaller computing devices cannot be used as internal nodes of the transcode tree.

4.5.2 Degree of quality degradation by transcoding

In our method, transcoding is repeatedly performed at each internal node from the root node to the leaf node in the transcode tree. Multiple transcoding may cause serious quality degradation of video. So, we investigated the average PSNR value among 500 frames using a video after (1) multiple transcoding has been performed without changing parameter values of the video (picture size, frame rate, and bit rate) and (2) multiple transcoding has been performed by changing parameter values of the video. Experimental results for cases (1) and (2) are shown in Fig.18 and Fig.19, respectively.

In Fig.18, transcoding is performed repeatedly without changing parameter values of a video with 640x480 pixels, 24fps and 3000kbps. Fig.18 shows that PSNR value decreases slightly before the third transcoding and keeps an almost constant quality after that.

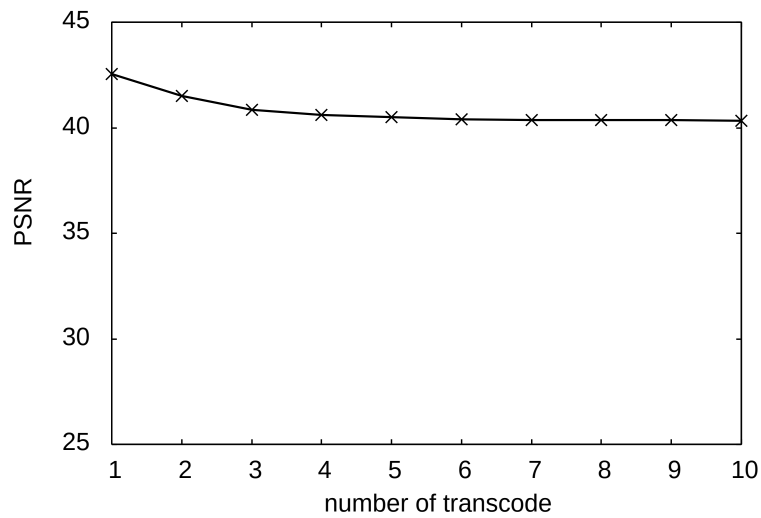


Figure 18. PSNR values of multiple transcode without changing parameter values of a video

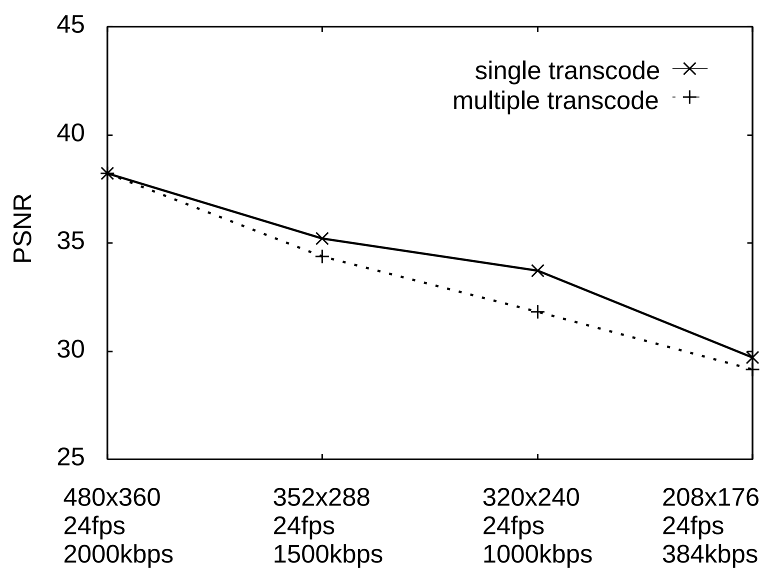


Figure 19. PSNR values of multiple transcode by changing parameter values of a video

Next, we applied transcoding four times to the original video with 640x480 pixels, 24fps, and 3000kbps to 208x176, 24fps. The parameter values of the video are changed to (i) 480x360 pixels, 24fps and 2000Kbps by the first transcoding, (ii) 352x240 pixels, 24fps and 1500Kbps by the second transcoding, (iii) 320x240 pixels, 24fps and 1000Kbps by the third transcoding, and (iv) 208x176 pixels, 24fps and 384Kbps by the fourth transcoding. We also changed parameter values of the original video to the above (i), (ii), (iii) or (iv) by single transcoding. We measured PSNR values of those obtained videos and compared those values of the same resolution videos obtained by one transcoding and multiple transcoding. The result is shown in Fig.19. Fig.19 shows that PSNR values are almost the same between the multiple transcoding case and the single transcoding case.

From the above experiments, we think that quality degradation of multiple transcoding is within the allowable range.

4.5.3 Overhead of Tree Reconstruction

In our method, the transcode tree is reconstructed periodically and/or when a new video segment starts. The overhead of the tree reconstruction consists of (i) aggregation of quality requirements for the new video segment from (part of) user nodes, (ii) calculation of the new transcode tree, and (iii) distribution of the new transcode tree to representative nodes of all layers.

For the above (i), even when the number of nodes is 100,000 [4] and each node sends a 50 Byte packet for quality requirement directly to the computation node u_c, 5 MByte information is sent to the node u_c which computes the transcode tree. If we assume that this information is sent in 10 seconds (it should be less than the period of the tree reconstruction), the average transmission speed becomes 4 Mbps. Since only the node with enough downstream bandwidth can be selected as u_c, this would not be a bandwidth bottleneck. Also, the required bandwidth can be reduced by letting each node send the quality requirement message along the current transcode tree so that each internal node merges the received messages into one message and forwards it to its parent node recursively.

In order to investigate the impact of the above (ii) and (iii), we measured the

[4] This number is actually much smaller since only the nodes which want to change their quality requirements for the next video segment send the messages.

size and the computation time of the transcode tree with the number of nodes from 1,000 to 100,000. Here, we assumed that $n = 2$ and $k = 5$, where n and k are the fanout of each internal node and the number of layer members, respectively. The experimental result is shown in Table 11.

Table 11. Size and Computation Time of Transcode Tree

number of nodes	computation time (sec)	size of tree (byte)
1000	0.016	3K
10000	0.140	30K
100000	1.497	300K

According to Table 11, the computation time was within 2 seconds even when the number of nodes is 100,000 (Pentium 4 2.4GHz with 256MB RAM on Linux2.6.10). So, computation time would not be a bottleneck.

The size of the transcode tree was 30 Kbyte when the number of all nodes is 10,000. The information of the tree is sent to representative nodes of all layers along the layer tree. If we assume that this is sent in 10 seconds, each representative node needs 24Kbps extra bandwidth. Even when the number of nodes is 100,000, the required bandwidth would be 240Kbps. Also, the tree size can be further reduced with the general compression algorithm like `gzip`.

4.5.4 User Satisfaction

In this section, we compare MTcast with the layered multicast method in terms of the user satisfaction degree for the quality requirements.

Similarly to [37], the satisfaction degree of user u ($0 \leq S_u \leq 1$) is defined as follows.

$$S_u = 1 - \frac{|u.q - u'.q|}{u.q} \tag{4}$$

Here, $u.q$ represents u's required quality and $u.q'$ represents the quality of the received video. When $u.q'$ is closer to $u.q$, S_u gets closer to 1.

The experiment has been conducted as follows: The physical network topology with 6000 nodes is generated with Inet3.0 [44] and 1000 nodes are selected as user nodes. Links directly connected to those user nodes are regarded as LANs. Links attached to LAN links are considered as MAN links, and other links are considered as WAN links. We assume that there are the following four types of user nodes: (1) user nodes with cell phone networks whose available downstream bandwidths are 100 to 500 Kbps; (2) user nodes with wireless LAN (2 Mbps to 5 Mbps); and (3) user nodes with wired broadband networks (10 Mbps to 20 Mbps). We assume that each user node has the same amount of available upstream bandwidth as the downstream bandwidth.

Table 12. Configuration of Available Bandwidth

	100k to 500k	2M to 5M	10M to 20M
case1	33%	33%	33%
case2	5%	33%	62%
case3	45%	10%	45%
case4	62%	33%	5%

We selected the quality requirement of each user node according to one of the following three distributions within the available bandwidth: (a) uniform distribution from 300 Kbps to 3 Mbps; (b) bandwidth is chosen from 300Kbps and 3M bps; and (c) sum of two normal distributions with 300 Kbps average and 50Kbps standard deviation and with 3 Mbps average and 1 Mbps standard deviation. On the other hand, the total sum of bandwidths of LAN links connected to each MAN link was used as the bandwidth of the MAN link. 6 Gbps was used as bandwidths for WAN links.

In the above simulation configuration, we measured the average user satisfaction degree ($\frac{1}{|U|}\sum_{u\in U} S_u$, U is the set of all users). We changed the number of user nodes from 1 to 1000 and measured the average satisfaction degree for the combination of the above three quality requirement distributions (a) to (c) and four different types of populations of user nodes shown in Table 12. The experimental results are shown in Fig. 20, Fig. 21 and Fig. 22. In the figures, X-axis and Y-axis represent the number of nodes and the average satisfaction degree, respectively.

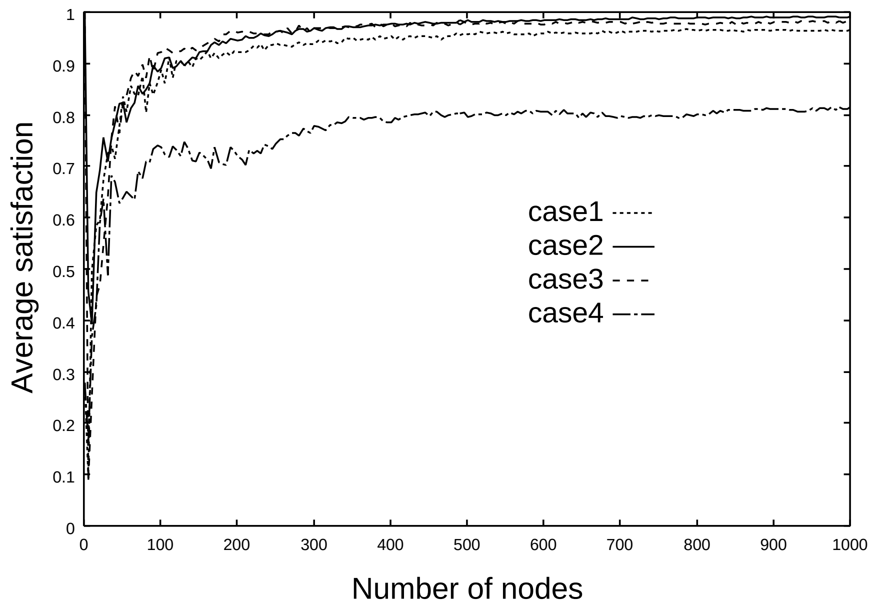


Figure 20. Average User Satisfaction by quality requirement (a)

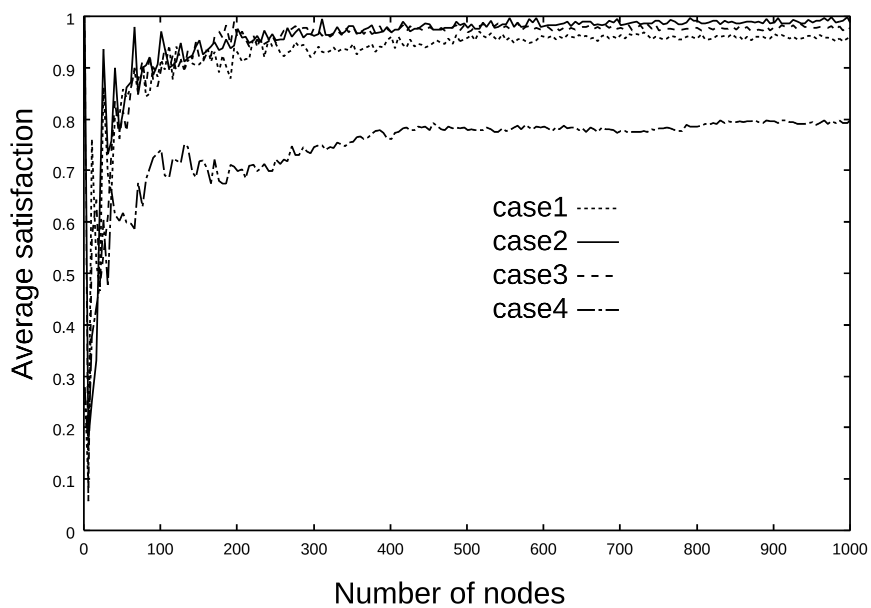


Figure 21. Average User Satisfaction by quality requirement (b)

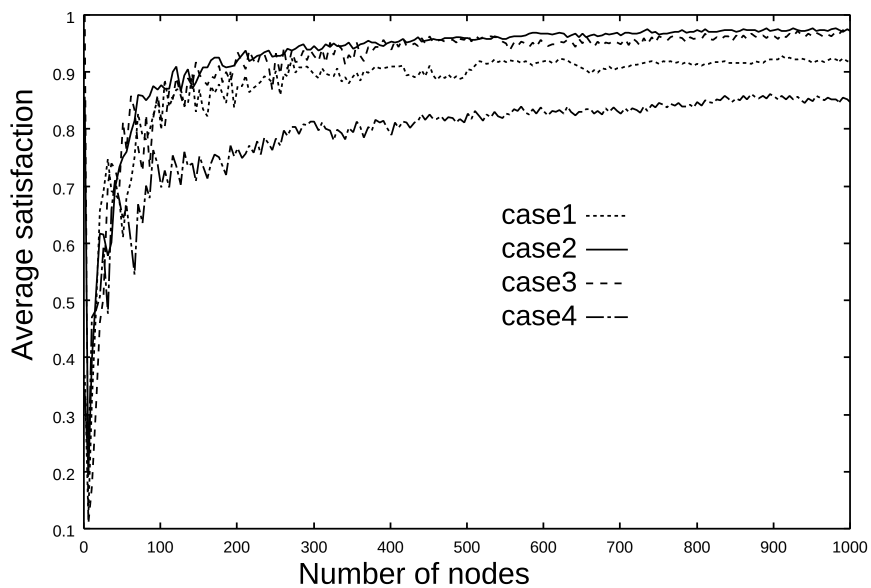


Figure 22. Average User Satisfaction by quality requirement (c)

From Fig. 20, Fig. 21 and Fig. 22, we see that MTcast can achieve pretty high satisfaction degree for various distribution of quality requirements from user nodes, when the number of user nodes are more than 100. The satisfaction degree is lower in case 4 than other cases. This is because the percentage of user nodes with higher bandwidth is much smaller in case 4. However, even in such a case, MTcast achieved more than 70% user satisfaction.

In order to measure variation of user satisfaction degree depending on the value of k, we measured average user satisfaction degrees for $k = 2, 3, 6$ and 9 which are derived when applying four different combinations of $u.n_{trans}(u.q)$ and $u.n_{link}(u.q)$ in Table 13. From Table 13, when $k = 2$ or $k = 3$, the system can be recovered from one node failure per layer, and when $k = 6$ or $k = 9$, the system can be recovered from two and three simultaneous node failures per layer, respectively (these are calculated by equation (3)). However, as the value of k increases, the average user satisfaction degree might decrease since the delivered quality is averaged among k members of each layer. The experimental result is shown in Fig. 23.

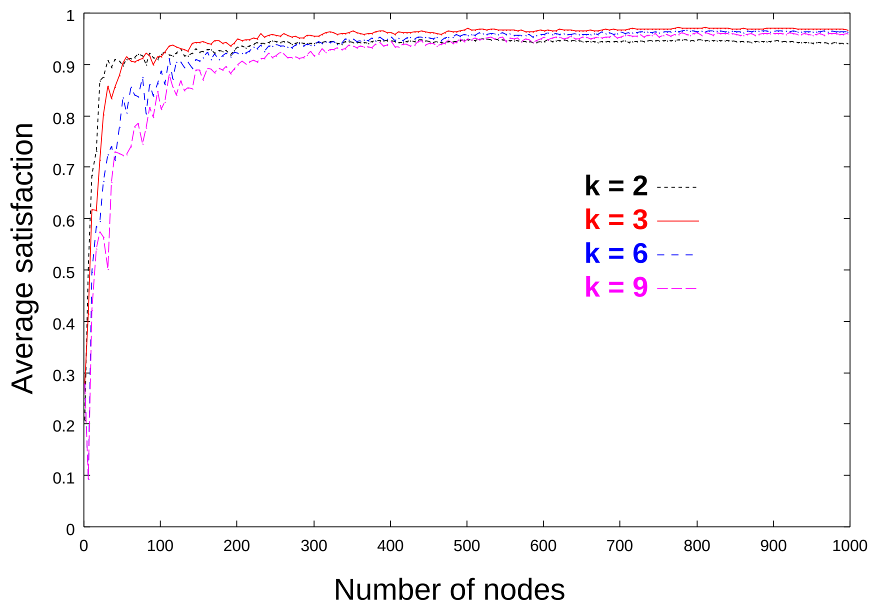


Figure 23. User Satisfaction vs. Allowable Failures per Layer

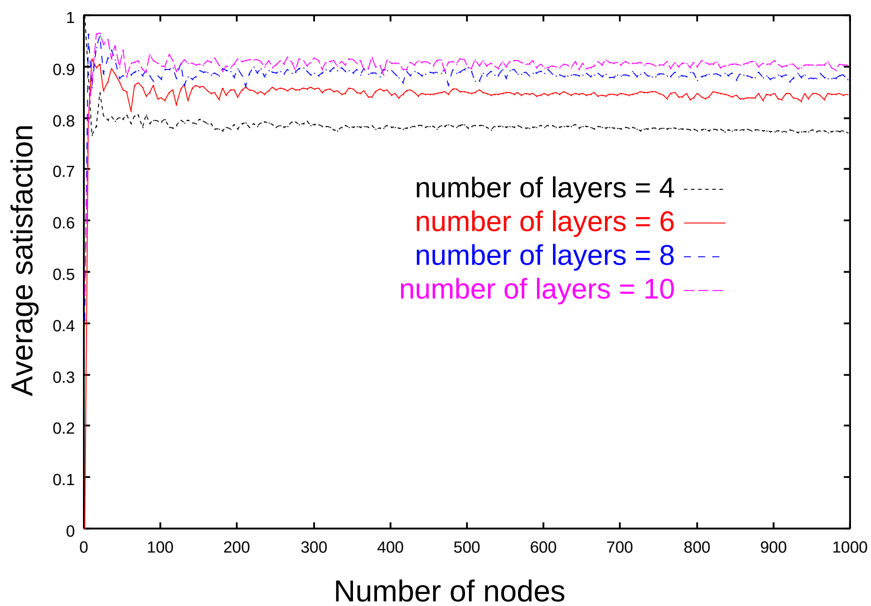


Figure 24. Average User Satisfaction by Layered Multicast

Table 13. Relationship among $u.n_{transcode}$, $u.n_{link}$ and f

	$p.n_{transcode}$	$p.n_{link}$	k	f
pref. 1	2	4	2	1
pref. 2	1	3	3	1
pref. 3	1	3	6	2
pref. 4	1	3	9	3

From Fig. 23, while the number of nodes is relatively small (i.e., less than 300), the average user satisfaction degree decreases as the value of k increases. However, as the number of user nodes increases, the decrease gets smaller. From the result, while the number of user nodes is small, we should keep the value of k small in order to keep the average user satisfaction degree high, and we should increase the value of k gradually to improve robustness against node failure as the number of users increases.

For comparison, we also measured the average user satisfaction degree when using the layered multicast method. Here, we assume that each node only receives streams. The average user satisfaction degree depends largely on the proportion of bitrates among multiple layers. So, we used the following way for allocating bitrates of layers: The average user satisfaction degree was considered as the evaluation function, and the optimal allocation of encoding rates were calculated for basic and extension layers using the Simulated Annealing method (the number of repetition times were 10,000).

With this optimization technique, we measured the average user satisfaction degrees. The results are shown in Fig. 24. From Fig. 23 and Fig. 24, when the number of nodes is sufficient (more than 200), MTcast achieves much higher satisfaction degree than the layered multicast with less than 10 layers.

4.6 Conclusions

In this Chapter, we proposed a new video delivery method called MTcast to achieve efficient simultaneous video delivery to multiple heterogeneous users. In the proposed method, the same video stream is transmitted from a video server to user nodes by step-by-step transcoding at each intermediate node. The main contributions of MTcast are the following: (1) quick failure recovery and new user's quick reception of video streams can be achieved owing to layers of user nodes, (2) the size and height of the tree are kept small by periodical tree reconstruction, and (3) higher user satisfaction can be achieved with reasonable resource consumption at user nodes.

The above (2) also allows users to play back video segments with various different quality. When we use MTcast with our energy consumption control technique in Chapter 3, users can increase playback quality for preferred video segments without shortening playable time at portable devices within the battery amount.

5. Conclusion

In this thesis, the following three research topics about achieving quality of service in distributed multi-media systems have been studied.

First, a testing method for QoS functions in distributed multi-media systems has been provided. Here, we use a statistical approach where test sequences take samplings of actual frame rates and/or time lags when an IUT is executed, and report test results from ratio of samplings with low quality below a threshold in a normal distribution of all samplings.

Secondly, a QoS adaptation method for streaming video playback for portable computing devices where playback quality of each video fragment is automatically adjusted from the remaining battery amount, desirable playback duration and the user's preference to each fragment, has been proposed.

Finally, a new video delivery method which achieves efficient simultaneous video delivery to multiple users with different quality requirements by relying on user nodes to transcode and forward video to other user nodes has been proposed.

In the delivery method, although the load of decode/encode have been distributed to each node, we only provided a centralized algorithm for constructing the transcode tree, although it works for the scale of 100,000 nodes. As part of future work, we want to design a distributed algorithm for tree construction to improve scalability further.

Acknowledgements

I would like to thank all those people who made this thesis possible and gave an enjoyable experience to me.

First, I am deeply indebted to my supervisor Professor Minoru Ito and Professor Hiroyuki Seki for their valuable suggestions, advice and support.

I would like to thank Professor Masaaki Mori of Shiga University for his invaluable comments, advice and support for so long time.

I would like to express my sincere gratitude to Associate Professor Keiichi Yasumoto for his adequate guidance, valuable suggestions and discussions throughout this work. This work could not be achieved without his support, encouragement and guidance.

I would like to thank Professor Teruo Higashino of Osaka University and Associate Professor Naoki Shibata of Shiga University for their invaluable comments, valuable suggestions and support.

I'm thankful to Mr. Morihiko Tamai for his valuable comments.

Finally, I would like to thank all of members of Ito Laboratory for their helpful advice and support.

References

[1] Alur, R. and Dill, D. L.: "Theory of timed automata", *Theoretical Computer Science*, Vol. 126, pp. 183–235 (1994).

[2] Alur, R. and Henzinger, T. A.: "Logics and Models of Real Time: A Survey", *"Real Time: Theory in Practice", LNCS 600*, pp.74–106 (1992).

[3] Bowman, H., Faconti, G. and Massink, M. : "Specification and verification of media constraints using UPPAAL", Proc. of 5th Eurographics Workshop on the Design, Specification and Verification of Interactive Systems, pp. 261–277 (1998).

[4] Cheung, S.C., Chanson, S.T. and Xu, Z. : "Toward Generic Timing Tests for Distributed Multimedia Software Systems", *IEEE Int'l. Symp. on Software Reliability Engineering* (2001).

[5] Fibush, D.K. : "Testing Multimedia Transmission Systems", *IEEE Design & Test of Computers*, Vol.12, No.4, pp.24-44 (1995).

[6] Grabowski, J. and Walter, T. : "Testing Quality-of-Service Aspects in Multimedia Applications",*Proc. of 2nd Workshop on Protocols for Multi-media Systems (PROMS)* (1995).

[7] Higashino, T., Nakata, A., Taniguchi, K. and Cavalli, A.R.: "Generating Test Cases for a Timed I/O Automaton Model", *Proc. 12th IFIP Workshop on Testing of Communicating Systems (IWTCS'99)*, pp. 197–214 (1999).

[8] Huang, C. M. and Wang, C. : "Synchronization for Interactive Multimedia Presentations", IEEE MULTIMEDIA, Vol. 5, No. 4, pp.44-62 (1998).

[9] ISO : "Information Processing System, Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour", *ISO 8807* (1989).

[10] Kone, O: "A Local Approach to the Testing of Real-time Systems", *The Computer Journal*, Vol. 44, No. 5 (2001).

[11] Lee, D. and Yannakakis, M.: "Principles and Methods of Testing Finite State Machines – A Survey", *Proc. of the IEEE*, Vol. 84, No. 8 (1996).

[12] Misic, V. B., Chanson, S. T. and Cheung, S.: "Towards a Framework For Testing Distributed Multimedia Software Systems", *Proc. of 1998 Int'l. Symp. on Software Engineering for Parallel and Distributed Systems (PDSE98)* (1998).

[13] Vissers, C. A., Scollo, G. and Sinderen, M. v.: "Specification Styles in Distributed Systems Design and Verification", *Theoretical Computer Science*, Vol. 89, No. 1, pp. 178 – 206 (1991).

[14] Walter, T., Scheferdecker, I. and Grabowski, J. : "Test Architectures for Distributed Systems - State of the Art and Beyond (Invited Paper)", *Testing of Communication Systems*, Vol.II, Chapman & Hall (1998).

[15] W3C: "Synchronized Multimedia Integration Language (SMIL) 1.0 Specification", http://www.w3c.org/TR/REC-smil/

[16] Yamada, M., Mori. T., Fukada, A., Nakata, A. and Higashino, T.: "A Method for Functional Testing of Media Synchronization Protocols", *Proc. of the 16th Int'l. Conf. on Information Networking (ICOIN-16)* (2002).

[17] Yasumoto, K., Umedu, T., Yamaguchi, H., Nakata, A. and Higashino, T.: "Protocol Animation based on Event-driven Visualization Scenarios in Real-time LOTOS," *Computer Networks*, Vol. 40, No. 5, pp. 639–663 (2002).

[18] Sun Microsystems: "Java Media Framework Home Page", http://java.sun.com/products/java-media/jmf/

[19] Agrawal, P., Chen, J-C, Kishore, S., Ramanathan, P. and Sivalingam, K.: "Battery Power Sensitive Video Processing in Wireless Networks," *Proc. IEEE PIMRC'98*, Vol. 1, pp.116–120 (1998).

[20] Cavallaro, A., Steiger, O. and Ebrahimi, T.: "Semantic Segmentation and Description for Video Transcoding," *Proc. of the 2003 IEEE Int'l. Conf. on Multimedia and Expo. (ICME2003)*, Vol. 3, pp. 597–600 (2003).

[21] Lahiri, K., Raghunathan, A. and Dey, S.: "Battery-Efficient Architecture for an 802.11 MAC Processor," *Proc. of the 2002 IEEE Int'l Conf. on Communications (ICC2002)*, pp.669–674 (2002).

[22] Lim, J., Kim, M., Kim, J and Kim, K.: "Semantic Transcoding of Video based on Regions of Interest, " *Proc. of Visual Communications and Image Processing 2003 (VCIP2003)* (2003).

[23] Lin, C.-Y., Tseng, B. L., Naphade, M., Natsev, A. and Smith, J. R.: "MPEG-7 Video Automatic Labeling System," *Proc. of the 11th ACM Int'l. Conf. on Multimedia*, pp. 98–99 (2003).

[24] Pering, T., Burd, T. and Brodersen, R.: "Dynamic Voltage Scaling and the Design of a Low-Power Microprocessor System," *Proc. of Driven Microarchitecture Workshop*, pp.107–112 (1998).

[25] Simunic, T., Benini, L., Acquaviva, A., Glynn, P. and Micheli, G. D.: "Dynamic Voltage Scaling for Portable Systems," *Proc. of the 38th Design Automation Conf. (DAC2001)*, pp.524–529 (2001).

[26] Xu, R., Li, Z., Wang, C. and Ni, P.: "Impact of Data Compression on Energy Consumption of Wireless-Networked Handheld Devices," *Proc. of the 23rd Int'l. Conf. on Distributed Computing Systems (ICDCS'03)*, pp.302–311 (2003).

[27] Yip, K.W. and Ng, T.S.: "Fast Power Control, Transmit Power Reduction and Multimedia Communications over WLANs," *Proc. of First Int'l Conf. on Information Technology and Applications (ICITA2002)* (2002).

[28] Cavallaro, A., Steiger, O. and Ebrahimi, T.: "Semantic Segmentation and Description for Video Transcoding," *Proc. of the 2003 IEEE Int'l. Conf. on Multimedia and Expo. (ICME2003)*, Vol. 3, pp. 597–600 (2003).

[29] Lim, J., Kim, M., Kim, J and Kim, K.: "Semantic Transcoding of Video based on Regions of Interest," *Proc. of Visual Communications and Image Processing 2003 (VCIP2003)* (2003).

[30] Lin, C.-Y., Tseng, B.L. and Smith, J. R.: VideoAnnEx: "IBM MPEG-7 Annotation Tool," http://www.alphaworks.ibm.com/tech/videoannex

[31] Lin, C.-Y., Tseng, B. L., Naphade, M., Natsev, A. and Smith, J. R.: "MPEG-7 Video Automatic Labeling System," *Proc. of the 11th ACM Int'l. Conf. on Multimedia*, pp. 98–99 (2003).

[32] Sharp Corp: Personal Server HG-01S, http://sharp-world.com/corporate/news/030117.html

[33] Tamai, M., Yasumoto, K., Shibata, N. and Ito, M.: "Low Power Video Streaming for PDAs," *Proc. of the 8th IEEE Int'l. Workshop on Mobile Multimedia Communications (MoMuC2003)*, pp. 31-36 (2003).

[34] Gregory, C. A., Gary, S. G., Alan, F. L., and Yuriy, A. R.,: "Video Coding for Streaming Media Delivery on the Internet," IEEE Trans. on Circuits and Systems for Video Technology, Vol. 11, No. 3 (2001).

[35] Nichols, K., Blake, S., Baker, F. and Black, D.L.: "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," IETF RFC 2474 (1998).

[36] Stephen, J., and Alexandros, E.: "Streaming Video using Dynamic Rate Shaping and TCP Flow Control," Visual Communication and Image Representation Journal, (1998). (invited paper).

[37] Liu, J., Li, B., and Zhang, Y. Q.: "An End-to-End Adaptation Protocol for Layered Video Multicast Using Optimal Rate Allocation," IEEE Trans. on Multimedia, Vol. 6, No. 1 (2004).

[38] Zhu, L., Ansari, N., Sahinoglu, Z., Vetro, A. and Sun, H.: "Scalable Layered Multicast with Explicit Congestion Notification", IEEE Int'l. Conf. on Information Technology Coding and Computing, pp, 331–335 (2003).

[39] Stevevn, M., Van J. and Martin V.: "Receiver Driven Layered Multicast," Proc. of ACM SIGCOMM'96, pp. 117–130 (1996).

[40] Lorenzo, V. and Jon, C.: "TCP-like Congestion Control for Layered Multicast Data Transfer," Proc. of IEEE INFOCOM'98, Vol. 3, pp. 996-1003 (1998).

[41] Brett, J. V., Celio, A. and Tatsuya, S.: "Source-Adaptive Multilayered Multicast Algorithms for Real-Time Video Distribution," IEEE/ACM Transactions on Networking, Vol. 8, No. 6, pp. 720–733 (2000).

[42] Liu, J., Li, B., and Zhang, Y. Q.: "A Hybrid Adaptation Protocol for TCP-Friendly Layered Multicast and Its Optimal Rate Allocation," Proc. of IEEE INFOCOM'02, Vol. 3, pp. 1520-1529 (2002).

[43] Hayder, M. R., Mihaela, S., Chen, Y.: "The MPEG-4 Fine-Grained-Scalable Video Coding Method for Multimedia Streaming over IP," IEEE Trans. on Multimedia, Vol. 3, No. 1 (2001).

[44] Winick, J. and Jamin, S.: "Inet3.0: Internet Topology Generator," Tech. Report UM-CSE-TR-456-02 (http://irl.eecs.umich.edu/jamin/) (2002).

[45] John, G. and Susie, J. A.: "Unbalanced Multiple Description Video Communication using Path Diversity," Proc. of IEEE Int'l. Conf. on Image Processing (2001).

[46] Wang, Y., Orchard, M. and Reibman, A. R.: "Multiple Description Image Coding for Noisy Channels by Pairing Transform Coefficients," Proc. of IEEE Signal Processing Society 1997 Workshop on Multimedia Signal Processing (1997). (Electronic Proceedings)

[47] Dovrolis, C., Prasad, R., Murray, M. and Claffy, K.: "Bandwidth Estimation: Metrics, Measurement Techniques, and Tools," IEEE Network, November-December, pp. 27–35, 2003.

[48] Tamai, M., Sun, T., Yasumoto, K., Shibata, N. and Ito, M.: "Energy-aware Video Streaming with QoS Control for Portable Computing Devices, " Proc. of The 14th ACM Int'l. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV2004), pp. 68-73, (June 2004).

[49] Banerjee, S., Christopher, K., Koushik, K., Bobby, B., and Samir, K.: "Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications," IEEE Infocom 2003, pp. 1521-1531 (2003).

[50] Padmanabhan, V., Wang, H., Chou, P., and Sripanidkulchai, S.: "Distributing streaming media content using cooperative networking," ACM/IEEE NOSSDAV (2002).

List of Major Publications

Journal Papers

1. Sun, T., Tamai, M., Yasumoto, K., Shibata, N. and Ito, M.: Energy-aware Video Streaming System with QoS Adaptation based on Inter-segment Importance, IPSJ Trans., (in Japanese), No. 46, Vol. 2, pp. 546-555 (2005).

2. Sun, T., Yasumoto, K., Mori, M.: A Method for Testing QoS in Multi-Media Systems, IPSJ Trans., (in Japanese)., Vol. 45, No. 2, pp. 475-486 (2004).

International Conference

1. Sun, T., Tamai, M., Yasumoto, M., Shibata, N., Ito, M. and Mori, M.: MTcast: Robust and Efficient P2P-based Video Delivery for Heterogeneous Users, Proc. of the 9th Int'l. Conf. on Principles of Distributed Systems (OPODIS2005) (December 2005).

2. Sun, T., Yasumoto, K., Mori, M., Higashino, T.: QoS Functional Testing for Multi-media Systems, Proc. of the 23rd IFIP Int'l. Conf. on Formal Techniques for Networked and Distributed Systems (FORTE2003), LNCS2767, pp. 319-334 (September 2003).

3. Sun, T., Yasumoto, K. and Mori, M.: A Method for QoS Functional Testing in Distributed Multi-media Systems, Proc. of the 1st Int'l. Conf. on Information Technology and Applications (ICITA2002) (Nov. 2002)

4. Tamai, M., Sun, T., Yasumoto, K., Shibata, N. and Ito, M.: Energy-aware Video Streaming with QoS Control for Portable Computing Devices, Proc. of The 14th ACM Int'l. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV2004), pp. 68-73 (June 2004).

5. Tamai, M., Sun, T., Yasumoto, K., Shibata, N. and Ito, M.: Energy-aware QoS Adaptation for Streaming Video based on MPEG-7, Proc. of 2004 IEEE Int'l. Conf. on Multimedia and Expo (ICME'2004) (Jun. 2004).

6. Yamaoka, S., Sun, T., Tamai, M., Yasumoto, K., Shibata, N. and Ito, M.: Resource-Aware Service Composition for Video Multicast to Heterogeneous Mobile Users, Proc. of 1st ACM Int'l. Workshop on Multimedia Service Composition (MSC'05) (ACM Multimedia 2005 Workshop), pp. 37-46 (Nov. 2005).