Informing Clients through Multimedia Communications: An Approach to Provide Interactivity

Marco Furini
University of Piemonte Orientale, Alessandria, Italy

marco.furini@mfn.unipmn.it

Marco Roccetti
University of Bologna Bologna, Italy

roccetti@cs.unibo.it

Abstract

One of the key problems in informing clients through multimedia streaming applications over the Internet is to customize the stream of information according to the client's requests. This is achievable only if client and server can interact along the application lifetime, which is possible only if the communication system supports the rigid timing constraints imposed by these interactive applications on their traffic. In the Internet scenario, these applications are very difficult to support, as the Internet provides a best-effort service to the traffic it carries, which means that the Internet does not make any promises about the end-to-end delay for an individual packet and about the variation of packet delay (network jitter) within a packet stream. These problems are confirmed by several experiments we performed over the Internet, which highlight that interactive applications achieve a quality that is frustrating. The contribution of this paper is the proposal of a novel mechanism to support interactive multimedia streaming applications over the Internet. Our mechanism adapts the multimedia stream transmission to the network conditions, by intentionally and slightly acting on the video QoS. Our mechanism has been validated through several experiments performed over the Internet. Results confirm that the supported interactive applications achieve a satisfactory quality and the user perceives a video quality only slightly affected by the QoS modification introduced by our mechanism.

Keywords: Multimedia communication, Interactive application, Quality of Service, Video Streaming, Consistent Information.

Introduction

Multimedia has long played an important role in the process of informing activities by changing the way we learn, think, work and live (Zeng & Yu, 1999). Not surprisingly, the use of multimedia information is growing at an exponential rate, imposing a great challenge on the way information are controlled and organized. In future years, this trend is expected to continue, thanks to the development in processor speed and to the advances in network technologies. Nowadays, learning, studying, researching, and communicating are examples of activities that users can do by means of multimedia streams.

Editor: Nalin Sharda

The reason of this success is that multimedia information is far more rich, educational and entertaining than traditional text-based information and nowadays users may access to broadband and/or wireless technologies (DSL, Fiber optics, Wi-Fi, Hedge/GPRS, CDMA, to name a few) to reach multimedia contents whenever and wherever they want.

Many multimedia applications are available (on-demand multimedia services, video-conferencing, distance learning, on-line games and pay-per-view, just to name a few), but the emerging multimedia applications are those that enable natural interaction among end-users: for instance, user can interact along the application lifetime with the contents provider in order to get a customized stream of multimedia information. A large interest is given to these *interactive* applications, which are becoming more and more attractive and popular over the Internet. On-line games, interactive-webtv, on-demand multimedia services are some examples of such interactive applications.

Unfortunately, despite their popularity over the Internet, these applications achieve a QoS that is far from what desired. The reason is that multimedia interactive applications impose rigid timing constraints on the traffic they produce and the respect of these timing constraints is difficult in the Internet scenario, as the Internet provides a **best-effort service** to the traffic it carries. In other words, the Internet makes its best effort to move the traffic from sender to receiver as quickly as possible. However, the best-effort service does not make any promises about the end-to-end delay for an individual packet and about the variation of packet delay (network jitter) within a packet stream. This causes the end-to-end delay to be unknown a priori and very variable along the application lifetime and poses serious problems to the supporting of interactive multimedia applications, which are subject to a very critical timing constraint: the overall end-to-end delay, experienced by the application traffic, should be not noticeable to the end-users. The critical role played by this end-to-end delay is described in several studies (for instance, Kurita, Iai & Kitawaki, 1995), which highlight how human perception is strongly affected by this delay. Briefly, these studies point out that the overall end-to-end delay is not noticeable by the human perception if it stays within a threshold, but if the delay goes above this threshold, it becomes noticeable to end-users. This threshold, called NIT (Natural Interaction Threshold) in this paper, represents the limit below which the interactions are well supported. Hence, an interactive multimedia application is well supported if its traffic has an overall end-to-end delay smaller than this NIT threshold along the application lifetime. The value of this NIT threshold depends on the characteristics of the application and on the level of interactivity requested by the end-users (i.e., the more interactive operations are involved, the lower the threshold value should be) (Kurita, Iai & Kitawaki, 1995). For instance, if we consider an interactive application where audio is involved, the delay from when a user speaks until the sound is manifested at the receiving hosts should be less than a few hundred milliseconds. In particular, delays smaller than 150 milliseconds are not perceived by a human listener, delays between 150 and 400 milliseconds can be acceptable, and delays exceeding 400 milliseconds result in frustrating, if not completely unintelligible, voice conversations (Sitaram & Dan, 2000). In summary, interactive multimedia applications impose rigid timing constraints on packet delay and packet jitter, but the network does not provide any timing guarantees. This causes QoS problems to users, who might perceive a quality that is far from what desired.

To investigate the reason of these QoS problems, we focus our attention on the communication sub-system: It is mainly composed of a server (for instance a video server), a client, a network and a mechanism that transports the information from the server to the client and vice versa. Usually, the multimedia stream is remotely available and the network is used to connect client and server. As we previously mentioned, this scenario arises QoS problems, which are due to the type of service provided by the Internet. Although QoS also depends on the quality of the line, the buffering technique and on the distance between client and server, but we can assume that these

dependencies could be simplified. In fact, our proposal is an end-to-end QoS mechanism and hence, as done in other end-to-end QoS approaches, it is possible to simplify some factors. At this level of abstraction, if client and server are directly connected (the network is not involved), no QoS problems happen, as the network does not introduce any delay. For simplicity, we can refer to this latter scenario as the *ideal* scenario for supporting interactive multimedia applications, while the *real* scenario sees the network (as well as the network delays) involved (Figure 1). Using this terminology, we can say that an interactive multimedia application is well supported if the time difference between the real play out and the ideal play out is negligible.

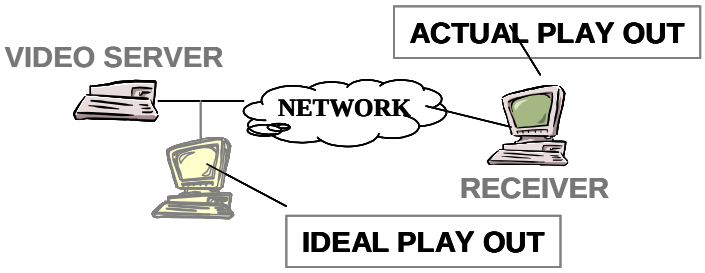


Figure 1: Network Scenario

The contribution of this paper is the proposal of a novel mechanism, called VISP (VIdeo Streaming Protocol), which supports video streaming interactive applications over the Internet. Our goal is to support the video play out in the real scenario, while attempting to simulate the video play out in the ideal scenario. To this aim, we introduce a new metric, called Video Time Difference (VTD), which represents the time difference between the video play out in the real scenario and the video play out in the ideal scenario. In this way, we can know whether the real video play out is close to the ideal video play out and hence, we can know whether an interactive application is well supported or not (note that an application is well supported if the overall end-to-end delay is smaller than the NIT threshold). In particular, during the video play out, our mechanism periodically computes the VTD and compares it with the NIT value: if the VTD is within the NIT value, it means that the network is introducing a delay that is not noticeable to the users; Conversely, if the VTD value is greater than the NIT value the network is introducing a delay that is noticeable by the user. Hence, by periodically checking the VTD we can have a measure of the quality achieved by the application.

If the application is not well supported (the VTD goes above the NIT), our mechanism is in charge of reporting the VTD within the NIT. Simply put, the client computes the time difference between the VTD and the NIT and requires the sender to drop a number of video frames that corresponds to this time difference. As we show, this dropping mechanism reports the VTD within the NIT value and hence reports the real video play out close to the ideal video play out. In essence, our mechanism intentionally modifies the video QoS in order to better support interactions. This has been done because applications with VTD above the NIT may be frustrating or even useless, while if the video QoS modification is very light, the user perceives a video quality that is still good and the application is well supported. To slightly affect the video QoS, we use the dropping frame algorithms proposed by (Furini & Towsley, 2001), which take into account the perceived quality at the user side when selecting the video frames to drop.

The evaluation of our mechanism has been done transmitting several MPEG and Motion JPEG video streams (encoded with different bandwidth requirements) over the Internet. The evaluation measures both the percentage of frame with VTD above the NIT and the perceived video play out quality at the user side. Results show that our mechanism can well support interactive multimedia applications over the Internet, and hence it provides benefits in informing clients through multimedia communications as it allows users to interact with the server in order to obtain a customized stream of multimedia information. On-line games and video-on-demand are examples of interactive multimedia applications that will benefit from using our mechanism.

Background

In literature there are several studies related to multimedia communications. In the following we briefly review some approaches introduced to measure the QoS perception and two possible solutions that aim at improving the QoS perceived by the client: buffering techniques and overlay networks.

QoS Perception

In multimedia communications, the perceived play out quality is perhaps the most important metric in evaluating the effectiveness of a mechanism. The perceived QoS is mainly affected by two characteristics of the multimedia stream: high-bandwidth requirement and a real time delivery constraint. To provide a good QoS, all the packets of the media stream have to be presented at the client with the same temporal relationship as they had at the server side.

This means that each packet has to meet a real time constraint. This contrasts with another characteristic of a media stream: the high bandwidth requirement, which may cause congestion in some network nodes and hence, it may introduce a random delay in the delivery of each packet.

The network jitter (the network delay variation of two consecutive packets) is very critical as it may cause *underflow* or *overflow* problems at the client buffer. In fact, each packet is delivered to the client buffer and the media player retrieves packets from it. If a packet cannot arrive at the client before the scheduled time the media player has no data to retrieve (underflow); if the client buffer is full when a packet arrives, it is discarded (overflow). In both cases, the media play out is affected. In literature there are techniques developed to handle the network jitter, as we better show in the following.

The perceived QoS may also be affected by the synchronization of different media streams (e.g., audio and video) that can compose a multimedia presentation. In this paper, without loss of generality, we suppose that audio, video and other data are multiplexed (i.e., physically combined in one data unit) at the server and de-multiplexed at the client. In this way, the synchronization between different media streams has not to take place. Instead, it is important to focus on the lost, late, corrupted or dropped frames in a video sequence. As suggested by Steinmetz (1996) we can distinguish three kinds of recovery mechanisms: the most sophisticated case tries to compensate the missing frame by presenting them for a longer time. This is certainly the best way as the viewer will not notice the discontinuity if frames are presented for a fraction longer than the regular frame. However, this method is not of practical value with the current video technology, frame rates are fixed and we cannot just adjust frames at will. Another technique is that one frame can be replicated, but the most common method is just to drop the corrupted frame and to continue with the next frame. Despite, this solution seems inadequate due to the possible jerkiness of the video, for a small number of frames there is no significant difference from the doubling technique.

Buffering Techniques

As we previously mentioned, the network jitter plays a critical role in the supporting of multimedia applications. *Buffering* or *smoothing* techniques (Ramjee, Kurose, Towsley & Schulzrinne, 1994) have been proposed to mask the network jitter. In essence, they mask the network jitter by introducing a start-up delay (during which the arrival stream is inserted in the client memory buffer) at the destination.

The client buffer is used to mask the network jitter to the end-users: when the receiver receives the first video frame, the video play out does not immediately start, but the video frame is stored in the local buffer, as well as all the successive arriving frames.

The memory buffer operates using a FIFO discipline and the start up delay is determined by the worst-case jitter and the bit rate of the information stream. The receiver starts the video play out only after the start-up delay (usually few seconds). In this way the receiver should play out continuously the arriving video stream.

Although these techniques are very effective in reducing the network jitter, they cannot be used to support interactive QoS applications, as the introduced start-up delay increases the overall end-to-end delay, a critical measure of interactive QoS applications. In fact, the introduced start-up delay is around few seconds, but, when handling interactive applications, the overall end-to-end delay has to stay within 500ms. This is why in multimedia transmissions it is preferable to reduce the jitter than handling it. For this reason, our approach does not consider buffering techniques, but the video play out immediately starts upon the reception of the first video frame. The network jitter is masked through a different approach, as we show when presenting the characteristics of our mechanism.

Overlay Networks

An overlay network is a network built using an existing network as substrate. It provides fault tolerance and faster recovery as compared to conventional routing techniques (Resilient Overlay Networks) and infrastructure for provisioning QoS within Virtual Private Networks (Virtual Network Service). A particular type of overlay network is a content delivery network (CDN) that is motivated by the congestion that may affect several network nodes. In fact, network congestion usually means delay between the server and the client. Hence, a CDN aims at reducing the number of nodes between the server and the client so that the probability of congestion decreases. Hence, a large group of servers is deployed. By using replicating servers, the content delivery is done from replica-server.

CDNs are used by content providers to improve the QoS provided to end-users as they reduce the load of each server and reduce the user's perceived latency. Their goal is to route around congested networks by using routing techniques that use a set of metrics to direct users to “best” replica.

Although the provisioning of QoS is a common goal for our mechanism and for overlay networks, the two approaches are very different. In fact, overlay networks may facilitate provisioning of QoS by introducing mechanisms (replica servers and/or routing techniques) that try to avoid congestion and hence try to minimize network latency creating a virtual network. On the other hand, our approach only involves the application layer and uses the underlying network as is. Hence, our proposal may take advantage of a network built as a virtual network as our mechanism lies at the very top of the application layer and provides client-server interactivity to the underlying network. If a virtual network is established between our mechanism and the real network, our mechanism may increase its benefits.

Proposed Mechanism

In this section we present the characteristics of VISP (VIdeo Streaming Protocol), a mechanism we propose to support interactive multimedia applications over the Internet. The support of these applications is not trivial in the Internet scenario, as these applications impose rigid timing constraints on packet delay and packet jitter, but the Internet provides a service that doesn’t make any promises about the end-to-end delay for an individual packet and about the variation of packet delay within a packet stream. For this reason, these interactive applications, although very popular over the Internet, achieve a QoS that is far from what desired.

Several studies investigated the effects of the overall end-to-end delay on these interactive applications and it has been highlighted that interactive applications achieve a QoS that can be consid-

ered either good or poor depending on the value assumed by the overall end-to-end delay: if, while supporting the application, this delay stays within a particular threshold, the achieved QoS can be considered good, otherwise it is considered poor. The value of the threshold, called NIT (Natural Interaction Threshold) in this paper, is not fixed but depends on the media considered in the application: for instance, if audio is involved, a good threshold is around 150 ms and an acceptable threshold is around 400 ms. Hence, if the overall end-to-end delay stays within 150 ms, the application is well supported, if the delay goes above 450 ms, the application may be frustrating or even useless.

To measure the achieved QoS, it is important to know, for each video frame that is played out, whether the overall end-to-end delay stays within the NIT or not. To this aim, we identified two possible scenarios when playing out a video: a) the network is not involved, as the video is locally available (in this case the client and the server are directly connected) and b) the network is involved, as the video is remotely available (client and server are connected through the Internet). In this paper we refer to the play out in the former scenario as the *ideal* play out, while the play out in the latter scenario is referred to as the *real* play out. Hence, the overall end-to-end delay corresponds to the time difference between the real and the ideal play out. This means that, to measure the achieved QoS, it is important to know, for each video frame that is played out, whether this time difference, called VTD (Video Time Difference), stays within the NIT or not.

For example, let us consider a common situation when delivering a video stream to a client. The video stream is usually encoded with a certain number of frames per second (fps), say δ. This means that the sender transmits video frames with a time difference of $\alpha=1/\delta$ time units from each other. If the first video frame is marked with timestamp *t*, then the *j-th* frame is marked with timestamp $t+(j-1)\alpha$. On the other side, the client, upon the reception of the first video frame, starts playing out the video stream, displaying the video frames with a time difference of $\alpha=1/\delta$ time units from each other. It is to note that the video play out immediately starts because the application poses rigid timing constraints on the overall end-to-end delay and hence, it is not possible to use *buffering* techniques that mask the network delay by delaying the initial start-up of the video play out (this increases the overall end-to-end delay).

In Figure 2 we show the difference between a QoS network (i.e., a network that provides some guarantees, such as sufficient bandwidth, low packet-loss and end-to-end delay) and the Internet. In QoS networks (Figure 2a), the VTD is kept constant along the application lifetime. Unfortunately, in networks that provide a best-effort service (Figure 2b), like the Internet, timing-

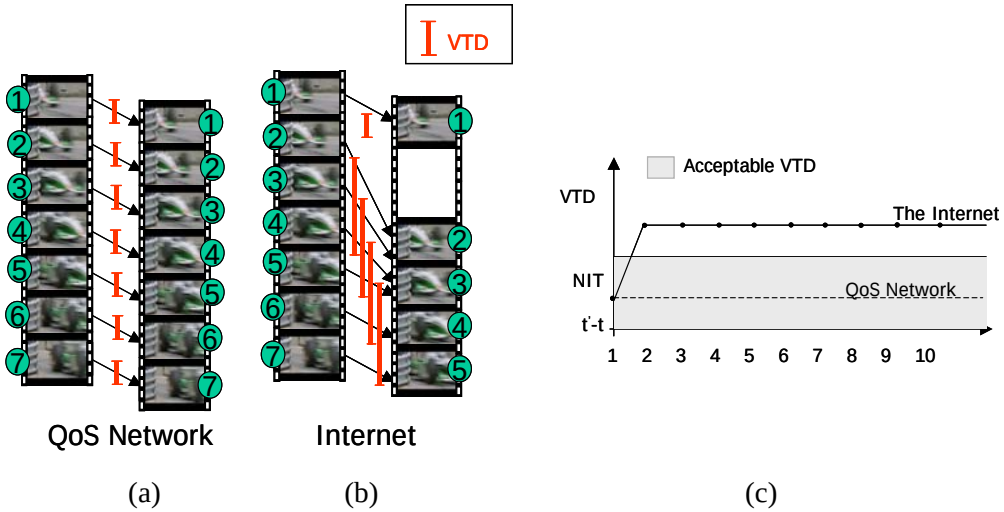


Figure 2: The effect of the network delay in video streaming applications: (a) QoS Network; (b) The Internet; (c) The VTD is affected

guarantees are not provided and video frames are subject to the variability of the network delay: while delivering the video stream, the traffic may be delayed, causing the VTD to increase. If this value goes above the NIT value, the overall delay is noticeable to the end-user.

For instance, if frame 1 is played out at time *t'*, then frame 2 should be played out at time $t'+\alpha$, frame 3 at time $t'+2\alpha$, and, generally speaking, frame *j* should be played out at time $t'+(j-1)\alpha$. If frame 2 is delivered later than expected (for instance it is delivered between $t'+2\alpha$ and $t'+3\alpha$), the receiver has no frame to play out at time $t'+\alpha$ and at time $t'+2\alpha$. This means that the video play out will freeze up to time $t'+3\alpha$, when it is resumed playing out frame 2. In this case the network jitter compromised the continuity of the video play out and, further, the delay experienced by frame 2 affects the play out time of all the successive frames. In fact, even though all the successive frames are delivered "in time", their play out is delayed by the network problems experienced while transmitting frame 2. This delay increases the time difference between the real play out and the ideal play out. The effects of this situation are presented in Figure2c, where the VTD is computed for each frame that is played out. A hypothetical NIT value is also depicted in order to better highlight frames with a VTD below or above the acceptable limit. Since the first video frame is played out at time *t'* and its timestamp is *t*, it follows that $VTD(1)=t'-t$. In QoS networks, the VTD remains constant along the application lifetime, but in best-effort networks, the VTD changes from time to time. In our example, since frame 2 arrived later than expected and it was played out at time $t'+3\alpha$, its VTD is equal to: $VTD(2)=t'-t+3\alpha$. Note that, even though all the successive frames are delivered without any problem, the VTD of the successive frames is affected by the network problem experienced while transmitting frame 2. If this value goes above the NIT (i.e. VTD(2)>NIT) then the VTD of all the successive frames is affected too ($VTD(j) \geq NIT$, for each $j \geq 2$). Needless to say, serious problems arise if the supported application has interactive features, as all the frames, but the first, have a VTD above the acceptable NIT. In general, if, among all the frames that compose the video stream, the percentage of frame with a VTD above the NIT is negligible, the quality achieved by the application is acceptable, but if this percentage is considerable (for instance, more than 5%) the quality may be frustrating.

For this reason, our mechanism reports the VTD within the NIT, by acting on the video QoS. In fact, by intentionally acting on the video QoS and by selecting the video frames to drop in a smart way (for instance using the dropping frames algorithms proposed in (Furini & Towsley, 2001)), the interactive application is well supported as the VTD is kept within the NIT and the user perceives a video play out quality only slightly affected. Hence, it is preferable to intentionally act on the video QoS and to report the VTD within the NIT, than having the VTD above the NIT for most of the application lifetime.

VISP Architecture

Our mechanism is composed of two parts: one is located at the receiver side and periodically measures the VTD for the video frames that are played out; the other is located at the server side and is in charge of marking each video frame with its ideal play out time and of reporting the VTD within the NIT value upon client request. Prior the transmission, a client-server clock synchronization mechanism is activated to synchronize the client and server clocks (in this paper we do not focus on this point, as several mechanisms and/or technologies can be used to this aim). Once the clocks are synchronized, the server can stream the video to the client and our mechanism measures the VTD through a timestamp mechanism. If the VTD is above the NIT, the client sends a message to the server, informing it about the time quantity (expressed in number of video frames) that the VTD exceeded the NIT value. When the server receives this message, it reports the VTD within the NIT, by dropping the necessary number of video frames.

In the following we provide a detailed description of our mechanism, but first we introduce three definitions that will be used throughout the paper.

Definition. The clock at the sender side is denoted with T_S, and $T_S(i)$ represents the ideal play out time of the frame *i*.

Definition. The clock at the receiver side is denoted with T_R, and $T_R(i)$ represents the (real) play out time of the frame *i* at the receiver side.

Definition. Let us consider a video frame *i*. The Video Time Difference of a frame *i*, denoted with VTD(*i*), is defined as the difference (in time) between the real play out of the considered frame, $T_R(i)$, and its ideal play out time, $T_S(i)$. Hence, the VTD of a frame *i* is equal to $VTD(i) = T_R(i) - T_S(i)$.

Video Stream Transmission

The mechanism at the server side is in charge of marking each video frame with a timestamp. In this way all the video frames are provided with a temporal order, so that the receiver can find out whether video frames arrive in the wrong order. Further, this timestamp is used by the client to compute the VTD of each video frame that is played out.

The timestamp, associated to each video frame, represents the ideal play out time of such frame. The ideal play out time is easily computed. In fact, the video stream is composed of a sequence of video frames that must be displayed within a fixed time from each other (see Table 1). For instance, the video may be encoded with 12 or 24 frames per second (fps). Hence, each video frame must be displayed 1/12 or 1/24 of a second within of each other, respectively. If the number of fps is denoted with δ and if the first video frame is marked with time *t*, then the second video frame is marked with time $t+\alpha$, where $\alpha=1/\delta$ and, generally speaking, a frame *i* ($i > 1$) is marked with $T_S(i)=t+(i-1)\ \alpha$.

Table 1: Video Transmission Algorithm

1. The timestamp of the first video frame represents the time at which the video frame is transmitted. If we denote this time with *t*, it follows that the first video frame is marked with $T_S(1)=t$.
2. A frame *i* ($i > 1$) is marked with $T_S(i)= T_S(i-1)+\alpha$, where $\alpha=1\delta$. Hence, a frame *i* ($i > 1$) is marked with $T_S(i)=t+(i-1)\ \alpha$

Note that the time spent for creating, as well as the time spent for reading a video frame, is not considered in the above analysis, as, due to the hardware technology, this time is negligible.

Video Play Out

The mechanism at the receiver side is in charge of displaying the video frames and of computing the VTD for each displayed frame. In essence, the receiver retrieves video frames from the network, orders and plays out them according to their timestamp. The algorithm is presented in Table 2. Since no buffer-

Table 2. Video Play out Algorithm

1. Video play out starts when the first video frame arrives at the receiver side, say at time t' and the frame is immediately played out;
2. The receiver plays out the frames at fixed period (i.e., one frame every $\alpha= 1/\delta$ time units;
3. Among the frames present in the local buffer, say k frames, it is selected (for play out) the frame with the lowest timestamp (i.e. a frame i is selected if TS(i)= min(TS (j) for each frame j in the buffer);
4. Once selected, a frame i is removed from the buffer and is played out at time TR (i)= TR (i-1)+ $z\alpha$ ($z\geq1$) only if: a) TR (i) $\geq$ TS (i) and b) TS (i) > TS (prev(i)), where prev(i) is the last frame that has been played out.
   If conditions a) and b) are not met, then frame i is discarded and a new frame selection must be done (by applying rule 3);

ing delay is introduced, upon the reception of the first video frame (suppose at time *t'*), the video play out is started at time $T_R(1)=t'$. The receiver should play out the arriving frames within a fixed period from each other (i.e., one frame every $\alpha=1/\delta$ time units); the video frame at the head position in the local buffer (i.e., the frame with the lowest timestamp) is candidate for the play out. The candidate frame, suppose frame *i*, is played out only if it has a timestamp lower than the timestamp of the frame that has been played out most recently (i.e., $T_S(i)>T_S(\text{prev}(i))$, where prev(*i*) is the last played out frame). This is done in order to avoid the play out of a frame *i* older than the frame that has been already played out (i.e., frame *prev(i)*). If $T_S(i)<T_S(\text{prev}(i))$ the frame *i* is discarded and another frame is picked up from the local buffer. If $T_S(i)>T_S(\text{prev}(i))$ the candidate video frame *i* is played out at time $T_R(i)$ only if $T_R(i) \geq T_S(i)$, with $T_R(i)=T_R(i-1)+z\alpha$ ($z\geq 1$); note that in the ideal scenario, z=1 along the application lifetime, as the video frames are always played on-time. Conversely, in the real scenario, the network may deliver video frames later than expected and hence it may happen that a video frame is not available when needed. If its play out is delayed, z is greater than one (z>1).

In addition to the video play out, the mechanism located at the receiver side measures the VTD for each video frame that is played out. If the VTD is above the NIT, the mechanism computes the time difference between the VTD and the NIT and translates this time difference into a number of frames. Then it asks the sender to drop the computed number of video frames. For instance, if the video is encoded with 24 fps, a frame lasts on the user's screen 41 ms); if the time difference between the VTD and the NIT is equal to 100 milliseconds then the number of frames to drop is equal to 3. When this message reaches the server, the dropping frames mechanism is activated.

Dropping Frame Mechanism

The goal of the dropping frame mechanism is to report the VTD within the NIT value. This mechanism is located at the server side and is activated upon client request. It is based on a Theorem proved in (Furini & Roccetti, 2002), which states that the VTD can be reduced of λ time units, by dropping a number of frames, say *k*, that corresponds to λ time units (i.e. $k\alpha = \lambda$), where $\alpha=1/\delta$ and δ denotes the number of frames that must be played every second. In essence, since α is known by both the client and the server, λ is measured by the client and transformed into *k*, it is sufficient, for the client, to send the *k* value to the sender. On the other side, the sender can drop *k* frames to report the VTD within the NIT.

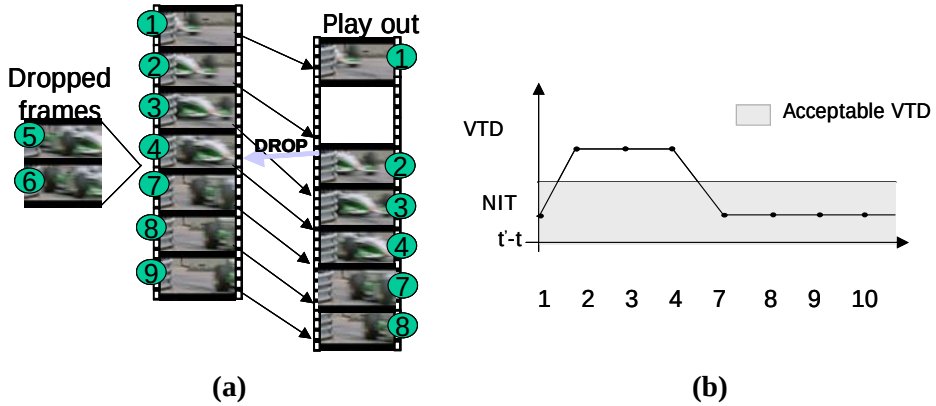


Figure 3: Video play out by using our mechanism. (a) if the VTD goes above the NIT, the server is asked to drop frame in order to report the VTD within the NIT. (b) The effect of our mechanism of the VTD.

By reporting the VTD within the NIT, we avoid the application from being frustrating. In fact, if the VTD goes above the NIT, the client-server interactions are seriously compromised. Hence, it is preferable to affect the video Qos (and to keep the VTD within the NIT value), than having the VTD above the NIT value. Further, it is worth noting that a good selection of the frames to discard (Furini & Towsley (2001), only slightly affects the quality of the perceived video.

In Figure 3 we show the effects of our mechanism: the scenario is the same of Figure 2b, but in this case, when the client finds out that the VTD goes above the acceptable value (for instance when playing out frame 2), it computes the number of frames (*k*) that has to be discarded in order to report the VTD within the NIT. Let us suppose (Figure 3a) that it is necessary to drop 2 frames, the sender discards (i.e., it does not transmit), frame 5 and frame 6. This means that, just after frame 4, the sender transmits frame 7, frame 8 and so on.

In Figure 3b we show the effects of our mechanism on the VTD. The benefits introduced by our mechanism starts when playing out frame 7. In fact, if our mechanism is not used (Figure 2c), frame 7 is played out with VTD(7) greater than NIT, but if our mechanism is used (Figure 3b), VTD(7) is within the NIT. Moreover, using our mechanism, all the frames transmitted after frame 7 are within the NIT value.

This means that our mechanism may introduce considerable benefits in supporting interactive video streaming applications allowing user to interact with the server in order to obtain a customized stream of multimedia contents.

In the next section, we evaluate our mechanism in the Internet scenario by transmitting several video streams between a server and a client.

Experimental Scenario

In this section we evaluate our proposed mechanism using the experimental scenario depicted in Figure 4: a client is connected to a server using the Internet. Two different Internet paths are used: one is 8-hops long and has an average round trip time of 80-90 ms; the other is 9-hops long and has an average round trip time of 300-400 ms.

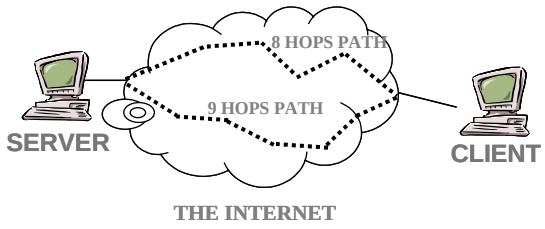


Figure 4: Experimental Scenario.

The traffic sent from the server to the client is a video stream. This type of traffic has been selected as it is subject to two critical requirements for the Internet scenario: high bandwidth requirements and stringent timing requirements. Other types of traffic (for instance the one produced by on-line games) do not require high bandwidth. Hence, by using video streams, we set up a very critical scenario for evaluating our mechanism. Several video streams with different characteristics (cartoons, music videos, news, movies) are used. The bandwidth requirements of these video streams vary from 274 kbps to 1.5Mbps. These videos have been streamed during different network conditions (peak hours, office hours, evening).

In such a scenario, the server streams a video to the client and, for each video trace, we compute a) the percentage of video frames with VTD above the NIT with and without using our mechanism and b) the perceived quality of the video play out.

The first investigation is done to check whether the overall network delay is noticeable by the end-user or not. This investigation is important to perform as the network delay affects the QoS perceived by the client. To minimize the effects on the perceived QoS the VTD experienced by each video frame should stay within the NIT threshold. For this reason, we measure the VTD for

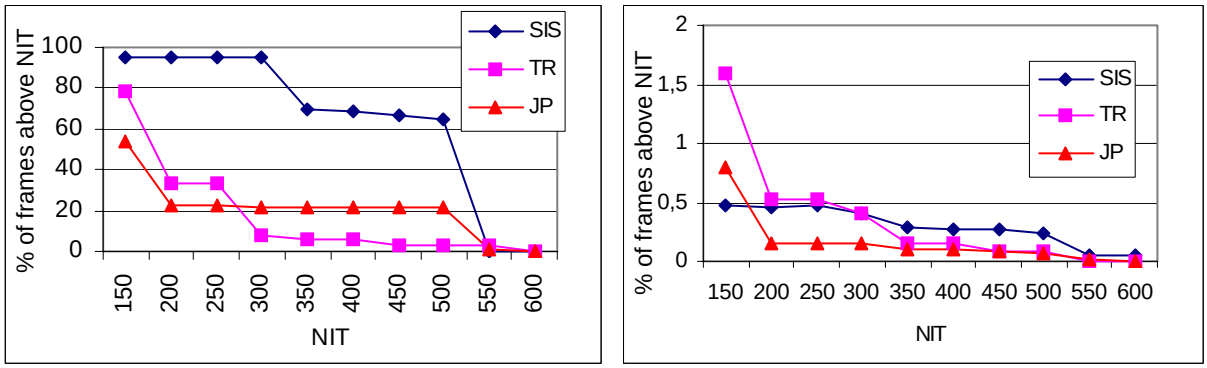
each frame that is played out and we compute the percentage of video frames with VTD above the NIT. Since this percentage depends on the NIT value and since the NIT value is not fixed but it is application dependent, for each video stream we perform several tests using different values of the NIT. In particular, the NIT value ranges between 150 ms (good interactivity) and 600 ms (very poor interactivity). To investigate the benefits introduced by our mechanism we compare results obtained from transmitting a video stream when our mechanism is used and when it is not.

The second investigation is related to the video play out quality perceived by the user and it is done to evaluate the dropping approach of our mechanism. In fact, to keep the above percentage very small, our mechanism drops video frames and hence it affects the quality of the video play out. Several studies highlight how difficult is to well define the perceived video quality as this depends on the characteristics of the human vision system. For this reason, different approaches may be used to this aim. An easy way is to count the number of dropped video frames, but this number is not much related with the perceived quality at the client side. A common way is to use a cost function, as this approach gives a better measure than the simple number of dropped video frames. However, this approach depends on how well the cost function is defined. In this paper we use this approach to accounting for the perceived video play out quality and we use the cost function proposed in (Zhi, Nelakuditi, Aggarwa & Tsang, 2000), which penalizes frame dropping mechanisms that drop neighboring frames (Figure 5). Briefly, this cost function takes two aspects into consideration: the length of a sequence of consecutive discarded frames and the distance between two adjacent, but non-consecutive, discarded frames. It assigns a cost c_j to each discarded frame j, depending on whether it belongs to a sequence of consecutive discarded frames or not. If frame j belongs to a sequence of consecutive discarded frames, the cost is l_j if the frame j is the l_j-*th* consecutively discarded frame in the sequence. Otherwise the cost is given by $1+1/\sqrt{d_j}$, where d_j represents the distance from the previous discarded frame.

$$C_j = \begin{cases} l_j & \text{if j is the } l_j th \text{ consecutevely discarded frame in a sequence} \\ 1+\dfrac{1}{\sqrt{d_j}}, & \text{where } d_j \text{ represents the distance from the previous discarded frame} \end{cases}$$

Figure 5. Cost function used to compute the perceived QoS.

Before presenting the obtained results, it is worth describing the video streams used in this evaluation. Two sets of video traces are used: one set is encoded with an intra-frame technique (Motion JPEG) and the other set is encoded with an inter-frames technique (MPEG). The type of encoding is important because, with inter-frames encoded videos, a *domino* effect may happen while discarding a single video frame (i.e., a discard of a frame may lead to the impossibility of decoding other video frames). Hence, when the server discards video frames it has to take into account all the dependency rules among the video frames. Conversely, if the video streams are encoded with an intra-frame technique, it is possible to discard frames without causing the domino effect. Hence, the selection of the video frames that have to be discarded is very important as the dropped video frames affect the perceived video play out quality. In this paper we don't propose any dropping algorithm, but we use the frame dropping algorithms proposed in (Furini & Towsley, 2001).



(a) **(b)**

Figure 6: Motion JPEG video streams. The percentage of video frames with VTD above the NIT. (a) without using our mechanism. (b) using our mechanism.

In Figure 6 we present results obtained from transmitting a set of Motion JPEG video traces (*Sleepless in Seattle*, *Total Recall* and *Jurassic Park*) encoded with 12 frames per second. In Figure 6a we show the percentage of video frames with VTD above the NIT when the video stream is transmitted without using our mechanism. It is possible to note that the percentage is considerable for all the NIT values and it decreases as the NIT increases. The percentage assumes reasonable values for NIT values greater than 550 ms, but with this NIT value the interactions are already compromised and the application is frustrating (the real video play out is not close to the ideal video play out and hence the overall end-to-end delay is noticeable to the end-user). In Figure 6b, we present results obtained from transmitting the same video traces, but while using our mechanism. Also in this case the percentage of video frames with VTD above the NIT is presented. It is possible to note that the percentage is much smaller than when our mechanism is not used. The maximum value (1.6%) is obtained for *Totall Recall* with an NIT of 150 ms. Hence, our mechanism masks the network problems to the end-user, who does not notice the overall end-to-end delay and hence the application is well supported for all the NIT values. The evaluation of our mechanism is completed with the investigation about the perceived video play out quality. In Figure 7 we present results obtained from applying the previously described cost function to the video play performed at the client side. Results are expressed as a percentage of additional cost for each video stream that is played out. The additional cost is presented because when the video is streamed to the client it is subject to two types of dropping: one is intentionally done by our mechanism, and the other is done by the network (i.e., packet loss). By presenting the percentage of additional cost, we exactly know the cost introduced by our dropping mechanism. Needless to say, the smaller the additional cost is, the lower the perceived video quality is affected. Figure 7 shows that the percentage of additional cost is very small, with the exception of a 12% value introduced for the Jurassic Park video for 150 ms NIT. This cost is necessary to obtain a percentage of frames with

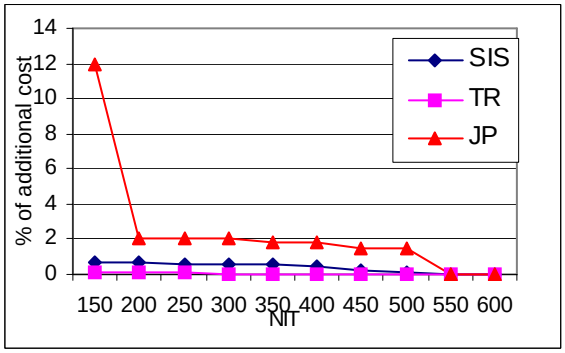
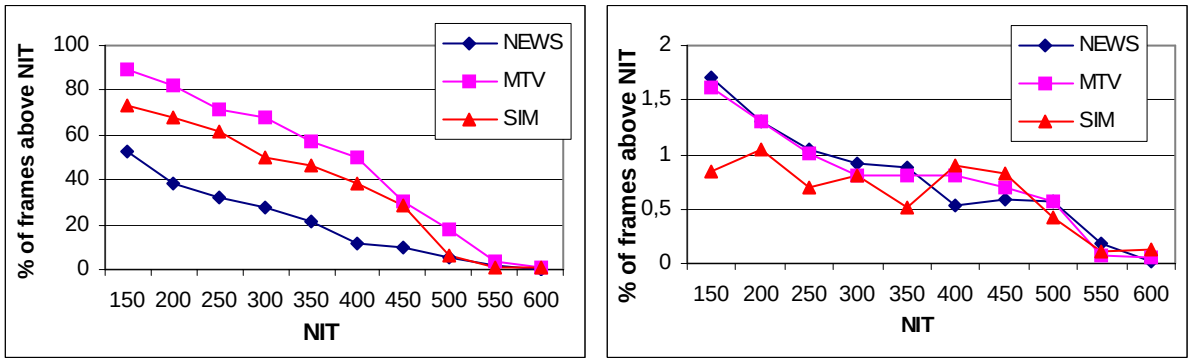


Figure 7: Percentage of additional cost introduced by our mechanism.

VTD above the NIT of 0.8%. If our mechanism is not used, the percentage is more than 50%. Hence, we think it is better to have a slightly worse video play out quality with very good client-server interactions, than having a very good video play out quality with very bad client-server interactions.



(a) **(b)**

Figure 8: The percentage of video frames with VTD above the NIT. (a) results obtained without using our mechanism. (b) results obtained using our mechanism.

In Figure 8 we present results obtained from transmitting a set of MPEG video traces (*News*, *Mtv* and *The Simpsons*) encoded with 12 frames per second. In Figure 8a we show the percentage of video frames with a VTD above the NIT when the video streams are transmitted without using our mechanism. It is possible to note that the percentage is considerable for all the NIT values and it assumes reasonable values for NIT values greater than 550 ms. This means that the interactions are not well supported, as the overall end-to-end delay is noticeable to the end-user. In Figure 8b, we present results obtained from transmitting the same video traces but while using our mechanism. Also in this case the percentage of video frames with VTD above the NIT is presented. It is to note that for each NIT value we present an average percentage of all the dropping algorithms used by the server (namely, the dropping algorithms proposed in (Furini & Towsley, 2001)). Also for the MPEG video traces, the percentage of video frames with VTD above the NIT is much smaller when our mechanism is used: the maximum value (1.6%) is obtained for the *News* with an NIT of 150 ms. Hence, our mechanism is effective in supporting interactive applications as it mask the overall end-to-end delay to the end-user. It is interesting to note that by using our mechanism, it may happen that the percentage of video frames with VTD above the NIT does not decrease if the NIT increases (see for example The Simpsons Figure 8b). This behavior is due to the type of video encoding (inter-frames) and to the dropping frame policies used (the presented values are an average value obtained from all the values produced by the different dropping frames policies used). In fact, the dropping mechanism may wait for a particular type of frame, instead of immediately drops the frame upon the client request. This is the reason why, sometimes, it may happen that the percentage of video frames with VTD above the NIT does not decrease is the NIT

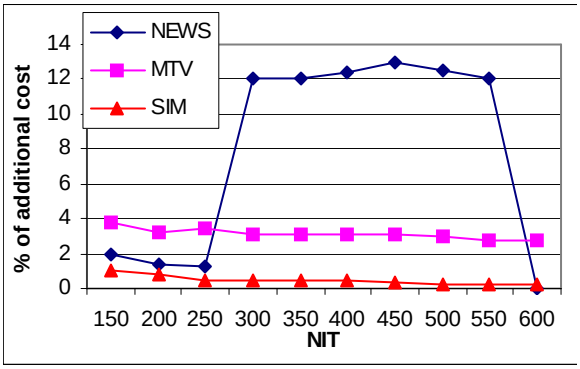


Figure 9: Percentage of additional cost introduced by our mechanism.

increases. Once again, to complete the evaluation of our mechanism, in Figure 9 we present the percentage of additional cost introduced by our mechanism. In this case, for each NIT, we present an average value of the discarding policies used. Also for the computed additional cost, it is interesting to note that sometimes (see for example, the news and the mtv video traces, Figure 9) the percentage of additional cost does not decrease if the NIT increases. Once again, this is due to the type of video encoding and to the dropping frames policies, which may be forced to drop a video frame that causes a domino effect on several other video frames. This behavior confirms how difficult is to select video frames to drop if the video is inter-frames encoded. However, despite these problems, our mechanism keeps the percentage of video frames with VTD above the NIT very small and introduces an acceptable QoS cost in the video play out quality.

Conclusion

In this paper we proposed a new approach for supporting interactive multimedia streaming applications over the Internet. We showed that the support of these applications is critical in the Internet, as this network provides a best-effort service to the traffic it carries, while interactive applications requires the network to support stringent timing requirements.

We highlighted that, to be well supported, interactive applications need the overall end-to-end delay to be not noticeable to end users and that a time threshold (NIT, Natural Interactive Threshold) denotes the limit below which the overall end-to-end delay is not noticeable (and hence the interactive application is well supported) and above which the overall end-to-end delay is noticeable (and hence the interactive application is not well supported). It is worth noting that this overall end-to-end delay is not merely the end-to-end network delay, but it can be seen as the time difference between the real video play out (client and server connected through the Internet) and the ideal video play out (client and server directly connected and network delays not involved). For this reason, we introduced a new metric, called VTD (Video Time Difference), which provides an easy way to know whether an interactive application is well supported or not: in fact, if the VTD stays within the NIT value, the overall end-to-end delay is not noticeable to end user and hence the interactive application is well supported; if the VTD goes above the NIT the application is not well supported as the overall end-to-end delay is noticeable to the end user.

Our mechanism aims to keep the VTD within the NIT and this is achieved by using dropping frame algorithms that report the VTD within the NIT value when the VTD exceeds the NIT: Our mechanism intentionally modifies the video QoS to better support interactions between the user and the server. We showed that the QoS modification is very light (we used dropping frames algorithm designed to this aim) and hence it is preferable to have a video play out with the QoS slightly modified than to have a frustrating or useless interactive application where interactions are not possible due to the high overall end-to-end delay.

Our mechanism has been evaluated in the Internet scenario and results showed the introduced benefits. In particular, we showed that in the Internet the percentage of video frames with VTD above the NIT is very high, causing the applications to be frustrating or even useless. Conversely, by using our mechanism this percentage is kept very small and the video play out quality in only slightly modified (to this aim we investigated the perceived quality of the video play out at the user side).

Hence, if our mechanism is used, the user can interact without any problem with the multimedia provider, which can provide a customized stream of multimedia contents. Several entertainment or edutainment applications may benefits from using our approach (for instance video on-demand and on-line games) as they can provide users with a customized multimedia stream of information.

We are currently investigating different algorithms regarding the VTD handling. To this aim we are investigating the relationship between the NIT and the video contents, in order to dynamically set the NIT along the application lifetime (for instance by investigating the motion of the video it is possible to select video frames to drop) and we are also studying how to reduce the QoS modification introduced by the dropping frame algorithms (for instance algorithms may individuate portions of the video stream less important than other and may drop video frames in these portions).

References

Furini, M., & Roccetti, M., (2002) Design and Analysis of a Mechanism for supporting Interactive Video Streaming Applications over the Internet, *Proceedings of the 6th Internet and Multimedia Systems and Applications (IMSA2002)*, Kauai HI-USA, pp. 324-329.

Furini, M., & Towsley, D. (2001) Real-Time traffic transmission over the Internet, *IEEE Transaction on Multimedia*, vol. 3, No. 1, 33-40.

Kurita, T., Iai, S., & Kitawaki, N. (1995) Effects of transmission delay in audiovisual communication, *Electronics and Communications in Japan*, 77(3):63—74.

Ramjee, R, Kurose, J., Towsley, D., & Schulzrinne, H. (1994) Adaptive Playout Mechanisms for packetized audio applications in wide-area networks, *IEEE Computer and Communications Societies on Networking for Global Communications*, pp. 680-688.

Sitaram, S., Dan, A. (2000) Multimedia Servers: Application, Environments, and Design, *Morgan Kaufmann Publishers*, San Francisco.

Steinmetz, R. (1996) Human Perception of Jitter and Media Synchronization, *IEEE Journal on Selected Areas in Communications*, Vol. 14, pp. 61-72.

Zeng, W., & Yu, H. (1999). Informing Client through Networked Multimedia Information Systems: Introduction to the Special Issues, *Special issue on Multimedia Informing Technology*, Vol.2, No. 4.

Zhi-Li Zhang, Srihari Nelakuditi, Rahul Aggarwa, Rose P. Tsang (2000). Efficient Server Selective Frame Discard Algorithms for Stored Video Delivery over Resource Constrained Networks, *Journal of Real-Time Imaging*.

Biographies

Marco Furini received the degree and the Ph.D. degree in computer science from the University of Bologna, Italy, in 1995 and 2001, respectively. He is currently a faculty member of the Computer Science Department at the Piemonte Orientale University.

From August 1998 to May 1999, he visited the Department of Computer Science, University of Massachusetts, Amherst. His scientific interests include multimedia communication systems, QoS issues over IP-Networks and architectures for e-music distribution.

Marco Roccetti is a professor with the Department of Computer Science of the University of Bologna. For the past 15 years he has held different research and management positions at the University of Bologna. He is a member of a number of international conferences program committees and is serving as Editor-in-Chief for the E-Letter publication of the Multimedia Technical Committee of the IEEE Communications Society. His research interests include digital audio and video for multimedia communications, wireless multimedia and network-centric computer-based entertainment.