Interactive Video Streaming Applications over IP Networks: An Adaptive Approach

Marco Furini[1] and Marco Roccetti[2]

[1]Computer Science Department
University of Piemonte Orientale
Spalto Marengo 33
15100 Alessandria Italy
Tel. +390131287446
furini@mfn.unipmn.it

[2]Computer Science Department
University of Bologna
Mura Anteo Zamboni 7
40127 Bologna Italy
Tel. +390512094503
roccetti@cs.unibo.it

TR-INF-2003-07-05-UNIPMN

Abstract

One of the key problems for delivering interactive video applications over IP networks is to keep the end-to-end delay below a pre-defined and application dependent threshold along the application lifetime. Since IP networks only provide a best-effort service, the overall end-to-end delay may go above the threshold, and hence may compromise the QoS perceived by the users. A start-up delay solution, effective for non-interactive QoS applications, cannot be used for interactive applications, as it compromises the achieved QoS by introducing an additional delay to the overall end-to-end delay. In this paper we propose a new approach, which adapts the video play out to the network conditions, in order to support these applications. Instead of using a start-up delay, our mechanism keeps the end-to-end delay within the acceptable threshold, by modifying the video QoS. A QoS evaluation is done to investigate the effects of the QoS modification introduced by our technique. The mechanism evaluation shows that our approach is effective, as the QoS is slightly affected and the end-to-end delay is kept within the threshold along the application lifetime.

KEYWORDS: Adaptive streaming of Multimedia Applications, Multimedia over Packet-based Networks, Adaptive Multimedia Communication Systems, Traffic Management and Control, QoS evaluation.

1 Introduction

Video streaming applications over IP networks are becoming more and more popular, but, despite their popularity, they achieve a QoS that is far from

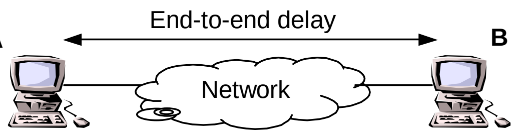


Figure 1: Model of a networked application.

what desired. Videoconferencing, distance learning, Video on Demand, video telephony, on-line games are examples of these applications.

QoS difficulties are mainly due to the traffic produced by these applications that is time-dependent and may require high network capacity. For instance, a video streaming application is very bandwidth consuming: even using compression algorithms such as MPEG [1], Motion JPEG [2] or H.261 [3], the compressed video stream can require high network capacity compared to the (usually) available in the Internet. In addition, to provide the applications with the necessary QoS, two time-constraints have to be met: minimal communication delay and unnoticeable network jitter to the user [5, 6]. These time-constraints are very critical to provide with the best-effort service provided by IP networks.

A sub-set of these QoS applications, called *interactive* applications, enables natural interactions (i.e., more life-like as possible) among end-users. These interactive applications pose a constraint on the end-to-end delay (it should not be noticeable to the end-users) and hence they are more difficult to support than non-interactive applications. In fact, this constraint is not considered in non-interactive applications: for instance, an audio broadcast application (e.g., an Internet radio) may have a large end-to-end delay, as users do not interact.

The respect of this constraint on the end-to-end delay is hence fundamental for interactive QoS applications. Several studies [5, 7, 8, 9, 10, 11, 12] investigated the effects of this delay on human perception and they showed that interactive applications are well supported if the end-to-end delay is kept within a threshold along the lifetime of the application. Conversely, if the end-to-end delay goes above this threshold, the interactions between end-users are seriously compromised. Hence, the threshold represents a bound for the human perceptions: below this bound the users do not perceive the end-to-end delay and hence the interactions are well supported; above this bound the end-to-end delay is noticeable to the end-users and hence the interactions are not well supported. Throughout this paper we denote this threshold with NIT, Natural Interaction Limit. It is to note that the value of this threshold is not fixed [8], but depends on the characteristics of the application and on the level of interactivity requested by the end-users. For instance, a threshold of 150 ms ensures full satisfaction to the end-users of an interactive audio applications[6]. This means that, if the end-to-end delay goes above 150 ms the users will experience a bad service, while for values lower than 150 ms, the users can interact without any problems.

The end-to-end delay, as pointed out by *Baldi* and *Ofek* [13], is composed by different components: the *processing* delay (the time spent at the end-hosts to compress/decompress video frames), the *network* delay (the time needed to move data from one end-host to the other end-host) and the *synchronization* delay at the receiver side (this delay is introduced in order to cancel the network delay jitter).

Among the components that affect the end-to-end delay, the network delay is the most variable. It is essentially composed by two components: *propagation*, or transmission, delay and *queuing* delay. The propagation delay can be easily computed as it depends on the network capacity and on the size of the data to transmit. For instance, to transmit f bytes over a network with a capacity of C bytes/sec, the ratio f/C gives the necessary time to transmit f bytes. Conversely, the queuing delay is very variable and unknown a-priori. In fact, data travel from source to destination along a path, composed of links and routers. In best-effort networks, this path is usually shared among traffic generated by other applications and it may happen that a network resource along the path is busy, causing the data to be delayed until the resource is available.

The network delay is very variable (network jitter) along the application lifetime and, for this reason, it is necessary to design an adaptive mechanism that takes into consideration the network conditions. Otherwise, the receiver may experience QoS problems. For instance, in a video streaming application the receiver should continuously play out the arriving stream, according to a pre-defined number of frames every second. Due to the network jitter, some video frames may be delivered later than expected, causing problems to the receiver's play out as video frames may not be available for play out when needed. This could cause video play out interruptions (a common situation for users that watch video streams while being connected to the Internet through a low capacity modem).

In literature there are mechanisms that mask the network jitter using *buffering* or *smoothing* techniques [9, 14, 15, 16]. Roughly, these mechanisms use a start-up delay to mask the network jitter. While being effective in supporting QoS applications, these techniques cannot be used to support interactive applications, as they increase the overall end-to-end delay. In the next section, we present a better overview of these mechanisms.

Before introducing our mechanism, we first note that when watching a video stream on a computer, there are two possible scenarios: i) the video stream is locally available at the user side (either the video is generated with a webcam, or it is stored on the user's hard-disk or on a CD-Rom, DVD, and so on) or ii) the video stream is located somewhere in the network through which the video application transmits the video stream at the sender side and retrieves the video stream at the receiver side.

From the user perspective, the ideal scenario to perform interactive operations is when the video is locally available. This avoids the use of the network (i.e., no network delay). In Fig. 2 we show a possible scenario for a video application. A video server is in charge of transmitting a video stream into the network. This stream is delivered to the receiver where the actual play out of the video stream is done. In this scenario, the network delays the transmission of the video stream as well as the interactive requests of the user at the

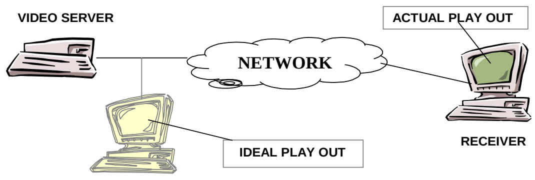


Figure 2: Network scenario while transmitting a video stream between a sender and a receiver. The ideal video play out and the actual video play out are highlighted.

receiver side. Needless to say, if the network delays these data with a value greater than NIT, the interactivity between the end-user and the video server is compromised. The ideal scenario, to avoid these possible problems, is when the user is directly connected to the video server. Throughout the paper, we refer to this scenario as the *ideal* scenario for providing interactions and to the actual scenario (where video is stored somewhere in the network) as the *real* scenario.

The contribution of this paper is to propose an adaptive mechanism that supports the video play out in the real scenario, while attempting to simulate the video play out in the ideal scenario. Our mechanism does not use any start-up delay, but the receiver begins the video play out upon the reception of the first frame. Our goal is to maintain the end-to-end delay within the acceptable threshold (i.e., below the NIT value), while allowing the receiver to continuously play out the video stream.

In essence, this means that the ideal and the actual video play out are synchronized. If the network jitter compromises this synchronization, our mechanism adapts the supported video application to the new network scenario, by modifying the video QoS of the delivered stream in order to re-synchronize the actual and the ideal play out.

The mechanism has been evaluated through several simulations, performed using real network delay traces (obtained transmitting Motion JPEG and MPEG video traces over our department LAN and over the Internet). In addition to the mechanism evaluation, we also measure the perceived QoS at the user's side in order to investigate the QoS modifications introduced by our mechanism. Results obtained show that our mechanism is well suited for supporting interactive video applications over IP networks, as the actual video play out is kept very close to the ideal video play out and the perceived QoS is only slightly modified.

The remainder of this paper is organized as follows. In section 2 we present our mechanism and its properties. We also highlight benefits of using our mechanism. In section 3 we present characteristics of the experimental scenario we use to test our mechanism. Results obtained from evaluating our mechanism are presented in 4 while conclusions are drawn in section 5.

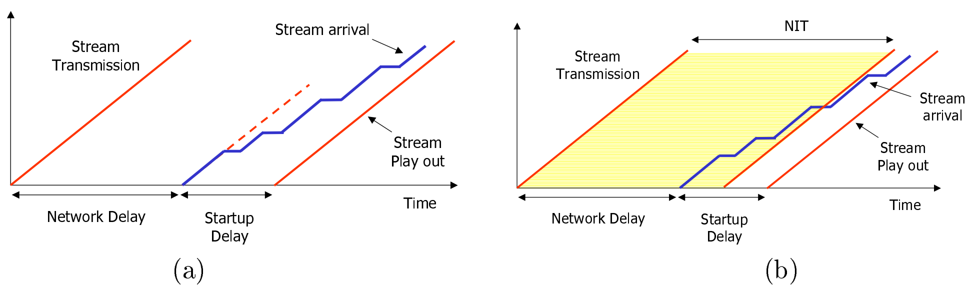


Figure 3: (a) smoothing mechanism. (b) problems in supporting interactive applications.

1.1 Related Work

As we already stated, in video streaming applications the network jitter may cause QoS problems to the receiver, as some video frames may arrive later than expected, causing video play out interruptions.

In literature there are mechanisms, called *buffering* or *smoothing* techniques [9, 14, 15, 16], that can be used to mask the network jitter.

These techniques, as shown in Fig. 3(a), 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.

In essence, 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, using the buffering techniques, it may happen that the overall end-to-end delay goes above the NIT, as shown in Fig. 3(b) where a hypothetical NIT value is depicted. If this happens, the interactive application is not well supported. In fact, NIT represents the limit above which the human perception (and hence interactions) is affected and, to support interactive applications, the end-to-end delay has to stay below the NIT value.

Since for video applications the NIT value is usually less than 500ms, it is not possible to use mechanisms, as buffering techniques, that usually introduce a start-up delay of few seconds in order to ameliorate the network jitter [17].

2 Proposed Mechanism

In this section we present characteristics and properties of the mechanism we propose to support interactive video streaming applications over IP networks.

As we already pointed out, to support interactive operations it is mandatory to keep the end-to-end delay within the Natural Interaction Threshold (NIT). NIT is application dependent and represents the upper bound to the end-to-end delay in order to provide natural interactions between end-users.

Since a start-up delay approach increases the overall end-to-end delay, our mechanism does not use any startup-delay and the receiver begins the video play out upon the reception of the first video frame.

The video play out without any start-up delay is effective in networks that provide some guarantees to the applications, such as sufficient bandwidth, low packet-loss and end-to-end delay, but may pose problems if used in best-effort networks. In fact, in these networks, the end-to-end delay may be very variable, causing frames to have an unpredictable arrival time. Needless to say, as we show in section 2.2, this may compromise both the continuity of the video play out (the receiver may not have received video frames that are requested for the play out) and the natural interactions between end-users. Due to this variability, it is necessary to take into consideration the network behavior while supporting an application. For this reason, our mechanism adapts the video transmission and the video play out to the network conditions.

As we stated, there is an ideal and an actual position to play out a video stream. Our mechanism plays out the video stream in the actual position while attempting to simulate the play out in the ideal position. For this reason, our mechanism measures the time difference between the actual and the ideal play out. If this difference is not noticeable to the users, the interactive applications is well supported. By supposing the clocks at the sender and at the receiver side synchronized, we can, through a timestamp mechanism, measure this time difference. This time-difference is denoted with VTD (Video Time Difference), and it is periodically measured at the receiver side. If the VTD is above the NIT, the application is not well supported: the actual play out is no longer synchronized with the ideal play out. Hence, the video application must be adapted to the new network conditions. This is done by our mechanism that re-synchronizes the actual and the ideal play out by acting on the video QoS. Roughly, this reports the VTD within the NIT. In essence, when the receiver finds out that the VTD is above the acceptable limit, it computes the number of video frames that is necessary to drop in order to report the VTD within the NIT and sends this value to the sender. At the other network side, the sender re-synchronizes the actual and the ideal play out, by dropping the requested number of frames. In the following we show that, by acting on the video QoS (i.e., by dropping some frames of the video), our mechanism is effective in reporting the VTD within the acceptable NIT and hence it is effective in supporting interactive features.

Before explaining the details of our mechanism, we introduce two definitions in order to simplify the description of our mechanism 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 (actual) play out time of the frame i at the receiver side. ◇

2.1 Transmission and Play Out Algorithms

Video streaming applications are usually composed of two main programs: one is located at the sender side and controls the transmission of the video stream into the network; the other, located at the receiver side, retrieves video frames from the network and plays them out. In this section we describe the details of the algorithm that controls the transmission at the sender side and the video play out algorithm at the receiver side.

2.1.1 Transmission Algorithm

A video stream is composed of a sequence of video frames that must be displayed a fixed time within of each other. This technique produces the motion effect. Briefly, the encoding process establishes the number of frame that must be displayed every second and the video play out algorithm has to display these frames according to the number of frames per second established during the encoding process. For instance, the video may be composed of 24 frames per second and, in this case, video frames must be displayed 1/24 of a second within of each other.

In the following we denote the number of video frames that must be displayed in one second with the parameter δ. Since video frames are transmitted over the Internet, it is important to mark these frames, so that the receiver can correctly reproduce the video stream. Our mechanism marks each video frame with a timestamp that represents the ideal play out time of the frame (i.e., the play out time as if the video would be played out at the sender side). The timestamps are given according to the following rules:

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 $T_S(i) = t + (i-1) \cdot \alpha$.

2.1.2 Video Play out algorithm

The goal of the video play out algorithm is to play out the video frames in order to produce the motion effect. In essence, the receiver retrieves video frames from the network, temporarily stores them into its local buffer and then plays out these video frames according to the following rules:

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;

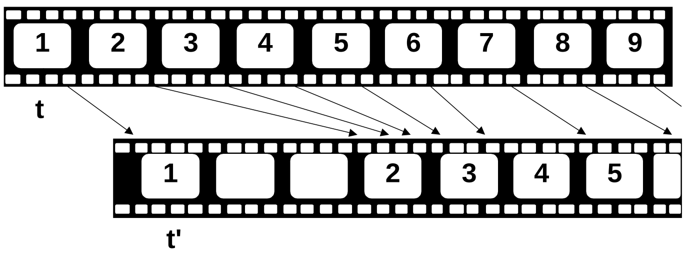


Figure 4: Network scenario while transmitting a video stream between a sender and a receiver.

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 $T_S(i) = \min(T_S(j)$ for each frame j in the buffer);
4. Once selected, a frame i is removed from the buffer and is played out at time $T_R(i) = T_R(i-1) + \alpha$ only if: a) $T_R(i) \geq T_S(i)$ and b) $T_S(i) > T_S(prev(i))$, 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);

In other words, rule 4 says that, if a selected frame i has a timestamp lower than the timestamp of the most recently frame that has been played out (i.e., $T_S(i) < T_S(prev(i))$) then frame i is discarded and a new frame selection (rule 3) must be done. This is done in order to avoid the play out of a frame i that has been transmitted before the transmission of the frame $prev(i)$, but, due to network problems, frame i arrives later that the play out time of the $prev(i)$ frame. Needless to say, it is not possible that $T_S(i) = T_S(j)$, if $i \neq j$.

Based on the previous rules, the algorithm plays out a frame i at time $T_R(i) = T_R(prev(i)) + \alpha$, where $prev(i)$ indicates the frame played out just before frame i and $\alpha = 1/\delta$. Note that, in the following we denote the play out of the first video frame with $T_R(1) = t'$.

2.2 Video play out problems caused by the network jitter

The algorithms described in the previous section are effective and do not pose any problems if the underlying network provides guarantees such as low communication delay and jitter. Conversely, possible problems may arise in the Internet scenario, as we describe in Fig. 4, where sender starts transmitting video frames at fixed rate (i.e., one frame every $\alpha = 1/\delta$ time units) at time t. At time t', the first video frame arrives at the receiver. Since there is no startup delay, the receiver immediately plays out frame 1. Frame 2 is supposed to be played out at time $t' + \alpha$, but due to network problems, frame 2 is delivered

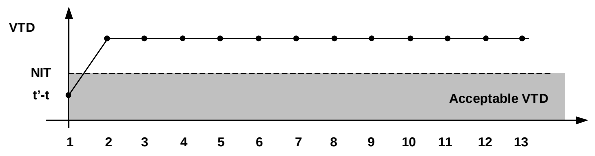


Figure 5: VTD measured while playing out the video stream.

later than expected. For example, let us suppose that frame 2 arrives between $t' + 2\alpha$ and $t' + 3\alpha$. Hence, at time $t' + \alpha$, as well as at time $t' + 2\alpha$, the receiver has no frame to play out. This means that the video play out will be freezed 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 "on time", their play out is delayed by the network problems experienced while transmitting frame 2.

As we describe in the next section, if the supported application is interactive, possible problems may arise.

2.3 Time difference between ideal and actual play out

The situation described in the previous section highlights the critical role played by the network jitter in the video play out. As we already stated, when interactive operations are involved, it is fundamental that the end-to-end delay stays within the NIT value. To investigate whether the application is well supported or not, our mechanism periodically measures the end-to-end delay and checks if it is within the NIT value or not. To measure the end-to-end delay experienced by a video frame, it is to note that only the play out time (and not the arrival time) is important. In fact, a user notices a video frame when it appears on the screen and not when it arrives in its buffer.

To point out the difference between the arrival time of a frame at the end-host and its play-out time, we consider again the example in Fig. 4. For example, frame 3 arrives just after $t' + 3\alpha$, but this frame appears on the user's display only at time $t' + 4\alpha$. Hence, from the user's point of view, it doesn't matter when frame 3 actually arrived ($t' + 3\alpha$), but when it is played out ($t' + 4\alpha$).

The actual play out time is compared to the ideal play out time of the frame (i.e., the associated timestamp) and the difference between these two values represents the end-to-end delay of the considered frame.

To measure this difference, we introduce the following metric, called Video Time Difference (VTD).

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 actual 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 $VTD(i) = T_R(i) - T_S(i)$. $\diamond$

VTD measures the difference between the actual play out time of a frame and its ideal play out time. Note that, if both sender and receiver applications reside on the same computer (i.e., no network is involved), the VTD is equal to zero and this represents the ideal condition for human interactions (i.e., $T_R(i) = T_S(i)$ for each frame).

To better understand the effects of the network jitter on the VTD, in Fig 5 we show the VTD measured for each played frame, with respect to the scenario described in Fig. 4. A hypothetical NIT value is also depicted in order to better highlight frames with a VTD below or above the acceptable limit.

Since we supposed that $T_S(1) = t$ and $T_R(1) = t'$, it follows that $VTD(1) = t' - t$. Frame 2 arrives later than expected (it was supposed to arrive before time $t' + \alpha$, but it arrives between $t' + 2\alpha$ and $t' + 3\alpha$). Hence, the play out of frame 2 happens at time $t' + 3\alpha$ and hence $VTD(2) = t' - t + 2\alpha$. Let us suppose that VTD(2) goes above the NIT value (i.e. $VTD(2) > NIT$). 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. In fact, $VTD(j) \geq NIT$, for each $j \geq 2$.

Needless to say, this situation poses a serious problem if the supported application has interactive features, as all the frames, but the first, have a VTD above the acceptable NIT. For this reason, there is a need to design a mechanism that reports the VTD within the acceptable NIT. We explain how our proposed mechanism deals with these problems in the following section.

2.4 Synchronization between ideal and actual play out

In this section we describe how our mechanism recovers from a situation where natural interactions are compromised (i.e., when the VTD is above the NIT limit). In order to report the VTD within the acceptable limit, we design a mechanism that acts on the video QoS, by discarding video frames. As we show in the following, the discarding mechanism allows to modify the VTD and hence it is possible to maintain the VTD within the acceptable value for human interactions.

First of all, we show that it is possible to reduce the VTD of an arbitrary time quantity. In fact, the following theorem states that the arbitrary time quantity can be translated in video frames and then by discarding these frames, the VTD is reduced.

Theorem 1 *The VTD can be reduced of ρ time units, by dropping a number of frames, say k, that corresponds to ρ time units (i.e. $k \cdot \alpha = \rho$), where $\alpha = 1/\delta$ and δ denotes the number of frames that must be played every second.*

Proof Let us consider the play out of a frame j and suppose that $VTD(j) = T_R(j) - T_S(j) = t' - t + \rho$.

By definition, $T_S(j) = t + (j-1) \cdot \alpha$. It follows that $T_R(j) = VTD(j) + T_S(j) = t' + \rho + (j-1) \cdot \alpha$.

If the network condition will not change while transmitting the successive frames, the VTD will not change, too. Hence, if we consider a frame z ($z > j$, $z = j + k + 1$), $VTD(z) = t' - t + \rho$.

Now, suppose that the sender, after transmitting frame j, avoids transmitting k consecutive frames and sends frame z just after frame j.

To compute the VTD of the frame z ($VTD(z) = T_R(z) - T_S(z)$) we consider that: i) the value $T_S(z)$ is known by definition and is equal to $t + (z-1) \cdot \alpha$; ii) the value $T_R(z)$ is equal to $T_R(z) = T_R(prev(z)) + \alpha$, where $prev(z)$ indicates the frame played out just before frame z.

Since the frame played out just before frame z is the frame j ($prev(z) = j$), it follows that $T_R(z) = T_R(j) + \alpha$.

$T_R(j)$ is known and is equal to $t' + \rho + (j-1) \cdot \alpha$. It follows that $T_R(z) = t' + \rho + j \cdot \alpha$.

Now, considering that, by definition, $\rho = k \cdot \alpha$, it is easy to compute $VTD(z)$ that is equal to: $VTD(z) = t - t'$.

Hence, this theorem proves that the discard of k consecutive frames reduces the VTD of ρ time units. $\diamond$

The previous Theorem allows us to reduce the VTD of a known quantity. Since, through the timestamp mechanism, we exactly know the amount of time that exceeds the acceptable limit NIT (i.e., $VTD - NIT$), it is easy to report the VTD within the acceptable limit.

In fact, the value $VTD - NIT$ (ρ in Theorem 1) is used to compute the number of frames that has to be dropped (k in Theorem 1) in order to report the VTD within the NIT limit. Our mechanism works as follows. The receiver can easily compute the value $VTD - NIT$. If this value is greater than zero, it is sent to the sender. After receiving this message, the sender can discard a number of frames that corresponds to the time quantity $VTD - NIT$. In this way, as Theorem 1 states, the VTD is reduced and hence is reported within the acceptable NIT limit.

To show the effects of our mechanism, in Fig. 6, we consider again the example depicted in Fig. 4, but now when the receiver finds out that the VTD goes above the acceptable value (for instance when playing out frame 2, it knows that VTD(2) is greater that NIT) it sends to the sender the value $VTD(2) - NIT$. When this message arrives at the sender, the synchronization mechanism is activated. The sender uses this value ($VTD(2) - NIT \equiv \rho$) to compute the number of frames (k) that has to be discarded in order to report the VTD within the NIT. Let us suppose that it is necessary to drop 2 frames, the sender discards (i.e., it does not transmit), frame 7 and frame 8. This means that, just after frame 6, the sender transmits frame 9, frame 10 and so on.

In Fig. 7 we show the effects of our mechanism on the VTD. The benefits introduced by our mechanism starts when playing out frame 9. In fact, if our mechanism is not used (Fig. 5), frame 9 is played out with VTD(9) greater than NIT, but if our mechanism is used (Fig. 7), VTD(9) is lower than NIT.

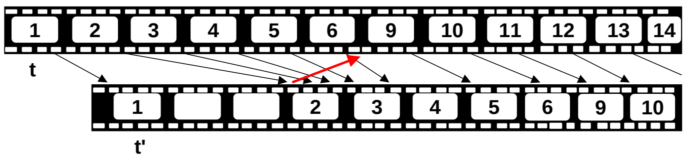


Figure 6: Network scenario while transmitting a video stream between a sender and a receiver when using our mechanism.

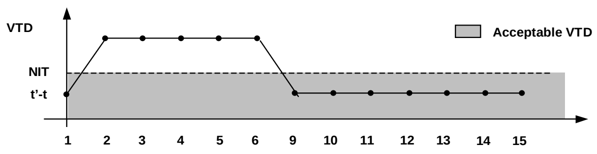


Figure 7: VTD measured while playing out the video stream when using our mechanism.

Moreover, using our mechanism, all the frames transmitted after frame 9 are within the NIT value.

This means that our mechanism may introduce considerable benefits in supporting interactive video streaming applications.

The drawback of our mechanism is that it affects the video QoS. However, the discard of a good selection of frames does not greatly affect the video QoS, as showed in [18, 19], where dropping frames techniques are used to reduce bandwidth allocation requirements.

3 Experimental Scenario

In this section we present characteristics and properties of the scenario we use to evaluate our mechanism. Results of this evaluation are presented in the next section.

3.1 Experimental Architecture

As already mentioned, the most important metric that influences the interactive features of an application is represented by the NIT. If the VTD is kept within the NIT, the application is well supported, otherwise it is not. The goal of our mechanism is to keep the VTD within the NIT.

To investigate the benefits introduced by our mechanism we have to compare situations where our mechanism is used and when it is not and the comparison has to be done under the same network conditions. To this end, we first collected video delay traces using an RTP-based video streaming application implemented at the University of Bologna [31] and then we developed a *trace-driven* simulator that reads in the transmission delay of each video frame and detects whether the VTD of the video frame is within the NIT or not. For each video trace, we computed the percentage of video frames with VTD above the NIT in both situations (with and without using our mechanism). This allows us to measure the benefits introduced by our mechanism.

Further, since the NIT is not fixed, but it is application dependent, we investigated the behavior of our mechanism with several different values of the NIT for each transmitted video, in order to cover different application requirements. To achieve this, we run the simulator several times with different NIT values and with the same network traffic conditions.

Another evaluation has been done in order to investigate the drawbacks of our mechanism. In fact, to keep the VTD within the NIT, our mechanism drops video frames. While the number of dropped video frames is easy to compute, it is not much related with the perceived QoS at the client side. For this reason, we use another approach to accounting for the perceptual playout quality. In particular, as suggested in [18, 19], we use a cost function to measure the perceived video quality. There are many ways to define a cost function, but its definition goes beyond the scope of this paper. Hence, we focus on the same cost function used in [18, 19], which penalizes frame dropping mechanisms that drop neighboring frames.

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. More details about this cost function can be found in [18, 19].

To have a complete evaluation of our mechanism, we propose several frame dropping algorithms and, through our developed trace driven simulator, we evaluate each of them using the same network traffic condition.

Before presenting the experimental results, it is worth analyzing the network scenario and the characteristics of the video streams we used to collect the network delay traces.

3.2 Video Delay Traces

The effects of our mechanism can be showed only if compared with situations where our mechanism is not used. To be comparable, the same network traffic conditions have to be used both when our mechanism is used and it is not. Since identical network traffic conditions are impossible to obtain in the Internet, we first collected network delay traces by transmitting a set of video streams. For this reason, we developed an RTP-based application, that, while transmitting a video stream towards a destination, measures the end-to-end delay experienced by each video frame.

To cover different situations, we collected network delay traces in both a 100Mbps department LAN and the Internet. We used two sets of video traces encoded with Motion JPEG and MPEG and the network delay traces have been collected during different time (peak hours, office hours, evening).

A trace-driven simulator has been developed to use the collected delay traces and to test the behavior of our mechanism. Results are presented in the following section.

3.3 Video streams

Two sets of video traces have been considered: one is encoded with Motion JPEG and the other with MPEG. We selected these two techniques to evaluate our mechanism with both intra-frame technique (like Motion JPEG) and inter-frame technique (like MPEG).

With video streams encoded with inter-frame mechanisms (like MPEG) a *domino* effect may happen (i.e., a discard of a frame may lead to the impossibility of decoding other video frames) and hence the discarding mechanism has to take into account all the dependency information among the video frames. Conversely, if the video streams are encoded with intra-frame technique (like Motion JPEG), it is possible to discard frames without causing the domino effect.

We used video traces with different characteristics (cartoons, music videos, news, movies) in order to have video streams with different bandwidth requirements and, hence, to have a complete testbed for our mechanism.

3.3.1 Motion JPEG

Motion JPEG in an intra-frame encoding mechanism that produces a sequence of video frames, where each frame is independently compressed using the JPEG technique. There is no correlation between successive frames and hence the server can immediately discard video frames when it receives the request from the receiver. The used video traces are *The Beauty and The Beast*, *Jurassic Park* and *Total Recall*, encoded with 24 fps.

3.3.2 MPEG

The used MPEG video traces are: *The Simpsons*, *MTV* and *News*, encoded with 24 fps. MPEG is an inter-frame encoding mechanism that yields a smaller average frame size than the Motion JPEG encoding and the resulting video stream is composed of frames that don't have the same importance, as some frames depend on other frames. For this reason it is important to know what type of frame the server can discard. Before introducing the proposed discarding algorithms, we briefly present the characteristics of an MPEG video stream.

MPEG videos are organized in Group of Picture (GOP) (in our experiments we use GOP composed of 12 frames). The frames may have different importance and are represented by three type of frames: I, P, and B. Each GOP is composed as: $IB_1B_2P_1B_3B_4P_2B_5B_6P_3B_7B_8$. To decode a B frame, both the previous and future I or P frames are needed. To decode a P frame, the previous P or I frame is needed. Only I frames can be decoded without using other frames.

This means that if a I frame is not present, the entire GOP (plus the two B frames of the previous GOP that depend on the I frame) cannot be decoded and are discarded. Hence, 14 frames are impossible to decode in an I frame is missing. If a P_1 frame is missing, then 11 frames are impossible to be decoded; if a P_2 frame is missing, then it is not possible to decode 8 frames; if a P_3 frame is missing, then 5 frames cannot be decoded. Only a missing B frame does not result in additional frame discard.

These dependency rules have been considered while testing our mechanism and, based on them we propose the following algorithms in order to discard frames at the server side when the client asks to drop frames. **Drop any frame (DAF)**: Frames are dropped without any consideration about the frame type; **Drop I frame (DIF)**: Only I frames are considered (14 frames cannot be played out for each discarded I frame); **Drop P_1 frame (DP1F)**: Only P_1 frames are considered (11 frames cannot be played out for each discarded P_1 frame); **Drop P_2 frame (DP2F)**: Only P_2 frames are considered (8 frames cannot be played out for each discarded P_2 frame); **Drop P_3 frame (DP3F)**: Only P_3 frames are discarded (5 frames cannot be played out for each discarded P_3 frame); **Drop B frame (DBF)**: Only B frames are considered.

4 Experimental Results

In the following, we present results obtained from analyzing our mechanism over a LAN and over the Internet. Since the goal of our mechanism is to keep the VTD within the NIT, we measure VTD for each delivered frame. VTD is affected by the network jitter and, in fact, it is very variable if the video is transmitted over best-effort networks. Fig. 8 highlights this variability and presents the VTD measured while transmitting a video trace of the cartoon *The Simpsons*. As already pointed out, this variability compromises the QoS of the supported interactive application. In fact, depending on the NIT value, some video frames may have the VTD above the NIT value. If this happens, the application is not well supported. The number of frames with a VTD above the NIT value depends on the NIT value. Since it is not fixed, but it is application dependent, we use several different values of the NIT, though using the same network traffic condition (i.e., we use the same network delay trace), for each transmitted video in order to investigate the behavior of our mechanism.

For each NIT value, we count the number of frames with the VTD above the NIT, and then we compute its percentage with respect to the entire video stream. This percentage is computed under the same traffic conditions for situations where the video streams are delivered with and without using our mechanism. A comparison analysis is given for each delivered video streams.

Further, we evaluate the QoS of each delivered video streams using the cost

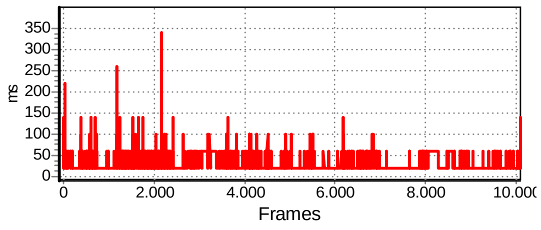


Figure 8: Video Time Difference obtained transmitting a clip of *The Simpsons*.

function presented in the previous section. For each delivered video streams a QoS evaluation is done both when our mechanism is used and when not. In fact, the video QoS is affected by both our mechanism and by the network. To understand how our mechanism affects the video QoS, it is fundamental to know how the network affects the video QoS. To better understand why the network affects the video QoS, it is worth analyzing Fig. 8. In particular, it is interesting to observe the behavior of the VTD curve (our mechanism is not used). The VTD curve goes up and down. While the increasing part is easy to understand (the network delivers packets later than expected and hence the play out is delayed), it is interesting to observe the decreasing part. In fact, the receiver always plays out frames at fixed and pre-defined number of fps, and hence the VTD should never decrease. However, the decreasing part is caused by the network packet loss that decreases the VTD. As Theorem 1 states, VTD can be reduced by discarding video frames. In this case, video frames are not intentionally dropped by our mechanism, but it is the network that drops these frames. Unfortunately, it is not possible to control the network behavior, as the network discards frames when it is congestioned. This network behavior causes the VTD to be variable and affects the video QoS. For this reason we evaluate the video QoS both when our mechanism is used and when it is not, using the same network traffic conditions. A comparison analysis is presented.

In conclusion, through a trace-driven simulation, we investigate the following measures:

- The percentage of video frames with VTD above the NIT with and without using our mechanism;
- The QoS cost introduced by our mechanism. In particular, with MPEG video streams, we investigate the QoS cost introduced by several dropping frames algorithms that we propose in order to discard video frames.

4.1 LAN Environment and Motion JPEG video streams

In Fig. 9(a) we present results obtained from transmitting the *The Beauty and the Beast*. With a 80 ms NIT, 5% of the frames has a VTD above the acceptable limit. This percentage decreases while increasing the NIT value and it drops near zero percent for NIT values greater than 180 ms. Conversely, if

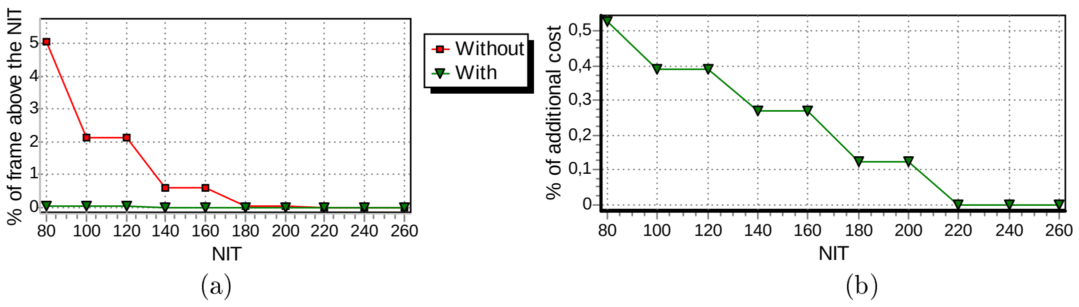


Figure 9: Transmission of *The Beauty and the Beast* over the LAN: (a) percentage of frames with VTD above the NIT. (b) cost of the dropping frames algorithm.

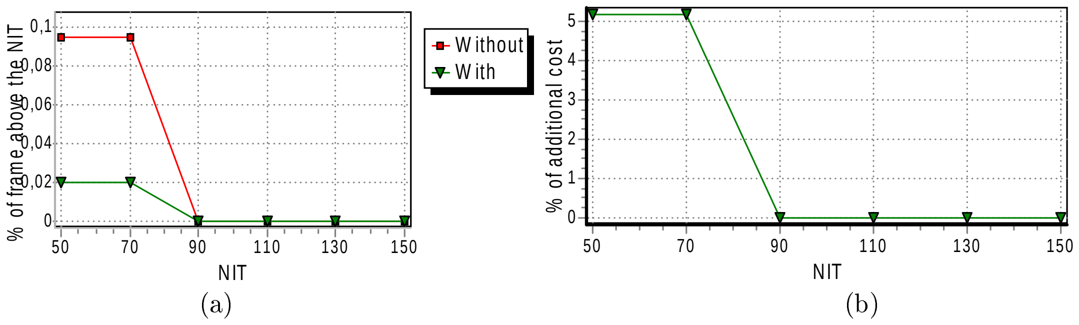


Figure 10: Transmission of *Total Recall* over the LAN: (a) percentage of frames with VTD above the NIT. (b) cost of the dropping frames algorithm.

our mechanism is used, the percentage is kept very close to zero, for all the NIT values.

In order to keep this percentage close to zero, our mechanism discards video frames and hence it affects the video QoS. In Fig. 9(b) we present the percentage of additional QoS cost introduced by our mechanism, with respect to the QoS cost computed when our mechanism is not used. It is possible to note that our mechanism slightly affects the perceived video QoS: the maximum additional cost (0.5%) is obtained with a NIT value of 80 ms and the benefits are considerable as it avoids 5% of the frame to have a VTD above the NIT value.

The reason of this great benefits, obtained by slightly affecting the video QoS, is highlighted by the example described in sections 2.3 and 2.4. In fact, in Fig. 5 the delay experienced by frame 2 causes the VTD to go above the NIT limit for all the successive frames. In Fig. 6 and 7 we showed that by discarding only 2 frames, only few frames have a VTD above the NIT limit, while all of the other frames are within the NIT limit.

In Fig. 10(a) we present results obtained from transmitting *Total Recall*. In this case, the video stream is transmitted quite well over the LAN. In fact, the

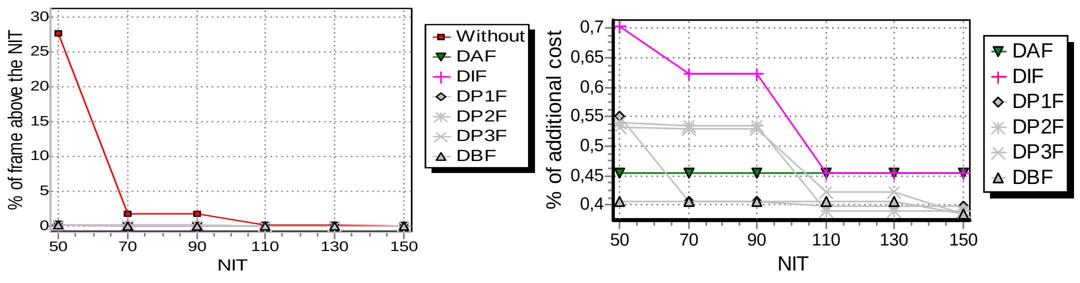


Figure 11: Transmission of *The Simpsons* over a LAN: (a) percentage of frames with VTD above the NIT. (b) cost of the dropping frames algorithm.

percentage of frames with VTD above the NIT value is kept within 0.1% for all the considered NIT values.

The Qos evaluation presented in Fig. 10(b) shows that for NIT values greater than 90 ms, there is no need of dropping frames as the video stream is transmitted without any problems.

The benefits introduced showed that our mechanism is well suited for supporting interactive video applications over a local network.

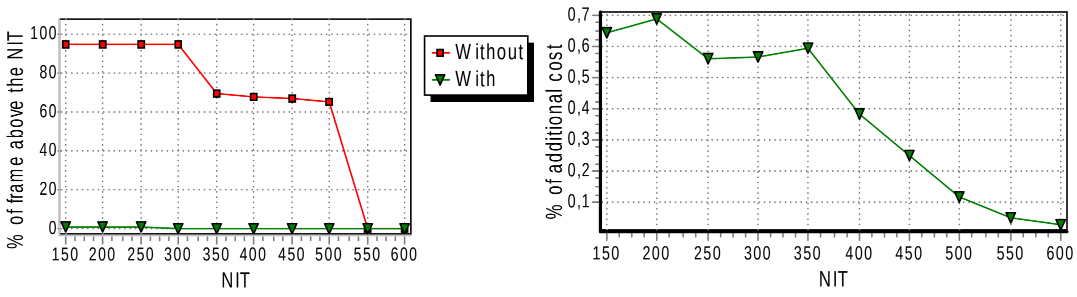
4.2 LAN Environment and MPEG video streams

In Fig.11 we present results obtained from transmitting the cartoon *The Simpsons*. Fig.11(a) shows that with a NIT of 50 ms, the transmission of the video stream without using the mechanism causes more than 25% of the frames to go above the acceptable limit. This percentage decreases while increasing the NIT value. For instance, with NIT values equal to 70-90 ms, the percentage drops to 2% and reaches almost zero percent with a NIT value of 150 ms.

Conversely, if we use the mechanism and the discarding algorithms presented in section 3, the percentage is kept very close to zero. In this case, all the proposed algorithms produce almost the same benefits (Fig.11(a)).

The QoS evaluation is presented in Fig. 11(b). It is possible to note that there are small differences among the proposed dropping frame algorithms, and their costs are similar to the original one. This is due to the small number of discarder frames. However, it is possible to note that DIF performs worse than the other algorithms, while DBF is the algorithm that produces the lowest additional cost (around 0.4% more than the original cost).

Similar results have been obtained with *MTV* and a *NEWS* clip. Readers can refer to [21, 22] for further details.



(a) (b)

Figure 12: Transmission of *Sleepless in Seattle* over the Internet (8 hops): (a) percentage of frames with VTD above the NIT. (b) cost of the dropping frames algorithm.

4.3 The Internet Environment and Motion JPEG video streams

In this section we evaluated our mechanism in the Internet scenario: one path is from Bologna to Trieste (9 hops) and the other from Bologna to Alessandria (8 hops). It is worth noticing that in the two previously mentioned Internet connections (Bologna-Trieste and Bologna-Alessandria) an average value of the Round Trip Time equal to 80-90 ms and 300-400ms was measured, respectively.

In Fig. 12(a) we present results obtained from transmitting *Sleepless in Seattle*. For NIT values up to 300 ms, more than 90% of the frames has a VTD above the acceptable limit. This percentage decreases while increasing the NIT value and it drops near zero percent for NIT values greater than 550 ms. Conversely, if our mechanism is used, the percentage is kept very close to zero, for all the NIT values.

In Fig. 12(b) we present the QoS evaluation. In this case the additional cost is kept within 0.7%. Hence, our mechanism is able to keep the video stream transmission within the acceptable NIT by slightly affecting the video QoS.

In Fig. 13(a) we present results obtained from transmitting *Total Recall*. For 150 ms NIT value, almost 80% of the frames has a VTD above the acceptable limit. This percentage decreases while increasing the NIT value and it drops near zero percent for NIT values greater than 600 ms. Conversely, if our mechanism is used, the percentage is kept very close to zero, for all the NIT values.

In Fig. 13(b) we present the QoS evaluation. In this case the additional cost is kept within 0.07%. Once again, our mechanism is able to keep the video stream transmission within the acceptable NIT by slightly affecting the video QoS.

In Fig. 14(a) we present results obtained from transmitting *Jurassic Park*. For 150 ms NIT value, more than 50% of the frames has a VTD above the acceptable limit. This percentage decreases while increasing the NIT value and it drops near zero percent for NIT values greater than 550 ms. Conversely, if

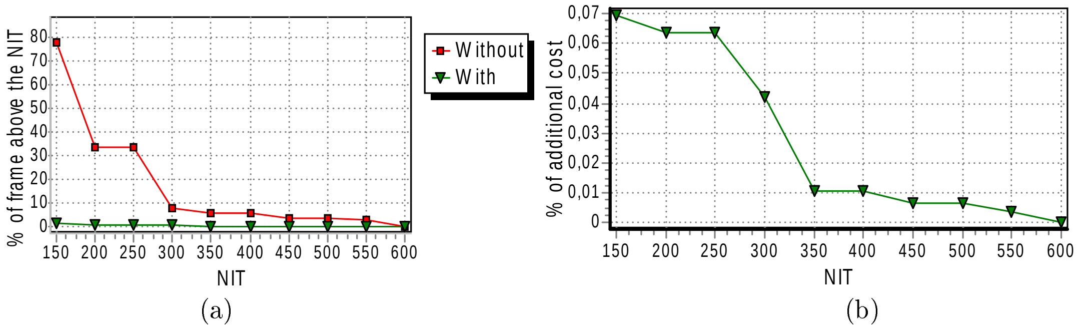


Figure 13: Transmission of *Total Recall* over the Internet (8 hops): (a) percentage of frames with VTD above the NIT. (b) cost of the dropping frames algorithm.

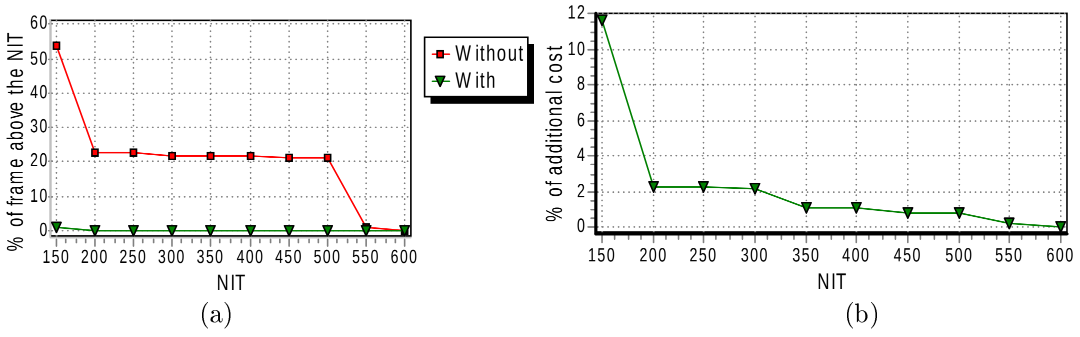


Figure 14: Transmission of *Jurassic Park* over the Internet (8 hops): (a) percentage of frames with VTD above the NIT. (b) cost of the dropping frames algorithm.

our mechanism is used, the percentage is kept very close to zero, for all the NIT values.

In Fig. 14(b) we present the QoS evaluation. In this case the additional cost is 12% for 150 ms NIT value and for NIT values greater than 200 ms the percentage drops to 2% and for NIT values greater than 300 ms the percentage is less than 1%.

In Fig. 15(a) we present results obtained from transmitting *Sleepless in Seattle*. For 100 ms NIT value, almost the entire video stream is above the acceptable limit. This percentage decreases while increasing the NIT value and it drops near zero percent for NIT values greater than 120 ms. Conversely, if our mechanism is used, the percentage is kept very close to zero, for all the NIT values.

In Fig. 15(b) we present the QoS evaluation. In this case the additional cost is kept within 1.8%.

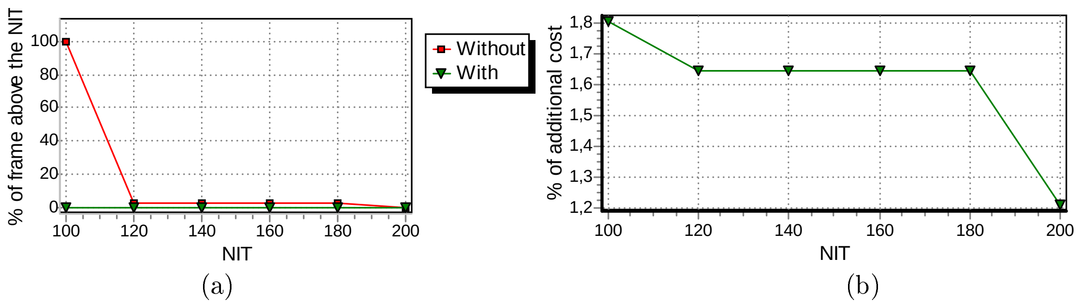


Figure 15: Transmission of *Sleepless in Seattle* over the Internet (9 hops): (a) percentage of frames with VTD above the NIT. (b) cost of the dropping frames algorithm.

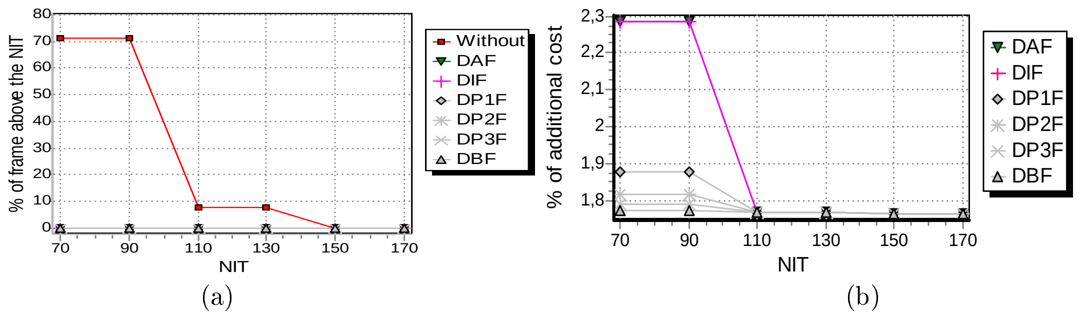


Figure 16: Transmission of *News* over the Internet (9 hops): (a) percentage of frames with VTD above the NIT. (b) cost of the dropping frames algorithm.

4.4 The Internet environment and MPEG video streams

In this section we present results obtained from transmitting MPEG video traces over the Internet.

Fig. 16(a) shows results obtained from transmitting a *NEWS* clip from Bologna to Trieste (9 hops). With a NIT of 70 ms, more than 70% of the frames have a VTD above the NIT. For a NIT value of 110 ms, the percentage drops to less than 10%, and reaches almost zero for a NIT value around 150 ms. Conversely, if the mechanism is used, the percentage is kept very close to zero and the additional cost ranges between 1.7% and 2.3%. Also in this case, DIF performs worse than the other algorithms, and DBF has the lowest costs with all the tested NIT (Fig. 16(b)).

In Fig. 17(a) we present results obtained from transmitting a video trace of *MTV* from Bologna to Alessandria (8 hops). In this case, the network is much more slower than in the previous experiments. With a NIT value of 100 ms, more than 90% of the frames have a VTD above NIT. Conversely, the dropping algorithms allow the mechanism to keep the percentage close to zero. In this case the dropping algorithms introduce a cost that ranges between 2.8% and

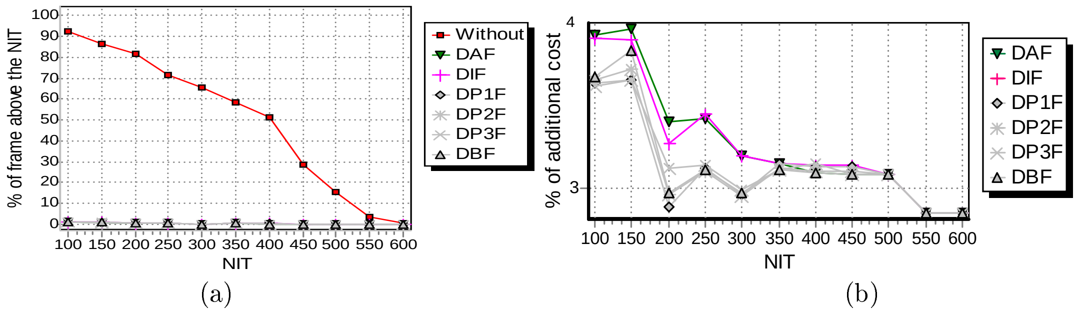


Figure 17: Transmission of *MTV* over the Internet (8 hops): (a) percentage of frames with VTD above the NIT. (b) cost of the dropping frames algorithm.

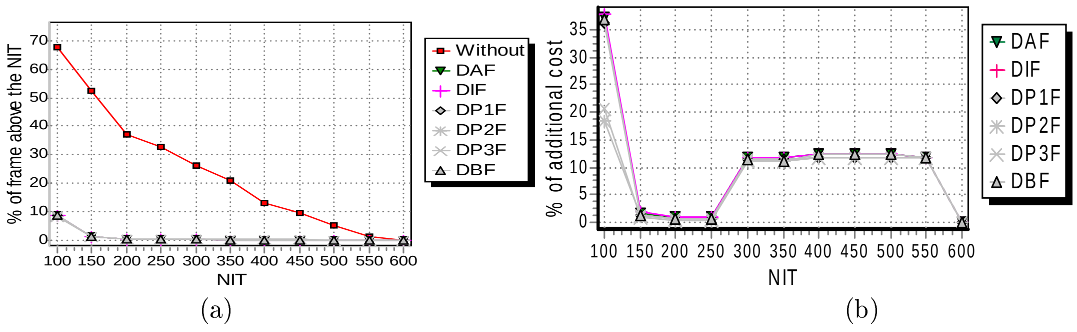


Figure 18: Transmission of *NEWS* over the Internet (8 hops): (a) percentage of frames with VTD above the NIT. (b) cost of the dropping frames algorithm.

4%. For NIT values greater than 250 ms, DIF performs worse that the other and DBF is the one with the lowest cost. For NIT values lower than 250, DAF is the one that produces the highest cost, while it is difficult to point out an algorithm that performs better than the others (Fig.17(b)).

In Fig. 18(a) we present results obtained from transmitting a video trace of a *NEWS* clip, from Bologna to Alessandria (8 hops). With a NIT value of 100 ms, 70% of the frames have a VTD above the NIT. Here, it is interesting to note that, due to the network problems, the mechanism is not able to maintain the percentage close to zero when the NIT is 100 ms, but 9% of the frames goes above the NIT limit. However, the benefits are still considerable with respect to the 70% of frames that goes above the NIT if the mechanism is not used.

In Fig.18(b) is possible to note that for a NIT of 100 ms, the cost increases up to 37% with respect to the original cost. This highlights the great network problems experienced while transmitting the video stream. The cost decreases to 2-3% for NIT values between 150 and 250 ms, while, for greater values, the additional cost is maintained around 12%. For a NIT of 600 ms, the mechanism discards a very few number of frames, and the cost is very close to the one produced by the original stream.

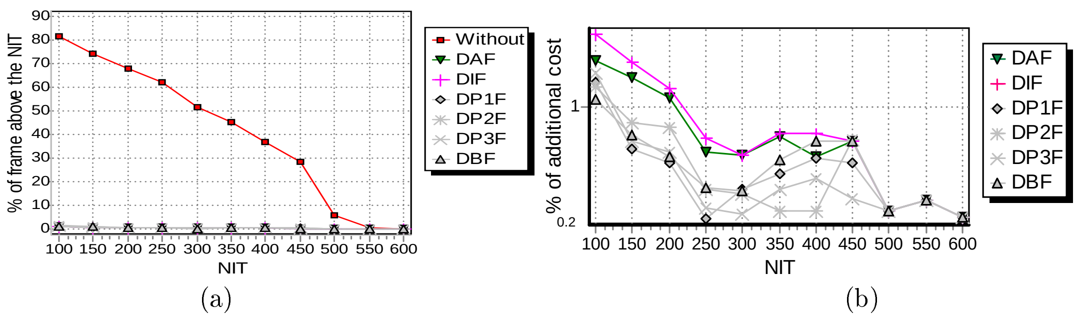


Figure 19: Transmission of *The Simpsons* over the Internet (8 hops): (a) percentage of frames with VTD above the NIT. (b) cost of the dropping frames algorithm.

Results obtained from transmitting a video trace of *The Simpsons*, from Bologna to Alessandria (8 hops) are reported in Fig.19. With a NIT value of 100 ms, more than 80% of the frames go above the acceptable limit and for NIT values greater than 500 ms the percentage drops to less than 10%. The benefits of the mechanism are remarkable, as it allows the percentage to stay near the zero percent and the additional cost is kept within 1.3% (Fig.19(b)).

4.5 Summary of results

All the performed experiments showed that the network jitter greatly affects the supported application: the percentage of video frames with VTD above the NIT is considerable. This means that, without a control mechanism, interactive applications are not well supported over IP networks. Conversely, using our mechanism the percentage of video frames with VTD above the NIT is highly reduced, proving that our mechanism is effective in keeping the VTD within the NIT along the application lifetime. Further, the QoS investigation shows that our mechanism only slightly affects the perceived QoS of the delivered video stream. Results obtained along with the QoS evaluation show that our mechanism is effective in supporting interactive video streaming applications over IP networks.

5 Conclusions and Future Work

In this paper we proposed a new approach for supporting interactive video streaming applications over IP networks. Along the application lifetime, our mechanism controls the network conditions and adapts both the video transmission and the video play out to the new network conditions when it is necessary. The adaptation to the network conditions is necessary to support interactive applications, as these applications have a constraint on the end-to-end delay: it must be lower than a pre-defined threshold, denoted with NIT, along the application lifetime. For this reason, interactive applications are critical to support in best-effort networks, as both the network delay and the network jitter affect the overall end-to-end delay and hence users perceive a QoS that is far from what desired.

We highlighted that the client can interact without any problems if it is directly connected to the server (ideal position), but it can have QoS problems if it is connected to the server through a best-effort network (actual position).

To provide natural interactions between end-users, the time difference (VTD) between the ideal and the actual play out must have a value lower than NIT. The time difference is periodically measured through a timestamp mechanism. If the VTD is above the NIT, it means that the actual play out is late with respect to the ideal play out. In this case, our mechanism, using the new network conditions, acts on the video QoS in order to re-synchronize the actual and the ideal play out.

Our mechanism has been evaluated through several simulations, performed by using real network delay traces obtained transmitting Motion JPEG and MPEG video streams over both LAN and the Internet. Results obtained showed the high variability of the VTD and the effectiveness of our mechanism in keeping the VTD within the NIT value. On the application side, this means that our mechanism ameliorates the system reaction to the end-user command. For instance, if we consider a video-on-demand applications, our mechanism allows the user to better interact (pause, fast forward, rewind) with the video server.

A QoS evaluation has been done in order to investigate the effects of our mechanism on the perceived QoS at the user side. The analysis showed that our mechanism affects the video QoS in a slightly way.

These evaluations showed that our mechanism is well suited for supporting interactive video applications (e.g., distance learning, pay per view, video on demand, etc.), over IP networks.

We are currently investigate the behavior of our mechanism in two more scenarios, UMTS and Internet2 (video streaming applications are likely to be the most used applications into these future environments and interactivity should heavily contribute to the success of these applications), and we are studying the relationship between the NIT and the video contents, in order to dynamically set the NIT along the application lifetime (for instance a distance learning application should be more interactive than a football match) and to drop frames in video situations less important than other (title list, scene cut off).

Acknowledgements
We are indebted to Italian M.I.U.R and C.N.R. for the partial financial support to our work under a grant.

References

[1] ISO/IEC, Information Technology-Coding of Moving Pictures and Associated Audio for Digital Storage Media up to about 1.5 Mbit/s, International Organization for Standardization, 1993.

[2] B. Furht: *A survey of multimedia compression techniques and standards. Part I. JPEG standards*, Real Time Imaging, vol. 1, pp. 49-67, 1995.

[3] ITU-T, Recommendation H.261. September 1994.

[4] M. Garret, W. Willinger: *Analysis, Modeling and Generation of Self-Similar VBR Video Traffic*, Proc. ACM SIGCOM, pp. 269-280, August 1994.

[5] S. Kadur, F. Golshani, G. Millard: *Delay-jitter control in multimedia application*, Multimedia Systems, No. 4, pp. 30-39, 1996.

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

[7] G. Karlsson: *Quality Requirements for Multimedia Network Services*, Proceedings of Radiovetenskap och kommunikation -96, pp.96-100, Lulea, Sweden, June 3-6, 1996.

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

[9] R. Ramjee, J. Kurose, D. Towsley, H. Schulzrinne: *Adaptive Playout Mechanisms for packetized audio applications in wide-area networks*, IEEE Computer and Communications Societies on Networking for Global Communications, pp. 680-688, Loas Alamitos, CA, June 1994.

[10] M. Roccetti, V. Ghini, G. Pau, P. Salomoni, M.E. Bonfigli: *Design and Experimental Evaluation of an Adaptive Playout Delay Control Mechanism for Packetized Audio for Use over the Internet*, Multimedia Tools and Applications, Kluwer Academic Publishers, Vol. 14, N. 1, pp. 23-53, May 2001.

[11] M. Furini, M. Roccetti: *Adaptive Video Transmission over the Internet: an Experimental Study*, Proc. of 2000 European Simulation Symposium (ESS'2000) , (D.P.F. Moeller Ed.), The Society for Computer Simulation International, Hamburg (Germany), pp. 159-163, September 2000.

[12] M. Claypool, J. Riedl: *End-to end quality in multimedia applications*, Chapter 40 in Handbook on Multimedia Computing, 1999.

[13] M. Baldi, Y. Ofek: *End-to-End Delay Analysis of Videoconferencing over packet-Switched Networks*, IEEE/ACM Transaction on Networking, vol. 8, No. 4, pp 479-492, August 2000.

[14] W-C. Feng, F. Jahanian, S. Sechrest: *Optimal buffering for the delivery of compressed prerecorded video*, Multimedia Systems, Vol. 5, No. 5, pp. 297-309, 1997.

[15] W-C. Feng, S. Sechrest: *Smoothing and buffering for the delivery of compressed prerecorde video*, Proceedings of the the IS&T/SPIE Symposium on Multimedia Computer and Networking, pp. 234-242, February 1995.

[16] J.D. Salehi, Z.L. Zhang, J. Kurose, D. Towsley: *Supporting Stored Video: Reducing Rate Variability and End-to-End Resources Requirements through Optimal Smoothing*, IEEE/ACM Transactions on Networking, Vol. 6, No.4, pp. 397-410, 1998.

[17] WEBTV NETWORKS INC., *Web TV technical specifications*, www.webtv.net/HTML/home.specs.hmtl

[18] Z.L. Zhang, S. Nelakuditi, R. Aggarwal, R.P. Tsang: *Efficient Selective Frame Discard Algorithms for Stored Video Delivery across Resource Constrained Networks*, Proc. IEEE INFOCOM'99, NYC, pp. 472-479, March 1999.

[19] M. Furini, D. Towsley: *Real-Time traffic transmission over the Internet*, IEEE Transaction on Multimedia, vol. 3, No. 1, pp. 33-40, March 2001.

[20] D. Wijesekera, J. Srivastava: *Quality of Service (QoS) Metrics for Continuos Media*, Multimedia Tools and Applications, Vol. 2, No.3, pp. 127-166, Sept 1996.

[21] M. Furini, M. Roccetti, *Interactive MPEG Video Streaming over IP networks: A Performance Report*, *Proc. of the Communications and Computer Networks (CCN2002) Conference*, MIT, Cambridge, USA, pp. 414-419, November 2002.

[22] M. Furini, M. Roccetti, *Interactive MPEG Video Streaming over IP networks: A Performance Report*, Tech-Rep, Computer Science Department, University of Piemonte Orientale, www.di.unipmn.it, 2002.

[23] S. Cen, C. Pu, J. Walpole: *Flow and congestion control for Internet streaming applications*, Proc. SPIE/ACM Multimedia Computing and Networking '98, San Jose, CA, pp. 250-264, January 1998.

[24] M. Claypool, J. Tanner, *The Effects on Jitter on the Perceptual Quality of Video*, ACM Multimedia, pp. 115-118, November 1999.

[25] D. Ferrari: *Delay Jitter Control Scheme for Packet-Switching internetworks*, Computer Communications, pp. 367-372, July 1992.

[26] M. Furini, M. Roccetti: *Design and Analysis of a Mechanism for supporting Interactive Video Streaming Applications over the Internet*, *Proc. of the 6th IASTED/IMSA Conference*, Kauai, Hawaii, pp. 324-329, August 2002.

[27] M. Furini, M. Roccetti: *Design and Analysis of a Mechanism for supporting Interactive Video Streaming Applications over the Internet*, Technical Report, DISTA Department, University of Piemonte Orientale (www.mfn.unipmn.it), January 2002.

[28] M. Podolsky, M. Vetterli, S. McCanne: *Limited retransmission of real-time layered multimedia*, IEEE Workshop on multimedia Signal Processing, Los Angeles, CA, December 1998.

[29] D. Stone, K. Jeffay: *An Empirical study of delay jitter management policies*, ACM Multimedia Systems, Vol. 2, n.6, pp. 267-279, January 1995.

[30] Y. Wang, Q. Zhu: *Error Control and Concealment for Video Communications: A Review*, Proc. of the IEEE, Vol. 86, pp. 974-997, May 1998.

[31] M. Bracuto: *An RTP-based Video Streaming Application for Use over the Internet*, Master Thesis, Computer Science Department, University of Bologna, 2002.