Design and Analysis of a Mechanism for supporting Interactive Video Streaming Applications over the Internet

Marco Furini[1]*and Marco Roccetti[2]

[1]DISTA Department
University of Piemonte Orientale
Corso Borsalino 54
15100 Alessandria Italy

[2]Department of Computer Science
University of Bologna
Mura Anteo Zamboni 7
40127 Bologna Italy

TR-INF-2002-01-01-UNIPMN

Abstract

Video streaming applications that provide interactive features to the end-user are becoming more popular also in the Internet environment. Needless to say, interactive operations play a fundamental role in these applications and studies showed that these operations are well supported if the end-to-end delay, experienced by the application traffic, is kept lower than a pre-defined, and application dependent, threshold. Unfortunately, the best effort nature of the Internet may compromise the QoS achieved by these applications, as the presence of the network jitter may cause the overall end-to-end delay to go above the threshold. Several mechanisms have been proposed in literature to ameliorate the network jitter and they usually have the drawback of increasing the overall end-to-end delay. In this paper we suggest another approach to support interactive video streaming applications: our mechanism acts on the video QoS in order to keep the end-to-end delay within the acceptable threshold. Our mechanism has been evaluated through several simulations and results obtained show that it is well suited for supporting interactive video streaming applications over the Internet, as it ameliorates the jitter without increasing the end-to-end delay, but only slightly affecting the video QoS.

1 Introduction

Quality of Service (QoS) applications over the Internet are becoming more and more popular, but, despite their popularity, they achieve a QoS that is far from what desired. Videoconferencing, distance learning, 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 be very bandwidth consuming. The great bandwidth requirements are highlighted by the video streaming applications; in fact, even using one of the several compression algorithms proposed in literature (for example, MPEG [12], Motion JPEG [8], H.261 [13]) the compressed video stream can still exhibit significant bit variability [11] and can require high network

*Corresponding author: marco.furini@mfn.unipmn.it

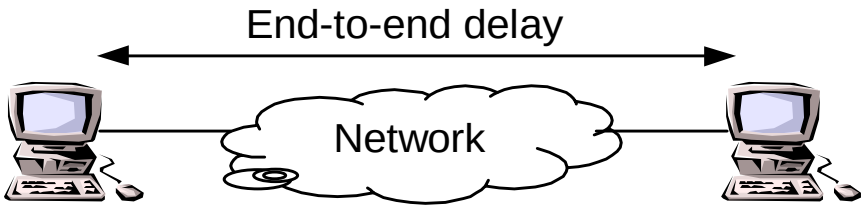

Figure 1: Model of a networked application.

capacity compared to the (usually) available in the Internet. For example, a video stream may require bandwidth that can span from several Kbps to several Mpbs. In addition to the great bandwidth requirement, the correlation of data with the time factor is another characteristic of the stream generated by the applications. In fact, to provide the applications with the QoS they need, the transmission of this traffic has, at least, two time-constraints: minimal communication delay and unnoticeable network jitter to the user [14] [23]. Although these time-constraints can be easily provided in networks that provide some guarantees to the applications, such as sufficient bandwidth and low packet loss, they are very critical to provide in best-effort networks, like the Internet. In fact, best-effort networks cannot guarantee low communication delays and/or low jitter.

A sub-set of these QoS applications enables natural interactions (i.e., more life-like as possible) among end-users. These particular applications are called *interactive* QoS applications. An example of interactive QoS application is the voice over IP, where natural conversation between two end-users is supported. In Fig. 1 we show a very simple network scenario for this application: two end-hosts are connected each other through the Internet. Roughly, this application works as follows. A user on the computer A speaks into a microphone, then computer A digitalized the voice and produces a data stream that is sent towards computer B. After the arrival, the stream is decoded and the voice of user A can be listened through the speakers of the computer B. The same thing happens in the opposite directions, with respect to the user B. This application supports the conversation between user A and user B and hence the application is called *interactive*.

Interactive applications are more difficult to support than normal QoS applications, as they also pose a constraint on the end-to-end delay, which should be not noticeable to the end-users. Conversely, if we consider a non-interactive application, such as an audio broadcast application (e.g., an Internet radio), a large end-to-end delay is allowed, as users do not interact.

The respect of this constraint on the end-to-end delay is hence fundamental for interactive QoS applications. For this reason, several studies [14] [15] [16] [19] [21] [9] [4] investigated the effects of this delay on human perception. These studies 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. It is to note that the value of this threshold is not fixed [16], but 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 has to be). For instance, if we consider interactive audio applications, a threshold of 150 ms ensures full satisfaction to the end-users. In this case, if the end-to-end delay goes above 150 ms [23] the users will experience a bad service, while for values lower than 150 ms, the users can interact without any problems. In Table 1 we show some possible values of the end-to-end delay with regards to an audio application: for end-to-end values below 150 ms the users may notice an imperceptible difference between audio and real speech. The difference increases as the end-to-end increases and for values above 600 ms the speech becomes intelligible and incoherent.

End-to-end delay represents the overall delay between the two end-users. This delay, as pointed out by *Baldi* and *Ofek* [1], is composed by different components: the *processing* delay (the time spent and 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, it is worth analyzing the network delay. This component is, in fact, the most variable. It is essentially composed by two components: *propagation* 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 the Internet, 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. This happens because the Internet is a best-effort network, and hence all the network traffic is treated with the same priority. Even though in the future (Internet2 [31]) there will be different classes of services [30] [17] and hence the queuing delay may be known a-priori, currently the queuing delay is unknown and depends on the overall network traffic characteristics.

The variability of the network delay (the network jitter) may cause QoS problems to the receiver.

For instance, in a video streaming application, the receiver should continuously play out the arriving stream, playing a pre-defined number of frames every second. If the network jitter is present, it is possible that the network delays the delivery of some video frames, causing problems to the receiver's play out as video frames may be not available for play out when needed. This could cause video play out interruptions and it is common for users that watch video streams while being connected to the Internet through a low capacity modem.

In literature there were studies aimed to ameliorate the network jitter: *buffering* or *smoothing* techniques [19] [6] [5] [22] are examples of mechanism developed to mask the network jitter to the end-users. Briefly, these techniques work as follows: 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 receiver starts retrieving (and playing out) video frame from the local buffer only after a portion of time (usually few seconds), called start-up delay. In essence, the application uses the client buffer to mask the network jitter. In this way the receiver's play out should not be affected by the network jitter and hence the receiver can continuously play out 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 start-up delay increases the overall end-to-end delay, a critical measure of the interactive QoS applications. In fact, as pointed out, interactive applications have a threshold that represents the limit above which the human perception (and hence interactions) is affected. If we denote this threshold with NIT (Natural Interaction Threshold), it is mandatory for the end-to-end delay to stay below the NIT value. Since this 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 [29].

For these reasons, we propose a mechanism that does not use any start-up delay and hence, the first video frame that arrives at the receiver is immediately played out.

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 stored at the user side (either the video is stored on the user's hard-disk or on a CD-Rom, DVD, and so on) or ii) the video stream is stored 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 stored. This avoids the network to introduce an excessive delay, causing the end-to-end delay to be noticeable and, sometimes, annoying. 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

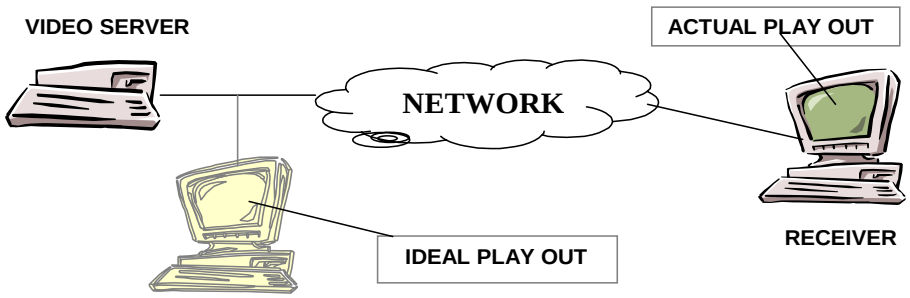

Figure 2: Network scenario while transmitting a video stream between a sender and a receiver. The ideal video play out and the actual video play out are highlighted.

is delivered to the receiver where there is the actual play out of the video stream. In this scenario, the network delays the transmission of the video stream as well as the interactive requests of the user at the receiver side. Needless to say, if the network delays these data with a value greater than the NIT, the natural interaction between the end-user and the video server is compromised. The ideal scenario to avoid these possible problems, would see the user directly connected to the video server, such that the video play out would be done without considering the network. 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 goal of our mechanism is to support the video play out in the real scenario, while attempting to simulate the video play out in the ideal scenario. Hence, our mechanism aims 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, in interactive applications, the end-to-end delay can be seen as the time elapsed between the user request and the relative effects on the user's screen. Needless to say, to provide interactions, this delay should be not noticeable.

The mechanism we propose in this paper is an improvement of the mechanism proposed in [9] where we introduced an adaptive mechanism for supporting video play out over the Internet. The mechanism in [9] used a three-handshake protocol to periodically measure the end-to-end delay and a QoS mechanism was used to maintain the end-to-end delay very close to the value measured with the handshake protocol. Hence, the mechanism adapts the video transmission to the network conditions: it aims to maintain the *status quo* of the network delay, while the mechanism proposed in this paper provides an absolute measure of the end-to-end delay.

To evaluate our mechanism we first collected real network delay traces (obtained transmitting video traces over our department LAN and over the Internet) and then, using these traces, we perform several simulations using our mechanism. Results obtained show that our mechanism is

Audio Latency	Effect of delay on human voice perception
> 600 ms	Speech is intelligible and incoherent
600 ms	Speech is barely coherent
250 ms	Speech is annoying but comprehensible
100 ms	Imperceptible difference between audio and real speech
50 ms	Humans cannot distinguish between audio and real speech

Table 1: Effects of latency on human voice perception

well suited for supporting interactive QoS applications over the Internet, as the actual video play out is kept very close to the ideal video play out.

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 results obtained from evaluating our mechanism. Conclusions are drawn in section 4.

2 Proposed Mechanism

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

As we already pointed out, to support interactive operations in a distributed environment, it is mandatory for each transmitted message to experience an end-to-end delay lower than a predefined threshold. The value of this threshold, named NIT (Natural Interaction Threshold) throughout this paper, is application dependent and represents the upper bound to the end-to-end delay in order to provide natural interaction between end-users. Further, since we are considering applications that transmit video stream, it is important to guarantee the continuity of the video play out (i.e., no interruptions during the video play out) at the receiver side. Roughly, this means that the receiver should always have frames to play out.

As we already stated, we don't use any of the techniques that may increase the overall end-to-end delay, such as smoothing techniques that introduce a start-up delay to ameliorate the network jitter. Conversely, with our mechanism, the receiver starts playing out the video upon the reception of the first video frame.

While being effective on networks that provide some guarantees to the applications, such as sufficient bandwidth, low packet-loss and end-to-end delay, the video play out without any start-up delay may pose problems if the underlying network is a best-effort network. In fact, in a best-effort

network the end-to-end delay may be very variable, causing frames to have an unpredictable arrival time. This may compromise the continuity of the video play out, as the receiver may not have received video frames that are requested for the play out. As we show in section 2.2, this network jitter may also compromise the natural interactions between end-users, as the overall end-to-end delay may exceed the NIT.

As we stated, our mechanism aims to maintain the actual play out very close to the ideal play out, in order to support interactive features. 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 applications is supported with sufficient QoS. By supposing the clocks at the sender and at the receiver side synchronized, we can, through a timestamp mechanism, measure this time difference. The time-difference is measured through a new metric, named VTD (Video Time Difference), that we introduce in order to measure the time difference between the ideal play out of a frame and its actual play out. Briefly, the VTD value is periodically measured at the receiver side, and if its value is within the NIT value, the application is well supported.

Unfortunately, the video stream is transmitted over a best-effort network, and hence it is possible that the network jitter causes the VTD to go above the acceptable threshold. For this reason, our mechanism is provided with a synchronization phase. In essence, when the receiver finds out that the VTD is above the acceptable limit, it informs the sender that a synchronization phase has to be activate. In essence, this synchronization phase affects the video QoS in order to report the VTD within the acceptable NIT. 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 is 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 α depends on the number of frame per seconds δ (i.e., $\alpha = 1/\delta$). 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;

2. The receiver plays out the frames at fixed period (i.e., one frame every $\alpha = 1/\delta$ time units, where δ is the number of frames that should be played during every second);

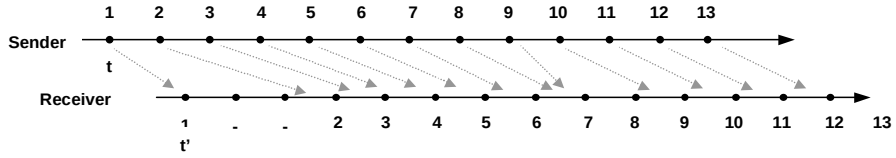

Figure 3: Network scenario while transmitting a video stream between a sender and a receiver.

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) $T_S(i) > T_S(prec(i))$, where $prec(i)$ 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, this last rule says that, if a selected frame i has a timestamp lower than the timestamp of the last frame that has been played out (i.e., $T_S(i) < T_S(prec(i))$) 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 $prec(i)$, but, due to network problems, frame i arrives later that the play out time of the $prec(i)$ 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 $T_R(i) = T_R(prec(i)) + \alpha$, where $prec(i)$ 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, if the previous algorithms are used in the Internet scenario, possible problems may arise, as we describe in Fig. 3, 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 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

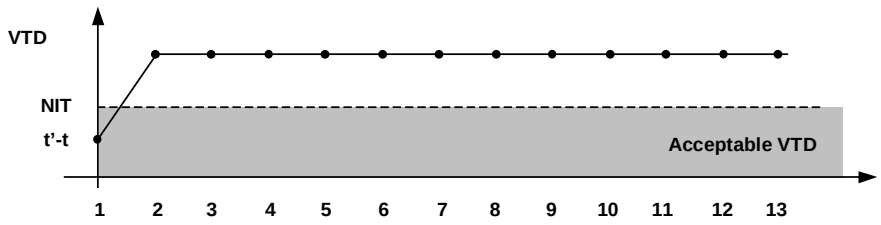

Figure 4: VTD measured while playing out the video stream.

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 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 situation causes problems if interactive operations are allowed, as we describe in the next section.

2.3 Time difference between ideal and actual play out

The situation described in the previous section highlights the importance of 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. Before introducing a mechanism that measures the end-to-end delay, it is to note that the user does not know when a frame arrives at the buffer of his/her host, but he/she notices it when this frame appears on his/her screen.

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. 3. 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 ($t' + 4\alpha$).

This 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 $VTD(i) = T_R(i) - T_S(i)$. ◇

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 4 we show the VTD measured for each played frame, with respect to the scenario described in Fig. 3. 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. As we already stated, this situation happens 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 QoS of the video, 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 $T_R(z) = T_R(prec(z)) + \alpha$, where $prec(z)$ indicates the frame played out just before frame z.

Since the frame played out just before frame z is the frame j ($prec(z) = j$), 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, the discard of k consecutive frames reduces the VTD of ρ time units. ◇

The previous Theorem allow 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. 5, we consider again the example depicted in Fig. 3, 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

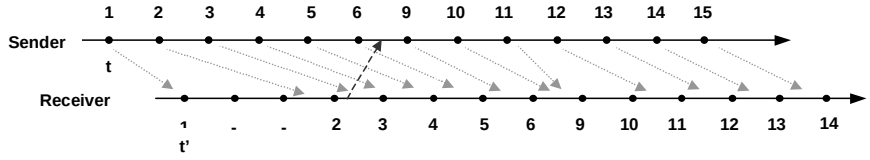

Figure 5: Network scenario while transmitting a video stream between a sender and a receiver when using our mechanism.

(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. 6 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. 4), frame 9 is played out with VTD(9) greater than NIT, but if our mechanism is used (Fig. 6), VTD(9) is lower than NIT. Moreover, using our mechanism, all the frames transmitted after frame 9 are within the NIT value.

This means that when our mechanism is not used, the considered application does not provide sufficient QoS to the interactive applications, as the VTD is often above the NIT. Conversely, our mechanism is able to report the VTD within the NIT, dropping only some video frames. Note that, while evaluating our mechanism, we show that the number of dropped frames is very small.

The drawback of our mechanism is the dropping of some frames and this affects the video QoS. In literature, there are techniques that act on the video QoS (by dropping frames) in order to solve bandwidth allocation problems (see for example, [28] and [10]) and they proposed several heuristic algorithms to discard frames in video stream encoded with Motion JPEG and MPEG techniques. They also showed that a good selection of the frames to discard does not greatly affect the video QoS. These techniques are easier to implement when video is encoded with intra-frame mechanism (like Motion JPEG), as inter-frame mechanisms (like MPEG) can cause a *domino* effect (i.e., a discard of a frame may lead to the impossibility of decoding other video frames). Our mechanism has been designed to handle Motion JPEG video stream and hence it is possible to discard frames without causing the domino effect. On the other hand, if video is encoded with inter-frame mechanisms, the discarding algorithm must take into account the characteristics of any video frame. In this paper we only consider Motion JPEG video streams.

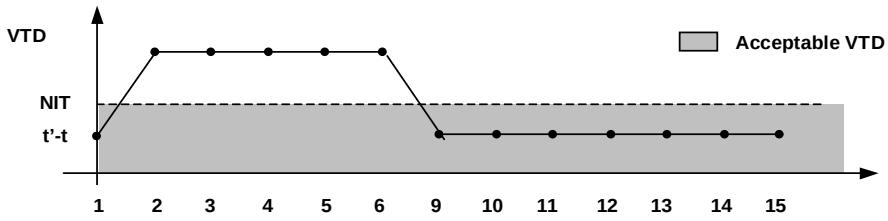

Figure 6: VTD measured while playing out the video stream when using our mechanism.

3 Simulation scenario and results

In this section we present results obtained from several simulations that have been done in order to test our proposed mechanism.

Simulations involve both our department LAN and the Internet and are performed using video delay traces obtained transmitting a set of video traces (each of 20 minutes long) encoded with Motion JPEG, with a resolution of 320x160 pixels and encoded with different number of frames per second (12 fps or 24 fps). These delay traces have been collected during different time (peak hours, office hours, evening) in order to test our mechanism in different situations.

A simulator that uses the collected delay traces to test the behavior of our mechanism has been developed and results obtained over our LAN and over the Internet are presented in the following sections.

3.1 LAN Environment

The first set of experiments has been done over our department LAN, a 100Mb/s Ethernet network. Two sets of experiments have been involved. The first set uses Motion JPEG video traces encoded with 12 frames per second (i.e. one frame every 80 millisecond), while the second set is composed of 24 fps Motion JPEG video traces (i.e. one frame every 40 millisecond). The different number of fps has been chosen in order to have different bandwidth requirements. In fact, the number of frames per second affects the bandwidth necessary to transmit the video over a network and hence, varying the fps, we obtained different network conditions. For instance, if we consider a Motion JPEG frame with a resolution of 320x160, its size may span from some kbytes to more than 10 kbytes. Approximately, a stream composed of frames with these characteristics may need around 960 Kbps. On the other hand, if we consider a video encoded with 24 frames per second, the needed bandwidth is, more or less, the double. Note that, we are considering video encoded with Motion JPEG, an intra-frames technique that produces frames whose encoding is inpedendent of each other.

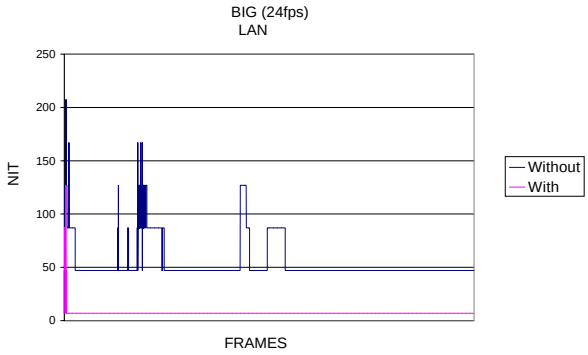

Figure 7: VTD obtained while transmitting a clip of the movie *Big*.

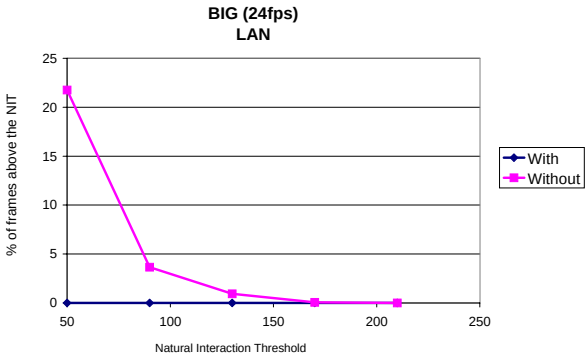

Figure 8: Transmission of *Big*: Percentage of packets with a VTD above the Natural Interaction Limit.

Although 24 fps is used for high quality video, and hence it can be considered oversized for some interactive multimedia applications, such as distance learning and video games, we analyzed these traces in order to test our mechanism in critical traffic condition.

In Fig. 7 we present results obtained from transmitting a video trace of the movie *Big*, encoded with 24 fps. We present the VTD obtained with and without using our mechanism. As shown, when our mechanism is not used, the VTD is variable and reaches the maximum value of 210 millisecond. This can cause problems to interactive application if the application's NIT is lower than 210 ms.

In Fig. 7 is worth noting the behavior of the VTD curve when 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

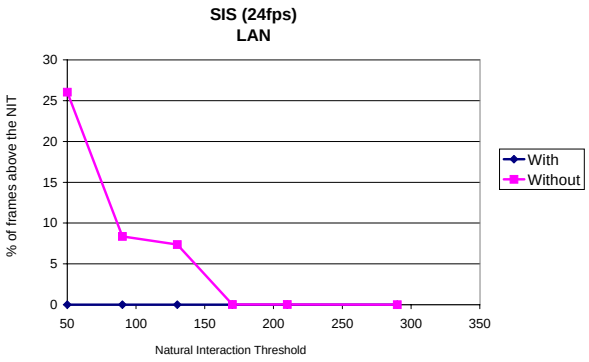

Figure 9: Transmission of *Sleepless in Seattle*: Percentage of packets with a VTD above the Natural Interaction Limit.

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.

So far, we've described the VTD when our mechanism is not used. Since our mechanism is activated when the VTD goes above the NIT limit and since the NIT limit is application dependent, to cover different situations we vary the NIT value from 50 to 210 milliseconds (ms) and we compute the percentage of frames whose VTD goes above the NIT limit.

As shown in Fig. 8, with a 50 ms NIT, more than 20% of the frames goes above the acceptable limit. This percentage decreases while increasing the NIT value. For instance, if the NIT value is equal to 90 ms, the percentage drops to 4% and reaches almost zero percent with a NIT value of 130 ms.

If our mechanism is used, the percentage is kept very close to zero, while dropping very few frames (i.e. 4, 3, 2, 1 frames dropped for a NIT of 50, 90, 130, 170, respectively).

The reason of this great benefits obtained by discarding very few frames is highlighted by the example described in sections 2.3 and 2.4. In fact, in Fig. 4 the delay experienced by frame 2 causes the VTD to go above the NIT limit for all the successive frames. In Fig. 5 and 6 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.

Another test has been performed by transmitting a video trace of the movie *Sleepless in Seattle*, encoded with 24 fps. If Fig. 9 we show results obtained. In this case, we vary the NIT value from

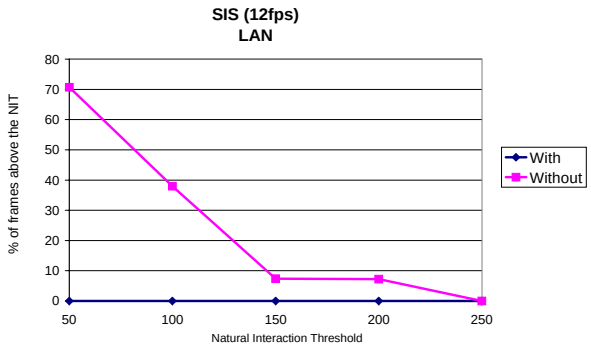

Figure 10: Transmission of *Sleepless in Seattle*: Percentage of packets with a VTD above the Natural Interaction Limit.

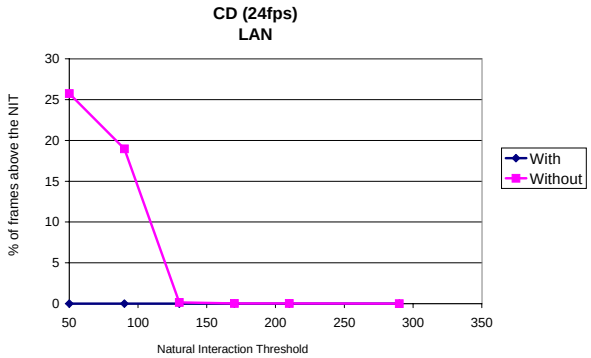

Figure 11: Transmission of *Crocodile Dundee*: Percentage of packets with a VTD above the Natural Interaction Limit.

50 to 290 ms. With a NIT of 50 ms the percentage of video frames that goes above the NIT is equal to 26%. This case, this value decreases as the NIT increases and the percentage goes very close to zero when the NIT is around 170 ms. The benefits obtained when using our mechanism are still considerable, as the percentage is kept very close to zero along the lifetime of the application, dropping very few frames: 6, 2, 2, 1 frames dropped for a NIT of 50, 90, 130, 170, respectively.

Very similar results have been obtained while transmitting *Sleepless in Seattle* encoded with 12 fps (Fig. 13), *Crocodile Dundee* encoded with 24 and 12 fps (Fig. 11 and Fig. 12, respectively) and *Big* encoded with 12 fps (Fig. 13). The benefits introduced showed that our mechanism is well suited for supporting interactive video applications over a local network.

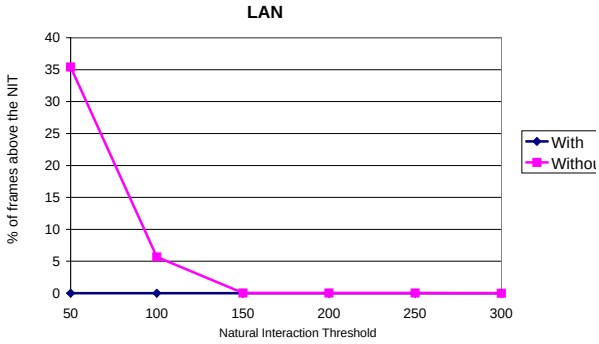

Figure 12: Transmission of *Crocodile Dundee*: Percentage of packets with a VTD above the Natural Interaction Limit.

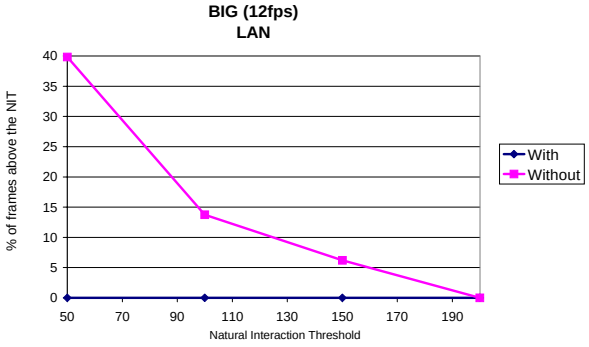

Figure 13: Transmission of *Big*: Percentage of packets with a VTD above the Natural Interaction Limit.

hop	Host
1	csgw-fe10-0.cs.unibo.it (130.136.1.254)
2	cesia-csgw.cs.unibo.it (130.136.254.254)
3	almr59.unibo.it (137.204.1.22)
4	192.12.47.22 (192.12.47.22)
5	192.12.47.5 (192.12.47.5)
6	poseidon.csr.unibo.it (137.204.72.49)

Table 2: Hops experienced from cartoonia.cs.unibo.it to poseidon.csr.unibo.it

hop	Host
1	cs-gw.cs.unibo.it (130.136.1.56)
2	130.136.254.254 (130.136.254.254)
3	almr55.unibo.it (137.204.1.20)
4	rc-unibo.bo.garr.net (193.206.128.97)
5	rt-rc-2.bo.garr.net (193.206.134.157)
6	mi-bo-1.garr.net (193.206.134.1)
7	ts-mi-1.garr.net (193.206.134.50)
8	ictp-rc.ts.garr.net (193.206.132.2)
9	sv3.ictp.trieste.it (140.105.16.63)

Table 3: Hops experienced from cartoonia.cs.unibo.it to sv3.ictp.trieste.it

3.2 The Internet Environment

In the previous section we evaluated our mechanism over our department LAN and results obtained showed the benefits introduced by our mechanism. However, to test our mechanism in the Internet, we evaluated it in two more scenarios: one from Bologna to Cesena (hops are shown in Table 2) and the other from Bologna to Trieste (hops are shown in Table 3).

As we already stated, the capacity needed for transmitting a Motion JPEG encoded video with 12 fps and with a resolution of 320x160 pixel is around 900 Kbps and (more or less) the double for a 24 fps video. While this quantity can be acceptable in a LAN, it can increase the packet loss if video is transmitted over the Internet. In some cases, the percentage of packet loss introduced by the network is considerable, reaching up to 30% of packet loss for a 12 fps video, and more than 40% for a 24 fps video traces.

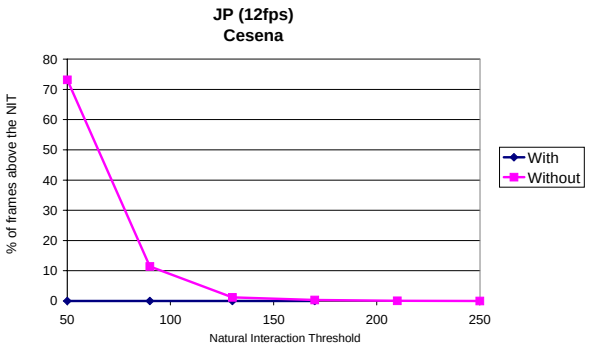

Figure 14: Transmission of *Jurassic Park*: Percentage of packets with a VTD above the Natural Interaction Limit.

In Fig. 14 we present results obtained from transmitting a video trace of the movie *Jurassic Park*, encoded with 12 fps, from Bologna to Cesena (Table 2). Once again, we measured the percentage of frames that goes above the NIT, with a NIT value that ranges between 50 and 250 ms. As shown, with a 50 ms NIT, more than 70% of the frames goes above the acceptable limit. This percentage decreases while increasing the NIT value. For a NIT value of 90 ms, the percentage drops to 11%, and reaches almost zero percent with a NIT value of 130 ms. Also in the Internet environment, if our mechanism is used, the percentage is kept very close to zero, while dropping very few frames (i.e. 3, 3, 1, 1 frames dropped for a NIT of 50, 90, 130, 170, respectively).

In Fig. 15 we present results obtained from transmitting a video trace of the movie *Big*, encoded with 24 fps, from Bologna to Cesena (Table 2). We vary the NIT value from 90 to 290 ms. As shown, a NIT of 90 ms causes 60% of the frames to go above the acceptable limit. This percentage decreases while increasing the NIT value. For a NIT value of 130 ms, the percentage drops to less than 20%, and reaches almost zero percent for a NIT value around 230 ms. Once again, with our mechanism, the percentage is kept very close to zero, while dropping very few frames (i.e. 5, 4, 3, 2 frames dropped for a NIT of 90, 130, 170, 210, respectively).

In Fig. 16 we present results obtained from transmitting a video trace of the movie *Crocodile Dundee*, encoded with 24 fps, from Bologna to Cesena (Table 2). We show the percentage of frames that goes above the NIT, with a NIT value that ranges between 90 and 170 ms. A 90 ms NIT value causes 6% of the frames to go above the acceptable limit. This percentage decreases while increasing the NIT value and reaches almost zero percent with a NIT value of 130 ms. If our mechanism is used, the percentage is kept very close to zero, while dropping very few frames (i.e. 2, 1, 0 frames dropped for a NIT of 90, 130, 170, respectively).

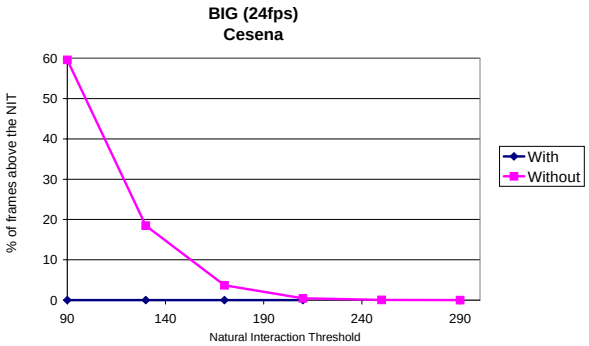

Figure 15: Transmission of *Big*: Percentage of packets with a VTD above the Natural Interaction Limit.

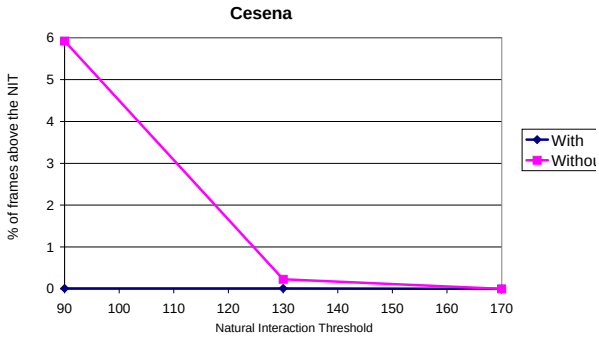

Figure 16: Transmission of *Crocodile Dundee*: Percentage of packets with a VTD above the Natural Interaction Limit.

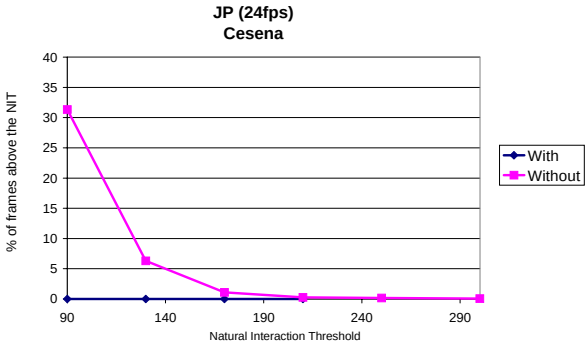

Figure 17: Transmission of *Jurassic Park*: Percentage of packets with a VTD above the Natural Interaction Limit.

In Fig. 17 we present results obtained from transmitting a video trace of the movie *Jurassic Park*, encoded with 24 fps, from Bologna to Cesena (Table 2). We show the percentage of frames that goes above the NIT, with a NIT value that ranges between 90 and 290 ms. As shown, with a 90 ms NIT, 30% of the frames goes above the acceptable limit. This percentage decreases while increasing the NIT value and reaches almost zero percent with a NIT value of 170 ms. If our mechanism is used, the percentage is kept very close to zero, while dropping very few frames (i.e. 13, 12, 3, 2 frames dropped for a NIT of 90, 130, 170 and 210 respectively).

In Fig. 18 we present results obtained from transmitting a video trace of the movie *Big*, encoded with 12 fps, from Bologna to Trieste (Table 3). We show the percentage of frames that goes above the NIT, varying the NIT from 150 to 1000 ms. As shown, with a 150 ms NIT, more than 17% of the frames goes above the acceptable limit. This percentage decreases while increasing the NIT value and for a NIT value of 550 ms, the percentage drops to 4%. In this case, the percentage reaches almost zero percent with a NIT value of 1000 ms. The VTD in this case reaches quite high values and this highlights the difficulties of delivering video streams over the current Internet. However, despite these difficulties, our mechanism allows to have a percentage very close to zero, but these benefits are obtained discarding a larger number of frames. In particular, our mechanism drops a number of frames that varies from 60 to 50 with a NIT of 150 and 1000, respectively.

In Fig. 19 we present results obtained from transmitting a video trace of the movie *Sleepless in Seattle*, encoded with 12 fps, from Bologna to Trieste (Table 3). We show the percentage of frames that goes above the NIT, with a NIT value that ranges between 150 and 800 ms. As shown, with a 150 ms NIT, around 18% of the frames goes above the acceptable limit. This percentage decreases while increasing the NIT value and for a NIT value of 350 ms, the percentage drops to 4%, and

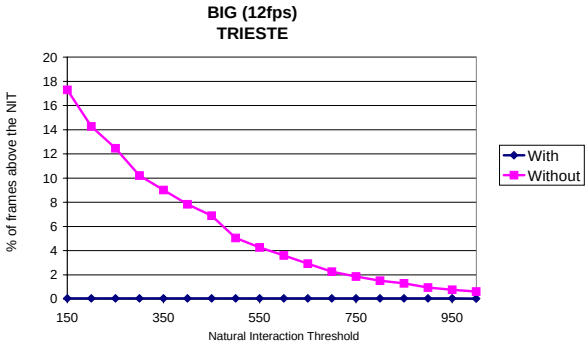

Figure 18: Transmission of *Big*: Percentage of packets with a VTD above the Natural Interaction Limit.

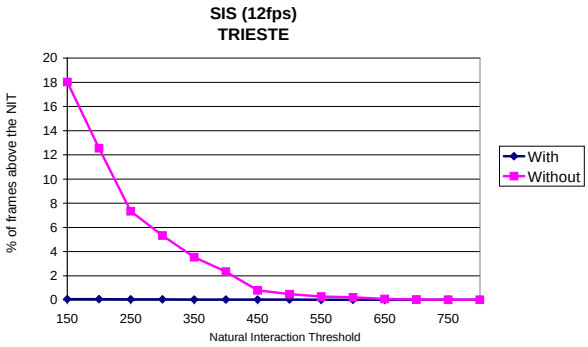

Figure 19: Transmission of *Sleepless in Seattle*: Percentage of packets with a VTD above the Natural Interaction Limit.

reaches almost zero percent for a 550 ms NIT value. In this case, our mechanism is able to keep the percentage close to zero, by discarding very few frames (i.e. 8, 8, 7, 6 frames dropped for a NIT of 150, 200, 220, 250, respectively).

Despite the network problems, all the performed experiments showed the benefits of our mechanism, compared to the number of dropped frames, in keeping the VTD within the acceptable limit in order to support interactivity in networked applications.

4 Conclusions

In this paper we proposed a new mechanism for supporting video streaming applications that provide interactive features to the end-user. As we pointed out, this type of application is very

critical to support in best-effort networks, as the interactions are greatly affected by network jitter.

In order to interact with who provides the video streaming, we highlighted that the client has an ideal position to play out the video (i.e., the video stream is locally stored at the user's side) and an actual position (i.e., where the user actually is: somewhere in the network). Our mechanism aims to maintain the actual video play out very close to the ideal play out.

To provide natural interactions between end-users, the time difference between the ideal and the actual play out must have a value lower than the NIT value (the upper bound to the end-to-end in order to provide natural interaction between end-users). This time difference is measured through a new metric, called VTD, and we proved that is possible, by dropping video frames, to reduce the VTD of an arbitrary time quantity. Through the VTD, our mechanism measures whether the time difference between the actual and the ideal play out is within the NIT threshold or not. If not, our mechanism computes the number of frames that is necessary to drop in order to report the VTD within the acceptable limit and then discards them.

Our mechanism deals with Motion JPEG video streams to simplify the selection of the frames that is necessary to drop. However, since it would be very interesting to handle also MPEG video streams, we will consider these streams in future investigations.

We evaluated our mechanism through several simulation, performed by using real network delay traces obtained transmitting Motion JPEG 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. These results showed that our mechanism is well suited for supporting video applications that provide interactive features to the end-user (e.g., distance learning, pay per view, etc.), over the Internet.

References

[1] Baldi, M., Ofek, Y., *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.

[2] Cen, S., Pu, C., Walpole, J., : *Flow and congestion control for Internet streaming applications*, Multimedia computing and networking, 1998.

[3] Claypool, M., : *The Effects on Jitter on the Perceptual Quality of Video*, ACM Multimedia, November 1999.

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

[5] Feng, W., Sechrest, S., *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.

[6] Feng, W., Sechrest, S., *Optimal buffering for the delivery of compressed prerecorde video*, Proceedings of the the IASTED/ISMM International Conference on Networks, January 1995.

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

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

[9] Furini, M., Roccetti, M., *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.

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

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

[12] 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.

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

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

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

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

[17] Nichols, K., Jacobson, V., Zhang, L. : *A Two-bit Differentiated Services Architecture for the Internet*, Internet Draft, 1998.

[18] Podolsky, M., Vetterli, M., McCanne, S., ; *Limited retransmission of real-time layered multimedia*, IEEE Workshop on multimedia Signal Processing, December 1998.

[19] Ramjee, R., Kurose, J., Towsley, D., Schulzrinne, H., : *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.

[20] Rangan, P.V., Kumar, S.S., Rajan, S. : *Continuity and Synchronization in MPEG*, IEEE Journal on Selected Areas in Communications 1996.

[21] Roccetti, M., Ghini, V., Pau, G., Salomoni, P., Bonfigli, M.E. : *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.

[22] Salehi, J.D., Zhang, Z.L., Kurose, J., Towsley, D., *Supporting Stored Video: Reducing Rate Variability and End-to-End Resources Requirements through Optimal Smoothing*, IEEE/ACM Transactions on Networking, September 1998.

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

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

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

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

[27] Xu, X., Myres, A., Zhang, H., Yavatkar, R., : *Resilient multicast support for Continuos media applications*, Workshop on Network and OS Support for Digital Audio and Video (NOSSDAV97), 1997.

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

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

[30] http://www.ietf.org/html.charters/diffserv-charter.html

[31] www.internet2.edu