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

Marco Furini
DIST A Department
University of Piemonte Orientale
Alessandria, Italy
email: marco.furini@mfn.unipmn.it

Marco Roccetti
Department of Computer Science
University of Bologna
Bologna, Italy
email: roccetti@cs.unibo.it

ABSTRACT
Interactive video streaming applications are becoming more popular also in the Internet environment and studies showed that interactive 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. In this paper we propose a new approach that 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.

KEY WORDS
Multimedia Networking, Multimedia over Packet-based Networks, Multimedia Communication Systems

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 on Demand, 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 if the video is compressed (MPEG [7], Motion JPEG [3], H.261 [8]) the resulting stream can require high network capacity compared to the (usually) available in the Internet. In addition, the transmission of this traffic has, at least, two time-constraints: minimal communication delay and network jitter [12]. These time-constraints 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, called interactive, enables natural interactions (i.e., more life-like as possible) among end-users. The Voice over IP application is an example of these interactive QoS applications.

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.

Several studies [9] [10] [4] investigated the effects of the end-to-end 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 application lifetime. The threshold is the limit below which the interactions are well supported; above this bound the end-to-end delay is noticeable and interactions are not well supported. It is to note that the value of this threshold is not fixed [9], 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 delay up to 150 ms ensures full satisfaction to the end-users, while for values above 150 ms the users will experience a bad service [12].

End-to-end delay is composed by different components [1]: the processing delay (the time used 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 (the delay introduced to cancel the network 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, and queuing delay. The propagation delay depends on the network capacity and on the size of the data to transmit. Conversely, the queuing delay is very variable and unknown a-priori, as data travel from source to destination along a path that is usually shared among traffic generated by other applications. Hence, it may happen that a network resource along the path is busy, causing the data to be delayed until the resource is available.

The variability of the network delay (the network jitter) may cause QoS problems to the receiver. For instance, if we consider a video streaming application, the network jitter may cause video play out interruptions, as video frames may be not available for play out when needed.

In literature there were studies aimed to ameliorate the network jitter: buffering or smoothing techniques [2] [11] are some of these studies. Although these techniques are very effective, they cannot be used to support inter-
active QoS applications, as they introduce a start-up delay that increases the overall end-to-end delay. 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. Since for video application the NIT 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 to ameliorate the network jitter [14]. For these reasons, we propose a mechanism that does not use any start-up delay.

Before introducing our mechanism, we first note that the video stream can be located in two possible sides: i) at the user side (either the video is generated with a webcam, or it is locally stored) or ii) somewhere in the network.

The ideal scenario to perform interactive operations is when the video is locally available. This avoids the network to introduce an excessive delay, causing the end-to-end delay to be noticeable and, sometimes, annoying. In Fig. 1 we show a possible scenario for a video application. A video server transmits a video stream into the network. The stream is delivered to the receiver where it is played out. 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 NIT, the interactive operations are compromised. Conversely, if the user is directly connected to the video server, the play out can be done without considering the network delay. Throughout the paper, we refer to this scenario as the ideal scenario for providing interactions in contrast to the actual scenario.

The goal of our mechanism is to support the video play out in the actual scenario, while attempting to simulate 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.

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 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

As pointed out, to support interactive operations, it is mandatory for each transmitted message to experience an end-to-end delay lower than NIT. NIT 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 video streaming applications, it is important to avoid interruptions during the video play out at the receiver side. Roughly, this means that the receiver should always have frames to play out.

Since our mechanism does not introduce a start up delay, 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 a start-up delay may pose problems if the underlying network is a best-effort network. In fact, in best-effort networks 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 shown in section 2.2, this network jitter may also compromise the natural interactions between end-users.

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 application 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. The time difference is measured through a new metric, named VTD (Video Time Difference), which is periodically measured at the receiver side.

Unfortunately, in the Internet the network jitter may cause the VTD to go above the acceptable threshold. For this reason, our mechanism is provided with a synchronization phase that reports 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 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, located at the sender side, controls the video stream transmission; 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. The encoding process establishes the number of frames that must be displayed every second and the video play out algorithm has to display the video frames according to this number. In the following we denote this number with the parameter \( \delta \).

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, which represents the ideal play out time of the frame, 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 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 \( t' \);

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;

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) \( T_S(i) > T_S(\text{prec}(i)) \), where \( \text{prec}(i) \) 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, the 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., \( T_S(i) < T_S(\text{prec}(i)) \)) then frame \( i \) is discarded and a new frame selection must be done. This is done to avoid the play out of a frame \( i \) that has been transmitted before the transmission of the frame \( \text{prec}(i) \), but, due to network problems, arrives later than the play out time of the \( \text{prec}(i) \) frame.

Based on the previous rules, the algorithm plays out a frame \( i \) at time \( T_R(i) = T_R(\text{prec}(i)) + \alpha \), where \( \text{prec}(i) \) indicates the frame played out just before frame \( i \). 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 if the underlying network provides guarantees such as low communication delay and jitter. Conversely, possible problems may arise in the Internet: in Fig. 2 a sender, at time \( t \), starts transmitting video frames every \( \alpha \) time units. At time \( t' \), the receiver 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, if frame 2 arrives between \( t' + 2\alpha \) and \( t' + 3\alpha \), at time \( t' + \alpha \), as well as at time \( t' + 2\alpha \), the receiver has no frame to play out. Hence, 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.

Figure 2. Video stream transmission.

Figure 3. VTD while playing out the video stream.
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

As we already stated, in interactive applications, the end-to-end delay must stay within the NIT value. It is also to note that there is a difference between the arrival time of a frame at the end-host and its play-out time. Consider again the example in Fig. 2: Frame 3 arrives just after \( t' + 3\alpha \), but it is played out at time \( t' + 4\alpha \).

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 \( \text{VTD}(i) \), 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 \( \text{VTD}(i) = T_R(i) - T_S(i) \).

To better understand the effects of the network jitter on the VTD, in Fig 3 we show the VTD measured for each played frame, with respect to the scenario described in Fig. 2. A hypothetical NIT value is also depicted in order to compare the VTD with the NIT.

Since we supposed that \( T_S(1) = t \) and \( T_R(1) = t' \), it follows that \( \text{VTD}(1) = t' - t \). Frame 2 arrives later than expected and it is played out at time \( t' + 3\alpha \). Hence \( \text{VTD}(2) = t' - t + 2\alpha \). If \( \text{VTD}(2) > \text{NIT} \) then the VTD of the successive frames is affected by the network problem experienced while transmitting frame 2. In fact, \( \text{VTD}(j) \geq \text{NIT} \), 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.

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 (\( \text{VTD} > \text{NIT} \)). To report the VTD within the NIT, we design a mechanism that acts on the video QoS, by discarding video frames.

In fact, as we show in the following Theorem, it is possible to reduce the VTD of an arbitrary time quantity, by dropping video frames. The proof is straight and it is reported in [5].

Since the previous Theorem allows us to reduce the VTD of a known quantity and since we exactly know the amount of time that exceeds the acceptable limit NIT (i.e., \( \text{VTD} - \text{NIT} \)), it is easy to report the VTD within the NIT.

Our mechanism works as follows. The receiver computes the value \( \text{VTD} - \text{NIT} \). If this value is greater than zero, it is sent to the sender. If the sender receives this message, it can discard a number of frames that corresponds to the time quantity \( \text{VTD} - \text{NIT} \). In this way the VTD is reduced and it is reported within the acceptable NIT limit.

To show the effects of our mechanism, in Fig. 4, we consider again the example depicted in Fig. 2, 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 \( \text{VTD}(2) \) is greater than \( \text{NIT} \)) it sends to the sender the value \( \rho \equiv \text{VTD}(2) - \text{NIT} \). When this message arrives at the sender, it is used 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. 5 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. 3), frame 9 is played out with \( \text{VTD}(9) \) greater than \( \text{NIT} \), but if our mechanism is used (Fig. 5), \( \text{VTD}(9) \) is lower than \( \text{NIT} \). 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 that it 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, [13] and [6]) and they showed that a good selection of the frames to discard does not greatly affect the video QoS.
3 Simulation scenario and results

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

Simulations involve both our department LAN and the Internet and are performed using video delay traces obtained transmitting a set of Motion JPEG video traces (each of 20 minutes long, 320x160 pixels, 12-24 frames per second). A simulator that uses the collected delay traces to test the behavior of our mechanism has been developed.

3.1 LAN Environment

The first set of experiments has been done over our department LAN, a 100M/s Ethernet network. Two sets of experiments have been involved: 12 fps video traces and 24 fps video traces. Although 24 fps can be considered oversized for some applications, such as distance learning and video games, we analyzed these traces in order to test our mechanism in critical traffic condition.

Due to the lack of space we only present a subset of the performed experiments. More can be found in [5].

Our mechanism is activated when the VTD goes above the NIT and since the NIT is application dependent, to cover different situations we vary the NIT value from 50 to 210 ms and we compute the percentage of frames whose VTD goes above the NIT limit. As shown in Fig. 6, 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. Conversely, 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 benefit obtained by discarding very few frames is highlighted by the example described in sections 2.3 and 2.4. In fact, in Fig. 3 the delay experienced by frame 2 causes the VTD to go above the NIT limit for all the successive frames. In Fig. 4 and 5 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.

Very similar results have been obtained while transmitting Sleepless in Seattle (12-24 fps), Crocodile Dundee (12-24 fps) and Big (12 fps). Details can be found in [5].

3.2 The Internet Environment

To test our mechanism over the Internet, we evaluated it in two more scenarios: one from Bologna to Cesena (6 hops) and the other from Bologna to Trieste (9 hops). Details can be found in [5].

The packet loss increases when 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.

In Fig. 7 we present results obtained from transmitting a video trace of the movie Big (24 fps), from Bologna to Cesena (6 hops). A NIT of 90 ms causes 60% of the frames to go above the acceptable limit. 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. 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. 8 we present results obtained from transmitting a video trace of the movie Sleepless in Seattle (12 fps), from Bologna to Trieste (9 hops). With a 150 ms NIT,
around 18% of the frames goes above the acceptable limit. For a NIT value of 350 ms, the percentage drops to 4%, and 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).

Similar results have been obtained from analyzing Big (12 fps), Crocodile Dundee and Jurassic Park (24 fps). Readers can refer to [5] for details.

Despite the network problems, all the performed experiments showed the benefits of our mechanism 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 approach for supporting interactive video streaming applications. We highlighted that the client has an ideal position to play out the video (i.e., the video stream is locally available 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 play out very close to the ideal play out.

The time difference between the ideal and the actual position is measured through a new metric, called VTD. To provide natural interaction, the VTD must be lower than the NIT value (the upper bound to the end-to-end delay in order to provide natural interaction). If, for network problems, the VTD goes above the NIT, our mechanism acts on the video Qos in order to report the VTD within the NIT.

Our mechanism has been evaluated through simulations and results obtained showed that our mechanism is well suited for supporting interactive video applications (e.g., distance learning, video on demand, etc.), over the Internet, providing benefits to the end-users. For instance, in video-on-demand applications, our mechanism ameliorates the system reaction to the end-user command (pause, fast forward, rewind).

References


