Application Level Hand-off Support for Mobile Media Transcoding Sessions

Sumit Roy, Bo Shen, Vijay Sundaram[1], Raj Kumar
Client and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2002-91
April 11th , 2002*

Mobility, transcoding, streaming media, hand-off

Media transcoding can be used to enable mobile devices that have low resolution and low bit-rate capabilities to access content created for stationary, desktop clients with high bandwidth connections. By deploying transcoding servers near the edge of a large network, it becomes possible to support clients with multiple resolutions, while limiting the bandwidth requirements in the core network. In this paper we look at the problem of supporting mobility of such device by seamlessly handing off media transcoding sessions between servers. We explore different hand-off solutions, starting with the transfer of sufficient amount of state so as to produce byte identical streams when compared to the case when there is no hand-off. However it is found that such a scheme introduces a considerable amount of hand-off delay at the client, due to the large amount of data that needs to be transferred. We propose, analyze and experimentally evaluate more efficient hand-off schemes that reduce delay by coarsening the granularity of the hand-off, and relaxing the requirement for byte identical outputs. We find that these scheme introduce no noticeable degradation in the output quality. These hand-off schemes can also be used to provide load-balancing at the transcoder, or for fault tolerance.

[1] Currently at the Department of Computer Science, University of Massachusetts, Amherst, MA, 01003, work performed while an intern at Hewlett-Packard Laboratories.

ABSTRACT

Media transcoding can be used to enable mobile devices that have low resolution and low bit-rate capabilities to access content created for stationary, desktop clients with high bandwidth connections. By deploying transcoding servers near the edge of a large network, it becomes possible to support clients with multiple resolutions, while limiting the bandwidth requirements in the core network.

In this paper we look at the problem of supporting mobility of such device by seamlessly handing off media transcoding sessions between servers. We explore different hand-off solutions, starting with the transfer of sufficient amount of state so as to produce byte identical streams when compared to the case when there is no hand-off. However it is found that such a scheme introduces a considerable amount of hand-off delay at the client, due to the large amount of data that needs to be transferred. We propose, analyze and experimentally evaluate more efficient hand-off schemes that reduce delay by coarsening the granularity of the hand-off, and relaxing the requirement for byte identical outputs. We find that these scheme introduce no noticeable degradation in the output quality. These hand-off schemes can also be used to provide load-balancing at the transcoder, or for fault tolerance.

1 Introduction

The ever increasing popularity of the Internet has brought heterogeneous devices together. Users are accessing web-pages and e-mail from their desktops, laptops, Personal Digital Assistants, as well as cellular phones. At the same time, there is greater expectation from users to obtain rich media services via their access devices. When delivering streaming video over the Internet, some key challenges and concerns include the ability of the receiving device to decode compressed video, regardless of its display space and available network capacity. This necessitates content adaptation or media transcoding algorithms that adapt compressed media, originally created for high-bandwidth, thick clients, into compressed media content that can be viewed by lower bandwidth thin clients, such as wireless hand-held devices. For example, the original compressed media content may have been coded at a higher resolution and a higher bit rate, say 720×480 at 2 to 8 Mbps for DVD quality, or 320×240 at 1.5 Mbps for desktop clients connected to the Internet through a T1 line. However, due to the characteristics of mobile wireless communication systems, i.e., low bandwidth channels and limited display space on clients, a 100 kbps video at a lower resolution may be desired. Current 3G wireless communication is trying to provide a $128 - 384$ kpbs communication channel. Therefore, a transcoder is needed to adapt the compressed media content to the appropriate size and bit rate.

With the introduction of fast compressed domain transcoding algorithms [1, 2, 3, 4] and efficient implementations on modern processors [5], real time video transcoding can be achieved. Thus, video transcoding services can be deployed as an Active Service [6] or become part of a Content Services Network [7].

A mobile, wireless client would thus try to access a video transcoding server that is close (in a network sense). However, since the client is moving during the session, it is likely to be closer to another transcoding server after some time. In the absence of any hand-off mechanisms, the network path length between the client and original transcoding server would constantly increase. The increase in latency would lead to intolerable inter packet delay and loss in video quality at the client.

Transport level hand-off and mobile IP like solutions can only be applied to achieve location transparency for the client [8]. Thus, new content can be obtained from a proxy server that is closer to the current location of the client. Appropriate buffering techniques at the client can also deal with handing-off simple media streaming sessions [9]. However, an on-going transcoding session has considerable state associated with it, and thus requires application level hand-off. The alternative is an ever increasing network distance between the original transcoding server and the moving client. Moreover, wireless clients like cell-phones are power constrained and one would like to have as small a memory buffer as possible on the client.

In this paper we explore some solutions to the problem of application level hand-off for supporting video transcoding sessions for mobile clients. We start with a simple implementation that essentially performs a process migration [10] by transferring complete state to the new transcoder. This produces byte identical streams when compared to the case when there is no hand-off. However, it is found that such a scheme introduces a considerable inter packet hand-off delay at the client due to the large amount of data that needs to be transferred between the transcoding servers. We propose, analyze and experimentally evaluate more efficient hand-off schemes that reduce this delay, while introducing no visible degradation in the output quality. These schemes require less processing overhead at the client, and require minimal buffering. These hand-off schemes could also be used to provide load-balancing at the transcoder and to add fault tolerance.

The rest of the paper is organized as follows: We formulate the hand-off problem in Section 2. Section 3 provides some background on the compressed domain video transcoding system. In Section 4 we describe the basic issues involved in session hand-off as applied to our scenario. We analyze the performance of the various proposed hand-off approaches in Section 5. Section 6 shows our experimental results using these approaches. Section 7 concludes the paper.

2 Problem Formulation

Consider a conventional video streaming service with the streams located at a content server, as illustrated in Figure 1. If the service has to support clients with different

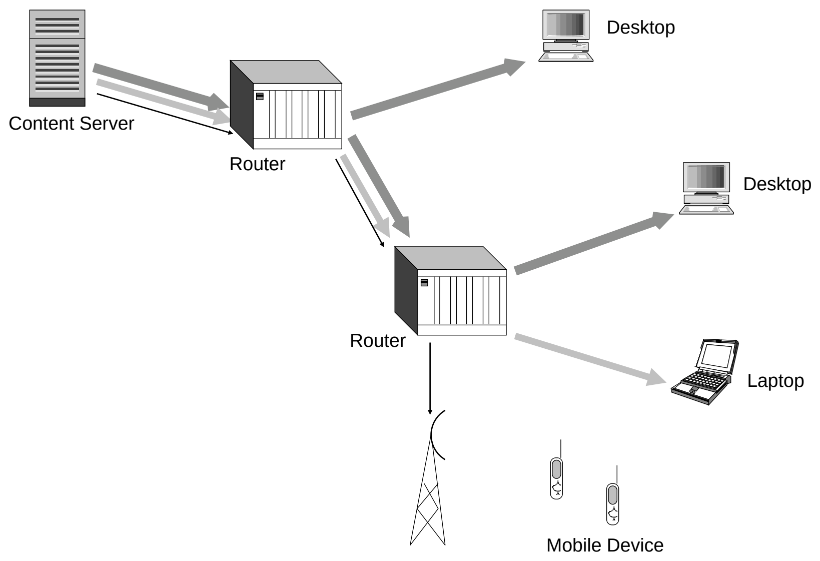

Figure 1: High bandwidth requirement with conventional multi-resolution media on Content Server

display sizes and available bandwidths, e.g. desktops, laptops, and mobile phones, multi-resolution copies of the video stream have to be created and streamed from the content server. These multiple versions have to traverse the network from the content server, via the routers, to the different clients. In the figure, the bandwidth requirements of the various clients have been illustrated by using different line thicknesses.

To reduce the bandwidth requirement in the network core, a transcoding service could be provided on media processing units [11], or co-located with video proxy servers [12]. When clients with different capabilities access the same content, only one stream (that for the most capable or *thickest* client) needs to be sent from the origin server. The content with lower bit-rate and lower resolution is created as close to the edge of the network as possible. This is illustrated in Figure 2. Note that we no longer have to stream the content for the laptop and mobile phones right from the content server. Instead, lower resolution and lower bit rate versions of the video stream are created by a transcoding server, in this case co-located at a router.

By reducing the bandwidth requirements in the network core, the loss probability of packets in the video stream is also reduced. This leads to a better video quality as experienced by the user. This scenario is especially relevant for streaming of live video, where there is a single source of the full resolution video stream. On the other hand,

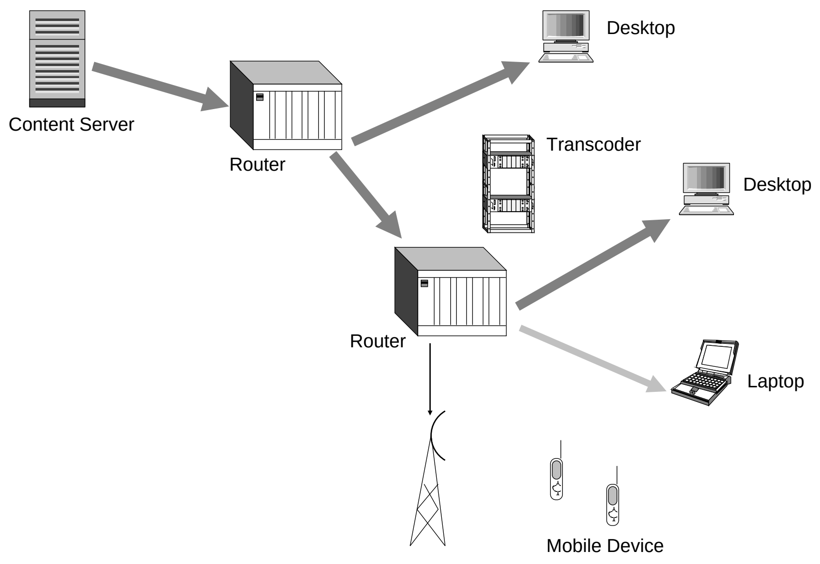

Figure 2: Lower core bandwidth requirements with Content Transcoding Service

pre-recorded video could be stored on video proxy-servers that could form part of a streaming media Content Distribution Network (CDN). In this case, one might argue that there is no load on the core network. On the other hand, by providing a video transcoding service, one needs to store only one version of the video near the edge. This reduces storage requirements in the CDN. Thus, by locating the transcoding servers near the edge of the network, both live and recorded streams can be supplied efficiently.

The media transcoding hand-off problem arises due to the mobility of thin devices that have initiated a transcoding session. The challenge is to seamlessly migrate the session so that it stays close to the client (in a network sense), thus providing high quality video to the user. It differs from the streaming session hand-off problem, since considerable state may be built up within the transcoding servers.

3 Compressed Domain Video Transcoding

Video transcoding as considered in this paper is defined as generating a compressed bit-stream at lower resolution and/or lower bit-rate given the bit-stream of the original video. The conventional transcoding approach requires that the original video be decompressed and that motion vectors be recomputed for the down-scaled video. This is followed by re-encoding in a conventional video encoder. Video compression standards such as MPEG [13] employ motion compensated prediction to exploit temporal redundancy and achieve a lower bit rate. Motion estimation is often employed in the motion-compensation process; however, this process is very compute-intensive and typically is at least 60% of the workload of the video encoder. Therefore, recomputing the motion vectors causes the problem of video down-scaling from a compressed bit-stream to be a computationally intensive task. This places a heavy burden on a transcoding server that may have to carry multiple transcoding sessions in real time.

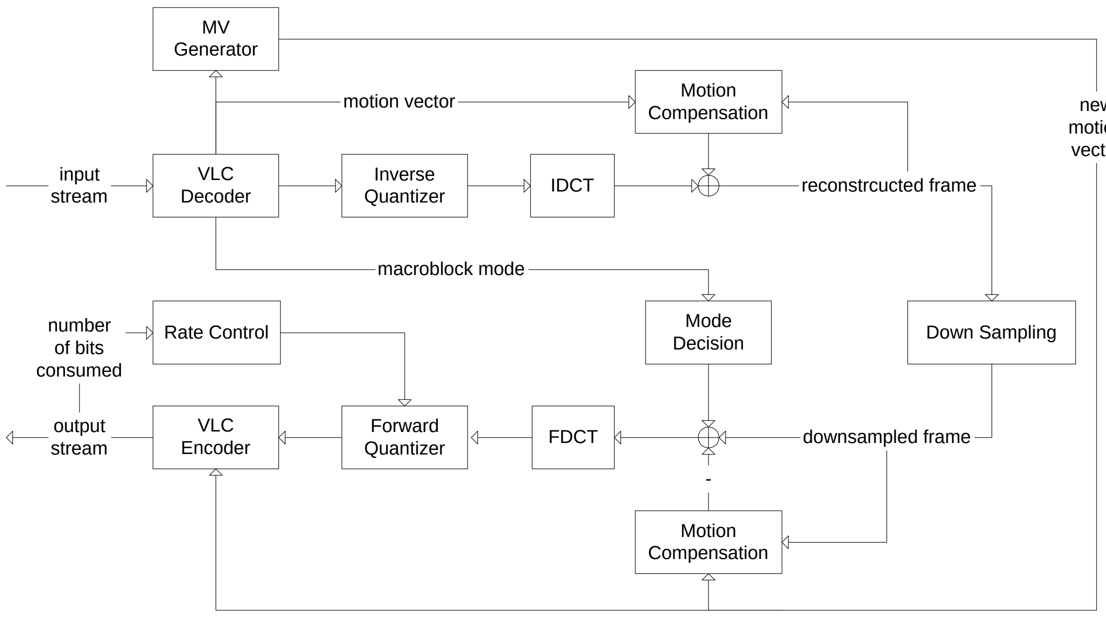

Figure 3: Compressed Video Transcoding System

Figure 3 illustrates the processing flow of a compressed-domain transcoding system for MPEG video [3]. The Variable Length Coder (*VLC*), *Quantizer*, and *Motion Compensation* modules in the decoder and encoder paths are standard building blocks for MPEG systems. The *MV Generator* module is responsible for generating the new motion vector by averaging the existing motion vectors. In the transcoding system, the spatial frames are reconstructed and down-scaled in the spatial domain but the motion vectors are estimated directly from the existing motion vectors in the original sequence. Therefore, a costly motion estimation process is saved. Similar systems can also be developed for the down-scaling of H.261 and H.263 bit-streams [1].

In addition, the coding type of the output macroblock is also decided using the coding types of the input macroblocks. This task is accomplished by the *Mode Decision* module. For instance, if down-scaling by a factor of two, there are four input macroblocks involved in generating one output macroblock. If most of the input macroblocks are intra, the output macroblock is coded as intra. Conversely, if most of the input macroblocks are predicted, the output macroblock is coded as predicted, in which case, the output macroblock is constructed using the new motion vector produced by the *MV Generator* module. On the other hand, if the output macroblock is decided to be predicted, but there is one intra macroblock, one skipped macroblock among the input macroblocks, we treat the intra macroblocks and skipped macroblocks as predicted macroblocks with zero-valued motion vector. Note that the skipped macroblocks in bi-directionally interpolated frames in MPEG or H.263 video may have non-zero-valued motion vectors.

The bit rate of the output bit-stream is controlled by the *Rate Control* module. The *Rate Control* module also uses the compressed domain information existing in the original stream. It first estimates the number of bits available to code the picture then computes a reference value of the quantization parameter based on buffer fullness and target bit rate. Finally, it derives the value of the quantization parameter from the generated reference according to the spatial activity of the macroblock. The spatial activity is derived from the DCT coefficient activity in the input macroblocks. A detailed analysis of the algorithm is available in [3, 4].

There are two types of state information associated with the transcoding process: reconstructible state information and dependent state information. Reconstructible state information can be recreated given an input stream. It consists of reference frame data (both original and down-sampled ones) and meta-data such as macroblock level side information. Dependent state information includes data derived from the output stream. For example, the *Rate Control* module takes the number of bits consumed so far to evaluate the bit budget that can be allocated for the next coding unit. The volume of reconstructible state information is usually much larger than that of the dependent state information. In general, a transcoder needs to maintain and communicate at least dependent state information for a session hand-off since the output stream is not shared between the transcoding servers.

4 Session Hand-off Overview

As discussed in Section 2, a transcoding session hand-off is required when the movement of a client causes the current transcoding server to be inefficient for the client's new location. For example, after a client moves, a different transcoding server may be more directly in line with the shortest network path between the original content server and the new location. When this occurs, migrating the transcoding session from the original transcoding server to the more direct transcoding server increases the efficiency of the overall system by reducing the excess network resources that must be used to route the media to the more distant transcoding server. In this section, we discuss the basic requirements necessary for a transcoding session hand-off between two transcoding servers, and we describe various hand-off options that can be used depending on which entities initiate and control the actual transcoding hand-off.

Figure 4 shows the path a video stream takes from the *Content Server* to the *Mobile*

Client. The client initially requests a video stream from *Transcoding Server 1 (TS1)*. This server requests the full resolution, high bit rate stream from the *Content Server*. After the *Content Server* starts sending the video, *Transcoding Server 1* reduces the resolution and bit rate of the video streams and forwards it to the *Mobile Client*. When the *TS1* receives a notification that the client has moved away from it, and closer to some *Transcoding Server 2 (TS2)* , it initiates a *Session Hand-off* handshake. *TS2* requests the same video stream from a content server (usually with an offset corresponding to the current position in the file). Once the second server starts transcoding and streaming to the *Mobile Client*, *TS1* can shut down its own session.

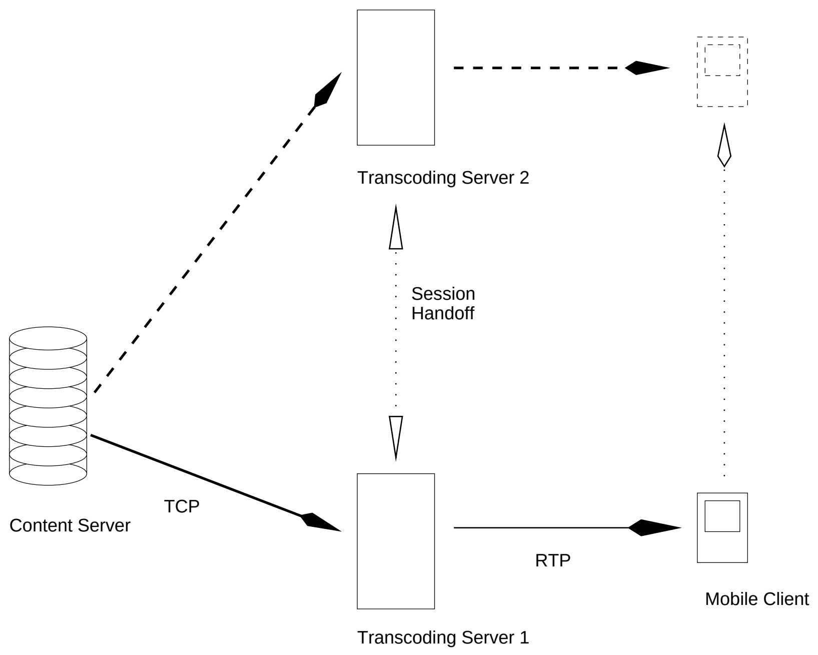

Figure 4: Transcoding Hand-off Setup

4.1 Basic Information Requirements

In principle three pieces of information should be reliably communicated for a successful hand-off:

1. The parameters for the transcoding session, including specifications from the clients request, the current position (offset) in the video stream, and some way of locating the video object, e.g., via a Uniform Resource Locator. This requirement would also be present when performing a simple streaming hand-off [9]. The *Content Server* should preferably have the ability to seek to a specified file offset. Otherwise, *TS2* could request the video stream from the beginning and discard incoming data from the *Content Server* until the offset is reached.

2. Video specific header information associated with the file, that the new transcoder will need in order to transcode the video file. Since this information is typically present at the beginning of the video stream, it is possible to cache it on *TS1* and transfer it from there. Sometimes this information is provided as part of the control handshake, for example using the Session Description Protocol [14]. Alternately, *TS2* could request some prefix of the video from the content server and parse it.

3. State information that the transcoder maintains internally while transcoding the video object. This includes reconstructible state information, like the pixel values for reference frames that *TS2* needs to reconstruct dependent frames, transcoded reference frames for computing residuals, as well as dependent state information like current rate control information to ensure that the bit-rate specified by the client is maintained.

4.2 Hand-off Protocol Design

The three elements in the path shown in Figure 4, the *mobile client*, the *content server*, and the *transcoding server*, need not be involved equally in the hand-off protocol. The hand-off could be initiated by the client, the content server, or the transcoder, or by some monitoring agent within the network. After initiation, the actual protocol could also be controlled by any of these entities. In the qualitative analysis below, we assume that a relatively high bandwidth network connection exists between the servers, and a reliable transport protocol like TCP is used. The rate controlled video stream would be carried over a protocol like RTP over UDP.

4.2.1 Client Controlled Hand-off

A client may decide to hand-off if it sees a lot of packet loss in the stream arriving from one transcoder, or if it receives a signal from the transcoder to switch to another transcoder. This approach has a number of problems. If the transcoders do not talk to each other, then all the information listed in Section 4.1 has to go a long path from the initial transcoder via the client to the new transcoder. This transmission would have to be over a reliable transport protocol. If the client is a mobile client, it has limited power, and it would be best if its involvement were minimal in the hand-off.

4.2.2 Content Server Controlled Hand-off

A content server may initiate a hand-off if it gets a signal from the transcoder that it is overloaded. However this requires that the server maintains state information for all of its transcoding sessions. This is clearly not a very scalable solution, even though a content server has limited content, it has to maintain information for every session. If the content server is implemented as a distributed video proxy, the state would have to be replicated in some fashion. Also, similar to the client controlled case, the server may have to update its state information from the old transcoder before it can send it to the new transcoder. Finally, this approach implies that transcoding can not be transparent to the content server.

4.2.3 Transcoder Controlled Hand-off

A transcoder is thus a preferred place to control the hand-off, since all the information needed to do the hand-off *i)* the video header information, *ii)* the file offset of the file that the transcoder is transcoding and *iii)* the transcoder state information, is available. In this scenario only the two transcoders need to talk to each other without the involvement of either the content server or the client. The only requirement at the client is the ability to buffer and reorder packets if there is an overlap of transmission from both transcoders. The content server only needs to be able to send a file from some specified offset. This approach thus maintains a high degree of transparency with regards to the client and original content server.

In the remainder of the paper, the term *hand-off* is applied to transcoder controlled hand-off. The hand-off initiation could still be due to a signal from the client, a network monitor, or due to a overload at the first transcoding server.

5 Analysis of Hand-Off Schemes

The analysis focuses on the extra propagation delay seen by the client due to a hand-off of the transcoding session. It is assumed that similar and relatively consistent network conditions exist for the links between the transcoders and the client at the instant of the hand-off. The transcoders perform transcoding in real time. The resulting bits are accumulated in a fixed-size packet buffer and subsequently sent to the client when the buffer is full. We assume that the packet size used for video streaming is small enough so that there is a negligible delay between the instant when the packet buffer is empty and the instant a full packet has been sent out. Based on these assumptions, we propose the following hand-off schemes:

5.1 Single Step Explicit Switch (SSES)

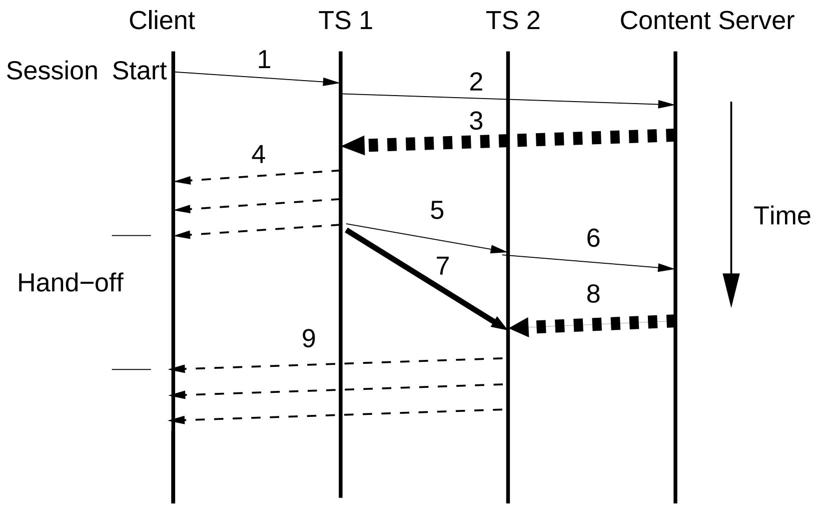

Figure 5: SSES Hand-off Timeline

Figure 5 shows the traffic between the entities involved in a transcoding hand-off. At some point in time the *Client* initiates a transcoding *Session Start* by sending message *1* to the first transcoding server *TS1*. *TS1* requests the original video object from the *Content Server* with message *2*. Data flow *3* denotes the stream of the original video object. *TS1* converts the stream to a lower resolution and lower bit rate version, and streams an approximately constant bit-rate stream, *4*, to the client.

Message *5* is the hand-off start message, and it can be combined with or immediately followed by the state transfer message *7*. While *TS1* transfers its state, *TS2* can concurrently initiate a new session with the *Content Server*, using message *6*. Once the *Content Server* starts streaming the original video to *TS2* using flow *8*, and the complete state has been transfered, *TS2* creates and transmits the transcoded stream *9* to the client. Note that *TS1* stops transcoding as soon as it initiates the transfer, and *TS2* can only proceed with its transcoding session by the explicit arrival of the state information. Hence we call this method single stage with explicit switching (SSES) hand-off.

For transcoding sessions without hand-off, the client receives packets in constant time interval. This is not the case with hand-off. To evaluate the performance of the hand-off scheme, we use inter packet delay to represent the interval between continuous packets that arrive at the client. The inter packet delay for SSES hand-off is thus expressed as follows:

Source Format	Resolution (pixels)	Frame Data (bytes)	Meta Data (bytes)	Total Transferred	
				→ CIF (bytes)	→ QCIF (bytes)
CCIR 601	720 × 480	2073600	648000	3352064	2895872
CIF	352 × 288	608256	190080	-	972608
QCIF	176 × 144	152064	47520	-	-

Table 1: Amount of state transferred for SSES. The total includes fixed overheads.

$$\mathit{Delay} = t_{\text{start}} + \max[t_{\text{xmt}}(S_1), t_{\text{setup}}]. \tag{1}$$

where the components are:

t_{start} :	Time for sending start message to TS2
$t_{\text{xmt}}(x)$:	Time to send x bytes from TS1 to TS2
t_{setup} :	Time for a session setup with the Content Server by TS2
S_1 :	The amount of state information

The amount of data that needs to be transferred S_1 depends on whether the transfer is done at a packet, frame, or group of pictures (GOP) boundary. Full state information has to be sent during hand-off, and this scenario can happen at the packet boundary, i.e., whenever the output packet buffer is full. The dominant terms in the state transfer data set include the information required to do reconstruction of the next frame of original input video and transcoded output video. This includes pixel information as well as side information such as macroblock mode. Table 1 gives the amount of data required for the the state transfer. The major components are the YUV pixel information of the reference frames for the original video, required for performing reconstruction at the new transcoder; macroblock information for the reference frames; as well as YUV pixel information for the reference frames for the down-sampled video.

The resulting state transfer time for transcoding between different standard video sizes on networks of different available bandwidths is shown in Figure 6.

Note that the t_{start} message only includes information to locate the original video file (e.g., using a URL), and the current file offset. Thus the time taken is governed by the latency between *TS1* and *TS2*. Also, the t_{setup} can be made very small by replicating content at the transcoding server. Hence the state transfer component tends to dominate the delay time. It is not possible to hide this delay unless one uses buffering at the client, or *TS2* can transcode much faster than real time. If there is

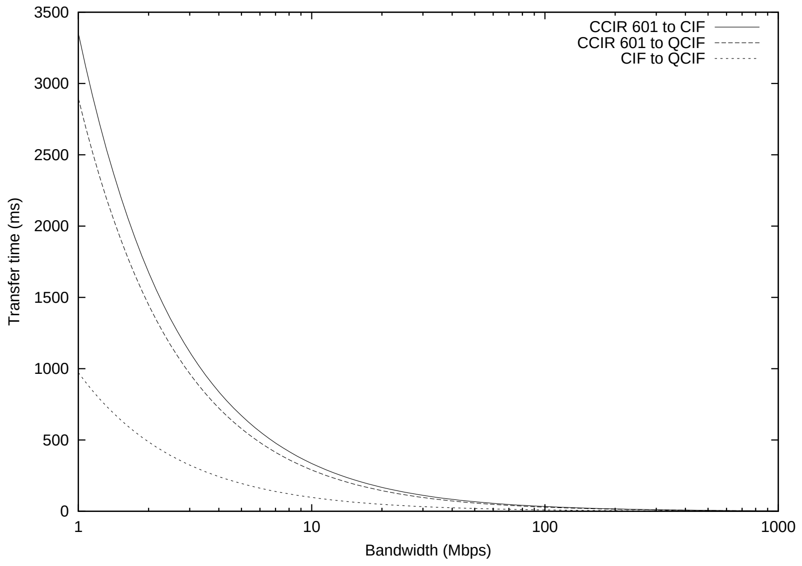

Figure 6: State transfer time for different transcoding pairs and bandwidths

no back-channel, then the new transcoder would send data at the same fixed rate as the original transcoder. Thus, after multiple hand-offs, the client would eventually drain out its buffer, and the video quality would deteriorate. On the other hand, by transferring complete state, the stream resulting from the hand-off is byte-identical to a stream that is not handed-off at all.

5.2 Two Step Explicit Switch (TSES)

In the SSES scheme, the dominant state transfer term arises since the transcoding hand-off can occur at any time in the session. To reduce the amount of state information that needs to be transmitted, we can select to have the hand-off at a frame/picture boundary. In this scenario, a hand-off request may appear when *TS1* is in the midst of transcoding a compressed frame. *TS1* starts the hand-off, but waits until the current frame is finished before sending the final switch message.

As shown in Figure 7, *TS1* sends the *Session Start* message *1* to the *Content Server* as before, and a transcoded stream *4* is generated. At hand-off time, message *5* is sent to *TS2* with the video content information and offset. *TS2* can contact the *Content Server* for the video stream via setup message *6*. At the same time, *TS1* sends partial state information *7* that *TS2* needs to start transcoding. *TS1* then finishes transcoding the current frame. After sending the last packet (composed by whatever transcoding result is left in the output buffer) to the client, *TS1* sends a switch message *9* to *TS2* along with the final state information. For example, if the current frame is a P-picture in MPEG's case, the reference picture needs to be sent to *TS2*.

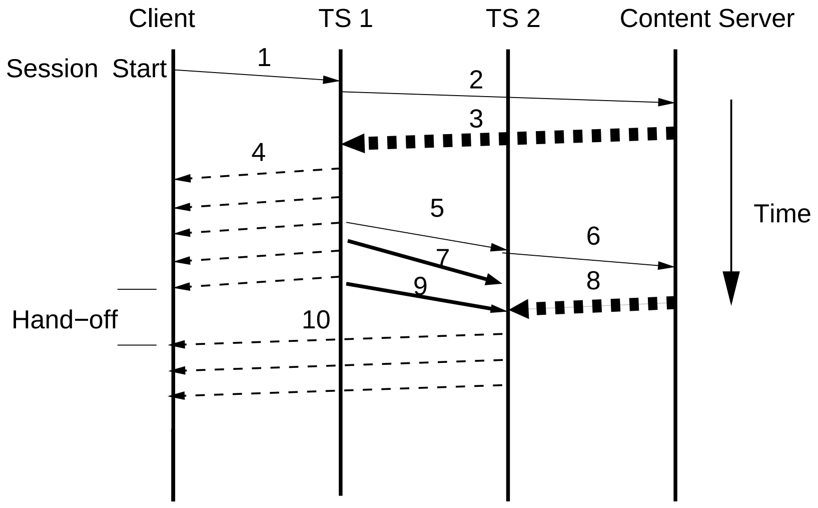

Figure 7: TSES Hand-off Timeline

In this scheme, hand-off start and switch happen in two stages and the switch message is explicitly sent by *TS1*. Therefore, this hand-off is called two-stage hand-off with explicit switch (TSES). When *TS2* first gets the start message, it establishes contact with the original server and start parsing the streamed packets from it after an initial setup. As an additional optimization, *TS2* can locally create the reconstructible state information as outlined in Section 3. Thus the amount of state that has to be transferred is reduced by duplicating some computation at the two transcoding servers. The parsing operation needs to only peek into the input bit stream and look for a picture start code. In either case, *TS2* does not send any bits until the immediate next picture header is encountered. If the switch message from *TS1* has not yet arrived, *TS2* waits for the dependent state information; otherwise, *TS2* starts transcoding and subsequently streaming the result to the client. Therefore, the client perceived inter packet delay is defined as:

$$\begin{aligned} \mathit{Delay} &= \max[t_{\mathrm{xmt}}(S_2), (t_{\mathrm{start}} + t_{\mathrm{setup}} \\ &\quad + t_{\mathrm{xcode}}(n) - t_{\mathrm{x\&s}}(n))]. \qquad (2) \end{aligned}$$

where the components are:

$t_{\mathrm{xmt}}(x)$: Time to send x bytes from TS1 to TS2

t_{start} : Time for sending start message to TS2

t_{setup} : Time for a session setup with the Content Server by TS2

$t_{\text{xcode}}(n)$	:	Time to transcode n frames on TS2
$t_{\text{x\&s}}(n)$	:	Time to transcode n frames on TS1 and send them to the client
S_2	:	The amount of state information

Here n is determined by the hand-off granularity, i.e. in case of MPEG it could be a picture or a GOP. The t_{start} and t_{setup} behave exactly as discussed for SSES. The $t_{\text{x\&s}}(n)$ term should be smaller than the $t_{\text{xcode}}(n)$, since *TS1* has to throttle the transfer of bytes to satisfy the bit-rate requirement at the client. *TS2* on the other hand can transcode the same data at full speed, since the only objective is to set up the reconstructible state information. Hence the term $t_{\text{xmt}}(S_2)$ is expected to dominate the delay.

If the hand-off boundary is a GOP, only rate control information needs to be sent, since a group of picture (GOP) can be relatively independent from the previous GOP. Note that there is still dependency if an open GOP [13] is involved at the switch point. With less dependency, S_2 becomes small, therefore the delay is reduced.

5.3 Two Step Implicit Switch (TSIS)

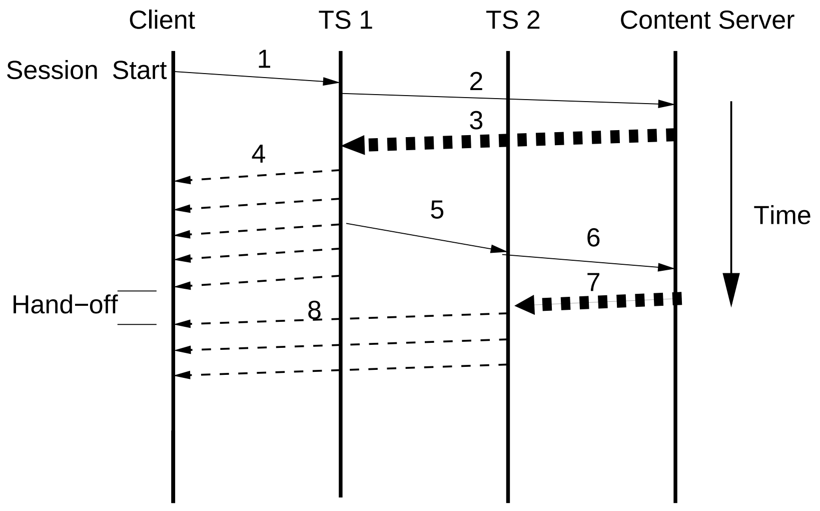

Figure 8: TSIS Hand-off Timeline

To further hide the delay from the client, one has to eliminate the $t_{\text{xmt}}(S_2)$ term from Equation 2. In this scenario the switch happens at a GOP boundary or some type of boundary that is relatively less time-dependent. As shown in Figure 8 and similarly to TSES, *TS1* continues transcoding after sending the start message *5* to prepare *TS2* for the setup. After establishing a connection with the content server using message *6* , *TS2* starts transcoding data immediately. The transcoding is only used to recover reconstructible state information that maybe used in the future. No output bit stream is generated, therefore no packets are sent to the client. There will be no explicit switching message sent from *TS1* and *TS2*, rather, *TS1* and *TS2* pre-agree that the switch will happen at the next GOP boundary. Hence this approach is called two-stage hand-off with implicit switching (TSIS). In this approach, no state information is sent from *TS1* to *TS2*.

TS1 will stop transcoding at the next GOP boundary; on the other hand, *TS2* will start the transmitting of transcoding results at the next GOP boundary. The client perceived inter packet delay is therefore derived as follows:

$$\begin{aligned} Delay &= t_{\text{start}} + t_{\text{setup}} \\ &\quad + t_{\text{xcode}}(N) - t_{\text{x\&s}}(N). \qquad (3) \end{aligned}$$

where the components are:

t_{start}	:	Time for sending start message to TS2
t_{setup}	:	Time for a session setup with the Content Server by TS2
$t_{\text{xcode}}(n)$	:	Time to transcode n frames on TS2
$t_{\text{x\&s}}(n)$	:	Time to transcode n frames on TS1 and send them to the client
N	:	The number of frames till the next GOP

The key issue here is that the transcoders pre-agree on how much computation will overlap, since the hand-off will always occur at a GOP boundary. The possible drawback of this technique is that the *Delay* term can be negative, i.e., the first few packets from *TS2* arrive before the last packets from *TS1*. This can be solved by having a small reordering buffer at the client. Moreover, no back-channel is required at the client to avoid buffer overflow. Instead the values of t_{start} and t_{setup} are estimated from the network. Then the amount of *Delay* can be controlled by adjusting the number of GOPs that the two transcoders overlap. Thus, the hand-off protocol can be adapted to network delays and client buffering capabilities.

From another standpoint, since the hand-off always happens at the GOP boundary, it could happen that the start message is sent right before the last frame of a GOP is transcoded on *TS1*. In this case, *TS2* may not have enough time to catch up, effectively, t_{setup} could dominate the hand-off delay. To overcome this problem, *TS1* and *TS2* can pre-agree to switch in two GOPs, therefore at least one GOP worth of time can be guaranteed for *TS2* to catch up.

6 Experimental Results

For the experiments we stream an MPEG1 video sequence between the content server and a client via two transcoding servers. The video sequence has 982 frames of 352 $\times$ 240 pixels, and is coded at 30 fps with 15-frame GOPs. To ensure repeatability, the hand-off always occurs exactly after 32 GOPs. The content server is an HP Netserver LH6000, with two 700 MHz Xeon processors, 512 Kbyte full speed L2 cache, 1.2 Gbyte main memory, and the content is stored on a hardware RAID 0 partition. The transcoding servers are two HP NetServer LP1000r PCs, with 1.4 GHz Pentium III processors, 512 Kbyte full speed L2 cache, and 256 Mbyte main memory. The client is an HP Kayak XU800 with a 733 MHz Pentium III processor, 256 Kbyte full speed L2 cache. All the machines are connected together via a 100 Mbps HP ProCurve 4000 Ethernet switch.

We repeat this experiment using the approaches described in the previous section. For TSIS we show results with one and two GOP overlaps. For each approach we record the packet arrival times at the client, and also capture the transcoded output to a file for quality assessment.

Figure 9 shows the difference in arrival time for each continuous pair of packets with respect to the session time. This delta time should ideally be a constant value for a constant bit-rate stream as produced by the transcoder. However, the figure clearly shows that the SSES scheme introduces a large delta at the hand-off point. This leads to a correspondingly degraded video experience at the client, since the video stream falls approximately 12 frames behind at this point. The delay time is about 400 ms for the down-scale-by-two transcoding. This agrees well with the numbers predicted from Figure 6. Some additional delay is caused by the TCP slow start property. Note that some RTP data packets arrive at the client after hand-off has started due to the threaded implementation of our transcoder.

In contrast, Figure 10 shows the same metric for the TSES scheme. Here there is a much smaller delta in the inter-packet arrival delay. However, note that the experiment had very good network conditions. Again, due to the threaded implementation of the transcoders, some packets from *TS1* arrive interleaved with packets from *TS2*.

Figure 11 shows the packet arrival versus the sequence numbers for each of the cases. Note that the break in the sequence number is due to the hand-off from one transcoder to the other. We have restricted the horizontal range to show the region of interest, i.e., around the hand-off point.

From the graph, the inter packet delay seen at the client for SSES is approximately 400 ms. Since we choose to have a TCP connection for the hand over of important

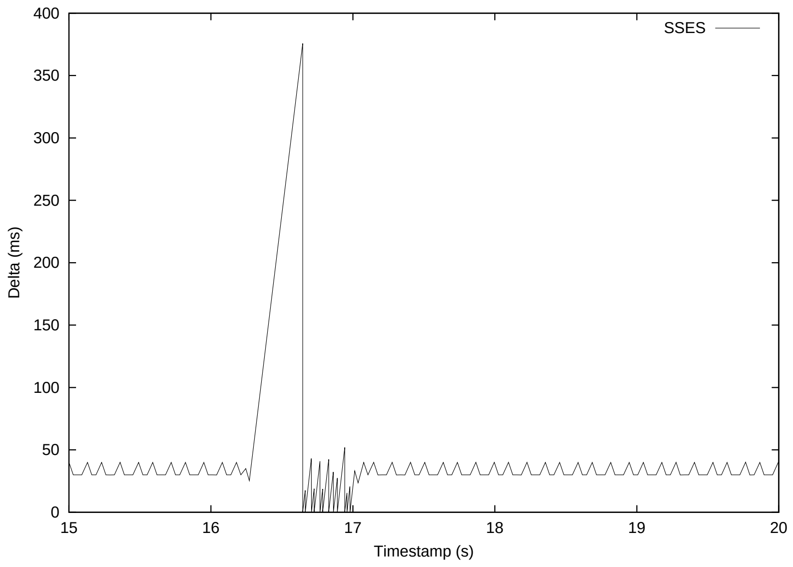

Figure 9: Packet Arrival Deltas for SSES scheme

state information, the slow start property may hinder the performance of the hand-off, especially when the state information contains a significant amount of data. The state information in this experiment includes reconstructible and dependent state information. For this set of experiments, the total state data transferred adds up to 900 kbytes.

For TSES hand-off, the client perceived delay is expected to be smaller regardless of the connection between *TS1* and *TS2* since less state information needs to be sent. In addition, the state information is sent in two separated instances. The TCP transmission time for the second part of the state information is further hidden by the difference between $t_{\text{xcode}}(n)$ and $t_{\text{x\&s}}(n)$. Indeed, the parsing of the input bit stream till the next picture header on *TS2* is usually much faster than the transcoding and streaming of the remaining part of the frame on *TS1*. As seen in the experiment, there is hardly any noticeable delay at the client. However, this may also be because the experiments are conducted in close to ideal lab environments.

In the TSIS scheme, the ideal case is when $t_{\text{start}} + t_{\text{setup}} + t_{\text{xcode}}(N) = t_{\text{x\&s}}(N)$, therefore the delay is zero. In practical cases, either a negative or positive delay is expected. A negative delay indicates that the client may receive packets out of order. In other words, a client buffer is needed or packet dropping is expected. The rationale of the selection of the GOP boundary as the switch point is because it gives *TS2* sufficient time to catch up.

We also calculated the Mean Square Error (MSE) of the handed-off video with respect

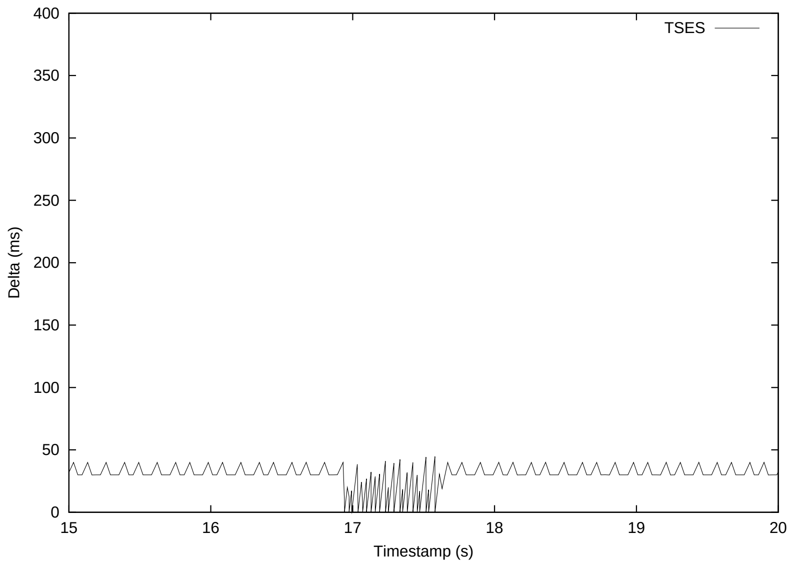

Figure 10: Packet Arrival Deltas for TSES scheme

to the output from a transcoding session without any hand-off. Figure 12 shows the MSE during the hand-off period for TSIS measured when the hand-off is done in 1 GOP. Note that SSES gives a byte identical stream. The large glitch for the first B frame is because the transfer was done across an open GOP and only incomplete state was transferred. However, the overall error is less than one unit, thus this relaxed hand-off approach still provides very good quality video.

Figure 13 shows the MSE when using TSIS with 2 GOP overlap. Some of the initial noise terms drop off, since the second transcoder has some extra time to catch up with the first transcoder.

7 Conclusion

This paper presents some solutions to the problem of handing-off media transcoding sessions, in particular to provide support to mobile clients. In general, an adaptive transcoding hand-off algorithm can be designed to select a method based on four parameters:

1. The network condition between the transcoding servers, *TS1* and *TS2*.
2. The network conditions between the transcoding servers and the *Content Server*.
3. The computational load on *TS1*.
4. The computational load on *TS2*.

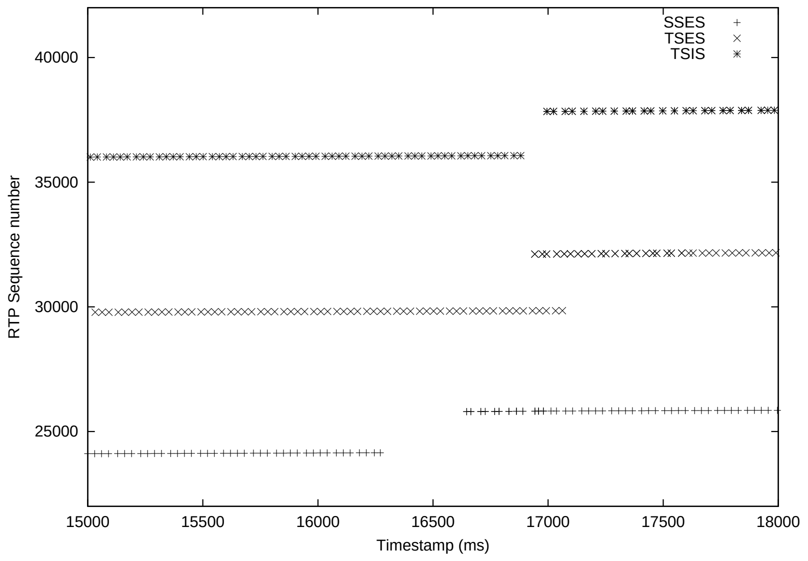

Figure 11: Packet Arrival Times

In particular, we have presented three different schemes, SSES, TSES, and TSIS. Among these schemes, SSES has the least amount of extra transcoding overhead. In addition, this hand-off scenario can sustain an abrupt failure at *TS1*. On the other hand, it is hard to hide the extra inter packet delay at the client since a large amount of state needs to be transferred to the new transcoder *TS2*. The TSES scheme reduces the amount of state transferred, and thus requires a medium amount of communication between the transcoding servers. We achieve this by having a coarser granularity in terms of the hand-off point, and by using two steps to transfer the state. However, the final message latency could still produce a large inter packet gap at the client and affect video quality. The final scheme, TSIS has a larger computational overhead. Minimal state is transferred between the transcoding server, and the resulting video stream is no longer bit-identical to the original stream. However, we show that the Mean Square Error introduced is very small. At the same time, this scheme has the potential of requiring least buffering at the client.

8 Acknowledgments

We would like to thank John Apostolopoulos and Susie Wee for their detailed and helpful comments on initial drafts of this paper. We also thank the anonymous reviewers for their feedback.

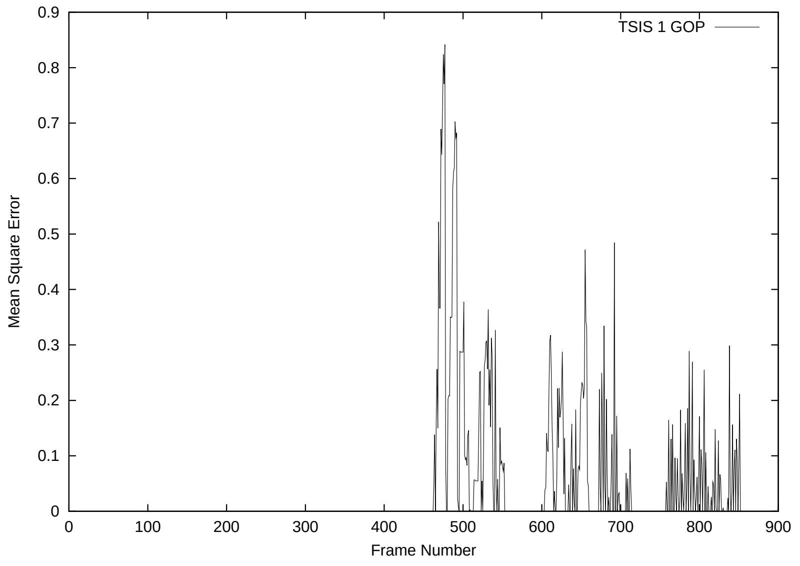

Figure 12: MSE during Hand-off for TSIS with 1 GOP overlap

Bibliography

[1] S. J. Wee and N. Feamster, "An MPEG-2 to H.263 transcoder," in *Proceedings of the SPIE Voice, Video, and Data Communications Conference*, (Boston, MA), September 1999.

[2] S. J. Wee, J. G. Apostolopoulos, and N. Feamster, "Field-to-frame transcoding with temporal and spatial downsampling," in *Proceedings of the IEEE International Conference on Image Processing*, (Kobe, Japan), October 1999.

[3] B. Shen, I. Sethi, and V. Bhaskaran, "Adaptive Motion-vector Resampling for Compressed Video Downscaling," *IEEE Transactions On Circuits and Systems for Video Technology*, vol. 9, pp. 926 – 936, September 1999.

[4] B. Shen and S. Roy, "A very Fast Video Spatial Resolution Reduction Transcoder," in *To appear in Proceedings of ICASSP 2002*, (Orlando, FL), May 2002.

[5] S. Roy and B. Shen, "Implementation of an Algorithm for Fast Down-Scale Transcoding of Compressed Video on the Itanium," in *Proceedings of the 3rd Workshop on Media and Streaming Processors*, (Austin, TX), pp. 119 – 126, December 2001.

[6] E. Amir, S. McCanne, and R. Katz, "An Active Service Framework and its Application to Real-time Multimedia Transcoding," in *Proceedings of SIGCOMM'98*, (Vancouver, B.C.), 1998.

[7] W.-Y. Ma, B. Shen, and J. Brassil, "Content Services Network: the Architecture

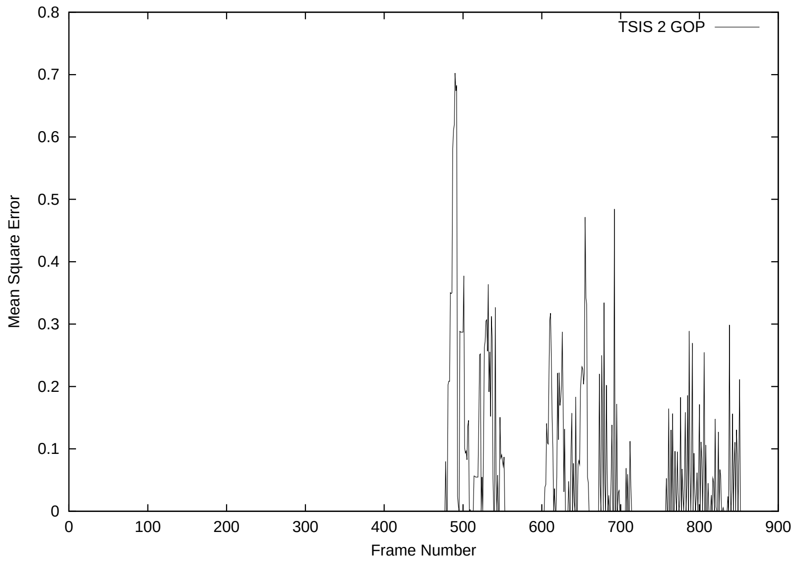

Figure 13: MSE during Hand-off for TSIS with 2 GOP overlap

and Protocols," in *Proceedings of the International Web Content Caching and Distribution Workshop*, (Boston), June 2001.

[8] H. Balakrishnan, S. Seshan, and R. H. Katz, "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks," *ACM Wireless Networks Journal*, vol. 1, December 1995.

[9] R. Karrer and T. Gross, "Dynamic Handoff of Multimedia Streams," in *Proceedings of NOSSDAV'01*, (Port Jefferson, NY), pp. 125 – 133, June 2001.

[10] D. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, "Process Migration Survey," *ACM Computing Surveys*, September 2000.

[11] J. Boyce, M. Cortes, and J. R. Ensor, "Audio/Video Messaging for Multiple Devices," in *Proceedings of NOSSDAV'00*, June 2000.

[12] S. Acharya and B. Smith, "MiddleMan: A Video Caching Proxy Server," in *Proceedings of NOSSDAV'00*, 2000.

[13] J. L. Mitchell, W. B. Pannebaker, C. E. Fogg, and D. J. LeGall, *MPEG Video Compression Standard.* Standard, Chapman and Hall, 1995.

[14] M. Handley and V. Jacobson, "SDP: Session Description Protocol." RFC 2327, April 1998.