Time Scale Modification for 3G-Telephony Video

Michele Covell
Google Inc.
Mountain View, CA USA
Email:michelecovell@gmail.com

Sumit Roy
Rhythm NewMedia
Mountain View, CA 94041 USA
Email: sroy@rhythmnewmedia.com

Bo Shen
and Frederic Huve
Hewlett-Packard Company
Email:bo.shen,frederic.huve@hp.com

***Abstract*— Streaming video to mobile devices like cellular phones is an important emerging application. However, mobile, thin clients have limited capabilities and offer restricted interfaces to the end-user and the Telco infrastructure has significant technical barriers which limit the introduction of new protocols. This paper describes our design of an end-to-end interactive video service. We outline the technological and user-interaction challenges of providing one typical type of interactive controls – time-scale modification (TSM), when streaming compressed audio-video data within the Telco infrastructure in a way that maintains audio-video synchronization and respects the video frame- and bit-rate contracts.**

I. Introduction

Streaming video to mobile devices, such as video-enabled handsets, opens promising new markets. This capability can reduce the perceived duration of on-hold telephone-operation delays, by showing the customer movie trailers or other interesting visual content. Adding a video channel to a support-desk call allows the caller to show the call-center expert the problem. Adding a visual back channel (expressions, gestures, posture) lessens the gap between remote and local interactions. Being able to select and watch professionally produced material, like movies or sports, while away from a home or office environment greatly expands the market for VoD. As seen with the revenue generated by the ring-back tones and the push-to-talk markets, new Telco-services markets can be quite large and can occur in unexpected areas.

The promise of fresh markets is one of the driving forces behind the use of 3G wireless standards [1]. The promise of these markets is largely unrealized to date. One major barrier to exploring new markets in the area of telephony-based video services is the difficulty in creating new interactive applications that conform to Telco requirements, particularly 3GPP-enabled Telco. The primary weakness of the telephony-oriented streaming is the lack of native support for interactive mid-session control.

Using time-scale modification (TSM) as an example, we propose a video streaming solution that provides support for interactive control using existing telephony standards. Thus, we do not require modification of the standard 3G handset, nor do we require that any specialized application be installed. For interactive changes in playback speed, this requires careful modification of the audio and video timing and synchronization. In addition, for sped-up playback, it requires dynamic transcoding at the video server. Video transcoding itself is a well studied topic, but much less work has been done in the context of sped-up video playback which requires careful design and implementation of dynamic sampling modification.

Note that this paper specifically addresses the problem of providing telephony based video services, like video voice mail, video-enabled customer support [2], etc. Internet based video systems, like the recently introduced VCast service from Verizon Wireless [3] or MobiTV from SprintNextel [4] use the handset as a data terminal for packet streaming services. Thus, they are based on a browser and media player paradigm, while our system leverages the call based approach already available on the phone.

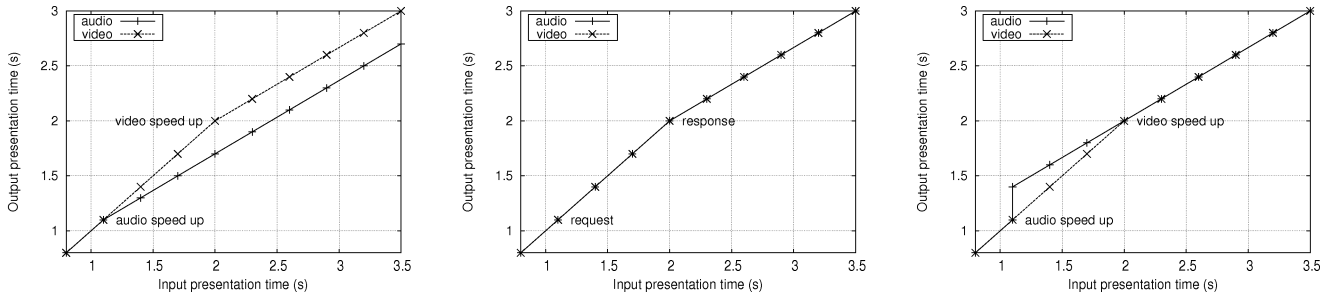
In the rest of the paper, we explain our timing and synchronization solutions in Section II. We study the frame rate and bit rate transcoding solutions unique to video TSM in Section III. We validate the algorithm designs with an implementation of a video server with TSM services, a video demonstration is provided in Section IV. Concluding remarks are provided in in Section V.

II. Timing and Synchronization

Timing and synchronization issues arise in several different forms when implementing interactive TSM controls under the standard 3G telephony mode. In this section, we discuss the general timing issues that arise with some of variable bit-rate (VBR) compressed formats and highlight how this complicates the addition of TSM controls to a 3G-telephony-compatible video server in which video and audio packets are often handled separately (through separate filter graphs, for example). Finally, we describe alternative approaches to solving these difficulties and examine the effect each approach has on the customer experience.

A. Seek points, Sample Time, Transmission Time, and Extra Time

Constant bit-rate (CBR) codecs, such as G.711, G.726, and G.729a, provide an unchanging mapping between packets and their *presentation time*, that is the time at which the data in the packet is presented to the customer. Due to this constant bandwidth usage on CBR codecs, they do not distinguish between the presentation time and the *transmission time*, i.e., the time at which the data packet is released from the server for transmission to the client. In contrast, variable bitrate (VBR) compressed audio and video, such as AMR audio and MPEG4 video, have irregular mappings between packets and presentation times. Furthermore, video VBR codecs can



(a) Decoupled TSM (b) Delayed-event TSM (c) Delayed-synchronization TSM

Fig. 1. **Server-side interactive speed up of playback for 3G handsets.**

have large spikes in the per-frame bandwidth requirements, especially on encodings with *seek points*. Video seek points are those frames within the video stream that are independently encoded such as I-frames. Due to the lower coding efficiency of I-frames, a pure I-frame can require as much as 3–10 RTP packets to transmit, even when encoded at the moderate bit-rates typically used in 3G telephony video channels (64-384 kbps).

Since 3G clients and gateways could drop packets during bursty transmission, the video server should smooth these bandwidth spikes by spreading out their transmission and sending some of the video data before its presentation time. This is possible with file-based content (like video mail or video snippets), since all the data packets are available to be sent from the server to the client at any time that the server chooses. Hint tracks in 3G files (when included) provide an *extra time* value for each packet, suggesting how much before the presentation time that the packet should be sent. However, this may cause problems when we apply TSM as will be discussed next.

B. Interactive Playback Speed up and Slow down

We argue that the customer should be allowed interactive control while using the unmodified 3G-handset SIP client, with this interactivity being provided through touch tones (DTMF) or voice commands. The generic handset does not predict or receive the state-dependent interpretation of these in-band signals. Thus, it has no reason to flush its local video or audio buffers. Any audio or video packets that are already at the client will be played sequentially. This continuous playback exposes the different lead times built into the transmission timing. A client-generated DTMF requesting playback rate change "now" is a poorly defined event. We discuss the problem using three alternative implementations: decoupled time-scale modification (TSM), delayed-event TSM, and delayed-synchronization TSM.

In decoupled TSM, the playback rate is changed at the server immediately within the audio and video filter graphs, without communication between them. In Figure 1, for example, the client requests playback speed up from normal speed (1.0 speed) to a 50% faster (1.5 speed) at 1000 ms. At a schedule time of 1000 ms on the server, the video filter graph could have already transmitted data with a presentation time of 2000 ms, while audio filter graph would be transmitting data with a presentation time of 1100 ms. Speeding up the audio by a factor of 1.5 will change what was 900 ms (= 2000-1100) of audio data into only 600 ms (= 900/1.5) of data. If this reduction in audio data is not corrected, the audio and video streams will desynchronize for the *remainder of playback* by 300 ms (=900-600). Visually, the problem is shown in Figure 1-a. The presentation times for the input (from-file) media tracks and the output (to-client) media tracks change to different rates as soon as the speed up request arrives. Due to the differences in the transmission-decode mappings, this event time corresponds to distinct presentation times on the audio and video tracks. As shown in Figure 1-a, since the mappings from the input-to-output presentation times changed slopes at two different input presentation times, the after-speed up display of audio and video are always desynchronized.

There are two different approaches to avoiding this desynchronization: delayed events (Figure 1-b) and delayed synchronization (Figure 1-c). With delayed events, whenever we receive a request for a playback rate change, we immediately apply that change to the video channel and note the presentation time at which that change was made in the video. Let us say that the video presentation time noted in this case is 2000 ms. We then create a delayed rate-change event for audio, for the equivalent presentation time in the audio stream. Thus, the audio modification was delayed: It is not applied immediately when the user requests the change. At the client, the display remains correctly synchronized throughout the rate change process but there is a delay before the rate change is seen or heard. Continuing with the example above, this delay would be 900 ms. This approach has the advantage of maintaining the synchronization throughout, but the disadvantage of seeming sluggish in responding.

With delayed synchronization (Figure 1-c), we apply the playback rate modification immediately to both audio and video channels, noting the presentation time on each. As noted above, unless corrected further, this results in the audio being played back at the client at an earlier time than the corresponding video. To avoid this long-term desynchronization, the audio filter graph inserts the correct amount of silence in its output stream to resynchronize. Again using the example

above, the audio filter graph would insert 300 ms worth of silence. At the client, immediately after the request for speed up, the audio would go silent for 300 ms (corresponding to the inserted silence), giving an initial desynchronization where the audio was played late relative to the video by 300 ms. Then, over the course of the next 600 ms, the audio and video would gradually resynchronize, as the audio played back faster than real time while the video continued to play at real time. After 900 ms, synchronization would be restored and the video would begin to play faster than real time. When slowing down playback, the "silence" is inserted into the video channel. That is, on receiving a request to slow down, the video timestamps are immediately offset by the amount of desynchronization that would have otherwise resulted. In all cases (speed up, slow down, seek), this approach temporarily desynchronizes the audio and video playback but corrects for that desynchronization as early as the data transmission allows. The advantage is that the user interface is quick and responsive. In our informal tests, the responsiveness of the interface is much more important than the temporary desynchronization.

The only other issue that must be addressed in playback speed modifications is how the server handles hinted files when streaming them at a changed speed. This becomes an issue since hinted files include an explicit suggestion for the extra time. When we speed up the playback, our presentation times change, so that they are closer together. If we use the unmodified extra times with the sped-up presentation times to calculate our transmission time, we will often result in "cross over": that is, the transmission time will not be a monotonic increasing function of packet number. Whenever the delta transmission time is negative, the change in extra time from one packet to the next might be large enough that the second packet nominally should be sent first. To avoid this reordering or clumping of packets, we scale the extra time by the same sped-up multiplier as we use on the presentation times for rates that are faster than real time.

In contrast with sped-up playback, we do not change the extra time when there is slower-than-real-time playback. Slowing the playback down spreads the packets out more widely than normal, so there is no issue of crossing over as there is with speed up. We could increase the extra time to further smooth the transmission rate. But there is no need, since the transmission rates on the slowed playback will always be below those that have already been within contract for regular playback. Furthermore, there is a disadvantage to expanding the extra time, i.e., a longer period of desynchronization on any subsequent playback changes due to this artificially expanded extra time. Since there is no requirement for the expansion and there is a disadvantage to the expansion, we instead use non-uniform scaling of the extra time, compressing it for faster-than-real-time playback and using it unchanged for slower-than-real-time playback.

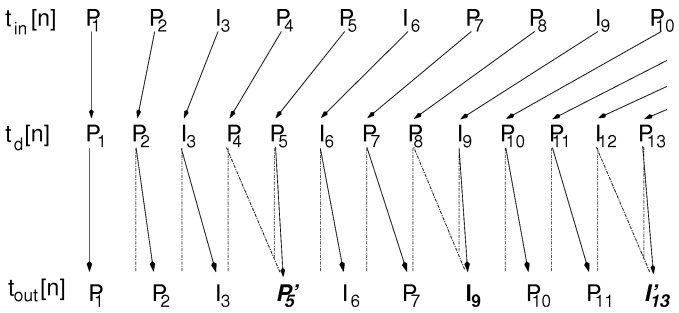


Fig. 2. **Dynamic trans-framing.**

III. Dynamic Sampling Modification

Existing TSM filters such as SOLA [6] provides audio playback speed control by estimating local periodicity in the audio and removes or inserts integer number of that period, with cross-fading to avoid audible clicks.

The sampling modification for video is more complicated since the use of the telephony side of the 3G network imposes strong constraints on the channel: we must remain at or below our contracted frame and bit rates for the session.

Slower-than-real-time playback is not an issue, assuming the content sampling rate conforms with the bit- and frame-rate contracts of the 3G channel. The contracts for frame and bit rates apply to the display frame and bit rates, which, for slower-than-real-time playback, are lower than the content sample and bit rates (due to the slow down). Similarly, seeking also is not an issue: We allow additional extra time to be inserted into the video transmission by "freezing" the video if needed and thus avoid over-budget spikes during the seek.

In contrast, faster-than-real-time playback will violate frame-rate and bit-rate contracts, unless we explicitly decrease the display frame rate *(trans-framing)* and bit rate *(trans-rating)*. We must reduce the content sample and bit rates to avoid having the sped-up display frame and bit rates exceed our contract. The approach taken to trans-framing and trans-rating must be both dynamic and computationally efficient. It must be dynamic since the target speed up rates are being controlled in an interactive manner by the customer. This means that we cannot simply transcode offline, but must provide on-line adaptation. It must be computationally efficient to maintain the capacity of the service provider's machines to provide economical services.

We propose a two-stage approach to this problem of transcoding downward during faster-than-real-time portions: first trans-framing, followed by trans-rating.

A. Dynamic trans-framing

We have taken the approach of allowing short durations of desynchronization between the audio and video, in order to allow more computationally efficient trans-framing. For a contracted frame rate of r_c, our approach introduces up to $1/r_c$ seconds of desynchronization in the client display . This degree of desynchronization is visible, but tends to be acceptable since it is not a sustained desynchronization.

Our approach to trans-framing is to decode all of the video frames as they are received from the file reader. We do this full decode since the video streams are typically encoded with very long GOP sizes (24-75 frames). By decoding as we receive the frames, we are able to smooth out that portion of our computational load across time and minimize the added delay that on-demand decoding would otherwise induce.

Using our algorithm, we determine when we need to drop a video frame in order to provide the sped-up playback without violating our contracted frame rate. When a "drop frame" event is triggered, we at most re-encode one frame *after* the dropped frame. Since we re-encode this frame, all following frames that are differentially encoded can be used *drift-free* without re-encoding. Thus, we re-use the original encoding of most of the frames, thereby minimizing both our computational load and our "second-generation" visual-quality degradation. The re-encode process can take on a more efficient approach such as [7]. It uses motion tracking across frames so that the motion information of the frame after the dropped frame can be estimated without another full and costly motion estimation process.

The computational saving of being able to do isolated re-encoding depends on s, the multiplier of the playback rate. At playback rates that are faster than real time, up to twice real-time ($1 < s \leq 2$), $(s-1) \times r_c$ out of r_c input frames are re-encoded, corresponding to $(s-1) \times r_c$ out of $(2-s) \times r_c$ output frames. For example, when the customer is running the video at 50% faster than real time ($s = 1.5$), then 1 out of 3 input frames are re-encoded, corresponding to 1 out of 2 output frames.

Selection of the actual frames that are removed is complicated due to two factors. First, the requested speed-up factor s is known, but can be changed dynamically. Moreover, the content sampling rate can be non-uniform in both MP4 and H.263 video. The actual content sample spacing is only known when the frame data is received. In contrast, the maximum allowed frame rate is part of the service contract, hence constant and known. Figure 2 illustrates our approach to deciding when to drop frames. When we get new frame data, we compute the desired presentation time, $t_d[n]$, using the input presentation-time offset from the previous frame and the desired presentation-time of the previous frame. We then compute the earliest allowed presentation time for the output frame:

$$\begin{aligned} t_d[n] &= t_d[n-1] + \Delta t_d[n] \\ \Delta t_d[n] &= (t_{in}[n] - t_{in}[n-1])/s \\ t_{out}[n] &= t_{out}[n-1] + \max\{\Delta t_d[n], \frac{1}{r_c}\} \end{aligned}$$

where $t_{in}[n]$ is the original (input) presentation time of the n^{th} sample and $t_{out}[n]$ is the final output presentation time of the n^{th} frame.

Whenever $t_{out}[n]$ is greater than or equal to $t_d[n+1]$, the n^{th} frame is dropped by re-encoding the $(n+1)^{th}$ frame. In Figure 2, this results in frames P_4, P_8, and I_{12} being dropped. The frames following the dropped frames are re-encoded, to removed dependencies on the dropped frame. For example, the re-encoded frame $\mathbf{P'_5}$ uses I_3 as its reference.

We modify the above approach slightly to increase our computational efficiency while maintaining all of the original-file seek points. If the frame following the dropped frame is an I frame (e.g., I_9), we do not re-encode, since this frame does not depend on the dropped frame (e.g., P_8 in Figure 2). If the dropped frame is an I-frame, we re-encode the following frame as an I-frame. In Figure 2, this is shown when I_{12} is dropped and P_{13} is then re-encoded as $\mathbf{I'_{13}}$. This choice on re-encoding again reduces our computational load, since I-frame encoding takes less computation, by not requiring motion-estimation searches. This I-frame encoding keeps the seek point in the middle of what would otherwise become a doubly-long GOP.

The above description seems to introduce a frame of delay into the transmission of the video frames, since the previous frame must be held until we get the next frame, to see if $t_d[n+1]$ is less than $t_{out}[n]$. This is not the case: there is no additional output-observable delay due to this dependency. The video packets are being released from the file reader according to the (possibly hinted) timing indicated by the *desired* frame placement. Once $t_{out}[n]$ is known, we can simply hold that frame for another $t_{out}[n] - t_d[n]$ seconds. This offset is the same as the offset that we are adding to the desired presentation time, so the release timing is correctly adjusted from the desired. If a second frame arrives during the delay period, we know we should re-encode. If it does not arrive during that period, we do not need to.

The only complication to this drop-timing rule occurs for multi-packet frame encodings. In that case, we might delay the first packet of a frame by $t_{out}[n] - t_d[n]$ seconds and receive one or more additional packets, all corresponding to the same (n^{th}) frame. In this case, assuming we do not see any packets from the $(n+1)^{th}$ frame during that delay, we go ahead and release each of the packets for the n^{th} frame, as their release time arrives. After releasing the first packet of the frame, we do not allow removal of that frame. If we find on a later packet that we should have dropped the n^{th} frame, we ignore this evaluation, since we have already partially released the frame. Typically, this will result in the $(n+1)^{th}$ frame being dropped (due to $t_{out}[n+1] > t_d[n+2]$) and a slightly larger desynchronization between the audio and video during the $(n+1)^{th}$ frame period. We have not seen a case where this extra de-synchronization is overly objectionable.

Computationally, this approach to trans-framing requires a full decode plus re-encoding of (at most) $r_c \times \min\{1, (s-1)\}$ frames per second.

B. Dynamic trans-rating

Once we have reduced the display frame rate to meet the contracted level, we may still need to adjust the bit rate to the contracted level. On sped-up playback, the frame-rate reduction will help to reduce the bit rate, however, if the original content was encoded using the full contracted bandwidth, it typically will not reduce it far enough to conform with the contracted rate.

The reason for the high bit rate is the increased frequency of I-frames within the sped-up stream. If the original stream, at the contracted frame rate, had an I-frame once every N_{GOP} frames, then the output stream will average an I-frame once every N_{GOP}/s frames, due to our approach of maintaining seek points. Intra-coded I-frames require more bits to encode than inter-coded P-frames, when encoding to the same perceptual quality. The ratio of the bits required for an I-frame and for a P-frame is g, the inter-frame coding-efficiency gain and can vary widely depending on the content.If we start from an original video stream encoded at the contracted maximum b_c bits and r_c frames per second, with a P-frame coding efficiency of g, then it will (on average) use b_p bits per P-frame and $g \times b_p$ bits per I-frame, where $b_p = b_c N_{GOP}/(r_c(g - 1 + N_{GOP}))$ In the sped-up frame-rate-adapted stream, each frame uses the same number of bits but, due to the more frequent I-frames, this uses up a total bit rate $b_c(s(g-1)+N_{GOP})/(g-1+N_{GOP})$.

We can use this formula, along with a running estimate of the inter-frame coding gain (g) and the I-frame separation (N_{GOP}) for this sequence, to estimate how much above the contracted bit rate the trans-framed sequence will be for any requested speed-up factor. These two estimates ($\hat{g}$ and $\hat{N}_{GOP}$) are easily estimated using a single-pole IIR filter and the observed values from the sequence. We start from initial estimates of $\hat{g} = 3$ and $\hat{N}_{GOP} = 3r_c$ (for 3 second long GOPs) and update the estimates each time we receive a new I-frame. To ensure a reasonable estimate, we limit the filter values so that $1 \leq \hat{g} \leq 8$ and $r_c \leq \hat{N}_{GOP} \leq 5r_c$.

With these estimates, we reduce the bit rate by a factor of $b_{over} = (s-1)(\hat{g}-1)/(s(\hat{g}-1)+\hat{N}_{GOP})$. On our test material, during twice real-time playback, b_{over} ranges from 4% to 20%, with an average of 10%. Given this estimated target bit rate, we can use solutions such as [8] to carry out an efficient trans-rating in real time.

IV. Experimental Results

In this section, we describe some example services that have been implemented to demonstrate and validate our approach to providing video services within the 3G telephony network. Since the audio/video synchronization is the key issue invovled in TSM, the result is best shown through the video demonstration [9] (MPEG-2 player required for viewing). In the demonstration, we show streaming to the video phones that are 3G-standard compliant. These video phones use SIP for session initiation/negotiation and RTP/UDP for streaming transport. They encode and decode G.711 μ-law audio and H.263 video.

The video phone first places a call to our video-enabled SIP server. The server uses the information embedded in the client-request URI to select the application logic for the client session, in this case the Video-Messaging service. The demonstration clip shows the coordinated use of audible and visual prompts, using the video channel to show the customer a menu of DTMF-to-function mappings. This menu is shown at the same time as the mappings are read aloud, so that audio-only customers can also be served. Next, the client selects to view the first available message by transmitting the appropriate DTMF.

The server can stream MPEG-4 simple profile or H.263 baseline video and AMR or G.711 μ-law audio to the terminal. For the demonstration, we stored 3GPP compliant files that contain time multiplexed H.263 video and AMR audio. Since the EyeBeam softphone does not support AMR (as indicated during call initiation), the server provides real-time trans-formatting for the audio.

The customer then skips forward to the desired location within the multimedia clip by sending DTMFs. We also show the ability to skip backward. Next, we demonstrate the ability to speed-up the clip while maintaining audio and video synchronization. The need for SOLA is shown by disabling this processing temporarily. Finally, we resume normal playback speed, and then jump to the next Video-Message.

All playback control is implemented on the server back-end with no modification being made to the generic SIP clients. This demonstration clip proves a working implementation of the algorithms discussed in the previous section.

V. Conclusion

We have proposed an approach to providing TSM controls within a SIP video server in a way that maintains synchronized playback, no matter what type of skew has been introduced between the streaming of the audio and video tracks (such as is commonly done for bandwidth smoothing). We have also proposed an approach to remaining within our frame- and bit-rate contract, even during accelerated playback, by dynamic trans-framing and trans-rating at the server. All of this work is to support interactive control of 3G video services using the telephony-oriented control stack. By doing so, we inherit the bandwidth and latency guarantees provided for telephony. Most importantly, we provide seamless intermixing of telephony calls and video services.

References

[1] 3GPP, "3GPP TS 26.234 Transparent End-to-End Packet Switched Streaming Service (PSS): Protocols and Codecs." http://www.3gpp.org/ftp/Specs/html-info/26234.htm, 2005.

[2] Collab, http://www.collab.pt/3100.htm.

[3] http://www.verizon.com

[4] http://www.sprintnextel.com

[5] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol." RFC 3261, http://www.ietf.org/rfc/rfc3261.txt.

[6] S. Roucous and A. Wilgus, "High quality time-scale modification for speech," in *ICASSP*, 1985.

[7] J. Youn, M-T Sun and C-W Lin, "Motion vector refinement for high-performance transcoding," IEEE Trans. On Multimedia, vol. 1, no. 1, pp. 30-40, Mar. 1999.

[8] P. A. Assuncao and M. Ghanbari, "A frequency-domain video transcoder for dynamic bit-rate reduction of MPEG-2 bit streams," *IEEE Trans. On Circuits and Systems for Video Technology*, vol. 8, pp. 953–967, Dec 1998.

[9] Video Demonstration, "Interactive 3G Video Messaging," http://www.hpl.hp.com/personal/Bo Shen/3GVideoMessaging.mpg.