Real-time Streaming Video Error Correction:
A Comparison of QVidium ARQ vs. Conventional ARQ and ProMPEG FEC

Overview

This paper begins with a basic primer on the primary video error correction mechanisms in use today for live, real-time streaming video, with the primary goal of explaining their relative advantages and drawbacks. We begin by describing **ProMPEG Forward Error Correction** (FEC), then general **Automatic Retransmission reQuest** (ARQ) implementations, and concluding with a description of **QVidium's patented ARQ** video error correction. Throughout the discussion, we emphasize how these various video error correction mechanisms affect the 3 primary concerns in a video delivery system of: ***latency***, the end-to-end delay, that the error correction mechanism adds to the video stream; ***robustness***, the ability to seamlessly recover any lost packets and minimize the probability of unrecovered packet loss and the resulting impairments on the quality of the recovered video feed; and, lastly, ***overhead***, the amount of extra bandwidth required for the error correction mechanism to be able to recover lost packets.

Forward Error Correction for Real-Time Video Streams

Forward Error Correction (FEC) is a method for error recovery that relies on the insertion of additional checksum packets to a video stream, calculated from blocks of video data packets. In contrast with other error correction and recovery mechanisms, such as Automatic Repeat reQuest (ARQ), FEC does not require feedback or an upstream channel, and can immediately recover lost packets without having to wait for feedback from the source. This is convenient for long latency links and communications connections without an upstream channel, such as certain satellite links.

Various forms of FEC have long been applied to digital audio-video streams, most notably for satellite transmission and more recently for Internet streaming, to help minimize the adverse impact of channel impairments on the audio-video signal. Advantages of FEC over other error correction mechanisms include scalability to large systems and its inherent multicast compatibility.

FEC augments a media stream with redundant data, called checksum packets, to help restore stream integrity based upon anticipated levels of packet loss. FEC groups data packets into an FEC block and generates checksum packets for each FEC block. A complete FEC block includes al the data packets for that block plus all the checksum packets for that block. The checksum packets generated from a given block are said to *cover* that block since missing data packets can be restored by combining the remaining checksum and data packets in that block. FEC *coverage* is the number of missing data packets that FEC can recover within a block. FEC algorithms generally limit the number of recoverable data packets in an FEC block to the number of checksum packets within that block.

FEC algorithms generally determine the number of checksum packets with which to augment each data block as a fixed percentage of the number of data packets in a block. Because packet loss is limited to the number of checksum packets, the ability for FEC to recover lost packets is therefore directly proportional to the additional bandwidth overhead allotted for FEC.

ProMPEG FEC (SMPTE 2022)

A relatively simple and well-known implementation of FEC for packet-switched transport is Parity-FEC, based upon the Internet Engineering Task Forces' (IETF) RFC 2733 [1]. The Pro-Mpeg Forum, a video transport standards body, adopted FEC implementations based upon RFC 2733 for the transport of video streams over IP networks. This has been adopted as the SMPTE 2022 standard for video forward error correction. SMPTE 2022 FEC computes the Exclusive-OR operation across corresponding bits of all data packets within and FEC block to create a single checksum packet for that block, called a parity packet. The complete FEC block has the property that the Exclusive-OR across all packets of that block, data and parity, yields a packet with all zeros in all bit locations. Any single bit error in any packet of the block would show up as a "one" bit when computing the Exclusive-OR. More importantly, if a receiver lost only a single packet within an FEC block, taking the Exclusive-OR of all received packets along with the parity packet would reproduce the lost packet. Through

this technique, a system implementing Parity-FEC can recover any single lost packet within an FEC block.

In order to protect against large contiguous losses of packets, called a burst-drop, SMPTE 2022 includes a packet interleaving mechanism for grouping FEC blocks. In this implementation of interleaving, a transmitter sequentially fills each row of a two-dimensional matrix with copies of the outgoing data packets. When a video data packet fills the last data row of the matrix, the FEC engine computes a final checksum row, generating one parity packet, by computing the Exclusive-OR of packet data down each column, to fill the checksum row, and then sends the entire checksum row as a burst of parity packets.

Optionally, SMPTE 2022 also allows the computation and transmission of a column of checksum packets by computing the Exclusive-OR for packet data across each row. Thus, one can select *Column-Only* or *Row & Column* FEC. Of course, Row & Column FEC consumes twice the additional bandwidth overhead compared to Column-Only FEC.

The bandwidth overhead consumed by FEC is generally 10% to 20% or more of the raw Audio/Video IP stream. Even then, the chances for correcting packet losses are small, and highly dependent upon the loss pattern. For example, if the data network loses more than one packet from a column, after processing any Row checksum packets, then all the data loss in that column would be un-recoverable and the system would incur a glitch in the video output stream. If, for example, the MPEG Transport stream was 2 Mbps, then after adding FEC, the network loading would be 2.2 to 2.4 Mbps, not including additional overhead for RTP headers. Furthermore, all this overhead would not even guarantee that all the packet loss would be recoverable. In practice, such a system would still sustain significant interruptions.

At the receiver, interleaving introduces a processing delay equal to the time required to fill the entire receiver's matrix. The receiver waits for the last data and parity packet within a block to arrive before it applies the received parity packets to the received block of data packets to recover any missing packets. If the last packet in a block was lost, then either a timeout, the appearance of a packet from a following block, or a combination of

both may force FEC immediate processing for the current FEC block. Interleaving introduces processing jitter both at the transmitter and at the receiver as a result of the periodic processing time in waiting to fill the interleave matrix.

For example, a typical ProMPEG FEC system would be configured with a matrix of 10 rows and 10 columns, 100 packets total. Assuming a stream data rate of 2 Mbps and an MPEG Transport Stream (TS) cell size of 188-bytes, filling an IP payload of 7 MPEG TS cells per IP packet (1316 bytes per IP packet), then FEC would introduce an additional delay of 526.4 milliseconds just to fill one matrix. Sometimes, ProMPEG FEC implementations use double-buffering, resulting of an additional delay caused by FEC of over 1 second.

Automatic Retransmission Request Error Correction (ARQ)

To improve the effectiveness of FEC, some video transport systems add feedback to allow the error correction system to dynamically respond to packet loss in the hopes of creating a more reliable system that can, at least theoretically, recover all packet loss. These systems generally rely upon positive acknowledgements for each transmitted packet. If the transmitter does not receive an acknowledgement within a predetermined timeout period, then the sender automatically retransmits the missing packet. The receiver can then count on eventually receiving all packets in a data stream if it waits long enough. The system can also be designed to give up after a certain number of failed timeout periods.

Engineers have long valued the effectiveness of feedback for achieving these design goals. The earliest implementations of the data wide-area network that we now know today as the Internet, for example, incorporated TCP/IP network protocol to create a robust transmission medium for data transmission over lossy and unreliable networks. TCP groups IP packets into blocks of data, and requires a positive acknowledgement from the receiver of the receipt of each entire packet block. At some point, if after the transmission of a certain number of data blocks the acknowledgement has not been received, then all further transmission will pause for as much as 60 seconds, or more, depending upon implementation, before giving up and resuming transmission. These

large, unpredictable pauses in the flow of a received data stream are acceptable for general communications systems, such as e-mail and web browsing, but are unacceptable for a video broadcaster who cannot tolerate even small glitches, especially when it might mean missing a critical touch-down during a football game.

To make video systems more reliable, and to attempt to adapt ARQ or feedback-based error correction for video transport, some systems may implement a type of ARQ with a short, fixed timeout and limit the number of timeouts to a small number. They may add this to an FEC implementation, so help improve upon the effectiveness of the FEC.

However, to our knowledge, these other systems do not attempt to adjust the timeout interval to match the network delays. This can result in either a very slow system with large latencies, or ineffective error correction. Furthermore, the requirement to always wait for a positive acknowledgement can significantly slow the maximum packet data rate, and hence the transmitted video bitrate, as the total round-trip delay of the network increases. Thus, such systems become completely impractical for satellite systems, where just the one-way delay is usually about 800ms. Also, an ARQ system that sends an acknowledgement for each packet will require and consume a large return bandwidth overhead, especially for high-definition (HD) video streams. Adding such an ARQ mechanism to FEC adds in the latency and overhead required by FEC to that of the ARQ.

QVidium ARQ – a New Approach

With these limitations in mind, QVidium took a completely different approach to custom tailor a video error correction system specifically designed for real-time audio/video feeds over IP networks, where latency must be minimized, while preserving the full integrity of the video stream and creating a robust system that has the potential to eliminate all packet loss. We started with the realization that the nature of video as a real-time stream comes with hard deadlines. Error correction only makes sense for that part of the video stream that has not yet been rendered, but is completely fruitless after the video segment has played out, We took the best of ARQ, its feedback mechanism, but made it orders of magnitude more efficient by changing from a positive acknowledgement system to a *negative* acknowledgement system, so that retransmission request feedback only

takes place when packets are lost. Technically speaking, this system is no longer an "automatic" retransmission request, in that the sender will no longer automatically send out replacement packets based on a timeout. Instead, the sender will send out replacement packets when the receiver (automatically) detects that packets are missing from the stream it is receiving.

We added two other critical features to our system. A QVidium ARQ receiver incorporates a smart ARQ receive buffer, that can automatically insert received, retransmitted packs into their correctly ordered sequence within a buffered video stream. We synchronize this receive buffer to the timing of the sending device to keep the temporal length of this buffer constant, and we set the size of this buffer based on a set of measurements that the receiver makes of the round-trip time to the sender, in order to minimize the size of this buffer and thereby minimize the overall QVidium ARQ system latency.

QVidium ARQ handles packet re-order problems, such as that due to dynamic network re-routing for load balancing and due to packet size variations (where some routers will send short packets ahead of longer packets). It also handles network jitter, since it measures the variation in packet arrival times from a video stream and automatically creates a jitter buffer to absorb these variations. Of course, the main purpose of this buffer is to allow time for several retransmission request attempts to be successful, and to insert the recovered packets into their proper sequence before the video stream is rendered.

QVidium ARQ has the advantages relative to all the other error correction mechanisms previously described by simultaneously:

1) Minimizing network bandwidth overhead to only requests and replacements of lost packets,

2) Minimizing the size of jitter and receive buffers using values that the receiver actually measures independently for each video transmission,

3) Freeing the system from any restrictions on data rate and thereby allowing the system to function well even over very large network delays, such as through satellite links and feeds to other countries, and

4) Maximizing system robustness by dynamically adapting to packet losses.