Pursuit - The Journal of Undergraduate Research at the University of Tennessee

Volume 6 | Issue 1 Article 21

April 2015

Forward Error Correction for Fast Streaming with Open-Source Components

John Reynolds
University of Tennessee, Knoxville, jross26@vols.utk.edu

Pursuit: The Journal of Undergraduate
Research at the University of Tennessee
Copyright © The University of Tennessee
trace.tennessee.edu/pursuit

PURSUIT

Forward Error Correction for Fast Streaming with Open-Source Components

JOHN REYNOLDS
Advisor: Dr. James S. Plank

The mechanisms that provide streaming functionality are complex and far from perfect. Reliability in transmission depends upon the underlying protocols chosen for implementation. There are two main networking protocols for data transmission: TCP and UDP. TCP guarantees the arrival of data at the receiver, whereas UDP does not. Forward Error Correction is based on a technology called "erasure coding", and can be used to mitigate data loss experienced when using UDP. This paper describes in detail the development of a video streaming library making use of the UDP transport protocol in order to test and further explore network based Forward Error Correction erasure codes.

1 Introduction

Video streaming is a widespread technology used in many differing applications. It is accomplished via the sending of video frame data across networks in capsules known as *packets*. The channels transmitting these packets may have errors in transmission, which can cause some of the packets being transmitted to fail to arrive at the receiver. The failure of packets to arrive is defined as *packet loss*. Unreliable networks, implemented with wired or Wi-Fi connections, cause a large amount of packet loss in a multitude of different network applications.

The NACK (Negative Acknowledgement) protocol is used by many applications to tell the sender that the receiver has experienced packet loss and requires a re-transmission of data; however, with *FEC* (Forward Error Correction) [4] the need for the NACK protocol and re-transmission is greatly lessened. FEC is an efficient solution to the packet loss problem. The development of this video streaming library with FEC embedded provides a platform for the participation in research on network based FEC erasure codes.

Forward Error Correction works as follows: data source symbols being sent across a network may be encoded using erasure codes to create redundant data. The data symbols, in this case, are packets containing video frames, and will be referred to as source packets. The redundant data packets created may then be decoded on the receiving end to recover lost source packets (otherwise referred to as “dropped” packets). These redundant data packets are referred to as both coding packets and repair packets.

Many different erasure codes exist. Our embedded FEC was designed to allow for the testing of different erasure codes in the future. The FEC component was initially implemented using Reed-Solomon [12] based codes. Reed-Solomon codes belong to a category of codes known as MDS (Maximum Distance Separable). MDS codes have a nice property: k source symbols can be encoded to create m repair symbols such that among the $k + m$ total symbols, any m erasures are tolerable.

This video streaming library is built for real-time environments. The receiver is always rendering the last frame that was received. When video data packets are transmitted over lossy channels using UDP, the receiving user may not receive all of the packets that were sent. This is because of the UDP protocol specification, which does not guarantee the arrival of packets. It is a transport protocol used in data transmission that allows for the sending of packets to a destination. UDP differs from TCP in a few key ways: TCP forms a direct connection with the destination IP guaranteeing the ordered arrival of packets, and UDP sends packets to the destination IP without forming a direct connection or guaranteeing arrival. Packet loss experienced when using UDP can cause the skipping of video frames, which results in a bad streaming experience if not handled properly. To be specific, a user experiences drops in FPS that are directly correlated to packet loss. In the sender, *FPS* is defined as the number of frames sent per-second. In the receiver, *FPS* is defined as the number of frames rendered per second. The receiver application will suffer large drops in FPS when video frames are skipped. In our experiments, we measure the effect that erasure coding and packet loss have on the FPS of the sender and receiver. Our results show that erasure coding causes little FPS degradation at the sender, and that it greatly improves the FPS at the receiver when there is packet loss.

2 Methods

The flow of data through the video streaming library is handled by many components. Before delving into the library’s creation, two more terms must be introduced and explained: RTP and VP8. While UDP is the transport layer protocol of choice, other protocols may be applied at the application layer (where layers refer to the IP suite) [6]. Specifically in this application, the Real-time Transport Protocol (RTP) is applied at the application layer [2].

RTP is a protocol that defines a standardized packet header used for the real-time streaming of audio and/or video data. This protocol is implemented on top of UDP formatted packets carrying the video data. RTP provides extra information alongside the actual video frame data being transmitted. Two important keys to video frame reconstruction are located in the RTP packet header: a timestamp and a sequence number. These can be found in every RTP packet. The timestamp and sequence number can be used by the receiver of the video stream to accurately rebuild the original video.

VP8 is an open-sourced video format owned by Google. A free-to-use codec (encoder/decoder), *libvpx* [10], is available to the public. Since the format of a video frame can differ depending upon the camera used to capture it, all captured frame formats are converted to the same YUV color space (commonly used instead of RGB) format using Google's open-sourced YUV conversion and formatting library, *libyuv* [11]. Then, for the sending of data, the YUV frames are compressed using *libvpx*. This allows the library to maintain a standard packet payload layout throughout the streaming process. Each packet can be uniformly formatted, regardless of the camera capturing the frames. This is due to each UDP/RTP packet having a VP8 payload [3].

With an understanding of UDP, RTP and VP8, the video-streaming functionality is comprehensible. First, a library (*video-capture* [7]) is used to locate hardware devices connected to a user's computer that are capable of capturing video. Then, a socket connection is opened by the sending program in a thread separate from the program's main thread. The main thread is connected to the local hardware camera device, and captures video frames. The program's main thread performs the native format to YUV conversion and the YUV to VP8 compression. It then creates the UDP/RTP packets, placing the VP8 data into them as the payload. After placing the VP8 payload data into the UDP/RTP packet, the packet is passed to the second thread, which is setup in an event-loop style sending the UDP/RTP/VP8 packets to the receiver through a socket.

The receiver is multi-threaded as well, and the receiving thread is spawned from the receiving program's main thread. The receiver thread runs an event-loop awaiting the arrival of UDP/RTP/VP8 packets. When a packet is received, the packet is parsed for all information relevant to the video frame. The sequence number retrieved from the RTP header tells the receiver specifically where each packet's bytes belong in a single frame relative to the other frame data. The RTP timestamp is used to determine the time that the frame the packet belonged to was captured. If a set of UDP/RTP/VP8 packets have the same timestamp, they belong to the same video frame. The receiver can use this fact to decode a set of packets in order to retrieve the YUV frame from VP8 payload data.

Packet loss occurs in lossy networks between the sender and the receiver. When packet loss occurs, data relevant to a given video frame is lost. To be specific, when UDP/RTP/VP8 packets are lost in transmission between the sender and the receiver, the quality of video produced is affected directly. Another type of error can occur due to noise in transmission channels (bit flips). However, this paper deals specifically with the loss of entire packets, and refers to such phenomena as *erasures*. UDP/RTP/VP8 packet erasures prohibit the frames to which they belong from being successfully decoded from VP8 to the YUV format. Forward Error Correction (FEC) is used to lessen the effect of packet erasures.

FEC redundancy from the UDP/RTP/VP8 packets uses erasure codes in the sender program. The size of any given UDP/RTP/VP8 packet varies depending upon the underlying YUV frame data. However, the FEC codec requires the source symbols (packets) to be the same size. Thus, all packets are padded to the size of the largest packet for the given frame. FEC Reed-Solomon encoding is applied to the set of packets composing a single video frame, and every frame sent from the sender to the receiver is encoded. As mentioned earlier, Reed-Solomon codes belong to a collection of codes with a certain property known as MDS. This means that *k* source symbols can be encoded to create *k* + *m* coding symbols (the *k* source symbols remain unchanged). As long as the receiving program acquires *k* packets, regardless of whether they are source (*k*) or coding (*m*), the receiver is able to decode to restore lost packets caused by channel erasures.

When the main thread in the sending application creates UDP/RTP/VP8 packets from the YUV frame data, they are passed to the sender thread to be dispatched, and are simultaneously stored in a buffer in preparation for erasure coding. With FEC enabled, before sending or encoding, the FEC information is appended to the source packets. In order to maintain compatibility with receivers that do not have FEC enabled, the source packets are sent with FEC information appended. The FEC information is only appended when the source packets are put into the buffer for erasure coding. The Reed-Solomon erasure coding is only performed when the buffer contains all packets for a given frame. After encoding, the buffer is cleared and the sending program prepends the FEC data to the coding bytes that were created according to RFC 6865 [1]. The coding data with FEC information attached is sent directly over UDP without a RTP packet header. A simple diagram displaying the flow of the sender can be seen in *Figure 1* below.

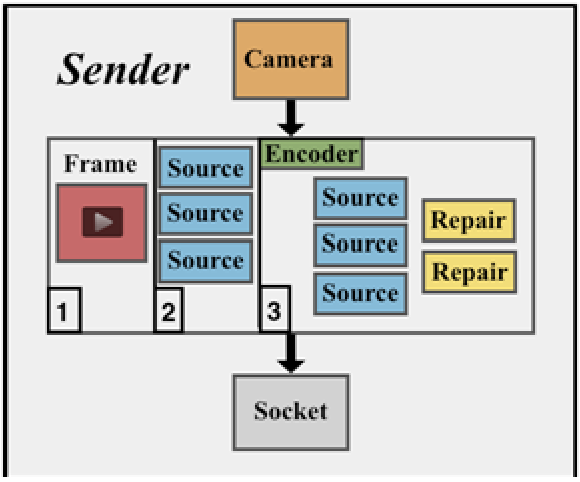

Figure 1: Sender program data flow diagram

Receivers with FEC enabled must handle both source and coding packets for a video stream. When the receiver receives a source packet, the procedure followed is similar to that of the receiver without FEC enabled. The receiver unwraps the source packet, determines the frame it belongs to, and stores it in a buffer waiting to be decoded from VP8 to YUV. The FEC information attached to both the source and coding packets can be used to detect packet loss. If no packet loss is detected, the VP8 data is simply merged back into a single block and the block is decoded. If packet loss occurs, the receiver must wait until *k* packets are received. If *k* packets are not received, the frame cannot be decoded. However, when *k* packets are received, the source packets and the coding packets received are decoded to reconstruct the lost source packets. After the lost packets are retrieved, the VP8 data is merged and decoded. With the MDS property, data recovery is possible for any combination of *m* packets lost as long as at least *k* packets are received. *Figure 2* is a simple diagram demonstrating this behavior.

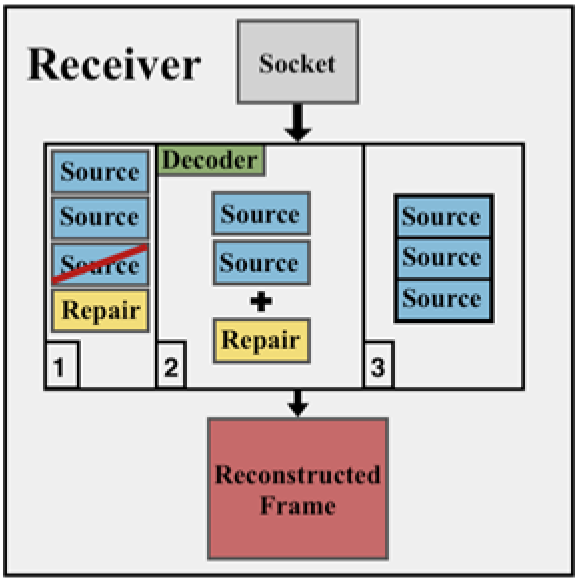

Figure 2: Receiver program data flow diagram

The OpenGL library is used for graphics at the receiver. The YUV frames that are decoded from VP8 are decoded in the second thread. When the YUV data has been retrieved, OpenGL context information is updated. The receiver program multithreads OpenGL using a "shared context" from an OpenGL wrapper, glfw3 [8], and the program's main thread renders the YUV frames to a window on the user's computer. The OpenGL frame buffers can be manipulated from thread two as YUV data becomes available and rendered in the main thread when thread two updates a frame buffer.

If frames known as *reference frames* are skipped, error propagation into successive frames will occur due to the inter-frame compression techniques used. Reference frames contain data relevant to the construction of future frames. In some prediction algorithms, the colors of blocks of pixels that are rarely changing color between frames are determined using the data in a reference frame. If the reference frame is lost, the following frames can no longer accurately predict a pixel color [9]. Due to this, it is suggested that reference frames be sent at higher intervals when dealing with lossy networks. Code can be written to automate this process using well-known protocols, but that is beyond the scope of this paper.

3 Results

The development of this video streaming library provides a practical networked application, making use of a real-world scenario, which presents the opportunity for further research on the topic of FEC. Tests were performed using the Reed-Solomon FEC codes in order to quantify successful library development. There are many metrics involved in generating measurements, and these metrics must be understood to fully grasp the impact erasure codes have on video streaming.

Code was written in the sender to count the number of frames sent and the rate at which they were sent. In the receiver, two measurements were taken. The number of frames received, and the rate at which they were rendered to the OpenGL frame buffer. Reed-Solomon codes allow a user to select the number of coding packets to generate from source packets. A limit is imposed due to the finite field arithmetic (Galois Fields) used in creating these, but that is both beyond the scope of this paper and irrelevant, because the number of coding packets tested being far under this threshold. With this information gathered, it is clear that FEC makes a significant improvement to the video streaming library when transmitting over lossy channels.

The tests were performed using two MacBook Pro computers streaming over a wide area network (WAN). Both computers were connected to their respective local networks via Wi-Fi routers. The sender was tested with a mid-2010 MacBook Pro with the built-in *iSight* camera. The receiver was running on a MacBook Pro 2013 model. iSight cameras are designed to fluctuate FPS to react to changing light values. If a room is dark, FPS will be lowered. If a room is bright, the camera increases FPS. For these tests, the room lighting was maintained at a level that kept the iSight's capture-able FPS to an average of 15. This number differs from the FPS sent from the sender to the receiver, because the frames captured from the built-in iSight must be broken up into chunks and placed in packets before sending.

Each test was performed by streaming video across the internet (WAN) for five minutes. The average number of source packets per frame is 4.6. The streaming was tested 5 times for every combination of number of coding packets (0, 1, 2) and percentage packet loss (0%, 10%, 20%). The numbers plotted are the results of those tests averaged together. They clearly demonstrate an increase in performance over the standard video streaming when using FEC erasure codes. The relationship between packet loss, FPS maintained, and the number of generated coding packets can be seen in *Figure 3* below.

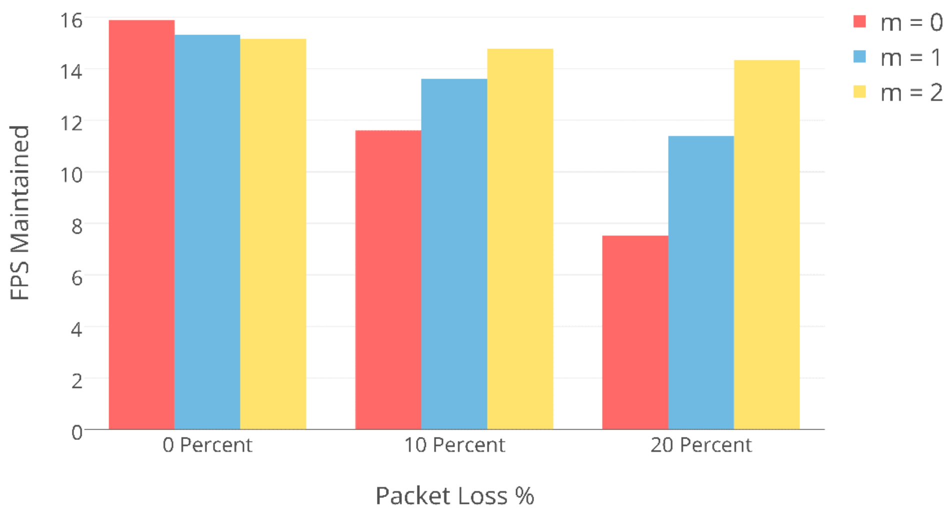

Figure 3: FPS Maintained w/ Packet Loss & Erasure Coding

The three different colors of bars represent the varying number of coding packets generated. The red (leftmost) bars represent video streaming with no FEC, the blue (middle) bars represent a single redundant packet created, and yellow (rightmost) represents two coding packets created. The x-axis represents the percentage packet loss experienced. Packet loss was induced in the sender with a program released by Apple named *Network Link Conditioner*. This program directly modifies networking hardware in the MacBook Pro to cause packet loss with a random probability distribution given a certain percentage. For example, a user can request a percentage packet loss to induce and the Network Link Conditioner will effectively generate random packet loss at that percent as packets are received. The Network Link Conditioner can be used to simulate real network packet loss, and was used for that purpose during the testing of the FEC embedded in the video streaming library.

Figure 4 demonstrates the correlation between packet loss with erasure coding and the number of frames skipped. The colors represent the same redundancy as shown in the previous graph. Over all tests, the average number of frames sent is 4563. The number of skipped frames is determined by two factors: failed frame decoding (with *libvpx* or *libyuv*), and too few packets received for reconstruction. As the packet loss percentage is increased, the number of frames skipped without FEC enabled is drastically

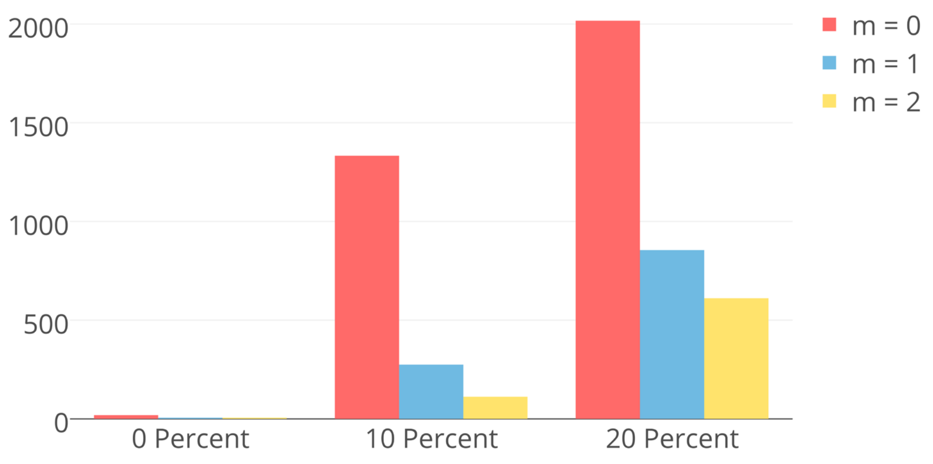

Figure 4: Frames Skipped vs. Packet Loss % + Erasure Coding

higher than that of frames skipped with FEC enabled.

4 Conclusion

Packet loss has negative side-effects. Real-time video streaming suffers frame skipping and lowered FPS because of packet loss. The effect that FEC has on video-streaming is substantial, and the development of this library will allow for the exploration of different FEC erasure codes. This is necessary for further research on codes such as *Raptor Codes* [5], which are built for varying networks such as those experienced in real situations. The library is built of entirely open-source components, and the code can be modified as needed to gain more information from the streams, set up FEC enabled streaming applications on devices such as tablets or smartphones for research, and become operating system independent. The future research this library allows is intriguing, and can lead to innovative optimizations.

5 References

Bankoski, J.,Wilkins, P., and Xu, Y. (2011) Technical overview of VP8, an open source video codec for the web. *IEEE International Conference on Multimedia and Expo (ICME)*:1-6.

Berglund, C. (2011).”GLFW license”. Retrieved 5 July 2013. http://www.glfw.org/license.html

Braden, R. (1989) Requirements for Internet Hosts - Application and Support. STD 3, RFC 1123. libyuv: (May 2012), libyuv. http://code.google.com/p/libyuv/

Plank, J. S. (1997)A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems. *Software – Practice & Experience*. 27(9): 995-1012.

Richardson, I.E. (2003) *H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia*. Chichester: John Wiley & Sons Ltd.

Rizzo, L. On the feasibility of software FEC. 1997.

Roca, V., Cunche, M., Lacan, J., Bouabdallah, A., Matsuzono K. (2013) Simple Reed-Solomon Forward Error Correction (FEC) Scheme for FECFRAME V (Status: PROPOSED STANDARD) (Stream: IETF, Area: tsv, WG: fecframe).

Roxlu (2014). Video Capture library: http://video-capture.readthedocs.org/en/latest/

Schulzrinne, H., Casner, S., Frederick, R.,Jacobson, V. (2003) RTP: A Transport Protocol for Real-Time Applications (Status: INTERNET STANDARD) (Stream: IETF, Area: rai, WG: avt).

Shokrollahi, A. (2006) Raptor codes. *Information Theory, IEEE Transactions on*. 52(6): 2551-2567.

Westin, P., Lundin, H., Glover, M., Uberti, J., and Galligan, F. (2014) RTP Payload Format for VP8 Video”, draft-ietf-payload-vp8-11 (Status: WORK IN PROGRESS) (Stream: EFTF).