Energy Savings for Video Streaming Using Fountain Coding

Anders Nilsson Plymoth
TelHoc AB
Stockholm, Sweden
Email: anders@telhoc.se

Zhi Zhang
Department Of Electrical and Information Technology
Lund University, Sweden
Email: zhi.zhang@eit.lth.se

***Abstract*—Reducing energy and power consumption is important for many reasons, including extending the lifetime of battery operated devices and reducing energy bills. An important source of energy loss in wireless networks comes from retransmissions of lost packets. Many of the transmission protocols in use today were designed mainly to be used in wired IP networks, and therefore performs bad in wireless scenarios. We present a protocol that more efficiently handles packet losses in wireless network with a special emphasis on video streams, and show that the use of fountain coding can reduce power consumption and save energy, especially in multicast settings. We show that fountain coding minimizes the number of retransmissions in lossy wireless networks. Our initial tests for video multicast scenarios with up to 10 users in networks with high loss rates show that power can be reduced by up to 200%.**

I. Introduction

With the increasing popularity of smart phones and other mobile devices, the use of mobile Internet is also quickly increasing. Video delivery forms a big portion of the traffic on mobile terminals and consumes much energy.

Reducing energy and power consumption is important for many reasons, including extending the lifetime of battery operated devices and reducing energy bills. An important source of energy loss in wireless networks comes from retransmissions of lost packets. Many of the transmission protocols in use today were designed mainly to be used in wired IP networks, and therefore performs bad in wireless scenarios. An area that currently experiences rapid growth is video streaming, and Cisco reports that video will amount to about 80% of all Internet traffic by 2019 [1]. Cisco further predicts that 67% of all IP traffic will be consumed by WiFi connected devices.

While it is generally well know that WiFi connections experiences packet loss, it is less known that these losses are typically much higher than most people realize. For unicast packets, WiFi employ up to four retransmissions and perform dynamic link rate adaptation to better cope with bad channel and signal propagation conditions. One study [2] found that during an important computer science conference as much as 28% of all transmissions failed. In residential areas, many wireless networks and devices coexist in a small frequency spectrum and experiences sustained noise and interference. However, retransmissions are very efficient and even with a loss rate as high as 50%, the resulting packet loss rate is only 6.25%.

Depending on the type of service, video streams can either be transported using TCP or UDP, but often rely on use the UDP protocol in order to reduce latency and for its support of multicast and broadcast streaming services. However, UDP multicast and broadcast packet are not retransmitted, and are therefore much more vulnerable to adverse wireless channel conditions.

A topic that is currently receiving a lot interest in the wireless research community is fountain codes [3], because their properties are particularly efficient in broadcast and multicast scenarios and for improving retransmission performance. Recent studies [4] also indicate that they may also save energy.

In this paper we introduce a protocol that is able to perform reliable UDP multicast streaming and which also supports the use different types of fountain coding. We study how fountain codes can save energy and improve power consumption over lossy wireless radio links, especially in video multicast and multi client scenarios, and for different number of users.

The remainder of this paper is organized as follows: section II describes related work and in section III a brief overview of fountain coding is given. Section IV describes our consumption reducing protocol, followed by a description of our setup in V and results in VI. Final conclusions is given in VII.

II. Related Work

While UDP may reduce latency and support multicast streaming, it does not have have any delivery guarantee for the correct order of packet reception, or the retransmission of lost packets. With UDP, the network layer is unable to recover from packet losses, which may lead to significant drop in video quality. Reliable User Datagram Protocol (RUDP) [5] is therefore a proposed protocol by Bell Labs to provide a solution where guaranteed-order packet delivery is desirable. Similar to TCP, RUDP implements features of acknowledgment of received packets, windowing and flow control, and retransmission of lost packets, but with less overhead. However, in lossy wireless networks, the high packet loss rate leads to an increase of retransmissions, and often multiple receptions of the same packets by the same device, which is a waste of energy.

Fountain codes have be proposed recently for video streaming [6] [7]. Fountain codes are rateless erasure codes in the sense that the encoder can create as many encoded symbols

as required on the fly. This is an advantage for wireless channels in which the channel conditions vary frequently or are unknown. Moreover, Fountain codes has low complexity both on the encoder and decoder sides compared with other Forward Error Correction (FEC) coding algorithms such as Reed-Solomon codes.

While much work has been performed on the study of fountain codes, including on how it can be used for video streaming [6], and improve the performance of video codecs [8], very little has been done in terms of assessing the impact it has on energy and power consumption. Our presented method also make its deployment much more straight forward, by instead of embedding the coding in the codec or network protocols, it can easily be dropped into applications. It can also be easily deployed at video servers, or at wireless access points.

III. Network and Fountain Coding

In network and fountain coding, intermediate nodes may send out packets that are linear combinations of a set of other packets. There are two main benefits of this approach: potential throughput improvements and a higher degree of robustness. Robustness translates into loss resilience and facilitates the design of simple distributed algorithms that improves performance, even if decisions are based only on partial information.

The main difference between network and fountain coding, is that in network coding any packets may be combined, while in fountain coding only packet from a particular source or stream are combined. In network coding [9], a router or a set if routers may identify that multiple paths are available through the network, and that by combining some packets the number of transmitted packets in the network can be reduced.

Fountain codes typically operate on a set of data such as a file, or a piece of a file, and randomly combines these pieces so that the order in which they are received becomes unimportant as long as the number matches at least the number of pieces in the source data. Due to its similarity to erasure coding [10], fountain codes are sometimes also referred to as rateless erasure codes. As a Forward Error Correcting (FEC) mechanism used for Automatic Repeat Request (ARQ) operations, it means that a sender and receiver do not have to consider the correct arrival of individual packets, which greatly simplifies retransmission operation and the needed amount of signaling.

For multicast or multiple receiver operations, especially in wireless scenarios such as in WiFi networks, it has even higher benefits. If individual receivers independently loose different packets in a stream, each of those individual packets needs to be retransmitted. In the case of fountain codes, only a single packets would be needed for all of the receivers, thus greatly reducing the number of needed transmissions and thereby the energy consumed.

In this study we consider two types of fountain coding, LT codes [11] and RaptorQ [12] codes.

Luby Transform (LT) codes are the first class of efficient practical Fountain codes. Potentially, a LT code can generate an unlimited amount of encoded data from the source, where the source data can be efficiently and completely recovered from reception of any combination of encoded data essentially equal in size to the source data.

The LT (Luby Transform) encoder produces packets from a block of source data as follows: Randomly choose the degree, d from a degree distribution, which depends of the size, K of source data. Here K represent the number of packets needed to represent the source data. The encoder then uniformly at random, chooses d out of these K packets and bitwise modulo 2 combines these packets to a new packet of equal size. In [11] the chosen distribution is a Soliton distribution that enables fast encoding and decoding operations by creating a mix of low and high degree packets. The decoder then reverses this operation by considering the operation as a linear equation system and solves this through Gaussian elimination.

However, LT codes do not have linear decoding properties, and in order to improve upon this, Raptor Codes have been proposed. Raptor codes uses a compound coding structure, which usually includes a high-rate outer LDPC [13] code and an inner LT code, which is able to nearly optimally minimize the needed number of packets for successful decoding. The difference is that in LT codes there is always a slight chance that a newly received packet is linear dependent in the decoder equation matrix, and thus provides no new information. The outer code greatly reduces this probability and thereby increases the effectiveness of the code.

A superior form of Raptor codes have been proposed, i.e., RaptorQ codes. RaptorQ is more efficient than Raptor coding in terms of flexibility and efficiency. It uses an enhanced two-step pre-coder and a superior LT encoding algorithm. Moreover, it supports a larger range of the size of source symbols and encoding symbols and can deliver huge chunks of data at a time.

RaptorQ is currently the most efficient known fountain code, although it is covered by heavy IPR protection.

IV. Protected Video Streaming

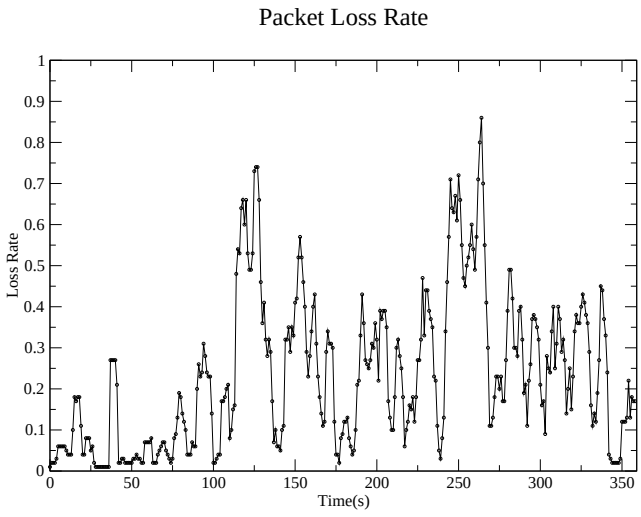

Fig. 1. Packet Loss Rate Over Time

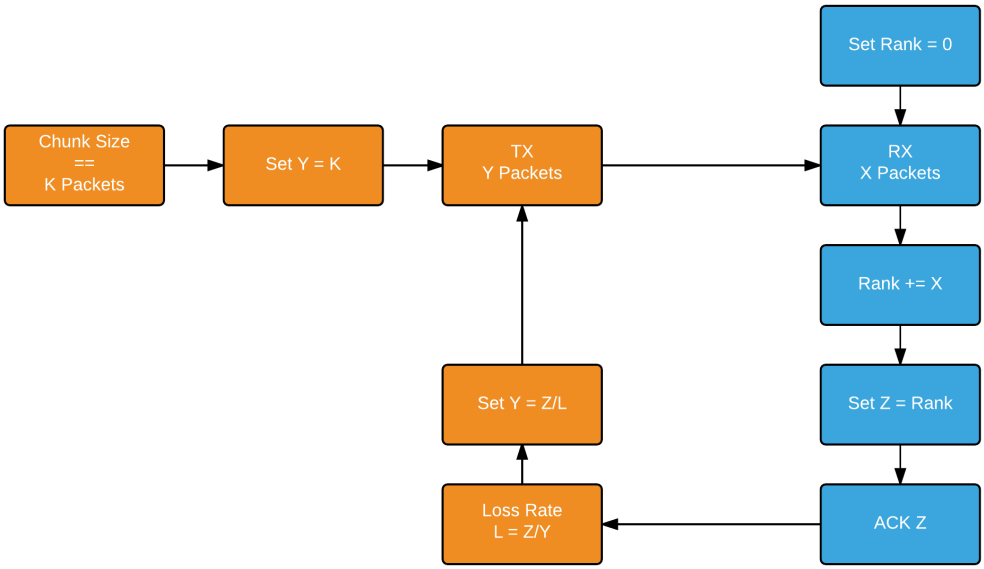

Fig. 2. Fountain Coding packet transmission over lossy channels

Video services delivered using UDP as the transport protocol are vulnerable to packet loss as no explicit acknowledgments or retransmissions are performed. While modern video codecs can accept some packet loss, this comes at the price of lower video quality. In [14] the authors studied how the quality of the video is impacted by packet loss, and found that the quality of the video degrades quickly as the packet loss ratio increases. By 10% it is so bad the video is almost unwatchable.

While it is reasonably well known that commercial and residential WiFi networks periodically experiences packet loss, it is less know how severe this loss actually is. This is because the WiFi protocol performs link adaptation and that this in combination with retransmissions is very efficient. The default is to perform 4 retransmissions, which means that even with a loss rate as high as 50%, the resulting perceived packet loss rate is only 6.25%. However, retransmissions are only performed for unicast packets, not for multicast or broadcast transmissions.

Figure 1 shows a 5 min measurement of the packet loss rate performed in a residential area in central Stockholm. The capacity fluctuates heavily due to variations in noise and interference, and the average packet loss was 23% with peaks up to 80% loss. This would have severe impact on video streams over UDP multicast or broadcast streams.

In order to protect video streams from these severe conditions while still supporting multicast and broadcast services, we developed a protected streaming protocol for UDP video packets, that operates similar to the Reliable UDP (RUDP) protocol [5]. This protocol transmits chunks of packets after which individual packets within the chunk are acknowledged using a bitfield in an acknowledgment packet. A server transmits these chunks upon receiving chunk requests from clients. When the server receives an acknowledgment, it transmits the packets indicated in the bitfield. See figure 3. This makes the protocol very flexible and ideal for supporting video streams from both a sender and receiver perspective. A receiver may choose to receive all the packets in the chunk by indicating this in the bitfield, or it may use available video encoding information to only request as much data as it needs for successful decoding. The receiver can then also choose to prioritize packets that include data of high importance such as keyframes within the video. The server on the hand, can also choose to aggregate acknowledgments from several receivers in a multicast scenario, and simultaneously support ensured reliable delivery to all of them.

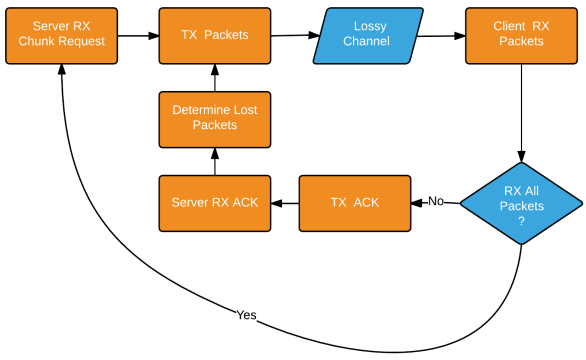

Fig. 3. Reliable UDP packet transmission over lossy channels

As mentioned above, fountain coding is an efficient method for transmitting packets over lossy wireless channels. Our

protocol therefore also support LT and RaptorQ coding of packets. This version of the protocol operates the same way as the RUDP version, but a fountain code is used for each transmitted packet. The symbols from the fountain code are generated from the requested chunk of data and inserted into the transmitted packets. Instead of using a bitfield in the acknowledgments, only the rank of the decoding matrix is included, which therefore decreases the size of the acknowledgment packets. The server therefore only needs to know how many packets that needs to be transmitted, not exactly which packets, and therefore determines the number of packets to transmit as difference between full rank and current rank. When the server receives acknowledgments from several clients for the same video stream, it uses the lowest rank among the clients.

In a basic fountain coding scenario a sender keeps transmitting symbols until a receiver decodes the message and signals the sender to stop. In a wireless network situation this may become inefficient because between the time the sender sends a packet, and a receiver decodes the message and sends the stop signaling packet, the sender may already have sent several unneeded packets. This leads to unnecessary packet transmissions and a waste of energy. The other option is that sender only sends a certain amount of packets equaling the rank of the message, and then waits for feedback acknowledgments from the receivers. The sender then sends another set of packets equaling the difference between full and current rank. The problem with that approach is that it becomes inefficient in lossy networks as some packets will be lost in each transmission phases, and therefore typically requires several cycles. In this protocol we instead estimate the packet loss rate of the channel by looking at the difference between the number of sent and requested packets. Using this loss rate, an extra number of packets can be transmitted equal to the expected number of lost packets, see figure 2.

V. Experimental Setup

The experimental setup consisted of a number of Nexus 5 (LG) mobile phones and a number of Raspberry Pi B+ devices [15] with 802.11abg WiFi cards. The Nexus 5 phones were running Android 6.1 and the Raspberry PI's were running either Raspbian Linux OS [16] or Android 5.1. The Raspberry PI running Raspbian were used as an WiFi access point and hosting a video file server that Android clients used to connect directly to.

This setup has several advantages, as video streaming is easily supported on Android devices and that the Linux OS allows controlling packet success rates. Packet loss rate were controlled using the internal firewall tool *iptables* where incoming and outgoing packets can be dropped according to some statistical distribution, i.e. a uniform distribution. As all the devices are located very close to each other in the same room, the *iptables* packet drops emulates wireless packets received but discarded due to noise and interference, the most common type of wireless packet loss.

It also allows power measurements to be more easily conducted, as Nexus 5 has both API and internal support for power measurements. Raspberry PI's are typically powered through a USB port, which allows easy power measurements by inserting a USB power meter between the device and the power source.

For all measurements, the short film "Tears of Steel" [17] were used, a freely available short movie often used and partly created for being used in video benchmarks. This video is streamed using standard video chunk procedures.

VI. Results

The first experiment considers the case of a single client consuming a single video stream from the server.

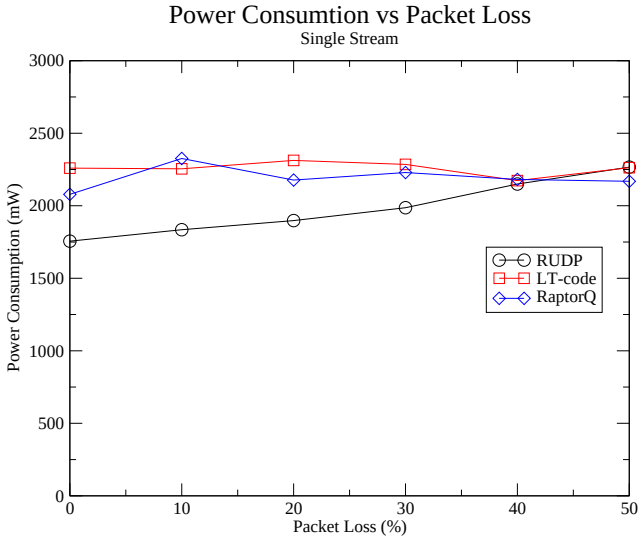

Fig. 4. Power consumption vs different packet loss rates for a single client receiver.

As can be seen from figure 4, the power consumption of LT and RaptorQ coding not change very much as the packet loss rate increases, while for RUDP it increases almost linearly. That the power consumption should increase should be an obvious conclusion, because as more packets are lost there is a greater need for packet transmissions which therefore increases power consumption. This should also apply to the coding cases, but here you must also take into account the power consumption of the decoding process. The decoding process consumes power, which is why the fountain codes consume more power for the lower loss rates. However, as packets starts dropping, packets will arrive more sparsely allowing the CPU to spread out its work a bit more, which in turn also cools it down a little bit. That is, the energy consumed is spread out more in time meaning the average power goes down. So this decrease in power consumption for LT coding matches very closely the increase in power caused by the extra packet transmissions.

For RaptorQ the same argument can be applied, but its encoder and decoder process is a little bit more complicated. As RaptorQ consists of an inner and outer code, it is also

typically decoded in two steps. While Raptor Codes have linear decoding time compared to the size of source data, the decoding effort is less linearly spread out in time than for LT codes. For LT codes, decoding progresses a little bit for each newly received symbol. Because of the two codes, Raptor Codes spends a bit more effort after receiving the last symbol in order to decode the whole message. This results in a more uneven distribution of energy over time, which is further complicated by the video decoder operating on the newly decoding data and the streaming software's data prefetch operation and bandwidth estimation policies. In summary, the LT codes even distribution of work in the decoding process translates into a more even process than RaptorQ, even though packets are arriving less often due to packet losses.

A. Multiple Clients and Live Streams

In the first experiment, only a single client was considered. Because fountain codes are rateless and consumers can receive packets in any order they are ideal for multi user and multicast scenarios, such as for live streams. As we saw in the single client scenario, the extra overhead of the fountain codes results in higher energy consumption compared to RUDP for lower loss rates, while being more efficient for extremely lossy channels.

With fountain coding, if several clients were to consume the same video stream at the same time, the sender wouldn't need to consider which individual packets of the stream each client have received, and the need for explicit feedback is essentially eliminated. This is especially important when multiple clients are considered over lossy channels. Depending on the size of the stream, and the size of the video chunks, it becomes likely that the different clients need different parts of the stream, and that the sender therefore needs to send a separate packet to each of them. Using fountain coding, this need is eliminated and a single packet will be sufficient for all of them which improves both power consumption and throughput.

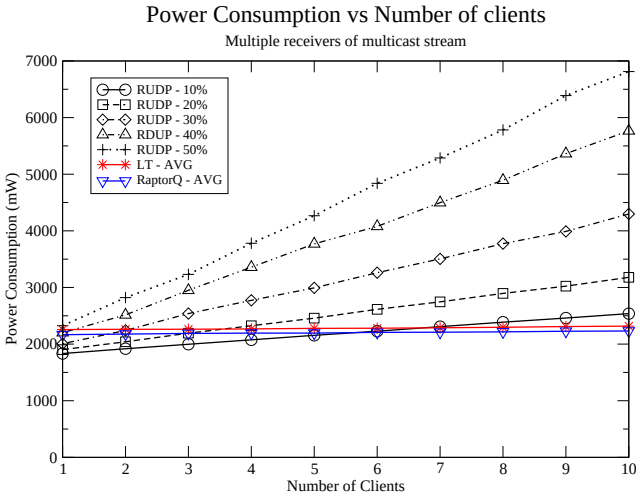

Fig. 5. Power consumption depending on number of receiving multicast clients

If we look at Figure 5 we can see the results for power measurements when the number of clients increases. The figure shows the power consumption for different packet loss rates for RUDP, but only the average for LT codes and RaptorQ because as saw above, this is not exactly dependent on the packet loss rate.

All clients are receiving the same video stream, but as the individual wireless channels are different (independent), clients drop different packets which needs to be retransmitted. The fountain codes maintain the same power consumption even as the number of clients increases. This means that for 10 clients the power consumption can be reduced between 14% and 205% compared to RUDP. For 2 clients, the consumption increases by 13% for 10% packet loss, to being reduced by 30% for 50% packet loss.

In this setup, all the clients experience the same level of packets loss rate, while in a more non experimental setup this might not be the case. The consumption would then be constrained by the client with the worst channel as the server needs to adapt to its need. It also means that other clients with better channels still receives packets which they don't need, although these packets will just be discarded. This is a classical problem, called the near-far problem, but which we will not specifically address in this paper. Client devices closer to the source server though, should be able predict this as packets are received in cycles depending on the chunk size. They could thus predict the remaining length of the cycle and turn off their radios and save more energy. We will analyze this is in future work.

B. Discussion

In the previous section VI-A we looked at the case of multiple receivers and argued that fountain codes would be a good case for multicast and live streams. UDP is often the protocol of choice in these scenarios because it offers less latency and in contrast to TCP, supports multicast. Another important reason is that modern video codecs accept some packet loss and so the retransmission features of TCP is not needed. In [14] it is studied how the quality of the video is impacted by packet loss, and the quality of the video degrades quickly as the loss ratio increases. By 10% it is so bad the video is almost unwatchable. One study [2] found that during conference setting, 28% of all packets were lost. This means that actual packet losses in WiFi network are much higher than commonly known, and multicast streams will be very vulnerable to this. This means that the schemes presented here, are of critical importance for achieving acceptable video qualities.

VII. Conclusion

In this paper we have presented a protocol scheme for video streaming over lossy wireless networks such as in WiFi networks. It uses a flexible Reliable UDP (RUDP) based protocol, and is very robust even over very lossy wireless channels and allows clients to request retransmission of individual packets, which may optionally allow a client to prioritize and request

packets of higher importance such as keyframes. We show how the energy consumption of the basic version of this protocol can be improved by using fountain codes to protect the video chunks. Because of the computational overhead of these fountain codes, for a single video stream there is no gain in power consumption except for higher loss rates. For multiple receivers very significant power savings can be made because the transmitter needs to independently consider the state of each receiver, while in a WiFi setting each receiver also receives the wireless transmission of packets they don't need. Using Fountain coding, the power consumption is more or less independent of both the packet loss rate and the number of receivers. For 10 users the gain is between 14% and 205%.

ACKNOWLEDGMENT

This work is supported by the European Celtic-Plus project CONVINcE and was partially funded by Finland, France, Sweden and Turkey.

REFERENCES

[1] Cisco. (2015, May) Cisco visual networking index predicts ip traffic to triple from 2014-2019; growth drivers include increasing mobile access, demand for video services. [Online]. Available: http://newsroom.cisco.com/press-release-content?articleId=1644203

[2] M. Rodrig, C. Reis, R. Mahajan, D. Wetherall, and J. Zahorjan, "Measurement-based characterization of 802.11 in a hotspot setting," in *Proceedings of the 2005 ACM SIGCOMM Workshop on Experimental Approaches to Wireless Network Design and Analysis*, ser. E-WIND '05. New York, NY, USA: ACM, 2005, pp. 5–10. [Online]. Available: http://doi.acm.org/10.1145/1080148.1080150

[3] D. J. MacKay, "Fountain codes," in *Communications, IEE Proceedings-*, vol. 152, no. 6. IET, 2005, pp. 1062–1068.

[4] A. N. Plymoth, P. Johansson, R. L. Cruz, O. Chipara, and W. G. Griswold, "Grapevine: hybrid cooperative opportunistic routing for challenged wireless networks using fountain coding," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 17, no. 1, pp. 61–70, 2013.

[5] T. Bova and T. Krivoruchka, "Reliable udp protocol," *draft-ietf-sigtran-reliable-udp-00. txt*, 1999.

[6] Q. Xu, V. Stankovic, and Z. Xiong, "Wyner ndash;ziv video compression and fountain codes for receiver-driven layered multicast," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 17, no. 7, pp. 901–906, July 2007.

[7] T. Schierl, S. Johansen, A. Perkis, and T. Wiegand, "Rateless scalable video coding for overlay multisource streaming in manets," *Journal of Visual Communication and Image Representation*, vol. 19, no. 8, pp. 500–507, 2008.

[8] S. Ahmad, R. Hamzaoui, and M. Al-Akaidi, "Unequal error protection using fountain codes with applications to video communication," *Multimedia, IEEE Transactions on*, vol. 13, no. 1, pp. 92–101, Feb 2011.

[9] C. Fragouli, J.-Y. Le Boudec, and J. Widmer, "Network coding: an instant primer," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 63–68, 2006.

[10] S. B. Wicker and V. K. Bhargava, *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.

[11] M. Luby, "Lt codes," in *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, 2002, pp. 271–280.

[12] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer, "L. minder," raptorq forward error correction scheme for object delivery," RFC 6330, August, Tech. Rep., 2011.

[13] R. G. Gallager, "Low-density parity-check codes," *Information Theory, IRE Transactions on*, vol. 8, no. 1, pp. 21–28, 1962.

[14] J. S. Mwela, "Impact of packet loss on the quality of video stream transmission," Ph.D. dissertation, Blekinge Institute of Technology, 2010.

[15] B. Trapp, "Raspberry pi: The perfect home server," *Linux J.*, vol. 2013, no. 229, May 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2492119.2492120

[16] M. Thompson and P. Green. (2012) Raspbian. [Online]. Available: https://www.raspbian.org/FrontPage

[17] I. Hubert. (2012) Tears of steel. [Online]. Available: https://mango.blender.org/