Best practices for mobile broadcast delivery and playback of multimedia content

Michael Luby, *IEEE Fellow*

***Abstract*— Although wide area wireless network capacity is growing significantly, the consumer demand for popular multimedia content delivered to mobile devices over these networks is growing even faster. An attractive alternative is to deliver multimedia content via broadcast, thereby economizing on the usage of network capacity. We describe best practice techniques for using forward error correction and interleaving the data to be broadcast for multimedia content, and receiver methodology for forward error correction decoding and de-interleaving received data for multimedia playback.**

***Index Terms*—broadcasting, mobile device playback, digital multimedia broadcasting, forward error correction, interleaved codes, just-in-time processing.**

I. Introduction

Internet content delivery is based on the ubiquitous use of HTTP, which allows the same common server infrastructure to be used to deliver all web content over all networks to all end devices. HTTP caches are deployed worldwide, providing scalable content delivery to hundreds of millions of end devices.

The delivery of popular multimedia content to mobile devices continues to experience explosive growth. The growth is driven by the availability of mobile devices with high quality displays and improvements in the underlying wireless network capacity (e.g., 3G/4G cellular data infrastructure). Examples of popular multimedia content include news and sports video clips, user-contributed media such as YouTube videos, streaming of live sports and concert events, high definition movies, video advertisements, TV episodes, video games, high definition pictures, and high fidelity music. Although the network capacity of wide area wireless networks is growing significantly, the consumer demand for popular multimedia content delivered to mobile devices over these networks is growing even faster.

As the strain on wireless network capacity grows with increasing appetite for mobile multimedia, the problem for infrastructure providers and operators is how to deliver more content to mobile devices economically, without negatively impacting user experience.

Manuscript received April 27, 2012. This work was supported by Qualcomm Incorporated.

Michael Luby is with Qualcomm Incorporated, Qualcomm Berkeley Research Office, Berkeley, CA 94704 USA (phone: 510-579-5568; e-mail: luby@qualcomm.com).

Relying solely on HTTP for delivery of content to mobile devices can cause wide area wireless network bottlenecks. Because HTTP sends a separate data flow to each device independently, the cost of wide area wireless network deployment and management needed to deliver a growing amount of multimedia content scales with the number of users and can become impractical. The user experience is good when the number of users accessing content using HTTP is low. However, the network becomes overloaded and the user experience deteriorates as more and more users access more and more content using HTTP. Failure of operators to keep up with users' capacity demand can result in a rapid and significant degradation of multimedia content playback quality – including playback stalls, rebuffering, and poor image quality. Battery life also suffers and other revenue-generating services, such as voice and messaging services, can also be adversely affected.

II. Broadcast delivery – eliminates bottleneck problems

Broadcast delivery allows a single data stream to reach multiple users simultaneously. Most importantly, it avoids separately sending redundant content to multiple users, a potential pitfall of the HTTP approach. For this reason, broadcast delivery of popular multimedia content is an attractive augmentation to unicast HTTP delivery, especially over wireless wide area networks where capacity may be limited.

One possible option under consideration for broadcast content distribution over wireless cellular networks is the Evolved Multimedia Broadcast Multicast Service (eMBMS) [1], specified by the Third Generation Partnership Program (3GPP). With eMBMS, even when the density of receiving devices is less than one per radio cell, the network efficiency of delivering content using 3GPP eMBMS is better than HTTP delivery. The advantages of delivery using 3GPP eMBMS increase as the popularity of the content increases, because the 3GPP eMBMS network capacity needed to deliver content remains constant as the number of receiving devices increases. Thus, the user experience remains good as the number of users receiving content from broadcast delivery ranges from a small number to a large number. This is quite an attractive alternative compared to HTTP delivery over the network, which requires network capacity to increase proportionally to

the number of receiving devices.

III. Broadcast content delivery use cases

We introduce two relevant use cases for broadcast content delivery – predictive broadcast delivery and on-the-fly broadcast delivery. For both use cases, scheduling broadcast delivery of content can be based on a number of factors, including end user subscriptions to services, previous viewing habits of end users, predictions of content popularity, etc. There are many other equally interesting use cases that are beyond the current scope, including streaming delivery of live multimedia content such as a sporting or music concert events.

A. Predictive broadcast delivery

Some content may be predictably popular and available for delivery prior to when it is to be consumed on mobile devices. Examples of such content include daily videos generated by major news organizations, popular video blogs, sports highlight video clips, financial overview video clips, scheduled video concerts, the top ten daily YouTube videos, best selling electronic books augmented with multimedia, video advertisements to be played back within other multimedia content, newly released songs of popular artists and main stream movies, mainstream TV episodes, software packages, video game software, firmware updates, etc.

This type of content can be pre-scheduled for broadcast delivery in order to pre-cache the content on mobile devices prior to end user consumption. It may be broadcast several minutes to several hours before it is used, e.g., delivered overnight or intermittently during the day as it becomes available. Users may also wish to establish their own profiles with schedules indicating which content they would like to receive in a pre-cached fashion.

B. On-the-fly broadcast delivery

Some content may become suddenly popular, for example as measured by a high amount of download requests. Examples of such content include breaking news event video clips, instant replay sports highlight clips, viral You Tube videos, independent films, music, video game software, software applications and patches, etc.

This type of content can be scheduled on-the-fly for broadcast delivery, i.e., scheduled for broadcast delivery while mobile devices are still currently downloading the content via HTTP. Implemented properly, content being played can be switched seamlessly and efficiently between HTTP and broadcast reception.

C. Broadcast content delivery benefits

Predictive and on-the-fly broadcast delivery allows operators to smooth out unpredictable surges in network traffic caused by popular content. Doing so preserves network capacity for other revenue generating services. Broadcast delivery can also provide a better experience to users. For example, pre-caching using broadcast can allow a media playback application to start working immediately, avoiding the normal startup delay ordinarily required when the application loads its initial content over the network. In addition, the network capacity saved due to the elimination of multiple redundant unicast data streams can be used to deliver higher quality media content (e.g., higher resolution).

IV. Broadcast content delivery challenges

With HTTP, content is delivered in response to explicit requests from end devices resulting from user input. With broadcast delivery, content is pushed to mobile devices by operator-controlled services based, for example, on user preferences, subscriptions, or previous behavior. Broadcast delivery of popular multimedia content can significantly reduce the strain on wireless wide area network capacity. However, there are also some challenges with broadcast delivery.

End users ultimately do not care how their content is delivered, be it via broadcast or HTTP – they expect a highly responsive and excellent user playback experience for the particular content they are interested in. One challenge with broadcast delivery is how to select which multimedia content to deliver, and when to deliver it. Another is how to ensure a broadcast multimedia user experience that matches or exceeds that provided when content is delivered using unicast HTTP. Video quality, timeliness of playback and battery consumption must not degrade from the expectations set by unicast HTTP.

A final challenge relates to the amount of resources consumed on the playback device to deliver a good user experience. Mobile device manufacturers expect that the resource requirements for multimedia processing are roughly fixed, whether content is delivered using unicast or broadcast. Device resources such as CPU, RAM, flash memory, and data paths between memories within the device must, in practice, not use more resources for processing broadcast content than are required for HTTP.

V. Broadcast content delivery strategies

To realize the benefits of broadcast, a strategy optimized for efficient processing of multimedia reception and playback is required. To appreciate the landscape, we begin with some background on how conventional broadcast delivery and playback applications operate, which involve compromises between fulfilling the objectives of efficient network utilization, good user experience and minimal device resource utilization.

A. Broadcast content delivery standards

In a broadcast delivery system, a server transmits data packets generated from multimedia content over a broadcast network, e.g., over an LTE network using 3GPP eMBMS [1].

Multimedia content is packaged and delivered as files for broadcast. The 3GPP eMBMS standard, as well as every other major broadcast/multicast standard, have adopted the FLUTE suite of protocols developed by the Internet Engineering Task Force (IETF) for this purpose, i.e., FLUTE [2], ALC [3], LCT [4], and the FEC building block [5].

B. Application layer forward error correction

Application-layer Forward Error Correction (AL-FEC) is an

integral ingredient in providing reliable file delivery within the FLUTE suite of protocols (see [7]). Due to a variety of reasons such as intermittent shadowing, fading, distance, or device state, some of the transmitted packets might not be received, and thus are considered lost. Using AL-FEC, lost information can be reconstructed in order to perform reliable delivery of content in the face of loss.

AL-FEC codes, hereafter shortened to FEC codes, allow for the recovery of lost data due to network packet loss without requiring resending of the lost data. For broadcast content delivery to protect against packet loss, FEC encoding is applied as follows. A content file to be transmitted is divided into source blocks and each source block is further divided into source symbols. The FEC encoder generates repair symbols for each source block from the source symbols of the source block. All symbols associated with a file are the same size, where the symbol size is typically chosen to fit into the payload of a data packet. All source blocks are all approximately the same size. The source symbols for all of the source blocks and the repair symbols generated from the source blocks are sent in packets in the broadcast transmission.

A receiver applies the FEC decoder to FEC encoded data received for each source block to recover that source block, where FEC encoded data for a source block is any combination of the source and repair symbols sent for that source block. The FEC decoder is able to fully recover each source block, even if some packets containing FEC encoded data for that source block are lost, as long as enough FEC encoded data for the source block is received. Ideally, an FEC code has the property that the amount of FEC encoded data needed to recover a source block is the size of the source block, but for many practical FEC codes an additional small overhead of additional FEC encoded data beyond the source block size is needed to ensure high reliability decoding of the source block. The source block size is chosen to be small enough that the FEC decoder can decode a source block using available RAM in the mobile receiving device.

C. Sequential broadcast

Sequential broadcast is a conventional transmission strategy that works as follows. The sender splits the content into source blocks and each source block is independently FEC encoded. The FEC encoded data is transmitted sequentially per source block, i.e., all FEC encoded data for a source block is transmitted after transmission of all FEC encoded data for all previous source blocks and before broadcast of all FEC encoded data for all subsequent source blocks.

Sequential broadcast makes it easy at the mobile device to efficiently receive and playback content. The receiving process at the mobile device receives the FEC encoded data for each source block as it arrives, immediately FEC decodes the source block, and writes the source block of the content to flash storage. Immediately after data reception, the recovered content is fully available for playback by the end user.

However, there are some network efficiency trade-offs inherent to sequential broadcast. Because the amount of RAM available for FEC decoding each source block is typically limited, the source block sizes cannot be large. Because the FEC encoded data for a source block is sent consecutively, the amount of FEC encoded data for the source block that is lost when there is a burst loss will typically be a substantial fraction of the size of that source block. Thus the amount of FEC encoded data that needs to be sent for each source block to protect against loss has to be relatively large compared to the size of the source block, and overall a substantial amount of overall FEC encoded overhead needs to be transmitted relative to the size of the content even when the overall amount of loss over the entire transmission of the content is not so large.

Consider Fig. 1, a sequential broadcast example delivering 100 MB of content transmitted over an 800 Kbps broadcast channel, where mobile devices experience 10 seconds bursts of packet loss on average 10 times per hour, and where mobile devices can decode source blocks up to 1 MB in size with the available RAM. The sender partitions the content into 100 source blocks of 1 MB each, generates FEC encoded data for each source block, and transmits the FEC encoded data for each source block sequentially. Even though not all source blocks will not be subject to an outage, an unpredictable set of the source blocks will be subject to an outage, and thus the amount of FEC encoded data that needs to be sent for each source block is at least 2 MB, i.e., 20 seconds of transmission, to ensure that at least 1 MB of the data transmitted for each source block is received at the mobile device.

Fig. 1 shows an example where there is a 10 second outage during reception of FEC encoded data for the second source block and there is another 10 second outage during reception of FEC encoded data for the third source block. For each of these two source blocks, 1 MB of the transmitted 2 MB of FEC encoded data is lost and only 1 MB of FEC encoded data arrives at the receiver to allow recovery of the source block, as shown at the bottom of Fig. 1.

The overall amount of FEC encoded data transmitted for the 100 MB content file to ensure reliable recovery of the entire content file at the mobile device is at least 200 MB, i.e., at least 2,000 seconds of transmission time. Thus, although the mobile device resources are minimized and end user experience is good with sequential broadcast, sequential broadcast can require a large amount of data to be transmitted over the network relative to the content size to ensure reliable delivery of the content.

D. Interleaved broadcast

Interleaved broadcast is another conventional transmission strategy that works as follows. The sender splits the content into source blocks and each source block is independently FEC encoded. The FEC encoded data is broadcast completely interleaved amongst the source blocks, i.e., all FEC encoded data for a source block is transmitted evenly interleaved with transmission of all FEC encoded data for all other source blocks.

Interleaved broadcast provides excellent protection against burst packet loss, as the loss is spread out equally over the FEC encoded data for all the source blocks of the content rather than being concentrated on one source block, and thus the amount of FEC encoded data that needs to be transmitted over the network relative to the content size to ensure reliable delivery of the content is commensurate with the overall

amount of loss during the transmission of the content.

Consider Fig. 2, an interleaved broadcast for the same example as in Fig. 1, i.e., for the example of delivering 100 MB of content transmitted over an 800 Kbps broadcast channel, where mobile devices experience 10 seconds bursts of packet loss on average 10 times per hour, and where mobile devices can decode source blocks up to 1 MB in size with the available RAM. The sender partitions the content into 100 source blocks of 1 MB, generates FEC encoded data for each source block, and transmits the FEC encoded data for each source block evenly interleaved amongst the FEC encoded data for all the other source blocks.

Fig. 2 shows the same examples of outage as shown in Fig. 1. However, in contrast to sequential broadcast shown in Fig. 1, each outage causes a small percentage of FEC encoded data lost equally from each of the source blocks at the receiver for interleaved broadcast shown in Fig. 2. In this example, because on average there are 10 outages per hour, with high probability there will be at most 10 outages during the less than half hour of reception at the mobile device, or at most 100 seconds of outage overall during the reception. Thus, since the FEC encoded data for each source block is spread equally across time during the entire transmission, at most 100 KB of data is lost from each source block due to the 100 seconds of outages. To protect against these outages for a single source block, the amount of FEC encoded data that needs to be sent for each source block is at most 1.1 MB, and the overall amount of FEC encoded data transmitted for the 100 MB content file to ensure reliable recovery of the entire content file at the mobile device is at most 110 MB, i.e., at most 1,100 seconds of transmission time. Thus, interleaved broadcast overcomes the network inefficiency drawbacks of sequential broadcast.

VI. Mobile device recovery and playback strategies

Conventional recovery of content on a mobile device receiving interleaved broadcast involves compromises. Conventionally, the interleaved FEC encoded data is written directly to flash storage as it is received without FEC decoding or de-interleaving. After all of the FEC encoded and interleaved data is received and stored, a post-processing step reads back the data from flash storage, FEC decodes each source block, and writes the recovered source blocks of the content back to flash storage. Only after the post-processing step completes can the end user access and playback the content. The post-processing step can be costly in terms of the amount of I/O required between flash storage and RAM, in terms of the CPU usage, and in terms of the amount of time it takes to complete the post processing. For the interleaved broadcast example in Fig. 2, the post-processing step can be lengthy, interrupt other applications that the end user is running, increase media playback startup delay, consume battery power unnecessarily, and cause twice as much data to be read from and written to flash memory as with the sequential broadcast example in Fig. 1 that is optimized for the mobile device. Furthermore, if the end user decides to not playback all or part of the content then the post-processing step needlessly consumes device resources for no benefit.

A. Just-in-time recovery and playback

We introduce just-in-time (JIT) recovery, which is an optimal mobile device strategy for reception and playback of broadcast delivered content. The end user is provided an excellent and highly responsive multimedia playback experience with JIT recovery, and the impact on mobile device resources is minimized. At the same time JIT recovery allows excellent network efficiency, since it is designed to work with interleaved broadcast or any other transmission strategy. JIT recovery can be extended to provide seamless integrated mobile device playback of multimedia content delivered via a combination of broadcast and HTTP.

JIT recovery requires minimal usage of mobile device resources during reception. The FEC encoded and interleaved data is minimally rearranged as it is received using a small amount of RAM before being written directly to flash memory. The reception process has little to no impact on other applications running at the same time. The user is free to playback other multimedia content or run other applications concurrent with reception, without fear of CPU, RAM or storage capacity being unduly impacted.

Unlike conventional processing, with JIT recovery there is no post processing to recover the entire content before playback of the content. When the user requests playback of portions of the content, JIT recovery reads the FEC encoded and interleaved data generated from the requested portions from flash storage into RAM, FEC decodes, and provides the recovered portions of content to the player just in time for playback. Thus, the multimedia content is available immediately for playback once reception is complete. The user may elect to playback arbitrary portions of the multimedia content, and the amount of data read from flash memory is proportional to the amount of multimedia content played back, and not proportional to the total original content size. This provides for a better overall user experience and usage of mobile device resources, especially when accessing only portions of large multimedia content collections.

JIT recovery can be extended to allow integrated delivery using a combination of unicast and broadcast. The extended strategy is especially relevant for the on-the-fly broadcast delivery use case.

B. Technical components of just-in-time recovery and playback

We now delve into an overview of the technology constituting JIT recovery. For simplicity, multimedia content is assumed to be large enough to require storage in flash memory (i.e., RAM alone would be insufficient). Reception processing includes receiving and storing of content data as it arrives. Once a mobile device has received enough data to begin playback, the user may choose to playback portions or all of the content. Playback processing includes accessing the appropriate data stored in flash memory and recovering the requested portions of content for playback to the user.

1) DRM protection of content

In some cases, especially for high value multimedia content, the content may be protected using encryption and digital rights management (DRM). DRM-protection is transparent to JIT recovery as long as the content is DRM-protected as streaming content – i.e., DRM-protected streaming content can be broadcast delivered as described, and the DRM-handling of the streaming content can be handled as it is handled for all other DRM-protected streaming content without any adverse impact on JIT recovery processing.

2) Flash memory

Mobile devices use flash memory to accommodate the persistent storage of large files. Flash memory has asymmetric read and write capabilities. Reading of an arbitrary sequence of pages of data from flash memory into RAM is fairly efficient but writing is only efficient when a large number of consecutive pages of data are written. Flash page sizes ranges from 1KB to 4KB, depending on the flash technology in use, and writing is typically efficient when the number of consecutive pages is a few hundred.

JIT recovery leverages these properties of flash memory. Reception processing organizes data in RAM so that each write to flash memory consists of a few hundred consecutive pages, where each page contains data generated from a particular block of the content. However, since the data may be received from an interleaved broadcast, different pages in the write may contain data generated from different blocks. Playback processing achieves the de-interleaving by reading a sequence of pages of data pertaining to a particular portion of requested content that may be scattered across non-sequential pages of flash memory.

3) RaptorQ

Among the several AL-FEC codes specified by IETF, the most advanced and suitable code for broadcast multimedia content delivery is RaptorQ, as specified in [6]. An in depth description and analysis of the RaptorQ code is provided in [8]. The IETF specification [6] consists of two major parts: the specification of the RaptorQ code, and the specification of how to use RaptorQ for file delivery in the context of the FLUTE suite of protocols. When used with the FLUTE suite of protocols, RaptorQ provides the most efficient usage of network capacity among the various options available.

Hereafter we assume RaptorQ is used with FLUTE to broadcast files, as described in [6]. A file is divided into source blocks, but also source blocks are further divided into sub-blocks as shown in Fig. 3. The partitioning also further divides each source block into source symbols and each sub-block into source sub-symbols, where each source symbol consists of one source sub-symbol from each of the sub-blocks of the source block. In the example shown in Fig. 3, the content is partitioned into two source blocks, each of which is further partitioned into four sub-blocks. The first source sub-symbols of the sub-blocks of a source block are concatenated to form the first source symbol of the source block, the second source sub-symbols of the sub-blocks of a source block are concatenated to form the second source symbol of the source block, etc., as shown in Fig. 3. The advantage of sub-blocking is that it provides perfect interleaving of the data across all sub-blocks of a source block – since data is lost in units of packets and since each packet contains data for all the sub-blocks of a source block, the amount of data received for each sub-block of a source block is guaranteed to be the same amount as for all other sub-blocks of the same source block, independent of packet loss patterns.

The RaptorQ code generates repair symbols from the source symbols of a source block, thus implicitly generating repair sub-symbols for each sub-block of the source block. All symbols associated with a file are the same size, where the symbol size is chosen to fit into the payload of a data packet. The RaptorQ code is a systematic code, meaning that the encoded symbols for a source block consist of the combination of the source symbols of that source block and the repair symbols generated from that source block.

The transmitting multimedia server applies the RaptorQ encoder to individual source blocks of the file to generate encoded data to send. Interleaved broadcast is used to optimize broadcast network capacity.

A receiver applies the RaptorQ decoder to FEC encoded data received for each sub-block to recover that sub-block. The RaptorQ decoder is able to fully recover each sub-block, even if some packets containing data for that sub-block are lost, as long as enough data for the sub-block is received. The partitioning into source blocks and sub-blocks recommended by the specification allows the receiver to recover even a very large file using a small amount of RAM, e.g., the sub-block size to which the RaptorQ decoding is applied may be 1 MB whereas the source block size may be 50 MB and the content size may be 1 GB. This is especially important for mobile and resource-constrained devices.

With the RaptorQ code, as described in [8], if there are K source symbols in a source block, the decoder can fully recover the source block from a set of K received encoded symbols with probability 99%, from a set of K+1 received encoded symbols with probability 99.99%, and from a set of K+2 received encoded symbols with probability 99.9999%, respectively. Furthermore, the encoding and decoding speed of the RaptorQ code is linear in the size of the source block – and overall the RaptorQ encoding and RaptorQ decoding CPU usage is negligibly small for the broadcast delivery of multimedia content applications described herein.

C. Technical description of just-in-time recovery and playback

The implementation of our solution minimizes device resource consumption. The receiving process works as follows. A buffer is allocated for each sub-block of the content, and as packets are received containing data for a particular sub-block it is placed into the buffer for that sub-block. Buffers containing at least a page size worth of data are considered eligible to be written to flash memory. When a large number of such buffers are eligible, they are written to flash memory in a batch to the next available set of consecutive pages, and a mapping is established to keep track

of which written pages correspond to which sub-blocks. Thus, the receiving process does a very partial de-interleaving of the packets as they are received, requiring very little RAM or CPU, and each write is to a large number of consecutive pages of flash memory. The partial de-interleaving ensures that each page of flash memory contains data about only one sub-block of the multimedia content.

When the user wants to playback portions of content, a playback process reads and processes the appropriate data from the flash memory and provides the appropriate portions of the multimedia content to the media player process for playback. Based on the portion of multimedia content to be played back, the appropriate sub-block of the multimedia content is identified, the mapping information created during the reception processing is used to determine which pages of flash memory to read, the data is read into RAM and then FEC-decoded on-the-fly before being provided to the media player. Because the sub-blocks are relatively small and the FEC decoding is very fast when using the RaptorQ decoder, it takes very little time to recover the source block contents. As a result, the startup time between a user request for content and when the playback begins is very short, e.g., a couple of hundred milliseconds.

The implementation only processes data that is required for playback, and thus the amount of data read and processed from flash memory is proportional to the amount of multimedia content played back. Furthermore, data is stored only once in the flash memory (in an interleaved and FEC-encoded form). It is not necessary to write a separate copy of the recovered multimedia content back to flash memory.

VII. Conclusion

Growing demand for multimedia content on mobile devices couples with user expectations for a high quality viewing experience creates a challenge for operators, infrastructure providers and mobile device manufacturers. By leveraging the best technologies and design strategies, it is possible to optimize broadcast delivery and mobile device playback to provide a great user experience at an affordable cost.

We introduced several strategies that can be applied to broadcast delivery to overcome common problems seen with today's current HTTP delivery method. By incorporating RaptorQ AL-FEC, using interleaved broadcast techniques, and practicing just-in-time data recovery, users can be ensured a quality viewing experience and good battery life with minimized CPU resource usage on their mobile devices. This results in a scalable strategy that achieves a good user experience, while maximizing use of existing resources. All of these technologies and strategies are available today.

References

[1] 3GPP TS 26.346 v10.3.0, *Multimedia Broadcast/Multicast Service (MBMS); Protocols and Codecs*, March 2012.

[2] T. Paila, M. Luby, R. Lehtonen, V. Roca, R. Walsh, *FLUTE - File Delivery over Unidirectional Transport*, IETF RFC 3926, October 2004.

[3] M. Luby, M. Watson, L. Vicisano, *Asynchronous Layered Coding (ALC) Protocol Instantiation,* IETF RFC 5775, April 2010.

[4] M. Luby, M. Watson, L. Vicisano, *Layered Coding Transport (LCT) Building Block,* IETF RFC 5651, October 2009.

[5] M. Watson, M. Luby, L. Vicisano, *Forward Error Correction (FEC) Building Block,* IETF RFC 5052, August 2007.

[6] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, L. Minder, *RaptorQ Forward Error Correction Scheme for Object Delivery,* IETF RFC 6330, August 2011.

[7] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft, *The Use of Forward Error Correction (FEC) in Reliable Multicast*, IETF RFC 5053, December 2002.

[8] Amin Shokrollahi and Michael Luby (2011) "Raptor Codes", Foundations and Trends® in Communications and Information Theory: Vol. 6: No 3-4, pp 213-322.

Michael Luby earned his B.Sc. in mathematics from the Massachusetts Institute of Technology (1975) and his Ph.D. in theoretical computer science from the University of California at Berkeley (1983).

He is currently VP of Technology at Qualcomm Incorporated, located in Berkeley California, and is working closely with multiple teams based in San Diego California. He has held several teaching, research and leadership positions with the University of Toronto, UC Berkeley, and the International Computer Science Institute. He was CTO and Co-founder of Digital Fountain Incorporated, which was acquired in 2009 by Qualcomm Incorporated.

Dr. Luby has made breakthrough contributions in the areas of coding theory, communication technologies and cryptography. He has received the 2002 IEEE Information Theory Society Information Theory Paper Award, the 2003 SIAM Outstanding Paper Prize, the 2007 IEEE Eric E. Sumner Communications Theory Award, the 2009 ACM SIGCOMM Test of Time award, and the 2012 IEEE Richard W. Hamming Medal. He is an IEEE Fellow and Member of the ACM.

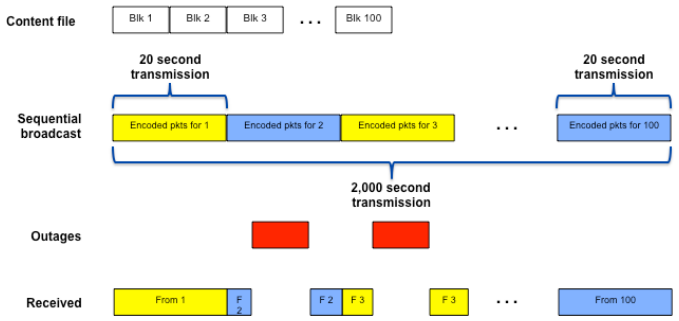

Fig. 1 – Sequential broadcast example

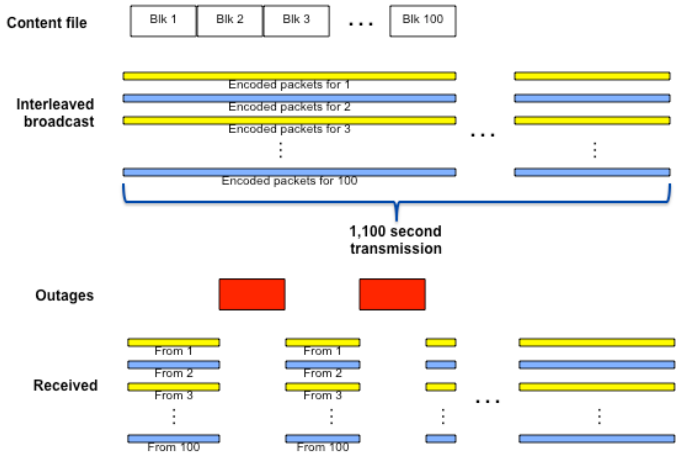

Fig. 2 – Interleaved broadcast example

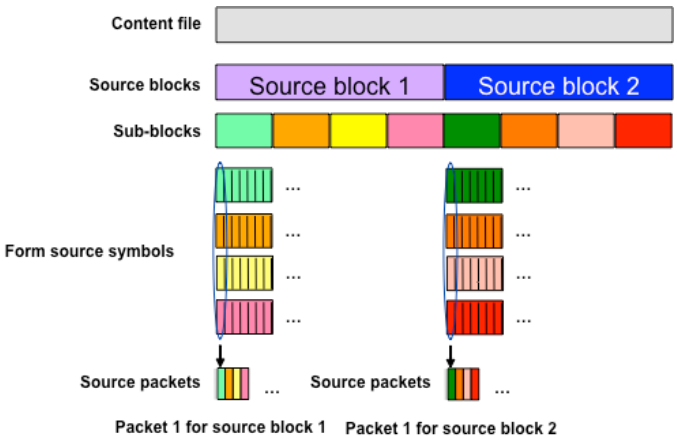

Fig. 3 – IETF RFC 6330 partitioning into source block and sub-blocks