APPLICATION LAYER ERROR CORRECTION CODING FOR RATE-DISTORTION OPTIMIZED STREAMING TO WIRELESS CLIENTS

*Jacob Chakareski**

Center for Multimedia Communication
Rice University, Houston, TX 77005 USA
cakarz@ece.rice.edu

Philip A. Chou

Signal Processing Research Group
Microsoft Corp, Redmond, WA 98052 USA
pachou@microsoft.com

ABSTRACT

This paper addresses the problem of streaming packetized media over a lossy packet network to a wireless client, in a rate-distortion optimized way. We introduce an incremental redundancy scheme that combats the effects of both packet loss and bit errors in an end-to-end fashion, without support from the underlying network or from an intermediate base station. The scheme is combined with an optimization framework that enables the sender to compute which packets it should send, out of all the packets it could send at a given transmission opportunity, in order to meet an average rate constraint while minimizing the average end-to-end distortion. Experimental results show that our system is robust and maintains a quality of service over a wide range of channel conditions. Up to 8 dB performance gains are registered over systems that are not rate-distortion optimized, at bit error rates as large as 10^{-2}.

1. INTRODUCTION

This paper addresses the problem of streaming packetized media over a lossy backbone packet network to a wireless client, using a sender-driven transmission scheme. Packets may be lost in the backbone network due to congestion, or they may be corrupted in the wireless link due to fading. The problem under consideration is illustrated in Figure 1.

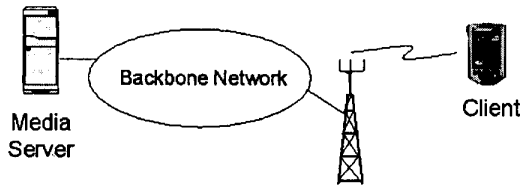

Fig. 1. Streaming on demand to a wireless client.

It is assumed that the server runs UDP Lite [1] or a similar network protocol, where only a partial (header) checksum is applied to packets at the network layer. Thus, only packets whose RTP/UDP/IP headers have not been corrupted are received by application. This also means that the application can observe bit errors in the packet payload. Furthermore, we assume that both the sender and receiver do not have control over any network resources available along the communication path between the two. For example, the sender cannot change the delivery priorities placed on its packets by the backbone network, the rate of the channel code applied by the base station to packets sent across the wireless link, etc.

We introduce a streaming system that consists of two components. One is an incremental redundancy transmission scheme that combats the loss and corruption effects of the communication channel. It operates end-to-end and hence does not need any cooperation from the underlying network infrastracture or from an intermediate base station. By inserting redundancy at the application layer, this scheme exploits to our advantage the ability of the client to observe corrupted payloads. The scheme is incremental in that it sends redundant information only if needed and hence it is bandwidth efficient. Thus we denote it Incremental Redundancy (IR) transmission. The second component is an optimization procedure, based on the Sensitivity Adjustment (SA) algorithm introduced in [2], which has been modified here to suit the settings of the problem under consideration. Using the optimization procedure, rather than streaming the packetized media in a fixed sequence according to presentation time, the sender can choose a transmission policy for each data unit that minimizes the expected end-to-end distortion of the entire presentation subject to a transmission rate constraint. The solution to this resource allocation problem is obtained by minimizing a Lagrangian, taking into account the data units' dependence relationships in addition to their different delivery deadlines and basic importances.

2. PRELIMINARIES

In a streaming media system, the encoded data are packetized into *data units* and are stored in a file on a media server. All of the data units in the presentation have interdependencies, which can be expressed by a directed acyclic graph as illustrated in Figure 2. Each node of the graph corresponds to a data unit, and each edge of the graph directed from data unit l' to data unit l implies that data unit l can be decoded only if data unit l' is first decoded.

Associated with each data unit l is a size B_l, a decoding time $t_{DTS,l}$, and an importance Δd_l. The size B_l is the size of the data unit in bytes. The decoding time $t_{DTS,l}$ is the time at which the decoder is scheduled to extract the data unit from its input buffer and decode it. (This is the decoder timestamp, in MPEG terminology.) Thus $t_{DTS,l}$ is the delivery deadline by which data unit l must arrive at the client, or be too late to be used. Only packets whose RTP/UDP/IP headers have not been corrupted are received by application. Packets containing data units that arrive after the data units' delivery deadlines are also discarded. The importance Δd_l is the amount by which the distortion at the receiver will *decrease*

*Jacob Chakareski was an intern with Microsoft Corporation.

if the data unit arrives on time at the receiver and is decoded.

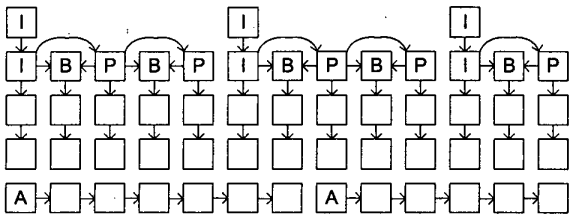

Fig. 2. Typical directed acyclic dependency graph for video and audio data units.

The communication channel, which in effect is an aggregate of the backbone packet network and the wireless link, is modeled as an independent time-invariant packet erasure channel with random delays at the packet level and as a binary symmetric channel (BSC) at the bit level. This means that if the sender inserts a packet into the network at time t, then the packet is lost with some probability, say ϵ_F, independent of t. However, if the packet is not lost, then it arrives at the receiver at time t', where the forward trip time $FTT = t' - t$ is randomly drawn according to probability density p_F. Furthermore, the individual bits of the received packet are independently and symmetrically corrupted with a probability BER_F. Therefore even though the packet may arrive at the client device, it might still be dropped by the network layer at the client side if its header has been corrupted. Thus we introduce a modified probability of packet loss, $\epsilon'_F = \epsilon_F + (1 - \epsilon_F)(1 - (1 - BER_F)^{N_h})$, that accounts for this. N_h is the size of the packet header in bits. In our experiments we use as the density p_F the shifted Gamma distribution with parameters (n_F, α_F) and right shift κ_F.

3. INCREMENTAL REDUNDANCY TRANSMISSION

A sender implementing the IR scheme enables a client to exploit corrupted payloads, by sending to the client redundancy packets (henceforth denoted *parity* packets) in addition to the regular data (henceforth denoted *systematic* packets). A parity packet contains in effect parity bits of a codeword, obtained when a systematic error-correction code is applied to a systematic packet. Parity packets are sent only if necessary.

A block diagram of the IR scheme is shown in Figure 3. We explain its details next. To verify the integrity of the data at the client, each data unit is pre-encoded by the server with a binary Cyclic Redundancy Check (CRC) code with a generator polynomial $g(x) = x^{16} + x^{15} + x^2 + 1$ [3]. It is assumed that bit errors cannot go unnoticed by the CRC code. The CRC encoded data represents a systematic packet, denoted Packet 0 in Figure 3, and is transmitted at the first transmission decision for the data unit. If the server receives a positive acknowledgement (ACK) before it decides to send this data unit again, nothing more is sent. An ACK means that the client has recovered the data unit without bit errors. However, if no feedback is received, then the server sends Packet 0 again. This is repeated until the server sees a negative acknowledgement (NAK) for one of the earlier transmissions. A NAK at this point means that the client has received a copy of the data unit, with bit errors.

After the first NAK arrives, the server generates a parity packet using a binary rate 1/2 Recursive Systematic Convolutional (RSC)

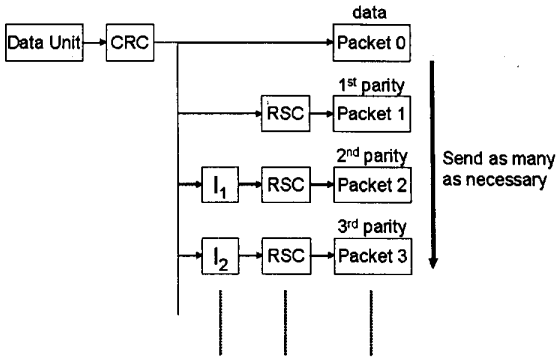

Fig. 3. The Incremental Redundancy transmission scheme.

encoder with generator polynomial matrix

$$G(x) = \begin{bmatrix} 1 & \frac{x^2 + x + 1}{x^2 + 1} \end{bmatrix}$$

[4]. It then sends this parity packet, denoted Packet 1, to the client. From that point on, whenever the server needs to transmit a packet for the data unit again, it generates another parity packet, denoted Packet k, by first applying a pseudo-random interleaver I_{k-1} to the systematic packet, and then using the RSC encoder to generate the parity packet, as illustrated in Figure 3. The role of the interleaver is to improve error correction performance if the client receives more than one parity packet. The server may transmit a packet for data unit l at any one of its transmission opportunities $t_{k,l}$, $k = 1, 2, \ldots, N_l$, according to its transmission policy, until it receives an ACK. Receiving an ACK at this point means that the client has recovered the data unit without bit errors, using the packets it has received. The server's transmission policy is determined by the optimization procedure described in the next section. The policy takes into account the server's history of previous transmissions as well as its history of acknowledgments.

We just explained how the server acts in the IR scenario. Now we explain how the client acts. Upon receiving the first parity packet, Packet k, $k \geq 1$, the client tries to decode the corrupted data unit using the List Viterbi algorithm [5]. If $k > 1$, the systematic packet is first interleaved with I_{k-1}, decoded, and then inverse interleaved with I_k^{-1}. If $k = 1$, no interleaving and inverse interleaving are needed. If decoding fails (i.e., the CRC check fails), a NAK for Packet k is sent to the server. Upon reception of the second parity packet, say Packet l, $l \geq 1$, $l \neq k$, the client tries to decode the data unit again but now uses the Turbo decoding procedure [6] with the 3 packets (1 systematic and 2 parity) received so far. If decoding fails once more, a NAK for Packet l now is sent to the server. Upon receiving a third parity packet, the Turbo decoding procedure is initiated once more, now with 1 systematic and 3 parity packets. This last step is iterated as long as the client observes a decoding failure and sends a NAK back to the server for the last received parity packet. If decoding succeeds the client issues an ACK for the last received parity packet.

Next, we study the performance of the IR transmission scheme. Figure 4 illustrates the dependence of the probability of decoding success (PoDS) on the total number of parity packets received by the client. It can be seen that for BERs $\leq 10^{-2}$, at most 2 parity

packets are needed for decoding. On the whole, including BER = 10^{-1}, not more than 4 parity packets suffice to observe success in decoding at every instance.

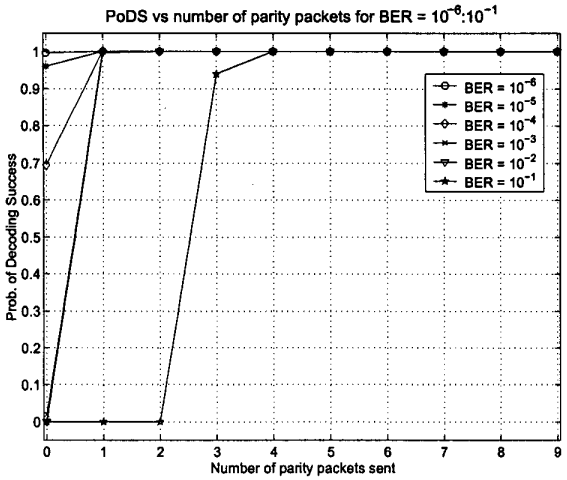

Fig. 4. Probability of Decoding Success vs. number of parity packets received.

4. R-D OPTIMIZATION USING SENSITIVITY ADJUSTMENT

Suppose there are L data units in the multimedia session. Let π_l be the transmission policy for data unit $l \in \{1, \ldots, L\}$ and let $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_L)$ be the vector of transmission policies for all L data units. A policy π_l is a transmission schedule according to which systematic or parity packets for data unit l are transmitted until an ACK is received.

Any given policy vector $\boldsymbol{\pi}$ induces an expected distortion $D(\boldsymbol{\pi})$ and an expected transmission rate $R(\boldsymbol{\pi})$ for the multimedia session. We seek the policy vector $\boldsymbol{\pi}$ that minimizes the Lagrangian $D(\boldsymbol{\pi}) + \lambda R(\boldsymbol{\pi})$ for some Lagrange multiplier $\lambda > 0$, and thus achieves a point on the lower convex hull of the set of all achievable distortion-rate pairs.

The expected transmission rate $R(\boldsymbol{\pi})$ is the sum of the expected transmission rates for each data unit $l \in \{1, \ldots, L\}$:

$$R(\boldsymbol{\pi}) = \sum_l B_l \rho(\pi_l). \tag{1}$$

where B_l is the number of bytes in data unit l and $\rho(\pi_l)$ is the *expected cost* per byte, or the expected number of transmitted bytes per source byte under policy π_l. The expected distortion $D(\boldsymbol{\pi})$ is somewhat more complicated to express, but it can be expressed in terms of the *expected error*, or the probability $\epsilon(\pi_l)$ for $l \in \{1, \ldots, L\}$ that data unit l does not arrive at the receiver on time under policy π_l:

$$D(\boldsymbol{\pi}) = D_0 - \sum_l \Delta D_l \prod_{l' \preceq l} (1 - \epsilon(\pi_{l'})), \tag{2}$$

where D_0 is the expected reconstruction error for the presentation if no data units are received and ΔD_l is the expected reduction in reconstruction error if data unit l is decoded on time. The product $\prod_{l' \preceq l}(1 - \epsilon(\pi_{l'}))$ is the probability that data unit l and all of its ancestors in the acyclic directed graph (see Figure 2) arrive at the receiver on time under their respective policies.

Finding a policy vector $\boldsymbol{\pi}$ that minimizes the expected Lagrangian $J(\boldsymbol{\pi}) = D(\boldsymbol{\pi}) + \lambda R(\boldsymbol{\pi})$, for $\lambda > 0$, is difficult since the terms involving the individual policies π_l in $J(\boldsymbol{\pi})$ are not independent. Therefore, we employ an iterative descent algorithm, called the SA algorithm, in which we minimize the objective function $J(\pi_1, \ldots, \pi_L)$ one component at a time while keeping the other variables constant, until convergence. For more details on the optimization procedure, the reader is referred to [7].

5. EXPERIMENTAL RESULTS

Here we investigate the distortion-rate performance for streaming one minute of packetized audio content using different methods. The audio content, the first minute of Sarah McLachlan's *Building a Mystery*, is compressed using a scalable version of the Windows Media Audio codec. The codec produces a group of twelve 500-byte data units every 0.75 seconds for a maximum data rate of 64 Kbps. All twelve data units in the mth group receive the same decoding timestamp, equal to $0.75m$.

We compare two streaming systems. Both of them perform R-D optimized scheduling of the packet transmissions at the sender, with the lagrange multiplier λ fixed for the entire presentation. The server uses its history of previous transmissions as well as its history of acknowledgements to determine which packets to transmit (or retransmit) at each transmission opportunity. System 1, *ACK only*, is the system introduced in [2]. Here only positive acknowledgements can be sent back to the server upon receipt of packets by the client. System 2, *N+ACK+P*, is the system described in this work. Two types of feedback packets are available: ACK and NAK. Details of how the server chooses the optimal transmission policy are given in [7].

All of the systems use the same playback delay (750 ms). The channel is specified with the following parameters: $\epsilon_F = \epsilon_B =$ 10 %, $T = 100$ ms, $\kappa_F = \kappa_B = 50$ ms $(0.5\ T)$, $\eta_F = \eta_B =$ 2 nodes, $1/\alpha_F = 1/\alpha_B = 25$ ms $(0.25\ T)$. Transmitted packets are dropped at random, with the modified drop rate, ϵ_F', to account for the non-zero BER. Those packets that are not dropped receive a random delay and their payload is corrupted with the given BER of the channel, using a pseudo-random number generator. The pseudo-random number generator is initialized to the same seed for each of the systems compared. Two cases for the packet header size are employed: regular (320 bits) and compressed (32 bits). A compressed IP/UDP/RTP header size of 32 bits is typical using header compression schemes such as [8].

We examine the signal-to-noise ratio (SNR) in dB of the end-to-end perceptual distortion, averaged over the one minute long audio clip, as a function of the available bitrate (Kbps). Figure 5 illustrates the SNR performance of the two systems for the case of regular header size.

It can be seen from Figure 5 that for low BERs, both systems perform similarly, with System 2 doing a bit better than System 1. The difference in performance becomes more exaggerated when the BER starts increasing. It is interesting to note that for BER = 10^{-3} the performance of System 1 becomes poor, while System 2 is still able to maintain a good quality of service. Finally, for BER $> 10^{-3}$ even System 2 performs poorly. This can be explained by the fact that at such high BERs every single packet is received with

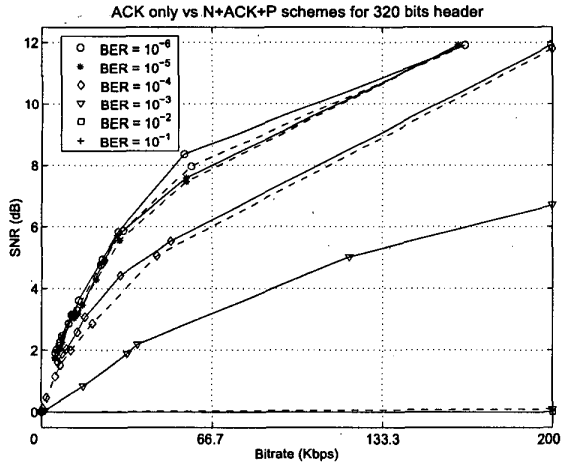

Fig. 5. RD performance for header size 320 bits and BER = 10^{-6}:10^{-1}. (dashed = Syst. 1, solid = Syst. 2)

a corrupted header and thus is dropped by the IP layer at the client side. Hence the client never gets a chance to see a transmitted packet and exploit the benefits of the IR transmission scheme.

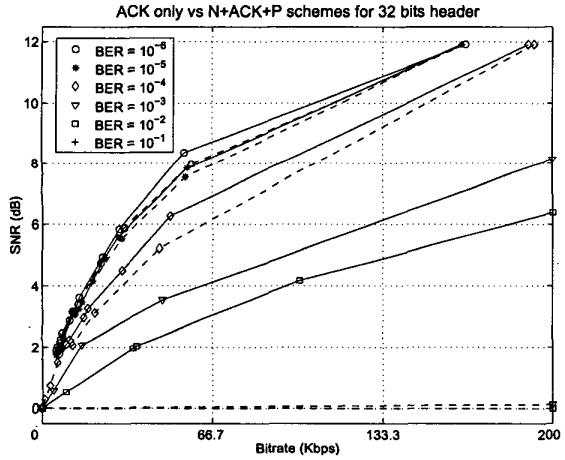

Fig. 6. RD performance for header size 32 bits and BER = 10^{-6}:10^{-1}. (dashed = Syst. 1, solid = Syst. 2)

A similar situation is observed for the compressed header case, as shown in Figure 6. The difference in performance is minimal for low BERs, with the gap in performance becoming more significant as the BER increases. The major difference between the two header size cases is in the performance of System 2 for BER = 10^{-2}. Due to the smaller header size now, the IR scheme can still be exploited by the client even at this BER. Lastly, for the same reasons as in the previous header size case, even System 2 performs poorly at extremely high BERs ($\geq 10^{-1}$).

6. CONCLUSIONS

A system for distortion-rate optimized streaming to wireless clients over lossy packet networks has been presented. The system consists of two components: a bandwidth efficient transmission scheme that enables a receiver to exploit corrupted payloads to its benefit. In combination with it, the sender is presented with a R-D optimization framework for packet scheduling in a sender-driven scenario. By exploiting the unequal sensitivity of a multimedia presentation to loss of different constituent components and the fact that the client can observe corrupted payloads, our system obtains superior performance over existing solutions and thus uses the available bandwidth in a most cost-effective way. However, for extremely high bit error rates (above 10^{-2}), header corruption limits the effectiveness of our system. Further improvement requires that the headers are protected below the application layer, i.e., at the link, network, and transport layers.

7. REFERENCES

[1] L.A. Larzon, M. Degermark, and S. Pink, *UDP Lite for Real Time Multimedia Applications*, Technical Report HPL-IRI-1999-001, HP Labs, Bristol,United Kingdom, April 1999.

[2] P.A. Chou and Z. Miao, *Rate-distortion optimized streaming of packetized media*, Technical Report MSR-TR-2001-35, Microsoft Research, Redmond, WA, February 2001.

[3] T. Ramabadran, and S.S. Gaitonde, *A tutorial on CRC Computations*, IEEE MICRO, pp. 62-75, August 1998.

[4] L.H. Charles Lee, *Convolutional Coding: Fundamentals and Applications*, Artech House, Norwood, MA, 1997.

[5] N. Seshadri and C. Sundberg, *List Viterbi decoding algorithms with applications*, IEEE Transactions on Communications, pp. 313-323, February 1994.

[6] C. Heegard, and S.B. Wicker, *Turbo Coding*, Kluwer Academic Publishers, January 1999.

[7] J. Chakareski and P.A. Chou, *On Computation of Distortion-Rate Curves for Wireless Streaming of Multimedia Presentations*, submitted to IEEE Data Compression Conference (DCC) 2002, Snowbird, Utah, USA, April 2002.

[8] C. Bormann et al., *Robust Header Compression (ROHC)*, IETF RFC 3095, `http://www.ietf.org/rfc/rfc3095.txt`, Proposed Standard, July 2001.