A HYBRID FEC-ARQ PROTOCOL FOR LOW-DELAY LOSSLESS SEQUENTIAL DATA STREAMING

Ying-zong Huang

Electrical Engineering and Computer Science
Massachusetts Institute of Technology

Sanjeev Mehrotra, Jin Li

Communication and Collaboration Systems
Microsoft Research, Redmond, WA

ABSTRACT

Interactive Internet Applications that rely on sequential streams for lossless data exchange often use retransmission protocols (e.g. TCP) for reliability and the guarantee of sequential data ordering. More so than for bulk file transfer or media delivery, lossless sequential streaming poses an even greater challenge for the common problem cases of retransmission protocols, such as lossy links or long network paths, manifesting as significant latency in the interactive user experience. We propose a hybrid FEC-ARQ protocol built on a packet streaming code that reduces to a simple strategy over sending or resending original data packets or check packets combining undecoded packets, based on actual network conditions. Experimental results show that our proposed protocol can significantly improve the total delay over retransmission and other schemes that use FEC, under a range of bandwidth and loss scenarios.

1. INTRODUCTION

Cloud Computing is taking off, and promises world-wide efficient and inexpensive computing and communications for individuals and businesses. Most Cloud Computing applications are interactive Internet Applications that function on the assumption of sequential streaming of reliable and in-order data delivery. That is, when a client A is connected to a server B, it is assumed that the data exchanged is guaranteed to arrive perfectly at the other side in the same order that it is written. This greatly simplifies the programming of the client and the server, as the task of state synchronization between the client and the servers or among multiple clients becomes straightforward, much like in single desktop computing.

As one of the core Internet protocols, TCP (Transmission Control Protocol) provides the sequential stream delivery required by many of the Cloud Computing applications. TCP relies on positive acknowledgement with retransmission to guarantee the reliability of data delivery. The technique is often called ARQ (Automatic Repeat reQuest) in the error control literature. Unfortunately, ARQ based loss recovery may incur high end-to-end delay, especially if the links on which losses occur have long path delay. This is because TCP only retransmits data after it receives confirmation that data is lost, and in high path delay links, this confirmation takes time. Moreover, a lost packet in an in-order stream effectively blocks all subsequent data from being delivered to the upper layer application, which manifests as poor responsiveness for the Internet Applications, so avoiding losses in the first place is even more critical.

In this paper, we propose an adaptive FEC-ARQ protocol to support low-delay sequential streaming for improved application responsiveness. Although the concept of combining FEC and ARQ is by no means new, our proposed protocol is significantly different from existing hybrid FEC-ARQ literatures (e.g. [1, 2, 3, 4]). These works are concerned with throughput and power for wireless transmission, quality management for real-time media, feedback complexity for scalable multicast, etc. They adopt some kind of block coding structure in forming the FEC code (e.g. [5, 6, 7]), and optimize mostly average throughput, not delay of a sequential stream. What makes our proposed FEC-ARQ unique is its capability to cater correctly to the in-order property in sequential streaming (i.e., to make explicit and minimize cascading delay from lost data mid-stream).

We begin with a very simple packet "streaming code" that has the correct notion of decoding for sequential data streams. Then we wrap a feedback-aware transmission policy around it to create a hybrid FEC-ARQ protocol for packet erasure channels. The proposed protocol manifests itself in the same interface as a traditional TCP protocol, so it allows most of the existing Internet Applications to use the protocol with minimal modification. It also respects message boundaries for those applications that require it. Furthermore, packets sent using this protocol can be notionally identified (and separated) as original data packets, retransmission packets, and checksum packets for the benefit of compatibility with potential receivers that only support original data, or only original data and retransmission (ARQ). Finally, the protocol is extensible to more complex network requirements than presented in this paper.

The paper is organized as follows. We describe the system model in Section 2, where we also describe the encoder and decoder structures of the streaming code that is core to

Table 1: Notation

m:	Index of the last sequentially decodable packet. For the encoder this is the index of the last packet known to be sequentially decodable.
s:	The number of packets that are decoded. After decoding, the index of the last sequentially decoded packet is given by $m+s$.
l:	Index of source packet when determining probability of sequential decodablity at the encoder.
k:	Index of the channel packet.
$n(\mathbf{y}[k])$:	The LIS packet index of the kth channel packet.
$\mathcal{P}$:	Set of positively acknowledged channel packets.
$\mathcal{N}$:	Set of negatively acknowledged channel packets.
$\mathcal{U}$:	Set of unacknowledged channel packets.
$\mathcal{F}$:	Set of all consumed yet undecoded source packets. For the encoder this is the set of packets which are not known to be sequentially decodable.
$\mathcal{Q}$:	Set of unconsumed source packets waiting in the source queue.
L:	The number of undecoded source packets ($L = \|\mathcal{F}\|$). The maximum LIS packet index of a check packet created by the encoder is given by $m+L$.

the protocol, and give some of the code's properties. Then, in Section 3, we describe how feedback informs the sender on choosing a transmission policy to form a hybrid FEC-ARQ protocol. In Section 4, we show experiments comparing the delay performance of the proposed protocol with that of pure ARQ under the same network conditions, followed by conclusions.

2. SYSTEM MODEL AND FORWARD ERROR CORRECTION CODE

We assume that the sequential data stream consists of a series of packets, called *source packets*, of potentially variable sizes, which must be delivered to the receiver losslessly and in-order. Let the sequence of source packets be denoted as $\mathbf{x}[1], \mathbf{x}[2], \mathbf{x}[3], ...$, where each packet $\mathbf{x}[i]$ is represented as a row vector of finite-field elements chosen from a suitable Galois-Field F. A common choice is to use $F = GF(2^8)$, which makes each finite-field element a byte in the data stream.

Let the sequence of packets sent on the network, called *channel packets*, be denoted as $\mathbf{y}[1], \mathbf{y}[2], \mathbf{y}[3], ...$, each of which is again a row vector of finite field elements in F. We consider the network as an erasure channel with packet loss probability ϵ. The network channel condition may fluctuate over time, leading to a time-varying packet loss probability $\epsilon_1, \epsilon_2, \epsilon_3, ...$ for each packet sent over the network. At every transmission opportunity, a channel packet is formed and sent based upon estimated network conditions, feedback, and transmission policies.

We define a packet to be *sequentially decodable* if it and all the packets upon which it depends are decodable. For a stream with in-order requirement, packet $\mathbf{x}[i]$ is considered sequentially decodable if $\mathbf{x}[j]$ for $j \leq i$ are decoded.

Let the delay of a source packet $\mathbf{x}[i]$ be the amount of time from when it becomes available at the sender, to the time it is sequentially decodable at the receiver. In particular, this notion of delay includes the delay incurred at the sender before information in $\mathbf{x}[i]$ is used, the path delay in the propagation across the network, and the delays at the decoder until $\mathbf{x}[i]$ becomes sequentially decodable, including time for feedback and retransmission if network packet loss is severe. The principal design goal of our system is to reduce the delay on a high fraction of the packets to ensure good responsiveness.

Next we describe how we encode and decode. Some of the notation we will use to index the packets and various sets of packets is shown in Table 1. It may also be helpful to refer to Figure 1.

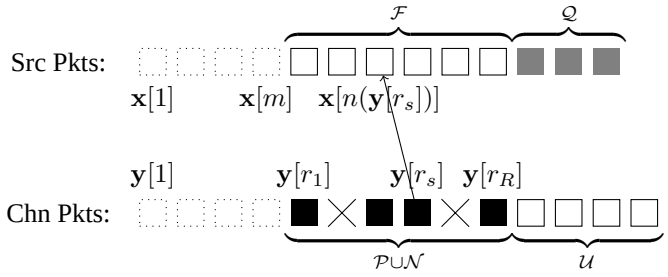

Fig. 1: A schematic diagram showing a snapshot of the system in mid-operation. Each block represents a packet. The past is to the left and the future is to the right. For the source packets from the encoder's perspective, ■ are unconsumed packets, □ are in-flight packets, and ⬚ are already decoded packets. For the channel packets from the decoder's perspective, □ are unseen packets, ■ are received packets, ╳ are lost packets, and ⬚ are packets no longer useful. The arrow connects the channel packet with its LIS packet.

2.1. Code Structure and Encoding

Most existing hybrid FEC-ARQ systems use a block-based forward error correction code for ease of implementation. However, it is not simple to choose the right block size: short block codes have lower coding delay but weaker error correction capability, while long block codes have stronger error correction capability but higher coding delay. In this work, we develop a packet streaming code, which is simply a randomly linear code that gradually combines new data with old. The code is similar to the code proposed by Martinian in [8] for delay-aware feedback-less systems. Although the

use of random linear codes is not particularly innovative, our FEC-ARQ framework that incorporates an adaptation of this code is. Our main contribution lies in the portion of the protocol that decides which channel packets to send, to minimize delay of sequentially decoding packets – the portion of total delay under our control.

A coding structure where sequential decodability is needed is to always include old data. This is best explained by the semi-infinite generator matrix $\mathbf{G}$ in the following encoding example:

$$\underbrace{\begin{bmatrix} \mathbf{y}[1] \\ \mathbf{y}[2] \\ \mathbf{y}[3] \\ \mathbf{y}[4] \\ \mathbf{y}[5] \\ \mathbf{y}[6] \\ \vdots \end{bmatrix}}_{\mathbf{Y}} = \underbrace{\begin{bmatrix} f_{1,1} & 0 & 0 & \dots \\ f_{2,1} & f_{2,2} & 0 & \dots \\ f_{3,1} & f_{3,2} & 0 & \dots \\ f_{4,1} & f_{4,2} & 0 & \dots \\ f_{5,1} & f_{5,2} & f_{5,3} & \dots \\ f_{6,1} & f_{6,2} & f_{6,3} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}}_{\mathbf{G}} \underbrace{\begin{bmatrix} \mathbf{x}[1] \\ \mathbf{x}[2] \\ \mathbf{x}[3] \\ \vdots \end{bmatrix}}_{\mathbf{X}} \quad (1)$$

Here, *every* channel packet is a linear combination of *all* source packets from $\mathbf{x}[1]$ up to some jth source packet $\mathbf{x}[j]$,

$$\mathbf{y}[k] = \sum_{i=1}^{j} f_{k,i}\mathbf{x}[i]$$

where $f_{k,i} \in F\backslash\{0\}$ are randomly drawn non-zero coefficients. Here, j is the index of the last included source packet (*LIS packet*) of $\mathbf{y}[k]$, denoted $n(\mathbf{y}[k])$.

Because of sequential decodability, this coding structure of using random linear combinations of all consumed source packets is fully general. Specifically, if some $\mathbf{x}[i]$ is not decodable, including it in a linear combination is necessary to help it to decode; once it is decoded, its inclusion in further linear combinations is not a detriment because its contribution can always be subtracted out, as will be shown in Sec 2.2.

The code we actually use makes a few modifications to the above. Without loss of generality, suppose now that $\mathbf{x}[m+1]$ is the first source packet *not* known to be sequentially decodable – all prior source packets are known to be sequentially decodable based upon acknowledgments. Obviously there is no need to include $\mathbf{x}[1], ..., \mathbf{x}[m]$ in linear combinations. Besides this, another modification is illustrated in the second encoding example that shows all channel packets with LIS packet $\mathbf{x}[m+1]$ or after:

$$\underbrace{\begin{bmatrix} \mathbf{y}[1] \\ \mathbf{y}[2] \\ \mathbf{y}[3] \\ \mathbf{y}[4] \\ \mathbf{y}[5] \\ \mathbf{y}[6] \\ \vdots \end{bmatrix}}_{\mathbf{Y}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & \dots \\ 0 & 1 & 0 & \dots \\ f_{3,m+1} & f_{3,m+2} & 0 & \dots \\ f_{4,m+1} & f_{4,m+2} & 0 & \dots \\ 0 & 0 & 1 & \dots \\ f_{6,m+1} & f_{6,m+2} & f_{6,m+3} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}}_{\mathbf{G}} \underbrace{\begin{bmatrix} \mathbf{x}[m+1] \\ \mathbf{x}[m+2] \\ \mathbf{x}[m+3] \\ \vdots \end{bmatrix}}_{\mathbf{X}} \quad (2)$$

The first channel packet $\mathbf{y}[1]$ is the source packet $\mathbf{x}[m+1]$ itself, the second channel packet $\mathbf{y}[2]$ is the source packet $\mathbf{x}[m+2]$ itself, the third and fourth channel packets $\mathbf{y}[3]$ and $\mathbf{y}[4]$ are different linear combinations of source packets $\mathbf{x}[m+1]$ and $\mathbf{x}[m+2]$, the fifth channel packet is the source packet $\mathbf{x}[m+3]$, and the sixth channel packet is a linear combination of packets $\mathbf{x}[m+1]$, $\mathbf{x}[m+2]$ and $\mathbf{x}[m+3]$. Note that by first channel packet, we mean the first channel packet after the ones that have already been created.

The difference between (1) and (2) just described is that in (2) we allow sending original source packets, in addition to the linear combinations starting from the first packet not known to be decoded. That is, we may send $\mathbf{x}[j]$ instead of a channel packet with LIS packet $\mathbf{x}[j]$, in cases where it does not affect sequential decodability to do so.

One such case is if $\mathbf{x}[j]$ is a newly consumed source packet, hence no channel packet with LIS packet $\mathbf{x}[j]$ or later has been sent.

Another case is if $\mathbf{x}[j]$ is the *first* source packet known to be not decodable. A full determination of this fact requires feedback and probability calculations (to be discussed later), but a simple example is if in the above example, all but $\mathbf{y}[1]$ and $\mathbf{y}[5]$ are lost, in which case resending $\mathbf{x}[m+2]$ is allowed. In general, it is necessary (but insufficient) for the first transmission of the source packet $\mathbf{x}[j]$ to be lost for $\mathbf{x}[j]$ to be undecodable.

In summary, the encoder sends two types of packets onto the channel: the original source packet or a random linear combination of all consumed and not-known-to-be-decoded packets. We call the latter check packets from now on.

2.2. Decoding

Due to erasures, the receiver gets only a subset of the channel packets sent. Let the last sequentially decodable source packet at the decoder be $\mathbf{x}[m]$, that is, the decoder can decode all $\mathbf{x}[1], ..., \mathbf{x}[m]$. A channel packet $\mathbf{y}[k]$ is still useful if its LIS packet $n(\mathbf{y}[k])$ has not been decoded. The decoder always keeps its list of received, still useful channel packets sorted ascending by their LIS packet index. Suppose $\mathbf{y}[r_1], ..., \mathbf{y}[r_s], ..., \mathbf{y}[r_R]$, $(n(\mathbf{y}[r_1]) \leq \cdots \leq n(\mathbf{y}[r_R]))$ are the received, still useful channel packets.

Every time a new network packet is received, the decoder makes an attempt to decode as many source packets as possible. For each received, still useful channel packet $\mathbf{y}[k]$ that is a check packet, we can remove all already decoded source packets from its linear combination as

$$\mathbf{y}'[k] = \mathbf{y}[k] - f_{k,1}\mathbf{x}[1] - \cdots - f_{k,m}\mathbf{x}[m]. \quad (3)$$

Then, we can attempt to decode using $\mathbf{y}'[r_1], ..., \mathbf{y}'[r_s]$, by looking at the effective generator matrix at the decoder, $\hat{\mathbf{G}}_s$, formed by taking rows of the original generator matrix $\mathbf{G}$ corresponding to a subset of the received, still useful channel

packets, and the columns $m+1$ to $n(\mathbf{y}'[r_s])$. We find the smallest s such that

$$\begin{bmatrix} \mathbf{y}'[r_1] \\ \vdots \\ \mathbf{y}'[r_s] \end{bmatrix} = \hat{\mathbf{G}}_s \begin{bmatrix} \mathbf{x}[m+1] \\ \vdots \\ \mathbf{x}[n(\mathbf{y}'[r_s])] \end{bmatrix} \quad (4)$$

is invertible, i.e. $\text{rank}(\hat{\mathbf{G}}_s) = n(\mathbf{y}'[r_s]) - m$.

A necessary condition is $s \geq n(\mathbf{y}'[r_s]) - m$ since $\text{rank}(\hat{\mathbf{G}}_s) \leq s$. Under most conditions, the smallest invertible $\hat{\mathbf{G}}_s$ is square and will result in $s = n(\mathbf{y}'[r_s]) - m$. So once decoded, we can udpate $m := n(\mathbf{y}'[r_s]) = m + s$, and repeat the process.

We see that (4) is the key decoding step of the receiver, where decoding a train of source packets following the last decoded source packet $\mathbf{x}[m]$ is attempted. If the rank of sub-generator matrix $\hat{\mathbf{G}}_s$ is to be no greater than s, then the s channel packets used to decode the source packets must not have linear combinations with any source packets after $\mathbf{x}[m+s]$. The only case that this still leads to a non-invertible $\hat{\mathbf{G}}_s$ is if some check packets are linearly dependent on others.

This leads to the following slight modification to the decoder. Whenever a channel packet is received with LIS index $m+s$, we examine if it is linearly dependent with any of the other pending (unused) channel packets with LIS up to $m+s$ in the natural course of rank determination for $\hat{\mathbf{G}}_s$ as above. If the packet is found to not increase the rank of $\hat{\mathbf{G}}_s$, it is treated as a lost channel packet and discarded. [1]

Thus, we can assume no linear dependencies in check packets, so the decoder can decide whether the next s undecoded source packets are decodable from the channel packets $\mathbf{y}'[r_1], ..., \mathbf{y}'[r_s]$ by using $s = n(\mathbf{y}'[r_s]) - m$ as a test rather than a consequential result. Whenever this test condition is met, source packets $\mathbf{x}[m+1], ..., \mathbf{x}[m+s]$ are decoded.

The decoder described is optimal for the code, in that it sequentially decodes each $\mathbf{x}[j]$ at the earliest possible time with respect to the received channel packets.

Since we have shown how both the encoder and decoder remove decoded packets from their decisions, for the remainder of the discussion in this paper, the reader can assume, without the loss of generality, that $m = 0$.

3. HYBRID FEC-ARQ PROTOCOL

Now we embed the packet streaming code of the previous section into a hybrid FEC-ARQ protocol. The goal of this protocol is to minimize expected sequential decodability delay for all of the packets as this is a measure that directly correlates with the interactive responsiveness of the applications being considered. Intuitively, this happens if the probability of sequential decodability is maximized as quickly as possible. However, if we only maximize the sequential decodability by sending check packets, the pending source packets may not get a slot for transmission. Also, whenever feedback indicates a source packet cannot be decoded, sending a new source packet or even another check packet may cause longer decoding delay. An important aspect of the protocol is thus to combine feedback with estimates of network conditions to balance sending check packets with sending (or resending, as we will see) source packets.

3.1. Feedback

Our proposed hybrid FEC-ARQ protocol can work with a number of feedback options but in this paper we assume a verbose feedback regime where the receiver acknowledges both received and lost packets. That is, each time a channel packet is received, the receiver acknowledges positively and each time a loss is detected, the receiver acknowledges negatively. [2]

Let $\mathcal{P}, \mathcal{N}, \mathcal{U}$ respectively be the positively, negatively, and not yet acknowledged channel packets. Let $\mathcal{F}$ be all consumed yet undecoded source packets. Let $\mathcal{Q}$ be the unconsumed source packets waiting in the source queue.

Given this feedback and the random erasure channel model, the encoder can precisely update itself on the probability that each source packet can be sequentially decoded from all the sent packets, deterministically using $\mathcal{P}$ and $\mathcal{N}$, and probabilistically using channel packets still in flight in the round-trip pipeline $\mathcal{U}$.

3.2. Probability Computation

Given the decoding logic in Sec 2.2, we can derive the probability of sequential decodability of source packets. Enumerating through the packet loss patterns in $\mathcal{U}$ works for arbitrary coding structures, but requires an infeasible complexity of $\mathcal{O}(2^{|\mathcal{U}|})$.

Due to the special structure of the packet streaming code of this paper, we next give a sequential decodability probability calculation with $\mathcal{O}(L^2)$ complexity, where L is the number of source packets which are not known to be sequentially decodable.

Let $\hat{\mathbf{G}}_l$ be the sub-generator matrix formed by taking columns $m+1$ to l of the rows of generator matrix $\mathbf{G}$ corresponding to received channel packets with LIS packet index $n(\mathbf{y}[k]) \leq l$, assuming that linearly dependent channel packets have already been thrown away. We know that we can sequentially decode source packets up to l if $\text{rank}(\hat{\mathbf{G}}_l) = l$.

Let R_{ol} be the number of original packets received with LIS packet index equal to l, and let R_{cl} be the number of

[1] For a sufficiently large field F, this dependency does not happen frequently, and furthermore, the encoder is capable of choosing linear combinations carefully to avoid this, by using the same rank determination procedure.

[2] Practically, this can be achieved by the use of channel packet sequence numbers and indicating the received packets, from which the not-received packets are inferred. This has the added benefit of giving the sender rich information on varying network conditions.

check packets received with LIS packet index equal to l. Let R_l be the rank of the matrix formed by taking rows of the generator matrix corresponding to these packets with LIS packet index exactly equal to l. Assuming that there are no dependencies in the random linear code, we can write the rank of this matrix as $R_l = \min(\min(R_{ol}, 1) + R_{cl}, l)$. Let $V_l = \text{rank}(\hat{\mathbf{G}}_l)$. Then, we can precisely write the rank of $\hat{\mathbf{G}}_l$ using recursion as

$$V_l = \min(V_{l-1} + R_l, l), \tag{5}$$

if there are no dependencies in the random linear code.

Let $S_l = S_{cl} + \min(S_{ol}, 1)$ be the effective number of sent packets with LIS packet index equal to l. Note that an original packet with LIS packet index l is grouped into a single packet via the term $\min(S_{ol}, 1)$. Also assume that the channel packets sent are linearly independent. Let T_l be the maximum value that R_l can take on and is given by $T_l = \min(S_l, l)$.

We know that we can sequentially decode up to l if $V_l = l$, but we can also sequentially decode up to l if $V_j = j$ for some $j > l$ since being able to sequentially decode up to a particular packet implies being able to sequentially decode up to all previous packets by definition. Let p_l be the probability that we can sequentially decode up to and including l. Let I_l be an indicator variable that we can sequentially decode up to l. Then, by conditioning on the rank of $\hat{\mathbf{G}}_l$, we get

$$p_l = \sum_{v=0}^{l} \mathbf{P}(I_l = 1 | V_l = v)\mathbf{P}(V_l = v), \tag{6}$$

where $\mathbf{P}(\cdot)$ is the probability of the given event.

To compute $\mathbf{P}(V_l = v)$, we can recurse on (5) as

$$\mathbf{P}(V_l = v) = \sum_{i=\max(0, v-T_l)}^{\min(l-1, v)} \mathbf{P}(V_{l-1} = i)\mathbf{P}(R_l = v - i) \tag{7}$$

for $v = 0, 1, \ldots, l-1$, and

$$\mathbf{P}(V_l = l) = \sum_{i=l-T_l}^{l-1} \mathbf{P}(V_{l-1} = i)\mathbf{P}(R_l \geq l - i) \tag{8}$$

for $v = l$. The recursion can start with $\mathbf{P}(V_0 = 0) = 1$.

To compute $\mathbf{P}(I_l = 1 | V_l = v)$, we can also recurse on (5) as

$$\begin{aligned}&\mathbf{P}(I_l = 1 | V_l = v) = \\ &\sum_{i=v}^{T_l+v} \mathbf{P}(I_l = 1 | V_l = v, V_{l+1} = i)\mathbf{P}(V_{l+1} = i) = \\ &\sum_{i=v}^{T_l+v} \mathbf{P}(I_{l+1} = 1 | V_{l+1} = \min(i, l+1))\mathbf{P}(R_l = i - v),\end{aligned} \tag{9}$$

for $v = 0, 1, \ldots, l-1$, and

$$\mathbf{P}(I_l = 1 | V_l = l) = 1. \tag{10}$$

Here we have also used the fact that if $V_l < l$ then, the only way for $I_l = 1$ is for $I_{l+1} = 1$. To start the recursion, we can use $\mathbf{P}(I_l = 1 | V_l = v) = 0$, for $v \neq l$ for $l = L$ which is currently the last packet in set $\mathcal{F}$.

To compute $\mathbf{P}(R_l = r)$ which is needed in the computation, let $U_{l,q}$ be the indicator variable whether channel packet q with LIS packet index l has been received or not, $q = 1, \ldots, S_l$. $\mathbf{P}(U_{l,q} = 1)$ is simply given by the value of ϵ for the corresponding channel packet, where ϵ is the loss rate of the channel when the packet was sent. For original packets which have been sent t times (which are grouped into a single channel packet), the effective packet loss rate on them is ϵ^t instead of ϵ. (Similarly, if packet loss rate is time-varying, the effective loss rate is $\prod_{i \in \mathcal{R}} \epsilon_i$ where $\mathcal{R}$ are the indices of channel packets sent for the same original packet). For channel packets which have been positively acknowledged through feedback, $\mathbf{P}(U_{l,q} = 1) = 1$, and for those which have been negatively acknowledged $\mathbf{P}(U_{l,q} = 1) = 0$.

Let $W_{l,q} = \min(W_{l,q-1} + U_{l,q}, l)$, be the cumulative sum of linearly independent channel packets with LIS packet index l using the first q such coded packets. This can take on the values $W_{l,q} = 0, \ldots, \min(l, q)$, and can be computed using a simple recursion as

$$\begin{aligned}&\mathbf{P}(W_{l,q} = w) = \\ &\sum_{i=\max(0,w-1)}^{\min(l,q-1,w)} \mathbf{P}(W_{l,q-1} = i)\mathbf{P}(U_{l,q} = w - i)\end{aligned} \tag{11}$$

for $w = 0, \ldots, \min(l-1, q)$, and

$$\begin{aligned}&\mathbf{P}(W_{l,q} = w) = \\ &\sum_{i=l-1}^{\min(l,q-1)} \mathbf{P}(W_{l,q-1} = i)\mathbf{P}(U_{l,q} \geq l - i)\end{aligned} \tag{12}$$

for $w = l$. To start the recursion, we can use $\mathbf{P}(W_{l,0} = 0) = 1$.

We can see that if the number of packets sent with LIS packet index equal to a particular m is $\mathcal{O}(1)$, then the computation needed to evaluate the given summations is also $\mathcal{O}(1)$ since T_l is $\mathcal{O}(1)$. The number of such summations to compute is $\mathcal{O}(L^2)$ and thus that is the total complexity.

3.3. Transmission Strategy

As we have seen from the coding structure being used in Section 2 as well as the decodability calculations in the current section, we obtain the following transmission policies commonly seen in FEC-ARQ schemes as among the transmission policies supported by this coding structure:

- Policy `S`: Sending a new source packet without coding.
- Policy `C`: Sending a check packet of only the undecoded packets up to some $\mathbf{x}[j]$.

- Policy R: Resending an already sent packet for the t-th time.

Based upon the probability of sequential decodability computation, we attempt to make transmission decisions so that the $\sum p_j$ is maximized over the set $j \in \mathcal{F} \cup \mathcal{Q}$ but at the same time do not wish to delay the sending of new original packets unnecessarily. It is clear that sending a new original packet from the set $\mathcal{Q}$ will result in p_j staying the same for all packets in set $\mathcal{F}$. However, it will result in p_j becoming non-zero for the first packet in set $\mathcal{Q}$. On the other hand, by sending a check packet, p_j will increase for packets in set $\mathcal{F}$, but all packet is $\mathcal{Q}$ will by delayed by one transmission opportunity. This delay will be incurred unnecessarily with probability p_l, where l is the LIS packet index of the check packet (since with probability p_l, we can already sequentially decode up to l). A simple way to decide on the policy is to pick the one which maximizes

$$\sum_{j \in \mathcal{F}} (p_j[k] - p_j[k-1]) - p_J[k-1]|\mathcal{Q}|, \qquad (13)$$

where $p_j[k]$ is the sequential decodability probability of source packet k after sending the new channel packet k, and $p_j[k-1]$ is probability before sending the packet. $|\mathcal{Q}|$ is the cardinality (number of elements) in the set. J is the LIS packet index of the newly sent channel packet, and the sets $\mathcal{F}$ and $\mathcal{Q}$ are the sets after sending the channel packet. The first term is an improvement in probability of sequential decodability and the second term is a penalty for unnecessarily delaying future packets.

For the case when the loss rate is relatively low in the channel, the improvement in the probability of sequential decodability is usually small, except for the case when we get a negative acknowledgment on an original packet. Otherwise, since the improvement is small, so long as $|\mathcal{Q}| \neq 0$, the penalty of the second term is larger than benefit, and thus the best solution is one where $p_J[k-1] = 0$, that is it is best to pick a packet which is definitely useful. This means that the only time a check packet is sent is if $\mathcal{Q} = \varnothing$. Also, in general the benefits of a check packet are larger for the later packets in $\mathcal{F}$ and also $p_J[k-1]$ is smaller for larger J, and thus if a check packet is sent, for a wide range of conditions it makes sense to have $J = |\mathcal{F}|$, that is all the packets in $\mathcal{F}$ are included in the check packet. These simplifications lead to the following simple protocol which works well under a wide range of source rate and network conditions for our simple system model.[3] It follows the precedence described in Table 2 when deciding among the transmission policies.

[3]Alternatively, the sender's choice of policy at each transmission opportunity can be based on complex objective functions of the decodability probabilities and network statistics, which may be beneficial in more complex network scenarios.

Table 2: Proposed Hybrid FEC-ARQ Protocol

1. If $\mathcal{F} \neq \varnothing$, and if $\mathbf{x}[j] \in \mathcal{F}$ is determined to be the first source packet with probability of being sequentially decodable close to zero [†], then send $\mathbf{x}[j]$ using Policy R. This means retransmission has priority for source packets almost certainly blocking the stream, and corresponds to an "ARQ mode".

2. If $\mathcal{Q} \neq \varnothing$, send the head-of-line source packet from $\mathcal{Q}$ using Policy S, that is, send a new source packet if $\mathcal{Q}$ is not empty. This corresponds to a "systematic transmission mode".

3. If $\mathcal{Q} = \varnothing$, $\mathcal{F} \neq \varnothing$, but [†] is not satisfied for any $\mathbf{x}[j] \in \mathcal{F}$, send using Policy C a redundancy packet with LIS packet being the last packet entering $\mathcal{F}$, that is, send a check packet using all available packets in $\mathcal{F}$. This corresponds to an "FEC mode".

4. If $\mathcal{Q} = \varnothing, \mathcal{F} = \varnothing$, send nothing.

4. EXPERIMENTAL RESULTS

4.1. Comparison Setup

In this section, we compare the proposed hybrid FEC-ARQ protocol against a number of benchmarks:

1. Pure ARQ: This is the algorithm used in TCP, in which the sender only resends source packets known to be lost. In this experiment, we let this protocol resend old packets when there are no new source packets waiting to be sent, which can only be beneficial. (This means no transmission opportunity is wasted for fair comparison with the proposed hybrid FEC-ARQ protocol.)

2. Block FEC-ARQ: This is a fixed block-length systematic $(K + R, K)$ MDS (Maximum Distance Separable) FEC code with fallback to retransmission. K is the number of source packets, and R is the number of check packets. We choose the amount of redundancy based on the packet loss rate. For packet loss rate ϵ, $R = \lceil \frac{\epsilon K}{1-\epsilon} \rceil$. In the experiments, we choose two block lengths: $K = 19$ (short block, SB FEC-ARQ) and $K = 38$ (long block, LB FEC-ARQ). Sender retransmits source packets if not enough packets arrived to decode a block. We also resend old packets when there are no new source packets waiting to be sent for the same aforementioned reason.

3. CO FEC-ARQ: It is the same as the hybrid FEC-ARQ protocol, except that when source packets are known to be undecodable, this protocol always sends a check

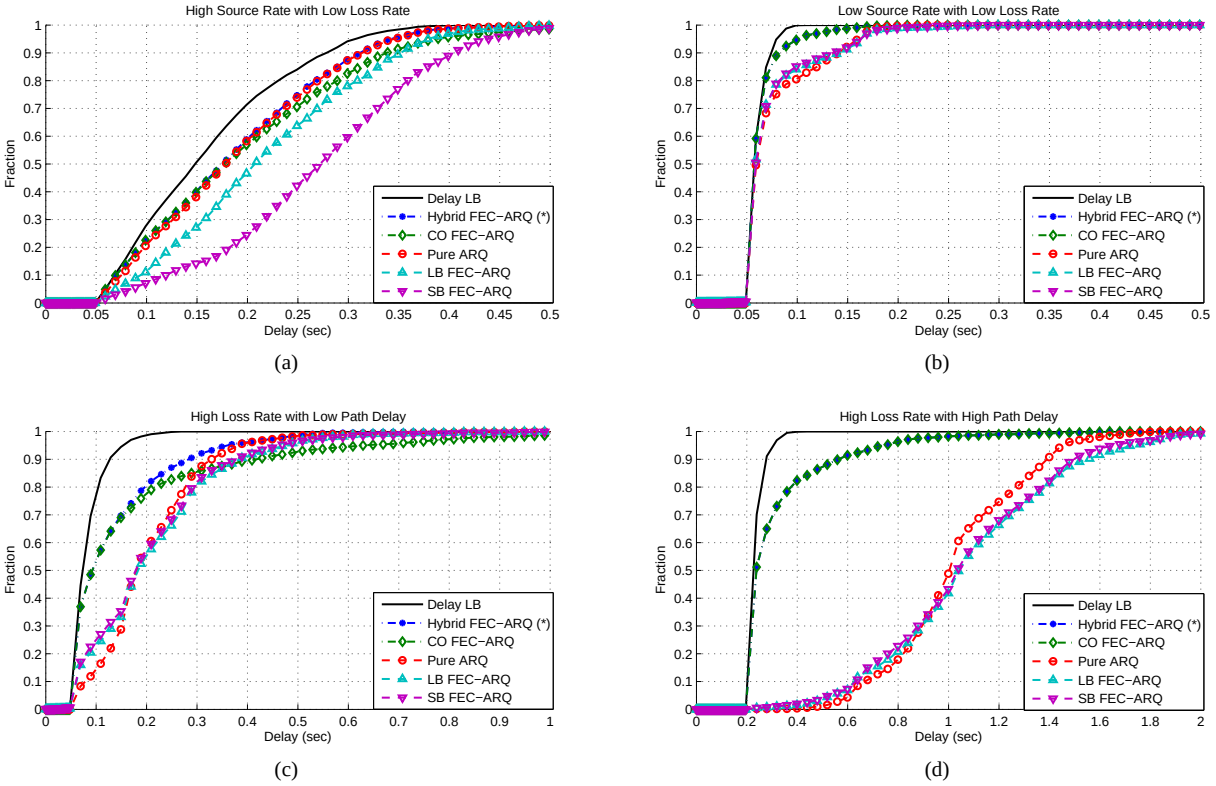

Fig. 2: Comparison of several FEC-ARQ schemes and Pure ARQ scheme against the Delay LB, under different network conditions. (*) marks the proposed protocol. $B = 1250$ in all cases. (a) $S = 940, T = 1000, \epsilon = 0.05, d = 50$; (b) $S = 500, T = 1000, \epsilon = 0.05, d = 50$; (c) $S = 500, T = 1000, \epsilon = 0.30, d = 50$; (d) $S = 500, T = 1000, \epsilon = 0.30, d = 200$.

packet. That is, all cases where `R` is used in hybrid FEC-ARQ are replaced with Policy `C`. (N.B. This is not an existing scheme, but is included in the comparison to better understand the behavior of the proposed hybrid FEC-ARQ protocol.)

4.2. Delay Lower Bound

We also establish a strict lower bound on the delay of every source packet in a stream. In order to sequentially decode $\mathbf{x}[j]$, at least j channel packets must be received for *any* code that uses source and channel packets of the same size. If $t_i(m)$ is the time at which the m-th non-erased transmission sent after the arrival of $\mathbf{x}[i]$ is received, then the earliest successful decoding of $\mathbf{x}[j]$ can be no earlier than

$$\max_{1 \le i \le j} t_i(j - i + 1)$$

which in turn gives the lower bound on the delay for $\mathbf{x}[j]$, for a particular realization of source and channel dynamics.

This lower bound is attainable by a strategy of retransmitting each packet lost in the channel at the very next transmission opportunity. This is clearly the optimal strategy, provided perfect knowledge of packet loss is obtained by the sender before the next transmission opportunity. In practice, there is non-trivial feedback delay, and this is not a tight bound under such circumstances. However, it serves as an excellent benchmark to compare against all FEC-ARQ schemes.

4.3. Results and Discussion

The comparison result is shown in Figure 2. The following experimental setup is used. Source and packets are both B bytes each. The source rate is S kbps. The network bandwidth is T kbps, and packets are lost i.i.d. at loss rate of ϵ. The path delay between the sender and receiver is d ms. Transmission opportunities are assumed to be regularly spaced at every $8B/1000T$ seconds, and source arrival is modeled as a Poisson process, with S/T new packets arriving at the sender on average every transmission opportunity.

We test the performance of the several protocols under four different network conditions in Figure 2: (a) low packet loss rate (5%) and minor redundant channel capacity (1%); (b) low packet loss rate (5%) and abundant redundant channel capacity (45%), (c) high packet loss rate (30%) and medium redundant channel capacity (20%), and (d) same as (c) but

with long path delay of 200 ms.

We subject all schemes to the same randomly generated realization of source arrival and channel loss patterns and compare the cumulative distribution function (CDF) of the delay experienced by the source packets in the sequence. For example, in Figure 2(c), the proposed hybrid FEC-ARQ protocol delivers, and the decoder sequentially decodes, 80% of the packets in the stream within 0.2 seconds of their insertion at the source.

Comparing the proposed hybrid FEC-ARQ protocol with other benchmark protocols, we notice that the proposed protocol has a superior delay performance.

Figure 2(a) shows a situation where there is little excess network bandwidth to support FEC. In such a scenario, block FEC-ARQ (either SB FEC-ARQ or LB FEC-ARQ) or CO FEC-ARQ leads to poor delay performance, as they may imprudently waste the very few extra transmission opportunities on check packets that happen not to be needed. We notice that Pure ARQ performs well in this scenario, as is well known. Nevertheless, hybrid FEC-ARQ still slightly outperforms it.[4]

In Figure 2(b), where there is abundant excess network bandwidth, FEC shines. With hybrid FEC-ARQ, the algorithm can actively insert many check packets into the network, and cause the delay performance of hybrid FEC-ARQ to be close to the lower bound. Block-based schemes (SB FEC-ARQ or LB FEC-ARQ) show some benefit, but the improvement over Pure ARQ is not nearly as dramatic.

When packet loss rate becomes significant, as in Figure 2(c), hybrid FEC-ARQ gains a superior advantage in delay performance compared with either Pure ARQ or block FEC-ARQ. CO FEC-ARQ (protocol without policy `R`) performs close to hybrid FEC-ARQ about half of the time. However, at 80-percentile delay performance and beyond, we notice that CO FEC-ARQ becomes inferior to the schemes that use some form of retransmission (hybrid FEC-ARQ, block FEC-ARQ and Pure ARQ). It is evident that activating Policy `R` is very beneficial in heavy loss cases, and ensures that transmission can recover from network conditions beyond the protection capability of FEC without adversely blocking the future packets pending delivery. We notice that block-based schemes (SB FEC-ARQ or LB FEC-ARQ) exhibit bifurcating behavior compared with Pure ARQ, performing better in some cases but worse in others, which may be typical of these simple switching FEC-ARQ strategies.

In the case of high packet loss rate coupled with high path delay, as in Figure 2(d), hybrid FEC-ARQ demonstrates significant performance advantage over Pure ARQ or block FEC-ARQ. We notice that hybrid FEC-ARQ improves the 80-percentile transmission delay by almost a factor of 3 (from 1.3 seconds to 0.4 seconds). The hybrid FEC-ARQ protocol is thus capable of noticably affecting application interactivity in the most difficult network conditions.

[4]It can be shown that hybrid FEC-ARQ never performs worse.

5. CONCLUSION

In this paper, we proposed a simple hybrid FEC-ARQ protocol for delivering sequential data streams losslessly and with low delay. This protocol, based on a packet streaming code well suited to sequential decoding, makes significant improvement over retransmission strategies such as used by TCP and outperforms FEC-ARQ schemes based on block codes. We have benchmarked the performance against a strict lower bound on delay performance and shown the protocol to perform well relative to it. Because the decodability probabilities can be calculated with low complexity, this protocol can easily be modified for soft decisions or be extended with other network protocol features such as congestion awareness, in future work.

6. REFERENCES

[1] F. Vacirca, A. De Vendictis, and A. Baiocchi, "Optimal design of hybrid FEC/ARQ schemes for TCP over wireless links with Rayleigh fading," *IEEE Transactions on Mobile Computing*, vol. 5, no. 4, pp. 289–302, 2006.

[2] R. El Azouzi, T. Peyre, and A. Benslimane, "Optimal design of hybrid FEC/ARQ schemes for real-time applications in wireless networks," in *International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems*, 2006, pp. 11–18.

[3] R. Puri, K. Ramchandran, and A. Ortega, "Joint source channel coding with hybrid ARQ/FEC for robust video transmission," in *IEEE Multimedia Signal Processing Workshop*, dec 1998.

[4] P. Chou, A. Mohr, A. Wang, and S. Mehrotra, "FEC and pseudo-ARQ for receiver-driven layered multicast of audio and video," in *IEEE Data Compression Conference (DCC)*, 2000.

[5] J. Hagenauer, "Rate-compatible punctured convolutional codes (RCPC codes) and their applications," *IEEE Transactions on Communications*, vol. 36, no. 4, pp. 389–400, Apr. 1988.

[6] L. Chen, X. Shi, S. Yan, Z. Wu, W. Zhang, and Y. Guan, "A novel scheme for type-II hybrid ARQ protocols using LDPC codes," in *IEEE Wireless Communications and Networking Conference*, Mar. 2007, pp. 682–686.

[7] H. Lou and J. Garcia-Frias, "Rate-compatible low-density generator matrix codes," *IEEE Transactions on Communications*, vol. 56, no. 3, pp. 321–324, Mar. 2008.

[8] E. Martinian, *Dynamic Information and Constraints in Source and Channel Coding*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA 02139, Sept. 2004.