






Low-Latency Network-Adaptive Error Control for Interactive Streaming

Salma Emara , *Member, IEEE*, Silas L. Fong , *Member, IEEE*, Baochun Li , *Fellow, IEEE*,
Ashish Khisti , *Member, IEEE*, Wai-Tian Tan, *Senior Member, IEEE*, Xiaoqing Zhu , *Senior Member, IEEE*,
and John Apostolopoulos, *Fellow, IEEE*

Abstract—We introduce a novel network-adaptive algorithm that is suitable for alleviating network packet losses for low-latency interactive communications between a source and a destination. Our network-adaptive algorithm estimates in real-time the best parameters of a recently proposed streaming code that uses forward error correction (FEC) to correct both arbitrary and burst losses, which cause a crackling noise and undesirable jitters, respectively in audio. In particular, the destination estimates appropriate coding parameters based on its observed packet loss pattern and sends them back to the source for updating the underlying code. Besides, a new explicit construction of practical low-latency streaming codes that achieve the optimal tradeoff between the capability of correcting arbitrary losses and the capability of correcting burst losses is used. Simulation evaluations based on statistical losses and real-world packet loss traces reveal the following: (i) Our proposed network-adaptive algorithm combined with our optimal streaming codes can achieve significantly higher performance compared to uncoded and non-adaptive FEC schemes over UDP (User Datagram Protocol); (ii) Our explicit streaming codes can significantly outperform traditional MDS (maximum-distance separable) streaming schemes when they are used along with our network-adaptive algorithm. In addition, we study different factors that can affect the performance of our network-adaptive algorithm.

Index Terms—Forward error correction, streaming codes, teleconferencing.

I. INTRODUCTION

SEVERAL current and emerging applications over the Internet demand real-time interactive streaming, such as high-definition video conferencing, augmented/virtual reality and

online gaming. At the core of these low-latency applications is the need to deliver packets reliably and with low-latency to provide the user with the expected functionality and responsiveness. Given that the Internet is a packet-switched network, where reliable packet delivery is not guaranteed, the urge for effective methods to protect live video communications over the Internet has never been greater.

At the network layer, packet erasures or losses over the Internet are inevitable. Packets are lost due to unreliable wireless links or congestion at network bottlenecks. To recover missing packets introduced by the network layer, two basic methods at the transport layer have been widely implemented: Automatic repeat request (ARQ) and forward error correction (FEC).

ARQ is a retransmission-based scheme, where the transmitter retransmits a packet based on feedback from receiver. If the communication is between distant users, the extra round-trip delay incurred by the retransmission may be intolerable for real-time streaming applications. Correcting an erasure using ARQ yields a 3-way delay (forward + backward + forward).

On the other hand, FEC does not require any retransmission. Instead FEC schemes increase the correlation among the transmitted symbols by sending redundant information. Using this redundant information, erased packets can be reconstructed using the surviving or correctly received packets. Low-density parity-check (LDPC) codes [1], [2] and digital fountain codes [3], [4] are two FEC schemes that are currently used in the DVB-S2 [5] and DVB-IPTV [6] standards for non-interactive streaming applications. Besides, there is on-going research on improving these codes for applications such as scalable video transmission [7]. These codes operate over long block lengths, typically a few thousand symbols.¹

Noteworthy caveats of LDPC and fountain codes involve (i) the need to wait for the arrival of longer block lengths and (ii) larger computation time to encode and decode these long block lengths. Thus, LDPC and fountain codes are preferable for applications in which the delay constraints are not stringent. However, when the delay constraints are strict and block lengths are short (e.g., a few hundred symbols) – like in low-latency streaming applications, LDPC and digital fountain codes become unfit.

¹A symbol is an element in the finite field. For example, codes over $GF(2^m)$ can have each symbol taking one of 2^m values, and is represented as an m -bit symbol. A code over $GF(256=2^8)$ has a symbol of 1 byte.

Manuscript received June 22, 2020; revised January 26, 2021 and March 20, 2021; accepted March 26, 2021. Date of publication March 31, 2021; date of current version March 29, 2022. This article was presented in part at 27th ACM International Conference on Multimedia, Nice, France, 2019. This work was supported in part by the Cisco Systems Inc., as well as a NSERC Collaborative Research and Development (CRD) grant. All work was carried out when Xiaoqing Zhu was with Cisco. The associate editor coordinating the review of this manuscript and approving it for publication was Professor Houqiang Li. (Corresponding author: Salma Emara.)

Silas L. Fong is with the Qualcomm Flarion Technologies, Bridgewater Township 08807 NJ USA (e-mail: silas.fong@ieee.org).

Salma Emara, Baochun Li, and Ashish Khisti are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto M5S 3G4 ON, Canada (e-mail: salma@ece.utoronto.ca; bli@ece.utoronto.edu; akhisti@ece.utoronto.ca).

Wai-Tian Tan and John Apostolopoulos are with the Cisco Systems, San José 95134 CA USA (e-mail: dtan2@cisco.com; johnapos@cisco.com).

Xiaoqing Zhu is with the Netflix Inc., 100 Winchester Cir, Los Gatos CA 95032 USA (e-mail: xzhu@netflix.com).

Color versions of one or more figures in this article are available at <https://doi.org/10.1109/TMM.2021.3070134>.

Digital Object Identifier 10.1109/TMM.2021.3070134

Apart from LDPC and digital fountain codes, for interactive applications, Reed Solomon (RS) and Random Linear Convolutional (RLC) codes can be used to recover losses over shorter block lengths as in [8]–[10]. However, the pattern of erasures (i.e., bursty or arbitrary erasures) play a role in the efficiency of a coding scheme. For example, to recover burst losses, longer RS codes are required resulting in higher delay. Meanwhile, RLC codes can have the same correcting capacity of RS codes, but with lower delay to recover bursty erasures than RS codes [11]. To better illustrate the effect of different loss patterns, bursty losses – usually induced by network congestion and result in undesirable jitters/pauses in audio, and arbitrary erasures – generated by unreliable wireless links – lead to crackling noise. Therefore, optimal and fast recovery for all loss types will influence the user experience.

Interestingly, having a block or convolutional structure by itself does not yield a minimum recovery delay. Therefore to achieve lower recovery delay, we require streaming codes (i.e., low-latency FEC schemes) that reconstruct earlier lost packets without waiting to correct all losses. Efforts in designing low-latency FEC schemes that operate over short block lengths to improve interactive communication include Raptor codes [12], RaptorQ codes [13], randomized linear codes [14] and others [4], [15], [16], and many works have used these codes in video streaming [17], [18].

Surely, the employment of FEC schemes to recover lost voice packets has led to the success of Skype [19]. In addition, adaptive hybrid NACK/FEC has been deployed in WebRTC to acquire a better trade-off between temporal quality, spatial video quality and end-to-end delay [16]. However, still FEC is not optimized for low-latency to recover burst losses.

If we follow existing FEC technologies (e.g., WebRTC [16], Skype [19], Raptor codes [12], RaptorQ codes [13] and randomized linear codes [14]) and choose the parity frames based on coding over the past multimedia frames using maximum-distance separable (MDS) codes, the resulting FEC streaming code is optimal for correcting arbitrary losses subject to the decoding delay d_d , but not for bursty losses. This is the research gap that we target, i.e., deficiency in an *optimal FEC streaming code* that can correct *both arbitrary and bursty erasures* subject to decoding delay d_d .

A. Main Contributions

Extensive research has been conducted to study the abilities of streaming codes in recovering bursty and arbitrary (isolated) losses [20], [21]. The authors in [20] and [21] have proposed a high-complexity construction of a class of FEC streaming codes that has the following properties:

- Correct both arbitrary and burst erasures
- Achieve the optimal trade-off between correcting arbitrary and burst erasures under a given maximum delay constraint

This motivates us to *design* a low-latency error control scheme based on low-complexity FEC streaming codes to satisfy Properties (a) and (b) and *implement* the design for employment in

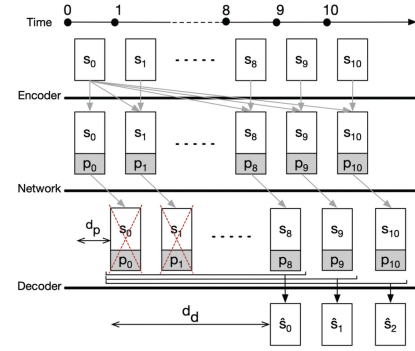


Fig. 1. The general framework of FEC to recover dropped packets.

real-world networks. Our real-time error control design has the following features:

- A new explicit construction of low-latency streaming codes achieving a delay-constraint optimal trade-off between the capability of correcting arbitrary erasures and the capability of correcting burst erasures.
- A network-adaptive algorithm that updates the parameters of our constructed low-latency streaming codes in real-time to adapt to varying network conditions.

To evaluate our network-adaptive FEC streaming design, we conduct extensive simulations and real-world experiments. The results of our simulations and real-world experiments show that our network-adaptive scheme perform remarkably better compared to uncoded and non-adaptive FEC schemes over UDP in terms of reliability. We extend our comparisons to adaptive MDS streaming codes, and we show that our network-adaptive code outperforms MDS streaming codes in highlighted scenarios and performs a lot better in terms of latency. We also discuss factors that may affect the performance of our scheme.

Note that the maximum delay constraint that appeared in Property (b) and Property (i) is a suitable delay metric for interactive communication, because any packet recovered beyond a certain threshold would yield undesirable pauses. On the contrary, the average delay metric is more relevant for non-interactive streaming applications such as video streaming, which has been used in [22]–[25] to study the tradeoff between throughput and average delay for non-interactive streaming where no packet is ever discarded. Besides our primary results in [26], we:

- Add simulation results over Fritchman channel.
- Express our experimental results in terms of more comprehensive metrics including frame loss rate (FLR), coding rate and Perceptual Evaluation of Speech Quality (PESQ) score [27].
- Compare our explicit FEC streaming code along with our network-adaptive algorithm with state-of-the-art FEC streaming codes.

II. CONCEPT OF FEC STREAMING CODES

The general framework of FEC streaming code is illustrated in Fig. 1. The source/sender periodically generates a sequence of multimedia frames s_i , where i is the number of the frame. Each multimedia packet is concatenated with a parity packet p_i , and



Fig. 2. Illustration of burst and arbitrary drops, with packet drops marked by dark squares.

they are encapsulated in a frame which travels to the destination with a propagation delay d_p .

An alternative approach is to transmit parity information as separate packets. Our approach has the advantage of not creating more packets, which lowers overhead for transmission over Wi-Fi, and also simplifies analysis. In Wi-Fi networks, each packet requires a preamble (overhead) of 92 or 192 microseconds (depending on whether we use short or long preamble). For example, 100 B payload on a relatively “slow” speed of 150 Mbit/s takes only 5.7 microseconds. What that means is the air-time cost of transmitting two spaced out 100 B packet is twice as sending one 200 B packet.

In addition, our construction of optimal streaming codes creates parity packets that does not contain information of the current multimedia packet, hence separately transmitting multimedia and parity packets will not benefit from time diversity. Details on the construction of optimal streaming codes is given in Section III. We also show in Section VI-C that sending parity and multimedia packets separately in our case can degrade the performance of our algorithm.

The network packets may be dropped by the network. The drops may be in an arbitrary manner due to unreliable (wireless) links or in a bursty manner due to network congestion. The pattern of drops can be arbitrary or bursty in nature, as illustrated in Fig. 2. The destination aims to recover the multimedia packets sequentially subject to a decoding delay constraint d_d , where lost multimedia packets can be recovered with the help of subsequent parity packets. For example, if packets 0 and 1 are dropped as illustrated in Fig. 1, then the parity packets in packets 2 to 8 may help recover packet 0 with decoding delay of 8 packets, and the parity packets in packets 2 to 9 may help recover packet 1 with the same decoding delay.

We discuss earlier that if we follow existing technologies such as WebRTC [16] and Skype [19] in choosing the parity packets based on coding over the past multimedia packets using MDS codes, the resulting FEC streaming code is optimal for correcting arbitrary losses subject to the decoding delay d_d , but not bursty losses. In other words, to make it optimal in correcting bursty losses as well, we would require more time leading to an increase in decoding delay d_d . However, in order to achieve the optimal tradeoff between the capability of correcting arbitrary losses and correcting burst losses subject to a decoding delay constraint, we have to carefully choose the parity packets. The existence of such optimal parity frames has been recently proved in [20], [21].

III. EXPLICIT CONSTRUCTION OF OPTIMAL STREAMING CODES OVER GF(256)

For the sake of completeness, we outline the details of our first explicit construction of optimal streaming codes over GF(256)

TABLE I
SUMMARY OF KEY NOTATIONS USED IN THIS PAPER

Notation	Definition
T	decoding delay constraint
B	maximum run length of recoverable burst erasures
N	maximum number of recoverable arbitrary erasures
n	codeword size
k	raw data size
\mathbb{Z}_+	set of non-negative integers
\mathbb{F}^k	set of k -dimensional row vectors over finite field \mathbb{F}
L	channel uses over which we run the network-adaptive algorithm, in other words, duration of algorithm

when $T \leq 11$. Readers who are interested in greater detail may also refer to [26]. Uninterested readers may take the following result for granted and skip the remaining part of this section: “Let $\mathbb{F} = \text{GF}(256)$. For any $T \leq B \leq N \leq 1$, a $(n, k, T)_{\mathbb{F}}$ -code that corrects any (B, N) -erasure can be efficiently generated.”

The explicit construction leverages a standard periodic interleaving approach, which constructs streaming codes based on block codes. The following definitions are standard [20], [21]:

- 1) An $(n, k, T)_{\mathbb{F}}$ block code of length n with k data symbols² is said to correct any (B, N) -erasure sequence, if every data symbol can be recovered with a delay of no more than T symbols. This is possible as long as the number of erasures is at most N arbitrary erasures or all the erasures are consecutive, i.e., bursty, with run length at most B .
- 2) A $(n, k, T)_{\mathbb{F}}$ streaming code which encodes a continuous stream of length- k data packets into a continuous stream of length- n codewords is said to correct any (B, N) -erasure sequence, if every data packet can be recovered with a delay of no more than T packets. This is possible as long as the number of erasures within every sliding window of size $T + 1$ sees either at most N arbitrary erasures or size- B bursty erasures.
- 3) For any $T \geq B \geq N \geq 1$, (i) the (T, B, N) -capacity equals $C(T, B, N)$ as defined in (1), and (ii) a $(n, k, T)_{\mathbb{F}}$ -code that corrects any (B, N) -erasure sequence is said to be *optimal* if $\frac{k}{n} = C(T, B, N)$, where $k = T - N + 1$ and $n = k + B$.

$$C(T, B, N) \triangleq \frac{T - N + 1}{T - N + B + 1} \quad (1)$$

- 4) Given a $(n, k, T)_{\mathbb{F}}$ -block code which corrects any (B, N) -erasure sequence, we can construct a $(n, k, T)_{\mathbb{F}}$ -streaming code which corrects any (B, N) -erasure sequence.

Hence, the search for an explicit construction of optimal streaming codes over GF(256) reduces to the search for an explicit construction of optimal block codes over GF(256). Table I summarizes our key notations.

Structures of Parity Matrices of Optimal Block Codes: Next, we state specific structures of the parity matrices of optimal codes, where an m -row N -diagonal matrix is described as follows:

²The encoder takes k data symbols and adds parity to produce an n symbol codeword.

$$\mathbf{D}_N^{m \times (N+m)} \triangleq \begin{bmatrix} d_0^{(0)} & \cdots & d_{N-1}^{(0)} & 0 & \cdots & \cdots & 0 \\ 0 & d_0^{(1)} & \cdots & d_{N-1}^{(1)} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & d_0^{(m-1)} & \cdots & d_{N-1}^{(m-1)} & 0 \end{bmatrix}$$

where $\{d_\ell^{(i)} \mid 0 \leq i \leq m-1, 0 \leq \ell \leq N-1\}$ assume arbitrary values.

Lemma 1 ([20]): Fix any $T \geq B \geq N \geq 1$ and let $k \triangleq T - N + 1$ and $n \triangleq k + B$. If $k \geq B$ (i.e., $k/n \geq 1/2$), there exists a \mathbf{P} having the form

$$\begin{bmatrix} \mathbf{D}_N^{(B-N) \times B} \\ \mathbf{0}_{N \times (B-N)} \quad \mathbf{P}_{\text{right}} \\ \mathbf{V}^{(k-B) \times B} \end{bmatrix} \quad (2)$$

such that $\mathbf{G} = [\mathbf{I}_k \quad \mathbf{P}]$ is the generator matrix of a (n, k, T) -code that corrects any (B, N) -erasure sequences, where $\mathbf{D}_N^{m \times (N+m)}$ is an m -row N -diagonal matrix, $\mathbf{P}_{\text{right}}$ is an $N \times N$ matrix, and $\mathbf{V}^{(k-B) \times B}$ is a $(k-B) \times B$ parity matrix of a systematic MDS code. On the other hand, if $k < B$ (i.e., $k/n < 1/2$), there exists a \mathbf{P} having the form

$$\begin{bmatrix} \mathbf{P}_{\text{left}} & \mathbf{D}_N^{(B-N) \times k} \\ \mathbf{V}_{\text{left}}^{(k-B+N) \times (B-k)} & \mathbf{0} \quad \mathbf{V}_{\text{right}}^{(k-B+N) \times (k-B+N)} \end{bmatrix} \quad (3)$$

such that $\mathbf{G} = [\mathbf{I}_k \quad \mathbf{P}]$ is the generator matrix of a (n, k, T) -code that corrects any (B, N) -erasure sequence, where $\mathbf{D}_N^{(B-N) \times k}$ is a $(B-N)$ -row $(k-B+N)$ -diagonal matrix, $[\mathbf{V}_{\text{left}}^{(k-B+N) \times (B-k)} \quad \mathbf{V}_{\text{right}}^{(k-B+N) \times (k-B+N)}]$ constitutes the $(k-B+N) \times N$ parity matrix of a systematic MDS code, \mathbf{P}_{left} is a $(B-N) \times (B-k)$ matrix, and $\mathbf{0}$ is the $(k-B+N) \times (B-N)$ zero matrix.

Deterministic Construction of Parity Matrices of Optimal Codes over GF(256): The structures of \mathbf{P} in (2) and (3) motivate us to construct optimal codes over GF(256). Assume $\mathbb{F} = \text{GF}(256)$ in the rest of the paper. Suppose we are given a $k \times B$ matrix $\mathbf{V}^{k \times B}$ where $k \geq 1$ and $B \geq 1$, and let $n \triangleq k + B$. Then, to construct a parity matrix $\mathbf{P} \in \mathbb{F}^{k \times B}$ if $k \geq B$ or $k < B$, we replace every non-zero $(i, j)^{\text{th}}$ element of \mathbf{P} in (2) or (3) respectively with the $(i, j)^{\text{th}}$ element in $\mathbf{V}^{k \times B}$. Let $\mathcal{C}(\mathbf{V}^{k \times B})$ denote the $(n, k, T)_{\mathbb{F}}$ -block code with generator matrix \mathbf{G} as constructed earlier. If $\mathcal{C}(\mathbf{V}^{k \times B})$ corrects any (B, N) -erasure sequence and $k = T - N + 1$, then $\mathcal{C}(\mathbf{V}^{k \times B})$ is optimal by Definition 3.

In search of a suitable $\mathbf{V}^{k \times B}$, we define $\mathbf{V}_{\text{Cauchy}}^{(T-N+1) \times B} = [v_{ij}^{\text{Cauchy}}]_{\substack{0 \leq i \leq T-N \\ 0 \leq j \leq B-1}}$ to be a $(T-N+1) \times B$ Cauchy matrix over GF(256) where $v_{ij}^{\text{Cauchy}} \triangleq (i+j+k)^{-1}$. Similarly, define $\mathbf{V}_{\text{Vand}}^{(T-N+1) \times B} = [v_{ij}^{\text{Vand}}]_{\substack{0 \leq i \leq T-N \\ 0 \leq j \leq B-1}}$ to be a $(T-N+1) \times B$ Vandermonde matrix over GF(256) where $v_{ij}^{\text{Vand}} \triangleq 2^{i \times j}$. Using computer search, we obtain the following.

Proposition 2: Let $\mathbb{F} = \text{GF}(256)$. For any $1 \leq N \leq B \leq T \leq 11$, the $(n, k, T)_{\mathbb{F}}$ -block code $\mathcal{C}(\mathbf{V}_{\text{Cauchy}}^{(T-N+1) \times B})$ corrects any (B, N) -erasure if $(T, B, N) \notin \{(10, 8, 4), (11, 5, 4)\}$. In addition, $\mathcal{C}(\mathbf{V}_{\text{Vand}}^{(T-N+1) \times B})$ corrects any (B, N) -erasure if $(T, B, N) \in \{(10, 8, 4), (11, 5, 4)\}$.

In view of Proposition 2, define for any $1 \leq N \leq B \leq T \leq 11$ the parity matrix of an optimal $(n, k, T)_{\mathbb{F}}$ -block code as

$$\mathbf{V}_{\text{optimal}}^{(T-N+1) \times B} \triangleq \begin{cases} \mathbf{V}_{\text{Cauchy}}^{(T-N+1) \times B} & \text{if } (T, B, N) \notin \{(10, 8, 4), (11, 5, 4)\}, \\ \mathbf{V}_{\text{Vand}}^{(T-N+1) \times B} & \text{otherwise.} \end{cases}$$

Deterministic Construction of Optimal Streaming Codes:

Using Proposition 2 and the definition of $\mathbf{V}_{\text{optimal}}^{(T-N+1) \times B}$, we conclude that $\mathcal{C}(\mathbf{V}_{\text{optimal}}^{(T-N+1) \times B})$ is an optimal block code. In addition, by periodically interleaving n instances of $\mathcal{C}(\mathbf{V}_{\text{optimal}}^{(T-N+1) \times B})$, we can construct a (n, k, T) -code with $k = T - N + 1$ and $n = k + B$, which is optimal if $T \leq 11$ by Proposition 2. Interested readers can view an example in [26]. For any $1 \leq N \leq B \leq T \leq 11$, we let $\mathcal{C}_{T,B,N}$ denote the optimal (n, k, T) -code that is constructed by interleaving n instances of $\mathcal{C}(\mathbf{V}_{\text{optimal}}^{(T-N+1) \times B})$. The optimal streaming codes $\mathcal{C}_{T,B,N}$ are the building blocks for the network-adaptive streaming scheme described in the next section.

IV. NETWORK-ADAPTIVE ALGORITHM

A Conservative Algorithm that Estimates Channel Parameters B and N in L Channel Uses: In addition to the explicit construction of optimal streaming codes, we also present a conservative algorithm demonstrated by Algorithm 1. The algorithm estimates conservative coding parameters B and N in L channel uses, which is the duration of the algorithm. Conservative in this context implies coding parameters B and N that do not yield the lowest rate, where $B = N = T$ and $C(T, T, T) = \frac{1}{T+1}$. Before tracking any packet erasures, the algorithm fixes the decoding delay denoted by T and the duration of the algorithm denoted by L . Also, initially the channel is assumed to be ideal, i.e., introducing no erasures. Therefore, the algorithm sets the starting value of B , N , which are denoted by \hat{B}_{-1} and \hat{N}_{-1} respectively, and N_{max} , which is the maximum number of arbitrary erasures, is set to 0.

Every packet transmitted at channel use $i \in \mathbb{Z}_+$ is assumed to either reach the destination in the same channel use or be erased. In practice, packets that are dropped in the network are considered erased. Depending on the application, packets received out-of-order could either be considered erased or are reordered at the application layer. In our experiments, we do not see out-of-order packets detected, hence in our implementation we considered out-of-order packets to be erased. However, in practice if we use Real Time Protocol (RTP), then the receiver will have a reordering buffer that reorders packets received out-of-order.

For every non-erased packet received at channel use $i \in \mathbb{Z}_+$, the algorithm first deduces the erasure pattern $e_{\mathcal{W}} \triangleq (e_{j-T}, e_{j-T+1}, \dots, e_j) \in \{0, 1\}^{T+1}$ for each sliding window $\mathcal{W} = \{j-T, j-T+1, \dots, j\}$ of size $T+1$ such that $j \leq i$, where an element of $e_{\mathcal{W}}$ equals 1 if the corresponding

Algorithm 1: Estimating Conservative B and N .

Result: \hat{B}_i and \hat{N}_i are generated at the destination for every packet i where $0 \leq i \leq L-1$. All correctible length- $(T+1)$ erasure patterns that occur by channel use i can be perfectly recovered by any code that corrects all (\hat{B}_i, \hat{N}_i) -erasures.

Inputs : T , L and e^L denoting decoding delay, duration, and length- L erasure pattern respectively.

Outputs: \hat{B}_i and \hat{N}_i for every packet i .

```

1 previous_seq_# ← -1
2 ( $\hat{B}_{-1}, \hat{N}_{-1}, N_{\max}$ ) ← (0, 0, 0)
3 for  $i \leftarrow 0$  to  $L-1$  do // packets 0 to  $L-1$  sent
4   if  $e_i = 0$  then // packet  $i$  not erased
5     current_seq_# ←  $i$ 
6     for  $j \leftarrow$  previous_seq_# + 1 to current_seq_# do
7        $\mathcal{W} \leftarrow \{j-T, j-T+1, \dots, j\}$ 
8        $\bar{B}_j \leftarrow \max\{\text{span}(e_{\mathcal{W}}), \hat{B}_{j-1}\}$ 
9        $\bar{N}_j \leftarrow \max\{\text{wt}(e_{\mathcal{W}}), \hat{N}_{j-1}\}$ 
10       $N_{\max} \leftarrow \max\{\text{wt}(e_{\mathcal{W}}), N_{\max}\}$ 
11      if  $\bar{N}_j = 0$  or  $\bar{N}_j = T+1$  then // trivial
12        ( $\hat{B}_j, \hat{N}_j$ ) ← ( $\hat{B}_{j-1}, \hat{N}_{j-1}$ )
13      else // compute 3 hypothetic rates
14         $R_B \leftarrow \begin{cases} 0 & \text{if } \bar{B} = T+1, \\ C(T, \bar{B}_j, \max\{\hat{N}_{j-1}, 1\}) & \text{if } \bar{B} < T+1 \end{cases}$ 
15         $R_N \leftarrow C(T, \max\{\bar{B}_{j-1}, \bar{N}_j\}, \bar{N}_j)$ 
16         $R_{\text{MDS}} \leftarrow C(T, N_{\max}, N_{\max})$ 
17        switch max{ $R_B, R_N, R_{\text{MDS}}$ } do
18          case  $R_B$  do //  $R_B$  largest
19            ( $\hat{B}_j, \hat{N}_j$ ) ← ( $\bar{B}_j, \max\{\hat{N}_{j-1}, 1\}$ )
20            break
21          case  $R_N$  do //  $R_N$  largest
22            ( $\hat{B}_j, \hat{N}_j$ ) ← ( $\max\{\bar{B}_{j-1}, \bar{N}_j\}, \bar{N}_j$ )
23            break
24          case  $R_{\text{MDS}}$  do //  $R_{\text{MDS}}$  largest
25            ( $\hat{B}_j, \hat{N}_j$ ) ← ( $N_{\max}, N_{\max}$ )
26        end
27      end
28    end
29    previous_seq_# ← current_seq_#
30  end
31 end

```

packet is erased. Let $\text{wt}(e_{\mathcal{W}}) \triangleq \sum_{\ell \in \mathcal{W}} e_{\ell}$ and

$$\text{span}(e_{\mathcal{W}}) \triangleq \begin{cases} 0 & \text{if } \text{wt}(e_{\mathcal{W}}) = 0, \\ p_{\text{last}} - p_{\text{first}} + 1 & \text{otherwise,} \end{cases}$$

be the *weight* and *span* of $e_{\mathcal{W}}$ respectively, where p_{first} and p_{last} denote respectively the channel use indices of the first and last non-zero elements in $e_{\mathcal{W}}$. Intuitively speaking, $\text{span}(e_{\mathcal{W}})$ is the minimum length over all intervals that contain the support of $e_{\mathcal{W}}$. For each deduced erasure pattern $e_{\mathcal{W}} = (e_{j-T}, e_{j-T+1}, \dots, e_j)$, the algorithm first calculates $\text{wt}(e_{\mathcal{W}})$ and $\text{span}(e_{\mathcal{W}})$, and then assign the values to $(\bar{B}_j, \bar{N}_j, N_{\max})$ according to

$$\begin{aligned} \bar{B}_j &:= \max\{\text{span}(e_{\mathcal{W}}), \hat{B}_{j-1}\}, \\ \bar{N}_j &:= \max\{\text{wt}(e_{\mathcal{W}}), \hat{N}_{j-1}\}, \text{ and} \end{aligned}$$

$$N_{\max} := \max\{\text{wt}(e_{\mathcal{W}}), N_{\max}\}.$$

Then one of the following updates will occur: (i) \bar{B}_j gets assigned to \hat{B}_j , (ii) \bar{N}_j gets assigned to \hat{N}_j , or (iii) N_{\max} gets assigned to both \hat{B}_j and \hat{N}_j .

To precisely explain when will each update occur, the estimates \hat{B}_j and \hat{N}_j will be output according to the following three mutually exclusive cases:

Case $\bar{N}_j = 0$: In this case,

$$\bar{B}_j = \bar{N}_j = \text{wt}(e_{\mathcal{W}}) = \text{span}(e_{\mathcal{W}}) = \hat{N}_{j-1} = \hat{B}_{j-1} = 0,$$

which implies that no erasure has yet occurred upon the receipt of packet j . Then, Algorithm 1 sets $\hat{N}_j = \hat{B}_j = 0$, meaning that the estimates for N and B remain to be 0.

Case $\bar{N}_j = T+1$: Here all the elements of $e_{\mathcal{W}}$ equal 1. This means that all the packets in the window $\{j-T, j-T+1, \dots, j\}$ are erased. In this case, no $(n, k, T)_{\mathbb{F}}$ -code can correct $e_{\mathcal{W}}$. Therefore, Algorithm 1 sets $\hat{N}_j = \hat{N}_{j-1}$ and $\hat{B}_j = \hat{B}_{j-1}$, i.e., the algorithm keeps the estimates of N and B unchanged.

Case $0 < \bar{N}_j \neq T+1$: In this case, with the terminology that ε^{T+1} is a (B, N) -erasure sequence if either $\text{span}(e_{\mathcal{W}}) \leq B$ or $\text{wt}(e_{\mathcal{W}}) \leq N$ holds, every length- $(T+1)$ erasure pattern ε^{T+1} that has happened up to channel use j can be categorized into the following two types: (i) ε^{T+1} consists of all ones, hence it is uncorrectable, or (ii) ε^{T+1} is simultaneously a $(\bar{B}_j, \max\{\hat{N}_{j-1}, 1\})$ -erasure sequence, a $(\max\{\bar{B}_{j-1}, \bar{N}_j\}, \bar{N}_j)$ -erasure sequence, and a (N_{\max}, N_{\max}) -erasure sequence.

By construction, every length- $(T+1)$ erasure pattern up to channel use $j-1$ can be either Type (i) or is an $(\bar{B}_{j-1}, \bar{N}_{j-1})$ -erasure sequence. Therefore, Algorithm 1 calculates the best estimates for \hat{B}_j and \hat{N}_j so that the following two conditions hold:

- (I) Every length- $(T+1)$ erasure pattern up to channel use j can be classified into either Type (i) or is a (\bar{B}_j, \bar{N}_j) -erasure sequence.
- (II) The loss in the maximum achievable rate induced by updating the estimates from $(\bar{B}_{j-1}, \bar{N}_{j-1})$ to (\bar{B}_j, \bar{N}_j) is minimized.

The presence of Condition (II) is essential because it guarantees that the algorithm cannot output the trivial estimates $\hat{B}_j = \hat{N}_j = T$ that lead to the lowest rate $C(T, T, T) = \frac{1}{T+1}$.

To get the best estimates of \hat{B}_j and \hat{N}_j so that Conditions (I) and (II) hold, Algorithm 1 computes three hypothetic rates based on the (T, B, N) -capacity as follows:

$$R_B := \begin{cases} 0 & \text{if } \bar{B}_j = T+1, \\ C(T, \bar{B}_j, \max\{\hat{N}_{j-1}, 1\}) & \text{if } \bar{B}_j < T+1, \end{cases}$$

$$R_N := C(T, \max\{\bar{B}_{j-1}, \bar{N}_j\}, \bar{N}_j)$$

$$\text{and } R_{\text{MDS}} := C(T, N_{\max}, N_{\max})$$

respectively, where R_B denotes the hypothetic maximum achievable rate if \hat{B}_j is assigned the value \bar{B}_j followed by \hat{N}_j being assigned the value $\max\{\hat{N}_{j-1}, 1\}$ (note that any

$(\hat{B}_j, \hat{N}_{j-1})$ -erasure sequence is also a $(\hat{B}_j, \max\{\hat{N}_{j-1}, 1\})$ -erasure sequence), R_N denotes the hypothetical maximum achievable rate if \hat{N}_j is assigned the value \hat{N}_j followed by \hat{B}_j being assigned the value $\max\{\hat{B}_{j-1}, \hat{N}_j\}$ (note that any $(\hat{B}_{j-1}, \hat{N}_j)$ -erasure sequence is also a $(\max\{\hat{B}_{j-1}, \hat{N}_j\}, \hat{N}_j)$ -erasure sequence), and R_{MDS} denotes the hypothetical maximum rate if both \hat{B}_j and \hat{N}_j are assigned the same value N_{max} . Algorithm 1 sets (\hat{B}_j, \hat{N}_j) as shown at the end of the pseudocode so that the resultant maximum achievable rate $C(T, \hat{B}_j, \hat{N}_j)$ equals $\max\{R_B, R_N, R_{\text{MDS}}\}$.

Combining the above three cases, we conclude that for all $0 \leq i \leq L-1$, Algorithm 1 generates estimates (\hat{B}_i, \hat{N}_i) such that Conditions (I) and (II) hold.

Interleaved Conservative Algorithm Based on Algorithm 1: By providing conservative estimates for B and N , surely Algorithm 1 yields a code that perfectly corrects all observed length- $(T+1)$ correctable erasure sequences. However, obviously Algorithm 1 generates a sequence of recommended coding rates that is monotonically decreasing over time. This is one noticeable concern.

To try to solve this issue, we propose the *network-adaptive algorithm* that is based on interleaving Algorithm 1 as follows: At each channel use $\ell = 0, L, 2L, \dots$, an instance of Algorithm 1 denoted by \mathcal{A}_ℓ is initiated. Each \mathcal{A}_ℓ lasts for $2L$ channel uses, and let $(\hat{B}_j^{(\ell)}, \hat{N}_j^{(\ell)})$ denote the corresponding estimates generated at channel use j . Then at each channel use j , the network-adaptive algorithm outputs the estimate $(\hat{B}_j^{(\ell)}, \hat{N}_j^{(\ell)})$ provided by \mathcal{A}_ℓ at channel use j where ℓ is the unique integer that satisfies $\ell + L \leq j < j + 2L$.

In simple words, each interleaved Algorithm 1 will run for $2L$ channel uses where the first L estimates are ignored by the algorithm and the last L estimates are output by the network-adaptive algorithm. Our construction guarantees that the coding rate generated by our network-adaptive algorithm is not always monotonically decreasing over time. Particularly, if there are no erasures for consecutive $2L$ channel uses, the next estimates of (B, N) would be $(0, 0)$.

V. NETWORK-ADAPTIVE STREAMING SCHEME

After receiving packets at the destination, the receiver can estimate the values of $\{(\hat{B}_j, \hat{N}_j)\}_{j \in \mathbb{Z}_+}$ as suggested by the network-adaptive algorithm described in Section IV. The parameter estimator at the destination uses the network-adaptive algorithm to generate the estimates (\hat{B}_i, \hat{N}_i) when the codeword transmitted at time i is received by the destination. To minimize estimation error at the source side induced by obsolete channel information, the destination feeds back the estimates instantaneously every time it receives a packet. This way more erasure patterns can be corrected at the cost of a small rate loss.

Code Transition: Initially, from channel use 0, the network-adaptive algorithm outputs $(\hat{B}_0, \hat{N}_0) = (0, 0)$. This implies that the source will use the trivial rate-one encoder, and the codeword transmitted is identical to the original message produced.

Whenever the algorithm provides new estimates for (B, N) at channel use j denoted by (\hat{B}_j, \hat{N}_j) , the source shifts to the

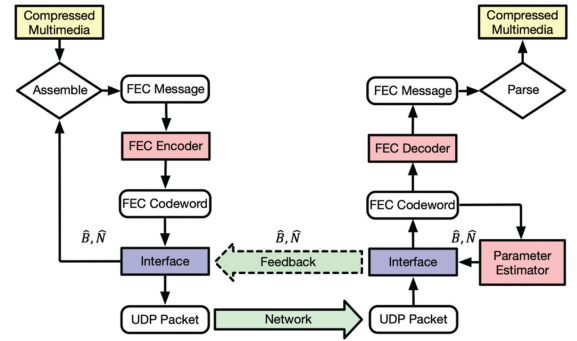


Fig. 3. Prototype of network-adaptive streaming scheme.

new encoder associated with the code $\mathcal{C}_{T, \hat{B}_j, \hat{N}_j}$ (defined at the end of Section III). To secure a smooth transition from using an old encoder with parameters $(B_{\text{old}}, N_{\text{old}})$ to using a new encoder with parameters $(B_{\text{new}}, N_{\text{new}}) \neq (B_{\text{old}}, N_{\text{old}})$, the source has to shield every transmitted packet by protecting it by either the old or new encoder.

For a smooth transition, suppose the source wants to shift to a new encoder associated with $\mathcal{C}_{T, B_{\text{new}}, N_{\text{new}}}$ starting from channel use i , it will use both the old and new encoders to encode the same message into old and new codewords from channel use i to channel use $i+T$. This will have the messages generated before channel use $i+T$ protected by the old codewords in $\mathcal{C}_{T, B_{\text{old}}, N_{\text{old}}}$, while the messages produced from channel use $i+T$ till the next encoder transition will be protected by the new codewords in $\mathcal{C}_{T, B_{\text{new}}, N_{\text{new}}}$. During the next encoder transition, the new encoder will be replaced by another newer encoder and be treated as an old encoder. This encoder transition procedure repeats guarding every transmitted packet.

Since every message is protected by either an old encoder with parameters $(B_{\text{old}}, N_{\text{old}})$ or a new encoder with parameters $(B_{\text{new}}, N_{\text{new}})$ during the encoder transition, any $(B_{\text{old}}, N_{\text{old}})$ -erasure sequence of length- $(T+1)$ that occurs before the transition can be corrected by the old encoder and any $(B_{\text{new}}, N_{\text{new}})$ -erasure sequence of length- $(T+1)$ that occurs during and after the transition can be corrected by the new encoder.

Prototype: The prototype of our proposed network-adaptive streaming scheme is shown in Fig. 3. The parameter estimator at the destination uses the network-adaptive algorithm to generate the estimates (\hat{B}_i, \hat{N}_i) for each $i \in \mathbb{Z}_+$. An FEC message generated at the source at each channel use i consists of a data buffer, an integer specifying the size of the buffer, a sequence number and the latest available estimates (\hat{B}, \hat{N}) fed back from the destination. The FEC message is then encoded into an FEC codeword and transmitted through the erasure channel.

The destination decodes all the messages generated before channel use $i-T$ for every codeword received at channel use i , that have not been decoded yet. The relevant decoder can be chosen by the destination based on the coding parameters contained in all the received codewords up to channel use i . Hence, every received codeword may result in more than one decoded message. For every reconstructed FEC message, the corresponding data buffer, the size of the buffer and the sequence number are extracted for further processing at the application.

It is important to note that our decoding error spread is limited to our sliding window of $T + 1$ packets. This is due to finite encoding memory of $T + 1$ packets.³

VI. SIMULATION AND EXPERIMENTAL RESULTS

To start with, we first design an implementation of our proposed network-adaptive streaming algorithm. To experiment our network-adaptive streaming algorithm, we test our algorithm on a simulated statistical erasure channel and on real-world channels. We compare our algorithm against both uncoded, non-adaptive schemes and MDS-adaptive schemes. The performance gains are evaluated in terms of frame loss rate (FLR), Perceptual Evaluation of Speech Quality (PESQ) score, and coding rate. We report that our proposed network-adaptive streaming algorithm outperforms uncoded and non-adaptive schemes in all contexts. Finally, our network-adaptive algorithm performs either better or close compared to MDS-adaptive coding, but with always higher coding rate and hence lower redundancy. We also study factors that can potentially affect the performance of our algorithm, such as network delay.

A. Implementation of Network-Adaptive Streaming Scheme

To explore the potential of our proposed network-adaptive streaming scheme described in Section V, we implement the proposed scheme for low-latency communication between a source and a destination in C++ programming language.

We assume that the source transmits a stream of compressed multimedia frames over the Internet to the destination. By using a standard video codec or voice codec, each compressed multimedia frame could be created from the raw data. Next, the compressed multimedia frame, together with the estimated coding parameters received from the feedback channel, is encapsulated in an FEC message. The FEC message is further encoded into an FEC codeword to be encapsulated in a network packet, which is then forwarded to the destination.

In all our experiments we focus on audio multimedia frames. This is because in an interactive video-streaming application such as video conferencing, voice is more delay-sensitive than video. This paper focuses on applying adaptive FEC schemes to protect the data with the most stringent delay constraint, i.e., voice. Since our application of choice in this paper is lower bandwidth audio/voice, we do not adopt congestion control in our experiments. The application of our FEC schemes on video and its interaction with congestion control schemes are interesting directions left for future research.

Every network packet is either received by the intended destination or dropped (erased). Each FEC codeword received at the destination is extracted from every network packet received, and one or more FEC messages are recovered based on the codeword. A recovered compressed multimedia frame is extracted from every recovered FEC message, and then decompressed using the video or voice codec back to raw data.

³Our source code studies the effect of finite decoding error since we cannot decode packets using past packets that were not successfully decoded.

The two interface modules between the streaming code and the network layer are illustrated in Fig. 3. The first module is at the source. The interface at the source simultaneously encapsulates every FEC codeword into a UDP packet and forwards every estimated coding parameter obtained from the feedback channel to the message assembler. The second module is the interface at the receiver-side that concurrently extracts the codeword buffer in every network packet to form an FEC codeword and forwards each estimated coding parameters over UDP to the feedback channel.

B. Parameters and Error Metrics

We compare the uncoded, non-adaptive, MDS-adaptive coding scheme FLRs achieved with our network-adaptive streaming scheme, as described in Section V. When comparing non-adaptive and MDS-adaptive schemes with our network adaptive streaming scheme, we choose a delay constraint of $T = 10$ packets. For non-adaptive schemes, we fix the coding parameters (B, N) . To this end, we fix the frame duration and bit rate for the compressed multimedia frame to be 10 ms and 240 kbit/s respectively, which are practical as existing audio codecs typically have frame duration 2.5–60 ms and bit rate 6–510 kbit/s [28], [29].

Consequently, every 300-byte compressed frame is generated every 10 ms. The 10 ms frame duration and the delay constraint T must be carefully chosen so that the resultant playback delay $T \times 10$ ms in addition to the propagation delay must be smaller than the 150 ms delay required by ITU for interactive applications [30], [31]. To illustrate, if the propagation delay is 100 ms, then the resultant playback delay must be less than $150 \text{ ms} - 100 \text{ ms} = 50 \text{ ms}$, which can be achieved by choosing suitable T and frame duration such that their product is below 50 ms.

For our experimental purpose, we assume that the propagation delay is less than 50 ms and choose $T = 10$ so that the resulting playback delay $T \times 10 \text{ ms} = 100 \text{ ms}$ besides the propagation delay is below 150 ms. In the network-adaptive algorithm described in Section IV, we set $L = 1000$. In other words, each interleaved Algorithm 1 runs for $2L \times 10 \times 0.001$ seconds = 20 seconds where the algorithm ignores the $L = 1000$ estimates generated in the first 10 seconds, and the next $L = 1000$ estimates are produced by the algorithm.

Let $M = 360$ be the number of 10-second sessions throughout the transmission, involving a total of $L \times M = 360\,000$ packets that last for one hour. In each session, $L = 1000$ packets are transmitted from the source to the destination. For simplicity, let the sequence number of a packet be its channel use index, starting from 0 and ending at $L \times M - 1$.

During each session $m \in \{1, 2, \dots, M\}$, the source transmits packets with sequence number between $L(m - 1)$ and $Lm - 1$. For each session m , let ε_m denote the corresponding FLR achieved by our network-adaptive streaming scheme. More precisely, $L(1 - \varepsilon_m)$ is the number of FEC messages with sequence number between $L(m - 1)$ and $Lm - 1$ which are perfectly recovered by the destination. We will express in the next two sections our simulation and experimental results

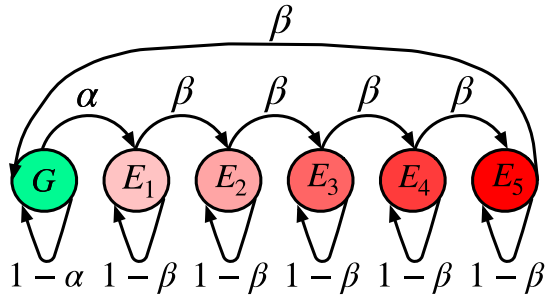


Fig. 4. Six-state Markov model: Fritchman channel simulated.

respectively in terms of the average FLR defined as $\frac{1}{M} \sum_{m=1}^M \varepsilon_m$ and the fraction of *low-fidelity* sessions with FLR larger than 10% defined as $\frac{1}{M} \sum_{m=1}^M \mathbf{1}\{\varepsilon_m > 0.1\}$.

C. Simulation Results for a Three-Phase Fritchman Channel

We validate the superiority of our network-adaptive streaming scheme as described in Section VI-A to non-adaptive streaming schemes. This is by simulating artificial packet erasures according to the Fritchman channel [32], which is a well-known statistical channel that is useful for approximating packet losses experienced at the network layer [33].

The Fritchman channel is a $(M + 1)$ -state Markov model which consists of one good state G and M bad states denoted by E_1, E_2, \dots, E_M . In the good state G , each channel packet is lost with probability $\epsilon \in [0, 1)$ whereas in the bad state, each channel packet is lost with probability 1.

If the state at time i is G , then the probability of transitioning to E_1 is α , and the probability of remaining in the same state G is $1 - \alpha$ at $i + 1$. When the state at i is E_M , the transition probability to state G is β , while the probability of staying in E_M is $1 - \beta$. If the current state is E_l for some $l \in 1, 2, \dots, M - 1$, then the probability of transitioning to E_{l+1} is β , and it will stay in state E_l with probability $1 - \beta$. In our simulated Fritchman channel, we use the six-state Markov chain, where $M = 5$. Fig. 4 summarizes the transition probabilities of our simulated Fritchman channel.

As long as the channel remains in bad states, the channel behaves as a burst erasure channel. The higher the number of bad states, the higher the probability of getting bursty erasures. In contrast, the channel behaves like an i.i.d. erasure channel when the channel stays in the good state. Hence, we expect to see more arbitrary erasures in the good state.

In our simulations, we wanted to simulate a dynamic channel, so we considered the following *three-phase Fritchman channel*: The simulation of the Fritchman channel consists of three phases, where α , β and ϵ are fixed during the first quarter and last quarter of the simulation. While in the middle phase, the probability of being in the good state G is 1. In other words, there are no consecutive bad states in the middle phase, implying that the middle phase introduces fewer burst erasures compared to the first and last phase.

For the three-phase Fritchman channel with constant parameters $(\alpha, \beta) = (0.005, 0.990)$, we focus on the case where $T = 10$ for all the graphs we show in this section. For the case where

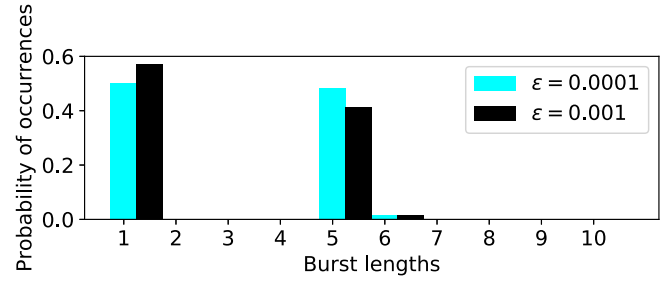


Fig. 5. Empirical burst-length distribution for $\epsilon = 0.0001$ and 0.001 .

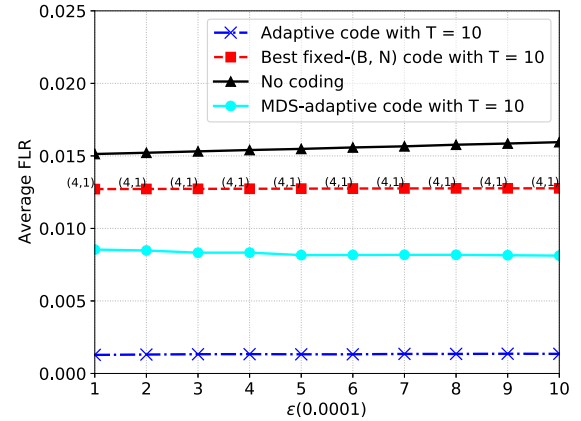


Fig. 6. Average FLR obtained from Fritchman channel.

$\epsilon = 0.0001, 0.001$, the empirical distribution of burst lengths is shown in Fig. 5. We exhibit a bimodal distribution with clusters at burst lengths 1 and 5.

FLRs and Coding Rates: We plot in Fig. 6 the average FLRs against the varying parameter ϵ for the uncoded scheme, the network-adaptive streaming scheme, and the best non-adaptive (fixed-rate) streaming code $\mathcal{C}_{T,B,N}$. The coding rate of the best non-adaptive (fixed-rate) streaming code does not exceed the average coding rate of the adaptive scheme. We achieve this by getting the corresponding value/s of (B, N) to the average coding rate of our network adaptive streaming code. Several values of (B, N) can correspond to the same coding rate. To choose the best non-adaptive streaming code, we try running all the corresponding fixed values of (B, N) on the same packet loss trace file and choose the (B, N) parameters resulting in the least averaged FLR over all the sessions.

The correspondent coding rates for each ϵ are plotted in Fig. 7, where the coding rate for the no-coding scheme is always one and thus not plotted. Figs. 6 and 7 show that compared to non-adaptive (fixed-rate) schemes, our adaptive scheme achieves approximately $10\times$ lower FLRs and slightly higher coding rates across all values of ϵ between 0.0001 and 0.001. The gain of the adaptive scheme compared to fixed-rate codes is attributed to the significantly improved estimation of instantaneous channel conditions, and hence instantaneous change in coding parameters.

We also show in Fig. 8 the variation of average FLRs for our adaptive streaming scheme and uncoded scheme across the 360 sessions, each lasting for 10 seconds, for $\epsilon = 0.0001$. It can be seen from Fig. 8 that our adaptive scheme achieves less

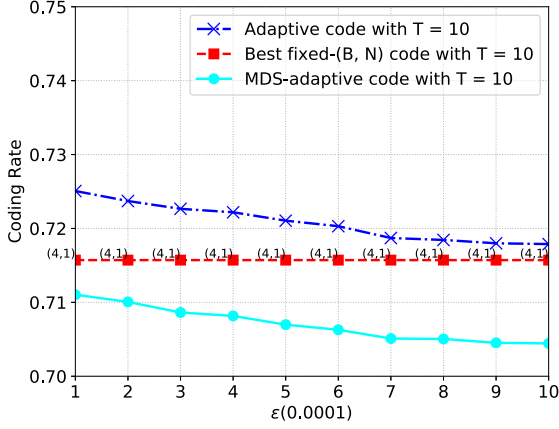
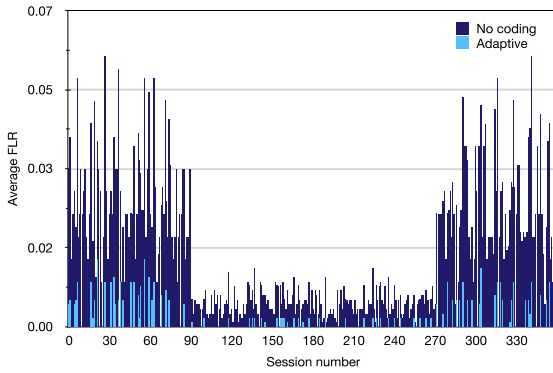
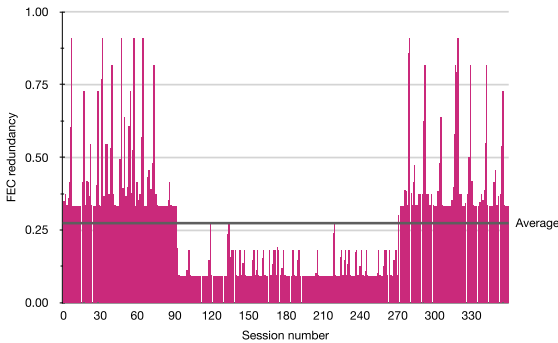


Fig. 7. Average coding rate obtained from Fritchman channel.

Fig. 8. Average FLRs for adaptive FEC over time for $\epsilon = 0.0001$.Fig. 9. FEC redundancy over time for $\epsilon = 0.0001$.

than half of the no-coding scheme loss rate for all sessions. In addition, we display the variation of the FEC redundancy (i.e., one minus coding rate) for our adaptive scheme in Fig. 9, which demonstrates how quickly it reacts to erasures, especially at the transition between the first and middle phases and the transition between the middle and last phases.

Our simulation results reveal that our adaptive code adapts significantly better than the other state-of-the-art strategies when channel dynamics occur. Since a feedback packet is sent instantaneously when every packet is successfully received as described earlier, the estimation error of the coding parameters due to channel dynamics is minimized.

The reason why our adaptive scheme significantly outperforms non-adaptive ones can be explained with the help of



Fig. 10. Packet losses recovered by different schemes.

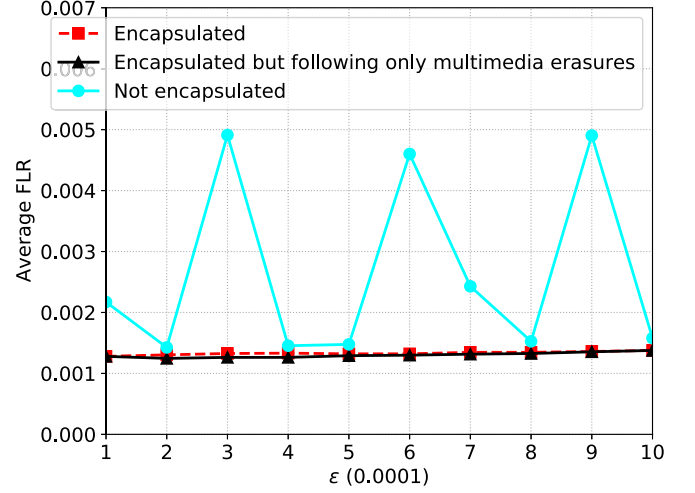


Fig. 11. Average FLR obtained from Fritchman channel not encapsulating and encapsulating the multimedia and parity packets together.

Fig. 10. Suppose 11 out of 40 of the network packets are dropped as shown in Fig. 10. Our adaptive coding scheme updates the code in this order: $\mathcal{C}_{10,1,1}$ and $\mathcal{C}_{10,5,2}$ before transmitting packets 4 and 18 respectively. Therefore, the subsequent five packets losses are all recovered by $\mathcal{C}_{10,5,2}$ as shown in Fig. 10, whereas the fixed-rate code $\mathcal{C}_{10,4,4}$ can only recover one packet.

To test if the performance of our algorithm will be affected when the parity and multimedia packets are transmitted separately, we perform three different experiments where we encapsulate and not encapsulate multimedia and parity packets and send them over Fritchman channel. When we transmit the multimedia and parity packets separately, for each multimedia packet we transmit the parity packet consecutively. We generate an erasure pattern for 720 000 packets, instead of 360 000 according to the aforementioned Fritchman channel, for the multimedia and parity packets.

In Fig. 11, we show the results of three experiments, where we test our algorithm using (i) the erasure pattern for 722 000 packets with parity and multimedia packets sent separately, (ii) the multimedia packet erasures only (even indexed erasures) on encapsulated packet, to ensure that any differences in performance is not due to different erasures observed by multimedia packet, (iii) the first 360 000 erasures from the pattern, which is following Fritchman channel erasures, on encapsulated packet, to adequately assess the performance difference on a Fritchman channel. These experiments are labeled “Not encapsulated,” “Encapsulated but following only multimedia erasures” and “Encapsulated” respectively in Fig. 11.

We observe in Fig. 11, that sending multimedia and parity packets separately degraded the algorithm’s performance. One

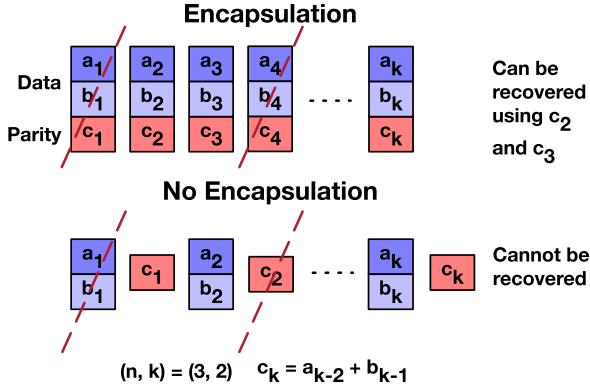


Fig. 12. Example showing when will separately sending multimedia and parity packets cause decoding error.

possible reason for that is demonstrated in Fig. 12. We have a packet loss at index 1 and 4, and using $(n, k) = (3, 2)$ where the parity is constructed using $c_k = a_{k-2} + b_{k-1}$. If we were to encapsulate the data and parity packets, we will be able to recover the lost packets at index 1 using c_2 and c_3 , for example. However, when we transmit the data and parity packets separately, the packet with index 1 is lost and cannot be recovered using c_2 , as c_2 has index 4. Following the multimedia packet erasures or the Fritchman channel erasures did not affect the performance greatly, which greatly emphasizes that not encapsulating the multimedia and parity packet is the main reason for the performance degradation. Indeed, our conclusion is only based on our experiments and our reasoning is giving only one example, therefore, we are not claiming that our conclusion generalizes for settings different than our experiments.

PESQ Scores: Earlier, we presented our simulation results in terms of FLRs for compressed multimedia frames where the duration and bit rate for each compressed frame are 10 ms and 240 kbit/s respectively. Next, we use the commonly adopted wideband PESQ score [27] for uncompressed multimedia (WAV) files to demonstrate the advantage in perceptual quality of using our network-adaptive streaming scheme over uncoded and non-adaptive streaming codes.

We first choose a one-hour uncompressed speech file with sampling frequency of 16 000 Hz and sample size of 16 bits and use the constant-bitrate (CBR) WMAv2 codec to generate compressed multimedia data with sampling frequency of 16 000 Hz and bit rate of 240 kbit/s. This is considerably high bit-rate to ensure any degradations comes from loss rather than compression. Then, we use the network-adaptive, uncoded and best non-adaptive scheme to transmit the compressed multimedia through the three-phase Fritchman channel as described in the previous subsection. Whenever a 300-byte compressed multimedia frame cannot be recovered by the destination, the lost frame is replaced with a 300-byte all-zero frame. The schematic diagram for the codec operations is shown in Fig. 13.

For each 10-second session, a PESQ score is computed between the original uncompressed WAV audio and the recovered WAV audio for all the schemes. Each PESQ score ranges from 1 – 5 where a higher score means a better speech quality [27]. We

plot in Fig. 14 the average simulated PESQ scores over $M = 360$ sessions for our network-adaptive streaming scheme, the uncoded scheme and the best non-adaptive streaming code $C_{T,B,N}$ whose coding rate does not exceed the average coding rate of the network-adaptive scheme.

We can see from Fig. 14 that our adaptive streaming scheme achieves a significantly higher average PESQ score than uncoded and non-adaptive streaming schemes. However, the average PESQ score over 360 sessions may not be an indicator for the performance of our network-adaptive streaming code in affected sessions, where it suffers from higher loss rates. Hence, in Fig. 15, we plot the average PESQ scores of only affected sessions in uncoded scheme by varying ϵ comparing non-adaptive scheme, uncoded scheme with our network adaptive scheme. Affected sessions are where the PESQ score is below 3.8, if we were using uncoded schemes. In other words, we are comparing different coding schemes only in catastrophic situations.

In Fig. 15, we can see that in catastrophic situations, probably where the burst length is high, the performance of non-adaptive and uncoded coding scheme is almost the same. This is due to using a small value of B , while the burst distribution is mainly at 1 and 5 as shown in Fig. 5. However, our network adaptive scheme is better than uncoded and non-adaptive coding schemes, which shows that our network-adaptive algorithm is capable of detecting erasure patterns and correcting them by choosing appropriate coding parameters (B, N) .

We plot in Fig. 16 the cumulative distribution function (CDF) of PESQ score (i.e., the fraction of the 360 sessions whose scores are less than a certain PESQ score) for each of the following schemes when $\epsilon = 0.0001$: Our adaptive streaming scheme, the best non-adaptive streaming code and the uncoded scheme. It can be seen from Fig. 16 that our adaptive scheme provides the best user experience compared to the uncoded and non-adaptive schemes. If the PESQ score of a session is lower than 3.8, many users will be dissatisfied by the unclear speech, and we will call such a session a *low-satisfaction* session. Fig. 17 shows that our adaptive streaming scheme has 0 sessions with low-satisfaction sessions compared to the uncoded and non-adaptive schemes, which have above 32% low-satisfaction sessions.

Optimal Streaming Codes vs. MDS-Based Streaming Codes: In order to demonstrate the advantage of using our constructed streaming codes described in Section III over traditional MDS-based codes, we consider the following *MDS-adaptive streaming scheme*: Instead of outputting the coding parameters (\hat{B}, \hat{N}) for an optimal block code which corrects a length- \hat{B} burst erasure and \hat{N} arbitrary erasures, the adaptive algorithm outputs a single coding parameter N of an MDS code which corrects only N arbitrary erasures such that the resultant coding rate $C(T, N, N) = \frac{T-N+1}{T+1}$ satisfies

$$C(T, N, N) \leq C(T, \hat{B}, \hat{N}) \leq C(T, N-1, N-1),$$

meaning that the resultant coding rate is the largest possible rate achieved by an MDS code that is less than $C(T, \hat{B}, \hat{N}) \triangleq \frac{T-\hat{N}+1}{T-\hat{B}+\hat{N}+1}$.

We plot in Fig. 6 the FLRs achieved by the MDS-adaptive streaming scheme against ϵ for the MDS-adaptive streaming

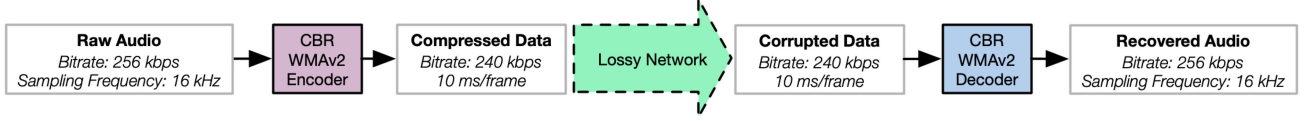


Fig. 13. Codec operations.

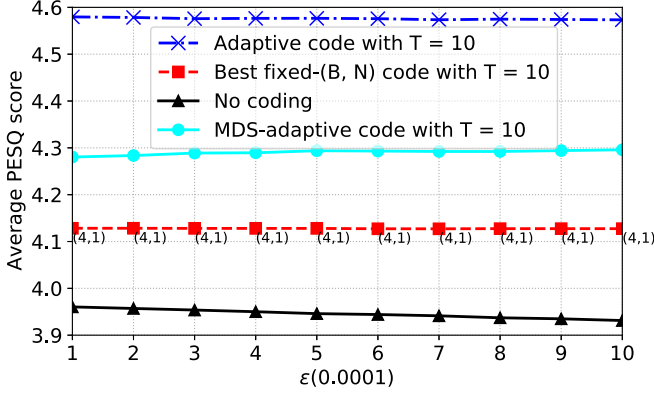


Fig. 14. Average PESQ score obtained from Fritchman channel simulation.

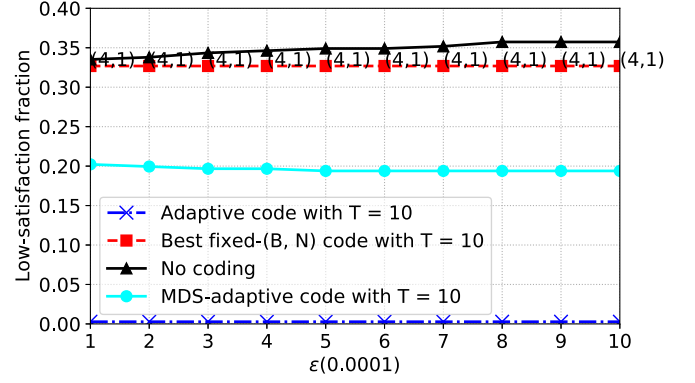


Fig. 17. Low-satisfaction fraction obtained from Fritchman channel simulation.

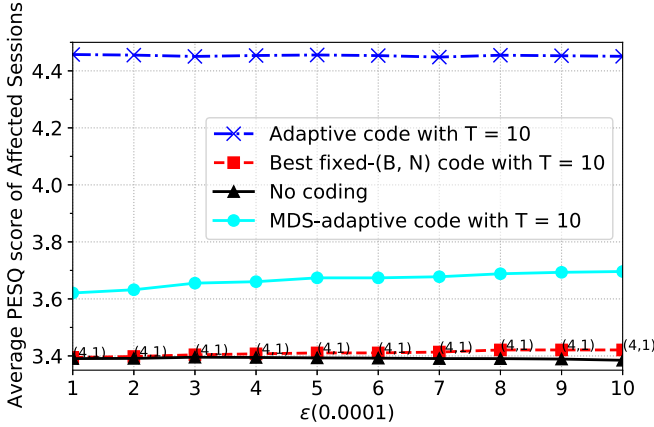
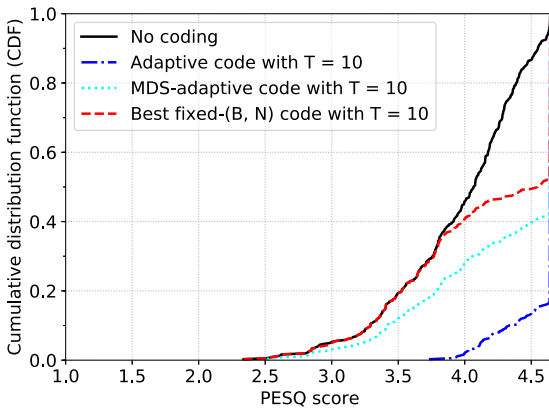


Fig. 15. Average PESQ score of affected sessions obtained from Fritchman channel simulation.

Fig. 16. The CDF of simulated PESQ score for adaptive FEC, MDS-adaptive FEC and non-adaptive FEC for $\epsilon = 0.0001$.

scheme, and plot in Fig. 7 their corresponding coding rates. Figs. 6 and 7 show that compared to the MDS-adaptive scheme, our adaptive scheme achieves approximately $8\times$ lower FLR and higher coding rate across all values of ϵ between 0.0001 and 0.001.

Fig. 14 shows that our optimal adaptive scheme achieves a higher PESQ score over MDS-adaptive schemes. As mentioned earlier, the average PESQ score for affected sessions gives a better indication of the performance of our scheme in worse environments, and our adaptive scheme has a 20% improvement in PESQ score compared to adaptive MDS-streaming code in affected sessions as shown in Fig. 15. Also, the number of affected sessions is far lower using our adaptive scheme compared to MDS-code as per Fig. 17. This is also illustrated in the CDF plot in Fig. 16 that shows our adaptive network scheme having higher fraction of sessions with higher PESQ scores.

It is possible that our-adaptive network scheme chooses an MDS code if it is optimum. The superiority of our adaptive scheme to the MDS-adaptive scheme is because the fraction of non-MDS codes chosen by our adaptive algorithm is approximately 40% or more for all $\epsilon \in [0.0001, 0.001]$. This implies that the optimal streaming codes chosen by our adaptive scheme are non-MDS more than 40% of the time. Consequently, the streaming codes chosen by the MDS-adaptive streaming scheme are inferior to those chosen by our adaptive scheme.

On another level, the MDS scheme can definitely achieve the same level of protection as our network adaptive scheme, but at the cost of higher decoding delay d_d . For example, for $\epsilon = 0.0001 - 0.001$, our network adaptive scheme can achieve similar FLRs (within 10%) if T was set to 4. Therefore, our network adaptive scheme can recover packets with a lower delay compared to MDS-based schemes.

D. Experimental Results for a Wi-Fi Network

In our real-world experiment, the source and the destination are connected over the same Wi-Fi network whose capacity is approximately 30 Mbit/s. To simulate cross traffic any user would experience while sharing the same Wi-Fi network, we introduce UDP cross traffic using Iperf. We call the UDP cross traffic *offered load*, whose throughput is fixed at the beginning of the experiment and kept unchanged during the experiment.

To simulate the packet losses experienced during the real-world experiments, we record packet loss traces. These traces are used to compare the performance of our adaptive streaming with the best non-adaptive (fixed rate) streaming code $\mathcal{C}_{T,B,N}$ whose coding rate does not exceed the average coding rate of the adaptive scheme.

FLRs and Coding Rates: The average FLRs for our network-adaptive streaming scheme as described in Section VI-A and the uncoded scheme are plotted against the ratio of the offered load occupied by Iperf traffic in Fig. 18(a). Also, the average FLR for the best non-adaptive streaming code $\mathcal{C}_{T,B,N}$ with parameters (B, N) is plotted in Fig. 18(a). Fig. 18(a) and Fig. 18(c) show that our adaptive streaming scheme achieves significantly lower average FLRs and higher average coding rate than non-adaptive streaming codes in all offered loads. The gain of the adaptive scheme compared to fixed-rate codes is attributed to the significantly improved estimation of instantaneous channel conditions. In real-world networks, errors occur in bursty manner, meaning that most of the time the networks are free of error. Fixed-rate codes compared with adaptive codes use a larger overhead to transmit the same amount of data because it cannot adapt to a higher coding rate when the channel is error-free.

For interactive audio, low-fidelity sessions lead to unclear speech or even call termination which directly affects user experience. Fig. 18(b) shows that our adaptive streaming scheme provides a substantially better audio quality than the uncoded and non-adaptive streaming scheme.

For the case where the offered load equals 40% of the capacity, we display the empirical distribution of burst lengths in Fig. 19. We observe a mean of burst lengths of 4.32 and standard deviation of 6.64. We also show in Fig. 20 the variation of average FLRs for our adaptive streaming scheme and UDP across the 360 sessions. It can be seen from Fig. 20 that for more than quarter of the sessions that experience packet loss, our adaptive scheme achieves less than half of the no-coding loss rate. In addition, we display the variation of the FEC redundancy (i.e., one minus coding rate) for our adaptive scheme in Fig. 21, which demonstrates how quickly it reacts to erasures.

PESQ Scores: Previously, we presented our experimental results in terms of FLRs for compressed multimedia frames where the duration and bit rate for each compressed frame are 10 ms and 240 kbit/s respectively. Next, we follow the audio and codec settings as described in Section VI-C (cf. Fig. 13) and use the network-adaptive, uncoded or best non-adaptive scheme to transmit the compressed multimedia through the Wi-Fi network subject to Iperf UDP cross traffic as described in the previous subsection.

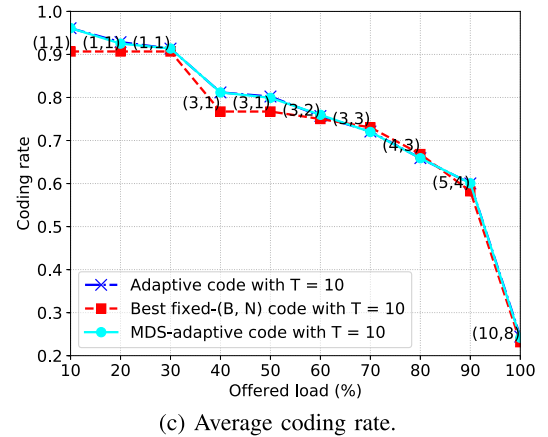
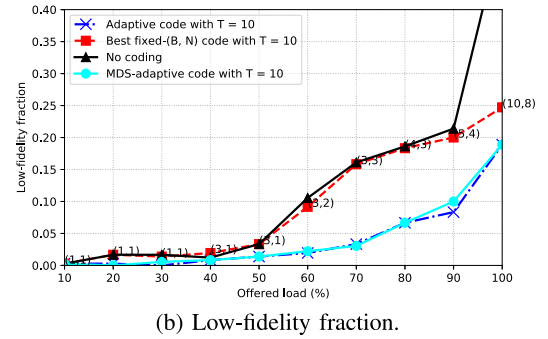
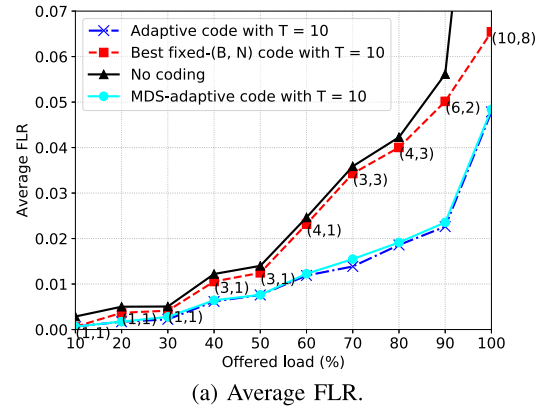
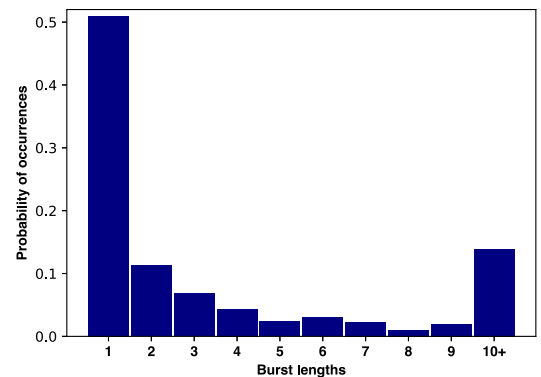


Fig. 18. Experimental results for different streaming schemes interfered by Iperf cross traffic.



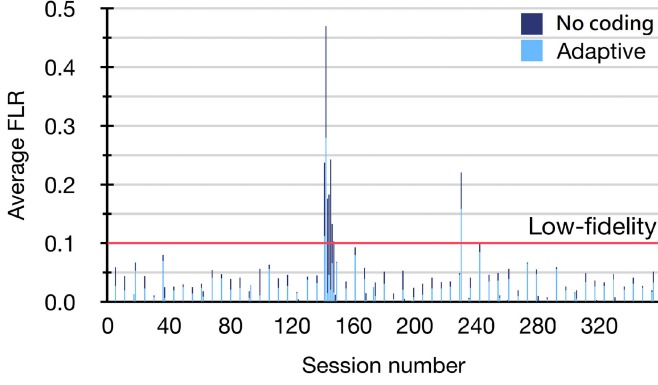


Fig. 20. Average FLRs for adaptive FEC over time for 40%-capacity offered load.

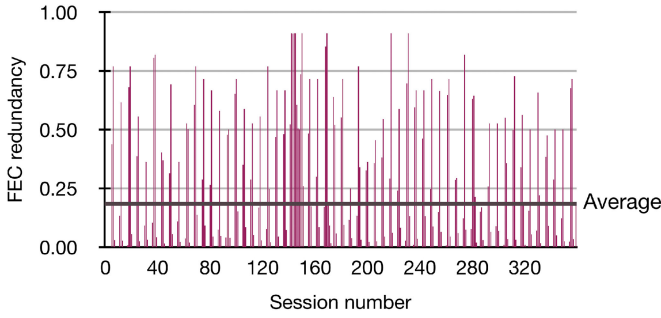


Fig. 21. FEC redundancy for 40%-capacity offered load.

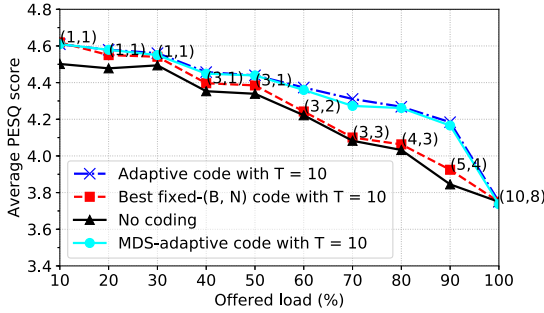


Fig. 22. Average PESQ score subject to Iperf traffic.

For each 10-second session, a PESQ score is computed between the original uncompressed WAV audio and the recovered WAV audio for our all schemes. The average PESQ scores over 360 sessions for our network-adaptive streaming scheme is plotted against the percentage of the network capacity occupied by Iperf traffic in Fig. 22.

In addition, we use the recorded packet loss traces to simulate the average PESQ scores for the uncoded scheme and the best non-adaptive streaming code $C_{T,B,N}$ whose coding rate does not exceed the average coding rate of the network-adaptive scheme. The average PESQ scores for the uncoded scheme and the best non-adaptive streaming code with parameters (B, N) are also plotted in Fig. 22.

We can see from Fig. 22 that our adaptive streaming scheme achieves a higher average PESQ score than uncoded and non-adaptive streaming schemes. In addition, Fig. 23 shows that

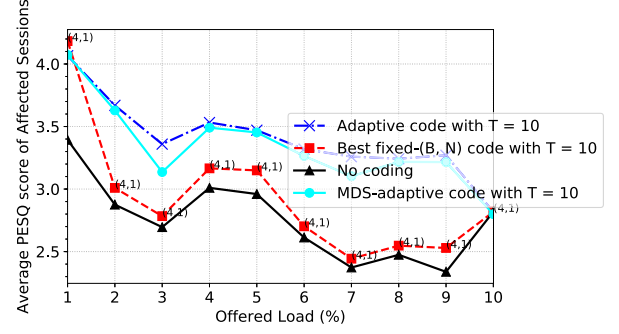


Fig. 23. Low-satisfaction fraction subject to Iperf traffic.

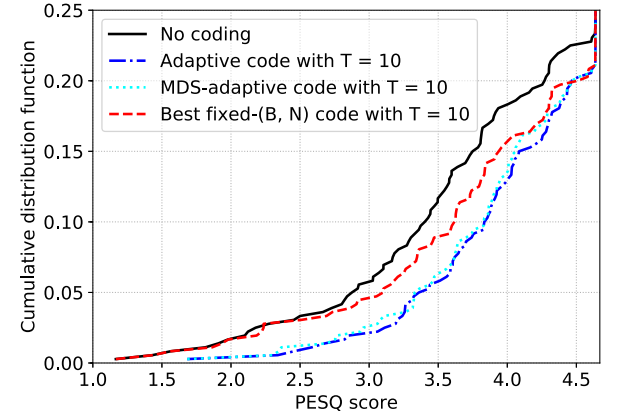


Fig. 24. Average PESQ of affected sessions subject to Iperf traffic.

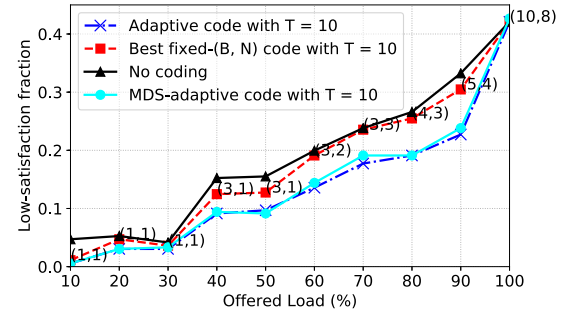


Fig. 25. The CDF of PESQ score for adaptive FEC, MDS-adaptive FEC and non-adaptive FEC for 40%-capacity offered load.

our adaptive streaming scheme significantly reduces the fraction of low-satisfaction sessions (with PESQ score lower than 3.8) compared to the uncoded and non-adaptive schemes. As we mentioned earlier, the average of PESQ scores of affected sessions may better show the enhancement in the performance in extreme situations, hence Fig. 24 shows the average PESQ scores of affected sessions is higher for our scheme compared to uncoded and non-adaptive streaming schemes.

We plot in Fig. 25 the CDF of PESQ score (i.e., the fraction of the 360 sessions whose scores are less than a certain PESQ score) for each of the following schemes under 40%-capacity offered load: Our adaptive streaming scheme, the best non-adaptive streaming code and the uncoded scheme. It can be seen from Fig. 25 that our adaptive scheme provides the best user experience compared to the uncoded and non-adaptive schemes.

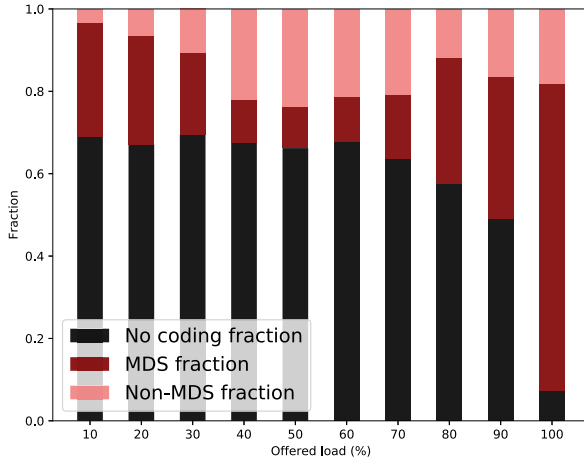


Fig. 26. Fraction of time when non-MDS codes are used.

TABLE II
PERFORMANCE OF DIFFERENT STREAMING STRATEGIES FOR 30%-CAPACITY OFFERED LOAD

strategy	FEC redundancy	average FLR	low-fi fraction
network-adaptive	19.03%	0.01190	0.03047
MDS-adaptive	20.17%	0.01757	0.03878
no coding	0%	0.03489	0.01856

(a)

network-adaptive	16.6%	0.02586	0.07202
MDS-adaptive	17.2%	0.02617	0.07756
no coding	0%	0.04003	0.17452

(b)

network-adaptive	16.10%	0.00486	0.00831
MDS-adaptive	17.24%	0.01040	0.01108
no coding	0%	0.02486	0.13573

(c)

Optimal Streaming Codes vs. MDS-Based Streaming Codes: We plot in Fig. 18(a), 18(b), 22, 23 and 25 the respective average FLR, low-fidelity fraction, average PESQ score, low-satisfaction fraction and the PESQ CDF under 40%-capacity offered load for the MDS-adaptive streaming scheme as described in Section VI-C, which show that our network-adaptive scheme is marginally better than the MDS-adaptive scheme. This can be explained by Fig. 26 which shows that the fraction of non-MDS codes chosen by the adaptive algorithm used by our adaptive scheme is less than 25% for all offered loads, implying that the optimal streaming codes chosen by our adaptive scheme are non-MDS less than 25% of the time.

Since it is unclear from Fig. 18(a), 18(b), 22 and 23 that our network-adaptive streaming scheme can significantly outperform the MDS-adaptive streaming scheme in real-world networks, we use the recorded packet loss traces obtained from a repeated experiment with 30%-capacity offered load to compare the network-adaptive streaming scheme with the MDS-adaptive streaming scheme for $T = 10$. Table II(a) shows that although the two schemes have similar rates, the network-adaptive streaming scheme achieves around 80% of the average FLR and 80% of

the low-fidelity sessions achieved by the MDS-adaptive streaming scheme, which implies that our constructed optimal streaming codes outperforms traditional MDS-based streaming codes in real-world networks.

The reduction in FLR is not surprising because our optimal streaming codes treat burst erasures and arbitrary erasures differently while MDS-based codes do not differentiate them. Even if T slightly deviates from 10, our experimental results displayed in Tables II(b) and II(c) show that the network-adaptive streaming scheme compared with the MDS-adaptive streaming scheme can achieve a considerable reduction in FLR. In particular, for $T = 11$, although the two adaptive schemes have similar rates, the network-adaptive streaming scheme achieves around 50% of the average FLR and 75% of the low-fidelity sessions achieved by the MDS-adaptive streaming scheme.

In our setting, due to the 150 ms one-way delay constraint and the standard 10 ms frame duration assumption, restricting $T \leq 11$ is a reasonable assumption. The search for efficient streaming codes over GF(256) or other practical fields for $T > 11$ is an interesting direction for future research.

E. Influencing Factors

Network Delay: In the Section VI-C and VI-D, we demonstrated results of the performance of our algorithm in Fritchman channel and Wi-Fi settings; however, the erasure pattern is not the only factor affecting the performance of our algorithm. Having a network delay, indeed, delays the feedback of new estimates of coding parameters to the sender, and hence delays the transitioning to new better coding parameters. Nevertheless, since our network-adaptive algorithm estimates the coding parameters based on the past window of L packets, where L is large compared to that transmitted during a round trip time, we are not expecting a delay in feedback to effect our performance significantly. We study the affect of this network delay on the three-phase Fritchman channel and Wi-Fi settings that we studied in Section VI-C and VI-D.

We used the traffic control `tc` in linux to configure the kernel delay over the local interface where we launch the sender and receiver. Simulating the Fritchman channel and Wi-Fi network, we vary the delay from 10–50 ms. We stop at 50 ms, since this is the assumed maximum delay allowed for $T = 10$ as stated earlier in Section VI-B. In Fig. 27, we plot the rate of change in FLR with respect to the FLR with delay 0 ms vs. the delay in ms. We observe that in the three-phase Fritchman channel with $\epsilon = 0.0001$ and in the Wi-Fi network with 40% offered load, the rate of change in FLR was increasing as the delay increases. However, the effect of delay is less obvious in the Fritchman channel.

Although the performance of our algorithm degrades with increase in network delay, our performance continues to be better than using the best-fixed (B, N) code. In Table III, we compare the average FLR using the best-fixed (B, N) code and our network-adaptive code for the Fritchman channel and Wi-Fi environments under different delays.

Dynamic Environments: In Fig. 27, we observe that in the three-phase Fritchman channel with $\epsilon = 0.0001$, the rate of

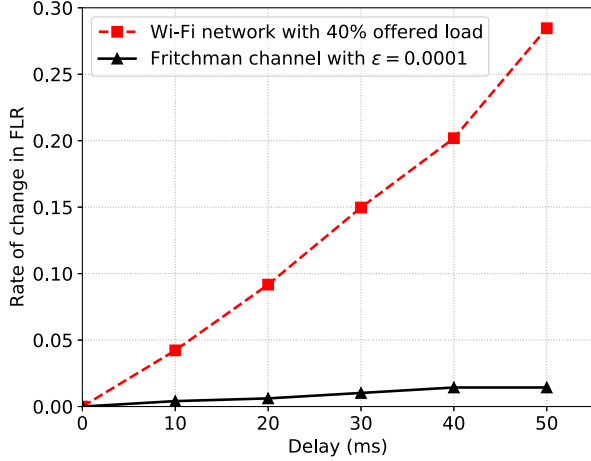


Fig. 27. Rate of change in average FLR vs. network delay.

TABLE III
AVERAGE FLR OF BEST-FIXED (B, N) AND OUR NETWORK-ADAPTIVE SCHEME ACROSS DIFFERENT DELAYS FOR WI-FI NETWORK WITH 40%-CAPACITY OFFERED LOAD AND THREE-PHASE FRITCHMAN CHANNEL WITH $\epsilon = 0.0001$

Best-fixed (B, N) coding	Network-adaptive coding with delay		
	10 ms	30 ms	50 ms
0.0105639	0.00642105	0.00708310	0.00791413

(a)

Best-fixed (B, N) coding	Network-adaptive coding with delay		
	10 ms	30 ms	50 ms
0.0127119	0.00135734	0.00136565	0.00137119

(b)

change in FLR with respect to the FLR with delay 0 ms reached 1.4% at delay 50 ms. This change is not significant compared to 28.4% reached in the Wi-Fi network when the offered load was 40%.

The difference between the Wi-Fi network and Fritchman channel is due to the difference in the number of transitions in the coding parameters that has to be made in one hour. In the Wi-Fi network, our algorithm decides 294 transitions compared to the 165 transitions to be made by the Fritchman channel.

There are also differences in burst erasures observed as shown in Fig. 5 and Fig. 19. Hence, not only is the number of transitions different, but also the distribution of coding parameters used is different.

It is important to note that while transitioning to new coding parameters, we send only $T + 1$ packets encoded using two different coding parameters. This helps to guard these packets by the old and new encoder. In a highly dynamic network environment, we only double the bandwidth for $T + 1$ packets and only during transitions, e.g., for 294 times in Wi-Fi network environment with 40% offered load, i.e., $294 \times (T + 1) (= 11) = 3234$ additional packets are transmitted, which is an additional 0.9% of the number of packets that is to be transmitted. Therefore, our network-adaptive algorithm is not inducing significant overhead during transitions.

To try decreasing the number of transitions, we can increase the estimation window size L , based on which we interleave

Algorithm 1 as described in Section IV, from 1000 to 2000. Intuitively, this would decrease the number of transitions, but will increase the average FEC redundancy. The question is: will this increase in FEC redundancy outperform the decrease in redundancy due to fewer transitions? After attempting the experiment, we observe the number of transitions decreased from 294 to 142, however, the total amount of data transmitted increased by 20%. Hence, decreasing number of transitions does not necessarily result in a decrease in total redundancy, since FEC redundancy increases.

Lossy Feedback Channel: As observed in Fig. 27, we may think that a lossy feedback channel may also defer the transitions to new coding parameters at the sender, which will lead to a degraded performance. However, this may occur to a lower extent. A lossy feedback channel may/may not delay the update of the new coding parameters, because it depends on whether we get a loss during a transition in the coding parameters or not. Since the number of transitions is lower compared to the feedback responses sent, the probability of losing the feedback response that has new coding parameters is low. Also, if we do lose a transitioning feedback packet, the receiver will continue sending the feedback response over the channel again. Basically, to get a degraded behavior we need to have lost feedback packets every transition to see a significant change in performance. This was unlikely to occur in our experiments, hence, we did not observe a change in the loss rate of recovered packets.

We performed experiments by introducing losses to the feedback channel using the three-phase Fritchman channel varying $\epsilon = 0.0001$ to 0.001. In addition, in the Wi-Fi network setting, we did the same by introducing losses to the feedback channel using the same erasure pattern for the forward channel. For both the experiments, we observe no change in FLR. Hence, we concluded that having a lossy feedback channel does not affect the performance of our algorithm in our performed experiments.

VII. CONCLUSION AND FUTURE WORK

In this paper, we design a network-adaptive FEC streaming scheme. Our scheme (i) estimates the coding parameters of streaming codes using a network-adaptive algorithm to correct both burst and arbitrary network packet losses, and (ii) explicitly construct low-latency optimal streaming codes over GF(256) for $T \leq 11$.

$O(T^3)$ is the time complexity defining the computation bottleneck of our network-adaptive streaming scheme. This upper bound bottleneck is due to the complexity of decoding a length- $(T + 1)$ block code using Gauss-Jordan elimination [34]. To be explicit, the computation bottleneck of our network-adaptive streaming scheme is close to $O(D^3)$ where D is the average number of packets lost in a sliding window of $T + 1$ packets. Therefore, the bottleneck is due to the average complexity of decoding the lost packets in a sliding window of size $T + 1$ packets⁴.

In simulated and real-world experiments, our results reveal that our network adaptive streaming scheme significantly

⁴Using Gauss-Jordan elimination method would resolve D unknowns in D linearly independent equations with a complexity of $O(D^3)$.

outperforms non-adaptive and uncoded ones in terms of FLRs and PESQ scores. We also highlight the advantage of using our network adaptive scheme compared to MDS-adaptive schemes. Additionally, we feature factors that affect the performance of our scheme in our experiments positively, such as encapsulating parity and multimedia frames, and negatively, such as network delay.

There are several interesting directions for future investigation, such as (i) finding the largest T such that optimal streaming codes exist over GF(256) remains open, (ii) obtaining theoretical performance bounds for our streaming codes over GF(256) under well-known Markov models, (iii) exploring the interplay between our adaptive streaming scheme, which adjusts the coding rate in real time and existing congestion control algorithms that adjust the sizes of streaming messages in real time. (iv) Also, machine learning techniques could be used to develop new network-adaptive algorithm, such as [35] to mitigate the delay effect on the performance.

REFERENCES

- [1] R. G. Gallager, "Low density parity check codes," *IRE Trans. Inf. Theory*, vol. 8, pp. 21–28, 1962.
- [2] D. J. C. MacKay and R. M. Neal, "Near shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 33, no. 6, pp. 457–458, 1997.
- [3] M. Luby, "LT codes," in *Proc. 43rd Annu. IEEE Symp. Foundations Comput. Sci.*, Nov. 2002, pp. 271–280.
- [4] A. Shokrollahi, "Raptor codes," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2551–2567, Jun. 2006.
- [5] ETSI, "Digital video broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for broadcasting, interactiv services, news gathering and other broadband satellite applications," ETSI, Tech. Rep. 1: DVB-S2 ETSI EN 302 307-1, Nov. 2014. [Online]. Available: <https://standards.globalspec.com/std/1674896/etsi-101-545-3>
- [6] ETSI, "Digital video broadcasting (DVB); Transport of MPEG-2 TS Based DVB Services Over IP Based Networks," ETSI, Tech. Rep. ETSI TS 102 and 034, Apr. 2016. [Online]. Available: <https://standards.globalspec.com/std/10004569/ts-102-034>
- [7] L. Yuan, H. Li, and Y. Wan, "A novel UEP fountain coding scheme for scalable multimedia transmission," *IEEE Trans. Multimedia*, vol. 18, no. 7, pp. 1389–1400, Jul. 2016.
- [8] X. Yang, C. Zhu, Z. G. Li, X. Lin, and N. Ling, "An unequal packet loss resilience scheme for video over the internet," *IEEE Trans. Multimedia*, vol. 7, no. 4, pp. 753–765, Aug. 2005.
- [9] V. Stankovic, R. Hamzaoui, and Z. Xiong, "Efficient channel code rate selection algorithms for forward error correction of packetized multimedia bitstreams in varying channels," *IEEE Trans. Multimedia*, vol. 6, no. 2, pp. 240–248, Apr. 2004.
- [10] A. Hameed, R. Dai, and B. Balas, "A decision-tree-based perceptual video quality prediction model and its application in FEC for wireless multimedia communications," *IEEE Trans. Multimedia*, vol. 18, no. 4, pp. 764–774, Apr. 2016.
- [11] A. Badr, A. Khisti, W.-T. Tan, and J. Apostolopoulos, "Perfecting protection for interactive multimedia: A survey of forward error correction for low-delay interactive applications," *IEEE Signal Process. Mag.*, vol. 34, no. 2, pp. 95–113, Mar. 2017.
- [12] 3GPP TS 26.346, "Multimedia Broadcast/Multicast Service (MBMS); Protocols and Codecs," Tech. Rep., Sep. 2010. [Online]. Available: <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1452>
- [13] M. Watson, T. Stockhammer, and M. Luby, *Raptor Forward Error Correction (FEC) Schemes for FECFRAME*. Aug. 2012. [Online]. Available: <https://tools.ietf.org/html/rfc6681>
- [14] V. Roca and B. Teibi, *Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Schemes for FECFRAME*. Jan. 2020. [Online]. Available: <https://tools.ietf.org/html/rfc8681>
- [15] J. Wang and D. Katabi, "ChitChat: Making video chat robust to packet loss," MIT Comput. Sci. Artif. Intell. Lab., Cambridge, MA, USA, Tech. Rep. MIT-CSAIL-TR-2010-031, Jul. 2010.
- [16] S. Holmer, M. Shemer, and M. Paniconi, "Handling packet loss in WebRTC," in *Proc. IEEE Int. Conf. Image Process.*, Sep. 2013, pp. 1860–1864.
- [17] H. Chen, X. Zhang, Y. Xu, Z. Ma, and W. Zhang, "Efficient mobile video streaming via context-aware raptorq-based unequal error protection," *IEEE Trans. Multimedia*, vol. 22, no. 2, pp. 459–473, Feb. 2020.
- [18] J. Wu, B. Cheng, and M. Wang, "Improving multipath video transmission with raptor codes in heterogeneous wireless networks," *IEEE Trans. Multimedia*, vol. 20, no. 2, pp. 457–472, Feb. 2018.
- [19] T. Huang, P. Huang, K. Chen, and P. Wang, "Could Skype be more satisfying? A QoE-centric study of the FEC mechanism in an Internet-scale VoIP system," *IEEE Netw.*, vol. 24, no. 2, pp. 42–48, Mar.–Apr. 2010.
- [20] S. L. Fong, A. Khisti, B. Li, W.-T. Tan, X. Zhu, and J. Apostolopoulos, "Optimal streaming codes for channels with burst and arbitrary erasures," *IEEE Trans. Inf. Theory*, vol. 15, no. 7, pp. 4274–4292, Jan. 2019.
- [21] M. N. Krishnan and P. V. Kumar, "Rate-optimal streaming codes for channels with burst and isolated erasures," in *Proc. IEEE Int. Symp. Inf. Theory*, Jun. 2018, pp. 1809–1813.
- [22] G. Joshi, Y. Kochman, and G. W. Wornell, "On playback delay in streaming communication," in *Proc. IEEE Int. Symp. Inf. Theory*, 2012, pp. 2856–2860.
- [23] G. Joshi, Y. Kochmanand, and G. W. Wornell, "The effect of block-wise feedback on the throughput-delay tradeoff in streaming," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2012, pp. 227–232.
- [24] D. Malak, M. Médard, and E. M. Yeh, "Tiny codes for guaranteeable delay," *IEEE J. Sel. Areas Commun.*, vol. 37, pp. 809–825, Apr. 2019.
- [25] A. Cohen, D. Malak, V. B. Bracha, and M. Médard, "Adaptive causal network coding with feedback," *IEEE Trans. Commun.*, vol. 68, no. 7, pp. 4325–4341, Jul. 2020.
- [26] S. L. Fong *et al.*, "Low-latency network-adaptive error control for interactive streaming," in *Proc. Assoc. Comput. Mach. Multimedia*, Oct. 2019, pp. 438–446.
- [27] ITU, "Recommendation P.862.2," ITU, Tech. Rep., Nov. 2007. [Online]. Available: <https://www.itu.int/rec/T-REC-P.862.2>
- [28] J. Valin, K. Vos, and T. Terriberry, *Definition of the opus audio codec*, 2012, Sep. [Online]. Available: <https://tools.ietf.org/html/rfc6716>
- [29] ITU, "Recommendation G.711," ITU, Tech. Rep., 1998. [Online]. Available: <https://www.itu.int/rec/T-REC-G.711>
- [30] ITU, "Recommendation G.714," ITU, Tech. Rep., May 2003. [Online]. Available: <https://www.itu.int/rec/T-REC-G.714-200305-I/en>
- [31] T. Stockhammer and M. Hannuksela, "H.264/AVC video for wireless transmission," *IEEE Wirel. Commun.*, vol. 12, no. 4, pp. 6–13, Aug. 2005.
- [32] B. D. Fritchman, "A binary channel characterization using partitioned markov chains," *IEEE Trans. Inf. Theory*, vol. 13, no. 2, pp. 221–227, Apr. 1967.
- [33] O. Hohlfeld, R. Geib, and G. Hasslinger, "Packet loss in real-time services: Markovian models generating QoE impairments," in *Proc. 16th Int. Workshop Qual. Service*, Jun 2008, pp. 239–248.
- [34] R. W. Farebrother, *Linear Least Squares Computations*. New York, NY, USA: Marcel Dekker, Inc., 1988.
- [35] S. Cheng, H. Hu, X. Zhang, and Z. Guo, "DeepRS: Deep-learning based network-adaptive FEC for real-time video communications," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2020, pp. 1–5.