Prioritized Information Recovery for Wireless Link-Layer Communication

Sohraab Soltani and Hayder Radha
Department of Computer Science, University of Texas at Tyler, Email: soltanis@msu.edu
Department of Electrical & Computer Engineering, Michigan State University, Email: radha@egr.msu.edu

Abstract—In this paper we develop Prioritized Automatic Code Embedding (PACE) link-layer protocol to achieve preferred data recovery order across connections, while maintaining stable and reliable data flows over wireless networks. We classify link-layer traffic arrivals into different priorities based on the packet delay constraint and the distortion associated with the loss of that packet. The traffic arriving in each priority class is modeled as a poisson process. Consequently, we formulate the link-layer buffer as a multiclass M/G/1 priority queuing system where the decoding process (service process) of the PACE buffer is captured by a nonhomogeneous geometric distribution. This formulation enables the determination of an optimal dynamic decoding schedule for heterogeneous link-layer traffic arrivals. PACE employs a novel rate-adaptive Low Density Parity Check (LDPC) channel code for error recovery. We model the underlying LDPC decoding process and use it to determine an optimal code selection policy for maximal bandwidth utilization. We demonstrate experimentally that PACE reduces the throughput-delay cost by 20%-70% in comparison with the IEEE802.11 ARQ and Hybrid ARQ (HARQ) protocols. Further, the PACE protocol achieves 20%-60% improvement in channel bandwidth utilization and 2-10dB PSNR gain in realtime video quality.

I. INTRODUCTION

To achieve superior link-layer wireless communication one need to target the following critical objectives: (i) achieving sustained traffic stability (maintaining continuous realtime flow under delay constraints), (ii) ensuring maximal reliability and throughput, (iii) exploiting side-information for channel estimation/prediction, and (iv) interacting with the higher layers for prioritized communication and improving quality of service. We believe that these objectives represent a viable and necessary set of requirements for wireless link-layer communication outlined above.

In recent years, various decoding schedulers [2]–[4] have been proposed to improve the overall throughput in wireless link-layer communication. The majority of these efforts either consider the standard ARQ approach [5] or the Hybrid ARQ (HARQ) schemes [6] such as: diversity combining [7], code combining [8], and incremental redundancy. Approaches outlined in [6]–[8] are based on code puncturing methods [9]. These techniques assume (i) the complete knowledge of channel quality; and (ii) a slow-fading wireless environment. Under such assumptions, a decoder scheduling scheme only attempts to balance the quality of service for heterogeneous traffic requirements. This design strategy only targets quality of service and largely ignores the other objectives of wireless link-layer communication outlined above.

In this paper, we build on the ACE framework [1] to achieve preferred data recovery order across connections, while maintaining stable and reliable data flows in wireless networks. Under the proposed Prioritized ACE (PACE) framework, our ACE based stable-and-reliable link-layer will employ a novel rate-adaptive Low Density Parity Check (LDPC) channel codes while interacting with the higher layers to provide a dynamic decoder scheduling service over varying wireless channel condition. Specifically, we develop a LDPC decoding model to capture the decoding process for link-layer traffic and use it to determine an optimal code selection strategy for maximal bandwidth utilization. Further, we find an optimal code embedding rate under the PACE framework to jointly meet the reliability, stability, and delay constraints of the wireless link-layer communication. We classify heterogeneous link-layer traffic arrivals into different priority classes based on packet delay constraints and the distortion suffered. The traffic arriving in each priority class is modeled as a poisson process. Consequently, we formulate the link-layer buffer as a multiclass M/G/1 priority queuing system where the decoding process (service process) of the PACE buffer is captured by nonhomogeneous geometric distribution [10]. Given the link-layer buffer model and the LDPC decoding model, we determine the optimal dynamic decoder scheduling under the PACE framework. This scheduling policy is a special

1For the definitions of reliability and stability under the ACE framework refer to [1].
case of a classic scheduling problem solved by Plambeck et al. in [13] and is asymptotically optimal.

The PACE protocol incorporates the LDPC model and the dynamic scheduling policy into the original ACE protocol [1]. We show experimentally that PACE improves the performance of the ACE protocol over wireless channels with varying conditions. Our contributions are:

- Develop a LDPC decoding model used to select the code with best error-correcting capability from an ensemble of codes.
- Model the link-layer buffer as $M/G/1$ priority multiclass queuing system to determine an optimal dynamic decoder scheduling policy to achieve improved quality of service while maintaining sustained stability.
- We develop the PACE protocol that provides reliable and stable wireless communication for high demand and delay sensitive heterogeneous link-layer traffic.

We present an extensive evaluation of our proposed link-layer protocol in comparison with the IEEE802.11 ARQ and HARQ protocols under varying channel conditions over real channel traces collected on 802.11b WLAN [1]. PACE is implemented using OMNET++ network simulator [19], and an Adaptive Low parity Density Code (A-LDPC) [18]. We begin by evaluating the impact of throughput-delay tradeoff on the performance of the PACE protocol. Empirical results show that PACE has the lowest throughput-delay cost in comparison with the IEEE802.11 ARQ and HARQ protocols. For instance, PACE reduces the cost by 20%-70% (over channels with IEEE802.11 ARQ and HARQ protocols. For instance, $P_{ACE}$ protocol. Empirical results show that $P_{ACE}$ has gradual improving. Specifically, the channel is in severe condition the BER reduces to 0.02 in $\tau_1$. During $\tau_2$ and $\tau_3$ the channel BER reduces to 0.01 and 0.005 and finally the channel is error-free in $\tau_4$. We consider the following scenarios:

1) **Ideal scenario**: Under an ideal scenario all four packets are transmitted over the channel and received without errors. This gives the throughput of one and all packets are delivered to the higher layer. However, since the channel BER is non-zero in the first three transmission intervals, the odds of receiving an error-free packet are very low.

2) **IEEE802.11 ARQ**: This scheme uses a retransmission for error recovery. As illustrated in Fig 1(a), in $\tau_1$, the sender transmits $K_1$. Since the channel BER is non-zero, the received packet is erroneous. Consequently, the receiver requests for a retransmission of $K_1$ in $\tau_2$ and also in $\tau_3$ because the channel BER is still non-zero. In $\tau_4$, $K_1$ is finally delivered to the receiver without errors and passed up to the higher layer. This creates the throughput of $1 - \frac{3k}{4k} = 0.25$ since $K_2$, $K_3$ and $K_4$ are not transmitted during the four time-slot window of this simple example.

3) **Hybrid ARQ (HARQ)**: Fig 1(b) shows the performance of HARQ. In $\tau_1$, the sender encodes $K_1$ data packets with $x_1$ parity bits and creates a codeword $C_1(K_1, x_1)$. The receiver fails to decode $C_1$ and requests for a retransmission of $C_1$. In $\tau_2$, the sender retransmits $C_1$. Since the channel BER is relatively
high (BER is 0.01), the receiver cannot decode $C_1$ and requests for the second copy of $C_1$. In $\tau_3$, the receiver successfully decodes $C_1$. Consequently, in $\tau_4$, the sender transmits $C_2(x_1, x_2)$. The channel is error-free in $\tau_4$ and so the receiver decodes $C_2$ successfully. Data packets $K_1$ and $K_2$ are delivered to the higher layer. This creates the throughput of $0.25 \leq 1 - \frac{4k^2}{4k^2 + x_1^2 + x_2^2} \leq 0.5$ since $K_3$ and $K_4$ are not transmitted and furthermore, some of the channel bandwidth is consumed for the delivery of parity bits $x_1$ and $x_2$. Notice that in practice, the number of parity bits is significantly less than the number of data bits in a codeword (i.e., $x_i \ll k$).

4) PACE: Fig.1(c) shows the PACE protocol performance. In $\tau_1$, a codeword $C_1(K_1, x_1)$ is sent. A receiver that fails to decode $C_1$, stores $C_1$ in its buffer and requests for additional parity bits (hereafter type-II parity bits) for $C_1$. In $\tau_2$, the transmitter sends $M_2 = [C_2(K_2, x_2), y_1]$. The receiver uses $x_2$ to decode $C_2$ and employs type-II parity bits $y_i$ to denote additional parity for $C_j$, $j < i$ transmitted in $\tau_i$ in addition to $x_1$ to decode $C_1$. The receiver fails to decode $C_1$ and $C_2$. Since the codeword $C_2$ corresponds to the realtime data packet $K_2$, it has higher priority than $C_1$. Therefore, the receiver requests for type-II parity bits for $C_2$ rather than $C_1$. As a result, in $\tau_3$, the sender transmits $M_3 = [C_3(k_3, x_3), y_2]$. In $\tau_3$, the receiver successfully decodes $C_3$ using $x_3$ and $C_2(k_2, x_2 + y_2^i)$. Consequently, the receiver requests for type-II parity bits for $C_1$. Accordingly, in $\tau_4$, the sender transmits $M_4 = [C_4(K_3, x_4), y_1]$. The receiver decodes $C_4$ using $x_4$ and $C_1 = (K_1, x_2 + y_2^i)$ successfully. Therefore, the data packets $K_1, \ldots, K_4$ are delivered to the higher layer. The throughput under PACE is $0.75 \leq 1 - \frac{\sum_{i=1}^{4} x_i + y_1^i + y_2^i}{4k} \leq 1$.

This example demonstrates the efficacy of PACE in improving throughput-delay performance in wireless link-layer communication. However, the description above has intentionally ignored important details. For PACE to become practical we need to address the following challenges:

- Optimal Parity Allocation: PACE uses channel codes for error recovery and hence it requires embedding parity bits in every packet. Since the transmission of parity bits consumes channel bandwidth, it is important to identify the optimal allocation of parity bits in every transmission based on the channel condition and the channel code algorithm. A practical design of parity allocation for PACE has to ensure (i) high likelihood of successful decoding; and (ii) efficient utilization of the channel bandwidth.
- Dynamic Decoder Scheduling: In every transmission, PACE should perform error recovery on the packets with higher priority and hence buffer less significant ones for future recovery. To achieve compliance with such policy, it is natural to consider two complementary modes of dynamic scheduling: (i) PACE should choose the order in which arriving packets are served to guarantee the delivery of the packets to the higher layer before they expired; and (ii) to reject or drop insignificant packets when the buffer length is judged to be excessive, thereby incurring penalties or lost potential throughput.

The rest of the paper shows how to address these two challenges. To that end, in the next section, we study the behavior of the link-layer traffic under the PACE framework and formulate necessary models which provide essential tools to develop a practical design of PACE.

III. MODEL FORMULATION

In this section, we develop the following models for PACE: (i) LDPC decoding model which captures the decoding process of link-layer traffic using LDPC codes; (ii) Buffer model which describes the link-layer buffer as a multiclass priority queuing system with a single server. These models will build the framework to determine optimal code selection and dynamic decoder scheduling strategies for the PACE protocol.

A. LDPC Decoding Model

PACE employs LDPC codes for decoding link-layer packets. Our objective in this section is to formulate the LDPC decoding process to select the code with best error correcting capability to maximize bandwidth utilization. PACE uses the LDPC check matrix represented by Tanner bipartite graph shown in Fig 2. The nodes of the graph are separated into two distinctive sets which are called variable and check nodes with the degree of $d_v$ and $d_c$, respectively. The PACE protocol uses an iterative belief propagation LDPC decoder [12] which attempts to correct errors in the received packet in $m$ iterations. The following equation, by Gallager [11] shows the reduction of errors as the function of the iteration number $m$:

$$
\epsilon^{(m)} = \epsilon^{(0)} - \epsilon^{(0)} \left[ 1 + (1 - 2\epsilon^{(m-1)}d_v - 1)^{d_c - 1} \right]^{d_c - 1} \left[ 1 - (2\epsilon^{(m-1)}d_v - 1)^{d_c - 1} \right]^{d_c - 1} \quad (1)
$$

where $\epsilon^{(0)}$ represents the cross-over probability of the Binary Symmetric Channel (BSC) that the packet is transmitted over and $\epsilon^{(m)}$ is the probability of error in the packet after the $m^{th}$ iteration.

Using Equation (1), we can formulate a relationship between the number of iterations in belief propagation method $m$, LDPC parity check matrix parameters $d_v$ and
The distortion level of the received packet after \( m \) iterations approaches zero. In practice, the LDPC belief propagation decoder is designed to decode a codeword \( C_i(k_i,x_i) \) when the distortion level of the received packet approaches zero. In practice, the LDPC belief propagation algorithm is configured such that the number of iterations \( m \) and the degree of variable nodes \( d_v \) are predetermined and constant [18]. However, the degree of check nodes \( d_c \) is determined by the algorithm based on the number of available parity bits. Consequently, a successful decoding depends directly on \( d_v \). On the other hand, Equation (4) suggests that error-correcting capability of the decoder cannot exceed the upper bound \( \alpha_u(\epsilon) = \epsilon \frac{d_v}{d_c} \). So, to maximize \( \alpha_u(\epsilon_i) \), we have the following optimization problem:

\[
\arg \max_{\epsilon_i} \alpha_u(\epsilon_i) \quad \text{subject to:} \quad \epsilon_i \leq \alpha(\epsilon_i) \leq \epsilon_i(d_v - 1)(d_c - 1)^{m-1} \approx 0 \quad \text{and} \quad \epsilon_i[x_i(d_v - 1)(d_c - 1)]^{m-1} \approx 0.
\]

Solving (5) leads us to the best possible value of \( d_v \) and the maximum error-correcting capability \( \alpha^*(\epsilon_i) \). Consequently, we select a code from an ensemble of codes which has the error-correcting capability close to \( \alpha^*(\epsilon_i) \). In Section IV, we will use this code to obtain the optimal parity allocation strategy for PACE.

B. PACE Buffer Model

Each arriving packet at the PACE receiver was generated to serve a specific protocol or application in the higher layers. Depending on the type of the protocol or application, the packet is confined to a specific delay constraint. In addition, the packet contains a certain level of distortion due to the impact of wireless link-layer transmission. Under the PACE framework, each packet is classified into a specific priority class based on its delay constraint and distortion. The PACE receiver then attempts to serve a specific priority class according to dynamic decoder scheduling rules. In Section IV-B, we will provide a thorough description of packet prioritization process and dynamic decoder scheduling policy. These procedures require a complete formulation of the PACE buffer. Therefore, in this section, we develop a comprehensive queuing model for the PACE buffer. To that end, we first model the arrival process and the service distribution of each priority class.

Arrival Process and Service Distribution: We consider the Markovian channel introduced in [1]. The link-layer wireless channel is modeled as a discrete Markov chain with \( N \) states \( S_1, \ldots, S_N \). Each state \( S_i \) is a representative of a BSC with a particular BER \( \epsilon_i \) which is valued from a finite set \( F_N \) with length \( N \). \( \epsilon_i \in F_N, |F_N| = N \). In [1], and specifically Equation (4), we measured the probability of successful decoding of a packet \( C_i(k_i,x_i) \) transmitted over the channel in state \( S_1 \). Using the LDPC decoding model, we reformulate this probability measure as follows:

\[
P(E_i \leq \alpha(\epsilon_i)) = P_{E_i}(\epsilon_i|x_i) = \sum_{d=0}^{\lfloor \alpha(\epsilon_i)x_i \rfloor} e^{-\lambda_i} \frac{\lambda_i^d}{d!}.
\]

Here \( E_i \) represents the packet distortion level that has a Poisson distribution with cumulative density function \( P_{E_i}(u) \) and rate \( \lambda_i \).

The PACE receiver stores the arriving packets in its buffer when the decodings with type-I parity bits are unsuccessful. Consequently, the arrival process of packets with distortion level \( E_i \) in the PACE buffer can be modeled as a poisson process with the rate \( \lambda_i \) which is equivalent to the probability of decoding failure with type-I parity bits:

\[
\lambda_i = 1 - P_{E_i}(\epsilon_i|x_i) = \bar{P}_{E_i}(\epsilon_i|x_i).
\]
On the other hand, each arriving packet at the PACE is confined to a specific delay constraint which is independent of the packet error. Consider the situation where a packet with delay constraints \( d_j \) arrives according to a Poisson process with rate \( \lambda_j \). Notice that from the PACE buffer vantage point, the arrival process of this packet is independent of the arrival process of a packet with error \( E_i \). However, both of these processes are Poisson processes. Therefore, if a packet with delay constraint \( d_j \) and distortion level \( E_i \) is classified into priority class \( l \), then the arrival process of this priority class is a poisson process with rate \( \lambda_l \) where

\[
\lambda_l = \lambda_i + \lambda_j.
\]

A packet departs the PACE buffer either if (i) it is successfully decoded with additional type-II parity bits. In this case the data bits embedded in the packet are delivered to the higher layer; or (ii) its delay constraint expires, meaning that the packet is timed out and is dropped from the buffer. In the latter case the service distribution of the PACE decoder has zero density since the packet is dropped. In the former case, the service distribution depends on the probability of successful decoding of a packet with type-II parity bits. The probability of successful recovery of a particular packet increases as more type-II parity bits are added to the packet. Consequently, the error correction process of every packet as more type-II parity bits are added to the packet. The probability of successful recovery on decoding trial \( t \) is

\[
f_G(t) = P(\text{Successful recovery on } t\text{ trial}) = p_l^t \prod_{k=1}^{t-1} (1-p_k^l),
\]

where \( p_l^t \) measures the likelihood of successful decoding (when the channel is in state \( S_i \)) on \( t\text{th} \) decoding trial using total parity bits (type-I and type-II) of \( z_r \). Therefore, the service distribution \( G_l \) for a priority class \( l \) is \( f_G(t) \) which is a truncated \( f_G(t) \) on \( d_j \):

\[
f_G^l(t) = \begin{cases} f_G(t) & \text{if } d_j - t > 0 \\ 0 & \text{otherwise} \end{cases}
\]

The service rate of priority class \( l \) is the expected value of the density function \( f_G^l(t) \). That is,

\[
\mu_l = E[f_G^l(t)] = \begin{cases} \frac{1}{p_l} & \text{if } d_j - t > 0 \\ 0 & \text{otherwise} \end{cases}
\]

According to Equation (8), the traffic arriving in each priority class is a poisson process. On the other hand, a decoding process of each priority class has a nonhomogeneous geometric distribution given in (10). Consequently, the PACE buffer can be modeled with a multiclass \( M/G/1 \) priority queueing system with a single server. Here the arriving customers represent packets with different priority classes and the PACE decoder is the server.

Consider the situation where the packets in the PACE buffer are classified into \( 1, \cdots, L \) priority classes, which arrive according to independent Poisson processes with respective rates \( \lambda_l \) and have service distributions \( G_l \) as calculated in Equations (8) and (10). Let \( W_l \) denote priority delay, the average waiting time of a packet with priority class \( l \) in the PACE buffer before it is successfully transmitted to the higher layer. Our objective is to compute the \( W_l \).

Under \( M/G/1 \) priority queueing system, \( W_l \) the average amount of decoding time for priority class \( l \) is as follows [15]:

\[
W_l = \lambda_l E[G_l]W_1 + \frac{1}{2} \lambda_l E[G_l^2]. \quad l = 1, \cdots, L
\]

On the other hand, the priority delay of an arbitrary class \( l \) is equivalent to the amount of decoding time in the system requires upon its arrival plus the decoding time that remains for other arrivals classes that are already under the service. Therefore,

\[
W_l = W_i + V_j. \quad j \neq i, j = 1, \cdots, L
\]

Using Equations (12),(13) and some mathematical manipulations, we can solve for individual \( W_l \):

\[
W_l = \frac{\sum_{j=1}^{L} \lambda_l E[G_l^2]}{2 \prod_{j=1}^{L} [1 - \sum_{j=1}^{L} \lambda_j E[G_j]]}.
\]

In this section, we modeled the PACE buffer as a multiclass \( M/G/1 \) priority queueing system and we obtained the arrival and service rates, and the expected delay for each priority class in the buffer. In Section IV-B, we will employ this model to find an optimal dynamic decoder scheduling policy for PACE.

IV. PACE Protocol

In this section, we describe the design architecture of the PACE protocol and the functionality of each of its components. Fig. 3 illustrates the architecture of PACE sender and receiver. PACE is built on the ACE framework developed in [1]. Specifically, in Fig. 3, the dark-colored components in sender and receiver sides are those components that are either added or modified in the PACE framework. In the following sections, we present a thorough description for these components.

A. PACE Sender

As illustrated in Fig. 3(a), the PACE sender has two components. The first component is Channel State Prediction where the link-layer wireless channel condition for the next transmission interval is predicted based on the receiver feedback. Specifically, the sender uses \( \delta_{i-1} \), the receiver channel estimate for \( \tau_{i-1} \) as its prediction of the channel BER in current transmission interval \( \tau_i \); so \( \delta_i = \delta_{i-1} \).

The second component is Parity Allocation where a new codeword is generated and an appropriate number of type-II parity bits is added to a packet for the next transmission. We use the LDPC model developed in Section III-A to select an appropriate code (based on the channel condition) to find an optimal parity allocation.
In [1], we proved that under the ACE framework, there exist only one optimal code embedding rate under which the reliability and stability of link-layer traffic is ensured. Recall that the operational code embedding rate measures the fraction of data bits that are embedded in a codeword.

For instance, a codeword $C_i(k_i, x_i)$ is generated based on the code rate $R_i = \frac{k_i}{k_i + x_i}$. This finding is repeated in the following Lemma:

**Lemma 2.** An optimal solution for code embedding rate that ensures reliability and stability in wireless transmission over a channel in state $S_i$ is a unique solution and is given by:

$$R_i = 1 - \frac{\hat{\epsilon}_i}{\alpha}, \quad \epsilon_i \ll \alpha$$

where $\alpha$ is error-correcting capability of a decoder.

**Proof:** See Appendix.

Lemma 2 determines the optimal code embedding rate for a general decoder with error-correcting capability $\alpha$. However, the LDPC decoder model selects a code (when the channel BER is $\epsilon_i$) with an error-correcting capability close to $\alpha^*(\epsilon_i)$ computed in (5). In practice, PACE uses $\alpha^*(\hat{\epsilon}_i)$ to measure the expected error-correcting capability of the LDPC soft decision. This is so since PACE sender uses $\hat{\epsilon}_i$ as an estimate of the expected error in the next transmission interval $\tau_i$. Therefore, an operational optimal code embedding rate for parity allocation is

$$R_i^* = 1 - \frac{\hat{\epsilon}_i}{\alpha^*(\hat{\epsilon}_i)}.$$

PACE first determines the amount of type-II parity bits ($y_i^k$) for a particular codeword $C_k$ transmitted in some $\tau_k$, $k < i$. According to Equation (15) $C_k$ with length $n_k$ requires at most $z_i^k = \frac{n_k \epsilon_i}{\alpha^*(\epsilon_i)}$ parity bits for error recovery. However, for $C_k$, $x_k + \sum_{k+1}^{i-1} y_i^k$ parity bits are already transmitted in the previous transmission intervals $\tau_k, \cdots, \tau_{i-1}$. Thus, type-II parity bits necessary to transmit in $\tau_i$ for $C_k$ is $y_i^k = z_i^k - x_k + \sum_{l=k+1}^{i-1} y_l^k$.

After allocating type-II parity bits for codewords in the PACE buffer, PACE has $n_i = n - \sum_k b_i^k$ bits to transmit a new codeword $C_i(k_i, x_i)$ where $n$ is the maximum number of bits that can be transmitted in a packet. Similarly, the amount of parity bits necessary for encoding $C_i$ is $x_i = \frac{n_i \epsilon_i}{\alpha^*(\epsilon_i)}$, and therefore the amount of new data in $C_i$ is $k_i = n_i - x_i$. The PACE sender then transmits a new codeword along with the additional type-II parity bits in a new link-layer packet denoted by $M_i = [C_i(k_i, x_i), y_i^k]$ to the receiver.

### B. PACE Receiver

Upon the reception of the link-layer packet $M_i = [C_i(k_i, x_i), y_i^k]$, the type-II parity bits are extracted and sent to the receiver buffer. PACE receiver first attempts to decode a codeword $C_i(k_i, x_i)$ with type-I parity bits $x_i$. If the decoding is successful, the data bits $k_i$ are sent to the higher layer. But if the decoding fails, then the codeword is sent to the receiver buffer for future recovery.

Under the PACE framework, each packet is mapped to a specific priority class based on its delay constraint and distortion suffered. The PACE receiver then attempts to serve packets with a specific priority class according to dynamic decoder scheduling rules. The Classifier and Scheduler components in Fig. 3(b) implement the packet prioritization and dynamic decoder scheduling policy in the PACE receiver.

1) **PACE Classifier:** Consider a codeword $C_i(k_i, x_i)$ was transmitted over the channel with state $S_i$. Let $d_j$ represent the delay constraint of the packet with the distortion level $E_i$. The PACE Classifier assigns this packet to a priority class $l$ based on the parameters $d_j$ and $E_i$ using a classification function $g : [i, j] \to l$, where

$$g = g(h_D(\tau_i, d_j), f_G^c(t_i)) \quad i \in R^N \times d \in R^M : l \in R^{M \times N}$$

where $h_D(\cdot)$ is the delay penalty function, and $f_G^c(t)$ is the error-correcting density function given in Equation (9). The domains of packet distortion and delay constraint values are presented by $R^N$ and $R^M$ respectively.

The delay penalty function $h_D(\tau_i, d_j)$ measures the cost of postponing the delivery of data bits to the higher layer in transmission interval $\tau_i$ based on packet delay constraint $d_j$. Specifically, $h_D(\cdot)$ ranges from zero to one where the cost of dropping a packet is one (the packet is never delivered). We use the following delay penalty function [16]:

$$h_D(\tau, d_j) = a \exp \left[ b(\tau - d_j) \right],$$

where $a, b > 0$.
where \( a \) and \( b \) are normalizing coefficients.

The PACE Classifier first computes the penalty weight for each packet stored in the PACE buffer. Specifically, the penalty weight \( w_l \) for a packet with distortion level \( E_l \) and delay constraint \( d_j \) is

\[
w_l = h_D(\tau_i, d_j) \left[ 1 - f(l)(\tau_i) \right].
\]  
(18)

Notice that \( w_l \) approaches one if and only if the delay penalty function \( h_D(\cdot, \cdot) \) reaches one and \( f(l)(\cdot) \) is very small. This is an indication of a critical situation where the packet is reaching its deadline and has to be decoded immediately and at the same time the likelihood of decoding the packet with the current number of parity bits is very low. Therefore, the PACE Classifier classifies a packet with the highest penalty weight to a higher priority class. That is the priority classes \( 1, \ldots, L \) are numbered so that

\[
w_1 \geq w_2 \geq \cdots \geq w_L.
\]

2) PACE Scheduler: Accordingly to the PACE buffer model, the \( l \)th priority class has the arrival rate \( \lambda_l \), service rate \( \mu_l \), and the priority delay \( W_l \), as obtained in Equations (8), (11) and (14). Consider a deterministic fluid analogy in which fluid of class \( l \) arrives at a constant rate \( \lambda_l \) and can be drained at rate \( \mu_l \) if the PACE receiver devotes all its capacity to class \( l \). If the fluid level of class \( l \) in the buffer is \( N_l(t) \) at time \( t \), then the oldest fluid of that class arrived \( N_l(t) / \lambda_l \) time units earlier. Thus the fluid level of class \( l \) is not increasing if

\[
N_l(t) / \lambda_l \leq W_l \rightarrow N_l(t) \leq \lambda_l W_l.
\]  
(19)

Therefore, the backlog of class \( l \) in the system at time \( t \) is given by:

\[
\eta(t) = N_l(t) / \lambda_l W_l
\]  
(20)

Consequently, for a workload process \( Q(t) \) defined as

\[
Q(t) = \sum_{l=1}^{L} \mu_l N_l(t),
\]  
(21)

the associated threshold level is

\[
q = \sum_{l=1}^{L} \mu_l \lambda_l W_l.
\]  
(22)

The PACE Scheduler performs the dynamic decoder scheduling policy which has two parts: (i) Sequencing rule: the PACE receiver at each decision point \( t \) (every transmission interval), decodes the oldest packet from the class \( l \) having the largest backlog \( \eta(t) \); and (ii) Rejection rule: The PACE receiver drops a packet of class \( L \) (class \( L \) has the lowest priority significance) from its buffer if and only if \( Q(t) > q \). Plambeck et al. in [13] proved that the above scheduling policy is asymptotically optimal under the heavy traffic condition.

In this section, we described the functionality of the Classifier and Scheduler components added in the PACE receiver. In the next section, we conduct extensive performance evaluations of PACE to demonstrate the advantage of the new components added to ACE in comparison with the original ACE protocol and other leading link-layer protocols.

V. EXPERIMENT

In this section, we present performance evaluations of the PACE protocol on real wireless channel traces collected on an 802.11b WLAN [1]. Specifically, we use 41 channel traces, each with unique average BER to simulate 41 various channel conditions. We compare the performance of the PACE protocol as opposed to the ACE, HARQ, and IEEE 802.11 ARQ protocols. In particular, we first show the impact of throughput-delay tradeoff on the performance of each protocol. Then, we measure the performances of a realtime video and a non-realtime TCP applications in conjunction with these link-layer protocols. For the following experiments, all four protocols PACE, ACE, HARQ, IEEE802.11 ARQ are implemented with OMNET++ network simulator [19]. We use an Adaptive LDPC (A-LDPC) codes [18] for channel coding operations in PACE and ACE, and Reed-Solomon codes [17] for HARQ2

A. Throughput-Delay Tradeoff

The throughput-delay tradeoff suggests that a higher throughput can be achieved with a higher tolerable delay. Consequently, applications with time sensitive delays (realtime delay constraints) suffer from low throughput. In this section, we measure the cost of this tradeoff on the link-layer protocol performance. We define the throughput-delay cost \( \zeta(t) \) as follows:

\[
\zeta(t) = 1 - \theta(t) \varsigma(t),
\]  
(23)

where the \( \theta(t) \) is the throughput cumulative density function (CDF), measuring the fraction of data bits, and \( \varsigma(t) \) is the delay CDF, measuring the fraction of packets that are delivered successfully to a higher layer by the time \( t \). Notice that the throughput-delay cost function \( \zeta(t) \) approaches zero if and only if both \( \theta(t) \) and \( \varsigma(t) \) approach one at time \( t \).

Consider a packet arrives at time \( t_0 \) and has not decoded by time \( t \geq t_0 \), then its “delay” at time \( t \) is by definition \( t_n = t - t_0 \). In Fig. 4, we show the throughput-delay cost \( \zeta(t_n) \) for each protocol with respect to the packet delay \( t_n \) over six different wireless channel conditions. Specifically, in Fig. 4(a), where the channel average BER is very low (0.001), we observe that the cost reduces dramatically for the packet delay greater than one. This is so since the likelihood of packet error over this channel is very low. Consequently, most of the packets are received without errors and passed up to the higher layer immediately. However, as the channel BER increases, the likelihood of receiving erroneous packets increases. Accordingly, IEEE802.11 ARQ performs retransmissions

---

2It is important to note that HARQ protocols use hard decision algorithms. Reed-Solomon codes are known to be Maximum Distance Separable (MDS) codes and therefore used in HARQ protocols.
to deliver an error-free packet. This in turn will result in the consumption of channel bandwidth and longer packet delays. Consequently, we observe that the throughput-delay cost for IEEE802.11 ARQ does not reduce significantly as the channel BER increases. For instance, the cost of IEEE802.11 ARQ hovers around 0.7 over the channel with BER 0.018. On the other hand, unlike HARQ, PACE and ACE employ adaptive parity allocation based on the channel condition. Accordingly, we observe that the cost function of these protocols is always lower than HARQ protocol regardless of channel condition. However, for the channel with low BER, where the packet prioritization has no significant impact, we observe that PACE performs better than ACE. We observe this because PACE selects an optimal code to maximize channel bandwidth utilization. In addition, for the channels with higher BER value, PACE still outperforms other protocols since it employs packet prioritization. For instance, the PACE cost reduses below the 0.3 point for packet delays greater than two while ACE, HARQ, and IEEE802.11 ARQ have respective costs of more than 0.4, 0.5 and 0.9. Overall, the PACE protocol gains 10% – 40% reduction in throughput-delay cost.

B. Realtime and Non-Realtime Application Performance

In this section, we evaluate the performances of a realtime video and a non-realtime TCP applications in conjunction with PACE, ACE, HARQ and IEEE802.11 ARQ protocols at the link-layer. Fig. 5 illustrates the simulation setup of this section. Specifically, a TCP application sender is sitting at the wired section of the network and transmitting the TCP packets to the wireless receiver. At the same time, a realtime video sender is also transmitting video packets. The TCP and video traffic both are redirected to the receiver by the access point (AP). Using OMNET++ INET framework, we implemented this simulation setup for each link-layer protocol. Specifically, for the TCP application sender, we use the TCPGenericCliAppBase module to simulate a generic TCP application at the sender and receiver. We evaluate the average throughput of TCP packets. The average throughput measures the fraction of channel capacity that is utilized for a transmission of TCP packets. Further, for the realtime video application, we use H.264/AVC JM14.0 codec [20] to serve as video encoder/decoder. The real-time video sender encodes Stefan CIF 30fps sequence and transmits it to the receiver at 300Kbps video rate. The decoder receives the video packets and decodes them if and only if the video packets arrive before the realtime deadline. Hence, those video packets that miss the deadline are unusable for video decoding. This leads to degradation in video quality as measured by Y-PSNR. Y-PSNR is a function of the mean square error (MSE) between the values of the original and decoded Y frame pixels:

\[ Y_{PSNR} = 10 \log_{10} \left[ \frac{255^2}{|Y_{org} - Y_{dec}|^2} \right]. \]  

(24)
We repeat this simulation to evaluate the average throughput and PSNR values over 41 channel traces. Therefore, a total of 164 simulations were conducted for this section.

Fig. 6(a) shows the average throughput of the TCP packets over various channel conditions. In this figure the solid line is the channel capacity. The channel capacity gives an upper bound on the average reliable information that can be transmitted over the channel. We observe that for a very low channel BER, IEEE802.11 ARQ achieves a higher throughput than other protocols. This is so since the channel is almost error-free and IEEE802.11 ARQ uses the channel to transmit only data bits. Consequently, its throughput is near channel capacity. On the other hand, PACE and ACE protocols achieve higher throughput than HARQ due to the efficacy of adaptive parity allocation. Meanwhile, as the channel BER increases slightly, the performance of IEEE802.11 ARQ decays significantly. The ACE and HARQ protocols manage to perform relatively better than IEEE802.11 ARQ protocol but still their performances are noticeably below the channel capacity. The PACE protocol, however, achieves the average throughput within a maximum of 0.1 distance of the channel capacity. Overall, PACE achieves 20% - 60% throughput improvement over the IEEE802.11 ARQ and HARQ protocols. In addition, the PACE protocol increases the performance of ACE by 10% regardless of channel condition. This performance gain clearly illustrates the impact of code selection and the decoding scheduling policy of the PACE protocol.

Fig 6(b) suggests that the average Y-PSNR experiences a similar trend. For low BER values, the PSNR value is high and close to the ideal video quality (here 33 dB). However, the PSNR value for IEEE802.11 ARQ and HARQ degrades for channels with BER more than 0.01. The PACE protocol manages to achieve significantly better video quality regardless of channel condition. For instance, the PSNR value is around 31 dB under PACE while ACE, HARQ, and IEEE802.11 ARQ achieve 28 dB, 27 dB, and 25 dB over the channel with BER 0.007. Overall, PACE achieves 2 - 10 dB PSNR gain with respect to the IEEE802.11 ARQ and HARQ protocols. Further, it improves the ACE performance by 1 - 3 dB.

VI. CONCLUSION

In this paper, we introduced Prioritized Automatic Code Embedding (PACE) protocol which takes into consideration the stability, reliability, and delay constraints in wireless link-layer communication. We developed a LDPC decoding model to capture the decoding process of link-layer traffic and the PACE Buffer model to describe link-layer buffer as a multiclass priority queuing system. Using these models, we determined the optimal code selection for parity allocation and dynamic decoder scheduling for heterogeneous link-layer traffic. We showed experimentally that PACE significantly outperforms IEEE802.11 and HARQ protocols and improves the performance of ACE over various wireless channel conditions.

APPENDIX

(Proof of Lemma 1): Equation (1) can be simplified significantly with following approximations:

\[
\epsilon_i^{(m)} \approx \epsilon_i^{(0)} \left[ 1 + \frac{1}{2} \frac{(m-1) \epsilon_i^{(m-1)} (d_i - 1)}{1 + (d_i - 1) \epsilon_i^{(m-1)} (d_i - 1)} \right]^{d_i - 1} - \epsilon_i^{(0)} \left[ 1 + \frac{1}{2} \frac{(m-1) \epsilon_i^{(m-1)} (d_i - 1)}{1 + (d_i - 1) \epsilon_i^{(m-1)} (d_i - 1)} \right]^{d_i - 1}.
\]

where \( \xi = \frac{(d_i - 1)(d_i - 1)}{(d_i - 1)} \).

This completes the proof.

(Proof of Lemma 2): Note that this proof was first presented in [1], and repeated in here to enable coherent exposition. To find the optimal code embedding rate for reliability, we first present an upper bound on code embedding rate for stability. The reliability analysis in [1], the following optimization problem is a convex problem with the following constraints suggest that based on complimentary slackness conditions, the KKT (Karush-Kuhn-Tucker) conditions are sufficient for the points to be primal and dual optimal (zero duality gap). KKT conditions suggest that based on complimentary slackness property for strong duality, we have

\[
L_i(k_i, \lambda_1, \lambda_2) = k_i + \lambda_1 (k_i \epsilon_i + x_i (\epsilon_i - \alpha)) + \lambda_2 (n_i - k_i + x_i ) \cdot \lambda_1, \lambda_2 > 0
\]

where the primal is convex, the Karush-Kuhn-Tucker (KKT) conditions are sufficient for the points to be primal and dual optimal (zero duality gap). KKT conditions suggest that based on complimentary slackness property for strong duality, we have

\[
\lambda_1 (k_i^* \epsilon_i + x_i^* (\epsilon_i - \alpha)) = 0,
\]

\[
\lambda_2 (n_i - k_i^* + x_i^*) = 0,
\]

where \( k_i^* \) and \( x_i^* \) are the optimal transmitted amount of data and parity symbols. On the other hand, with the

© 2012 ACADEMY PUBLISHER
channel in state $S_1$, and maximum network utilization of $n_k$ symbols, the amount of transmitted data symbols is bounded above by $k_i = |C_i| R_i \leq k^*_i$, where $R_i$ is a channel coding rate. By substituting $k^*_i = n_i R^*_i$, $x^*_i = n_i(1 - R^*_i)$ and solve the above equations for $R^*_i$, we have

$$R_i \leq R^*_i = 1 - \frac{\epsilon_i}{\alpha}. \quad (31)$$

To guarantee the stability condition the buffer has to be always non-empty and at the same time should not overflow. To ensure that the buffer is always non-empty, the buffer length limiting probability distribution should not carry any weight at the value zero (i.e., $F(0,j) = 0, j = 1, 2$). The analysis in [1] shows that the link-layer buffer has the following limiting distributions

$$F(b, 1) = \beta_i (1 - e^{\lambda b}) \quad (32)$$
$$F(b, 2) = \omega_i - \frac{\beta_i(k_i - c)}{c} e^{\lambda b}, \quad (33)$$

where $\lambda = \frac{\beta_i}{2} - \frac{\omega_i}{k_i - c}$. By substituting $b = 0$ in equations (32)(33) we have $F(0, 1) = 0$ $F(0, 2) = \omega_i - \frac{\beta_i(k_i - c)}{c}$. The steady state probability of the buffer at length zero is always zero when the decoding is successful at the link layer (e.g., $Z(t) = 1$). To ensure that the buffer is non-empty when the link layer fails to decode new data, the following equality has to be satisfied

$$\omega_i = \frac{k_i - c}{k_i}. \quad (34)$$

To avoid buffer overflow, the expected number of data symbols injected into the buffer when the channel is in state $S_1$. Since the buffer has finite capacity, the following condition has to be satisfied to prevent buffer overflow

$$k_i \omega_i \leq c. \quad (35)$$

where

$$\omega_i = Pr(\text{Successful Decoding in } \tau_i) = F_{E_i}(\alpha x_i). \quad (36)$$

By using equations (36) and (35), we obtain $F_{E_i}(\alpha x_i) = 1 - \frac{\epsilon_i}{\alpha} \leq \frac{1}{2}$. Therefore, $x_i \leq \frac{F_{E_i}^{-1}(0.5)}{\alpha} = \frac{\lambda x_i}{\alpha} = \frac{|C_i|}{\alpha}$. Correspondingly, by substituting $x_i = |C_i| (1 - R_i)$, the lower bound of code embedding rate is

$$R_i \geq 1 - \frac{\epsilon_i}{\alpha}. \quad (37)$$

Equations (31) and (37) suggest that an optimal solution for code embedding rate that ensures reliability and stability conditions is a unique solution:

$$R^*_i = 1 - \frac{\epsilon_i}{\alpha}, \quad \epsilon_i \ll \alpha \quad (38)$$

**REFERENCES**


Sohrab Soltani received the Ph.D. degree from Michigan State University in 2009, the M.S. degree from Michigan State University in 2006, and the B.S. degree (with honors) from Shiraz University in 2002 (all in Computer Science). Currently, he is an Assistant Professor of Computer Science at University of Texas at Tyler (UTT) and the Director of the Stochastic Wireless and Visual Information Laboratory at UTT. His research interest includes wireless communication, wireless networking, stochastic modeling, and channel coding.

Hayder Radha received the Ph.M. and Ph.D. degrees from Columbia University in 1991 and 1993, the M.S. degree from Purdue University in 1986, and the B.S. degree (with honors) from Michigan State University (MSU) in 1984 (all in electrical engineering). Currently, he is a Professor of Electrical and Computer Engineering (ECE) at MSU, the Associate Chair for Research and Graduate Studies of the ECE Department, and the Director of the Wireless and Video Communications Laboratory at MSU. Professor Radha was with Philips Research (1996-2000), where he worked as a Principal Member of Research Staff and then as a Consulting Scientist in the Video Communications Research Department. He became a Philips Research Fellow in 2000. Professor Radha was a Distinguished Member of Technical Staff at Bell Laboratories where he worked between 1986 and 1996 in the areas of digital communications, image processing, and broadband multimedia.

Professor Radha served as Co-Chair and Editor of the ATM and LAN Video Coding Experts Group of the International Telecommunications Union - Telecommunications Section (ITU-T) between 1994-1996. He currently serves on the Editorial Board of IEEE Transactions on Multimedia and the Journal on Advances in Multimedia. He is an elected member of the IEEE Technical Committee on Multimedia Signal Processing and the IEEE Technical Committee on Image and Multidimensional Signal Processing (IMDSP). He also served as a Guest Editor for the special issue on Network-Aware Multimedia Processing and Communications of the IEEE Journal on Selected Topics in Signal Processing. Professor Radha is a recipient of the Bell Labs Distinguished Member of Technical Staff Award, the AT&T Bell Labs Ambassador Award, AT&T Circle of Excellence Award, the MSU College of Engineering Withrow Distinguished Scholar Award for outstanding contributions to engineering, and the Microsoft Research Content and Curriculum Award. He is also a recipient of National Science Foundation (NSF) Theoretical Foundation, Network Systems, and Cyber-Trust awards. His current research areas include wireless communications and networking, sensor networks, network coding, video coding, stochastic modeling of communication networks, and image and video processing. He has more than 100 peer-reviewed papers and 30 US patents in these areas.