Optimizing FEC Transmission Strategy for Minimizing Delay in Lossless Sequential Streaming

Sanjeev Mehrotra, *Senior Member,* Jin Li, *Senior Member,* and Ying-zong Huang *Student Member*

Abstract

As Cloud Computing is taking off, the presence of high performance interactive Internet applications is exploding. By nature, these applications require responsive client-server data exchange and lossless, in-order delivery. Previous work has shown that by using forward error correction (FEC), it is possible to reduce the data streaming latency caused by retransmissions of lost packets. However, the prior schemes only send FEC packets when there are no original packets pending transmission. In this paper, we further expand the hybrid FEC-ARQ protocol and show that sometimes, the transmission latency can be further reduced by preempting original data packets with FEC packets. We have formulated the decision of whether to send new original data packets, FEC packets, or resend original data packets as a transmission policy. An optimal transmission policy is selected to minimize the delay experienced by the application subject to a constraint on the amount of overhead. By using this optimal policy, we significantly improve the delay performance over straightforward FEC schemes while controlling the amount of overhead due to FEC.

S. Mehrotra and J. Li are with Microsoft Research, Redmond, WA, 98033, USA, e-mail: {sanjeevm, jinl}@microsoft.com.

Y. Huang is with MIT, Cambridge, MA, USA, e-mail: zong@mit.edu.

Optimizing FEC Transmission Strategy for Minimizing Delay in Lossless Sequential Streaming

I. Introduction

With the rapid penetration of broadband networks and the rise of Cloud Computing, online interactive applications are flourishing. Web-based applications, which use the browser as a thin client, are proliferating as the software can be installed and maintained on centralized servers as opposed to distributing the software on potentially millions of client computers. Some examples of web applications are wikis, online auctions, web-mail, and online retail sales. Software as a service (SAAS) is projected to grow to $15 billion by 2012 [1], increasing its share in the enterprise software market from 10.7% in 2007 to 18.2% in 2012. As another example, multi-player online games are seeing rapid adoption as well. Many online games have associated online communities, making them a form of social activity beyond single player games. Also, the rising popularity of Flash/Silverlight/HTML5 and Java has led to an Internet revolution providing a unified platform to deliver streaming audio, video, and other forms of interactivity to the client.

One crucial aspect that affects the user experience of an interactive software application is its responsiveness. Whenever the client sends an input (e.g keyboard/mouse commands), the requests must be sent to the server in a distant data center, which processes the incoming commands, and then sends updated data, audio, or video back to the client for rendering. The responsiveness of the application is directly related to the timely interchange of the request and the response between the client and the server.

Unlike interactive multimedia applications, such as VoIP and video conferencing, most interactive software applications operate as a state machine. Therefore, the data has to be delivered losslessly and in-order so that the client and server state are in sync. TCP (Transmission Control Protocol) provides reliable and ordered delivery of content over the network and thus is commonly used. However, TCP and its variants (such as [2]–[4]) were designed from the start to handle bulk data transfer (file download / static web page download), and therefore optimizes throughput, while making no attempt to minimize the delay experienced by individual packets. Its use of packet retransmission upon loss (ARQ) leads to higher delay on individual packets when loss is present. This can lead to poor performance for interactive applications.

A. Related Work

There has been a lot of previous work on improving the quality of real-time media (audio/video) applications. However, we note that there are different delay and reliability requirements when dealing with interactive software

TABLE I
MEDIA STREAMING VS INTERACTIVE APP VS FILE DELIVERY

Media streaming	Interactive App	File delivery
Strict deadline	Delay sensitive	No deadline
Best effort	Reliable delivery	Reliable delivery
No ordering	In-order	In-order
Low delay	Delay-aware	Delay agnostic

applications. We summarize the differences of the quality of service requirements between interactive software applications vs. that of file delivery and media streaming in Table I-A.

Note that many semi-interactive media applications with an end-to-end (E2E) delay tolerance of multiple-seconds, such as video on demand (VOD) or internet TV, actually belong to the same category as file delivery. As these applications can typically build up a client buffer of several seconds worth of content and simply use retransmissions to combat packet losses, the traditional TCP algorithms work fine. In addition modern streaming solutions such as Smooth Streaming start with small initial buffers [5], [6] and avoid initial startup latencies. However, interactive applications, such as online game, remote desktop, and web applications, have a end-to-end delay requirement of only hundreds of milliseconds. With such stringent E2E delay requirements, the increased latency caused by a retransmission becomes significant.

Although we can attempt to use delay sensitive congestion control strategies such as [7] to minimize congestion induced packet loss, non congestion-induced packet loss is still fairly prevalent in the internet, especially on wireless links where the signal-to-noise ratio (SNR) may be sometimes low, or in long distance, cross continent Internet links [8], [9]. In addition, when the interactive application is sharing a bottleneck with a flow using a loss based congestion control such as TCP, it may experience congestion induced packet loss as well [10], [11].

An effective technology to reduce the delay caused by packet loss is Forward Error Correction (FEC), that is, sending additional encoded packets to protect the data packets. In congestion induced packet loss cases, sending FEC packets will result in the reduction of the rate that can be used to send source (innovative data) packets since the overall rate into the network has to be held constant in order to avoid further congestion induced loss. However, a lower source rate with lower delay may still be preferable for many interactive cases where the source has some level of rate control.

FEC has been promoted widely in media (audio and video) streaming applications, e.g., in [12]–[16], and has been used in practice in interactive (VoIP/conferencing scenarios) and in multicast / broadcast media distribution. There has also been work in optimizing proactive retransmissions for media transmission so as to minimize distortion subject to a rate constraint, where distortion is caused by lost packets as well as packets which exceed their deadline [17]. However, the use of proactive retransmissions or FEC in protecting reliable lossless data (such as in interactive software or web applications) has been less common.

In [18], Rizzo and Vicisano have used FEC to support reliable multicast thereby reducing the bandwidth usage

needed. In [19], Sundararajan *et. al.* describe how to modify TCP by using random linear codes to protect against packet loss over the network. The idea is to retain all existing TCP mechanisms for congestion control and triggering of retransmission, but apply FEC (more specifically, random linear codes across all data in the window) at the sender and receiver, thereby masking loss in the network and improving responsiveness. In [20], we have optimized FEC transmission strategy but for *lossy* online game data with a deadline constraint. However, none of the previous work has concentrated on finding an optimal transmission strategy for delay minimization for real-time lossless interactive data.

B. Contributions

This paper combines and extends a series of our previous works, such as [21] and [22], in which we have developed a hybrid FEC-ARQ protocol for optimized sequential (in-order) delivery. Our protocol is functionally compatible with TCP, though it is not packet-level compatible. Whenever a transmission opportunity arises, the protocol will either retransmit a lost packet, send a new packet (if present), or send a FEC packet. The naive approach, adopted in the earlier work [21], is to simply opportunistically send FEC packets whenever there is a "free" transmission opportunity. Moreover, the FEC packet in this work is simply a linear combinations of all unacknowledged source packets. However, in the later work [22], we showed that only sending FEC packets when there are "free" transmission opportunities is not an optimal solution. In certain situations, e.g., in networks with high packet loss ratio, or when the traffic is bursty, or in congested cases (when the maximum application rate is higher than the network capacity), it sometimes makes sense to preempt sending a new source packet with a FEC packet of previously sent source packets. Although the source packets waiting in the sender queue get delayed, the overall delay experienced by the application can be reduced. We also showed that sometimes it makes sense to only create an FEC packet of the first few packets rather than all the unacknowledged packets. We formulated the problem of figuring which packet to send as an optimized transmission policy problem, where for each transmission opportunity, we can choose to send one of three types of packets: 1) a new source packet, 2) a FEC packet, or 3) a resent packet.

Though some of this paper covers similar material as discussed in [21] and [22], this paper is able to explain the algorithm in greater detail and discuss more corner cases of the algorithm. Moreover, this paper also presents new results which validate that the proposed algorithm runs well under real network conditions by using data from a real network trace as opposed to just a simulated network channel with random loss and fixed delay. We also show more detailed experiments that evaluate how the performance of our protocol (delay, overhead, and application bitrate) are affected as a function of channel loss rate as well as burstiness of the application traffic.

The rest of the paper is organized as follows. In Sec. II, we go over the transmission strategy in detail, explaining the definitions of overhead and choice of packets and policies. The cost function is explained in Sec. II-B and the definition of overhead and computation of the overhead packets is discussed in Sec III. In Sec. IV, we show detailed simulation results to demonstrate the effectiveness of the proposed scheme.

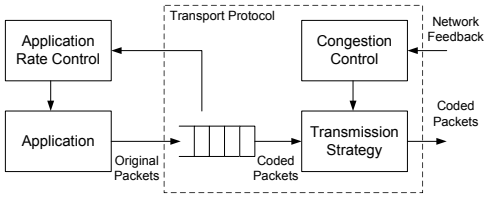


Fig. 1. Block Diagram.

II. Transmission Strategy

Fig. 1 shows a block diagram of a typical network setup of an interactive application that uses our proposed protocol. Since our protocol is functionally compatible with TCP, it may be used by any application that is currently using TCP but demands responsiveness and low delay. The sender application produces original source packets to send to the receiver. These packets typically come in a burst and consist of data which the receiver will process in order. The packets are sent to the transport module. The transport module typically has a buffer to temporarily hold the packets. The packets leave the buffer only when they have been acknowledged by the receiver. If the sending buffer is full, the sending application receives feedback of this event from the transport module and reduces its sending rate. For example, for an application that is sending audio/video, it can re-compress the audio/video at a lower bit rate. For game applications, it can reduce the game status update interval to reduce the sending rate. However, once the packets enter the transport module's buffer, they must be delivered losslessly to the receiver.

The transport module consists of two components. One is the congestion control module which estimates the available bandwidth in the communications channel, determines the current sending rate, and backs off (reduces sending rate) when congestion is detected. It tries to find a fair share of the bandwidth for the sending application while trying to minimize self congestion induced loss and queuing delay. The hybrid FEC-ARQ protocol developed in this paper can work with many existing congestion control modules, e.g., TFRC rate control. The second module is a transmission strategy module. It determines which type of packet to send at each transmission opportunity.

Since delay is the most important factor in determining the perceived user performance of interactive applications, the overarching goal for the transport module is to minimize the *expected* delay incurred by each packet while ensuring reliable in-order delivery. The delay incurred by the packets has several components – e.g., waiting time in the sender's queue, propagation delay, network queuing delay, retransmission delay, and decoding delay if a coding scheme is used. The requirement of in-order delivery can also cause additional delay as a packet may need to wait for prior missing packets to be delivered or decoded.

For the following discussion, we define *original packets* as the data packets which the application wishes to send from the sender to the receiver. For a stream with an in-order reliable delivery requirement, original packet i is defined to be *sequentially decodable* (i.e. usable) if and only if it and all prior packets $j \leq i$ are delivered or decoded. Let *sequential decodability delay* (SDD) refer to the time span between when a packet enters the sender queue (from the application) to the time it becomes sequentially decodable. This delay is important for interactive

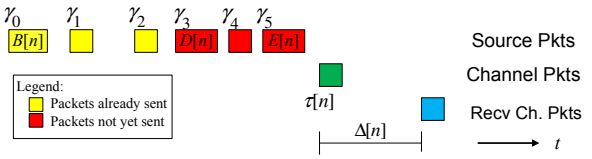


Fig. 2. Timeline - First three packets have been sent, but not necessarily decodable. Last three have not yet been sent. At time $\tau[n]$, the nth coded packet is generated which reaches the receiver after $\Delta[n]$ time.

applications.

Let *coded packets* refer to the packets that actually enter the network. These packets can be original, FEC packets, or resent packets. Let *transmission delay* be the delay sending these coded packets from the sender to the receiver. This delay consists of the network propagation delay and queuing delay. The SDD on the original packets is a function of transmission delay incurred by the coded packets as well as loss rate suffered by the coded packets and the coding strategy being used.

A. Choice of packets and policies

The transmission strategy can send one of three types of packets: original packet, FEC packet, or resent packet. The FEC packets consist of linear combinations (over a Galois field) of existing unacknowledged (undecodable) packets in the sender queue. Let $\mathbf{x}[l]$ be the lth original source packet which is represented as a vector of bytes, each of which is an element in $GF(2^8)$. Then, if $\mathbf{y}[k]$ is the kth packet sent from the sender to the receiver, it can be written as $\mathbf{y}[k] = \sum_{l=b[k]}^{e[k]} f_{k,l}\mathbf{x}[l] = \mathbf{f}_k^*\mathbf{x}$, where $f_{k,l}$ are coefficients from $GF(2^8)$. If an original packet is sent, then $\mathbf{y}[k] = \mathbf{x}[b[k]]$, for some $b[k]$ and $e[k] = b[k]$. Because of the in-order requirement, it can be shown that for FEC packets, without loss of optimality, $b[k]$ can be assumed to be the index of the first undecoded original packet in the sender queue. The transmission strategy chooses from amongst the following three transmission policies.

- Sending a new source packet without coding.
- Sending a FEC packet of only the first certain number of undecoded packets.
- Resending an already sent packet which has timed out or been negatively acknowledged.

B. Cost function used to decide amongst policies

At any given transmission opportunity, the cost that we use to decide amongst the various policies is to minimize the expected SDD. For our discussion, we define the following terms, which are shown in the timeline in Fig. 2.

- n is the current transmission opportunity.
- $B[n]$ is the index of the first unacknowledged packet in the sender queue prior to transmission n.
- $E[n]$ is the index of the last packet in the sender's queue.
- $D[n] \leq E[n]$ is the index of the first packet which has not yet been sent.
- $\tau[k]$ is the time when coded packet k leaves the sender.
- $\Delta[k]$ is the transmission delay experienced by coded packet k (propagation delay plus queuing delay).

- γ_l is the time original packet l enters the sender queue.

The expected SDD for original packet l can be written as $D_l = \sum_{\delta \in \mathcal{D}} \delta$ Prob(SDD $= \delta$), where $\mathcal{D}$ is the set of possible values for SDD given by $\tau[k] + \Delta[k] - \gamma_l$ over k. The probability of achieving this SDD is $p_l[k] - p_l[k-1]$, where $p_l[k]$ is the probability that all original packets up to and including l are decodable (i.e. packet l is sequentially decodable) using transmissions up to and including transmission k. This probability can be computed exactly with reasonable complexity as shown in [21]. This gives

$$D_l = \sum_{k=0}^{\infty} (p_l[k] - p_l[k-1])(\tau[k] + \Delta[k] - \gamma_l). \tag{1}$$

The SDD is affected by the transmission delay through the term $\Delta[k]$, the time spent in the sender queue by $\tau[k] - \gamma_l$, and the network loss and coding strategy by $p_l[k]$.

We assume that the congestion control module is able to achieve a smooth transmission rate and queuing delay [7], Thus $\tau[k+1] - \tau[k] = T$ (the time between successive transmission opportunities is relatively constant) and $\Delta[k] = \Delta$ (transmission delay is stable and approaches the network propagation plus queuing delay). Then, rearranging terms in (1), we get

$$D_l = (\tau[s_l] + \Delta - \gamma_l) + \sum_{k=s_l}^{\infty} (1 - p_l[k])T, \tag{2}$$

where s_l is the first packet transmission opportunity that comes after packet l enters the queue, that is $s_l = \min\{j : \tau[j] \geq \gamma_l\}$. We can view this expected delay in terms of waiting times. With probability 1, packet l waits until the first transmission opportunity that comes after it enters the queue plus the network transmission delay. With probability $1 - p_l[k]$ it waits an additional time of T for the next transmission opportunity. At a given transmission opportunity n for M original packets, γ_l and $\tau[s_l]$ are the same for all transmission policies. We can remove these common terms to obtain the cost function to be optimized as

$$C = \sum_{l=0}^{M-1} \sum_{k=\max(s_l, n)}^{\infty} (1 - p_l[k]). \tag{3}$$

To simplify further, we only consider source packets starting from $l = B[n]$ (all other packets have already been decoded) and ending at $E[n]$ which is the last packet entering the sender queue. We could also consider other packets past $E[n]$ that will enter the sender's queue, but this will be application-specific. For each packet n, we only consider certain terms in the summation over k. For packets which currently have non-zero probability of decodability ($p_l[n-1] \neq 0$), we only consider the first term in the summation, and for original packets which have $p_l[n-1] = 0$, we look at the first L_l terms which is defined to be the expected time till p_l becomes non-zero. This gives,

$$C \approx \sum_{l=B[n]}^{D[n]-1} (1 - p_l[n]) + \sum_{l=D[n]}^{E[n]} \sum_{k=n}^{n+L_l-1} 1. \tag{4}$$

L_l can also be estimated as the expected number of transmission opportunities needed to successfully deliver all packets prior to original packet l. L_l can be computed using the current expected number of missing packets Q_n

and the current loss estimate ϵ as

$$L_l = \frac{Q_n + l - D[n]}{1 - \epsilon}. \tag{5}$$

The expected number of missing packets can easily be computed from the probabilities p_l. If we remove common terms and simplify, we get the following cost functions for sending a FEC and an original packet respectively

$$C_{FEC} = \sum_{l=B[n]}^{D[n]-1} (1 - p_l[n]) + \frac{(E[n] - D[n] + 1)(Q_n + 1)}{1 - \epsilon},$$
$$C_{ORIG} = \sum_{l=B[n]}^{D[n]} (1 - p_l[n]) + \frac{(E[n] - D[n])Q_n}{1 - \epsilon}. \tag{6}$$

$p_l[n]$ is the new probability of sequential decodability if that packet is sent and Q_n is the new value for the expected number of missing packets up to the last packet sent – if an FEC packet is sent, the last packet sent stays at $D[n]$ and if an original packet is sent, it increases to $D[n] + 1$. Using (6), we compute the cost for each possible FEC packet (each value of $e[k] = B[n], B[n] + 1, \ldots, D[n] - 1$, with $b[k] = B[n]$) and for an original packet ($b[k] = e[k] = D[n]$) and send the one with minimum cost. The case when $b[k] = e[k] = B[n]$ is evaluating the benefit of retransmitting the first packet in the sent queue, and in cases when packets in the sent queue have timed out, the algorithm will choose such a strategy.

C. Estimating Loss Rate

The value for ϵ used by (6) is estimated using a sliding window of certain number of packets into the past. The loss for this window is computed (ϵ_W) and the overall loss rate is updated using $\epsilon \leftarrow \eta\epsilon + (1 - \eta)\epsilon_W$ using some weight η.

D. Example of cost computation

As an example, consider $\epsilon = 0.1$, $B[n] = 1$, $D[n] = 4$, $E[n] = 6$. That is, there are six packets in the burst, out of which four have been sent but not yet acknowledged as being decodable. The values for $p_l[n-1]$ would be the following.

l	1	2	3	4	5	6
p_l	0.9000	0.8100	0.7290	0.6561	0	0

Then, depending on whether we send a FEC packet which encompasses all the original source packets or whether we send a new original packet (packet 5), p_l would become

FEC	0.9656	0.9412	0.9258	0.9185	0	0
ORIG	0.9000	0.8100	0.7290	0.6561	0.5905	0.

For FEC, the expected number of missing packets with index ≤ 4 would become $Q_n = 0.0905$, and for an original packet the expected number of missing packets with index ≤ 5 would be $Q_n = 0.5000$. Using (6), we would get $C_{FEC} = 3.8838$, and $C_{ORIG} = 2.4255$. Thus, in this case between the two, we should send the original packet. As another example, suppose the loss rate is still 0.1, but now $B[n] = 1$, $D[n] = 10$, and $E[n] = 11$. That is,

almost the entire burst has been sent, but no packets have yet been acknowledged. Using the same computations as above, the costs are $C_{FEC} = 5.3622$ and $C_{ORIG} = 6.0465$, and thus here we preempt a source packet to send a FEC packet. In general, the advantage of sending a FEC packet increases as loss rate increases and as the ratio of unacknowledged packets (in-flight) to waiting packets (in sender queue) increases.

III. Overhead

The optimization presented in the previous section was simply to minimize the average packet delay subject to the current buffer conditions presented (packets waiting in the sender queue). The current buffer conditions are a function of both the application traffic pattern as well as the transmission rate. In this section, we analyze additional constraints which relate to the amount of "overhead" which we are allowed to use.

The normalized overhead (referred to as just "overhead") is defined as the number of actual packets sent on the network minus the minimum number of packets that need to be sent divided by the minimum number of packets. Since we require lossless transmission, the minimum number of packets that need to be sent is simply the number of original packets plus the number of lost packets. For example, if we wish to send 95 packets in a network with 5% packet loss, then sending 100 packets is zero overhead (since 5 out of 100 packets will be lost). If we send 110 packets corresponding to the 95 original source packets, then we say that the overhead is 0.10 (10/(95+5)), that is 10 additional packets are sent for the minimum 100 packets that need to be sent.

If the feedback on whether a packet is received or lost is accurate, we note that only using retransmissions has a overhead of zero since only those packets which are actually lost are retransmitted. For example if the client has sufficient buffer relative to the network round-trip time (RTT) (say 5 seconds of buffer with 200ms RTT), then it can simply re-request (using ARQ) the missing packets. For any reasonable loss probability, the packet will arrive within the buffered time period. For unicast scenarios with sufficient client buffer and where the server is not overloaded, this is actually the best way to deliver content.

For interactive scenarios, where the client cannot afford a significant buffer, hybrid FEC-ARQ is used to improve performance. Since there is no way to know which packets are actually lost, the use of FEC packets will result in some overhead.

In the above scenario, if the source wishes to send 95 packets in a certain unit of time (say 1 second), and if the channel allows for 110 packets, then the overhead can be at least 10% (with no reduction in source rate) and can even be higher if we allow it (at the expense of source rate). However, if the overhead constraint is set to 10%, then the allocation is fairly straightforward (simply let the application transmit at full rate and use 15 packets for FEC and ARQ). If the channel rate is reduced to 105 packets / sec, then we can achieve an overhead of at least 5%. If the constraint allows for up to 10%, then we can achieve any overhead between 5%-10%, by sacrificing source rate to achieve better delay performance. If the channel rate is further reduced (say to 90 packets/sec), and if our constraint is still up to 10% overhead, then we can use anywhere between 0-10% overhead – in this case, we have to reduce source rate even without FEC.

If we don't consider source characteristics, it may seem that simply maximizing the overhead (and minimizing

the rate) may result in minimal delay. However, this results in a low application source rate. Here we consider applications which wish to maximize their source rate by giving them a buffer (the sender queue). That is the application simply pushes as many packets as it can into a buffer of a certain size. The goal of the optimization strategy is to try to minimize the delay for all packets currently in the buffer.

A. Computing FEC Overhead

An overhead constraint is needed to ensure that the percentage of non-innovative packets (overhead) does not take more than a certain amount. This constraint is met by simply looking at estimated overhead that any given FEC packet will give. If sending that particular FEC packet results in a the constraint being violated, it is simply removed from consideration.

The overhead can be computed using a deterministic term (based upon feedback) and a probabilistic term for the in-flight coded packets (those which have not been acknowledged or timed out). At a given transmission opportunity n, let w be the number of packets that are known to the sender to have been useless by the receiver (from feedback), and let t be the total number of packets received (from feedback). We can compute the expected fraction of packets which are overhead (more than the needed amount defined as FEC packets minus lost packets) as

$$u = \frac{w + \sum_{k \in \mathcal{F}} p_{e[k]}[k-1]}{(t + |\mathcal{F}|) - (w + \sum_{k \in \mathcal{F}} p_{e[k]}[k-1])}, \tag{7}$$

where $\mathcal{F}$ is the set of in-flight coded packets, and $|\mathcal{F}|$ is the number of such packets. The probability $p_{e[k]}[k-1]$ is the probability that we were already able to decode up to $e[k]$ given transmissions up to $k-1$, and thus is the probability that the kth coded packet with ending position $e[k]$ was useless.

For any given packet that we are considering on transmitting at n, we can update the set $\mathcal{F}$, and can calculate an updated value of u. We control the amount of overhead u to below a certain threshold U_{MAX}. If sending a particular FEC packet causes u to be above this threshold, we do not consider it for transmission. We note that sending original packets for the first time and resending lost packets cannot increase u.

We believe that this definition of overhead – as a fraction of overhead packets to needed packets – is more useful than the typical definition of redundancy which is the fraction of FEC packets to source packets. For example, if the loss rate is 5% and if 5% of the total packets are FEC, then most of the FEC packets are actually used to recover lost packets, and thus there is no overhead.

IV. Experimental Results

In this section, we show the performance of the proposed hybrid FEC-ARQ protocol and transmission policy optimization. We simulate the setup as shown in Fig. 1. The notation used in the experiments is as follows. The application produces P packets of size B bits with an inter-burst gap of G seconds. This gives a maximum application source rate of $S = PB/G$ bits/sec. We assume that if the source rate exceeds network bandwidth and the sending buffer is full, the application rate control module will kick in, and excess packets will be dropped. The sending buffer size is Q bits. The congestion control module provides a transmission opportunity to send a single

TABLE II

SDD PERCENTILES (90%, 95%, 99%) FOR $Q = 16$ PACKETS, $U_{MAX} = 10$, $S = 500$KBPS, $R = 400$KBPS. L IS THE LOSS RATE, D IS THE ROUND TRIP DELAY IN SEC. TYPE SHOWS THE STRATEGY BEING USED, "NO COST" REFERS TO THE OPPORTUNISTIC FEC STRATEGY IN [21], "COST" REFERS TO FEC WITH USE OF COST FUNCTION, AND "ARQ" REFERS TO RETRANSMISSION ONLY. "LB" IS THE LOWER BOUND OBTAINED FROM AN "ORACLE" WHICH HAS INFORMATION ON EXACTLY WHICH PACKETS WILL BE LOST, AND THUS THEY ARE IMMEDIATELY RETRANSMITTED. U IS THE ACTUAL FRACTION OF OVERHEAD PACKETS.

L	D	Type	90%	95%	99%	U
0.05	0.15	ARQ	0.47	0.51	0.63	0.00
0.05	0.15	No Cost	0.43	0.47	0.51	0.11
0.05	0.15	Cost	0.37	0.41	0.51	0.11
0.05	0.15	LB	0.33	0.33	0.37	0.00
0.15	0.15	ARQ	0.63	0.71	1.01	0.00
0.15	0.15	No Cost	0.53	0.59	0.67	0.11
0.15	0.15	Cost	0.47	0.53	0.63	0.22
0.15	0.15	LB	0.37	0.39	0.43	0.00
0.15	0.40	ARQ	1.64	1.79	2.55	0.00
0.15	0.40	No Cost	0.76	0.80	1.46	0.29
0.15	0.40	Cost	0.56	0.58	0.68	0.39
0.15	0.40	LB	0.52	0.54	0.58	0.00

B-bit packet every T seconds giving a network transmission rate of $R = B/T$ bits/sec. We assume that the channel has a delay of D seconds (round trip time) and a loss rate of L. U_{MAX} is the maximum amount of overhead that is allowed.

For all the experiments, we show two figures, one is the cumulative density function (CDF) of the sequential decodability delay (SDD), and the other is the CDF of the instantaneous application bit rate defined as the number of packets from the burst that are sent divided by the spacing between the bursts. We compare the following four transmission strategies.

- The best achievable bound. This is the performance if the sender has immediate knowledge of which packets will be lost and retransmits them immediately at the next transmission opportunity.
- The strategy using only retransmission (ARQ).
- The strategy adopted in [21]. This is referred to as the "no cost" FEC or opportunistic FEC. In this strategy, an FEC packet of all unacknowledged source packets is sent whenever the sender queue is empty. Otherwise an original packet is sent.
- The cost based transmission strategy developed in this paper.

For all simulations, we set $B = 8000$ bits (1KB) and $P = 10$ packets (the burst length). We first show the achievable performance if we are allowed to send as much FEC as the network allows (set the overhead constraint $U_{MAX} = 10$) in a congested network, when $S = 500$Kbps, $R = 400$Kbps, $D = 0.15$sec, $L = 0.05$, and $Q = 16$packets. We note that the total sending rate is constrained by the network bandwidth, and the source

TABLE III

SDD PERCENTILES AND FRACTION OF OVERHEAD PACKETS FOR $Q = 16$ PACKETS, $U_{MAX} = 10$, $S = 300$KBPS, $R = 400$KBPS.

L	D	Type	90%	95%	99%	U
0.05	0.15	ARQ	0.46	0.58	0.68	0.00
0.05	0.15	No Cost	0.36	0.38	0.40	0.28
0.05	0.15	Cost	0.36	0.38	0.41	0.30
0.05	0.15	LB	0.34	0.35	0.38	0.00
0.15	0.15	ARQ	0.70	0.86	1.15	0.00
0.15	0.15	No Cost	0.41	0.54	0.66	0.18
0.15	0.15	Cost	0.46	0.50	0.62	0.24
0.15	0.15	LB	0.37	0.40	0.44	0.00
0.15	0.40	ARQ	1.67	1.77	3.10	0.00
0.15	0.40	No Cost	0.65	0.66	0.83	0.33
0.15	0.40	Cost	0.62	0.65	0.75	0.54
0.15	0.40	LB	0.58	0.61	0.66	0.00

TABLE IV

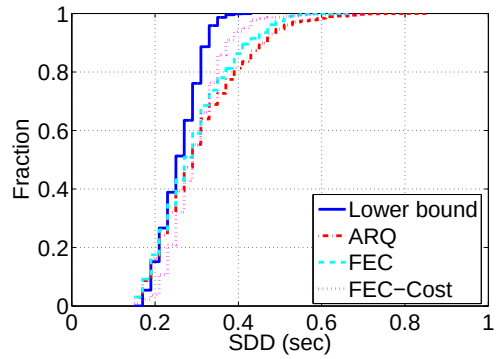
SDD PERCENTILES AND FRACTION OF OVERHEAD PACKETS FOR $Q = 16$ PACKETS, $U_{MAX} = 10$, $S = 300$KBPS, $R = 400$KBPS, USING REAL COLLECTED NETWORK TRACES. THIS PARTICULAR TRACE FROM NORTH AMERICA TO EUROPE HAD AN AVERAGE LOSS RATE OF 5% AND RTT=200MS.

L	D	Type	90%	95%	99%	U
0.05	0.20	ARQ	0.34	0.40	0.57	0.02
0.05	0.20	No Cost	0.32	0.35	0.48	0.10
0.05	0.20	Cost	0.31	0.34	0.39	0.09
0.05	0.20	LB	0.26	0.28	0.31	0.00

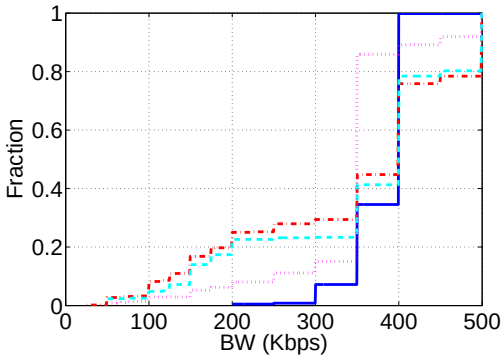
application will reduce its sending rate through notification of the sending buffer being full. The results are shown in Fig. 3 and summarized in Table IV. The 90th percentile of SDD reduces by over 14% when using the cost function vs. not using the cost function, the 95th percentile by over 12%, and the 99th percentile is about the same. The actual percentage of overhead packets is 11.2%. We also see that the application bit rate is smoother, with the application capable of delivering a bit rate of at least 300Kbps 90% of the time, rather that 70% of the time if the cost function is not used. This is due to the fact that lowering SDD results in the sender queue being emptier.

Keeping other parameters the same, if we increase Q to 32 packets, we observe that few FEC packets are sent (regardless of whether we use the cost function or not). The reason is that since $S > T$, the buffer is almost always full, and since Q is relatively large, the penalty for sending FEC packets is high since $E[n] - D[n]$ in Eqn. (6) is large.

In the second experiment, we increase the loss rate to $L = 0.15$ (see Fig. 4 and Table IV). From Table IV, we observe that cost based transmission policy reduces 90th percentile SDD by 11%, 95th percentile SDD by 10%,



(a)



(b)

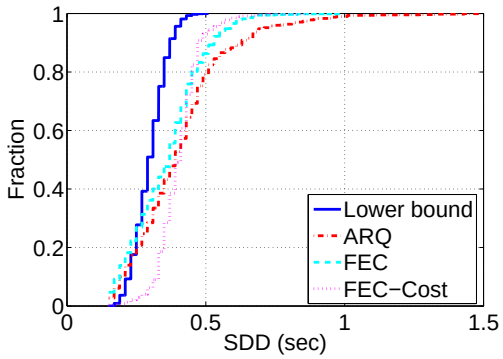
Fig. 3. Results with $Q = 16$packets, $L = 5\%$, $D = 0.15$sec. (a) CDF of SDD, (b) CDF of application bitrate.

and 99th percentile SDD by 6%. The application sending bit rate is also much smoother with the cost function than without. However, both of the schemes reach a median (50th percentile) bit rate of about 250 kbps.

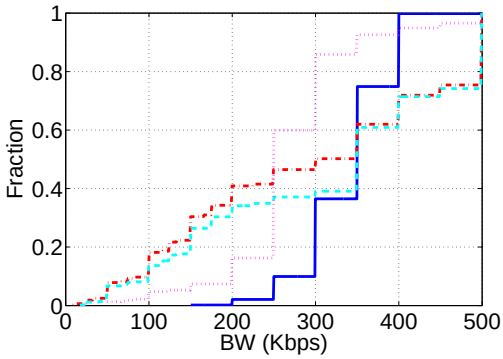
In the third experiment, we further increase delay to $D = 0.4$seconds. From Fig. 5, we see that by using the cost function, we are able to achieve a result very close to the lower bound (in terms of SDD and application bit rate). From Table IV, we can see that this comes at the expense of increasing the percentage of overhead packets to close to 40%.

In the fourth experiment, we consider the case when the network is not congested ($S < R$). We reduce the the maximum source rate S to below capacity (300 kbps) and keep other parameters the same. We observe that the cost function based transmission policy gains no advantage over the opportunistic FEC (see Fig. 6 and Table IV). Opportunistic FEC is able to achieve a result close to the lower bound especially in the high-loss, high-delay case where we have spare capacity. This confirms the results presented in [21]. This is basically a case where even without any additional constraints, we can use up to 12% overhead.

Finally, in Fig. 8, we show the effect of modifying the fraction of overhead packets allowed, U_{MAX}, in the $L = 0.15$, $D = 0.4$sec case. We see that the SDD performance keeps getting better as U_{MAX} is increased (at the expense of reducing application rate). We note that although the $U_{MAX} = 10$ case is intended to show the best



(a)



(b)

Fig. 4. Results with $Q = 16$packets, $L = 15\%$, $D = 0.15$sec. (a) CDF of SDD, (b) CDF of application bitrate.

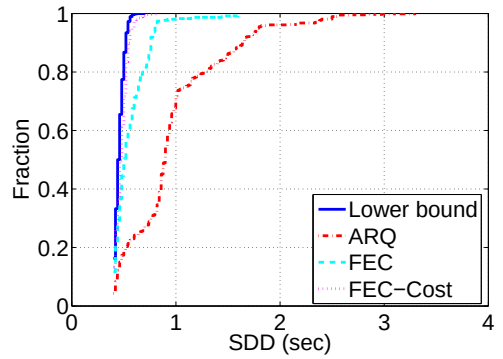
case, the actual overhead for this is only $U = 0.39$.

A. Results using real network trace

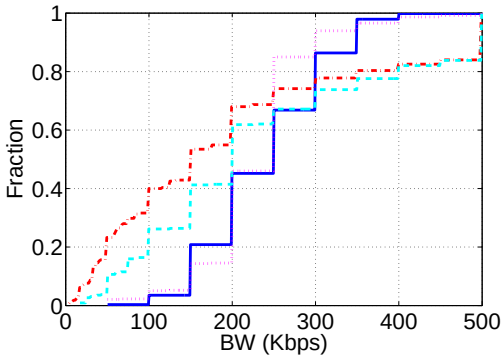
In Fig. 7 and in Table IV, we show the performance using real packet trace to drive the simulation. In the trace collection, we connect from one machine in North America to one in Europe, and measure RTT for each packet sent, and whether the packet is received or not. The average loss rate for this trace was 5% and the RTT was approximately 200ms. However, the losses were not necessarily random, and there were some periods of burst loss. We see that if we use this trace, the results are similar and we still see gains in the SDD. The SDD reduces from 600ms in pure ARQ case to 500ms in opportunistic FEC to 400ms when using the cost based optimization presented here.

B. Function of loss rate

In Fig. 9, we show how the overhead, the 99% SDD, and application bitrate are affected by loss rate (all other parameters are same as in the original experiment) for the cost based scheme and the opportunistic FEC scheme. We see that in as the loss rate increases, so does the overhead to minimize delay. This comes of course at the



(a)



(b)

Fig. 5. Results with $Q = 16$packets, $L = 15\%$, $D = 0.40$sec. (a) CDF of SDD, (b) CDF of application bitrate.

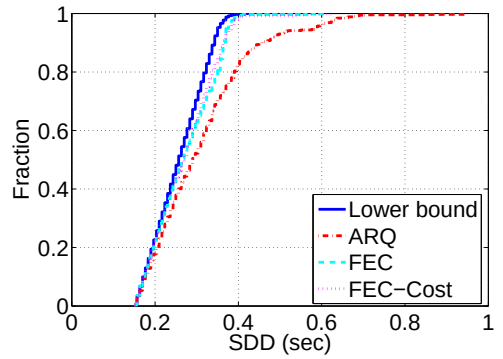
expense of application bitrate. The opportunistic FEC reverts back to a pure ARQ scheme as loss rate increases as there are very few free opportunities to send FEC packets (maximum source rate is much larger than capacity). Thus, we also see that the improvement in SDD from using the cost function increases as loss rate increases.

C. Function of burst length

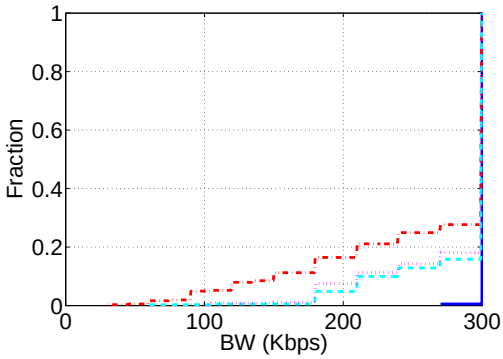
In Fig. 10, we show how the overhead, the 99% SDD, and application bitrate are affected by burst length for the cost based scheme and the opportunistic FEC scheme. We note that as burst length increases, the advantage of the cost based scheme over the opportunistic scheme also increases. This is because the cost based scheme will periodically insert FEC packets into the burst whereas the opportunistic FEC will wait for the entire burst to finish.

V. Conclusion

We have presented a hybrid FEC-ARQ protocol that is functionally compatible with TCP but optimized for low-delay data delivery. The protocol uses a cost based transmission strategy to optimally choose amongst transmission policies of sending 1) a source packet, 2) a FEC packet of the first certain number of undecoded packets, and 3) a resent packet which has timed out or been negatively acknowledged. Through extensive experimental results, we have



(a)



(b)

Fig. 6. Results with $Q = 16$packets, $L = 15\%$, $D = 0.40$sec, but source rate lowered to below capacity (300Kbps).

shown that the proposed scheme achieves better delay performance than the opportunistic FEC scheme especially for cases when the application traffic is bursty, the maximum source rate exceeds that of the network capacity, the network packet loss rate is high, and/or the network delay is high. We have also examined the performance of the scheme as a function of loss rate and burstiness of the application traffic.

REFERENCES

[1] J. Humphreys, "Worldwide virtual machine software 2008-2012 forecast," May, 2008, IDC.

[2] S. Floyd and T. Henderson, *The NewReno Modification to TCP's Fast Recovery Algorithm*, Apr. 1999, rFC 2582.

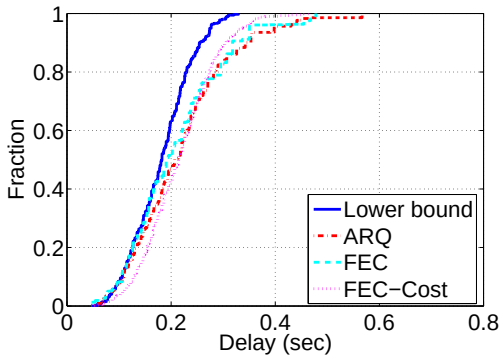
[3] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," in *INFOCOM*. IEEE, Apr. 2006, pp. 1 – 12.

[4] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: motivation, architecture, algorithms, performance," *IEEE/ACM Trans. Networking*, vol. 14, pp. 1246 – 1259, Dec. 2006.

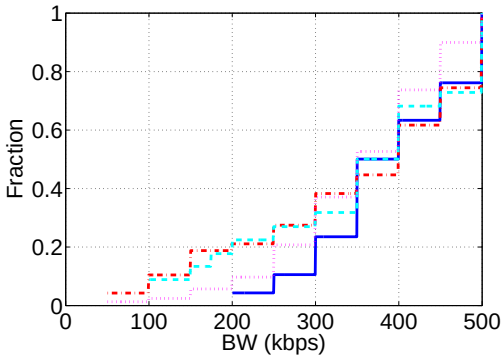
[5] Smooth streaming : The official microsoft iis site. [Online]. Available: http://www.iis.net/download/SmoothStreaming

[6] S. Mehrotra and W. Zhao, "Rate-distortion optimized client side rate control for adaptive media streaming," in *Proc. Workshop on Multimedia Signal Processing*. IEEE, Oct. 2009.

[7] S. Mehrotra, J. Li, S. Sengupta, M. Jain, and S. Sen, "Hybrid window and rate based congestion control for delay sensitive applications," in *Proc. of IEEE Globecom*. IEEE, Dec. 2010.



(a)



(b)

Fig. 7. Results with $Q = 16$packets, Loss rate and round trip time determined by results collected from real network packet traces. Average loss rate for this trace was 5% and average round-trip time was approx. 200ms.

[8] A. Mondal, R. Cutler, C. Huang, J. Li, and A. Kuzmanovic, "Surecall: Towards glitch-free real-time audio/video conferencing," in *In Proceedings of IEEE IWQoS*. IEEE, Jun. 2010.

[9] V. Vasudevan, S. Sengupta, and J. Li, "A first look at media conferencing traffic in the global enterprise," in *Passive and Active Measurement Conference (PAM)*, Apr. 2009.

[10] M. Allman, W. M. Eddy, and S. Ostermann, "Estimating loss rates with tcp," in *ACM SIGMETRICS Performance Evaluation Review*, 2003.

[11] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling tcp throughput: A simple model and its empirical validation," in *Proc. ACM SIGCOMM*, 1998, pp. 303 – 314.

[12] T. Tsugawa, N. Fujita, T. Hama, H. Shimonishi, and T. Murase, "TCP-AFEC: An adaptive FEC code control for end-to-end bandwidth guarantee," in *Packet Video Workshop*, Nov. 2007, pp. 294–301.

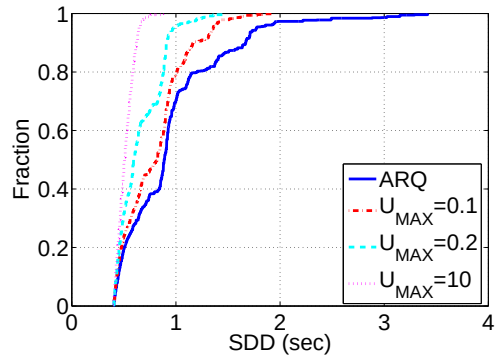
[13] R. Puri, K. Ramchandran, and A. Ortega, "Joint source channel coding with hybrid ARQ/FEC for robust video transmission," in *IEEE Multimedia Signal Processing Workshop*, Dec. 1998.

[14] P. A. Chou, A. E. Mohr, A. Wang, and S. Mehrotra, "FEC and pseudo-ARQ for receiver-driven layered multicast of audio and video," in *Proc. Data Compression Conference*. Snowbird, UT: IEEE Computer Society, Mar. 2000, pp. 440–449.

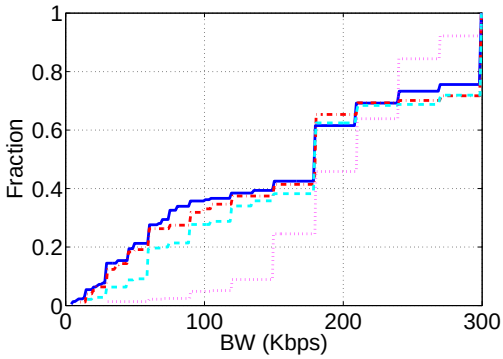
[15] J.-C. Bolot, S. Fosse-Parisis, and D. Towsley, "FEC-based error control for internet telephony," in *INFOCOM*. IEEE, 1999.

[16] I. Rhee and S. Joshi, "FEC-based loss recovery for interactive video transmission experimental study," in *International Conference on Multimedia Computing*. IEEE, 1998.

[17] P. Chou and Z. Miao, "Rate-distortion optimized streaming of packetized media," *IEEE Trans. Multimedia*, vol. 8, no. 2, pp. 390–404, Apr. 2006.



(a)



(b)

Fig. 8. Results showing effect of different U_{MAX}.

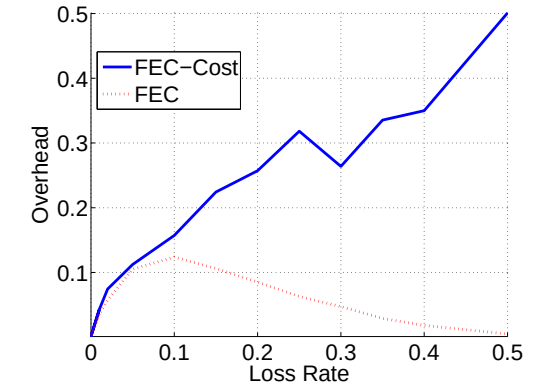
[18] L. Rizzo and L. Vicisano, "RMDP: An FEC-based reliable multicast protocol for wireless environments," *ACM SIGMOBILE: Mobile Computing and Communications Review*, vol. 2, no. 2, pp. 23–31, April 1998.

[19] J. K. Sundararajan, D. Shah, M. Medard, M. Mitzenmacher, and J. Barros, "Network coding meets TCP," in *INFOCOM*. IEEE, Apr. 2009.

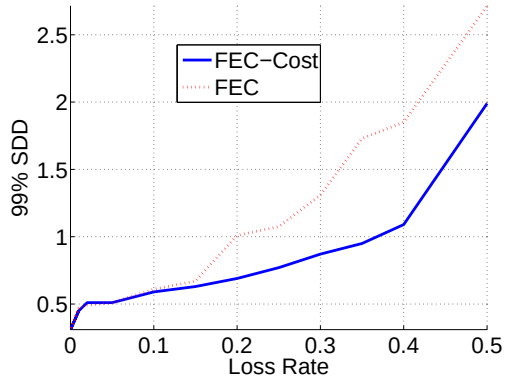
[20] C. Zhang, C. Huang, P. Chou, J. Li, S. Mehrotra, K. Ross, H. Chen, F. Livni, and J. Thaler, "Can every vote count? ensuring latency fairness for cloud-based social gaming," Microsoft Research, Tech. Rep. MSR-TR-2010-126, 2010.

[21] Y. Huang, S. Mehrotra, and J. Li, "A hybrid FEC-ARQ protocol for low-delay lossless sequential data streaming," in *Proc. Int'l Conf. Multimedia and Expo*. IEEE, Jun. 2009, pp. 718–725.

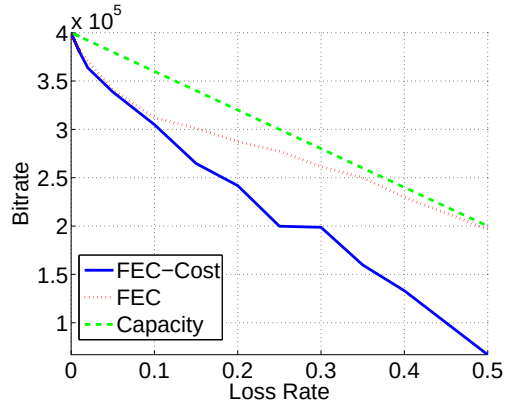
[22] S. Mehrotra, J. Li, and Y. Huang, "Minimizing delay in lossless sequential data streaming," in *Proc. Int'l Conf. Multimedia and Expo*. IEEE, Jul. 2010.



(a)



(b)



(c)

Fig. 9. (a) Overhead, (b) 99% SDD, and (c) Application bitrate as function of loss rate.

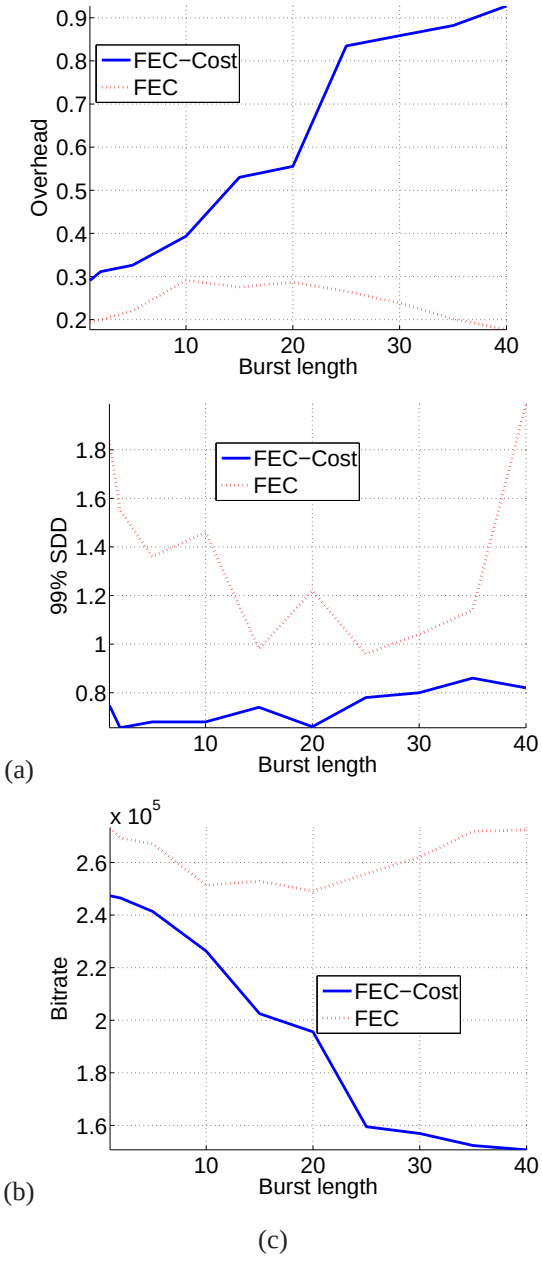


Fig. 10. (a) Overhead, (b) 99% SDD, and (c) Application bitrate as function of burst length.