Integrating Packet FEC into Adaptive Voice Playout Buffer Algorithms on the Internet

Jonathan Rosenberg, Lili Qiu, Henning Schulzrinne
dynamicsoft, Cornell University, Columbia University
jdrosen@dynamicsoft.com, lqiu@cs.cornell.edu, hgs@cs.columbia.edu

Abstract—**Transport of real-time voice traffic on the Internet is difficult due to packet loss and jitter. Packet loss is handled primarily through a variety of different forward error correction (FEC) algorithms and local repair at the receiver. Jitter is compensated for by means of adaptive playout buffer algorithms at the receiver. Traditionally, these two mechanisms have been investigated in isolation. In this paper, we show the interactions between adaptive playout buffer algorithms and FEC, and demonstrate the need for coupling. We propose a number of novel playout buffer algorithms which provide this coupling, and demonstrate their effectiveness through simulations based on both network models and real network traces.**

Keywords—**Packet voice, IP telephony, packet FEC, playout buffer adaptation**

I. Introduction

Transport of real time voice on the Internet is an increasingly important application. Internet telephony (the provision of telephony services using the Internet) has emerged as an important service, poised to replace circuit-switched telephony service in the future. However, the currently available, best effort Internet service model was never engineered to handle voice traffic, since it provides variable loss rates and delays. A number of studies have attempted to measure these parameters [1], [2], [3]. The results indicate substantial variability (depending a lot on the locations of the senders and receivers), with typical packet loss rates of 0 to 20 percent and one way delays from 5 ms to 500 ms. Unfortunately, when packet loss rates exceed 10% and one way delays exceed 150 ms, speech quality can be quite poor.

Efforts are underway to address these problems at the network layer, using differentiated services [4] and Integrated Services [5], [6]. However, we anticipate that many public and private internets will continue to provide best-effort delivery for a long time. Therefore, compensation for loss and delay end-to-end using adaptive applications which adjust their encoding parameters, such as bitrate, in response to network feedback on loss, delay, and jitter, is an important problem.

Compensating for loss using end-to-end protocols and algorithms can be done using a number of mechanisms [7][8], including local repair (interpolation of missing data using the surrounding packets) and interleaving. There has been much interest in the use of packet-level FEC for sending redundant information ahead of time to compensate for loss, based on parity codes, [9], [10], Reed Solomon codes [11], and redundant speech codecs [12][13] [14] [15]. Oftentimes, the term FEC is only applied to the traditional channel coding approaches, such as parity and Reed Solomon codes. For purposes of this paper, we define FEC as any mechanism which sends additional information along with the media stream, the purpose of which is to aid in packet recovery.

Compensating for jitter is accomplished primarily through adaptive playout buffer algorithms [16] [17] [18] [19]. (also known as adaptive playout delay algorithms). These algorithms generally work by taking some measurements on the delays experienced by packets, and updating the playout delay on a talkspurt by talkspurt basis.

The study of FEC for loss recovery, and playout buffer adaptation for jitter compensation, have proceeded independently. However, we have observed that there is a coupling between the two. All of the FEC mechanisms send some redundant information which is based on previously transmitted packets. Waiting for the redundant information results in a delay penalty, and consequently an increase in size of the playout buffers. When network loss rates are high, accepting the delay penalty for increased recovery capabilities is appropriate. However, when network loss rates are low, the FEC may not provide useful information, and increasing the playout buffer sizing to wait for it is not appropriate. The result is that playout buffer adaptation should depend on *both* FEC and network loss conditions *and* network jitter. However, existing tools that utilize FEC (such as *rat* from UCL and *freephone* from INRIA) use *decoupled* adaptation algorithms. These algorithms compute some playout delay as if FEC were absent, compute some delay needed to make use of FEC, and then combine these two delays together. These decoupled algorithms may insert insufficient delay when network loss probabilities are substantial, and too much delay when they are not, resulting in poor performance. We further observe that existing playout buffer adaptation algorithms generally aim to minimize the network losses at the expense of delay. However, some modern speech codecs can tolerate loss rates as high as 5% and still work well. As such, it would be beneficial to be able to tune these algorithms to achieve a target loss (possibly non-zero) to meet the operating range of the codecs.

In this paper, we explore these problems in more detail. In section II we demonstrate, through simple analysis, the need for coupling packet loss and jitter into playout buffer adaptation. In section III we briefly review some existing playout buffer adaptation algorithms. In section IV, we present several new algorithms which achieve the desired coupling and tuneability objectives. Section V shows the improvements offered by our algorithms through simulations. Section VI summarizes our work and discusses future directions.

II. The Coupling Effect

The main role of playout buffer adaptation algorithms is to trade loss for delay. These algorithms choose a playout delay D for each talkspurt, where D is defined as the difference between playout time and generation time for all packets in a talkspurt (although some algorithms adjust the delay mid-talkspurt to compensate for errors in buffer sizing). Increasing D results in a larger end-to-end delay, but decreases the fraction of late packets. The algorithms work by choosing a target "optimal" operating point of loss and delay, and adjusting the playout delay to come as close to the optimal point as possible. If an algorithm chooses some delay D for a talkspurt, the probability receiving it on time and playing it out to the application (which we denote the playout probability p_R) is:

$$p_R = (1-p)P[n_i < D] \tag{1}$$

where p is the loss probability for a packet, and n_i is the delay of the packet (measured as the difference between arrival time and generation time), given the packet arrives. We use the expression $P[X]$ to denote the probability of event X. This expression assumes that the network losses and delays are independent random variables.

The dependency of the playout probability p_R on the network loss probabilities and the network delay distribution is a simple multiplicative one. The result is that many playout adapatation strategies can completely ignore the loss rate p. In particular, any algorithm which tries to choose the minimum D which achieves the highest possible p_R need not consider p at all. These algorithms need only find the smallest D for which $P[n_i < D] = 1$. All of the algorithms described in [18], [19] fall into this category.

The dependency of p_R on the delay distribution and loss probabilities becomes more complex when FEC is introduced. The result is that the choice of D is more strongly influenced by p. Consider the FEC mechanisms described by Bolot and Garcia [14]. These algorithms use multiple low-bitrate versions of a packet, each version being piggybacked on a subsequent packet. A packet is played out so long as a receiver gets the original or any one of the $K-1$ redundant versions of the packet on time. If packets are lost independently with probability p, the probability of playing out a packet (p_R), given that the receiver is willing to wait for $l \leq K$ of the packets, is $1-p^l$.

This says nothing, however, about how long a receiver must wait for these l packets. Computing the probability under a delay constraint will require us to factor in the probability that the l packets arrive in time. Assume that packets are generated at the sender at intervals of Δ. A playout delay of D at the receiver means that a packet must be present D seconds after its generation at the sender in order to be used. A packet is played out if the original packet arrives on time, or if a redundant packet arrives in time. To be in time, the redundant packet's network delay, plus the amount of time between its generation and the original packet generation, must be less than the playout delay. More formally, if we assume network delays are independent from packet to packet, the probability of playing out a packet when the playout delay is D is:

$$p_R = 1 - \prod_{j=0}^{K-1} (1-(1-p)P[n_i + j\Delta < D]) \tag{2}$$

Note the more complex dependency of p_R on the network loss probability p and the network delay distribution n_i. The result is that the choice of D is more complex, and will most likely depend strongly on p. Consider once more the basic playout adaptation target: to choose the minimum D which results in the maximum possible p_R. When p is zero, D need only be set to the largest network delay possible. However, as p increases, the value of D rapidly increases up to a maximum value of $(K-1)\Delta$ plus the largest delay possible. Note that p now plays a crucial role in choosing D once FEC is enabled. A decoupled algorithm might choose D to always be $(K-1)\Delta + \max n_i$, which results in overly large playout delays when network losses are low. A different decoupled algorithm might choose D to always be some quantile of n_i. This algorithm will result in overly small playout delays when network losses are high. As such, coupling loss and delay into buffer adaptation is critical for proper loss and delay tradeoffs over a wide range of network conditions.

The astute reader might observe that if loss rates are low, the sender should not be sending FEC in the first place. Such sender adaptation might eliminate the need for coupling FEC to playout buffer adaptation at receivers. This may be true for unicast, but not for multicast. In multicast, it is likely that some receivers will experience more loss than others. The sender may choose to include FEC in order to improve reception quality for lossy receivers. In this case, receivers experiencing low loss will want to adapt in order to reduce their playout delays, since the FEC is not needed. Even in the unicast case, receiver adaptation is useful when the sender is incapable or unwilling of performing adaptation. Receiver adaptation is also useful when feedback is not sent frequently.

III. Existing Playout Buffer Algorithms

Much work has gone on in the area of playout buffers algorithms since the early 70's. Recent work in addressing the problem specifically for the Internet is found in [18] and [19]; our work builds on theirs, so we review it briefly.

Ramjee et. al. [18] propose four adaptation algorithms. Each algorithm computes, in some fashion, an estimate of the mean network delay seen up to the arrival of the i^{th} packet, $\hat{d}_i$, and a variation measure in this delay, $\hat{v}_i$. The playout delay is adjusted at the beginning of each talkspurt. If t_i is the generation time of a packet which is the first in a talkspurt, the playout time p_i of the first packet is computed as $p_i = t_i + \hat{d}_i + \mu\hat{v}_i$, with $\mu = 4$. For a subsequent packet j in the same talkspurt, its playout time is computed as $p_j = p_i + t_j - t_i$.

All four algorithms maintain a running estimate of $\hat{d}_i$ and $\hat{v}_i$, updated for each packet. All four compute $\hat{v}_i$ in the same fashion, $\hat{v}_i = \alpha\hat{v}_{i-1} + (1-\alpha)\left|\hat{d}_i - n_i\right|$, where n_i is the network delay of the i^{th} packet. The four algorithms thus differ only in their computation of $\hat{d}_i$:

Exp-Avg: This algorithm estimates the mean delay through an exponentially weighted average, much like the estimation of the variance above. In particular, $\hat{d}_i = \alpha\hat{d}_{i-1} + (1-\alpha)n_i$, with $\alpha = 0.998002$.

Fast Exp-Avg: This algorithm is similar to the first, except it adapts more quickly to increases in delays by using a smaller weighting factor as delays increase:

$$\hat{d}_i = \begin{cases} \beta\hat{d}_{i-1} + (1-\beta)n_i & : \quad n_i > \hat{d}_{i-1} \\ \alpha\hat{d}_{i-1} + (1-\alpha)n_i & : \quad n_i \leq \hat{d}_{i-1} \end{cases}$$

For the algorithm to respond quickly to increasing delays, $\beta < \alpha$. [18] used $\beta = 0.75$.

Min-Delays: This algorithm attempts to be more aggressive in minimizing delays. It uses the minimum delay of all packets received in the current talkspurt (let $\mathcal{S}_\rangle$ be this set of delays) as the average delay, i.e. $\hat{d}_i = \min_{j\in\mathcal{S}_\rangle}(n_j)$

Spk-Delay: It has been observed [1] that network delays often exhibit *spikes*, which are sharp increases in delay followed by nearly simultaneous reception of a large number of packets. *Spk-Delay* contains a spike detection algorithm which finds these spikes. During a spike, the delay estimate tracks the delays closely, and after a spike, an exponential weighted average is used. We avoid a detailed description here, referring the reader to [18] for details.

In [19], Moon et. al. propose an algorithm similar to *Spk-Delay*, which we call *Window*. This algorithm also looks for spikes. During spike mode, the delay of the first packet in a talkspurt is used as the playout delay. In normal (non-spike) mode, the playout delay is chosen by finding the delay which represents the q^{th} quantile among the last w packets received by the receiver. This determination is easily made by incrementally updating a histogram of the delays among the last w packets. In their simulations, a value of $10,000$ is used for w.

IV. New Playout Buffer Algorithms

In this section, we present our new algorithms which are capable of coupling and meeting non-zero loss targets.

A. Virtual Delay Algorithms

Our first contribution are a class of algorithms we call *virtual delay* playout adaptation algorithms. These algorithms are all modifications of existing algorithms to allow them to compensate for FEC. They are based on the following simple observation. Without FEC, the probability of playing out a particular packet is given in Equation 1. With FEC, we can generalize this simple formulation. A packet is played out if it either arrives before its playout time, or is recovered before its playout time. If p_N is the probability that a packet neither arrives nor is recovered, and d_V is the difference between the time a packet either arrives or is recovered (given it arrives or is recovered) and the time it was sent, the playout probability is $p_R = (1-p_N)P[d_V < D]$. This formulation is identical to the one in Equation 1, but with the simple random variables (p and d) replaced by the more complex ones (p_N and d_V). We can therefore use any existing playout adaptation algorithms which compute the playout delay D as some function of the packet delays by substituting d_V for d in the computation. Formally:

Definition 1: The virtual delay d_V of a packet is the difference in time between the earlier of the arrival and recovery times, and the generation time. If a packet neither arrives nor can be recovered, the virtual delay is undefined.

Any existing playout buffer adaptation algorithm is then *virtualized* by using the virtual delays to drive it instead of network delays. In most cases, it is not necessary to derive an expression for d_V to do this. A receiver just implements the FEC algorithm as it normally would. The instant it recovers a packet, the virtual delay of the packet is computed as the difference in time between transmission of the packet and the current time. If a packet is correctly received, the virtual delay is the network delay of the packet, unless it is recovered before it is received. The virtual delays are then used in the adaptation algorithm instead of the network delays.

It is important to note that implementing the virtualized version of the algorithms is a natural consequence of a layered implementation of FEC and playout buffer adaptation, where the recovery using FEC is done before the playout buffer adaptation. The playout buffer adaptation component cannot differentiate recovered packets from received packets. In this case, it will compute the arrival time of a packet as either its recovery time or arrival time, whichever is lower. The result is that the playout buffer adaptation algorithm will unknowingly compute virtual delays instead of real delays, and thus become virtual.

A.1 Supporting Target Loss Probabilities

It is the job of the playout adaptation algorithm to trade loss for delay. For the algorithms in [18], this tradeoff is controlled by the variation multiplier μ, set to a large static value (here, 4). The result are algorithms which target near zero loss. To be able to target non-zero losses, we must make this parameter adaptive.

Our algorithm assumes a target value for p_R. Unfortunately, you can't always get what you want, and this target may be unachievable given the current network loss rate. Therefore, we compute a current target loss rate p_C:

$$p_C = \max(p_T, f(p)) \tag{3}$$

where $f(p)$ is the achievable minimum application loss rate given a network loss rate of p. For Reed Solomon FEC,

$$f(p) = p\left(1 - \sum_{i=k}^{n-1}\binom{n-1}{i}(1-p)^i p^{n-1-i}\right) \tag{4}$$

and for the redundant codecs, $f(p) = p^K$.

We maintain a running estimate of the network loss rate $\hat{p}$. At the end of each talkspurt, $\hat{p}$ is used to compute p_C via Eq. 3. Then, the actual application loss rate p_L is measured. The variation multiplier μ from [18] is computed for the next talkspurt based on the following algorithm:

if $(p_C < p_L - \theta) \wedge (\mu \le \mu_{\text{max}} - \delta_{inc})$
  $\mu \leftarrow \mu + \delta_{inc}$;
else if $(p_C > p_L + \theta) \wedge (\mu \ge \mu_{\text{min}} + \delta_{dec})$
  $\mu \leftarrow \mu - \delta_{dec}$;
else
  $\mu \leftarrow \mu$

θ is a threshold which introduces hysterisis into the algorithm. μ_{min} and μ_{max} are the minimum and maxmimum allowed values for μ. δ_{inc} and δ_{dec} represent the step size for the coefficient. In our simulations, we used $\mu_{min} = 0$, $\mu_{max} = 8$, $\theta = .05$, $\delta_{inc} = 0.4$ and $\delta_{dec} = 0.2$. We found performance was best when the rate of increase in μ (δ_{inc}) was larger than the rate of decrease (δ_{dec}). We believe this is because it allows the algorithm to act more conservatively - rapidly increasing delays (and thus reducing losses) but gradually reducing them. As such, short lived decreases in jitter or loss won't cause the algorithm to undershoot the needed delay.

This small enhancement can be applied to all of the algorithms in [18]. When coupled with virtualization, we call the resulting algorithms *adaptively virtual*.

B. *"Previous Optimal" Algorithm*

Ideally, an optimal playout delay algorithm would work as follows. It would work offline, and know the packet delays and losses ahead of time. It would choose a playout delay which would meet an arbitrarily chosen criteria. Unfortunately, this algorithm is non-causal. As with many non-causal filters, this algorithm can be made causal by delaying the output of the algorithm for a talkspurt. In other words, the optimal playout delay for the previous talkspurt can be used as a playout delay for the next talkspurt. More generally, we can use any function of the optimal playout delays from previous talkspurts as the playout delay of the current talkspurt. For reasons of simplicity, we considered exponentially weighted averages of previous talkspurts.

At the end of each talkspurt, we compute the optimal playout delay D_{opt} for that talkspurt. We then choose the playout delay D_w for the next talkspurt as $D_w = \rho D_{w-1} + (1-\rho)D_{opt}$. We chose $\rho = .25$, which allowed the algorithm to adapt quickly. We also observed that in cases where the target loss is quite low (less than 2%), we needed to add a variation multiplier, so that the actual playout delay D_{act} is $D_{act} = D_w + \mu\hat{v}_w$ where $\hat{v}_w$ is computed in the same fashion as [18], except that it is driven by D_w. We used the adaptive algorithm described in Section IV-A.1 to compute μ, but with $\mu_{MAX} = 6$, which we found to give slightly better results.

We define the optimal playout delay for a talkspurt as the minimal playout delay that achieves a specified application loss rate. For a particular choice of playout delay D, the fraction of packets played out are all of those that arrive with a virtual delay less than the playout delay ($V_D^i < D$), divided by the total number of data packets sent, N. Interestingly, the optimal playout delay can only be equal to one of the values of V_D^i. For any delays between two adjacent V_d^i, the number of packets for which $V_D^i > D$ remains the same, but the delay increases. This means D_w equals the minimum V_D^j over j which satisfies:

$$\frac{1}{N}\sum_{i=1}^{N} I(V_D^i < V_D^j)) < (1-p_R)$$

$I()$ is the indicator function, equal to 1 when its argument is true, 0 when false.

Computing D_w is O(N). To reduce the complexity, we quantize the virtual delays using a linear quantizer with L steps. The quantizer uses a step size of 5 ms, and sets the $\frac{L}{2}^{th}$ quantized value to the virtual delay of the first packet received for the talkspurt. Any packet with a delay more than 400 ms larger or smaller than the first packet is quantized to $L-1$ and 0 respectively.We maintain a probability mass function of virtual delays in an array. At the end of the talkspurt, the probability distribution function is computed by summing this array. We then perform a search over the L entries in the distribution array to compute D_w. This search is O(1) (with a constant factor L).

C. *Model-Based "Analytical" Playout Adaptation Algorithm*

The analytical algorithm works by attempting to model the impact of network loss and delay on the application playout probability and the end to end delay. It then uses this model to choose a playout delay D.

Our model attempts to compute p_R, the probability of playing out a packet to the application, whether it arrived in time or was recovered in time, as a function of the network loss probability p, the network delay distribution and the playout delay D. We assume packet losses are independent. The model assumes Reed Solomon codes, where $n-k$ FEC packets protect k data packets. Any k of the n packets in a block must be received to recover the remaining $n-k$. We also assume the n packets in the block are received in order. There is not sufficient space in the paper to derive this expression; details can be found in [20]. The final expression turns out to be:

$$\begin{aligned} p_R \;=\;& (1-p)*P[n_i \le D] + \\ & (1-(1-p)P[n_i \le D])\frac{1}{k} \\ & \left(\sum_{g=k+1}^{n}(S(g-k)-S(g))P[X_2(g)]\right. \\ & \left.+\sum_{j=1}^{k}S(n-j+1)P[X_2(n)]\right) \end{aligned} \tag{5}$$

where:

$$\begin{aligned} P[X_2(g)] &= \sum_{r=k}^{g-1}\binom{g-1}{r}(1-p)^r p^{g-1-r} \\ S(j) &= P[n_i \le D - j\Delta] \end{aligned} \tag{6}$$

What we desire, in fact, is the inverse of this function: to compute D as a function of p_R and the delay and loss. Since the function cannot be inverted in closed form, we obtain D by trying various values, computing p_R based on those values using Eq. 5, and comparing p_R against the desired value. This is effectively a search. In fact, since the function is monotonically increasing with D, we perform a binary search. At the end of each talkspurt, the current target loss rate p_C is computed, using equation 3. We start with an arbitrary value for D, and compute p_R from above. If the result is below p_C by more than some threshold (we used 5%), a higher value of D is tried, and if the result is above p_C, a lower value is tried.

The computation of p_R from Eq. 5 requires an expression for the network delay distribution $P(n_i < X)$ for a range of X, and an estimate of p. We measure the loss probability p by computing the percentage of lost packets in each talkspurt ($\tilde{p}$), and applying an exponential filter to average this value, yielding $\hat{p} = 0.25\tilde{p} + 0.75\hat{p}$.

To compute the network delay distribution, we maintain the delays of the last 1000 packets in a queue. Each delay is first quantized, using a linear quantizer with a step size of 5 ms and upper limit of 5 s. The frequency of each delay is maintained in a histogram. When a new packet arrives, the delay of the oldest packet is removed from the histogram, and the delay of the newest is added. The delay distribution is computed using a cumulative sum of the frequencies, and is done only at the end of each talkspurt.

V. Simulations

The objective of our simulations is two-fold: first, to demonstrate that our new algorithms outperform the decoupled ones; second, to determine the performance of the new algorithms compared to each other.

A. Simulation Model

In our simulation model, the sender generates speech packets every 20 ms. Each packet consists of an IP/UDP/RTP [21] header, totaling 40 bytes, in addition to 24 bytes of speech. This is equivalent to a 9.6 kb/s speech codec. The speech is protected with a (5,3) Reed-Solomon code. This means that every three packets, two additional FEC packets are generated. As long as the receiver gets any three of the five packets, the three media packets are recovered. These two FEC packets are piggybacked on the first two data packets in the next block, so that the FEC is sent spaced apart. Recent results [22] have shown that spacing the FEC in this manner yields better performance.

To model packet loss, we used both Bernoulli and Gilbert processes; studies have shown these to be reasonable models [22]. However, delay models, particularly those that capture correlation, are not easy to find. As a result, we based our simulations on a network model which combines real measured traces on the Internet with simulated losses. Our real traces are based on measurements taken in the September of 1997. We used three traces. The first is between a host at GMD Fokus in Germany, and a host at Columbia U. in New York. The second is between a host at USC Santa Cruz and the host at Columbia, and the third is between the host in Santa Cruz and a host at U.Mass Amherst in Massacusetts. To generate each trace, packets were sent between the endpoints for 14 minutes, and the delay and loss of each packet was recorded. The trace gives us a real sample path of an end-to-end delay process. We used the algorithm described in [23] to remove clock skew from these traces. The average loss probabilities in the three traces were .001, .057, and .026 respectively. The average one way delays were 2 ms, 6.4 ms and 13.3 ms, respectively. The jitter was 2.64 ms, 3.46 ms, and 8.18 ms, respectively, computed as the standard deviation in the delay.

To support a wider range of simulations, we salted these traces with additional losses. Of the packets which did arrive in the actual trace, each was subject to a simulated loss with probability p drawn from a Bernoulli process. By varying p, we were able to adjust the network loss rates from the actual value in the trace ($p = 0$) to 1 ($p = 1$). We also ran simulations with Gilbert loss, and observed similar results.

At the receiver, the playout buffer algorithms were implemented, and the FEC was used to recover lost packets when possible.

B. Coupled vs. Uncoupled

In this section, we compare uncoupled versions of the Exp-Avg, Spk-Det and Window algorithms with our virtually adaptive versions of these algorithms. For brevity, we omit the simulation results of the fast exponential average and min-delay algorithms, since according to our simulation results, these two algorithms usually do not outperform the others. For the window algorithm, we use a 1000 packet window instead of the 10,000 packet widow used in [19], which we found to be too large to adapt adequately.

We use two uncoupled versions of these algorithms. The first version is the original algorithm - no extra delay is added to compensate for FEC. In the second version, we added 80 ms of delay to the original output of each algorithm. This corresponds to $N - 1$ packet intervals, enough to receive all the FEC for any packet in the absence of jitter.

The adaptively virtual versions of the algorithms operate with a loss target of zero.

Figure 1 contains six graphs. The two at the top compare the performance of the uncoupled *Exp-avg* algorithm with its adaptively virtual extension; the two in the middle compare the uncoupled *Spk-det* algorithm with its extension; the two at the bottom compare uncoupled *window* algorithm with its extension. Each graph in the figure has three curves, corresponding to the two uncoupled versions of the algorithm and the adaptively virtual extension. The left column shows the average application loss probability after FEC and playout buffer adaptation $(1-p_R)$ vs. the network loss probability p, varied by salting the traces. The right column shows the average D across all talkspurts vs. the network loss probability p. All plots use trace two.

In all three pairs of graphs, the results are the same. The original version of the algorithm shows increasing application loss probabilities with increasing network loss rate. This is because the playout delay computed by the algorithm is not large enough to make use of the FEC packets. The plots in the right column show this as well. The original version of each algorithm has a constant playout delay as the network loss varies. For the original versions of Exp-Avg and Spk-Det, the increase in application loss probability is linear with the network loss probability. This is because the playout delay is not sufficiently large to allow the FEC to be consistently used (80 ms is needed).

When we add 80 ms of delay to the output of the original algorithms, the loss rates drop substantially. Now, there is sufficient delay to make full use of FEC. In fact, the loss rates track those of our adaptively virtual extensions. However, the decoupled algorithm now tends to have consistently large playout delays, even when network loss rates are small and the delay is not needed.

Our adaptively virtual extensions perform much better. The right graphs show that in all cases, the network delays start low, and gradually increase as network loss rates increase. The result is that the end to end delay of our extension is generally lower than the uncoupled version which adds 80ms, but with the same low application loss probabilities. This is exactly the desired behavior.

C. Comparisons of New Algorithms

In this section, we compare the performance of our new algorithms (adaptively virtual extensions, previous optimal, and analytical) against each other. The simulation environment is identical to that described in Section V-A. Our simulations cover two metrics: the ability to make good use of FEC with small delays, and the ability to achieve a target application loss rate with the minimal delay.

C.1 Using FEC with Minimal Delays

Figure 2 depicts the application loss rate (left column) and average playout delay (right column) vs. network loss rate. Each graph shows the performance of the adaptively virtual Exp-Avg algorithm *Exp-Avg Ext*, the adaptively virtual Spk-Det algorithm *Spk-Det Ext*, the adaptively virtual window algorithm *Window Ext*, the previous optimal *Prev-opt (Bin)* algorithm and the analytical algorithm, all targeting zero loss. We also include a plot of an unrealizable non-causal "optimal" algorithm. This algorithm computes the playout delay for all talkspurts at the beginning of the trace (assuming knowledge of the packet delays and losses in the entire trace), based on minimizing the average delay for the entire trace for a given application loss probability. This optimal algorithm is actually the lower bound described in [19], but with the virtual delays replacing the network delays in the computation.

The figure has three pairs of graphs; each pair is obtained from a different trace. The results indicate fairly consistent behaviors across traces. The adaptively virtual window algorithm tends to have the highest delays, and low loss rates, but not the lowest. The analytical and previous optimal algorithms generally have lower end-to-end delays than the adaptively virtual window algorithm (with the exception of the previous optimal algorithm in trace 1), with application loss probabilities that are generally equal to or less than the adaptively virtual window algorithm. This indicates that the analytical and previous-optimal algorithms are generally preferable to the adaptively virtual window algorithm.

The virtually adaptive Exp-Avg algorithm appears to perform quite well. Its delays are consistently closest to the optimal (only Spk-Det does noticeably better in trace 3), and its loss probabilities are only slightly higher than the previous optimal and analytical algorithms. The virtually adaptive Spk-Det algorithm also maintains low delays, but its application loss probabilities are generally the highest. This would indicate that the adaptively virtual Exp-Avg is generally preferable to the adaptively virtual Spk-Det. This result contradicts [18], where the more sophisticated Spk-Det outperforms Exp-Avg. We believe this is because Spk-Det attempts to track the network delays too closely and loses packets whenever its delay estimate is small. The results by Moon et. al. in [19] agree with ours.

C.2 Achieving a Loss Target

All of our algorithms are capable of achieving a specified loss target, input as a parameter to the algorithm. In Fig. 3, we consider the ability of the algorithms to meet a wide range of targets under a fixed network condition. This fixed network condition corresponds to the unsalted traces 1, 2 and 3. The target loss rate is varied from 0 to 15%. As with the other plots, the application loss rate and playout delay are shown.

Ideally, the application loss rate should equal the target, corresponding to a straight line with slope one on the left graphs. The non-causal optimal algorithm achieves this goal, of course. The plots on the right show that the optimal algorithm decreases the end to end delay as the target increases, as expected.

The results here are consistent with those in the previous section. The adaptively virtual window algorithm is the worst performer, consistently overshooting the required delay and, as a result, undershooting the target. The previous optimal algorithm consistently undershoots the target loss, although it comes close in all traces but the first (it's worthwhile to note that trace 1 has little jitter or loss, making it difficult to achieve a large target loss rate). Not surprisingly, it also tends to overshoot the optimal delay. The analytical algorithm tends to have very good delay properties, coming close to the optimal delay. Its ability to meet the loss target varies, though. It consistently undershot it in trace 1 (although, in all fairness, all other algorithms undershot it in this trace as well), but overshot it in all the others, although not by too much. Performance is especially good in trace 2, where the analytical algorithm is the best performer. The adaptively virtual Exp-Avg and Spk-Det have similar performance. They both came consistently close to the target with reasonably good delays.

Overall, the simulations demonstrate that our algorithms were generally able to meet the goal of achieving a desired target loss rate.

VI. Conclusion and Future Work

In conclusion, we have demonstrated that there is a need to coupled both loss and delay into adaptive playout buffer algorithms when FEC is used. We have presented a number of novel algorithms to perform this coupling. One such algorithm is in fact a class of algorithms called *adaptively virtual* algorithms that extend existing algorithms. Our algorithms also allow us to control the target application loss probabilities.

Our future efforts are focused on alternate definitions of optimality (instead of minimal D for achieving a target p_R), which we have found to be too sensitive to the tail of the network delay distributions. We are also investigating other mechanisms for adapting μ in the adaptively virtual algorithms, such as multiplicative increase and linear decrease. Finally, we are exploring the sensitivity of our algorithms to the various tunable parameters they make use of.

VII. Acknowledgments

The authors would like to thank Dina Katabi, S. Keshav, Sue Moon, Ramachandran Ramjee and Yin Zhang for their help and comments.

References

[1] Jean-Chrysostome Bolot, "End-to-end packet delay and loss behavior in the Internet," in *SIGCOMM Symposium on Communications Architectures and Protocols*, Deepinder P. Sidhu, Ed., San Francisco, California, Sept. 1993, ACM, pp. 289–298, also in *Computer Communication Review* 23 (4), Oct. 1992.

[2] Vern Paxson, "End-to-end internet packet dynamics," in *SIGCOMM Symposium on Communications Architectures and Protocols*, Cannes, France, Sept. 1997.

[3] Amarnath Mukherjee, "On the dynamics and significance of low frequency components of internet load," *Internetworking: Research and Experience*, vol. 5, no. 4, pp. 163–205, Dec. 1994.

[4] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated service," Request for Comments (Informational) 2475, Internet Engineering Task Force, Dec. 1998.

[5] J. Wroclawski, "Specification of the controlled-load network element service," Request for Comments (Proposed Standard) 2211, Internet Engineering Task Force, Sept. 1997.

[6] S. Shenker, C. Partridge, and R. Guerin, "Specification of guaranteed quality of service," Request for Comments (Proposed Standard) 2212, Internet Engineering Task Force, Sept. 1997.

[7] C. Perkins and O. Hodson, "Options for repair of streaming media," Request for Comments (Informational) 2354, Internet Engineering Task Force, June 1998.

[8] Colin Perkins, Orion Hodson, and Vicky Hardman, "A survey of packet loss recovery techniques for streaming audio," *IEEE Network*, vol. 12, no. 5, pp. 40–48, Sept. 1998.

[9] Nachum Shacham and Paul McKenney, "Packet recovery in high-speed networks using coding and buffer management," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, San Francisco, California, June 1990, IEEE, pp. 124–131.

[10] J. Rosenberg and H. Schulzrinne, "An RTP payload format for generic forward error correction," Internet Draft, Internet Engineering Task Force, June 1999, Work in progress.

[11] L. Rizzo, "Effective erasure codes for reliable computer communication protocols," *ACM Computer Communication Review*, vol. 27, no. 2, pp. 24–36, Apr. 1997.

[12] C. Perkins, I. Kouvelas, O. Hodson, V. Hardman, M. Handley, J. C. Bolot, A. Vega-Garcia, and S. Fosse-Parisis, "RTP payload for redundant audio data," Request for Comments (Proposed Standard) 2198, Internet Engineering Task Force, Sept. 1997.

[13] Jean-C. Bolot and Andres Vega-Garcia, "The case for fec-based error control for packet audio in the internet," *ACM Multimedia Systems*, pp. –, 1997.

[14] Jean-Chrysostome Bolot and Andres Vega Garcia, "Control mechanisms for packet audio in the internet," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, San Fransisco, California, Mar. 1996.

[15] Vicky Hardman, Angela Sasse, Mark Handley, and Anna Watson, "Reliable audio for use over the internet," in *Proc. of INET'95*, Honolulu, Hawaii, June 1995.

[16] Danny Cohen, "Issues in transnet packetized voice communications," in *Proceedings of the Fifth Data Communications Symposium*, Snowbird, Utah, Sept. 1977, ACM, IEEE, pp. 6–10 – 6–13.

[17] Warren A. Montgomery, "Techniques for packet voice synchronization," *IEEE Journal on Selected Areas in Communications*, vol. SAC-1, no. 6, pp. 1022–1028, Dec. 1983.

[18] Ramachandran Ramjee, Jim Kurose, Don Towsley, and Henning Schulzrinne, "Adaptive playout mechanisms for packetized audio applications in wide-area networks," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, Toronto, Canada, June 1994, pp. 680–688, IEEE Computer Society Press, Los Alamitos, California.

[19] Sue B. Moon, Jim Kurose, and Don Towsley, "Packet audio playout delay adjustment: performance bounds and algorithms," *ACM/Springer Multimedia Systems*, vol. 5, no. 1, pp. 17–28, Jan. 1998.

[20] Jonathan Rosenberg, Lili Qiu, and Henning Schulzrinne, "Integrating FEC into adaptive voice playout buffer algorithms on the internet," Technical Report CUCS-00X-99, Columbia University, New York, New York, Aug. 1999.

[21] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications," Request for Comments (Proposed Standard) 1889, Internet Engineering Task Force, Jan. 1996.

[22] Jean-Chrysostome Bolot, Sacha Fosse-Parisis, and Don Towsley, "Adaptive FEC-Based error control for interactive audio in the internet," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, New York, Mar. 1999.

[23] Sue Moon, Paul Skelly, and Don Towsley, "Estimation and removal of clock skew from network delay measurements," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, New York, Mar. 1999.

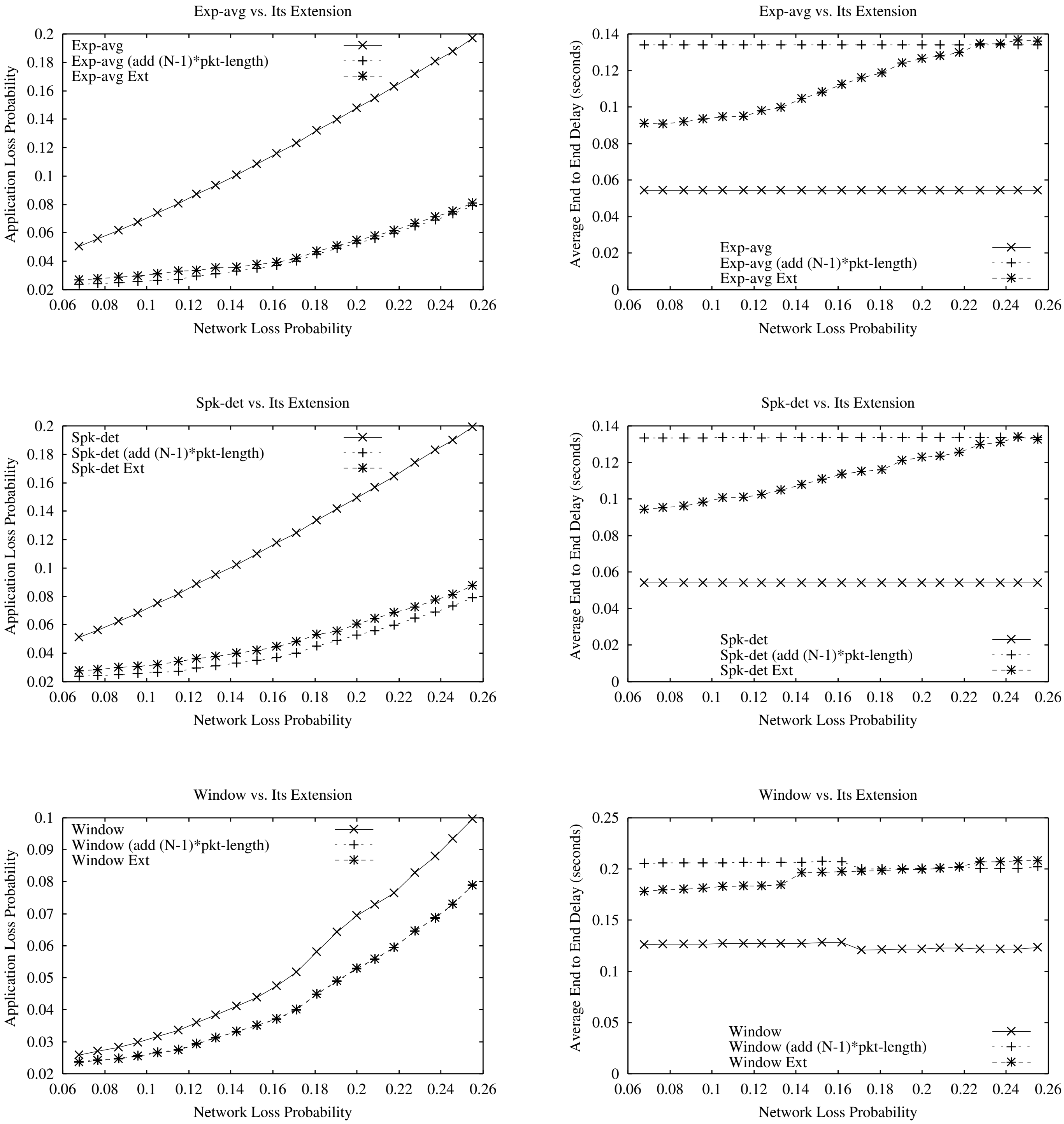

Fig. 1. Performance of Adaptively Virtual Algorithms on Trace 2

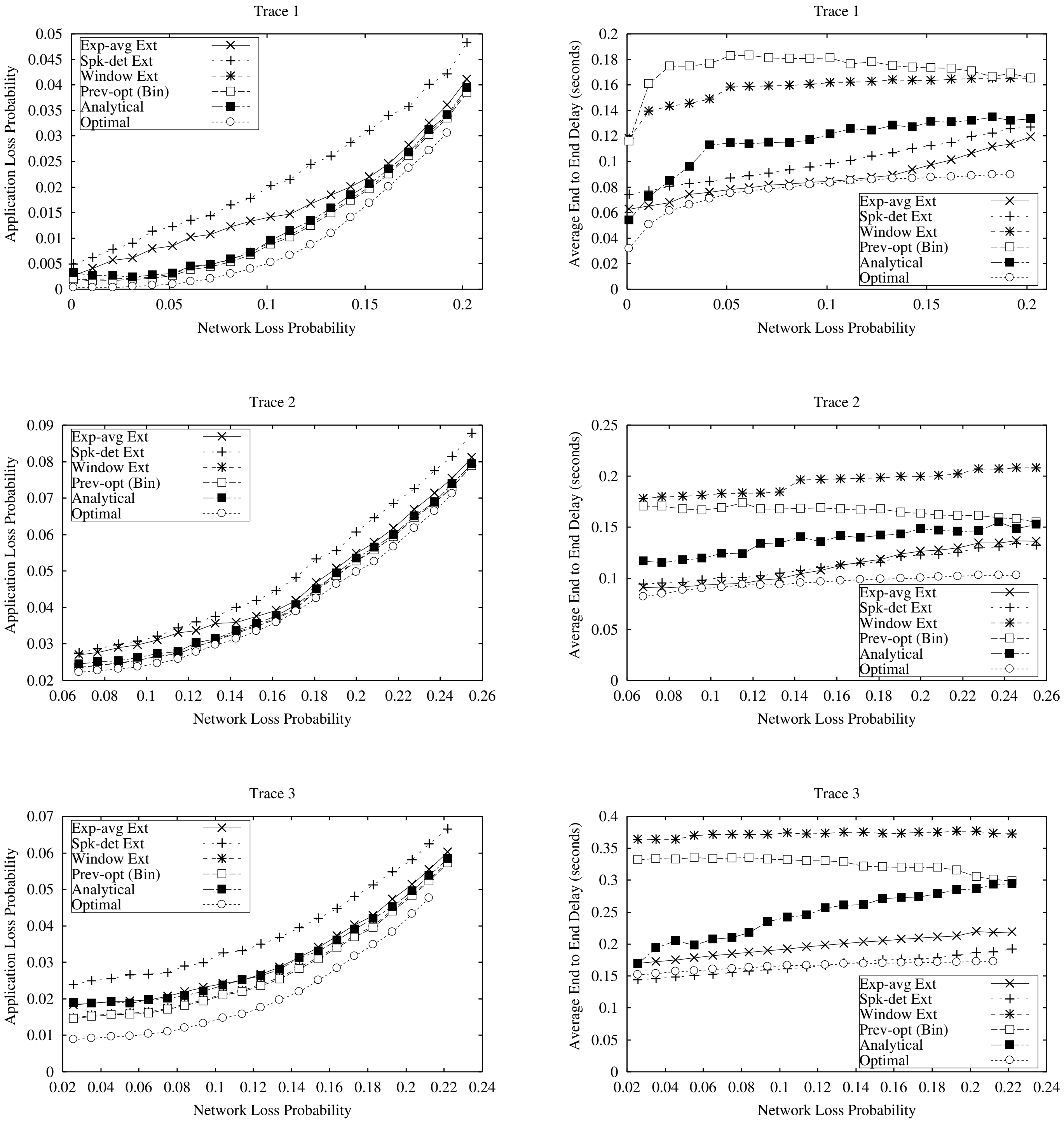

Fig. 2. Comparison of Loss and Delay Performance across All Algorithms

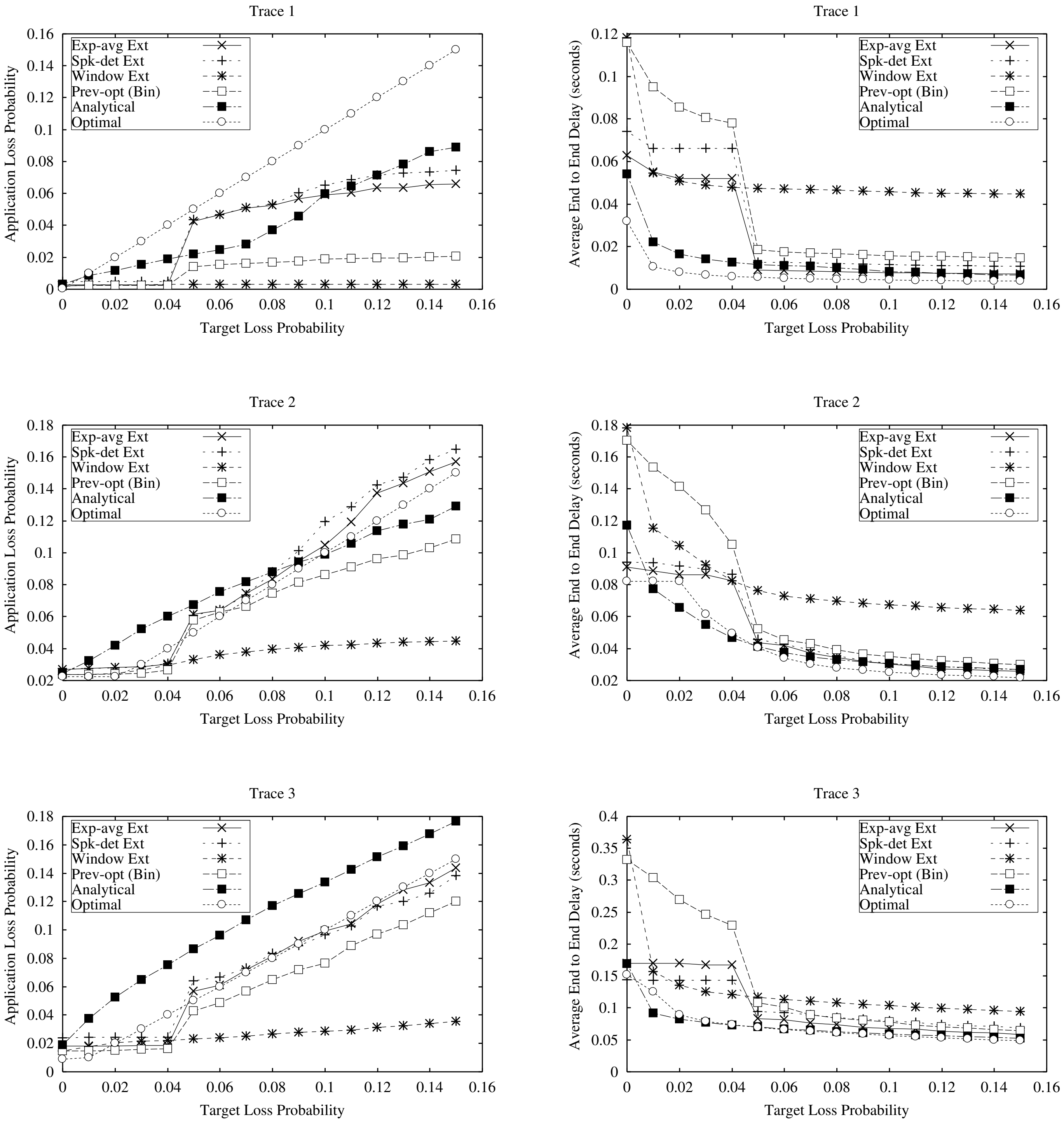

Fig. 3. Performance of Algorithms in Achieving Varying Target Loss Probability