Multi-layer Active Queue Management and Congestion Control for Scalable Video Streaming

Seong-ryong Kang, Yueping Zhang, Min Dai, and Dmitri Loguinov

Abstract—Video streaming is becoming an increasingly important part of the current Internet; however, before high-quality streaming becomes a reality, the best-effort model of the Internet may need to be adapted to provide certain scalable services specifically targeted at video flows. In this paper, we study one such mechanism and propose a new video streaming framework, which allows applications to mark their own packets with different priority and use multi-queue congestion control inside routers to automatically drop the less-important packets during buffer overflows. We describe priority AQM algorithms that provide optimal performance for video applications and study a variation of Kelly’s congestion controls as part of this framework. Through simulation and analytical investigation, we find that our AQM based solution allows the application to maintain a much higher quality of video for the end-user compared to similar scenarios in a best-effort network. We call the combined framework PELS — Partitioned Enhancement Layer Streaming.

I. INTRODUCTION

Video applications usually transport multimedia data that are highly sensitive to quality-of-service (QoS) characteristics (e.g., delay or packet loss) of their end-to-end path. It is thus often believed that these applications will require more sophisticated services than the current model of the best-effort Internet before they can offer a high-quality streaming environment to end-users. In response to this demand, significant research effort is currently under way to improve the QoS model (or lack thereof) in the current Internet [3], [6], [8], [13], [16], [40]. One dimension of this related work argues in favor of network QoS that guarantees a certain “better than best-effort” performance to end-flows (these methods loosely favor of network QoS that guarantees a certain “better than best-effort” performance to end-flows (these methods loosely)

Furthermore, during heavy congestion, even the lines due to large delays involved in the retransmission loop. However, none of the existing QoS methods provide a scalable, low-overhead, low-delay, and retransmission-free platform required by many current real-time streaming applications. To fill this void, this paper investigates novel AQM algorithms that not only can provide a provably “optimal” performance under random loss, but also possess very low implementation complexity.

One of the characteristics of video packets that does not match the best-effort service is that they often carry information of different importance. Thus, video applications can clearly differentiate between the more-important and the less-important packets. In all layered video coding schemes, the base layer is more important than the enhancement layer. Furthermore, the lower sections of the enhancement layer are more important than the higher sections because their loss renders all dependent data virtually useless. Thus, treating all video packets equally (as in the current Internet) usually leads to significant quality degradation during packet loss and low useful throughput during congestion, both of which cause video streaming to become less appealing in practical settings.

With the presence of unequal importance among video packets, the first goal of this work is to achieve “high end-user utility,” which means that the majority of packets that are transmitted across the bottleneck link must carry useful information that can be decoded by the receiver. In video applications, which use motion compensation and variable-length coding (VLC), a single lost packet in the base layer may affect several other frames and render them all useless, even though some of them arrive to the receiver without any loss. Furthermore, the enhancement layer is not immune to packet loss either since strong dependence between the coded data allows packet loss to affect consecutive chunks of data that are significantly larger than those actually lost in the network. Hence, even under moderate packet loss, the bottleneck link may be used to transmit a large number of packets that eventually get dropped by the decoder.

In addition to high utility, many applications (such as video telephony and other interactive services) require low end-to-end delays between the server and the client. Additional problems with delays arise during retransmission of lost packets, which have a higher tendency of violating their decoding deadlines due to large delays involved in the retransmission loop. Furthermore, during heavy congestion, even the retransmitted packets may be dropped in the same congested queues [27], which means that in such cases, retransmissions only clutter the network and increase congestion for the regular packets. Thus, our second goal is to provide a retransmission-free network service to video flows. This direction generally aligns well with FEC-based approaches, except that our goal is to avoid all bandwidth overhead associated with error-correcting codes and occupy network channels only with the actual video data.

To improve the quality of video delivered over the Internet, we investigate a new streaming framework in which each application marks its own packets with different priorities and uses AQM inside the routers to drop the less-important packets during congestion. Such preferential (instead of random) dropping of packets allows the application to maintain a much...
higher quality of video for the end user compared to similar scenarios in a best-effort network. We also find that the use of multi-queue AQM allows scalable video applications to maintain high useful link utilization without retransmitting any of the lost packets or sending any error-correcting codes. Thus, we achieve both goals of high utility and low end-to-end delay.

While our implementation relies on Kelly’s utility-based controllers [23], it is important to realize that the proposed framework can be used with any congestion control (including end-to-end methods such as AIMD, TFRC, or even TCP) and can be deployed in the current Internet with minimum modifications to the existing infrastructure.

The rest of the paper is organized as follows. Section II discusses the background and related work. Section III presents a stochastic loss model and its application to video traffic. Section IV describes the proposed video streaming framework including packet-marking techniques and optimal router queue management mechanisms. Section V studies congestion control applied to the proposed framework. Section VI discusses simulation results and Section VII concludes the paper.

II. BACKGROUND AND RELATED WORK

Improving the best-effort model of the current Internet has become a widely studied research area. Many of these studies focus on Active Queue Management (AQM) [8], [13], [16], [40] that provides unequal treatment to the flows, while others range from offering hard guarantees in the form of IntServ [5] to more scalable models such as DiffServ [3], [6]. We briefly overview some of the more recent and promising approaches.

A. Priority QoS Methods

Several studies investigate the performance of video streaming over the DiffServ architecture. Gurses et al. [18] use a temporally-scalable H.263+ video scheme and three-color markers (TCM) that allow ingress routers to promote packets (i.e., increase their priority). However, this work does not employ congestion control or allow the end-flows to benefit from unequal priority of the packets since DiffServ can arbitrarily remark them according to ingress/egress policies of peering ISPs. Shin et al. [38], [39] study the problem of optimally assigning relative priority indexes to video packets depending on their impact on the quality of received video. Besides using a fairly complex packet prioritization scheme, the work does not use congestion control or discuss how the network should treat marked packets. Zhao et al. [45] employ MPEG-4 FGS for video streaming and use several computationally intensive packet prioritization schemes, but also without studying network support of the proposed architecture.

Among non-DiffServ methods, Kuzmanovic et al. [25] propose TCP-LP, which provides a TCP-like low-priority service that seeks out bandwidth left-over from the high-priority streams. Tang et al. [41] present a video streaming protocol that uses low-priority dummy packets to probe for new bandwidth. The dummy packets are sent only upon packet loss and for the duration of one round-trip delay. Hurley et al. [20] propose ABE (Alternative Best Effort) that requires applications to choose between two conflicting types of service (i.e., low delay or low packet loss). A similar approach is used in BEDS (Best Effort Differentiated Service) [14]. Among related studies, Internet-2’s QBSS (QBone Scavenger Service) [35] is most similar to our approach as it provides service differentiation by allowing end-flows to mark their own packets with the low-priority bit. However, the current QBSS does not support more than two priorities or directly benefit video traffic.

Additional work compares the effect of uniform and priority dropping on video quality. Among these studies, Bajaj et al. [1] propose a model in which the utility of received video depends only on the packet loss rate within each layer; however, the model does not consider where within the layer each packet loss occurs. This model is clearly unrealistic (since there is dependency between data within each layer, as well as between the layers) and leads the authors to conclude that priority dropping does not necessarily offer improvement over best-effort (uniform) loss. Fidler [12] studies the effect of priority dropping in DiffServ networks and shows in simulation that it performs better than uniform dropping.

Finally, note that many studies in DiffServ provide service differentiation through packet coloring/marking at ingress/egress routers (e.g., [6], [9], [10], [11], [19]). However, since different ISPs on a network path can arbitrarily remark incoming packets, application-specific priorities are changed to suit the ISP and no longer reflect the requirements of end-flows.

B. Active Queue Management

Active Queue Management (AQM) schemes perform special operations in the router to achieve better performance for end flows. These operations include dropping random packets (e.g., RED), re-arranging the order in which packets are served (e.g., WFQ), and randomly marking packets from more aggressive flows (e.g., ECN). While WFQ focuses on providing fairness to competing flows [8], [40], RED/ECN attempt to avoid congestion by randomly dropping or marking packets with a certain probability that increases with the level of congestion [13], [16], [17]. As such, these methods are not specifically tailored to multimedia applications and thus cannot directly improve video quality of Internet streaming.

Additional studies combine congestion control with AQM and provide much smoother sending rates to end-flows since routers can detect network conditions more accurately than end systems. Lapsley et al. [26] study optimization-based congestion control and propose router-based Random Early Marking (REM) that works with cooperating end-flows to maximize their individual utilities. Katabi et al. [22] present XCP (xPlicit Congestion notification Protocol) that conveys information about the degree of congestion in network paths to application sources using two separate AQM controllers (one for utilization and one for fairness). Several other studies include Kelly’s optimization methods [21], [23], [24], [32] and Low’s work [28], [29], [30], [31].

C. Structure of MPEG-4 FGS

Recall that FGS (Fine Granular Scalability) [36] is the streaming profile of the ISO/IEC MPEG-4 standard, which is
Fig. 1. Scaling of MPEG-4 FGS using fixed-size (left) and variable-size (right) frames.

III. STOCHASTIC MODEL OF PACKET LOSS

In the first part of this section, we investigate the performance of video streaming in the best-effort Internet assuming random packet loss. We consider two loss models and study the expected amount of recovered data in each video frame. Unlike [1], we use realistic assumptions about the dependency between data in each FGS frame and derive the expected percentage of useful data transmitted over the bottleneck link.

Recall that many studies of Internet QoS (e.g., RED/ECN) attempt to improve TCP performance by changing drop behavior of the network from bursty to uniformly random [16], [17]. Thus, it may be argued that future networks will deploy such packet drop mechanisms more often than the current Internet, which explains the reasoning behind our first model based on independent Bernoulli loss (see below for details). Even though the trend of QoS support in future networks aligns well with the Bernoulli model, the current Internet is still largely based on FIFO queues, in which buffers overflow and packets are lost in bursts. Hence, we extend our attention to a more general distribution of packet loss and model the network as an alternating ON/OFF process with heavy-tailed ON (loss) and model the network as an

A. Bernoulli Packet Loss

In this section, we investigate the effect of random packet drops on video quality using the example of MPEG-4 FGS (similar results apply to non-FGS layered coding)\(^1\). We start by building a simple packet-loss model for FGS sequences, derive the expected amount of useful data recovered from each frame, and define the effectiveness of FGS transmission over a lossy channel. Note that in our analysis, we only examine the enhancement layer and assume that the base layer is fully protected. Even under such conditions, best-effort networks deliver very low performance to scalable flows, which progressively degrades as the streaming rate becomes higher.

Assume that long-term network packet loss \(p\) can be modeled by a sequence of independent Bernoulli variables \(X_i\). Each \(X_i\) is an indicator function that determines whether packet \(i\) is lost or not: \(X_i = 1\) iff packet \(i\) is dropped in the network. Then, \(P(X_i = 1) = 1 - P(X_i = 0) = E[X_i] = p\) is the average packet loss. Even though this model is a great simplification of real networks and results in the probability of obtaining a loss-burst of length \(k\) proportional to \(e^{-k}\) (i.e., the tail of loss-burst lengths is exponential), it provides a good starting point. We analyze the case with Pareto burst lengths in the next subsection.

Assume that FGS frame sizes \(H_j\) are measured in packets and are given by i.i.d. random variables with a probability mass function (PMF) \(q_k = P(H_j = k), k = 1, 2, \ldots\) The exact distribution of \(H_j\) is not essential and typically depends on the coding scheme, frame rate, variation in scene complexity, and the bitrate of the sequence. The question we address next is what is the expected amount of useful packets that the receiver can decode from each frame under \(p\)-percent random loss? Thus, our goal is to determine the expectation of \(Z_j\), which is the number of consecutively received packets in a frame \(j\).

Theorem 1: Assuming independent Bernoulli packet loss with probability \(p\), the expected number of useful packets recovered per frame is:

\[
E[Z_j] = \frac{1-p}{p} \sum_{k=1}^{\infty} (1-(1-p)^k) q_k. \tag{1}
\]

\(^1\)Note that motion-compensated enhancement layers suffer even more degradations under best-effort loss and are not modeled in this work. However, the expected amount of improvement from AQM in such schemes is even higher than that in FGS.
TABLE 1

EXPECTED NUMBER OF USEFUL PACKETS (BERNOULLI MODEL)

<table>
<thead>
<tr>
<th>Packet loss</th>
<th>$H = 100$</th>
<th>$H = 1,000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>Simulations</td>
<td>Model (5)</td>
</tr>
<tr>
<td>0.0001</td>
<td>99.49</td>
<td>99.49</td>
</tr>
<tr>
<td>0.001</td>
<td>95.06</td>
<td>95.06</td>
</tr>
<tr>
<td>0.01</td>
<td>62.60</td>
<td>62.57</td>
</tr>
<tr>
<td>0.1</td>
<td>8.98</td>
<td>8.99</td>
</tr>
<tr>
<td>0.2</td>
<td>4.01</td>
<td>4.00</td>
</tr>
<tr>
<td>0.9</td>
<td>0.10</td>
<td>0.11</td>
</tr>
</tbody>
</table>

Proof: Assume that $G_j$ is the random distance from the beginning of frame $j$ to the next packet-loss event. Then, all $G_j$ are geometric random variables with respect to each frame $j$ and can assume any integer value in the range $[1, \infty)$. Note that when the first packet-loss location $G_j$ is no more than $H_j$, the decoder recovers exactly $G_j - 1$ packets from the frame. Otherwise, all $H_j$ packets are recovered.

Next, taking a conditional expectation of $Z_j$, we can write:

$$E[Z_j|H_j = k] = \sum_{i=1}^{k} (i-1) p_i + \sum_{i=k+1}^{\infty} k p_i,$$  \hfill (2)

where $p_i = P(G_j = i) = q^{i-1} p$ is the geometric PMF of $G_j$ and $q = 1 - p$. Then, (2) becomes:

$$E[Z_j|H_j = k] = p \sum_{i=1}^{k} (i-1) q^{i-1} + k \sum_{i=k+1}^{\infty} q^{i-1} p \quad \text{(3)}$$

$$= p \sum_{i=1}^{k} (i-1) q^{i-1} + k \left( 1 - p \sum_{i=1}^{k} q^{i-1} \right).$$

Substituting $l = i - 1$ into (3), we get:

$$E[Z_j|H_j = k] = p \sum_{l=0}^{k-1} l q^l + k \left( 1 - p \sum_{l=0}^{k-1} q^l \right)$$

$$= p \sum_{l=0}^{k-1} \frac{d}{dq} q^l + k \left( 1 - p \frac{1 - q^k}{1 - q} \right)$$

$$= \frac{1 - p}{p} (1 - (1 - p)^k).$$ \hfill (4)

Expanding the conditional expectation in (4) to arbitrary frame sizes $H_j$, we obtain (1).

To better understand the result of Theorem 1, we first examine one particular distribution of $\{H_j\}$, in which all FGS frames have the same fixed size $H$. Let $E[Z_j^H]$ be the expected number of useful packets if all frames are of size $H$. Under these conditions, (1) becomes:

$$E[Z_j^H] = \frac{1 - p}{p} (1 - (1 - p)^H).$$ \hfill (5)

This model is compared to actual simulation results in Table I for $H = 100$ and $H = 1,000$. As the table shows, even under a reasonably low packet loss of 1%, the expected number of useful packets in each frame is only 62 for $H = 100$; however, the decoder successfully receives (on average) 99 packets per frame. When we use a larger frame with $H = 1,000$, the decoder can use only 98 packets out of each 990 packets it receives over the network. Moreover, under $p = 0.01$, only 9 useful packets are recovered from each frame regardless of the actual size of the frame. This means that the bottleneck link transmits 10 ($H = 100$) to 100 ($H = 1,000$) times more packets than the receiver is able to utilize in decoding its video.

It is easy to notice in (5) that as streaming rates become higher (and $H$ becomes larger), $E[Z_j^H]$ tends to $(1 - p)/p$ and the recovered (useful) percentage of each frame tends to zero. This is shown in Fig. 2 (left) for $p = 0.1$, in which the number of useful packets in the best-effort case quickly saturates at $(1 - p)/p = 9$ as $H$ becomes large. The same side of the figure also plots the number of packets that could have been recovered in the “optimal” case, where all $H (1 - p)$ packets are useful in decoding (which is clearly the best possible scenario under $p$-percent packet loss).

To quantify the effect of FGS packet transmission on video quality, we define utility $U^H$ of received FGS video as the ratio of the average number of FGS packets used in decoding a video frame (i.e., $E[Z_j^H]$) to the total number of received FGS packets (i.e., $H - \mu H$):

$$U^H = \frac{E[Z_j^H]}{H (1 - p)} = \frac{1 - (1 - p)^H}{H p}.$$ \hfill (6)

For instance, we get $U^H = 0.1$ for $p = 0.1$ and $H = 100$, which means that only 10% of the received FGS packets are useful in enhancing the base layer. This result is further illustrated in Fig. 2 (right), which plots the utility of best-effort streaming and the “optimal” utility for different values of $H$ and $p = 0.1$. As the figure shows, (6) matches simulation results and drops to zero inverse proportionally to the value of $H$, the latter of which means that as $H \rightarrow \infty$ (i.e., sending rates become higher), the decoder receives “junk” data with probability 1.

Next, we briefly study the result of Theorem 1 for arbitrary frame-size distributions and show that (5) provides an upper bound on $E[Z_j]$ in any video sequence with the average frame size $H$.

Lemma 1: For a given average frame size $H$, the expected number of useful packets recovered from a frame is always upper-bounded by that in sequences with $H_j = H$:

$$E[Z_j] \leq E[Z_j^H].$$ \hfill (7)

Proof: Notice that we can re-write (1) as:

$$E[Z_j] = \frac{1 - p}{p} E[Z_j^H].$$ \hfill (8)
Let \( u(x) = 1 - (1 - p)^x \) and notice that \( u(x) \) is a strictly concave function of \( x \). Then, using Jensen’s inequality, \( E[u(H_j)] \) is less than or equal to \( u(E[H_j]) \). Applying this observation to (8), we have:

\[
E[Z_j] = \frac{1 - p}{p} E[u(H_j)] \leq \frac{1 - p}{p} u(E[H_j]) = E[Z^H_j]. \tag{9}
\]

We next verify the result of Lemma 1 assuming a lognormal frame-size distribution, which has the following PDF:

\[
f(x) = \frac{1}{\sqrt{2\pi}\sigma x} e^{-(\log x - \mu)^2 / 2\sigma^2}. \tag{10}\]

For the sake of this example, we use \( \sigma = 1.5 \) and compute \( \mu \) based on the mean of the lognormal distribution \( E[H_j] = e^{\mu + \sigma^2 / 2} \). Fig. 3 (left) illustrates the expected number of useful packets in each frame of this sequence and the same metric in a similar constant-size sequence. As the figure shows, \( E[Z_j] \) matches simulations well and is in fact upper-bounded by \( E[Z^H_j] \). For instance, for \( E[H_j] = 100 \), the lognormal case recovers only 7 useful packets from each frame, which is 28% less than that in the fixed-frame-size case.

Similar observations apply to utility \( U \), which we define as:

\[
U = \frac{E[Z_j]}{(1 - p)E[H_j]} = \frac{E[1 - (1 - p)H_j]}{pE[H_j]}.
\tag{11}\]

Assuming that \( E[H_j] = H \), \( U \) is upper-bounded by \( U^H \):

\[
U \leq U^H = \frac{1 - (1 - p)^H}{pH}.
\tag{12}\]

This result is illustrated in Fig. 3 (right), which depicts \( U \) and \( U^H \) for different values of \( E[H_j] \) and \( p = 0.1 \). For example, for \( E[H_j] = 100 \), \( U \) is only 0.08, which is 20% lower than \( U^H \).

The next question we address is how many useful packets can be recovered in each frame if the pattern of network packet loss is not uniformly random? We penetrate this problem by deriving a general model for \( E[Z^H_j] \) under more realistic packet-loss patterns. Note that since \( U^H \) always provides an upper bound on \( U \) and the exact distribution of \( H_j \) is application-specific (i.e., unknown), the rest of the paper only deals with constant frame sizes and no longer considers variable \( H_j \).

Several studies have shown that MPEG frame sizes can be modeled by a lognormal distribution [37], which explains our interest in it.

\[\text{Fig. 3. Simulation results and models (1), (5) for constant and lognormal frame sizes, respectively (left). The utility of received video for the same two cases (right). In both figures, } p = 0.1.\]

**B. General Loss Model**

In this section, we extend the Bernoulli model and study a more general pattern of packet loss, which is close to that found in the current Internet. Several studies analyzed the characteristics of Internet packet loss and reached a number of conclusions. Some studies (e.g., [43]) found that loss-burst lengths could be modeled as exponential, while several others (e.g., [4], [27], [34]) reported that the distribution of loss-burst lengths was heavy-tailed. We overcome this uncertainty by deriving closed-form models for both cases, as well as the more generic case when loss-burst lengths have an arbitrary distribution.

We explore the recurrent behavior of packet loss using a simple stochastic model from renewal theory. Assume that the packet loss process \( V(t) \) goes through ON/OFF periods, where all packets are lost during each ON period and all packets are delivered during each OFF period. Then, we can write:

\[
V(t) = \begin{cases} 
1 & \text{ON at time } t \\
0 & \text{OFF at time } t
\end{cases}
\tag{13}\]

Suppose that the duration of the \( i \)-th ON period is given by a random variable \( X_i \) and the duration of the \( i \)-th OFF period is given by \( Y_i \) (\( X_i \) and \( Y_i \) may be drawn from different distributions). Fig. 4 illustrates the evolution of alternating process \( V(t) \). The figure also shows that if \( V(t) \) is sampled at a random instant \( \tau_j \) and the process happens to be in the OFF state, the distance to the next packet loss is given by some residual process \( R(\tau_j) \), whose distribution we will derive shortly.

Assume that \( X_i \) and \( Y_i \) are independent of each other and sets \( \{X_i\} \) and \( \{Y_i\} \) consist of i.i.d random variables. Then, the process \( V(t) \) is an alternating renewal process, where each renewal cycle \( W_j \) contains an adjacent pair of ON/OFF periods (i.e., \( W_j = X_j + Y_j \)) and the \( n \)-th renewal occurs at time epoch \( T_n = \sum_{j=1}^{n} W_j \).

Next, we derive long-term network packet loss \( p \), which is the fraction of time that the process is in the ON state.

**Lemma 2:** Under the ON/OFF process model, the long-term network packet loss \( p \) is:

\[
p = P(V(t) = 0) = \frac{E[X_i]}{E[X_i] + E[Y_i]}.
\tag{14}\]
\textbf{Proof:} Notice that network packet loss is the ratio of the total number of lost packets to the total number of transmitted packets, averaged over a sufficiently long period of time. This can be written as:

\[
p = \lim_{t \to \infty} \frac{1}{t} \int_0^t V(u) du. \tag{15}
\]

We can view each ON duration of \( V(t) \) as the amount of reward \( \bar{P}_j = X_i \) for each renewal cycle \( W_j \). Then, the cumulative reward \( C(t) \) of this process over each interval \([0, t]\) is \( C(t) = \int_0^t V(u) du \). Applying the renewal-reward theorem to \( C(t) \), we have \cite{42}:

\[
p = \lim_{t \to \infty} \frac{C(t)}{t} = \frac{E[\bar{P}_j]}{E[W_j]}. \tag{16}
\]

Substituting \( E[\bar{P}_j] \) and \( E[W_j] \) in (16), we get (14). \qed

Given network packet loss \( p \), we are primarily interested in the location of the first ON event after each frame starts, which determines the number of consecutively received packets in that frame. Suppose that random variable \( \tau_j \) represents the time instants when the \( j \)-th frame starts its transmission over the network. Then, we can safely assume that points \( \tau_j \) are uncorrelated with the cycles of packet loss \( V(t) \) since the former is an application-specific parameter, while the latter depends on many factors (such as network congestion and cross-traffic), which are not related to the contents of the streaming traffic. Thus, we can view \( \tau_j \) as being uniformly distributed within each renewal cycle of \( V(t) \) and \( R(\tau_j) \) as the residual life of \( Y_i \) before the next renewal.

Notice that at \( \tau_j \), there are two possible scenarios:

- \( V(t) \) is in the ON state;
- \( V(t) \) is in the OFF state.

In the former case, the amount of useful data recovered in the frame is \( Z_j^H = 0 \). However, in the latter case, this amount will depend on the residual life \( R(\tau_j) \) of the current OFF cycle (see Fig. 4). Denote by \( F_Y(x) \) the distribution of \( Y_i \) and assume that \( E[Y_i] < \infty \). Then, recall that \( F(x) \) can be expressed as \cite{42}:

\[
F(x) = P(R(t) \leq x) = \frac{1}{E[Y_i]} \int_0^x (1 - F_Y(u)) du. \tag{17}
\]

Noticing that the distribution of \( X_i \) does not affect \( E[Z_j^H] \), we have the following result.

\textbf{Theorem 2:} Assuming a fixed frame size \( H \), the expected number of useful packets in a frame is:

\[
E[Z_j^H] = (1 - p) \int_0^H \bar{F}(x) dx, \tag{18}
\]

where \( \bar{F}(x) = 1 - F(x) \) is the tail distribution of \( R(t) \).

\textbf{Proof:} Consider a frame \( j \) that starts at time instant \( \tau_j \). Conditioning on \( V(t) \) being in the OFF state at \( \tau_j \), the number of recovered bits/bytes is the random variable \( Z_j^H = \min(R(\tau_j), H) \), which leads to the following:

\[
E[Z_j^H] = (1 - p) \left( \int_0^H x f(x) dx + H \int_{H}^{\infty} f(x) dx \right), \tag{19}
\]

where term \( 1 - p \) is simply \( P(V(t) = 0) \) and \( f(x) \) is the PDF of \( R(t) \). Using integration by parts, the first integral in (19) becomes:

\[
\int_0^H x f(x) dx = HF(H) - \int_0^H F(x) dx. \tag{20}
\]

The second integral in (19) is:

\[
\int_{H}^{\infty} f(x) dx = H - HF(H). \tag{21}
\]

Adding (20) and (21) and rearranging the terms, we establish (18). \qed

Note that model (18) is based upon limiting distributions of conventional renewal theory \cite{42}, which provides an asymptotic distribution of \( R(t) \) as \( t \to \infty \). In order to verify the model for finite \( t \ll \infty \), we study exponential and Pareto distributions of \( Y_i \) in the next two lemmas.

\textbf{Lemma 3:} For exponential \( Y_i \) with rate \( \lambda \), the expected number of useful packets in a frame is:

\[
E[Z_j^H] = \frac{1 - p}{\lambda} \left( 1 - e^{-\lambda H} \right). \tag{22}
\]

\textbf{Proof:} Since \( F_Y(x) \) is an exponential distribution, from (17) or the memoryless property of exponential distributions, \( F(x) = 1 - e^{-\lambda x} \). Substituting \( \bar{F}(x) = 1 - F(x) = e^{-\lambda x} \) in (18), we get (22). \qed

This model is compared to simulation results in Table II for \( H = 100 \) and \( H = 1,000 \). For this simulation, we use \( E[X_i] = 1/(1 - p) \) and \( E[Y_i] = 1/p \), which leads to \( \lambda = p \). As shown in the table, the model follows simulation results very well. Also note that for this setup, (22) models the same pattern of packet drop behavior described in Theorem 1 (i.e., Bernoulli loss) and observe that the data in Table II matches that in Table I. Further note that as \( H \to \infty \), (22) converges to \( (1 - p)/p \) just as in the Bernoulli case.

Next, we study Pareto-distributed \( Y_i \) and examine how burst lengths of OFF periods affect \( E[Z_j^H] \).

\textbf{Lemma 4:} For Pareto \( Y_i \) with finite mean \( E[Y_i] < \infty \), the expected number of useful packets is:

\[
E[Z_j^H] = \frac{1 - p}{\lambda} \left( 1 - \lambda^H \right). \tag{23}
\]

\textbf{Proof:} Since \( F_Y(x) \) is a Pareto distribution, from (17) or the memoryless property of Pareto distributions, \( F(x) = \frac{1}{x - \alpha} \). Substituting \( \bar{F}(x) = 1 - F(x) = \frac{1}{x - \alpha} \) in (18), we get (23). \qed

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
\textbf{Packet loss} & \textbf{H = 100} & \textbf{H = 1,000} \\
\hline
\textbf{p} & \textbf{Simulations} & \textbf{Model (22)} & \textbf{Simulations} & \textbf{Model (22)} \\
\hline
0.0001 & 99.49 & 99.49 & 951.66 & 951.53 \\
0.001 & 95.06 & 95.06 & 631.44 & 631.48 \\
0.01 & 62.60 & 62.57 & 98.87 & 98.99 \\
0.1 & 8.98 & 8.99 & 9.02 & 9.00 \\
0.2 & 4.01 & 4.00 & 4.02 & 4.00 \\
0.9 & 0.10 & 0.11 & 0.10 & 0.11 \\
\hline
\end{tabular}
\caption{Expected Number of Useful Packets (Exponential Model)}
\end{table}
TABLE III
EXPECTED NUMBER OF USEFUL PACKETS (PARETO MODEL)

<table>
<thead>
<tr>
<th>H</th>
<th>Packet loss p</th>
<th>$\alpha = 2$</th>
<th>$\alpha = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Pareto</td>
<td>Model (23)</td>
</tr>
<tr>
<td>100</td>
<td>0.0001</td>
<td>99.50</td>
<td>99.49</td>
</tr>
<tr>
<td>100</td>
<td>0.01</td>
<td>95.22</td>
<td>95.14</td>
</tr>
<tr>
<td>100</td>
<td>0.1</td>
<td>68.67</td>
<td>66.03</td>
</tr>
<tr>
<td>100</td>
<td>0.2</td>
<td>21.64</td>
<td>15.02</td>
</tr>
<tr>
<td>100</td>
<td>0.9</td>
<td>12.23</td>
<td>7.25</td>
</tr>
<tr>
<td>1000</td>
<td>0.0001</td>
<td>953.73</td>
<td>952.48</td>
</tr>
<tr>
<td>1000</td>
<td>0.01</td>
<td>692.98</td>
<td>666.34</td>
</tr>
<tr>
<td>1000</td>
<td>0.1</td>
<td>237.32</td>
<td>165.23</td>
</tr>
<tr>
<td>1000</td>
<td>0.2</td>
<td>41.77</td>
<td>17.64</td>
</tr>
<tr>
<td>1000</td>
<td>0.9</td>
<td>7.72</td>
<td>7.92</td>
</tr>
</tbody>
</table>

$E[Z_j^H] = \begin{cases} 
\frac{(1-p)\beta}{2-\alpha} & \left(\frac{H}{\beta} + 1\right)^{2-\alpha} - 1 \quad \alpha \neq 2 \\
(1-p)\beta \log \left(\frac{H}{\beta} + 1\right) & \alpha = 2 
\end{cases}$, \hspace{1cm} (23)

where $\alpha$ and $\beta$ are the shape and location parameters of the Pareto distribution, respectively.

**Proof:** Consider a shifted Pareto distribution $F_Y(x) = 1 - (x/\beta + 1)^{-\alpha}$ where $x \geq 0$, $\alpha > 1$, and $\beta > 0$. Notice that the domain of this distribution is $(0, \infty)$, which allows us to construct a well-formed renewal process and model arbitrarily small durations $Y_i$ of OFF periods. Using $E[Y_i] = \beta/(\alpha - 1)$, we have the following distribution of $R(t)$:

$$F(x) = 1 - \frac{1}{E[Y_i]} \int_0^x \left(\frac{u}{\beta} + 1\right)^{-\alpha} du = 1 - \left(\frac{x}{\beta} + 1\right)^{1-\alpha}. \hspace{1cm} (24)$$

Notice that this is a more heavy-tailed distribution than the original one, which implies that residuals $R(\tau_j)$ are expected to be larger than the original OFF periods $Y_i$. Substituting $F(x) = (x/\beta + 1)^{1-\alpha}$ into (18), we have:

$$E[Z_j^H] = (1-p) \int_0^H \left(\frac{x}{\beta} + 1\right)^{1-\alpha} dx. \hspace{1cm} (25)$$

Separately expanding the integral in (25) for $\alpha = 2$ and $\alpha \neq 2$, we get both cases in (23).

We compare this model to simulation results in Table III for $H = 100$ and $H = 1000$, like in the exponential case, we use $E[X_i] = 1/(1-p)$, which leads to $\beta$ being equal to $(\alpha - 1)/p$. First, notice in Table III that simulation results align well with those predicted by the model. Second, observe that the Pareto case delivers more useful packets on average than the exponential case previously shown in Table II. This was expected from the properties of Pareto $F_Y(x)$, which tends to create larger inter-loss gaps than the exponential model. This is schematically shown in Fig. 5 where the Pareto loss events are more bursty and each frame has a higher probability to start within a very large OFF burst.

C. Optimal Preferential Streaming

In this section, we discuss an “optimal” streaming method that can provide high end-user utility and significant quality improvement over that in best-effort streaming. Notice that in order to achieve the maximum end-user utility (i.e., $U^H = 1$), routers must drop the upper parts of the FGS layer during congestion and transmit only the lower parts since consecutive lower portions of the FGS layer can enhance the base layer, while any gaps in the delivered data typically render the remainder of the layer useless. Fig. 6 depicts the difference between the ideal and random drop patterns in an enhancement frame and shows that all dropped packets must occupy the upper portion of the FGS layer for the network to achieve optimality.

Since for a given loss rate $p$, the “optimal” AQM scheme drops $pH$ packets from the upper part of the FGS frame and protects the remaining $H(1-p)$ packets, all received FGS packets are consecutive and thus can be used to enhance the base layer. Hence, the utility of this framework is always 1 regardless of the values of $p$ or $H$. For example, assuming the same scenario with $p = 0.1$ and $H = 100$, preferential streaming delivers ten times more useful packets than best-effort streaming under the Bernoulli model. The main question now is whether optimal streaming is possible in practice and how to achieve it using scalable AQM methods. We address this issue next.
In this section, we introduce a new video streaming mechanism called Partitioned Enhancement Layer Streaming (PELS), which operates in conjunction with priority-queuing AQM routers in network paths. In the PELS framework, applications partition the enhancement layer into two layers and voluntarily mark their packets using different priority classes, which allows network routers to discriminate between packets based on their priority (no per-flow management is required).

Recall that coded video frames carry information that has different importance to the end user — the lower layers are more sensitive to packet loss than the higher layers. The base layer (being most sensitive) is required for displaying video appropriately at the receiver and thus is transmitted using the highest priority class. This ensures that the base layer is dropped only when the entire FGS layer is discarded by the routers.

The reason for splitting the FGS layer into two priorities is also simple to understand. Bytes in the lower part of the FGS layer are more important than those in the higher part because the former includes the information needed to properly decode the latter. Hence, to protect the lower parts of FGS even under moderate congestion (low packet loss), we must next ensure that no end-flow gains anything. Since PELS application sources can arbitrarily mark their FGS packets with high priority (i.e., green), packets and always receive non-negligible service from the network. In fact, starvation in low-priority (i.e., red) queues is equivalent to 100% loss in these queues and has very little effect on the resulting quality since it affects only the upper parts of each enhancement frame (more on this below).

IV. PREFERENTIAL VIDEO STREAMING

A. Router Queue Management

In this section, we discuss queuing disciplines necessary to support PELS and how applications should assign priority to their packets. To separate video traffic from the rest of the flows, the proposed PELS architecture must maintain in each network router two types of queues — the PELS queue and the Internet queue. The PELS queue is further subdivided into green, yellow, and red priority queues to service marked multimedia packets, while the Internet queue serves all other (non-multimedia) Internet traffic in a regular FIFO fashion. To ensure that network bandwidth is shared “fairly” between PELS applications and other Internet traffic, we employ weighted round-robin (WRR) scheduling between the PELS and Internet queues. Recall that WRR can provide a desired level of fairness between several types of traffic by allocating a certain fraction of the outgoing link to each queue as shown in Fig. 7. This allows de-centralized administrative flexibility in selecting the weights and assigning proper “importance” to different classes of traffic on a per-router basis.

It is easy to see that the PELS queue must employ a strict priority queuing discipline to maximize the resulting video quality for a given total throughput budget. Since the higher parts of the FGS frames cannot be decoded without the presence of the lower parts, each router has no reason to transmit the higher parts before sending the lower ones. This implies that network routers must use queuing mechanisms that do not allow low-priority packets to pass until all high-priority packets are fully transmitted. Note that, in general, strict priority queuing is frowned upon since it leads to starvation in low-priority queues and denial-of-service effects for certain flows; however, this situation does not arise in PELS since each flow sends a certain amount of high-priority (i.e., green) packets and always receives non-negligible service from the network. In fact, starvation in low-priority (i.e., red) queues is equivalent to 100% loss in these queues and has very little effect on the resulting quality since it affects only the upper parts of each enhancement frame (more on this below).

Since PELS application sources can arbitrarily mark their packets, we must next ensure that no end-flow gains anything by marking all of its FGS packets with high priority (i.e., green). Such “misbehaving” sources will increase congestion in the green queues, which will result in (uniform) random losses in their base layers and will quickly degrade the resulting quality of their own video. Similarly, end-flows have little incentive in sending too many yellow packets or being congestion-indifferent. Thus, if each application is a selfish, independent entity that attempts to maximize the utility of its video at the receiver, it will send red packets to probe for congestion and back-off (i.e., reduce the total sending rate) during the loss of any red packets to protect the yellow/green queues from upcoming congestion. Note that PELS does not protect against Denial-of-Service (DoS) attacks, in which case it offers similar performance to that of the regular best-effort Internet.

We should make several other observations. First, notice that PELS congestion control (studied later in this paper) assumes certain “constancy” of the end-to-end path (all packets take the same route) and the presence of PELS-enabled AQM at the bottleneck router. The former assumption is common to all flows using congestion control (i.e., multi-path routing and/or route changes make the control loop produce unpredictable results). The latter assumption is also mild since it does not require that all routers deploy PELS at the same time. In fact,
PELS works just as well in a mixed environment where some routers support it and others do not.

Our second observation is that priority queuing in PELS is low-overhead, flexible, does not require support from DiffServ or use of per-flow management, and can be implemented using priority queues available in many existing router hardware/software solutions. Finally, PELS does not require communication between routers and leaves the decisions of how to mark packets to the end-user (i.e., pushes complexity outside the network).

B. FGS Partitioning and Packet Coloring

In a practical network environment (such as the Internet), packet loss and available bandwidth are not constant and change dynamically depending on cross traffic, link quality, routing updates, etc. Hence, streaming servers must often probe for newly available bandwidth as part of congestion control and continuously send low-priority packets under the assumption that these probes (and only they) will get lost during congestion.

Fig. 8 illustrates one possible partitioning of FGS bytes into two priority classes (i.e., yellow and red) that can achieve “optimal” utility discussed in section III.B. The figure shows that the server sends $x_i$ bytes from each enhancement frame $i$ (where $x_i$ is given by congestion control and is derived from $R_{\text{max}}$ using rate scaling algorithms [7]). The transmitted section of each FGS frame is divided into two segments – the lower segment of size $(1-\gamma)x_i$ is all yellow and the upper segment of size $\gamma x_i$ is all red. The division into red and yellow packets depends on how conservative (many red packets and large $\gamma$) or optimistic (few red packets and small $\gamma$) the server wants to be.

In an ideal network with stationary packet loss $p$ in the enhancement layer, the server can select $\gamma$ such that $\gamma x_i$ is equal to $px_i$. This will ensure that all red packets are lost and that exactly $(1-p)x_i$ yellow packets are recovered for decoding (this is the best scenario under any circumstances). In practice, however, keeping red packet loss $p_R$ at 100% is not prudent since any slight increase in $p$ (caused by a new flow joining the network or a change in network conditions) will “spill” the loss into the yellow queue, effectively creating a best-effort FIFO situation for the yellow packets. Thus, proper and dynamic selection of $\gamma$ is important (see the next section).

The other issue to address is congestion control. Even though red queues can be used to isolate increasing packet loss $p$ without reducing the sending rate of the flow (i.e., by proportionally increasing $\gamma$), the resulting situation will lead to “trashing” the network with numerous red packets that eventually get dropped at the bottleneck link. To prevent waste of bandwidth on the path to the bottleneck, the server must implement elastic congestion control and reduce its rate whenever it loses either yellow or red packets (the loss of green packets means that there is not enough bandwidth to support the base layer and no meaningful streaming can continue). Since all flows in our model use PELS and the same congestion control, they all back-off during the loss of red packets and keep the amount of “waste” to a minimum.

C. Selection of $\gamma$

Recall that partitioning of the FGS layer into yellow/red packets attempts to ensure that only the upper sections of each frame are dropped during congestion; however, the performance of PELS depends on the selection of $\gamma$ and the level of congestion at the bottleneck link. In order for PELS to be effective, we must ensure that when flows probe for new bandwidth, they do not incur such high levels of congestion as to force packet loss in the yellow priority queue. Hence, given any control interval $k$ with packet loss $p(k)$ in the FGS layer, how can the server make sure that there will be no loss among yellow packets during interval $k + 1$?

Intuitively, $\gamma$ should be adjusted according to packet loss measured during interval $k$ to keep the resulting red loss $p_R = px_i/\gamma x_i = p/\gamma$ at a certain threshold $p_{\text{thr}}$. The most optimistic approach suggests $p_{\text{thr}} \approx 1$ (which leads to the largest utility $U \approx 1$) and the most pessimistic approach keeps $p_{\text{thr}} \approx p$ (which leads to the best-effort utility in the enhancement layer).

Based upon these observations, we seek a middle ground in which $p_{\text{thr}}$ can be stabilized between 70 and 90% using simple closed-loop control methods that adjust $\gamma$ based on the following rules:

- Increase $\gamma$ when $p$ increases
- Decrease $\gamma$ when $p$ decreases.

Considering this general intuition, we next investigate a proportional controller that adjusts $\gamma$ based on the measured packet loss $p(k)$ and target red packet loss $p_{\text{thr}}$:

$$
\gamma_i(k) = \gamma_i(k-1) + \sigma(p_i(k-1)/p_{\text{thr}} - \gamma_i(k-1)),
$$

where index $i$ represents flow number, $p_i(k)$ is the measured average packet loss in the entire FGS layer for flow $i$ during interval $k$, and $\sigma$ is the controller’s gain parameter. Note that $(1-p_{\text{thr}})\gamma x_i$ is the amount of cushion left by the server for the yellow packets. For example, $p_{\text{thr}} = 0.75$ means that 25% of the red queue works to protect the yellow queue against sudden (unexpected) increase in packet loss.

Note that, in general, the measurement of $p_i(k)$ is coupled with congestion control and should be provided by its feedback loop (we discuss this in section V). Next notice that the controller in (26) is stable if the following is satisfied.

**Lemma 5:** The controller (26) is stable if $0 < \sigma < 2$.

**Proof:** Taking the $z$-transform of (26) we obtain:

$$
\Gamma_i(z) = \frac{z^{-1}\Gamma_i(z) + \sigma(z^{-1}P_i(z)/p_{\text{thr}} - z^{-1}\Gamma_i(z))}{1 - (1-\sigma)z^{-1}}.
$$

(27)
The poles of system (26) are the roots of its characteristic equation $1 - (1 - \sigma)z^{-1} = 0$ (notice that $P_1(z)$ does not depend on $\Gamma_i(z)$). Thus, the system has a single pole $z = 1 - \sigma$. For the control system to be stable, it is sufficient and necessary that the absolute value of $z$ be less than 1 (i.e., $|z| < 1$). Solving this inequality, we obtain $0 < \sigma < 2$.

Note that in a real network environment, feedback delays are often involved. Assuming arbitrary round-trip delay $D_i$ for flow $i$, (26) becomes:

$$\gamma_i(k) = \gamma_i(k - D_i) + \sigma(p_i(k - D_i)/p_{thr} - \gamma_i(k - D_i)).$$ (28)

Then we have a stronger version of the previous lemma, which shows stability of the resulting controller under arbitrary (heterogeneous) delays.

**Lemma 6:** The controller (28) is stable iff $0 < \sigma < 2$.

**Proof:** The $z$-transform of (28) is:

$$\Gamma_i(z) = z^{-D_i}\Gamma_i(z) + \sigma(z^{-D_i}P_i(z)/p_{thr} - z^{-D_i}\Gamma_i(z))$$

$$= \frac{\sigma z^{-D_i}P_i(z)/p_{thr}}{1 - (1 - \sigma)z^{-D_i}}.$$ (29)

The new pole is given by $z = (1 - \sigma)^{1/D_i}$, which leads to the same stability condition $0 < \sigma < 2$.

Next, we derive the effect that (26), (28) have on the packet loss in the red queue.

**Lemma 7:** Assuming that packet loss $p(k)$ converges to some stationary point $p^*$, both controllers (26)-(28) converge red packet loss $p_R$ to $p_{thr}$.

**Proof:** Since $\gamma(k)$ does not change in the steady state, we can write: $\gamma(k) = \gamma(k - 1) = \gamma^*$. From (26), we get:

$$\gamma^* = \frac{p^*}{p_{thr}},$$ (30)

where $p^*$ is the stationary packet loss of the network. Recalling that the red packet-loss can be expressed as $p_R(k) = p(k)/\gamma(k)$, the stationary red packet-loss is given by:

$$\frac{p_R^*}{\gamma^*} = \frac{p^*}{p_{thr}}.$$ (31)

Substituting $\gamma^*$ from (30) into (31), we get $p_R^* = p_{thr}$.

To illustrate that selection of $\sigma$ is important, but not drastically difficult, Fig. 9 depicts the behavior of $\gamma(k)$ with different $\sigma$ (we use a heavy-loss case with $p = 0.5$ and $p_{thr} = 0.75$ in this example). As the figure shows, $\gamma(k)$ is stabilized at the stationary point $\gamma^* = p/p_{thr} \approx 0.67$ when $\sigma = 0.5$, while it is unstable when $\sigma = 3$.

The resulting utility of received video in PELS under dynamically changing $\gamma$ is lower-bounded by the following (assuming that only yellow packets are recovered from the FGS layer):

$$U^H \geq \frac{H(1 - \gamma)}{H(1 - p)} = \frac{1 - p/p_{thr}}{1 - p}.$$ (32)

Thus, the utility of PELS is at least 0.96 for $p = 0.1$ and $p_{thr} = 0.75$ and at least 0.996 for $p = 0.01$ and the same threshold. Although PELS does not achieve “optimality” for $p_{thr} < 1$, it comes very close to it and at the same time avoids the pitfalls of the optimal method.

**V. CONGESTION CONTROL FOR VIDEO**

Congestion control is necessary for streaming applications to provide a high level of video quality to end users and avoid wasting network resources with packets that are eventually dropped in congested queues. Many control methods dynamically adjust the sending rate of end-flows based on network feedback and aim to achieve a stable tradeoff between under-utilization of resources and network congestion (i.e., packet loss).

Since we already assumed that PELS flows do not share common queues with TCP traffic (as shown in Fig. 7), congestion control for PELS is not limited to best-effort, TCP-friendly alternatives. Thus, among recent game-theoretic and optimization methods [21], [23], [24], [28], [29], [30], [31], [32], we selected Kelly’s congestion control framework called proportional fairness [23], since it is stable, efficient, and fair under various network conditions. In this section, we study Kelly controls, apply them to PELS streaming, and investigate whether their performance provides the necessary foundation for achieving our goals of smooth, high-quality video streaming.

In general, it is important to remember that PELS is independent of congestion control and can be utilized with any end-to-end or AQM scheme. Thus, the complexity of implementing Kelly controls inside routers should be decoupled from that of PELS since the latter does not require the presence of the former. Kelly controls are studied here as an example of one possible scheme that supplements PELS with smoothly changing rates. We leave the study of additional methods for future work.

**A. Continuous-Feedback Control**

Although Kelly’s controls have attracted significant attention, their application to video streaming is limited to [7] in which Dai et al. use an application-friendly form of the controller given by:

$$\frac{dr(t)}{dt} = \alpha - \beta p(t)r(t),$$ (33)

where $r(t)$ is the rate of the flow, $p(t)$ is packet loss (feedback from the network), $\alpha$ and $\beta$ are constant parameters. Since in real applications, rate adjustment is not continuous, we use a discrete form of (33). However, notice that the classical discrete Kelly control studied by [21] and others shows stability problems when the feedback delay becomes large [44].
Hence, we employ a slightly modified discrete version of this framework called Max-min Kelly Control (MKC) [44]:

\[ r_i(k) = r_i(k - D_i) + \alpha - \beta r_i(k - D_i) p_i(k - D_i^-), \]

where \( r_i(k) \) is the rate of source \( i \) during interval \( k \), \( D_i^- \) is the backward delay from the router to source \( i \), \( D_i \) is the round trip delay of flow \( i \), and packet loss \( p_i \) is fed back from the most-congested resource \( l \) (this provides max-min resource allocation instead of proportionally fair). The packet loss is computed inside router \( l \) at discrete intervals and inserted into all passing packets:

\[ p_l(k) = \sum_{j \in S_l} r_j(k - D_j^-) - C_l \sum_{j \in S_l} r_j(k - D_j^-), \]

where \( S_l \) is the set of sources sending packets through router \( l \), \( D_j^- \) is the forward delay from source \( j \) to the router, and \( C_l \) is link capacity of router \( l \). The stability of system (34)-(35) is formalized as follows.

**Lemma 8:** System (34)-(35) is locally asymptotically stable under heterogeneous delays iff \( 0 < \beta < 2 \).

**Proof:** See [44].

We apply (34) for rate control in PELS streaming and investigate its control characteristics including:

- Convergence to a single stationary point;
- Fairness between flows.

From (34) and (35), we next derive stationary rates \( r_i^* \) of end-flows in the equilibrium point and show that (34) has no oscillations in the steady state.

**Lemma 9:** Regardless of the feedback delay, the stationary rate \( r_i^* \) of each flow is:

\[ r_i^* = \frac{C_l}{N} + \frac{\alpha}{\beta^*}. \]

**Proof:** Since the rate of each flow does not change with time in the steady state, \( r_i(k) = r_i(k - D_i) = r_i^* \). Using this observation in (34), we get

\[ r_i^* = \frac{\alpha}{\beta^*}, \]

where \( p^* \) is the stationary packet loss. From (37), stationary points of all flows are the same since the system is stable and converges packet loss \( p(k) \) to \( p^* \). Hence, we can write \( r_i^* = r_j^* = r^* \) and thus we have from (35):

\[ p^* = \frac{N r^* - C_l}{N r^*}. \]

Substituting (38) into (37), we get (36).

Thus, unlike AIMD or TCP, MKC does not penalize flows with higher RTT and further converges to a single stationary point (i.e., exhibits no oscillations in the steady-state).

Next notice that priority queuing in PELS imposes increased delays on red packets and that the utilization of each priority class directly affects delay characteristics of all queues with lower priority. Since green packets have much smaller queuing delays than yellow or red packets, it is tempting to provide feedback only in green packets. However, since the base layer is usually sent at significantly lower rates than the enhancement layer, this method introduces unnecessary feedback delays due to large inter-packet spacing of the base layer. Thus, we conclude that network feedback must be inserted by the router into all passing packets (regardless of their color) for timely delivery to the end-flows. Below, we discuss implementation details and mechanisms to discard out-of-sequence (i.e., outdated) feedback that may arrive in red/yellow packets.

### B. PELS Implementation

We implemented new agents and a priority-based AQM mechanism for PELS streaming in the ns2 network simulator [33]. PELS application sources mark their packets with three priority levels (i.e., green, yellow, and red) and employ MKC for rate control. Computation of packet loss \( p(k) \) is performed by the router on a discrete time scale of \( T \) time units and then injected into the header of each packet passing through the router (note that feedback information is a queue-specific metric). Each new computation of \( p(k) \) increases router’s local epoch number \( z \) to prevent sources from reacting to the same feedback more than once as well as to suppress outdated values of \( p(k) \) created by re-ordering inside PELS queues.

Label (router ID, \( z \), \( p(k) \)) is provided to end-flows through the header of the packets queued at the bottleneck link.

Once received by the end-user, feedback \( p(k) \) is sent in ACKs to the source, which applies rate adjustments according to (34) if and only if it has not seen this feedback sequence \( z \) before. The use of epoch numbers allows the source to keep the frequency of its control loop in sync with that of the router and ensures stability of the resulting system.

We next describe the above two algorithms in more detail. Upon arrival and queuing of a packet \( j \), the router increments its local counter \( S \) by the size \( s_j \) of the packet: \( S = S + s_j \). Then, once every \( T \) time units, the router computes new total rate \( R \), new packet loss \( p \), increments its epoch number \( z \), and resets the byte counter:

\[ R = \frac{S}{T}, p = \frac{R - C}{R}, z = z + 1, S = 0. \]

To verify the “freshness” of feedback, each PELS source \( i \) checks feedback sequence number \( z \) in the acknowledgment and ignores feedback with \( z \) less than or equal to its current epoch number \( z_i \); otherwise, \( z_i \) is set to \( z \) and a new sending rate is computed using (34). When there are multiple routers along an end-to-end path, each router compares its \( p_i \) with that inside arriving packets and overrides the existing value only if its packet loss is larger than the current loss recorded in the header. End-flows use the router ID field to keep track of feedback freshness and react to possible shifts of the bottlenecks.

Selection of interval \( T \) depends on the desired responsiveness of the PELS framework to network conditions, but does not affect stability of the system as a whole. To analytically reflect the implementation of the PELS framework where the router purposely delays its feedback by \( T \) units, we need to modify the model of packet loss in (35) to become:

\[ p_l(k) = \sum_{j \in S_l} r_j(k - T - D_j^-) - C_l \sum_{j \in S_l} r_j(k - T - D_j^-). \]
Stability of system (34)-(40) is proved using the same arguments as in Lemma 8 [44] and is omitted from this paper.

VI. SIMULATION RESULTS

In this section, we present simulation results of PELS including the properties of $\gamma(k)$, MKC congestion control, PELS queuing delay, and PELS video quality. We start by describing the simulation setup.

A. Simulation Setup

For ns2 simulations, we use two simple topologies shown in Fig. 10 and 11. In Fig. 10, there are several PELS and TCP sources connected to a path with a single bottleneck link. The capacity of the bottleneck is 100 mb/s, while the access links are 200 mb/s each. In all simulations, one video frame (48,000 bytes including the base layer) consists of 240 packets, 200 bytes each (these numbers are derived from MPEG-4 coded CIF Foreman). We mark 40 packets in each frame as green to indicate that they will be lost in the network and protect the yellow queue). The topology in Fig. 11 is a simple multi-link path selected to demonstrate MKC’s max-min fairness and contrast it with Kelly’s proportional fairness. The two bottleneck links $R_1 - R_2$ and $R_2 - R_3$ have the same capacity 4 mb/s, while other links are 8 mb/s.

We start our investigation with the behavior of $\gamma(k)$.

B. Stability Properties of $\gamma$

In this section, we show simulation results regarding stability of $\gamma(k)$ computed by end-flows using dynamically varying packet loss $p(k)$. Fig. 12 (left) shows the evolution of $\gamma(k)$ obtained by running PELS streaming simulations in ns2 with two different average packet-loss rates and $\sigma = 0.5$. In the beginning, $\gamma$ drops from the initial value of 0.5 to the lowest possible threshold $\gamma_{\text{low}} = 0.05$ since there is no packet loss (i.e., the flows probe for new bandwidth). When packets start being dropped during congestion, $\gamma$ increases until it is stabilized at $\gamma^* = p^*/p_{\text{thr}}$. Small oscillations of $\gamma(k)$ after it reaches the stationary point is caused by small variation in feedback $p(k)$.

Fig. 12 (right) illustrates red packet drop rates $p_R(k)$ corresponding to the values of $\gamma(k)$ on the left side of the figure. As shown in the figure, red packet loss does in fact stabilize at the target threshold rate $p_{\text{thr}} = 75\%$ regardless of the value of $p$ (i.e., 7% or 14%). Since the red loss never reaches 100%, all of the yellow packets are protected and experience (ideal) zero-loss conditions.

C. Delay Characteristics of PELS

Recall that AQM routers in the PELS framework employ three priority queues for preferential treatment of green, yellow, and red packets. Fig. 13 illustrates delays of green (top), yellow (center), and red (bottom) packets. Theses delays are obtained by running ns2 simulations in which at every 50 seconds, two new flows entered the system with the initial rate of 128 kb/s (i.e., the rate of the base layer).

First notice that green and yellow packets have very small delays compared to those of red packets. The average delays of green and yellow packets are only 16 and 25 ms, respectively, while the average delays of red packets reach as high as 350 ms. Further notice that after 100 seconds, red packet delays increase every 50 seconds since each new flow further reduces the available bandwidth and increases congestion in the red queue. These results are expected from the use of priority queuing in the routers and have no harmful effect on PELS flows as loss or delays in the red queue have minimum impact on the video quality (in fact, the purpose of red packets is to be lost in the network and protect the yellow queue).

Further note that since red/yellow packets are randomly delayed and/or reordered with respect to green packets, the application must wait some time before declaring them lost. Since retransmission is not involved, this decision is very simple, especially in the context of video. Each streaming packet...
session starts with a certain buffering (startup) delay. This allows the decoder to fill up its buffer to a certain threshold, which is used throughout the session to overcome network delay jitter and (in our case) allow delayed red/yellow packets to arrive before their decoding deadlines. Thus, if red/yellow packets are not received before their corresponding frame is scheduled for playback, we declare such packets lost and proceed to the next frame. No additional handling of delayed (or reordered) red/yellow packets is necessary.

D. Properties of PELS Congestion Control

We next study characteristics of MKC congestion control coupled with the PELS queuing framework. We perform two sets of simulations using the setup in Figs. 10 and 11 in order to examine fairness and convergence properties of MKC. Fig. 14 (left) illustrates convergence of PELS flows over a single-link path for \( \alpha = 20 \text{ kbps} \) and \( \beta = 0.5 \). For this simulation, we first start 18 flows \( F_1, \ldots, F_{18} \) with the initial rate \( r_0 = 128 \text{ kbps} \). After flows \( F_1, \ldots, F_{18} \) reach their equilibrium rates (which are 2.78 mb/s each, or 50.04 mb/s total), additional two flows \( F_{19} \) and \( F_{20} \) start 50 seconds apart with the same initial rate and at \( t = 50 \) and \( t = 100 \) seconds, respectively. As the figure shows, flow \( F_{19} \) converges to its new stationary rate (i.e., 2.63 mb/s) 30 seconds after it starts at \( t = 50 \) seconds. It maintains the stationary rate until another flow \( F_{20} \) is injected into the system at \( t = 100 \) seconds. After another 30 seconds, all flows converge to the new fair rate of 2.5 mb/s.

Fig. 14 (right) depicts simulation results of MKC control over a multi-link path. For this simulation, we only use three PELS flows \( G_1, G_2, \) and \( G_3 \), each of which takes a different path shown in Fig. 11. As shown in Fig. 14 (right), all flows \( G_1, G_2, \) and \( G_3 \) converge to the steady rate of 1 mb/s. Note that this equilibrium is a max-min fair allocation of the bottleneck bandwidth at each of the links \( R_1 \rightarrow R_2 \) and \( R_2 \rightarrow R_3 \). Notice that under the same conditions, proportional fairness would favor flows \( G_2 \) and \( G_3 \) by giving them double the rate of \( G_1 \), which happens because proportional fairness rewards flows that take shorter routes.

E. PSNR Quality Evaluation

In this section, we compare the proposed preferential streaming scheme with the best-effort method using PSNR quality curves. Through simulation, we obtained packet loss statistics of each FGS frame and then applied them to the Foreman video sequence offline. We enhanced each base-layer frame using consecutively received FGS packets and plotted PSNR quality curves accordingly. Aggregate packet loss was calculated in the routers at \( T = 40 \text{ ms time intervals} \).

Our main challenge in this section was to properly select a “generic” brand of alternative approaches that adequately represents existing (non-PELS) proposals. Although there are numerous methods of streaming video over the Internet (including TCP, FEC-protected transmission, and various non-AIMD methods), we aim to compare PELS with an alternative framework that: 1) does not retransmit any lost packets; and 2) does not send any error-correcting codes. Since no such framework exists outside of IntServ’s bandwidth-reservation methods, we use a surrogate method called “best-effort-MKC,” which runs AQM-enabled MKC and unconditionally protects the base layer from packet loss. If loss is allowed in the base layer and retransmission is suppressed, best-effort streaming simply becomes impossible due to propagation of losses throughout each GOP (Group of Pictures). Thus, we protected the entire base layer in “best-effort-MKC” and allowed random loss only in the FGS layer to keep this approach even remotely competitive with PELS.

We first examine PSNR of the Foreman sequence reconstructed with 10% network packet loss in Fig. 15 (left). As shown in the figure, “best-effort-MKC” streaming improves the base-layer PSNR by 24% on average, while PELS enhances it by 60%. Next, we examine a case with higher packet loss. Fig. 15 (right) illustrates the PSNR curve of the same Foreman sequence reconstructed with 19% packet loss. In this case, while the “best-effort-MKC” method improves the base-layer PSNR only by 16%, PELS improves it by 55%.

From the curves in Fig. 15, we also observe that the PSNR of best-effort streaming varies by as much as 15 dB (even though the sending rates of MKC are perfectly smooth) and provides a highly-fluctuating quality that is similar to that achievable with AIMD [7]. On the contrary, PELS maintains a much higher PSNR throughout the entire sequence and keeps quality fluctuation to a minimum, which can be further reduced using sophisticated R-D scaling methods [7] (not used in this work).
conclusively established that uniform random loss is damaging to realistic stochastic loss models for streaming video and scalable video traffic in best-effort networks and proposed a preferential streaming framework (PELS) that can provide a provably optimal utility to future multimedia applications using low-overhead, scalable AQM methods. We also offered a realistic stochastic loss model for streaming video and conclusively established that uniform random loss is damaging to streaming applications, while priority dropping can significantly improve the end-user’s utility of the network.

VII. CONCLUSIONS

This paper studied the effect of random packet loss on scalable video traffic in best-effort networks and proposed a preferential streaming framework (PELS) that can provide a provably optimal utility to future multimedia applications using low-overhead, scalable AQM methods. We also offered a realistic stochastic loss model for streaming video and conclusively established that uniform random loss is damaging to streaming applications, while priority dropping can significantly improve the end-user’s utility of the network.

REFERENCES


