

Adaptive Receiver-Driven Streaming from Multiple Senders

Nazanin Magharei, Reza Rejaie
Department of Computer and Information Science
University of Oregon
{nazanin,reza}@cs.uoregon.edu

ABSTRACT

This paper presents design and evaluation of an adaptive streaming mechanism from multiple senders to a single receiver in Peer-to-Peer (P2P) networks, called P2P Adaptive Layered Streaming, or *PALS*. *PALS* is a receiver-driven mechanism. It enables a receiver peer to orchestrate quality adaptive streaming of a single, layer encoded video stream from multiple congestion controlled senders, and is able to support a spectrum of non-interactive streaming applications. The primary challenge in design of a multi-source streaming mechanism is that available bandwidth from each peer is not known a priori, and could significantly change during a session. In *PALS*, the receiver periodically performs quality adaptation based on aggregate bandwidth from all senders to determine (i) overall quality (i.e. number of layers) that can be collectively delivered from all senders, and more importantly (ii) specific subset of packets that should be delivered by each sender in order to gracefully cope with any sudden change in its bandwidth. Our detailed simulation-based evaluations illustrate that *PALS* can effectively cope with several angles of dynamics in the system including: bandwidth variations, peer participation, and partially available content at different peers. We also demonstrate the importance of coordination among senders and examine key design tradeoffs for the *PALS* mechanism.

Keywords

Quality adaptive streaming, Peer-to-peer networks, Congestion control, Layered encoding

1. INTRODUCTION

During recent years, Peer-to-Peer (P2P) overlays have become an increasingly popular approach for streaming multimedia content from a single source to many receivers without any special support from the network (e.g., IP multicast or content distribution infrastructure) [1, 2]. A P2P Streaming mechanism requires two key components: (i) *Overlay Construction* mechanism that determines how participating peers connect together to collectively form an overlay, and (ii) *Content Delivery* mechanism that manages how the content is being streamed to each peer through the overlay. The

goal is to maximize delivered quality to individual peers while minimizing the overall network load associated with content delivery.

Previous studies have often adopted the idea of application level multicast where participating peers form a tree structure and each peer simply “pushes” all (or a specific subset) of its received content (e.g., packets of a certain layer) to its child peers (e.g., [3, 4, 5]). This class of solutions have primarily focused on the design of overlay construction mechanism to maintain an optimal tree structure in order to minimize network load. However, they often incorporate simple push-based content delivery with static content-to-parent mapping. This class of solutions are unable to maximize delivered quality to individual peers since each peer only receives content from a *single* parent peer who may not have (or may not be willing to allocate) sufficient outgoing bandwidth to stream content with the desired quality to its child peer. This problem is further aggravated by the following issues: (i) heterogeneity and asymmetry of access link bandwidth among participating peers[6], (ii) dynamic of peer participations, and (iii) the competition among peers for available bandwidth from parent peers. A promising approach to maximize delivered quality to individual peer is to allow a receiver peer to receive content from multiple parent peers. If parent peers are properly selected by the overlay construction mechanism (e.g., to avoid a shared bottleneck), they can provide higher aggregate bandwidth and thus *stream* higher quality content to the receiver. This multi-sender approach could also lead to a better load balancing among parents and across the network.

Efficient streaming of content from multiple sender peers is challenging. We assume that all connections between peers perform TCP-friendly congestion control [7, 8] to ensure that P2P streaming applications can properly co-exist with other applications. This implies that the available bandwidth from a parent is not known a priori and could significantly change during a session. Given the variable nature of available bandwidth, the commonly used *static* approach of receiving separate layers of a layer encoded stream from each sender [3, 5]) can easily lead to a poor performance. More specifically, this static approach results in poor quality when the available bandwidth from a sender is lower than one layer bandwidth, or becomes inefficient when a sender has significantly higher than one layer bandwidth but it is not used for delivery of other layers. To accommodate the variations in available bandwidth, a *dynamic* coordination mechanism among senders can be deployed in order to *adaptively* determine: (i) the maximum quality (i.e., number of layers) that can be delivered by all senders, and (ii) the proper distribution (or mapping) of the target quality among senders in order to fully utilize their available bandwidth while ensuring in-time delivery of individual packets.

This paper presents a receiver-driven coordination mechanism for quality adaptive streaming from multiple congestion controlled sender peers to a single receiver peer, called *P2P Adaptive Layered Streaming* or *PALS*. Given a set of sender peers, the receiver passively monitors the available bandwidth from each sender. Then, it periodically determines the target quality (*i.e.*, the number of layers) that can be streamed from all senders, identifies required packets that should be delivered during the next period, and requests a subset of required packets from each sender. This receiver-driven approach not only maximize the delivered quality but also ensures its stability despite the variations in available bandwidth. More importantly, *PALS* achieves this goal with a relatively small amount of receiver buffering (*e.g.*, tens of seconds worth of playout). Therefore, *PALS* can accommodate a spectrum of *non-interactive* streaming applications ranging from playback to live (but non-interactive) sessions in P2P networks.

We note that *PALS* is a receiver-driven mechanism for streaming (*i.e.*, content delivery) from multiple senders. Therefore, it must be deployed in conjunction with an overlay construction mechanism (*e.g.*, [9]) that provides information about potential parents (*i.e.*, senders) to each peer. Clearly, behavior of the overlay construction mechanism affects overall performance of content delivery across the entire group. In this paper, our goal is to study the streaming delivery of content from multiple senders to a single receiver. Therefore, we assume that a list of potential senders are provided to a receiver peer. Further details of the overlay construction mechanism and global efficiency of content delivery are outside the scope of this paper and will not be discussed. While we motivated *PALS* mechanism in the context of P2P streaming, it should be viewed as a generic coordination mechanism for streaming from multiple congestion controlled senders across the Internet.

The rest of this paper is organized as follows: We sketch an overview of the design space for a multi-sender streaming mechanism in Section 2. Section 3 provides an overview of *PALS* mechanism and describes its key components. We present our simulation-based evaluations in Section 4. In Section 5, we present the related work. Finally, Section 6 concludes the paper and presents our future plans.

2. EXPLORING THE DESIGN SPACE

Before describing the *PALS* mechanism, we explore the design space for a multi-sender streaming mechanism to clarify design issues and justify our design choices. Note that streaming content should have a layered structure in order to accommodate bandwidth heterogeneity among peers by enabling each peer to receive a proper number of layers. To design a multi-sender streaming mechanism, there must be a coordination among senders to address the following two key issues:

- *Aggregate Quality*: How is the aggregate deliverable quality from all senders determined? *i.e.*, how many layers can be delivered by all senders?
- *Content-to-Sender Mapping*: How is the aggregate deliverable quality mapped among senders? *i.e.*, which (or part of) layer should be delivered by each sender?

If the available bandwidth from each sender is known, the aggregate quality can be easily determined as the number of complete layers that can be streamed through the aggregate bandwidth. The

number of mapped layers to each sender is proportional to its contribution to the aggregate bandwidth. This approach to layer-to-sender mapping raises the issue of how the residual bandwidth from each sender should be used. If the residual bandwidth from each sender is not used for delivery of partial layer, the residual bandwidth across different senders can not be accumulated and thus the aggregate delivered quality can not be maximized. For example, if two senders provide 2.6 and 3.7 layer bandwidth, they can stream 6 layers only when their residual bandwidth (0.6 and 0.7 layer) are utilized for delivery of partial layer. Assigning partial layer to a sender requires proper division of a layer across multiple senders (*i.e.*, which particular packets of a layer are determined by each sender).

Aside from the above design choices for a coordination mechanism, a multi-sender streaming mechanism may perform coordination in an adaptive or static fashion, and should determine where the machinery of coordination should be placed as follows:

(i) *Adaptive vs Static Coordination*: The coordination mechanism can be invoked just once at the startup phase to determine the aggregate quality and its mapping among senders. Such *static* approach is feasible when available bandwidth from senders are known and stable. However, if senders are congestion controlled, available bandwidth from each sender is not known a priori and can significantly change during a session. In such dynamic environment, the coordination mechanism should be invoked periodically in order to adaptively determine aggregate quality and its mapping among senders. In adaptive coordination approach, proper adaptation period should be selected in order to achieve responsiveness to bandwidth variations while ensuring instability of delivered quality.

(ii) *Placement of the Coordination Machinery*: The final issue is to determine where the coordination machinery should be implemented. The coordination mechanism can be implemented either in a distributed fashion by all senders or in a central fashion by a single sender or receiver peer. In both approaches, senders should be informed about the status of the receiver, including its buffer state, playout time and lost packets. Distributed coordination requires active participation of all senders and close interactions among them. Such approach is likely to be sensitive to pair-wise delay among senders and sender dynamics, and results in a higher volume of coordination traffic. In central approach, the receiver is in a unique position to implement the coordination mechanism since it is the only *permanent* member of the session, (*i.e.*, receiver-driven coordination). Furthermore, the receiver has a complete knowledge about delivered (and thus lost) packets, available bandwidth from each sender, and aggregate delivered quality. While the receiver cannot predict the future available throughput for each sender, it should be able to leverage a degree of multiplexing among senders to its advantage. Another advantage of the receiver-driven approach is that it does not required a significant processing overhead by the senders which provides a better incentive for senders to participate. The coordination overhead for a receiver-driven approach should not be higher than associated overhead for any conceivable distributed coordination among senders.

In summary, PALS adopts a receiver-driven coordination mechanism in an adaptive fashion in order to minimize coordination overhead, maximize delivered quality, and gracefully accommodate dynamics of bandwidth variations.

3. PALS MECHANISM

PALS is a coordination mechanism for streaming a single content (*e.g.*, video stream) from multiple senders over the Internet. An

s_j	Sender j	$SRTT_j$	EWMA RTT from s_j
L_i	Layer i	$SRTT_{max}$	Max. value among $SRTT_j$
n	No of active layers	Δ	Length of sliding window
N	Max. no of layers	K	Estimated no of incoming packets during Δ
T_{ewma}	EWMA aggregate BW	C	Per layer BW
T_{ewma}^j	EWMA BW from s_j	N_c	No of packet assignment rounds
buf_i	Buffered data for L_i	tp_r	Receiver's playout time
bw_i	Allocated BW for L_i	$Delay_r$	Maximum delay between senders & receiver
$BUF[i]$	Target buffer for L_i	BUF_{add}	Total buffered data before adding a layer
$PktSize$	Packet size	τ	Look ahead interval
r	Degree of redundancy	p	period of added redundancy

Table 1: Summary of Notations

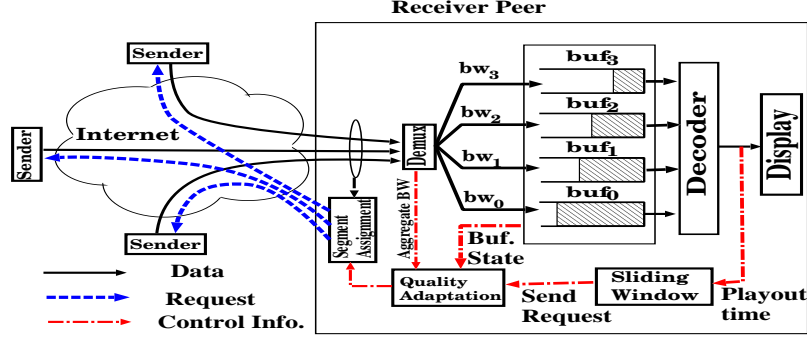


Figure 1: Internal Architecture of a PALS receiver

overlay construction mechanism provides information about a sufficient number of senders¹ to each peer in a demand-driven fashion. The receiver contacts a proper number of senders to serve as its parents. Each selected sender establishes a separate congestion controlled UDP connection to the receiver (*i.e.*, using RAP [7] or TFRC [10]). We assume that senders are heterogeneous and scattered across the Internet. This implies that senders may have different available bandwidth and round-trip-time (RTT) to the receiver, and any subset of them might reside behind a shared bottleneck. Because of the inherent dynamics of peer participation in P2P networks, a sender may leave at any point of time. To accommodate bandwidth heterogeneity among senders, we assume that video streams are layer encoded. However, the framework can be easily extended to accommodate multiple description encoding as well. For clarity of the discussion, we do not consider stream bandwidth variability in this paper and assume that all layers have the same constant-bit-rate (C). General information about the delivered stream (*e.g.*, maximum number of layers (N), layer bandwidth) can be provided to the receiver along with the list of sender peers during the initial setup phase.

To simulate live but non-interactive streaming sessions, we assume that each sender can only provide a limited window of future packets to a receiver. Once a receiver selects a group of sender peers, it delays its playout time (tp_r) with respect to the minimum playout time among selected senders (tp_i), ($tp_r = \text{MIN}(tp_i) - \text{Delay}_r$). This ensures that all senders can provide a window of Delay_r seconds worth of future packets relative to the receiver playout time. In a nutshell, all participating peers in a session are viewing the same stream but the playout time of each peer is delayed where the delay

is roughly proportional to its hop-count distance from the source through the overlay. Delay_r is an important configuration parameter that is controlled by the receiver. As the receiver increases Delay_r , its session becomes closer to the playback mode since it has more room to request future packets. Table 1 summarizes the notation that we use throughout this paper.

3.1 An Overview

The primary design goal of *PALS* is to effectively utilize available bandwidth from each sender to maximize delivered quality while maintaining its stability despite independent variations in bandwidth from different senders. The basic idea in *PALS* is simple and intuitive. The receiver peer monitors the available bandwidth from its parents and periodically requests a list of packets from each parent. Each parent peer delivers requested packets to the receiver in the given order through a congestion controlled connection. In a nutshell, content of delivered packets are determined by the receiver whereas the rate of packet delivery from each sender is controlled by a congestion control mechanism.

The machinery of the *PALS* protocol is mostly implemented at the receiver, as shown in Figure 1. The receiver passively monitors the Exponentially Weighted Moving Average (EWMA) bandwidth from each sender (T_{ewma}^j) and thus can determine EWMA of aggregate bandwidth from all senders (T_{ewma}). The receiver deploys a *Sliding Window* (SW) scheme to properly identify timestamps of required packets for each window as playout time progresses, and ensures in-time delivery of requested packets as we discuss in subsection 3.2. At a beginning of each window (Δ), the receiver assumes that the current value of T_{ewma} remains unchanged for one window and takes the following steps: First, it estimates the total number of incoming packets (K) during this window: $K =$

¹Throughout this paper, we use the terms “sender”, “parent” and “sender peer” interchangeably.

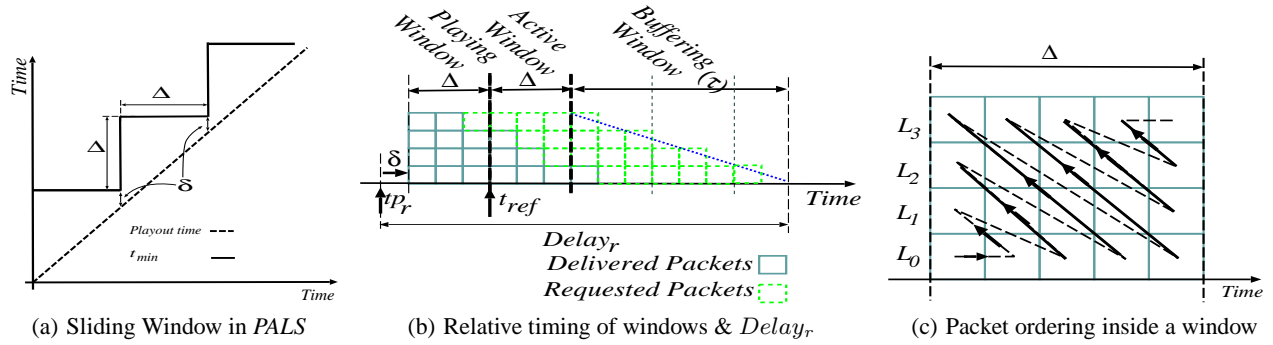


Figure 2: Sliding window & packet ordering in PALS

$$\frac{T_{ewm} * \Delta}{PktSize}$$

Second, the *Quality Adaptation* (QA) mechanism is invoked to determine (i) the number of *active* layers that can be played during this window (n), and (ii) an ordered list of required packets for active layers that should be delivered during this window. The required packets are determined based on the estimated budget of incoming packets (K), stream bandwidth (i.e., $n \times C$) and receiver's current buffer state ($buf_0, buf_1, \dots, buf_n$). For example, if the receiver expects to receive 500 packets during a period ($K = 500$) where four layers are currently being played, the QA mechanism may allocate $k_0 = 200$, $k_1 = 150$, $k_2 = 100$, $k_3 = 50$ packets to layer $L_0 \dots L_3$, respectively. By controlling the distribution of incoming packets among layers, the QA mechanism loosely controls the distribution of aggregate bandwidth among active layers (i.e., bw_0, bw_1, \dots, bw_n) during one window, which in turn determines evolution of receiver's buffer state. Further details on the quality adaptation mechanism is described in subsection 3.3.

Third, given an ordered list of required packets for a period, the *Packet Assignment* (PA) mechanism divides selected packets into disjoint subsets (possibly from different layers), and sends a separate request to each sender. Each request contains an *ordered* list of assigned packets to the corresponding sender. The number of requested packets from each sender is proportional to its contribution in aggregate bandwidth. Therefore, senders are likely to deliver all the requested packets during the corresponding window. Each sender only maintains a single list of pending packets for each receiver peer and simply delivers requested packets in the given order at the rate that is determined by its congestion control mechanism. Ordering the list of requested packets from each sender allows the receiver to prioritize requested packets based on its own preferences (e.g., based on encoding-specific information). This in turn ensures graceful degradation in quality when bandwidth of a sender suddenly drops since available bandwidth is utilized for delivery of more important packets. The receiver can send its request to each sender through an out-of-band TCP connection or piggy-back them with the ACK packets to that sender. Further details of the packet assignment mechanism is presented in subsection 3.4.

3.2 Sliding Window

Once the receiver initiates media playback, PALS deploys a sliding window scheme in order to identify and properly prioritize required packets in each window to accommodate "in-time" delivery of requested packets. The sliding window scheme works as follows: the

receiver maintains a window of time $[t_{ref}, t_{ref} + \Delta]$ called *active window*. This window represents a range of timestamps for packets (from all active layers) that must be requested during one window. t_{ref} and Δ denote the left edge and length of the window, respectively. As the playout time progresses, the window is slid forward in a step-like fashion ($t_{ref} \leftarrow t_{ref} + \Delta$) when the gap between the left edge of the window and the playout time reaches a minimum threshold δ , ($t_{ref} - tp_r \leq \delta$). Figure 2(a) shows the relative position of the window with respect to the playout time that clearly demonstrates the step-like sliding strategy. This sliding strategy allows the receiver to request and receive packets for a new window while the receiver is playing delivered packets for the previous window. Periodic sliding accommodates in-time delivery of requested packets since their timestamps are sufficiently (at least $\delta + \Delta$) ahead of the receiver's playout time.

Figure 2(b) depicts status of the window right after a sliding has occurred. The requested packets for n active layers in a window can be divided into the following three groups based on their timestamp:

- *Packets from Previous Window* ($t_{ref} - \Delta \leq ts < t_{ref}$): these are packets of the n active layers that are missing from the previous window, also called *playing window*, often due to packet loss in the network. Therefore, the ratio of these packets are usually very small. Since there is still sufficient time for in time delivery of these packets, the receiver can perform explicit loss recovery by re-requesting these packets.
- *Packets from Active Window* ($t_{ref} \leq ts < t_{ref} + \Delta$): All missing packets of the n active layers within this window must be requested at this point. Because of the receiver buffering, some of these packets are often delivered ahead of time. This implies that required budget for requesting these packets is often less than one window worth of packets.
- *Packets from Buffering Window* ($t_{ref} + \Delta \leq ts < t_{ref} + \Delta + \tau$): Any excess packet budget can be leveraged to request packets from the buffering window. τ is the length of the buffering window which is called the LookAhead interval. These packets are determined and ordered by the QA mechanism in order to shape up receiver's buffer state as we discuss in subsection 3.3. Note that the timestamp of requested future packets (i.e., size of the buffering window) is determined by the availability of future data which is specified by $Delay_r$, i.e., $\tau = Delay_r - (2 * \Delta + \delta)$.

At macro level, the above three groups of selected packets are ordered based on their corresponding windows, *i.e.*, first packets of the playing window, then active window, and finally buffering window. Micro-level ordering of selected packets within the buffering window is determined by the QA mechanism as we describe in subsection 3.3. Packets within the playing and active windows are ordered based on the following strategy: Given two packets a and b with timestamps ts_a and ts_b from layer L_a and L_b ($L_b > L_a$), packet b is ordered before a only if the following two conditions are satisfied, $\frac{n}{\Delta} > \frac{|L_a - L_b|}{|ts_a - ts_b|}$ and $ts_a > ts_b$, where n denotes the number of active layers. This approach simply orders packets in a diagonal pattern with the slope of $\frac{n}{\Delta}$ (as shown in Figure 2(c)). This ordering strategy ensures that packets from lower layers or with lower timestamps are given higher priority. Therefore, lower priority packets that may not be delivered will be located at the right, top corner of the window and have more time for delivery in the next window. As the above condition indicates, the slope of the ordering adaptively changes with n to strike the balance between importance of lower layers and shorter time for delivery of packets with lower timestamps. Clearly, other ordering patterns can be adopted if layer number or timestamp has a higher priority.

The window size (Δ) is a key parameter that determines the trade-off between stability of delivered quality and responsiveness of the QA mechanism to variations of aggregate bandwidth. Decreasing window size improves responsiveness at the cost of lower stability in delivered quality and higher control overhead (*i.e.*, request messages). Note that the window size should be at least several times longer than average RTT between receiver and different senders since RTT determines: (i) minimum delay of the control loop from the receiver to each sender, and (ii) timescale of variations in congestion controlled bandwidth from the sender.

Since a new request is sent during the delivery of packets in the last request, a *PALS* receiver may observe duplicate packets. More specifically, after the window is slid forward, there is half an RTT worth of packets in flight from each sender. These packets will arrive during the next window and might be requested again which results in duplicates. We minimize the ratio of duplicate packets in *PALS* as follows: requested packets that are not delivered in one window, are re-requested from the same sender in the next window. Each sender removes any packet from a new request that was delivered during the last RTT. As our simulation results show, the ratio of duplicate packets is extremely low in *PALS*.

Coping with Bandwidth Variations: Periodic window sliding is not sufficient to effectively cope with sudden changes in aggregate bandwidth during one window. More specifically, sudden increase or decrease in available bandwidth from a sender could result in low utilization of its bandwidth or late arrival of packets, respectively. In *PALS*, the sliding window mechanism is coupled with several secondary mechanisms to address this issue. The receiver employs three mechanisms to cope with a major drop in bandwidth as follows:

- (i) *Overwriting Requests:* Any new request from the receiver overwrites the outstanding list of packets that is being delivered by a sender. More specifically, when a sender receives a new list from the receiver, it starts delivery of packets from the new list and abandons any pending packet from the previous list. This mechanism “loosely” synchronizes slow senders with receiver’s playout time and prevents them from falling

behind, *i.e.*, delivering late packets.

- (ii) *Packet Ordering:* As we mentioned earlier, requested packets from each sender are ordered based on their importance. Therefore, the effect of any drop in bandwidth is minimized since available bandwidth is used for delivery of more important packets.
- (iii) *Event-driven Sliding:* During each window, the receiver monitors the progress in evolution of buffer state once per RTT. In an extreme scenario when the aggregate available bandwidth is significantly lower than the estimated value and the available buffer state is not sufficient to maintain current layers until the end of this window, the receiver drops a layer, slides the window forward, invokes the QA mechanism and sends a new request to each peer.

If available bandwidth from a sender significantly increases during a period, the sender may deliver all the requested packets at a higher rate and become idle. This in turn reduces bandwidth utilization of that sender. *PALS* incorporate two ideas to address this problem: First, the receiver requests a percentage of extra packets (beyond estimated number of incoming packets based on EWMA bandwidth). The percentage of requested extra packets from a sender is determined based on the deviation of the sender’s per-window bandwidth from its EWMA average bandwidth. Similar to the packet in the buffering window, these extra packets are selected and ordered by the QA mechanism as well. Second, *PALS* also incorporate the idea of *reverse flow control* to keep all senders busy. The receiver keeps track of delivered packets by each sender to determine whether a sender is likely to complete their assigned packets (including the extra packets) before the end of the current window. If such an event is detected, the receiver sends a small request for that particular sender half an RTT before sender becomes idle to ensure utilization of the sender’s bandwidth for the remaining part of the window.

3.3 Quality Adaptation

The QA mechanism has two degrees of control that adapts delivered quality in two different timescales:

- *Coarse-grained Adaptation:* Over long timescale, the QA mechanism can add or drop the top layer to adjust the number of playing layers in response to long-term mis-match between aggregate bandwidth and stream bandwidth.
- *Fine-grained Adaptation:* Over short timescale (once per window), the QA mechanism controls evolution of receiver buffer state by adjusting the allocation of aggregate bandwidth among active layers which is used to absorb *short-term* mis-match between stream bandwidth and aggregate bandwidth.

The basic adaptation mechanism works as follows: when aggregate bandwidth is higher than stream bandwidth ($n * C \leq T_{ewma}$), called *filling window*, the QA mechanism can utilize the excess bandwidth to request future packets and fill receiver’s buffers with a proper inter-layer distribution ($buf_0, buf_1, \dots, buf_n$). Once receiver’s buffers are filled to a certain level (BUF_{add}) with the required inter-layer buffer distribution, the QA mechanism can increase stream bandwidth by adding a new layer to avoid buffer overflow. In contrast, when aggregate bandwidth is lower than stream bandwidth ($T_{ewma} < n * C$), called *draining window*, the

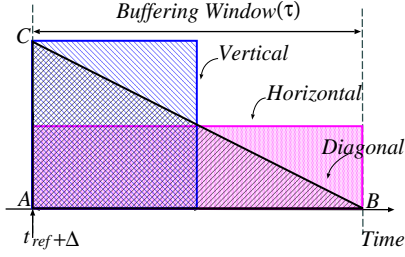


Figure 3: Horizontal, Vertical and Diagonal buffer distribution

QA mechanism drains receiver’s buffers to compensate the bandwidth deficit while maintaining proper inter-layer distribution for the remaining buffered data. If the amount of buffered data or its distribution among active layers is inadequate to efficiently absorb bandwidth deficit during a draining window, the QA mechanism drops the top layer.

BUF_{add} and inter-layer buffer distribution are key factors in coarse-grained and fine-grained adaptations, respectively. The larger the value of BUF_{add} becomes, the longer it takes to add a new layer when excess bandwidth is available, the less likely it is to drop the newly added layer in a near future. Therefore, increasing BUF_{add} further decouples the delivered quality from the variations of aggregate bandwidth, and improves stability of delivered quality.

Since stream has layered structure, a key question is “how the receiver buffer state should be evolved as it is filled or drained?”. This is determined by two factors: (i) Target buffer distribution, and (ii) Packet ordering. In essence, target buffer distribution determines overall distribution of BUF_{add} amount of buffered data across all active layers whereas packet ordering controls evolution of the buffer state as individual packets arrive. A given BUF_{add} value can be distributed across active layers in different ways as shown in Figure 3. In general, more buffering should be allocated to lower layers because of their importance for decoding higher layers. The conservative or horizontal approach of allocating buffering only to lower layers improves buffering efficiency since the buffered data is more likely to be available, and long-term stability in delivered quality. In contrast, the aggressive or vertical approach of allocating buffering only to lower timestamps of active layers achieves short-term improvement in quality but it is less efficient² and does not achieve long term stability. A skewed or diagonal buffer distribution can effectively leverage the tradeoff between long-term stability in quality and buffering efficiency. By changing the slope of such distribution one could achieve proper weight for stability or efficiency.

Packet ordering determines how the target buffer distribution is filled and drained over short timescales. There are two criteria for ordering packets, namely layer ID and timestamp. As shown in Figure 4(a), using layer ID as the primary criteria results in the horizontal ordering whereas ordering primarily based on packet timestamp leads to the vertical pattern. The snake-shape (or diagonal) pattern orders packets based on both timestamp and layer ID where packet a has a higher priority than b if the following condition is met $slope > \frac{|L_a - L_b|}{|t_{s_a} - t_{s_b}|}$ (similar to the criteria in subsection 3.2) where $slope$ determines the slope of diagonal pattern and thus the

²In general, the higher the amount of buffering for the dropped layer, the lower the buffering efficiency. In other words, allocating that buffer to lower layers would have been more useful for QA, and thus more efficient[11].

required weight for layer ID vs timestamp. The effect of packet ordering on the actual buffer state is primarily visible when delivered packets do not fill the entire target distribution. To illustrate this effect, Figure 4 depicts the buffer state for diagonal buffer distribution with different packet ordering scheme.

Quality Adaptation in PALS: PALS incorporate the following ideas to leverage the above fundamental tradeoffs in a balanced fashion. The value of BUF_{add} and its distribution is adaptively determined as a function of Look ahead (τ) and number of active layers (n) as shown in Figure 3. Toward this end, BUF_{add} is equal to half of the total available data in the future window or $BUF_{add} = 0.5(\tau * n * C)$, i.e., area of triangle ABC. Furthermore, BUF_{add} is being distributed in a diagonal fashion across all active layers with the dynamic slope of $\frac{n}{\tau}$. the corresponding snake-shape ordering for requested packets. In this approach, as the quality of stream (n) increases, the PALS behavior becomes more conservative in adding, and more resilient in dropping a layer. Similarly, using larger value of $Delay_f$, which results in larger LookAhead value, increases the value of BUF_{add} and results in more conservative target buffer distribution which is desirable since more future data is available as shown in Figure 5.

Diagonal buffer distribution in PALS implies that packets of lower layers are requested well ahead of their playout times. This strategy provides multiple opportunities for requesting packets of lower layers. This in turn leads to a higher degree of resiliency against packet loss for lower layers that are more important. Furthermore, PALS deploys an implicit loss recovery mechanism by requesting any missing packets within the playing window for the last time as we discussed earlier. In summary, multiple requesting opportunities coupled with implicit loss recovery in PALS provide sufficient number of opportunities for delivery of lost packet and / or overwritten request.

PALS requires a startup phase when the receiver buffers sufficient amount of data before it initiates stream playout. During the startup phase, only packets of the base layer (L_0) are requested from senders. In PALS, the playout is initiated when two conditions are satisfied: (i) the receiver has two windows’ worth of buffered data for the base layer, and (ii) EWMA bandwidth from all sender is more than the consumption rate of a single layer (C).

3.4 Packet Assignment

Once the QA mechanism identified and ordered all the required packets for a window, it passes an ordered list of selected packets to the packet assignment mechanism. Then, the packet assignment mechanism maps the ordered list of packets among active senders such that two conditions are met: (i) the number of allocated packets to each peer is proportional to its contribution in the aggregate bandwidth, and (ii) delivered packets by different senders

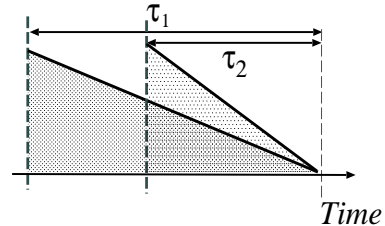


Figure 5: Effect of LookAhead (τ) on the amount of future buffering

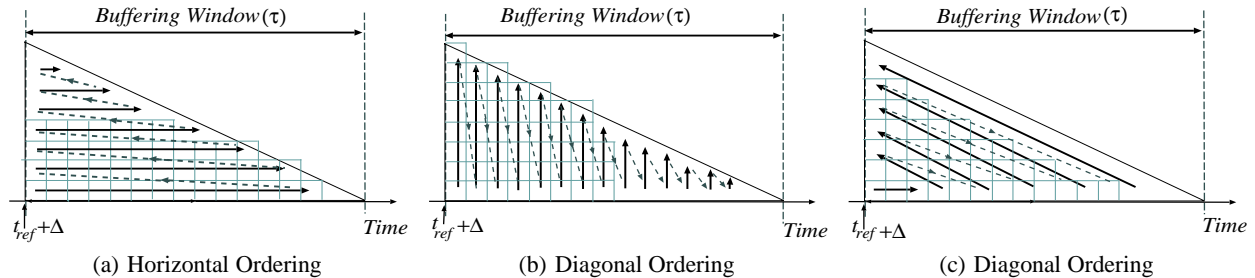


Figure 4: Effect of Packet Ordering on the evolution of buffer state

arrive in the order that is rather similar to the ordering of the aggregate list despite independent variations in bandwidth from different senders. Maintaining the order of delivery for requested packets ensures that the evolution of the buffer state remains close to the determined plan by the QA mechanism. To achieve these goals, *PALS* adopts a *weighted round-robin* strategy for packet assignment among senders. Consider an example where three senders s_0 , s_1 and s_2 contribute 50%, 30% and 20% of the aggregate bandwidth, and all senders can provide all layers. The ordered list of all selected packets for a window is assigned to senders as follows: First the PA mechanism divides the list of K packets into N_c equal-size chunks where each chunk (except possibly the last one) contains $k_c = \frac{K}{N_c}$ packets. Starting from the first chunk, the PA mechanism assigns $k_c * 0.5$ packets to s_0 , then $k_c * 0.3$ packets to s_1 , and then $k_c * 0.2$ packets to s_2 . N_c is a configuration parameter that determines how much the order of delivered packet could diverge from the global ordering specified by the QA mechanism. This strategy attempts to proportionally distribute less important packets at the end of requests from different senders.

Partially Available content: In practice, a sender may not receive or (may not cache) all the layers, or all the segments of the layers it receives. The information about available packets at each sender can be provided to the receiver as the session progresses (for live sessions) or at the beginning of playback sessions. Our basic packet assignment mechanism can not accommodate partially available content among senders.

We devised the following two-pass packet assignment mechanism to accommodate this practical scenario. Given the EWMA bandwidth from each sender, we can determine their packet budgets as follows: $k_i = \frac{T_{ewma}^i}{PktSize}$. In the first pass, the PA mechanism sequentially examines each packet from the ordered list and keeps track of the number of assigned packets to each sender ($assigned_i$). If the packet is only available at one sender, it is assigned to that sender. Otherwise, it is assigned to a sender that has the minimum ratio of assigned to total packet budget (i.e., $\frac{assigned_i}{k_i}$) because it can provide an earlier delivery time and better maintain the original ordering. For example, if both senders s_1 and s_2 can provide packet x , have the total packet budget of 300 and 100, and their number of already assigned packets is 50 and 10, packet x is assigned to s_1 . At the end of the first pass, all packets are assigned to a sender, but the number of assigned packets to some senders might be larger than their packet budget ($assigned_i > k_i$) which implies that number of assigned packets is less than the budget for some other senders.

In the second pass, we only examine those senders whose the number of assigned packets is more than their packet budget, starting from the sender with maximum surplus. For each sender, we examine those assigned packets that are available at other peers (i.e.,

exclude packets that are only available at this sender) in the given order. If each packet can be provided by another sender with packet deficit, it is assigned to the other sender. The second pass continues until no sender has any packet surplus or no more improvement can be achieved. Note that if distribution of packet among senders is proportional to the distribution of bandwidth (T_{ewma}^i) among senders, then this algorithm will identify the proper packet-to-sender mapping that fully utilizes available bandwidth from all senders. We examine the performance of this mechanism in Section 4.

4. PERFORMANCE EVALUATION

We use packet level simulator, namely *ns2*, to extensively evaluate performance of *PALS* mechanism under different dynamics in P2P systems and explore several key tradeoffs in design of receiver-driven coordination mechanisms. In our simulations, all sender peers employ RAP [7] mechanism to perform TCP-friendly congestion control and are able to provide a limited window of future packets ($Delay_r$ seconds) for all layers of a requested stream to the receiver. To properly measure congestion controlled bandwidth from a sender independent of *PALS* behavior, we have decoupled *PALS* from the underlying congestion control mechanism. Once a sender receives the first request, it starts sending requested packets at a rate that is determined by RAP. At each packet departure time, if there is an outstanding *PALS* packet, it will be mapped to the outgoing packet. Otherwise, an empty packet is sent. This decoupling allows us to assess ability of *PALS* mechanism to utilize available bandwidth from senders³. After a startup phase, the receiver emulates “streaming” playout of delivered packets while considering decoding dependency among packets. For example, a lost or late packet for the second layer implies that corresponding packets of higher layers can not be decoded and are useless.

Figure 6 depicts the basic topology in our simulations with default parameters. Each flow goes through a shared and unshared link with cross traffic. By changing the volume of cross traffic on these links, we can control which one ultimately becomes a bottleneck in each simulation and generate a desired scenario with shared or unshared bottleneck. We have also used a single TCP flow on the reverse direction to avoid any phase effect in simulations with larger number of flows. The amount of buffer size for each link is properly provisioned. Presented results are averaged over 50 runs with different random seeds. Unless otherwise stated, the following default parameters are used in our simulations: $N_c = 5$, $C = 80$ KBps, $N = 10$, $M = 3$, $\delta = 500$ msec, $PktSize = 1$ KByte.

³It is worth noting that despite this decoupling any change in *PALS* parameters could result in a different pattern of requests from the receiver which in turn affects dynamics of ACK packets and short term variations of congestion controlled bandwidth.

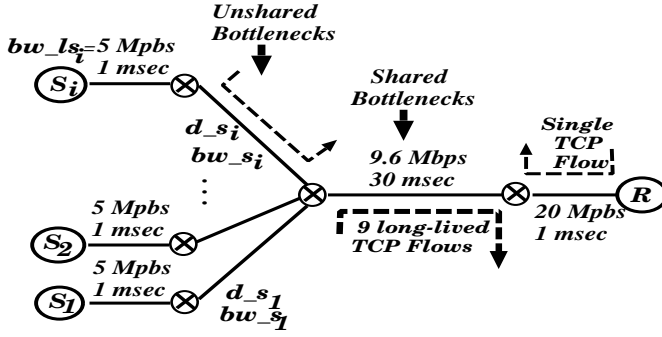


Figure 6: Simulation Topology

We have extensively evaluated *PALS* and examined the effect of wide range of parameters under a variety of scenarios including: shared and unshared bottleneck among senders, and different degree of bandwidth heterogeneity. Due to the limited space, we only present a representative subset of our results in this sections. In particular, we focus on senders with shared bottleneck since bandwidth sharing among senders introduces further dynamics to available bandwidth. Furthermore, we emphasize on scenarios with a moderate degree of heterogeneity among senders for the following reason. When the degree of heterogeneity is too high, the behavior of one sender become dominant and the dynamics of multi-sender delivery is not shown. In contrast, assuming senders with homogeneous bandwidth is rather unrealistic. Further results can be found in the related technical report [12]. We explore the following issues in this section: (i) the importance of inter-sender coordination, (ii) key design tradeoff in receiver-driven coordination, (iii) ability of *PALS* to cope with dynamics of bandwidth variations and peer participation, (iv) sensitivity of *PALS* to different pattern of partially available content among senders.

4.1 Importance of Coordination

As we discussed earlier, to maximize delivered quality from multiple congestion controlled senders despite variations in bandwidth, it is important to dynamically coordinate delivered packets by each sender. To illustrate the importance of inter-sender coordination, we compare the performance of *PALS* with the following two mechanisms that employ *static* content-to-sender mapping.

- *Single Layer per Sender (SLS)*: In this approach each sender s_i delivers packets of a designated layer (L_i) at the rate that is determined by its congestion control mechanism. *SLS* represents the common “one layer per sender” approach that has been proposed in several previous studies (e.g., [13, 5]).
- *Multiple Layer per Sender (MLS)*: This approach is more elaborate and assigns multiple layers to each sender [14]. Given an ordered list of senders based on their available bandwidth, *MLS* starts from the first sender (with maximum bandwidth) and sequentially assigns the maximum number of consecutive new layers (i.e., $l_i = \lfloor \frac{T_{ewma}^i}{C} \rfloor$) to each sender. For example, if layer bandwidth is $C=80\text{KBps}$, the layer to sending mapping for three senders with 250, 175 and 100 KBps average bandwidth would be (L_0, L_1, L_2) , (L_3, L_4) and (L_5) , respectively.

Both *SLS* and *MLS* incorporate the following miscellaneous mechanisms. Each sender delivers packets of the assigned layers through

a RAP connection (i.e., RAP) based on their timestamp, and across different layers based on their layer number, i.e., vertical ordering. Receiver reports its playout time in ACK packets to enables senders to estimate receiver’s playout time. To ensure in-time delivery, each sender only transmits those packets whose timestamp is at least one RTT larger than the receiver’s playout time. This implies that a sender skips a range of timestamps when its available bandwidth drops in order to remain loosely synchronized with the receiver’s playout time. A sender can also utilize its excess bandwidth to send available future packets of assigned layers (up to *Delay_r* seconds) and increase the buffered data at the receiver. Both *SLS* and *MLS* incorporate an explicit loss recovery mechanism. Drop packets are detected by RAP at the sender side and retransmitted based on their priority within the packet ordering scheme if there is sufficient time for their in-time delivery. In summary, *SLS* and *MLS* represent two well designed multi-sender streaming mechanisms that leverage interactions between the receiver and each sender to accommodate timing and loss recovery. However, they do not use any coordination among senders and rely on static layer-to-sender mapping.

We compare *PALS* with *SLS* and *MLS* in a scenario where a variable number of heterogeneous senders reside behind a shared bottleneck. The results for homogeneous senders are similar and can be found in the related technical report [12]. For a scenario with i senders, only senders s_1 to s_i participate in delivery. We use the topology shown in Figure 6. However, we have changed the following parameters from their default values: d_{Ls_i} values are set based on the following equation $d_{Ls_i} = d_{Ls_1} + (i - 1) \cdot 0.5\text{ms}$ where $d_{Ls_1} = 1\text{msec}$ to achieve heterogeneous bandwidth. The shared bottleneck has $BW_{bn} = 32\text{Mbps}$ bandwidth and 20 long-lived cross traffic. Finally, receiver access link was reduced to $BW_r = 4.5\text{Mbps}$. With these parameters, as the number of sender increases, initially the shared bottleneck is the limiting link and then receiver’s access link becomes the bottleneck. Note that the aggregate bandwidth to the receiver is independent of the content delivery mechanism, and only depends on the number of senders. Table 2 shows the available bandwidth from each sender in these scenarios in terms of layer bandwidth ($\frac{T_{ewma}^j}{C}$).

n	T_{ewma}^1	T_{ewma}^2	T_{ewma}^3	T_{ewma}^4	T_{ewma}^5	T_{ewma}^6
2	2.5	2.4				
3	1.9	2.1	1.9			
4	1.7	1.6	1.5	1.4		
5	1.4	1.3	1.27	1.17	1.14	
6	1.2	1.2	1.1	1.02	1.01	1.01

Table 2: Available bandwidth for each sender across different scenarios

Figure 7(a) depicts the average delivered quality by *SLS*, *MLS* and *PALS* for different number of senders, ranging from 2 to 6. We have also shown the maximum deliverable quality (i.e., the ratio of aggregate bandwidth to layer bandwidth) in each scenario as an upper bound for average delivered quality. This figure shows that the average delivered quality by *PALS* is higher than the other two mechanisms and is indeed very close to the maximum deliverable quality. The small gap between the delivered quality by *PALS* and the maximum deliverable quality represents the residual aggregate bandwidth that is insufficient for adding another layer. Lower delivered quality by *SLS* and *MLS* is primarily due to the inability of these mechanisms to utilize residual bandwidth from each sender. For example in a scenario with 4 senders, the residual bandwidth

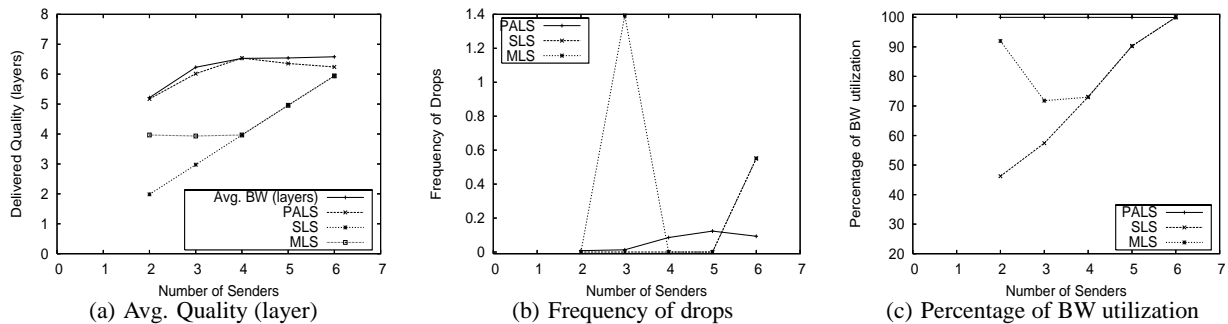


Figure 7: Effect of Inter-sender Coordination, PALS parameters: $\Delta = 6\text{sec}$, $Delay_r = 40\text{sec}$

from all senders (shown in Table 2) is sufficient to deliver two more layers. However, without any coordination among senders, these residual bandwidth can not be utilized.

Figure 7(b) shows the frequency of layer drops to quantify the stability of delivered quality by these mechanism in the same simulations presented in Figure 7(a). These variations occur because of layer drops (only in PALS) or undelivered packets. Variations of delivered quality by SLS and MLS is zero in those scenarios where all senders have a plenty of residual bandwidth. In these scenarios, each sender can deliver packets of designated layers ahead of time which results in a plenty of receiver buffering. Table 2 shows that all senders have a plenty of residual bandwidth in scenarios with 2, 4 and 5 senders. However, when residual bandwidth for at least one sender is low (e.g., s_2 when $n = 3$, or s_3-s_6 when $n=6$), MLS and SLS are very sensitive to variations of available bandwidth from these sender(s) because senders with low residual bandwidth can not accumulate sufficient buffering, and thus are forced to skip portions of designated packets to ensure in-time delivery.

Figure 7(c) depicts the utilization of aggregate bandwidth from all senders in same simulations. PALS is the only mechanism that fully utilizes aggregate available bandwidth from all senders. In SLS and MLS, the aggregate bandwidth is not fully utilized due to the limited availability of future packets at each sender. In other words, each sender can only send a limited amount of future packets (depending on the value of $Delay_r$) and then becomes idle. The larger the value of residual bandwidth, the more future packets (i.e., larger $Delay_r$ values) are required to fully utilize available bandwidth.

As expected, none of the mechanisms experienced late packets in our simulations. Because of their static mapping, SLS and MLS do not deliver duplicate packets. However, we observed less than 0.05% duplicate packets in all PALS simulations. *In summary, the behavior of SLS and MLS significantly depends on the amount of residual bandwidth at individual senders. When residual bandwidth at some senders is low, they can efficiently utilize their available bandwidth but they become too sensitive to bandwidth variations which results in instability of delivered quality. In contrast, when all senders have a plenty of residual bandwidth, they can buffer future packets and provide a stable quality. However, they exhibit poor bandwidth utilization specially when the amount of future packets among senders are limited (i.e., $Delay_r$ is not too large). In a nutshell, the content delivery mechanisms that rely on static layer-to-sender mapping can not maximize delivered quality because of their inability to efficiently utilize residual bandwidth*

from each sender.

4.2 Effect of PALS Parameters

We turn our attention to the effect of key configuration parameters, including window size and LookAhead, on the performance of PALS mechanisms. This also illustrates underlying dynamics and key tradeoffs of the receiver-driven coordination mechanism. Window size (Δ) determines the frequency of adaptation in PALS mechanism. LookAhead (τ) controls the amount of future buffering that determines the amount of BUF_{add} and its inter-layer distribution, and thus affects the tradeoff between responsiveness and stability of delivered quality. To explore these issues, we consider a scenario when three heterogeneous senders are behind a shared bottleneck using the topology shown in Figure 6 with default parameters. Table 3 summarizes averaged bandwidth and RTT that are obtained by each sender in this simulation. PALS uses diagonal shape for buffer distribution along with diagonal pattern for packet ordering in these simulations.

	sender1	sender2	sender3
AvgRTT	79 msec	110 msec	144 msec
AvgBW	1.59 Mbps	1.16 Mbps	0.82 Mbps
Dev.BW	0.47 Mbps	0.35 Mbps	0.24 Mbps

Table 3: Bandwidth and RTT for individual senders in subsection 4.2

Window and LookAhead: Figure 8 depicts different angles of PALS performance as a function of window size for different values of LookAhead parameter. Figure 8(a) shows the average delivered quality and illustrates that neither depends on Δ nor τ . In other words the average quality is only determined by the average bandwidth. This is mainly due to the event-driven sliding of the window coupled with reverse flow makes the behavior of PALS that reduces the effect of window size on average quality. Figure 8(b) shows the degree of stability by depicting the frequency of layer drop events that are triggered by the QA mechanism. This figure reveals that frequency of changes is generally very low (often less than 1.5 drops every 100 seconds, or 0.015) and is reduced by increasing the LookAhead parameter. Furthermore, it does not depend on the window size when LookAhead is not too low. This figure illustrates the key tradeoff between responsiveness to variations in bandwidth and stability in delivered quality. Using large LookAhead parameter results in large BUF_{add} which accommodates stability at the cost of responsiveness. In contrast, when LookAhead is small, the

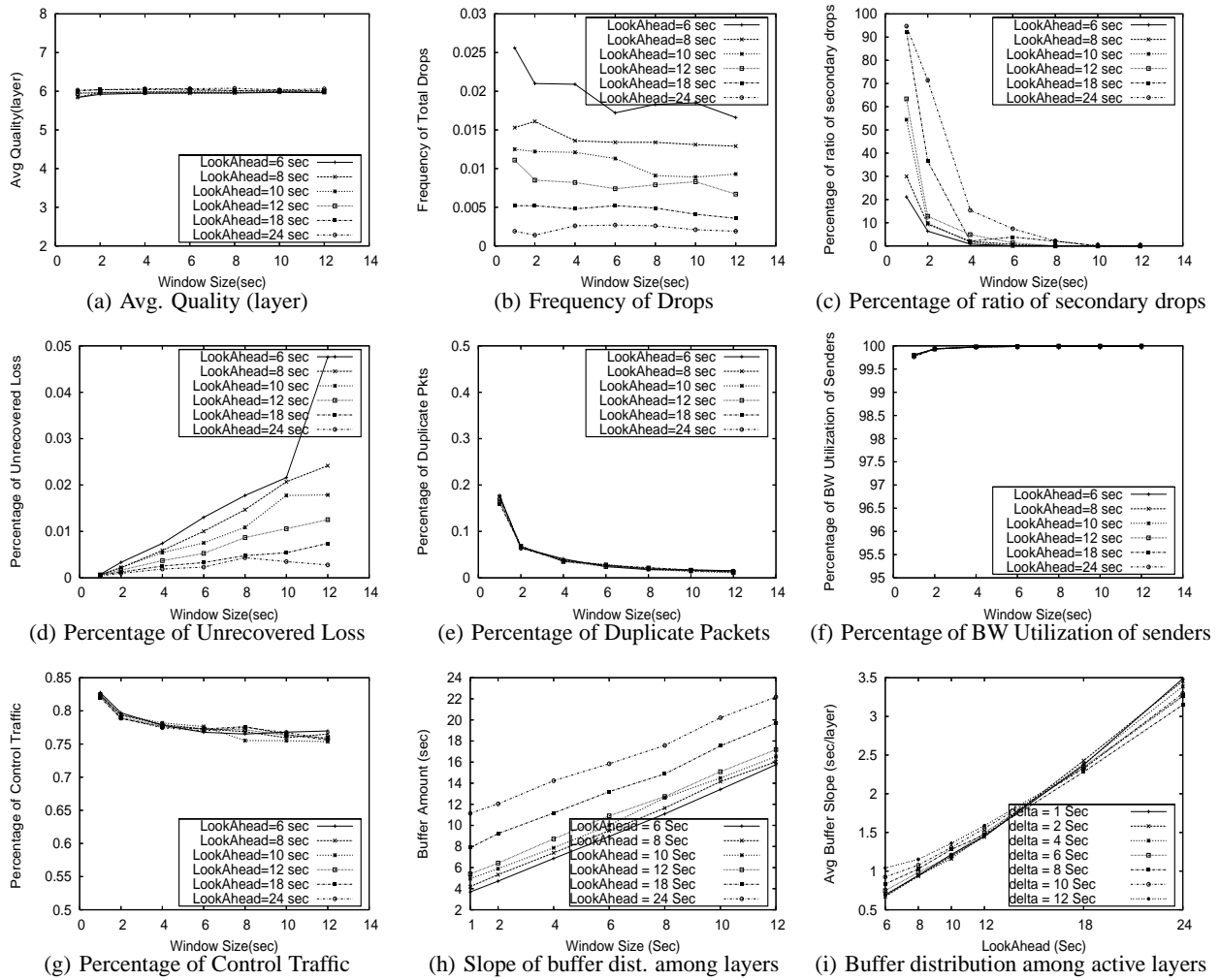


Figure 8: Effect of Window size and LookAhead on different aspect of PALS performance

layer add condition can be easily satisfied which results to an immature adding of a new layer that can not be sustained and leads to instability in quality. We further explore the underlying causes of the observed layer drops by dividing them into two groups: (i) *Primary drops* that occur due to insufficient aggregate buffering to absorb a drop in bandwidth at the receiver, and (ii) *Secondary drops* that occur when total buffered data is sufficient but its useful portion within the active window is inadequate to absorb bandwidth deficit, *i.e.*, buffered data is distributed across multiple window. The distinction between these drop events allows us to identify the effect of buffer distribution on variations of delivered quality. Figure 9(b) depicts the ratio of secondary drops and clearly shows that this ratio increases with the LookAhead parameter but this increase is significantly smaller as the window size grows. Since the useful portion of buffered data depends on the shape of buffer distribution and the portion of buffered data that falls within the next window (*i.e.*, $|\tau - \Delta|$). Therefore, increasing the window size or decreasing the LookAhead would result in decrease in the useful portion of buffer which leads to smaller ratio of secondary drops.

Figure 8(d) shows the percentage of unrecovered packet losses. These are the packets that were lost (once or multiple times) and did not get another transmission opportunity because of their relative priorities. Note that losses are weighted by their impact on delivered quality. In other words, when a packet is lost, all its decoding-dependent packets from higher layer are also considered lost although they are actually delivered. This figure illustrates that the percentage of unrecovered losses is generally small (less than 0.03%), but it increases as the window size grows or LookAhead shrinks. This behavior is related to the average number of transmission opportunities for each packet which is determined by the ratio of LookAhead to window size. *In summary, Figure 9(b), 9(e) and 8(d) show two important tradeoff for selecting window size and LookAhead as follows: Increasing Δ reduces the ratio of secondary drops but results in higher ratio of unrecovered losses. Furthermore, increasing τ improves the stability of delivered quality at the cost of higher ratio of secondary drops, and larger delay with respect to parent peers. In a nutshell, there is a sweet spot for window size and LookAhead parameters that should be set according to application's requirements.*

Efficiency and Overhead: Figure 8(e), 8(f) and 8(g) represent various dimensions of efficiency and overhead for PALS mechanism. Figure 8(e) indicates that the percentage of duplicate packets is in general very small (less than 0.3%) and can be rapidly dropped by increasing window size. This supports our explanation that duplicate packets could occur only when window is sliding since increasing window size rapidly reduces the ratio of duplicates. Figures 8(f) shows that the utilization of aggregate bandwidth is always very close to 100% except for very small window. We also examined the average percentage of overwritten packets (not shown here) across all senders and found out that both the average and per-sender percentage of overwritten packets remain around 25% independent of window size or LookAhead parameter. Since the deviation of average bandwidth from each sender is around 30% in these simulation, only extra packets from each sender are overwritten. While this appears to suggest that PALS over-estimates the ratio of requested extra packets from each sender in this scenario, any reduction in the ratio of extra packet could result in a lower bandwidth utilization. The overhead of generated control traffic (*i.e.*, request of the receiver) in PALS reduces as window size increases but remains below 0.85% in all scenarios as shown in Figure 8(g). In Summary, these results show that PALS can ef-

fectively utilize available bandwidth from senders, and provide a stable quality with a low ratio of control overhead and very small ratio of duplicate packets.

Buffer State: Finally, Figure 8(h) and 8(i) depict the average amount of buffered data across active layers, and the slope of distribution for actual buffered data (using τ as x-axis), respectively. These figures collectively illustrate how closely the actual buffer state follows the target buffer distribution despite the underlying dynamics in bandwidth variations, packet selection and packet assignment. Figure 8(h) shows the average amount of buffered data across all layers linearly increases with both Δ and τ . The direct effect of τ on the actual amount of future buffering (the gap between consecutive lines for the same value of Δ) supports our earlier explanation on why increasing τ reduces the frequency of layer drop (in Figure 8(b)). We note that when window size is larger than LookAhead, the active window is less than half-full right after the window is slid. Therefore, the average per layer buffering at the receiver should be around $\Delta + 0.5 * \tau$. However, when LookAhead is larger than window size, the buffer data is gradually accumulated over several windows at the rate proportional to excess bandwidth. Therefore, the per layer buffering would be less than $\Delta + 0.5 * \tau$. Figure 8(h) clearly demonstrates these two scenarios. Figure 8(i) demonstrates that increasing LookAhead can effectively increase the slope of actual buffer state. However, increasing Δ results in a larger total buffering and slightly reduces the effect of LookAhead on the slope of buffer state. *In Summary, our results indicate that the LookAhead parameter can effectively control actual distribution of buffered data across layers despite various dynamics in the system.*

Buffer Distribution & Packet Ordering: Now, we take a closer look at the impact of packet ordering in the buffering window on PALS performance. Packet ordering primarily affects fine-grain dynamics of buffer evolution that determines receiver's buffer state and its efficiency. Figure 9 depicts both the ratio of secondary drops and buffering efficiency in PALS for three different ordering schemes, namely vertical, horizontal, diagonal, with the adaptive diagonal buffer distribution in PALS as described in subsection 3.3. This figure clearly illustrates an interesting tradeoff between ratio of secondary drops and buffering efficiency that can be leveraged by packet ordering. More specifically, vertical ordering scheme has the minimum ratio of secondary drops but lowest buffering efficiency. In contrast, horizontal ordering scheme has the maximum ratio of secondary drop but highest efficiency. The diagonal ordering strikes the balance by achieving a very good buffering efficiency and with sufficiently large window size can minimize the ratio of secondary drops. This tradeoff is a direct effect of the portion of delivered packets fall within the next window. Vertical and horizontal orderings place the maximum and minimum portion of buffered data in the next window, respectively. *In summary, this results demonstrate that our choice for diagonal packet ordering can properly leverage this tradeoff.*

4.3 Dynamics of Bandwidth Variations

As we discussed earlier, by increasing the LookAhead parameter in PALS, the amount of receiver buffering grows and its QA mechanism becomes more conservative in adding layers, and more resilient in dropping a layer. Therefore, in essence, PALS becomes less responsive as LookAhead parameter increases. To qualify this effect, we have examined PALS performance over a wide range of dynamics in available bandwidth by using different types of cross traffic including long-lived TCP, http and flash-crowd. Here, we de-

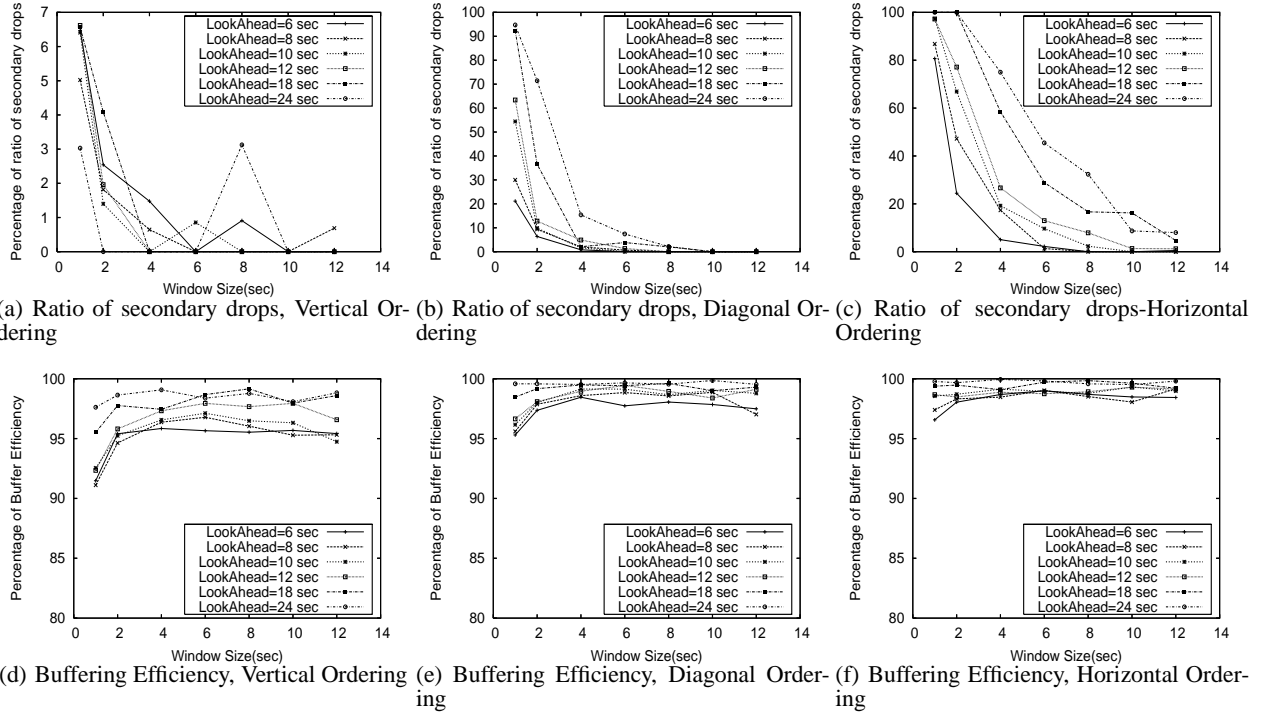


Figure 9: Effect of packet ordering on the ratio of secondary drops and buffering efficiency

scribe a representative scenario where 3 senders stream content to a receiver through a shared bottleneck with flash-crowd cross traffic. We use the topology shown in Figure 6 with the default parameter and only introduce flash crowd traffic to the shared bottleneck from $t=110$ second to $t=140$ second. Flash-crowd cross traffic is simulated with 300 short-lived TCP flows where each flow starts at a random time between $t=110$ sec and $t=140$ sec, and remains active for a random period between 0.8 to 1.2 seconds.

Figure 10 illustrates behavior of *PALS* in this scenario as a function of time for two different LookAhead parameters, $\tau = 10, 32$ seconds. Figure 10(a) and 10(d) show the aggregate bandwidth and delivered quality for τ equal to 10 and 32 seconds, respectively. As we described earlier, when τ is smaller, *PALS* is more responsive to the variations of bandwidth, in particular when sudden change in the aggregate bandwidth occurs. In contrast, using larger τ values lead to a significantly more stable behavior such that *PALS* is even manage to avoid any layer drop despite the major drop in available bandwidth. The observed negative spikes in these figures are due to unrecovered losses for the top layer. Such unrecovered losses occur when loss rate suddenly increases and thus bandwidth drops, since the allocated bandwidth to loss recovery is limited.

To illustrate the role of buffering on *PALS* behavior in these two scenarios, we have shown the evolution of buffer state (*i.e.*, among of per-layer buffering) in terms of its playout time⁴ for these simulations in Figure 10(b) and 10(e). Comparison between these two figures clearly shows that using larger LookAhead values results in more total buffering and more skewed distribution across lay-

ers (*i.e.*, larger gap between lines in Figure 10(e)) which in turn increases receiver's ability to effectively absorb a major drop in bandwidth by draining the buffer data. As shown in Figure 10(e) right after flash-crowd traffic starts, the buffered data for higher layers are drained at a faster rate to protect lower layers.

To show the effect of bandwidth variations on the dynamic of requested packet by the QA mechanism, Figure 10(c) and 10(f) depict the timestamps of requested packets in each window along with the position of the buffering window for part of the above simulations. Groups of requested packets for active layers in each window are shown as parallel column on top of the corresponding window. The relative gap between each packet and its corresponding window is proportional to its timestamp. Even though all packets are requested at the beginning of the window, they are separated in this graph for clarity (the left most column represents requested packets for the base layer). Comparison of these figures shows that the relative difference, and thus the total range, of requested timestamps from different layers increases with the LookAhead parameter. Figure 10(f) clearly shows that once the available bandwidth drops, the gap between timestamp of requested packets and the window decreases indicating that receiver's buffers are being drained. Packets with timestamp lower than the corresponding buffering window are located within the playing window, and requested by the explicit loss recovery mechanism (*e.g.*, around $t=123$ and $t=127$ sec). Figure 10(f) also demonstrates that many packets have multiple number of transmission opportunities. This number can be easily determined by drawing a horizontal line from the first request for a packet and counting number of future windows that fall below this line. Clearly, there are more opportunities for packets of lower layers to be requested because they are initially requested well ahead of their playout times. A packet might be requested multiple times

⁴We show the buffered data in terms of its playout time which is the time it takes to playout the buffered data. This presents the timing aspect of buffer that is not captured by its absolute value in byte.

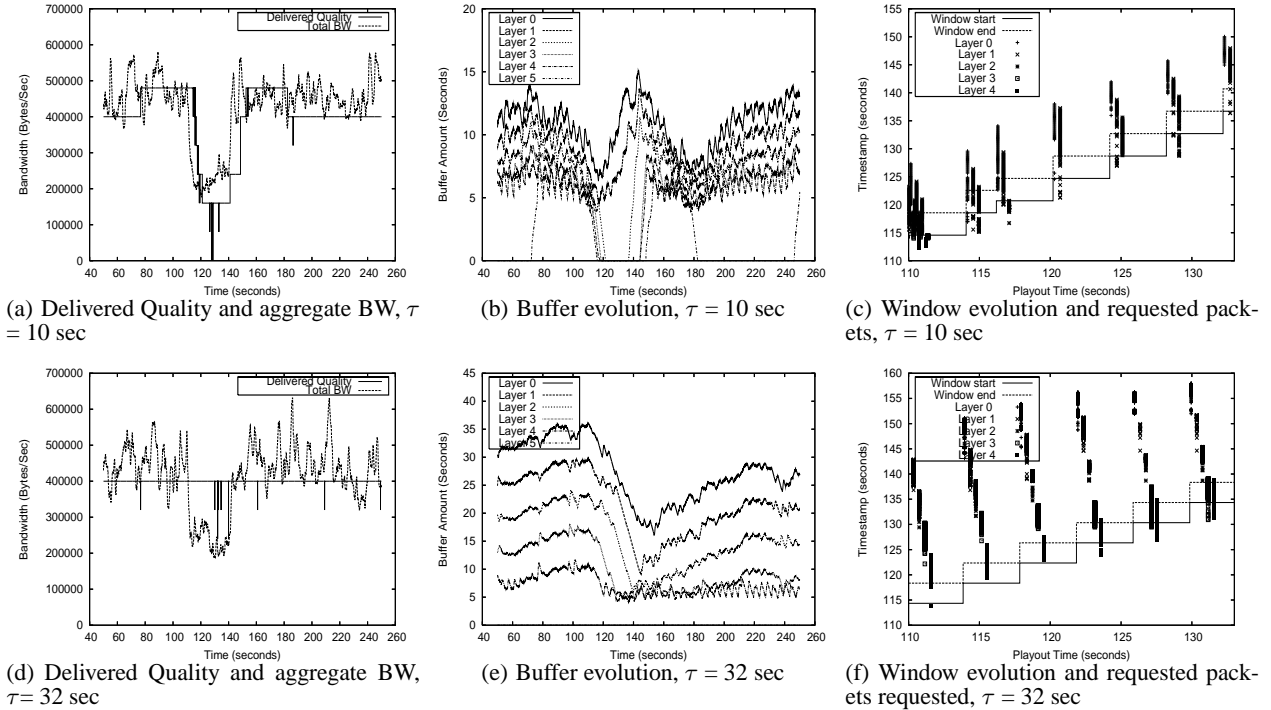


Figure 10: Behavior of PALS with flash crowd cross traffic, $\Delta = 4$ seconds

due to packet loss or overwritten request.

4.4 Dynamics of Peer Participation

Departure or arrival of an active sender can change aggregate bandwidth and thus delivered quality to a receiver. In essence, this effect is very similar to the dynamics of bandwidth variations that we examined in subsection 4.3. To examine the effect of sender dynamics on *PALS* behavior, we use our default simulation topology with 4 senders and the following $d_{s(i)}$ values 1,13,16,19msec. All other parameters are set to their default values. Initially, three senders s_1 , s_2 and s_3 are active and obtain average bandwidths 149 KBps, 138 KBps, and 133 KBps, respectively. We stop sender s_2 and time $t=100$ second, and start sender s_4 at time $t = 150$ second to simulate slow replacement of a departed sender.

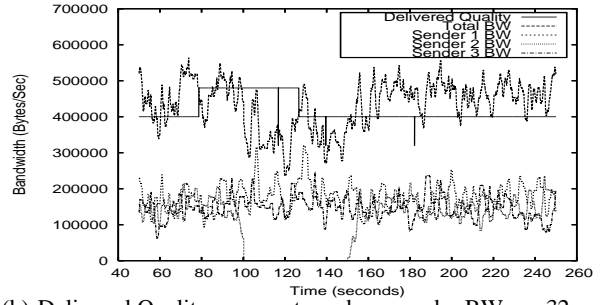
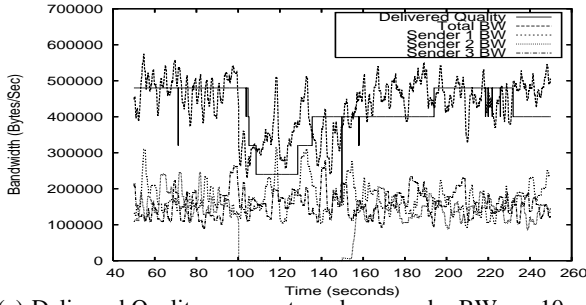
Figure 11(a) and 11(b) depict aggregate and per sender bandwidth as well as delivered quality in this scenario for two different LookAhead parameters, $\tau = 10$ and 32 seconds. Comparison between these figures reveals two interesting points: First, arrival or departure of one of three senders does not result in 33% change in aggregate bandwidth due to the shared nature of bottleneck. More specifically, when a sender departs (or arrives), the other active senders obtain (or lose) portion of their bandwidths in response to this change. Note that the variations of bandwidth for senders with unshared bottleneck are rather independent and do not exhibit this behavior. Second, a drop in the aggregate bandwidth due to the departure of a sender can be effectively absorbed by increasing the LookAhead parameter. This in turn provides more time for the receiver to replace the departed sender. For example, Figure 11(b) shows that setting τ to 32 seconds enables the receiver to sustain all layers for 50 seconds until the departed sender is replaced. *In summary, PALS can effectively cope with the dynamics of sender*

participation when the LookAhead parameter is sufficiently large. This implies that the receiver does not require to quickly detect departure of an active sender.

4.5 Partially Available Content

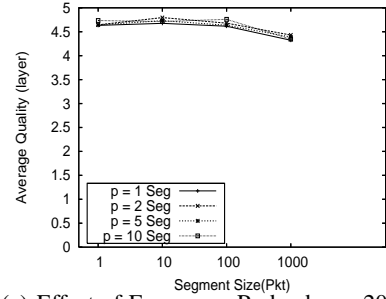
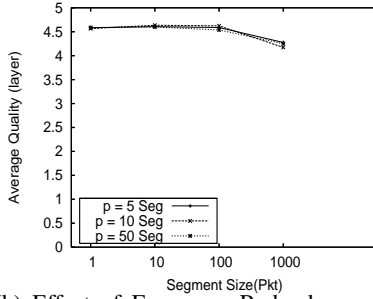
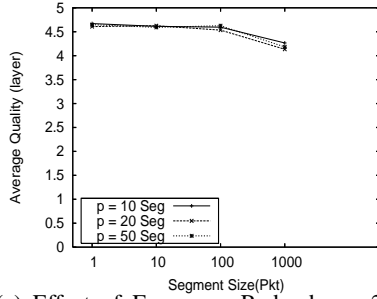
So far, we have assumed that all senders can provide all requested packets by the receiver. However, in practice, each sender peer may not receive (or may not cache) all layers of the stream or all packets of the received layers. In this subsection, we examine the ability of the two-pass packet assignment mechanism in *PALS* to stream partially available content from multiple senders. In particular, partially available content limits the flexibility for the packet assignment mechanism to map the required content among senders. Such scenario limits the ability of *PALS* mechanism to fully utilize available bandwidth from one (or more senders) which in turns degrades the quality of delivered stream. We define the *average content bandwidth* at a sender peer as the average bandwidth of its content across all timestamps. For example, if a sender has 3 layers for half of the timestamps and 2 layers for the other half, its average content bandwidth is $2.5 \cdot C$. When the average content bandwidth at a sender is lower than its available bandwidth, the sender becomes *content bottleneck* and its available bandwidth can not be fully utilized. The following minimum requirements for available content among senders must be met to ensure that senders do not become content bottleneck: (i) at least one copy of any required packet by the receiver must be available among senders, and (ii) the distribution of aggregate content among senders must be proportional to their contributions in aggregate bandwidth.

Once the minimum requirements are met, there are three related factors that could affect the performance of *PALS* mechanism as follows: (i) *Granularity of Content*: each sender may cache indi-



(a) Delivered Quality, aggregate and per-sender BW, $\tau = 10$ sec (b) Delivered Quality, aggregate and per-sender BW, $\tau = 32$ sec

Figure 11: Effect of sender dynamics on PALS behavior



(a) Effect of Frequency-Redundancy 2%, (b) Effect of Frequency-Redundancy 4%, (c) Effect of Frequency-Redundancy 20% - $\Delta: 6, Delay_r = 20$

Figure 12: Effect of period of redundancy on PALS performance

vidual segments of any received layer where a segment consists of s consecutive packets. Therefore, the segment size determines the granularity (and thus the pattern) of available content at each sender. Using a small segment size of one packet results in caching scattered packets of a stream. In contrast, if segment size is equal to stream length, each sender caches complete layers of each stream.

(ii) *Degree of Redundancy*: Availability of multiple copies of some packets among senders provides more flexibility for the packet assignment mechanism and improves its performance. For example, two senders might be able to deliver a particular packet. We define the degree of redundancy (r) as the ratio of total number of extra packets across all sender to the total number of single copies across all senders. For example, if 30% of packets have 2 extra copies, the degree of redundancy is 60%.

(iii) *Pattern of Redundancy*: The pattern of added redundancy may also determine its impact on the performance of packet assignment among senders. We assume that the redundancy is added with the granularity of one segment. To control the pattern of redundancy at each peer, we introduce the notion of *period* of added redundancy (p) which determines how often the redundant segments should be added. For example, there are different ways to add 20% redundancy to a stream with 5 layers as follows: adding one extra segment to each group of corresponding segments from different layers (i.e., $p = 1$ segment), or adding 2 segments to every other group of segments (i.e., $p = 2$ segment), and so on. It is worth noting that for a given n of active layers and the degree of redundancy r , we can only define period of redundancy p when two conditions are met: $p * r * n$ is an integer value, and $1 \leq p * r * n \leq n * (M - 1)$.

M denotes the number of senders. In a nutshell, we can uniquely determine the pattern of distribution for n layers across senders with three parameters: segment size, degree of redundancy and frequency of redundancy. Toward this end, we start from a single copy of all layers, then once every p consecutive segments, we add $n * p * r$ randomly selected but unique copies of the corresponding segments from different layers as shown in Figure 13.

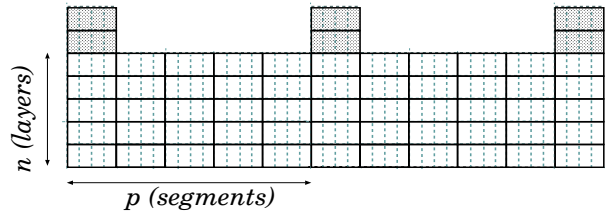


Figure 13: Effect of the period of added redundancy on its pattern

Finally, each group of segments from different layers are randomly mapped to senders where the number of segments per sender is proportional its contributions in aggregate bandwidth.

To study the sensitivity of PALS performance to different patterns of partially available content, we consider a scenario with three senders behind a shared bottleneck using the topology in Figure 6. But we change the following parameters: $bw_{Ls}(i) = bw_{Ss}(i) = [10, 15.3, 20.6]$ Mbps, $d_{Ls}(i) = [4, 25, 36]$ msec, $bw_r = 4$ Mbps,

Available bandwidth and RTT for three senders are summarized in table 4

	sender1	sender2	sender3
<i>AvgRTT</i>	98 msec	140 msec	160 msec
<i>AvgBW</i>	1.5 Mbps	0.92 Mbps	0.85 Mbps
<i>DevBW</i>	0.2 Mbps	0.11 Mbps	0.1 Mbps

Table 4: Avg. BW and RTT of each sender in subsection 4.5

We have examined performance of *PALS* over a wide range of pattern for partially available content. Due to the limited space, we only present a subset of results that represents the main findings. Further results can be found in the related technical report. Number of layers (n) is 5 in all of our simulations. To illustrate the effect of redundancy period (p) on the performance of *PALS* mechanism, Figure 12(a) depicts average delivered quality as a function of segment size for different period of redundancy where $r = 2\%$, $Delay_r = 20$ second, and $\Delta = 6$ seconds. Note that the x axis has a logarithmic scale. This figure shows that the delivered quality is not very sensitive to the value of p . Figures 12(b) and 12(c) show the same simulation with a higher degree of redundancy, namely $r = 4\%$ and $r = 20\%$. The selected values of p across these figures are different because of the dependency of p to the r value. These figures collectively demonstrate that the period of redundancy does not have a significant effect on *PALS*. The reason for this behavior is that the performance is sufficiently high when segment size is small and there is not much improvement to achieve by adding redundancy. There is some room for performance improvement when segment size is large. However, in these scenarios, redundant packets, regardless of the degree of redundancy, are significantly apart (by $s * p$ packets) and thus can not provide flexibility to packet assignment across many windows.

Figure 14(a) presents the sensitivity of *PALS* performance to different window sizes where $p = 10$ segments, $Delay_r = 40$ sec, $r = 2\%$. This figure shows two interesting points: First, as the segment size increases, the performance is degraded. This effect is significantly more visible for smaller window sizes. To explain this effect, we note that as the segment size increases, the distribution of average content bandwidth among senders not only becomes coarser but also further diverges from the distribution of available bandwidth among senders. Furthermore, the pattern of content distribution has less flexibility within each window since each sender can provide packets of the specific same layers. These two effects collectively limit the ability of packet assignment mechanism to fully utilizes the available bandwidth and result in the content bottleneck in some senders. Second, a small increase in the window size can significantly improve the performance, specially in scenarios with large segment sizes. The main reason for this improvement is the increase in the number of packets that should be assigned in each window, which in turn provides more flexibility for the packet assignment mechanism.

To quantify the effect of redundancy on performance, we repeat the simulations in Figure 14(a) with the exact same parameters but increase the degree of redundancy, from 2 to 20%, as shown in Figure 14(b). Comparison between Figure 14(b) and 14(a) indicates that the significant increase in redundancy only moderately improves the performance, and its impact is lower than window size.

Finally, Figure 14(c) shows the effect of $Delay_r$ on *PALS* performance when $\Delta = 1$ second and $r = 1\%$. This figure illustrates that increasing $Delay_r$ degrades *PALS* performance specially for large segment size. This is the direct effect of $Delay_r$ on the distribution of requested packets across different layers. More specifically when $Delay_r$ increases, the distribution of requested packets across active layers becomes more skewed (*i.e.*, more packets are requested from lower layers). Therefore, a coarser mapping of content mapping among sender due to larger segment size is unable to effectively utilize available bandwidth from senders. *In summary, our results illustrate that segment size is the primary factor that limits the flexibility of packet assignment. However, a minor increase in window size can compensate the effect of segment size.*

5. RELATED WORK

There have been a few previous studies on various aspects of multi-sender streaming to a single client. Apostolopoulos et al. [15] proposed a mechanism for streaming multiple description (MD) encoded content from multiple servers. They presented a distortion model for MD encoding and used this model to study the effect of server and content placement on delivered quality of both MD and single description (SD) encodings. Multi-sender streaming in P2P networks was casted as resource management problem by Cui et al [16]. Using global knowledge about participating peers, they formulated the problem as an optimization problem to maximize delivered quality to each peer while minimizing server and network load. Neither of these previous studies have considered congestion controlled connections among peers and thus they did not address dynamics of bandwidth variations over the Internet. Nguyen et al. [17] presented a multi-sender streaming mechanism from congestion controlled senders. In their proposed solution, the receiver periodically reports throughput and delay of all senders back to them using control packets. Then, senders run a distributed algorithm to determine which sender is responsible for sending each segment of the stream. They assumed that senders do not reside behind a shared bottleneck, and their aggregate bandwidth is sufficient for delivery of full quality stream. Compare to their approach, *PALS* is receiver-driven, and does not make any assumption about aggregate bandwidth or location of participating peers.

There has been a wealth of research on both distributed [5, 18, 19] and central [3] approaches to construct overlay for P2P streaming. *PALS* mechanism is complementary to these approaches and can be integrated with them. Both *PALS* and RLM [20] are receiver-driven mechanisms and perform layered quality adaptation. However, there is a fundamental difference between them. In RLM, congestion control and quality adaptation mechanisms are tightly coupled. More specifically, the receiver performs coarse-grained congestion control by adjusting the number of delivered layers to regulate overall bandwidth of incoming stream which implicitly leads to adaptation of delivered quality. Given a set of sender peers, the *PALS* receiver does not have any control over senders' transmission rates, but it only coordinates how the available bandwidth from each sender is used. *To our knowledge, PALS is the first mechanism for quality adaptive streaming from multiple, congestion controlled senders with arbitrary distribution across the Internet.*

6. CONCLUSIONS AND FUTURE WORK

In this paper, we presented *PALS*, a receiver-driven coordination mechanism for quality adaptive streaming of layered encoded video from multiple congestion controlled senders. We described the mechanism, its key components and their design space, as well as various design tradeoffs. We have conducted simulation-based

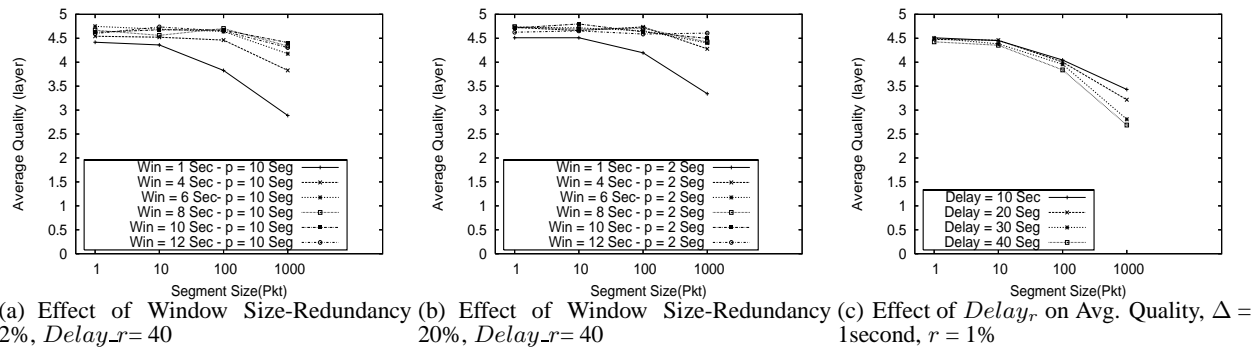


Figure 14: Effect of degree of redundancy and $Delay_r$ on PALS performance

evaluation and showed that PALS can gracefully cope with different dynamics in the system including bandwidth variations, peer participation, and partially available content.

We plan to expand this work in two directions. On the design side, we plan to explore several issues including performance of PALS over smoother TCP-friendly congestion control mechanism such as TFRC, adding support for VBR layered encoding streams as well as performing per-sender QA based on available bandwidth from individual senders rather than aggregate bandwidth. On the evaluation side, we are currently prototyping the PALS protocol and will conduct detailed experiments over PlanetLab in a near future.

7. ACKNOWLEDGMENTS

We would like to thank Vikash Agarwal for his work on the first version of PALS simulation code.

8. REFERENCES

- [1] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr., "Overcast: Reliable multicasting with an overlay network," in *OSDI*, pp. 197–212, Oct. 2000.
- [2] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "Enabling conferencing applications on the internet using an overlay multicast architecture," in *ACM SIGCOMM*, ACM, 2001.
- [3] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," in *Workshop on Network and Operating System Support for Digital Audio and Video*, (Miami Beach, FL), 2002.
- [4] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava, "On peer-to-peer media streaming," in *ICDCS*, July 2002.
- [5] M. Castro, P. Druschel, A.-M. Kermarrec, A. R. A. Nandi, and A. Singh, "SplitStream: High-bandwidth content distribution in a cooperative environment," in *ACM SOSP*, Feb. 2003.
- [6] S. Saroui, P. K. Gummadi, and S. D. Gribble, "Measurement study of peer-to-peer file system sharing," in *SPIE Multimedia Computing and Networking*, Jan. 2002.
- [7] R. Rejaie, M. Handley, and D. Estrin, "RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet," in *IEEE INFOCOM*, (New York, NY.), Mar. 1999.
- [8] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proceedings of the ACM SIGCOMM*, 2000.
- [9] R. Rejaie and S. Stafford, "A framework for architecting peer-to-peer receiver-driven overlays," in *NOSSDAV*, 2004.
- [10] Padhye, J. Kurose, D. Towsley, and R. Koodli, "A model-based TCP-friendly rate control protocol," in *Workshop on Network and Operating System Support for Digital Audio and Video*, June 1999.
- [11] R. Rejaie, M. Handley, and D. Estrin, "Quality adaptation for congestion controlled playback video over the internet," in *Proceedings of the ACM SIGCOMM*, (Cambridge, MA.), Sept. 1999.
- [12] N. Magharei and R. Rejaie, "Adaptive Receiver-Driven Streaming from Multiple Senders," in *Technical Report CIS-TR-05-06*, University of Oregon, August 2005.
- [13] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," in *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pp. 177–186, ACM Press, 2002.
- [14] Y. Cui and K. Nahrstedt, "Layered Peer-to-Peer Streaming," in *acm-mm*, 2003.
- [15] J. Apostolopoulos, T. Wong, W.-T. Tan, and S. Wee, "On multiple description streaming with content delivery networks," in *Proceedings of the IEEE INFOCOM*, 2002.
- [16] Y. Cui and K. Nahrstedt, "Layered peer-to-peer streaming," in *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, pp. 162–171, ACM Press, 2003.
- [17] T. Nguyen and A. Zakhor, "Distributed video streaming over the internet," in *SPIE Multimedia Computing and Networking*, Jan. 2002.
- [18] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, "Promise: Peer-to-peer media streaming using collectcast," in *Proceedings of the eleventh ACM international conference on Multimedia*, pp. 45–54, ACM Press, 2003.
- [19] D. A. Tran, K. A. Hua, and T. Do, "Zigzag: An efficient peer-to-peer scheme for media streaming," in *IEEE INFOCOM*, 2003.
- [20] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicast," in *ACM SIGCOMM*, 1996.