Adaptive Multi-Source Streaming in Heterogeneous Peer-to-Peer Networks

Vikash Agarwal, Reza Rejaie
Department of Computer and Information Science
University of Oregon
{*vikash,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, streaming applications have become increasingly popular in Peer-to-peer (P2P) systems. Limited deployment of IP multicast and Quality-of-Service (QoS) over today's Internet have motivated a new paradigm based on P2P overlays where participating peers form an overlay structure to cooperatively distribute streaming content among themselves without any special support from the network.[1, 2] Supporting streaming applications in P2P networks requires two key components: *(i)* Overlay Construction, and *(ii)* Content Delivery. The first component determines how each peer should select its *parent* peer(s) (*i.e.*, point-of-attachment to the overlay) whereas the second component controls how requested content is streamed from selected parent(s). While most of the research on P2P streaming has focused on the overlay construction mechanisms,[3–5] content delivery has received little attention.[6] However, *streaming* multimedia (in particular video) content among participating peers in P2P networks is a challenging problem. Note that each peer desires to receive video streams with the maximum quality that can be streamed through its access link bandwidth. Because of the inherent *heterogeneity and asymmetry* in bandwidth connectivity among participating peers,[7] a single parent peer is unlikely to have (or may not be willing to allocate) sufficient outgoing bandwidth to stream desired quality for a receiver peer. Therefore, a receiver peer requires to select multiple parents who can collectively stream the video with the desired quality to the receiver. If sender peers are properly selected by the overlay construction mechanism (*e.g.*, they do not share a bottleneck whenever it is feasible), this multi-sender approach can provide higher overall bandwidth to the receiver and thus deliver a higher (and potentially the desired) quality. Furthermore, this approach could lead to better load balancing among parents and therefore less congestion across the network.

Design of a mechanism for streaming from a distributed group of heterogeneous peers introduces several challenges. First, since each sender peer should perform TCP-friendly congestion control (CC),[8, 9] the available bandwidth from each sender is not known a priori and could significantly change during a session. Second, connections from different senders are likely to exhibit different characteristics (*e.g.*, bandwidth and RTT), and even worse, each sender peer can potentially leave the session during delivery. Therefore, any mechanism for streaming from multiple senders should be able to smoothly adapt quality (and thus the bandwidth) of the delivered stream with variations in aggregate bandwidth from all senders (*i.e.*, be quality adaptive). More importantly, a multi-source quality adaptive streaming mechanism requires a coordination mechanism among senders. This implies that the commonly used "static" approach[3,5] of requesting separate layer (or descriptions) of a layer-structured video from each sender is unable to achieve this goal for the following reasons: First, the access link (or available network) bandwidth from a sender may not even be sufficient for delivery of a single layer. Second, the excess bandwidth from a high bandwidth sender can not be utilized for delivery of other layers. Third, the unpredictable and independent variations in available bandwidth from different senders either complicates inter-layer synchronization or requires significant amount of buffering. *In summary, efficient streaming of a video from multiple congestion controlled senders requires a coordination mechanism among senders to ensure that (i) available bandwidth from each sender is fully utilized, and (ii) delivered quality is gracefully adapted with aggregate bandwidth from all senders.*

This paper presents a coordination mechanism for quality adaptive streaming from multiple congestion controlled sender peers to a single receiver peer. We call this mechanism *P2P Adaptive Layered Streaming* or *PALS* . *PALS* is a receiver-driven approach that allows a receiver to orchestrate the adaptive delivery of layer structured (*i.e.*, either layer or multiple description encoded) streams from heterogeneous sender peers. Given a set of sender peers, the receiver monitors the available bandwidth from each sender and periodically determines the target quality (*i.e.*, the number of layers) that can be streamed from all senders. Then, the receiver determines required packets from different layers for each period, properly divides them among senders, and requests a subset of packets from each sender. This enables the receiver to closely control the total amount of buffering and its distribution across active layers in order to effectively absorb short-term variations in aggregate bandwidth. *PALS* achieve this goal with relatively low amount of receiver buffering (*e.g.*, seconds worth of playout), and thus it can be used in a spectrum of *non-interactive* streaming applications ranging from playback to live (but non-interactive) sessions in P2P networks.

PALS is part of a larger architecture that we are developing for peer-to-peer streaming. The architecture integrates *PALS* with a "bandwidth-aware" overlay construction mechanism, called *PRO*.[10] *PRO* enables each receiver to independently select a proper subset of peers that can collectively maximize its aggregate bandwidth. Then, *PALS* coordinates quality adaptive streaming from these senders. This division of functionality provides a great deal of flexibility because it decouples overlay construction from delivery mechanism. This also allows *PALS* to be integrated with other overlay construction mechanisms. This paper focuses only on design and evaluation of *PALS* and therefore we do not discuss how a set of proper sender peers is identified. Although we motivated *PALS* mechanism in the context of P2P streaming, it should be viewed as a generic coordination mechanism for any multi-source streaming system across the Internet.

PALS is built on our previous work on the design of sender-based quality adaptation (QA) mechanism for unicast streaming of layer encoded video.[11] While the design goal of *PALS* is in essence similar to our previous work, *PALS* introduces the following new challenges that requires a new approach: *(i)* the receiver should simultaneously deal with multiple connections with potentially independent variations in bandwidth, *(ii)* the receiver can "loosely" control delivered segments (and thus the buffer state) during each period due to the delay in the control loop, *(iii)* limited availability of future segments (for live sessions) affects the flexibility of QA mechanism in requesting and buffering these segments. Finally, our initial idea for receiver-driven streaming from multiple senders was presented in our earlier work.[12] The contributions of this paper are *(i)* complete design of the *PALS* mechanism, and *(ii)* detailed simulation-based evaluations of *PALS* that provides useful insights on design tradeoffs and underlying dynamics of multi-source streaming mechanisms. The rest of this paper is organized as follows: In Section 2, we briefly present the related work. Section 3 describes an overview of the *PALS* framework. Further descriptions on key components of *PALS* are provided in Section 3.2, 3.3 and 3.4. We present our simulation-based evaluations in Section 4. Finally, Section 5 concludes the paper and presents our future plans.

2. RELATED WORK

There have been a few previous studies on various aspects of multi-sender streaming to a single client. Apostolopoulos et al.[13] 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.[14] 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.[15] 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

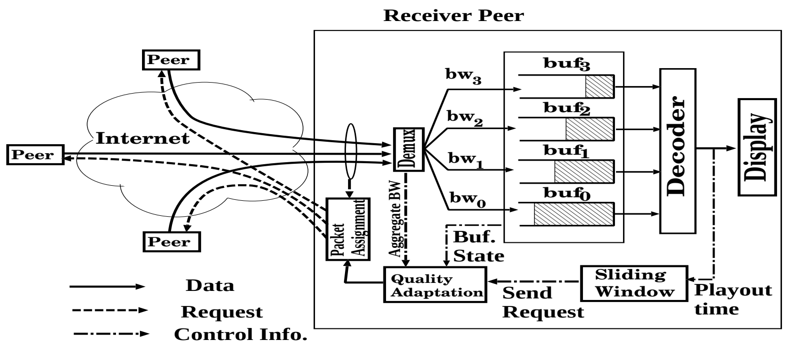


Figure 1. Internal Architecture of a PALS receiver

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, 16, 17] 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[18] 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.*

3. P2P ADAPTIVE LAYERED STREAMING

Assumptions & Target Environment: The target environment for the *PALS* mechanism is shown in Figure 1 where a receiver peer coordinates streaming of a single video from a group of sender peers. Each sender establishes a separate UDP connection to the receiver and performs TCP-friendly congestion control (*e.g.*, RAP[8] or TFRC[19]) over that connection. Senders could have different bandwidth and RTT to the receiver, any subset of them might reside behind a shared bottleneck, and any of them could leave the session without any prior notice. An overlay construction mechanism (*e.g.*, PRO[10]) provide information about senders to the *PALS* mechanism. Details of the overlay construction mechanism is outside the scope of this work, and will not be further discussed in this paper. To accommodate bandwidth heterogeneity, *PALS* assumes that video streams are layer encoded. However, the framework can be extended to accommodate multiple description encoding. 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). Instead, we primarily focus on three other dynamics in the system: *(i)* dynamics of bandwidth variations, *(ii)* dynamics of peer participation, and *(iii)* dynamics of partially available content. General information about a requested stream (*e.g.*, maximum number of layers, layer bandwidth, stream length) can be provided to the receiver during the initial setup phase or through an out-of-band channel.

To model 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 = $MIN(tp_i) - Delay_r$). More specifically, the receiver conservatively assumes that all senders can only provide a window of $Delay_r$ seconds worth of future packets relative to the receiver playout time ($tp_{max} = tp_r + Delay_r$). $Delay_r$ is a 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 our notation throughout this paper.

3.1. PALS Framework: An Overview

The primary design goal of *PALS* is to effectively utilize available bandwidth from different senders and dynamically maximize delivered quality while maintaining its stability despite independent variations in bandwidth from different senders.

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^j_{ewma}	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 and receiver
$BUF[i]$	Target buffer for L_i	BUF_{add}	Total buffered data before adding a layer
$PktSize$	Packet size	ν	No of available/cached layers per sender

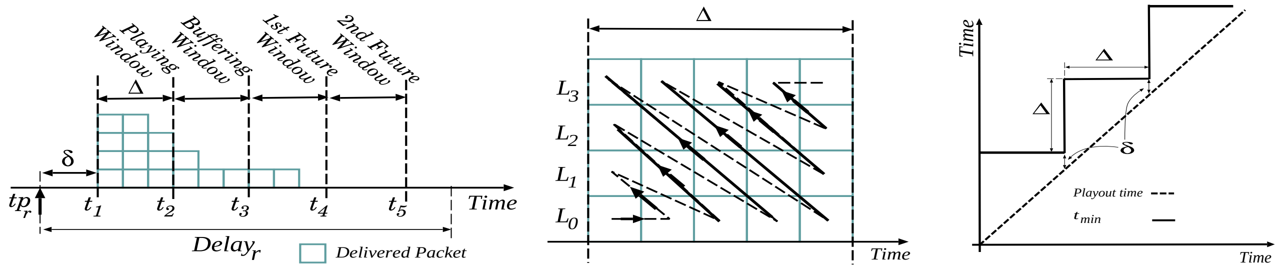
Table 1. Summary of Notations

The machinery of the *PALS* protocol is mostly implemented at the receiver, as shown in Figure 1. The basic idea in *PALS* is simple and intuitive. The receiver passively monitors the Exponentially Weighted Moving Average (EWMA) bandwidth from each sender (T^j_{ewma}) and thus can determine EWMA of aggregate bandwidth from all senders (T_{ewma}). The receiver deploys a *Sliding Window* (SW) approach to coordinate delivery from a group of senders. At a beginning of each window (or period), the receiver assumes that the current value of T_{ewma} remains unchanged for one window and estimates the total number of incoming packets (K) during this window: $K = \frac{T_{ewma}*\Delta}{PktSize}$. Then, the *Quality Adaptation* (QA) mechanism is invoked to determine *(i)* the number of *active* layers that can be played during this window, and *(ii)* specific packets from each active layers that should be requested for delivery during this window, based on the estimated budget of incoming packets, stream bandwidth (*i.e.*, $n \times C$) and receiver's current buffer state (buf_0, buf_1, .., buf_n). For example, if the receiver expects to receive 50 packets during a period (K = 50) where four layers are currently being played, the QA mechanism may allocate k_0 = 20, k_1 = 15, k_2 = 10, k_3 = 5 packets to layer L_0 .. 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, .. ,bw_n) during one window, which in turn determines evolution of receiver's buffer state.

Finally, the *Packet Assignment* (PA) mechanism divides selected packets into disjoint subset (possibly from different layers), and sends a separate request to each sender that contains an *ordered* list of assigned packets to that sender. Number of requested packets from each sender per window is proportional to its contribution in aggregate bandwidth. Each sender simply transmits requested packets in the given order at the rate that is determined by its congestion control mechanism. Thus, each sender only maintains a single list of pending packets for each receiver peer. Ordering requested packets from each sender has two benefits: *(i)* it provides sufficient flexibility for the receiver to implement different quality adaptation strategies, *(ii)* it allows the receiver to prioritize requested packets based on its own preferences (*e.g.*, in an encoding-aware fashion). This in turn ensures graceful degradation in quality when bandwidth of a sender suddenly decreases since the available bandwidth is used 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 about the three key components of the *PALS* mechanism are presented in the following subsections.

3.2. Sliding Window

In this section, we present further details on the Sliding Window mechanism and its interaction with other components. The receiver maintains a window of time [t_{min}, $t_{min} + \Delta$] called *active buffering window*. This interval represents a range of timestamps for packets (from all active layers) that must be requested and received during one window. t_{min} and Δ denote the left edge and length of the window, respectively. As shown in Figure 2(c), when the gap between the left edge of the window and the playout time reaches a minimum threshold δ, ($t_{min} - tp_r \leq \delta$), the window is slided forward in step-like fashion ($t_{min} \leftarrow t_{min} + \Delta$). Figure 2(a) depicts status of the window right after a sliding has occurred and t_{min} = t_2. At this point, most of the packets with timestamp lower than t_{min} have already been delivered. The requested packets at the beginning of the current buffering window can be divided into three groups based on their relative timestamp as follows:
Packets from Last Window ($t_{min} - \Delta \leq ts < t_{min}$): these packets belong to the previous window but they have not been delivered yet. However, there is still sufficient time for delivery of these packets. These packets are determined by loss recovery mechanism and their number should be very small.



(a) Relative timing of windows & $Delay_r$ (b) Packet ordering inside a window (c) Sliding Window in *PALS*

Figure 2. Sliding window & packet ordering in *PALS*

Packets from Buffering Window ($t_{min} \leq ts < t_{min}$+$\Delta$): these packets belong to the current window and must be requested at this point. Note that some of the packets from the current window are often delivered ahead of time. Thus, these group often does not constitute a full window worth of packets.
Packets from Future Windows (t_{min}+$\Delta \leq ts < tp_r$+$Delay_r$): These packets belong to the future window(s) and can be delivered if excess bandwidth is available or most of the packets within the current window were already delivered. These packets are identified by the QA mechanism as we describe in the following section. Note that the timestamp of these packets are limited by availability of future data which is specified by $Delay_r$.

Since the QA mechanism is invoked once per window, the window size (Δ) is a key parameter that determines the tradeoff between stability of delivered quality and responsiveness of the QA mechanism to variations of aggregate bandwidth. The window size should be set to multiple of smoothed RTT (SRTT) (*e.g.*, Δ = 5*$SRTT$) for a sender 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. In *PALS*, all senders periodically report their SRTT to the receiver. Since the receiver deals with multiple senders with potentially different SRTTs, the receiver conservatively selects the largest SRTT value to set the window. Due to the unpredictable variations in bandwidth from senders, a *PALS* receiver may request a packet that is being delivered and receive duplicate packets. Note that right after the window is slided forward, there is half an RTT worth of packets in flight from each sender that will arrive during the next window and should not be requested again. While the receiver can roughly estimate the number (and identity) of these packets, such an estimate could be inaccurate when bandwidth of one (or more) sender(s) suddenly changes towards the end of a window. Thus, the receiver may request redundant copies of some packets. As we show our evaluation, ratio of duplicate packet is very low and can be further reduced by increasing window size.

Sliding window mechanism is not sufficient to effectively cope with sudden change in aggregate bandwidth during one window. Therefore, the sliding window mechanism is coupled with several secondary mechanisms to address this issue:
Overestimating Bandwidth: The receiver employs two mechanisms to cope with major drops in bandwidth:
1) Overwriting Requests: Any new list of requested packets from the receiver *overwrites* any outstanding list 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 prevents slow senders from falling behind the playout time.
2) Packet Ordering: At a macro level, requested packets can be ordered based on their corresponding windows. Micro-level ordering of packets within each window is done as follows: ordering of packets from future windows is determined by the QA mechanism as we describe in the next subsection. However, micro-level ordering of requested packets from playing and buffering windows is done based on the combination of their layer number and timestamp using the following algorithm. 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 if the following condition is satisfied, $\frac{n}{\Delta} > \frac{L_a - L_b}{ts_a - ts_b}$, 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(b)) to ensure that packets from lower layers or with lower timestamps are given higher priority. Thus, potentially undelivered packets will be at the right, top corner of the window that have more time for delivery in the next window. As the above condition indicates, the slope of the ordering adaptively changes with both n and Δ.
Underestimating Bandwidth: If available bandwidth from a sender significantly increases during a period, the sender may deliver all the requested packets and become idle for the remaining portion of the period. This in turn reduces bandwidth utilization of that sender. To tackle this problem, the receiver simply requests a percentage of extra packets (*e.g.*, 20%)

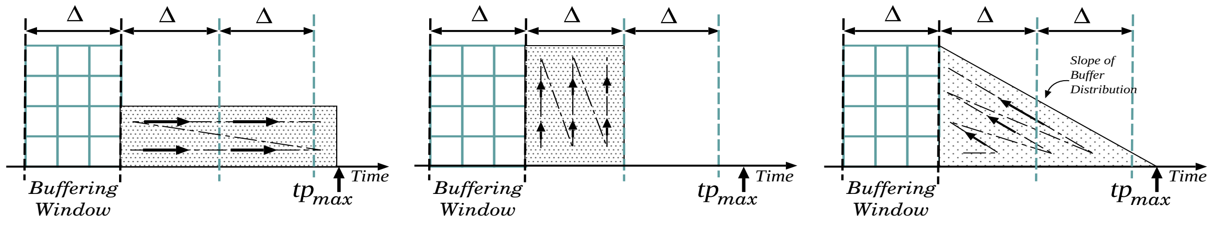
beyond estimated number of incoming packets. This percentage is a configuration parameters, called *window overlap*. Extra packets are selected (and ordered) by the QA mechanism as we discuss in next subsection. In our experiments, we have decoupled *PALS* from the underlying CC mechanism. Each sender sends a packet at a rate that is determined by its CC mechanism. If there is an outstanding *PALS* packet, it will be mapped to an outgoing packet. Otherwise, an empty packet is sent. This decoupling allows us to determine how effectively *PALS* utilizes available bandwidth without interfering with the behavior of the CC mechanism.

3.3. Quality Adaptation

The QA mechanism is invoked once per window to determine required packets based on variations of EWMA of aggregate bandwidth from all senders. Note that variations of aggregate bandwidth over timescales shorter than a window are smoothened by a window worth of buffering at the receiver. 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. Once receiver's buffers are filled to the appropriate level (BUF_{add}), 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 QA mechanism can drain receiver's buffers to compensate the bandwidth deficit. If the amount of buffered data or its distribution among active layers is inadequate to absorb bandwidth deficit during a draining window, the QA mechanism drops the top layer. In a nutshell, the QA mechanism has two degrees of control to match stream bandwidth with average aggregate bandwidth.
1) Coarse-grained Adaptation: it can add/drop the top layer to adjust the number of playing layers in response to *long-term* mis-match between available bandwidth and stream bandwidth.
2) Fine-grained Adaptation: it can control evolution of buffer state by adjusting fine-grained distribution of aggregate bandwidth among active layers to absorb *short-term* mis-match between stream bandwidth and aggregate bandwidth.

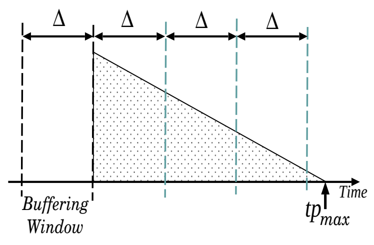
BUF_{add} is the basic design parameter that controls the level of smoothing for delivered quality. The bigger the value of BUF_{add}, the longer it takes to add a new layer during a filling window, the less likely to drop the layer during a draining window, thus the more decoupled the delivered quality becomes from variations of aggregate bandwidth. Distribution of total buffered data across active layers (*i.e.*, buf_0, buf_1, .., buf_n) is the main factor in design of a layered quality adaptation mechanism. Figure 3(a) and 3(b) demonstrate two extreme buffer distributions for the same BUF_{add}. The conservative distribution utilizes excess bandwidth to request packets of lower layers in future windows whereas the aggressive distribution uses the excess bandwidth to request future packets of all layers with lowest timestamps. The conservative distribution achieves long-term smoothing of quality by distributing the benefit of excess bandwidth across multiple windows at the cost of risking short term drop in quality. In contrast, the aggressive distribution achieves short-term improvement in quality at the cost of lower degree of long-term smoothing. Because of variations of aggregate bandwidth during each window, evolution of buffer state depends on *(i)* how requested packets are ordered within each window, and *(ii)* how requested packets are assigned to different senders (which is addressed in the next subsection). Figure 3(a) and 3(b) depict corresponding ordering for both conservative and aggressive distributions where packets are ordered only based on their layer number or timestamp, respectively.

Quality Adaptation in PALS: To leverage the tradeoff between long-term smoothing and short-term improvement in a balanced fashion, the QA mechanism in *PALS* dynamically adjusts *(i)* BUF_{add}, *(ii)* distribution of total buffered data among active layers, and *(iii)* ordering of requested future packets, with the number of active layers. As shown in Figure

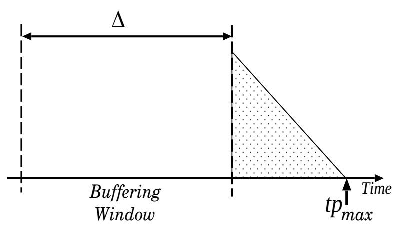


(a) Conservative buffer distribution (b) Aggressive buffer distribution (c) Buffer distribution in PALS

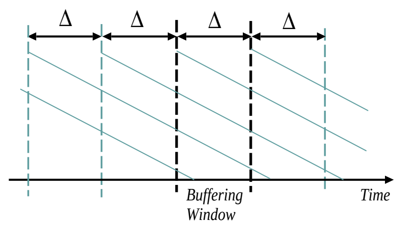
Figure 3. Conservative, aggressive and *PALS* buffer distribution along with their corresponding packet orderings



(a) Small window size



(b) Large window size



(c) Effect of window size on number of opportunities for packet request

Figure 4. Effect of window size on amount of future buffering

3(c), the QA mechanism uses a diagonal buffer distribution (with slope equal to $\frac{n*C}{Delay_r-(\delta+2\Delta)}$) and the corresponding snake-shape ordering of requested packets.

Note that the window size affects both amount and slope of buffer distribution in *PALS* as shown in Figure 4. This figure clearly illustrates that for a fixed $Delay_r$, as window size increases, the room for future buffering (*i.e.*, size of the triangle) decreases which in turn limits the flexibility and thus achieved smoothing of the QA mechanism. More importantly, the window size affects the slope of buffer distribution which determines how buffered data spans over multiple future windows. As shown in Figure 4(c), given a fix $Delay_r$, decreasing the window size, increases the overlap between buffered distribution across consecutive windows which in turn increases the average number of opportunities to request a packet.

Loss Recovery: By providing multiple opportunities for requesting packets specifically for lower layers, *PALS* achieves higher degree of robustness against losses in lower layers. All the missing packets of playing layers within the buffering window are requested for the last time and might be dropped by the network. To recover these losses in each window, *PALS* explicitly re-request the missing packets of the playing window. These missing packets are usually located at the right side of the playing window among higher layers. Thus, they have sufficient time for delivery. To ensure that explicit loss recovery does not starve flow of fresh packets, the portion of allocated bandwidth for loss recovery is limited to EWMA of observed loss rate by the receiver.

3.4. Packet Assignment

Once all selected packets for each window are ordered, they must be proportionally assigned to different senders. The assignment should ensure that more important packets are delivered despite any drop in available bandwidth from any sender. To achieve this goal, we adopt a *weighted round-robin* strategy for packet assignment. 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 flexibility of packet assignment. This strategy attempts to proportionally distribute less important packets at the end of different lists.

Partially Available content: In practice, each sender may not have all the layers. 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. Given this information, the receiver can perform packet assignment in two passes. In the first pass, the receiver assumes that all senders have the entire content and assigns requested packets in a round-robin fashion, as we described earlier. Then, the receiver examines whether assigned packets to each sender are actually available at that sender or not. Any subset of assigned packets that are available at the corresponding sender, are assigned to that sender, and the remaining packets are considered in the second pass. In the second pass, the receiver performs per-packet assignment. Toward this end, it starts from the first unassigned packet (with the lowest timestamp), and searches for another sender that can provide the packet. This process continues until all packets are examined. If multiple senders can accommodate a packet during the second pass, the receiver assigns the packet to a sender that has lower ratio of assigned-to-total packets in order to ensure earlier delivery time for that packet.

4. PERFORMANCE EVALUATION

We have extensively evaluated *PALS* mechanism using *ns2* simulator to examine the importance of coordination among senders, and evaluate its performance under the following dynamics in P2P systems: *(i)* dynamics of bandwidth variation, *(ii)* dynamics of available content, and *(iii)* dynamics of peer participation. All sender peers employ RAP[8] mechanism to perform TCP-friendly congestion control and are able to provide all layers of a requested stream to the receiver. This latter assumption is relaxed in Section 4.5 when we examine dynamics of content availability. In our simulations, a receiver accepts two parameters, $Delay_r$ and Δ, that are defined as a function of $SRTT_{max}$. The receiver emulates per-packet data consumption and monitors delivered quality while considering vertical dependency among layers, *e.g.*, missing or late packets for L_i implies that corresponding packets of higher layers are useless. The simulation topology in Figure 5(a) is used for most of our simulations. The presented results are averaged over 20 runs with different random seeds. Unless otherwise stated, the following default parameters are used in our simulations: window overlap = 20%, N_c = 5, C = 80 KBytes/sec, δ = $2 * SRTT_{max}$. For each *PALS* sender, three long-lived TCP flows are introduced as the bottleneck cross traffic. Due to lack of space, we only present a subset of our results in this section. Further results can be found in the related technical report.[20]

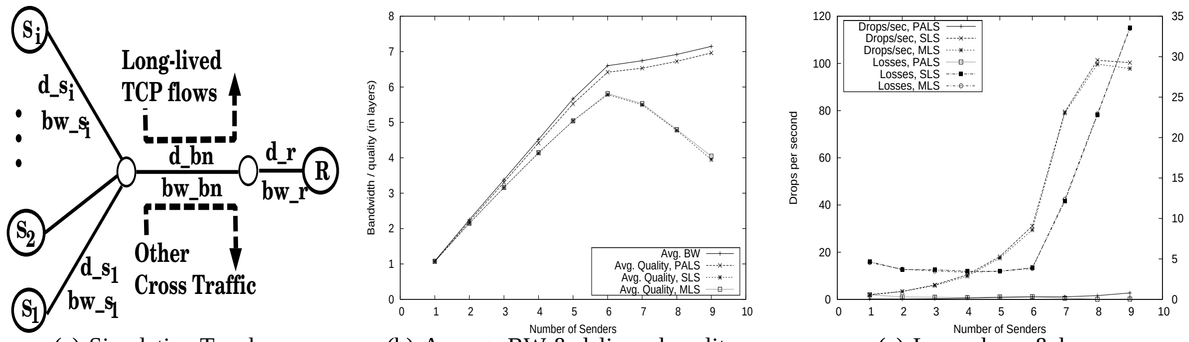
Performance Metrics: We use the following metrics to fully capture *PALS* performance: *Bandwidth Utilization* presents portion of aggregate bandwidth that is utilized for sending *PALS* packets. *Delivered Quality* is captured by two metrics: *(i)* Average Delivered Quality presents weighted average of number of delivered layers. *(ii)* Layer Drops presents total number (or frequency) of layer drops during a simulation. We also keep track of *Number of Duplicate Packets* and *Unrecovered Losses* at the receiver. We did not observe any late packet in our simulations. Note that the ratio of control bytes to data bytes (*i.e.*, request overhead) in our simulations changes with Δ and window overlap. This ratio was always less than 1% in our simulations, and could be further improved by applying more efficient encoding to control information.

4.1. Necessity of Coordination Among Senders

One of the key challenges in any multi-source streaming mechanism is to coordinate in-time delivery of required packets among senders such that *(i)* aggregate bandwidth is fully utilized, and *(ii)* delivered quality is maximized with minimum variations. To examine this issue, we compare the performance of *PALS* with the following two strawman mechanisms that do not use any coordination among senders (*i.e.*, layer assignment to each sender is static). *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 defined by its congestion control mechanism. SLS represents a commonly used layer assignment strategy in many previous studies. *Multiple Layer per Sender (MLS)*: This approach is more elaborate and assigns multiple layers L_j (for any $j \leq i$) to sender s_i. The layer assignment strategy in MLS provides higher degree of redundancy for packets of lower layers that are more important. To ensure in-time delivery of transmitted packets by SLS and MLS mechanisms, timestamp of transmitted packets are specified as follows: the receiver reports its most recent playout time (tp_{rcv}) in each ACK packet to the corresponding sender. At each packet transmission time, each sender sends a new packet from the assigned layer(s) with the lowest timestamp between [tp_{rcv}, tp_{rcv} + $Delay_r$]. In the case of MLS, all packets of each layer are sent before packets of any lower layers.

Figure 5(b) and 5(c) present the performance of SLS, MLS and *PALS* with different number of heterogeneous senders. For these simulations, we use the topology shown in Figure 5(a) with the following parameters: bw_s_i = bw_bn = bw_r = 5Mbps, and delay for sender-side access links are such that RTT of sender s_i is $30ms + (i-1) * 30ms$. For a scenario with i senders, only senders s_1 to s_i participate in delivery. $Delay_r$ and Δ are set to $60 * SRTT_{max}$ and Δ = $16 * SRTT_{max}$, respectively.

The results for homogeneous senders are very similar. Note that aggregate bandwidth to the receiver for all three mechanisms is similar, but their ability to effectively utilize the bandwidth is different. When the number of senders is small, access link of each sender is the bottleneck, and thus the aggregate bandwidth increases with the number of senders. However, once the number of senders exceeds a threshold (6 senders), receiver's access link becomes the bottleneck and the aggregate loss rate increases with the number of senders. Figure 5(b) depicts average delivered quality for all three approaches. *PALS* can effectively utilize the aggregate bandwidth and maximize delivered quality. The small gap between aggregate bandwidth and delivered quality by *PALS* is primarily due to utilized bandwidth for duplicate and future packets. For small number of senders, average delivered quality by both SLS and MLS is lower than *PALS* but still comparable. However, once receiver's access link becomes the bottleneck, further increase in the number of senders rapidly reduces delivered quality by SLS and MLS because of their inability to recover losses. The key difference among these mechanisms



(a) Simulation Topology (b) Average BW & delivered quality (c) Layer drops & losses

Figure 5. Benefit of coordination for different number of heterogeneous senders

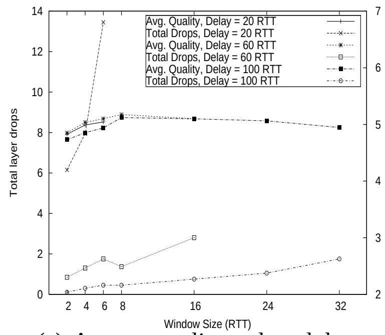
is the stability of delivered quality that is measured by number of drops per second and shown in Figure 5(c). While *PALS* manages to deliver smooth quality (*i.e.*, less than 3 average drops per second) for any number of senders, both SLS and MLS exhibit high variations in quality (around 15 drops/second) even for small number of senders. Even worse, these variations linearly increases with the aggregate loss rate which is particularly problematic for a large number of senders. It is worth noting that inter-layer decoding dependency among packets significantly magnifies the impacts of individual losses (specially from lower layers) on delivered quality. *PALS* minimizes the probability of residual loss in lower layers because the receiver *implicitly* performs loss recovery by coordinating delivery of most important packets with proper ordering among senders. However, SLS and MLS not only do not perform any loss recovery but they also are unable to effectively utilize available bandwidth of different senders due to the lack of coordination among them. More specifically, excess bandwidth at one sender can not be used to recover lost packets from other senders. It is worth clarifying that adding a loss recovery mechanism to individual senders does not entirely address this problem because there is no coordination among senders to prioritize recovery of lost packets. Quantifying the impact of loss recovery on performance of SLS and MLS remains as future work.

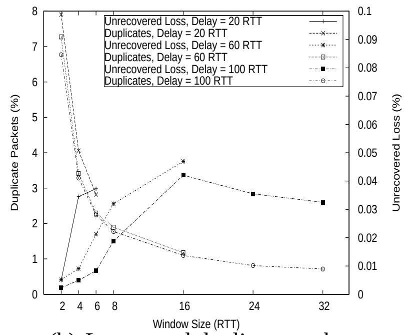
4.2. Examining Delay-Window Tradeoff

As we mentioned earlier, there appears to be a tradeoff between two key parameters of the *PALS* mechanism, $Delay_r$ and Δ. To examine this tradeoff, we consider a scenario where three heterogeneous senders with shared bottleneck stream content to a single receiver. For these simulations, we use the topology shown in Figure 5(a) with the following parameters: bw_s_i = 5Mbps, d_s_i = [5,35,95]ms, d_bn = 20ms, d_r = 5ms, bw_bn = 13Mbps, bw_r = 20Mbps. Both $Delay_r$ and Δ are shown in terms of $SRTT_{max}$ and the maximum window size is limited to $\frac{Delay_r}{3}$ in all simulations. Bandwidth utilization for this set of simulations is 94%, and is independent of $Delay_r$ and Δ.

Figure 6(a) shows the average delivered quality and number of layer drops for three different $Delay_r$ values as Δ increases. This figure demonstrates three points: First, average delivered quality only depends on aggregate bandwidth and does not vary with $Delay_r$ or Δ. Second, for a given window size, increasing $Delay_r$ rapidly reduces variations in delivered quality because it increases the opportunity for future buffering, *i.e.*, larger triangle in Figure 4(b). Third, for a fixed $Delay_r$, increasing window size leads to higher degree of variations in delivered quality. In particular, for small $Delay_r$ values, variations in delivered quality is very sensitive to Δ. To explain the last two points, we recall that the total amount of buffering depends on the difference between $Delay_r$ and Δ. Any increase in total buffering results in smoother delivered quality because it provides better protection to absorb bandwidth variations, and also adding a layer becomes more conservative.

Figure 6(b) depicts the ratio of duplicate and lost packets for the same simulations. This figure clearly shows that the ratio of duplicate packet remain below 8% in all scenarios. More importantly, it exponentially drops with the window size and is independent of $Delay_r$. This indicates that the sliding window mechanism is the primary (and possibly the only) cause for requesting duplicate packets. Unrecovered losses remain below 0.05 of one percent in all cases, but they slightly increase with window size because of the smaller number of opportunities to retransmit each packet. Figure 6(c) depicts receiver's buffer distribution in these simulations. Buffer distribution for each simulation is shown by average buffered data per-layer over the entire simulation, and vertical bars present average slope of buffer distribution among active layers in each simulation. Per-layer buffering is normalized by layer bandwidth and shown in terms of "seconds worth of playout". This figure shows that the average buffered data per-layer almost linearly increases with $Delay_r$ independent


(a) Average quality and total drops


(b) Losses and duplicate packets

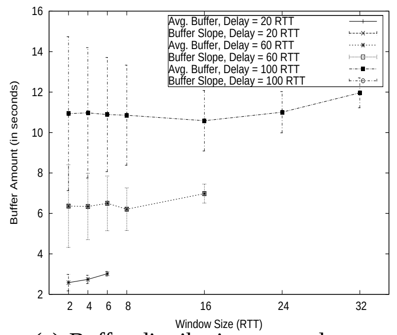

(c) Buffer distribution across layers

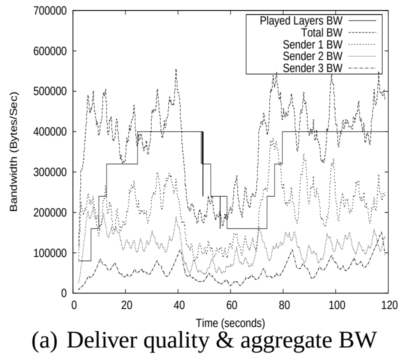
Figure 6. Delay-window tradeoff

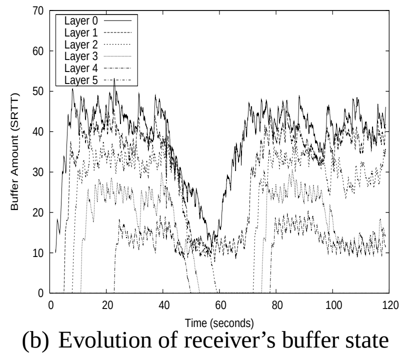
of Δ. However, increasing Δ results in a less skewed buffer distribution among layers which decreases buffer efficiency as we discussed in Section 3.3. *In summary, these results collectively show that using higher $Delay_r$ parameter leads to more opportunity for buffering and smoother quality. In practice, $Delay_r$ is often limited by either the desired degree of interactivity or available buffering at the receiver. For a selected $Delay_r$ parameter, increasing Δ can exponentially reduce ratio of duplicate packets at the cost of higher variations in quality and lower efficiency for buffer distribution.*

4.3. Dynamics of Bandwidth Variation

To examine *PALS* performance under different dynamics of bandwidth variations, we have simulated various scenarios including senders with or without shared bottleneck and different cross traffic on the bottleneck (*e.g.*, long-lived TCP flows, on-off CBR, and flash-crowd). Here, we present our results for a scenario where three senders stream content through a shared bottleneck with flash-crowd cross traffic that starts at t=40 and stops at t=70 second during the 120 second simulation. For these simulations, we use the same topology as in section 4.2, but add flash-crowd cross traffic to the shared bottleneck. $Delay_r$ and Δ are set to $60 * SRTT$ and $6 * SRTT$, respectively. We model flash-crowd traffic by 300 short-lived TCP flows where each TCP flow starts at a random time between t=40 and t=70 second and remains active for a random period between 0.8 to 1.2 seconds. Therefore, almost all the TCP flows remain in their slow-start phase and produce an abrupt change in network load.

Figure 7(a) shows the variations of per-sender and aggregate bandwidth along with variations of delivered quality. Delivered quality gradually reaches 5 layers during the initial 27 seconds of the simulation. Once flash-crowd cross traffic starts, the aggregate bandwidth drops significantly and triggers 3 gradual layer drops. Setting $Delay_r$ to a relatively large value enables the receiver to buffer more data. Therefore, *PALS* becomes more conservative in adding a new layer during filling windows, and more resilient in dropping existing layers during drain windows, as shown in Figure 7(a). The negative spikes in quality (around t=50 and t=55 sec) are due to single unrecovered losses in the top layer. Such unrecovered losses might occur right before dropping the top layer when loss rate usually increases but the allocated bandwidth for recovery is not sufficient to retransmit all losses. Therefore, less important losses (from higher layers) may not be retransmitted.


(a) Deliver quality & aggregate BW


(b) Evolution of receiver's buffer state

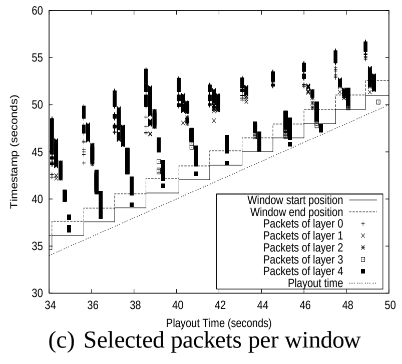
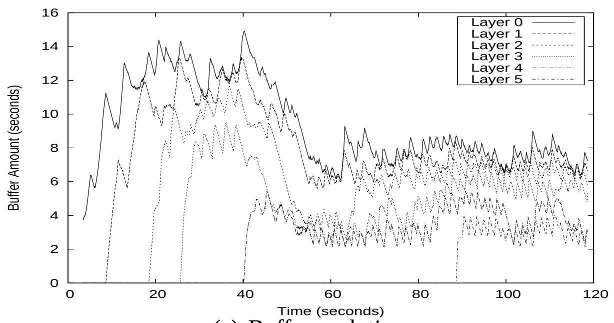

(c) Selected packets per window

Figure 7. Behavior of PALS with flash crowd cross traffic

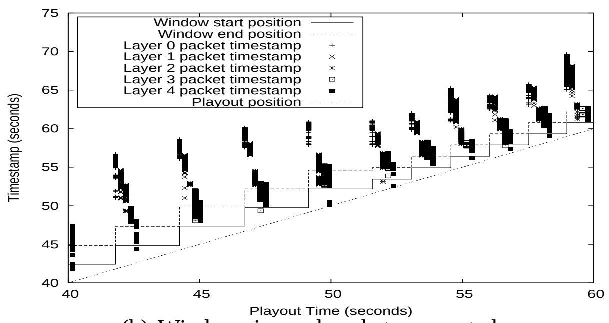
Figure 7(b) depicts the evolution of receiver's buffer state as a function of time for the same simulation. The buffered data for each layer is shown in terms of $SRTT_{max}$ worth of playout. This figure shows that a new layer is added only after existing layers reach sufficient level of buffering. In contrast, when the available bandwidth drops, higher layers (that are shown by lower lines in the figure) are drained more rapidly and eventually dropped in order to protect lower layers. Figure 7(c) shows the timestamp of the requested packets in each window along with the evolution of window position and playout time for part of the same simulation. The playout time increases at a constant rate whereas window is slided forward in a step-like fashion. Group of requested packets from different layers in each window are shown as parallel columns on top of each window. Even though packets of all layers are requested at the start of the window, they are separated in this figure for clarity (the left most column shows packets of L_0). The relative distance of each packet from its corresponding window is proportional to its timestamp. This figure shows that packets of lower layers usually have higher timestamps since these layers are buffering more data. The relative difference between timestamps of packets from different layers depends on the slope of buffer distribution. 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 window (but still ahead of playout time) are occasionally requested for explicit loss recovery mechanism (*e.g.*, packets of L_4 around t=47 sec). This figure clearly demonstrates that each packet has multiple number of opportunities to be requested. 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 time. A packet might be requested multiple times due to packet loss or overwritten request at senders. As shown in Figure 7(c), re-requested random losses usually have scattered timestamps whereas timestamps of overwritten packets are often clustered.

4.4. Dynamics of Peer Participation

Any change in sender peers can affect delivered quality to a receiver in two ways: First, sender departure or arrival might change aggregate bandwidth to the receiver and thus affect the delivered quality. This effect is very similar to dynamics of bandwidth variations that we examined in subsection 4.3. Second, any change in senders could affect $SRTT_{max}$ which determines adaptation frequency (or Δ) for *PALS* receiver. Lower $SRTT_{max}$ enables the receiver to reduce its window size either to achieve the same degree of smoothness with lower $Delay_r$ (and thus less buffering) or to improve its performance with the same $Delay_r$ value (as we showed in subsection 4.2). To examine the effect of $SRTT_{max}$ on *PALS* behavior, we simulate a scenario similar to subsection 4.2 with four senders (d_s_i = [5,35,65,125]). Initially s_2, s_3 and s_4 participate, then receiver replaces s_4 with s_1. Figure 8(a) depicts the evolution of buffer state where a departing sender (at t=50 sec) with $SRTT$ = 300 ms is replaced with another sender with $SRTT$ = 60 ms. Once $SRTT$ of the new sender is provided to the receiver, it adjusts its window size and proportionally decreases $Delay_r$ based on the new $SRTT_{max}$ among remaining senders which is 180 ms. This figure illustrates how reducing Δ and $Delay_r$ rapidly decreases total amount of receiver buffering and its distribution. Note that the receiver should properly increase its total buffering to cope with any increase in $SRTT_{max}$ due to any change in senders. Figure 8(b) shows the behavior of sliding window mechanism and timestamps of requested packets in the same simulation. This figure demonstrates that once Δ and $Delay_r$ are adjusted, *(i)* the window becomes smaller and thus it is slided more frequently, and *(ii)* total amount of buffering is reduced and more evenly distributed.



(a) Buffer evolution



(b) Window size and packets requested

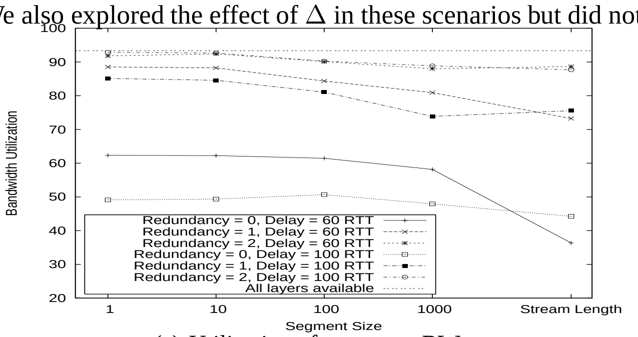
Figure 8. Effect of changing senders on buffer state and sliding window mechanism

4.5. Dynamics of Content Availability

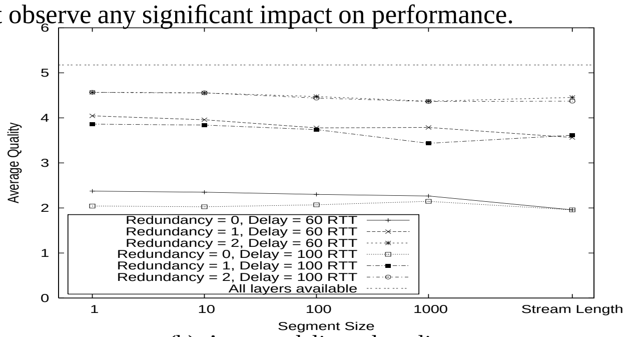
In practice, each sender peer may not receive all layers, or may not keep (*i.e.*, cache) all segments of the received layers. In this subsection, we examine how pattern and bandwidth of cached content at each sender affect *PALS* performance. To properly present cached content at each sender, we divide each stream into equal-sized segments where each segment consists of a group of consecutive packets. Available content at a sender can be specified by two parameters: *(i)* segment size (in terms of packets) and *(ii)* number of cached layers per segments (ν). More specifically, each sender caches ν randomly-selected layers for each segment of a stream. Note that segment size directly determines pattern of cached content at each sender. Using 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 ν complete layers of each stream. Bandwidth of available content at each sender is equal to cumulative bandwidth of all cached layers, *i.e.*, bandwidth = $\nu * C$. To fully utilize available bandwidth from a sender, bandwidth of cached layers by the sender should be higher than its available bandwidth to the receiver. This implies that number of cached layers at different senders should be proportional to their contributions into aggregate bandwidth. For example, if three senders s_1, s_2 and s_3 contribute equal to 3, 2 and 1 layer worth of bandwidth, they should cache at least 3, 2 and 1 layer, respectively. We use the notation [3,2,1] to present this distribution of layers among senders. When each sender only provides the minimum number of layers (and thus there is a single copy of each layer among senders), degree of *redundancy* for available content is zero. In general, degree of redundancy r means that each sender has extra r layers. We note that adding the same number of extra layers to senders with different bandwidth leads to a non-uniform degree of redundancy among senders. However, this issue is inevitable when redundancy is adjusted at a per-layer granularity. Examination of more fine-grained redundancy remains as future work.

To examine the dynamics of content availability, we simulate the scenario similar to section 4.2 but each sender can only provide a subset of active layers. We adjust segment size and degree of redundancy to control pattern and bandwidth of cached content among senders. In these simulations, three senders can collectively stream six layers where layer distribution of [3,2,1], [4,3,2] and [5,4,3] presents the redundancy degree of 0, 1 and 2, respectively.

Figure 9 depicts the impact of degree of redundancy and segment size on utilization of aggregate bandwidth and average delivered quality for two different $Delay_r$ parameters: $60 * SRTT$ and $100 * SRTT$ with Δ = $8 * SRTT$. As a reference for comparison, we also show the result when all layers are available to all senders. These figures illustrate that increasing segment size results in lower bandwidth utilization and lower delivered quality. This effect is more visible when degree of redundancy is minimal. This result can be explained as follows: total number of requested packets in each window is not equally divided among active layers (*i.e.*, more packets of lower layers are requested). Therefore, increasing segment size reduces the flexibility of the packet assignment mechanism to utilize available bandwidth of a sender specially when the sender can only provide the highest layer. This figure also show that minor increase in the degree of redundancy can significantly relieve this restriction and improve performance. When degree of redundancy is zero, higher $Delay_r$ values result in lower quality but they are less sensitive to segment size. This is due to the fact that increasing $Delay_r$ widens the range of timestamps for requested packets within a single window. This decreases the negative impact of large segment size but it results in more skewed distribution of requested packets per window and further drops bandwidth utilization. For higher degree of redundancy, these constraints are relieved and $Delay_r$ does not have a major impact on *PALS* performance. We also explored the effect of Δ in these scenarios but did not observe any significant impact on performance.



(a) Utilization of aggregate BW



(b) Average delivered quality

Figure 9. Impact of content availability and segment size on BW utilization and unrecovered losses

5. 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 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 CC 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.

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* 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* 2001.
3. V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," in *NOSSDAV*, 2002.
4. D. Xu, M. Hefeeda, S. Hambrusch, and B.Bhargava, "On peer-to-peer media streaming," in *ICDCS*, 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*, 2003.
6. T. S. E. Ng, Y. Chu, S. G. Rao, K. Sripanidkulchai, and H. Zhang, "Measurement-based optimization techniques for bandwidth-demanding peer-to-peer systems," in *IEEE INFOCOM*, 2003.
7. S. Saroiu, P. K. Gummadi, and S. D. Gribble, "Measurement study of peer-to-peer file system sharing," in *SPIE MMCN*, 2002.
8. 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*, 1999.
9. S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *ACM SIGCOMM*, 2000.
10. R. Rejaie and S. Stafford, "A framework for architecting peer-to-peer receiver-driven overlays," in *NOSSDAV*, 2004.
11. R. Rejaie, M. Handley, and D. Estrin, "Quality adaptation for congestion controlled playback video over the internet," in *ACM SIGCOMM*, 1999.
12. R. Rejaie and A. Ortega, "PALS: Peer-to-Peer Adaptive Layered Streaming," in *NOSSDAV*, 2003.
13. J. Apostolopoulos, T. Wong, W.-T. Tan, and S. Wee, "On multiple description streaming with content delivery networks," in *IEEE INFOCOM*, 2002.
14. Y. Cui and K. Nahrstedt, "Layered peer-to-peer streaming," in *NOSSDAV*, 2003.
15. T. Nguyen and A. Zakhor, "Distributed video streaming over the internet," in *MMCN*, 2002.
16. M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, "Promise: Peer-to-peer media streaming using collectcast," in *ACM Multimedia*, 2003.
17. D. A. Tran, K. A. Hua, and T. Do, "Zigzag: An efficient peer-to-peer scheme for media streaming," in *IEEE INFOCOM*, 2003.
18. S. McCanne, V. Jacobson, and M. Vettereli, "Receiver-driven layered multicast," in *ACM SIGCOMM*, 1996.
19. Padhye, J. Kurose, D. Towsley, and R. Koodli, "A model-based TCP-friendly rate control protocol," in *NOSSDAV*, 1999.
20. V. Agarwal and R. Rejaie, "Adaptive multi-source streaming in heterogeneous peer-to-peer networks," *Tech. Rep. CIS-TR-04-01, University of Oregon, CS Dept.* , 2004. *http://www.cs.uoregon.edu/˜ reza/PUB/CISTR-04-01.pdf*