SPANC: Optimizing Scheduling Delay for Peer-to-Peer Live Streaming

K.-H. Kelvin Chan, S.-H. Gary Chan, *Senior Member, IEEE*, and Ali C. Begen, *Member, IEEE*

Abstract—**In peer-to-peer (P2P) live streaming using unstructured mesh, packet scheduling is an important factor in overall playback delay. In this paper, we propose a scheduling algorithm to minimize scheduling delay. To achieve low delay, our scheduling is predominantly push in nature, and the schedule needs to be changed only upon significant change in network states (due to, for examples, bandwidth change or parent churns). Our scheme, termed SPANC (Substream Pushing and Network Coding), pushes video packets in substreams and recovers packet loss using network coding. Given heterogeneous contents, delays, and bandwidths of parents of a peer, we formulate the substream assignment (SA) problem to assign substreams to parents with minimum delay. The SA problem can be optimally solved in polynomial time by transforming it to a max-weighted bipartite matching problem. We then formulate the fast recovery with network coding (FRNC) problem, which is to assign network coded packets to each parent to achieve minimum recovery delay. The FRNC problem can also be solved exactly in polynomial time with dynamic programming. Simulation results show that SPANC achieves substantially lower delay with little cost in bandwidth, as compared with recent approaches based on pull, network coding and hybrid pull–push.**

Index Terms—**Network coding, optimization, peer-to-peer (P2P) streaming, scheduling, substream pushing.**

I. Introduction

IN RECENT years, we have witnessed many successful applications of peer-to-peer (P2P) technologies for live streaming, such as PPLive,[1] PPStream[2], and SopCast.[3] In P2P live streaming, peers collaboratively organize themselves into an overlay and share their upload capacities to serve others. In order to provide robustness against peer churns and to meet the streaming bandwidth requirement, a mesh overlay is usually constructed in a distributed manner, where each peer connects to some other peers as its *parents*. By retrieving packets from its parents, a *child* can aggregate and assemble a full stream to achieve stream continuity. Given a number of parents, a child needs to determine which parents are to deliver which packets, given their heterogeneous bandwidths, delays and available contents. This is called *scheduling*, which incurs delay due to control messaging, packet buffering and transmission.

In an overlay, the scheduling delay can be substantial if the scheduling is not designed properly. The playback delays for popular P2P live streaming applications nowadays have been measured ranging from tens of seconds to several minutes [1]. In order to achieve low delay in P2P live streaming, reducing or optimizing such scheduling delay is therefore critical.

Traditionally, mesh-pull is used on mesh overlay due to its simplicity, robustness, and high-bandwidth utilization (as used in Coolstreaming [2]). Mesh-pull is based on buffermap exchange, because of which a child knows exactly the packets that each parent has and can explicitly *pull* packets from each of them. However, the buffermap formation and the pulling process significantly delay packets in parent buffer [3]. Because scheduling delay propagates and accumulates along the overlay path, mesh-pull often results in high delay, especially in large network.

Push mechanism has hence been proposed to reduce scheduling delay, where parents push their packets to a child based on a *predetermined* schedule (without explicit pull from the child). The schedule only needs to be changed when there is significant change in network conditions (e.g., in terms of failures, bandwidth, or loss). In a traditional tree-push approach, the video stream is divided into independent substreams of similar bandwidth (achieved simply by, for example, packet multiplexing), and each substream is pushed along different "structured" overlay trees [4]–[6]. A peer can then reassemble the whole stream after receiving all the substreams from different trees.

In this paper, we study the use of network coding combined with substream pushing for P2P live streaming. Given an arbitrary mesh overlay, we focus on efficient scheduling to minimize scheduling delay of a child. Our scheme, substream pushing and network coding (SPANC), is predominantly push in nature and achieves optimal delay in polynomial time. SPANC is independent of mesh construction algorithms (i.e., search algorithms for parents) and can be used on any overlay. Given a set of parents, the child computes an optimal push schedule for its

Manuscript received August 28, 2009; revised December 24, 2009 and May 16, 2010; accepted June 07, 2010. Date of publication June 21, 2010; date of current version October 15, 2010. This work was supported in part by the General Research Fund from the Research Grant Council of the Hong Kong Special Administrative Region, China (611209) and by the Cisco University Research Program Fund, a corporate advised fund of the Silicon Valley Community Foundation under Grant SVCF08/09.EG01&GMGS08/09.EG05. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Thinh Nguyen.

K.-H. K. Chan and S.-H. G. Chan are with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Kowloon, Hong Kong (e-mail: chankh@cse.ust.hk; gchan@cse.ust.hk).

A. C. Begen is with the Video and Content Platforms, Research and Advanced Development, Cisco Systems, Inc., San Jose, CA 95134 USA (e-mail: abegen@cisco.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2010.2053524

[1]Online. Available: http://www.pplive.com.

[2]Online. Available: http://www.ppstream.com.

[3]Online. Available: http://www.sopcast.com.

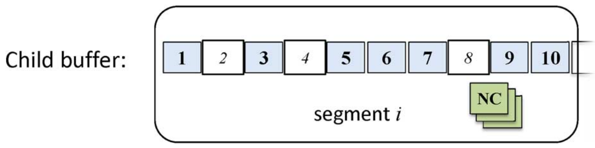


Fig. 1. Child recovers a segment using NC packets.

parents. As long as the network conditions do not change considerably, the schedule does not need to be adjusted, thereof achieving push-based performance. In SPANC, parents push network-coded (NC) packets with the substream packets to the child. Any lost substream packets can be recovered by the piggybacked NC packets, leading to efficient and fast recovery.

Given the available bandwidth from each parent to the child, there are indeed multiple feasible substream assignments so that each substream is assigned to only one parent. However, different assignments result in different delays at the child. For example, closer parents with more updated contents should push more substreams to reduce the delay at the child. Given the available contents and uplink bandwidths of parents, there is hence an optimal assignment to achieve the lowest delay. This is the so-called substream assignment (SA) problem in this paper.

In SPANC, video packets are sequenced and consecutive packets are grouped into segments. For each segment, the parents generate some NC packets and send them to the child in parallel with the source packets (this saves the turnaround time for packet recovery at some expense of overhead). The child, upon receiving the source[4] and NC packets, tries to recover the whole segment. Refer to Fig. 1. Suppose a segment is composed of ten packets and a child lost three packets in a segment from substream push. With three NC packets, the three lost packets from substream push can be recovered without the need of retransmission, as the substream packets and the NC packets are pushed by the parents at the same time. Given the heterogeneous available contents, uplink bandwidths and loss rates of the parents, we study how to optimally assign NC packets to each parent to minimize the recovery delay. This is the so-called fast recovery with network coding (FRNC) problem in this paper.

There has been relatively little work on the analytic formulation and optimization of scheduling problem. We address such issues in this work by means of the following approaches.

1) *Formulation and optimized solution of the SA problem*: We formulate the SA problem as an optimization problem to minimize the overall delay of the packets. We show that, by transforming to a max-weighted bipartite matching (MWBM) problem, the SA problem can be solved exactly and efficiently in polynomial time.
2) *Formulation and optimized solution of the FRNC problem*: We first estimate the number of required NC packets for recovery. The FRNC problem is to minimize the worst-case delay of the NC packets, so that the recovery delay at the child is minimized. We formulate the problem and present an optimized and efficient solution in polynomial time with dynamic programming.
3) *Simulation studies and comparisons*: With the optimal solutions of SA and FRNC problems, we present a simple and distributed scheduling algorithm called SPANC which achieves low scheduling delay. Using simulations, we study SPANC and compare it with current approaches based on pull, network coding and hybrid pull–push. Our results show that SPANC indeed achieves substantially lower playback delay with little bandwidth cost. SPANC can also perform well with peer churn.

[4]"Source packet" refers to the original packet in the video stream.

The remainder of this paper is organized as follows. We first overview related work in Section II. Then, we present in Section III the system design of SPANC. We present the formulations and solutions of the SA and FRNC problems in Sections IV and V, respectively. In Section VI, we discuss illustrative simulation results. We conclude in Section VII.

II. Related Work

Traditional approaches for P2P live streaming can be classified into two categories: tree-push and mesh-pull. In tree-push approach, peers organize themselves into overlay multicast trees [6]. The video is usually decomposed into substreams, which are pushed along different trees. A snowball scheduling which focuses on minimizing delay is proposed and analytically studied in [7]. Multiple description coding (MDC) is studied in [5], [8] to improve bandwidth utilization. Clearly, the tree-push approach achieves low delays. It requires little or no packet scheduling complexity. However, it needs to maintain a structured overlay among peers, which requires much effort and is challenging with peer churns. Therefore, SPANC focuses on mesh overlay, which can achieve resilience to peer dynamics and is easier to implement [9].

In mesh-pull, a peer pulls on the per-packet basis based on periodic exchange of buffermap. There has been much work on mesh construction [10]–[13]. Bandwidth allocation in P2P systems has also been discussed in [14]. Our work is orthogonal to them and focuses on the *scheduling* problem given a mesh; SPANC can apply on their work to achieve better performance. The scheduling problem in mesh-pull has been richly studied. The global scheduling problem has been analytically studied in [15] by modeling it as a minimum-cost network flow problem. The local scheduling problem given a set of pull parents and their buffermap has been found to be NP-hard in [2]. Various pull scheduling approaches have been proposed in [2] and [16]–[18]. Though mesh-pull has been demonstrated to provide simplicity, robustness and high bandwidth utilization, the pull mechanism leads to long delay [3]. In SPANC, parents actively push substreams and NC packets to a child based on a predetermined schedule, which is shown to achieve lower delay.

Recently, a hybrid pull–push approach has been proposed to incorporate the benefits of low-delay pushing and high-bandwidth utilization of pulling [3], [19], [20]. In this approach, packets are divided into substreams and are pushed to peers. The missing packets are then recovered using traditional pulling. In [3], the substream scheduling is done greedily, which does not achieve delay optimality. A substream trading scheme is proposed in [21] that aims to provide differentiated video quality with the aid of layered coding. A max-flow model has been used to address the substream assignment problem [22]. However, the pulling of the lost packets still incurs high overall playback delay. In SPANC, we further reduce the delay by the use of network coding and pushing NC packets *optimally*.

A preliminary version of this work has been reported in [23]. As compared to it, the current work further reduces the scheduling delay through the use of NC. The new approach (SPANC) is a push-based scheme without the need of the previous pull mechanism. Moreover, we optimize the NC packet schedule by an efficient algorithm based on dynamic programming.

Network coding has been proposed for multicast network and has been shown to improve the throughput in P2P network [24]–[26]. The benefits of using network coding in P2P live streaming has also been analytically studied in [27]. Lava is a scheme which uses network coding with traditional pulling, while R^2 uses a randomized push algorithm [28], [29]. They have not focused on reducing the source-to-peer delay of a P2P network. A multiple-segment playback buffer is required in R^2 to accommodate randomized selection, which leads to some unnecessary source-to-peer delay. On the other hand, SPANC is based on estimation of NC packets and does not require large buffer for randomized selection. It also addresses the scheduling problem by optimizing push schedule to achieve minimal delay.

III. System Design of SPANC

We describe the system design of SPANC in this section. We start our discussion by presenting our model in Section III-A. The NC recovery is discussed in Section III-B.

A. Model

Let R bits/s be the streaming rate of the video stream. The stream is composed of packets of constant size L bits that are uniquely identified by sequence numbers. The packets are interleaved into N substreams of bitrate R/N bits/s, i.e., substream j contains all packets whose sequence numbers and j are congruent modulo N. A group of every S consecutive packets is called a *segment*, and hence a segment period is given by

$$T = \frac{SL}{R}. \tag{1}$$

We consider a simple multithread design of the system, where each child is served by a new thread of the parent. Each thread is allocated a certain bandwidth (which can be implemented simply by certain bit transmission quota per some time interval). As compared with the round-robin scheduling, such multithread system has the strength that a parent–child connection of low end-to-end bandwidth would not starve out other children in the service queue (the so-called head-of-line blocking). Using this design, without loss of generality, we can focus on a certain *child* and its *parents*, as illustrated in Fig. 2. The child has a set of parents denoted by $\mathcal{P}$. For parent i in $\mathcal{P}$, it allocates a certain uplink bandwidth B_i bits/s to the child for substreams and NC packets.

We consider that control messages are sent through reliable channel (using TCP) while data packets are subject to losses (due to deadline missing or using UDP). For any parent $i \in \mathcal{P}$, the transmission loss rate is ϵ_i. We assume packets received by parents are generally sequential within a segment, as parents are also receiving their substream and NC packets for streaming. Each parent in $\mathcal{P}$ then notifies the child the latest packet of each

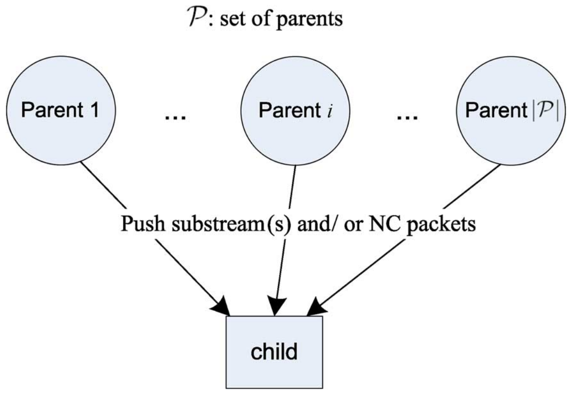

$\mathcal{P}$: set of parents
Parent 1
...
Parent i
...
Parent $|\mathcal{P}|$
Push substream(s) and/ or NC packets
child

Fig. 2. Parent–child relationship.

TABLE I
NOMENCLATURE

Notation	Definition
$\mathcal{P}$	Set of all parents
N	Number of substreams
S	Number of packets in a segment
R	Streaming rate (bits/s)
L	Packet size (bit)
T	Segment period (s) ($T = SL/R$)
$\widehat{l}_j$	The sequence number of the last packet for substream j in the coming segment at the child
$l_{i,j}$	The latest packet sequence number at parent i for substream j
ϵ_i	Loss rate of data packets between parent i and the child
B_i	Total uplink bandwidth that parent i allocated to the child (bits/s)
m_i	Maximum number of substreams that parent i allocated to the child
M	$M = \sum_{i \in \mathcal{P}} m_i$
r_i	Uplink bandwidth for NC packets of parent i (bits/s)
τ_i	The time when the most updated vector $[l_{i,j}]$ is received at the child from parent i
$C(i,j)$	Delay cost of assigning parent i to substream j
X	Total number of requested NC packets per segment
$D_i(x)$	Delay cost of receiving x NC packets from parent i
t_i	Segment formation time for parent i (s)

substream in its buffer, denoted by a vector $[l_{i,1}, l_{i,2}, \ldots, l_{i,N}]$, where $l_{i,j}$ is the latest packet sequence number of substream j at parent $i \in \mathcal{P}$. Let τ_i be the time when the vector $[l_{i,j}]$ is received at the child from parent i. Some important notations are summarized in Table I.

When a child first joins the system, it connects to its parents and asks for their buffer information (i.e., the $[l_{i,j}]$'s). Upon the receipt of the vector $[l_{i,j}]$ from all its parents, the child computes a new push schedule for upcoming packets. The child recomputes a new schedule when it experiences changes in network conditions, such as the departure of a parent, or a change in the uplink bandwidth (e.g. by 10%), etc. In the worst case of peer churn, the child using our algorithm would adapt the schedule for each segment. The child computes the schedule in two phases. The child first computes a substream push schedule for its parents according to the SA algorithm (presented in Section IV). With the SA solution, the child estimates the number of NC packets required for upcoming segments. Then it computes the number of NC packets to be pushed by each parent according to the FRNC algorithm (presented in Section V). The child then issues its requests for substreams and NC packets to corresponding parents. The same schedule will be used until a new schedule is computed or the child is dead.

B. NC Recovery

NC packets are used to recover lost packets in a segment of the child. The NC packets are computed at a parent by generating a linear combination of the source packets in a segment, where coding coefficients are arbitrarily chosen (similar to [28], [29]). The child, upon receiving N linearly independent packets (including both source and NC packets), can recover the whole segment by traditional NC decoding schemes (such as Gaussian elimination).

For a segment to be fully recovered, the total number of received NC packets and source packets for the segment must be no less than S. Let X be the total number of requested NC packets per segment by the child. Note that, as the child does not continuously feed back to parents for overhead consideration, X must be sufficiently high so that it can mitigate statistical fluctuation of packet loss while low enough not to incur too much bandwidth cost. This estimation is performed as follows.

For every segment, the child records X', the number of NC packets necessary for recovery, which is simply the number of source packets lost in the segment. To smooth out statistical fluctuation, the child also keeps a "smoothed" version of X', denoted as $\overline{X}$: with a new sample X', the child updates $\overline{X}$ as $\overline{X} \leftarrow (1-\alpha_1)X' + \alpha_1\overline{X}$ and the variance as $\sigma_{X'} \leftarrow (1-\alpha_2)|\overline{X} - X'| + \alpha_2\sigma_{X'}$, where $0 \le \alpha_1, \alpha_2 \le 1$ are some smoothing factors.

In order to recover error with high probability, the child sets a "cushion" to accommodate the statistical fluctuations. X is hence set to be

$$X \leftarrow \overline{X} + \beta\sigma_{X'} \tag{2}$$

where β is some multiplier greater than zero. In our scheme, X is updated by the child only when the estimation differs by more than a certain threshold (i.e., 2 in our simulations). Parents would not resend its packets to a downstream child. If the child fails to carry out NC recovery continuously, it would recompute a new schedule based on (2).

IV. SA Problem

We first formulate the SA problem as a delay optimization problem (Section IV-A), and then present an exact polynomial-time solution (Section IV-B). As mentioned before, given our operation model, without loss of generality we can focus on a child with multiple parents as shown in Fig. 2.

A. Problem Formulation

Let m_i be the maximum number of substreams that can be pushed to the child for parent $i \in \mathcal{P}$ ($m_i \in \mathbb{N}$) given by

$$m_i = \left\lfloor B_i \cdot \frac{N}{R} \right\rfloor. \tag{3}$$

Given the latest packet vector $[l_{i,j}]$ from parent i (received at some time τ_i), and the maximum number of substreams allocated to the child m_i, the SA problem is to find a substream assignment achieving the minimum total source-to-child delay. Note that not all parents in $\mathcal{P}$ have to be assigned.

Let $C(i,j)$ be a function denoting the (delay) cost of packets if parent i is assigned to substream j, where a larger $C(i,j)$ means higher source-to-child delay. Note that SA problem does not assume any form of $C(i,j)$.

The goal of the SA problem is to find an assignment of substreams to parents in $\mathcal{P}$, i.e., find $A : \{1,2,\cdots,N\} \to \mathcal{P}$, such that substream j is assigned to parent $A(j) \in \mathcal{P}$ for $j \in \{1,2,\cdots,N\}$, so as to

$$\text{Minimize} \sum_{j=1}^{N} C\left(A(j), j\right) \tag{4}$$

subject to the bandwidth constraint

$$\left|A^{-1}(i)\right| \le m_i, \quad \forall i \in \mathcal{P} \tag{5}$$

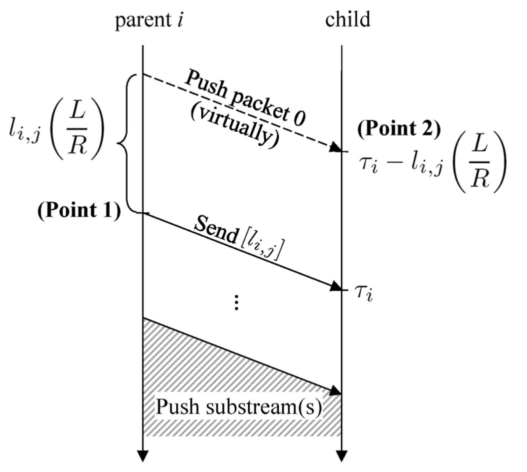
where $A^{-1}(i)$ is the set of assigned substream(s) to parent i in A, and $|A^{-1}(i)|$ is the size of the set.

One may consider a delay cost as follows. We illustrate in Fig. 3 the timeline to send substreams from parent i to the child, where Point 1 is the time the parent i sends packet sequence numbers of $[l_{i,j}]$ to the child. Since the packet sequence numbers are received at time τ_i, $\tau_i - l_{i,j}(L/R)$ (Point 2) represents the arrival time of packet 0 if it *were* pushed to the child. This can be interpreted as the "virtual" starting time for substream j at the child from parent i, which can also be viewed as a reference for the "timeliness" of the substream. Clearly, the earlier this virtual arrival time is, the lower is the delay. Therefore, we may consider the delay cost, $C(i,j)$, as

$$C(i,j) = \tau_i - l_{i,j}\left(\frac{L}{R}\right). \tag{6}$$

B. Exact Solution Based on Max-Weighted Bipartite Matching

The SA problem can be solved exactly in polynomial time, as it can be transformed to an equivalent MWBM problem. This is shown as follows. We consider a complete bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with bipartition $\mathcal{V} = \mathcal{X} \cup \mathcal{Y}$, as illustrated in Fig. 4. $\mathcal{X} = \{x_i^{(k)} : \text{parent } i \in \mathcal{P}; k = 1,\cdots,m_i\}$. We have $|\mathcal{X}| = M$, where $M = \sum_{i\in\mathcal{P}} m_i$. Each $x_i^{(k)} \in \mathcal{X}$ represents a possible assignment slot for substream. $\mathcal{Y} = \{y_j : j = 1,2,\ldots,N\}$, where $y_j \in \mathcal{Y}$ represents substream j, therefore $|\mathcal{Y}| = N$. The objective of SA problem in (4) is to minimize $\sum_{j=1}^{N} C(A(j),j)$,



parent i
child
$l_{i,j}\left(\frac{L}{R}\right)$
Push packet 0 (virtually)
(Point 2)
$\tau_i - l_{i,j}\left(\frac{L}{R}\right)$
(Point 1)
Send $[l_{i,j}]$
τ_i
Push substream(s)

Fig. 3. Timeline for virtual arrival between parent i and the child.

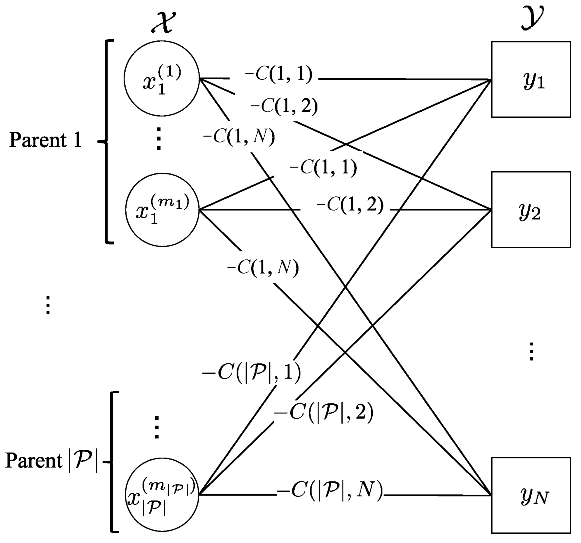


Fig. 4. Bipartite graph for SA solution.

which is equivalent to maximize $-\sum_{j=1}^{N} C(A(j),j)$. Therefore, the weight of each edge $(x_i^{(k)}, y_j)$ is set to be $-C(i,j)$.

Clearly, the MWBM solution on $\mathcal{G}$ is the optimal solution of the SA problem, whose solution is to assign substream j to parent i for every matching $(x_i^{(k)}, y_j)$ in the MWBM problem. The MWBM problem can be solved efficiently by the Hungarian Algorithm [30]. Therefore, the time complexity of SA problem follows with the MWBM problem, which is $O(N^3 + M^3)$.

V. FRNC Problem

In this section, we describe the formulation and solution of the FRNC problem, which minimizes the recovery time by assigning NC packets to parents.

A. Problem Formulation

Let r_i bits/s be the uplink bandwidth of parent i for sending NC packets to the child. Given the SA solution A, the residual uplink bandwidth of parent i is given by

$$r_i = B_i - \left|A^{-1}(i)\right| \cdot \left(\frac{R}{N}\right). \tag{7}$$

Let x_i be the number of NC packets received from parent $i \in \mathcal{P}$ for each segment ($x_i \in \mathbb{N}$). In other words, if ϵ_i is the loss rate of parent i, parent i expects to send $\lceil x_i/(1-\epsilon_i)\rceil$ NC packets. To recover a segment, we hence require

$$\sum_{i\in\mathcal{P}} x_i \geq X. \tag{8}$$

Note that, for stability and continuity, all NC packets of a segment have to be sent within the segment period T, which means

$$\begin{aligned}\left\lceil \frac{x_i}{1-\epsilon_i}\right\rceil &\leq \left(\frac{r_i}{L}\right)\cdot T\\ x_i &\leq \left(\frac{r_i}{L}\right)\cdot T\cdot(1-\epsilon_i)\end{aligned} \tag{9}$$

where T is given in (1).

A parent generates and sends NC packets of a segment after it has received most of the packets in the segment.[5] This incurs a waiting time called *segment formation time* labeled as t_i for parent i. To estimate t_i, the child first computes $\widehat{l_j}$, the sequence number of the last packets for substream j in its upcoming (yet-to-be received) segment. Clearly, if $\widehat{l_j}$ is no larger than $l_{i,j}$, parent i has already had all the packets for substream j in the coming segment of the child; therefore no waiting time is needed. On the other hand, if $\widehat{l_j}$ is larger than $l_{i,j}$, parent i is expected to wait for $(\widehat{l_j} - l_{i,j})/N$ packets (NC or source packets) to arrive.[6] Given the above, the child estimates the *segment formation time* t_i for parent i as

$$t_i = \frac{L}{NR}\left(\sum_{j=1}^{N}(\widehat{l_j} - l_{i,j})^+\right) \tag{10}$$

where $x^+ = \max(0,x)$.

Let $D_i(x_i)$ be the *delay (cost)* of receiving x_i NC packets from parent $i \in \mathcal{P}$. It represents the delay introduced to the child as shown in Fig. 5, consisting of the time when $l_{i,j}$ is received at τ_i, segment formation time and transmission delay according to

$$D_i(x_i) = \begin{cases} 0, & \text{if } x_i = 0\\ \tau_i + t_i + \frac{1}{r_i}\left\lceil\frac{x_i}{1-\epsilon_i}\right\rceil, & \text{if } x_i > 0.\end{cases} \tag{11}$$

[5]A parent forms NC packets if the segment is fully received, or with period S, whichever is earlier. If the segment is incomplete, the parent still forms NC packets based on the received packets in the segment.

[6]Note that $\widehat{l_j} - l_{i,j}$ must be an integral multiple of N as they refer to the sequence number of packets in the same substream.

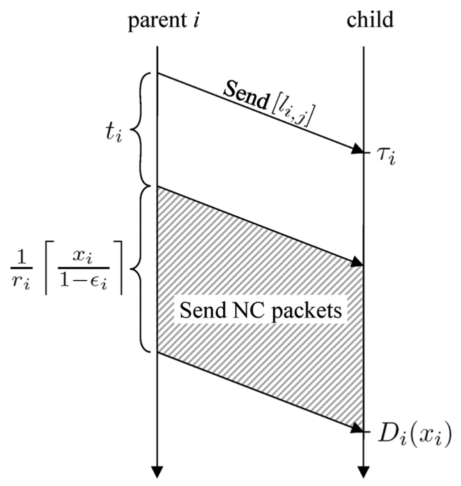


Fig. 5. Timeline for delay components for FRNC between parent i and the child.

The FRNC problem is to find an assignment which minimizes the worst-case delay (cost) over all parents, i.e., to find x_i's so as to

$$\text{Minimize} \max_i D_i(x_i), \tag{12}$$

$\forall i \in \mathcal{P}$, subject to (8) and (9).

B. Exact Solution Using Dynamic Programming

The FRNC problem can be solved efficiently by dynamic programming as follows. Let $\Phi[u,v]$ be the minimum delay cost of receiving $v \le X$ NC packets ($v \in \mathbb{N}$) from parents $\{1\} \cup \{2\} \cup \ldots \cup \{u\} \subseteq \mathcal{P}$, i.e.,

$$\Phi[u,v] = \min \left\{ \max_i \left(D_i(x_i)\right), \forall i = \{1, 2, \cdots, u\} \right\} \tag{13}$$

subject to (9) and

$$\sum_{i \in [1 \ldots u]} x_i \ge v. \tag{14}$$

Then, the solution to the FRNC problem is equivalent to $\Phi[|\mathcal{P}|, X]$.

For the boundary case of $u = 1$, if (9) is satisfied, then $x_1 = v$ packets would be received from parent 1 with delay cost $\Phi[1,v] = D_1(v)$; otherwise, it is impossible to satisfy both (9) and (14) and we have $\Phi[1,v] = \infty$. In other words

$$\Phi[1,v] = \begin{cases} D_1(v), & \text{for } v \le \left(\frac{r_1}{L}\right) \cdot T \cdot (1-\epsilon_1) \\ \infty, & \text{otherwise.} \end{cases} \tag{15}$$

We next derive the recurrence of $\Phi[u,v]$ for $1 < u \le |\mathcal{P}|$. Without loss of generality, we first decide x_u. From (9) and (14), the possible range of x_u are $0 \le x_u \le \min(v, (r_u/L) \cdot T \cdot (1 - \epsilon_u))$, with delay cost $D_u(x_u)$. After that, the remaining $v - x_u$ packets are to be received from parents 1 to $u-1$, which reduces to a subproblem of finding $\Phi[u-1, v-x_u]$. In other words, $\Phi[u,v]$ is given by the minimum of $\max(D_u(x_u), \Phi[u-1, v-x_u])$ over all possible x_u's, i.e.,

$$\Phi[u,v] = \min_{x_u} \max\left(D_u(x_u), \Phi[u-1, v-x_u]\right) \tag{16}$$

for $1 < u \le |\mathcal{P}|$, where

$$0 \le x_u \le \min\left(v, \left(\frac{r_u}{L}\right) \cdot T \cdot (1-\epsilon_u)\right). \tag{17}$$

With the boundary condition of (15) and the recurrence relation of (16), the FRNC problem can hence be solved by *dynamic programming* by building a 2-D array of $\Phi[u,v]$ as presented in Algorithm 1. Clearly, the time complexity is $O(|\mathcal{P}|X^2)$.

Algorithm 1 Pseudo code of the FRNC dynamic programming solution.

```
/* Initialization */
for $1 \le u \le |\mathcal{P}|, 0 \le v \le X$ do
    $\Phi[u,v] \leftarrow -\infty$
end for
/* Boundary Case */
for $0 \le v \le X$ do
    if $v \le (r_1/L) \cdot T \cdot (1-\epsilon_1)$ then
        $\Phi[1,v] \leftarrow D_1(v)$
    else
        $\Phi[1,v] \leftarrow \infty$
    end if
end for
/* Recurrence */
for $u = 2$ to $|\mathcal{P}|$ do
    for $v = 0$ to $X$ do
        for $0 \le k \le \min(v, (r_u/L) \cdot T \cdot (1-\epsilon_u))$ do
            if $\Phi[u,v] > \max(\Phi[u-1, v-k], D_u(k))$ then
                $\Phi[u,v] \leftarrow \max(\Phi[u-1, v-k], D_u(k))$
                $backtrack[u,v] \leftarrow k$
            end if
        end for
    end for
end for
/* Backtracking */
$v \leftarrow X$
for $u = |\mathcal{P}|$ downto 1 do
    $x_u \leftarrow backtrack[u,v]$
    $v \leftarrow v - backtrack[u,v]$
end for
return $x_i$, for $i = 1, 2, \cdots, |\mathcal{P}|$
```

VI. Illustrative Simulation Results

We have conducted simulation study on SPANC. We present the simulation environment and metrics in Section VI-A and illustrative results in Section VI-B.

A. Simulation Environment and Metrics

We compare SPANC with the following approaches.

- *Pull*: The pull scheme adopts a rarest-first algorithm (used in, for example, CoolStreaming [2]). After periodic buffermap exchange, pull requests are issued at a child to request packets from parents. If a packet is lost, the same packet is pulled again (for at most two times in our simulation).
- *Hybrid*: A hybrid pull–push scheme reduces the delay by pushing substreams. Similar to the one used in [3], the substream assignment algorithm is done in a greedy fashion. If a packet is lost, the same packet is pulled by the child.
- *Lava*: In order to compare SPANC with a pure network coding approach, we compare our scheme with Lava [28]. Since SPANC and Lava have not focused on reducing the source-to-peer delay, Lava can provide qualitatively similar results for our comparison. In Lava, a child pulls NC packets of a segment from its parents. The earliest segment is given higher priority so as to ensure streaming continuity.

The pull and hybrid schemes repull packets after a certain timeout if a pulled packet is lost. We use a timeout value of 1 s for pull and hybrid, which can optimize their performances, in terms of high lost detection and low delay.

We have implemented an event-driven simulator in C++. We generate Internet-like two-level topologies using BRITE consisting of many (5000) routers with default parameters [31]. BRITE also provides the underlying link latency in milliseconds. Peers are randomly attached to different underlying routers. We assume the data packets are sent through UDP. The Internet traffic is rather complex and diversified [32]. For simplicity, we model the transmission loss rate for data packets between two peers as uniformly distributed from 2% to 10%, which are quite typical in the Internet environment [33]. The uplink bandwidth of the server is 4 Mbps, while the uplink bandwidth of a normal peer is uniformly distributed between 512 kbps and 1 Mbps. Unless otherwise stated, we use the following baseline parameters: $R = 512$ kbps, $L = 8$ kb, $N = 8$, $T = 2$ s, $\alpha_1 = \alpha_2 = 0.875$, $\beta = 3.0$ and the number of peers $= 500$.

Our focus is on packet scheduling given a mesh topology. In our simulation, we have used a random mesh topology for illustration purpose; note that SPANC can apply on any overlay construction schemes such as [10]–[13]. (Our results are qualitatively the same with other topology.) Every peer sequentially joins the system and is randomly assigned to a default number (i.e., 10) of peers as parents. In SPANC, every parent randomly allocates some uplink bandwidth $B_i \sim U[0, R/2]$ for substream pushing and NC packets.

We use the following performance metrics in our evaluation.

- *Residual Loss Rate*: It is the percentage of source packets that cannot be successfully received after loss recovery at a peer.

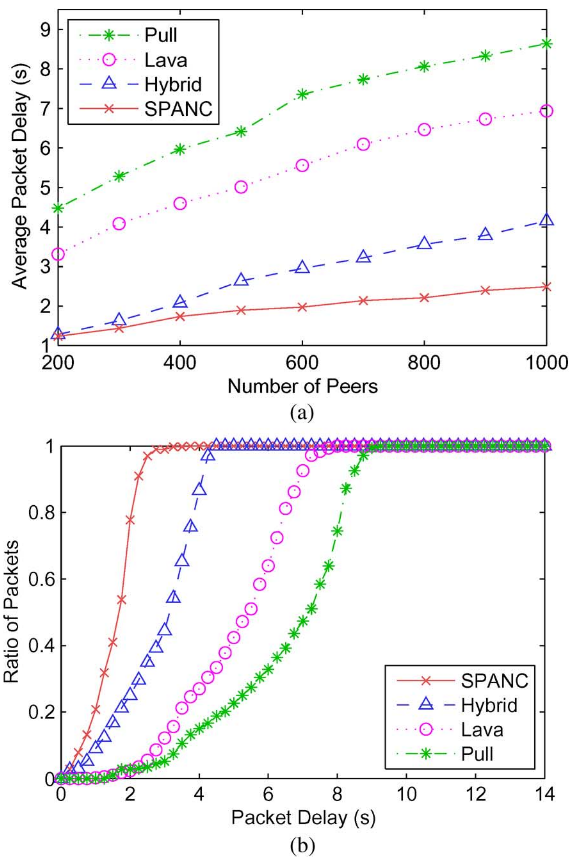


Fig. 6. Average packet delay. (a) Average packet delay versus number of peers. (b) Packet delay distribution.

- *Packet Delay*: It refers to the source-to-peer delay of each packet for a peer, after loss recovery with NC packets. We are interested in both its distribution and average.
- *Playback Delay*: While the packet delay shows the general behavior of packet arrival, the video playback of a peer is determined by the highest packet delay. We call the maximum packet delay of all packets as *playback delay* of a peer (i.e., the packet delay of the latest received packet). We are also interested in both its distribution and average.
- *Bandwidth Dilation*: It is defined as the percentage of extra upload bandwidth consumed over the raw streaming rate R. It measures the bandwidth overhead due to packet retransmission, NC packet transmission, or control messages.

B. Illustrative Results

We compare in Fig. 6(a) the average packet delay of different schemes versus number of peers. Generally, the average packet delay increases with number of peers, as the overlay size of the network increases. The average packet delay of pull is relatively high, due to the buffermap exchange and the pull requests roundtrip. Lava reduces packet delay because of the use of network coding. Hybrid is better than pull and Lava, because some of the packets are pushed instead of pulled. SPANC achieves substantially lower delay. This is due to its pure push nature and the schedule optimality which pushes packets in a timely manner to peers.

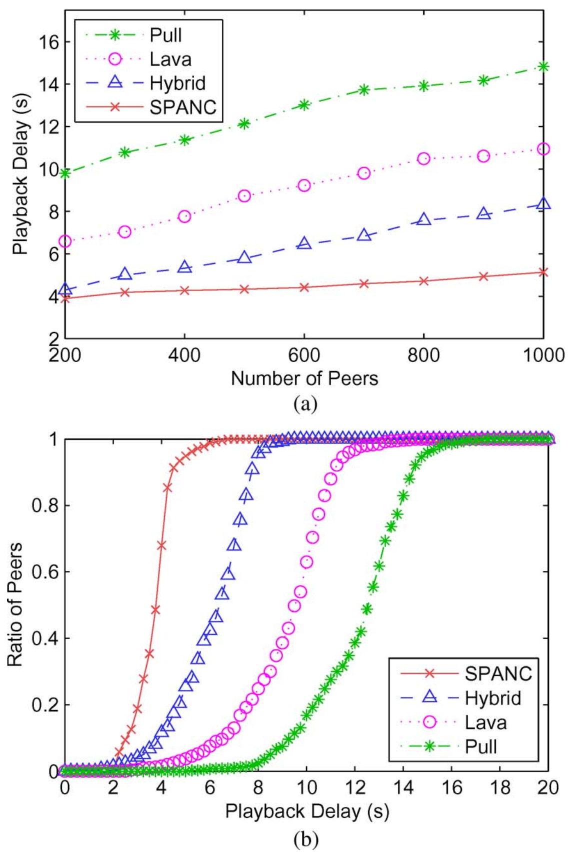


Fig. 7. Playback delay. (a) Playback delay versus number of peers. (b) Playback delay distribution.

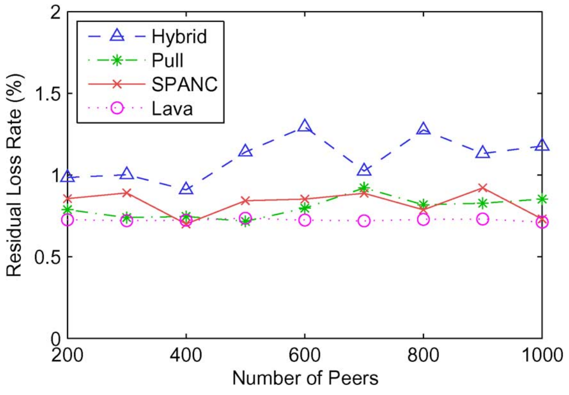


Fig. 8. Residual loss rate versus number of peers.

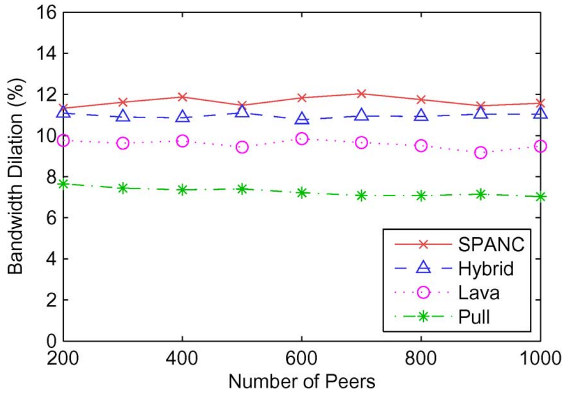


Fig. 9. Bandwidth dilation versus number of peers.

Fig. 6(b) shows the CDF of the packet delay of various schemes for the baseline case. We observe that pull-based schemes (i.e. pull and Lava) have higher packet delays and delay variation. This is because the mechanism of pull requests are initiated at child side, which greatly delays a packet in parent buffer after it is received. Similar situations also occur for the hybrid scheme, where some packets are pulled. In SPANC, the packet delays are rather concentrated and low. This is again due to the pure push nature and the schedule optimality of SPANC.

We compare in Fig. 7(a) the playback delay of the three schemes versus number of peers. Similar trend is observed as with the packet delay. SPANC achieves the lowest playback delay. (It reduces the playback delay substantially by 65%, 55%, and 40% as compared with pull, Lava, and hybrid, respectively.) The playback delay is usually resulted from the recovered lost packets. In pull and hybrid, specific lost packets are pulled after timeout. This passive recovery method greatly affects the playback delay of a peer. In SPANC, lost packets are recovered actively by pushing NC packets in advance of the lost packets being discovered. This aggressive recovery scheme can greatly reduce the messaging delay and timeout compared with the other schemes. Therefore, the playback delay of SPANC outperforms the other schemes.

The CDF of the playback delay of various schemes is shown in Fig. 7(b). The playback delay of peers in pull, Lava, and hybrid clearly show more variation as compared with peers in SPANC, which can be explained by the pure push nature of SPANC.

Fig. 8 compares the residual loss rate against number of peers for different schemes. The residual loss rate for all schemes are low (around 1%). The low residual loss rate achieved by SPANC confirms that our NC recovery is effective to recover packet loss.

Fig. 9 compares the bandwidth dilation of various schemes versus number of peers. The bandwidth dilation of all schemes are generally insensitive to the number of peers in the network. The pull scheme has the lowest dilation because every packet is pulled explicitly so data packets are seldom redundant. The other schemes have similar dilation. In Lava, the redundant packets are due to linearly dependent NC packets received at the child. In hybrid, redundant packets are due to asynchrony between push and pull, due to propagation delay of control messages [3]. In SPANC, the surplus in NC estimation allows good NC recovery, but also introduces some redundant NC packets. From this and previous figures, we see that SPANC achieves its better delay performance with a slightly higher bandwidth dilation.

We investigate the impact of segment period T on the delay performance in Fig. 10(a). In general, the playback and packet delay both increase with the segment period T, due to the segment-based NC recovery. When T is small (such as 1 s), both delays increase slowly with T. However, both delays increase rapidly with T when T is too large (such as 4 s).

We also study the impact of segment period T on the residual loss rate in Fig. 10(b). The residual loss rate generally decreases

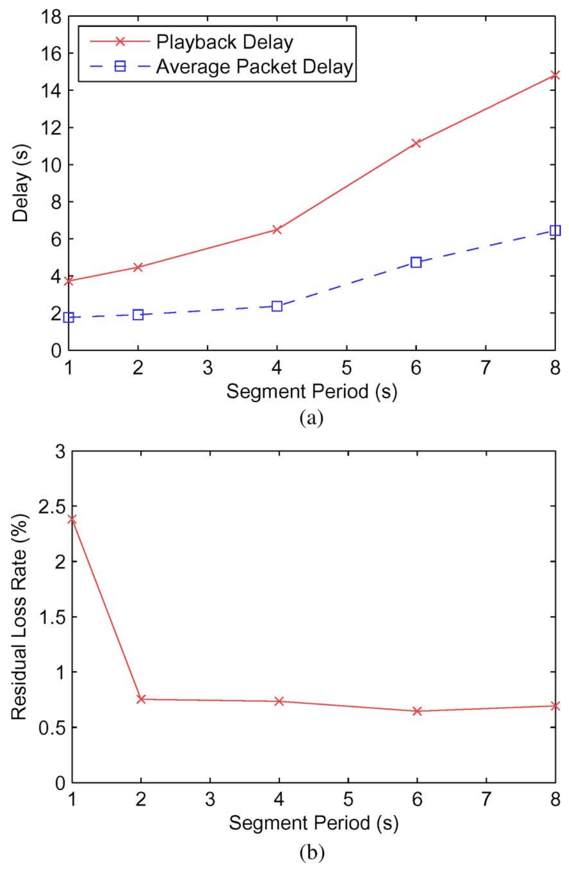


Fig. 10. Impact of segment period T. (a) Average packet delay and playback delay versus segment period T. (b) Residual loss rate versus segment period T.

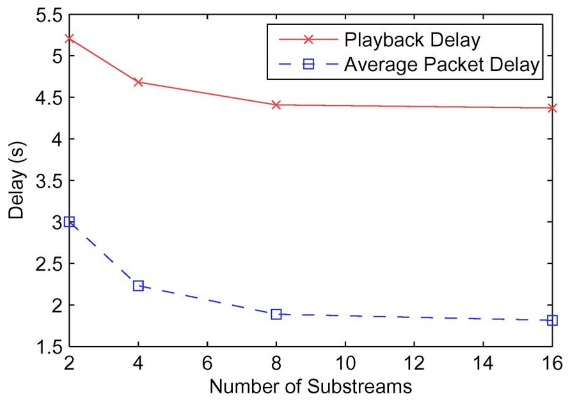


Fig. 11. Average packet delay and playback delay versus number of substreams N.

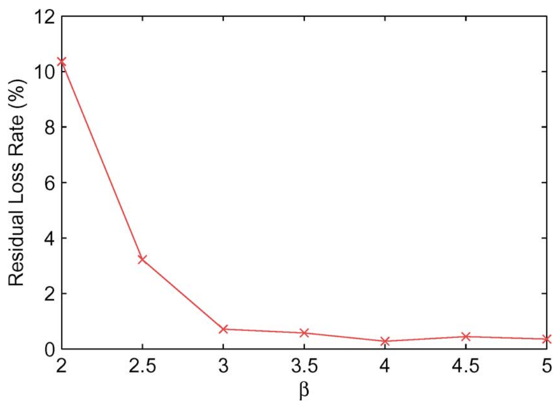


Fig. 12. Residual loss rate versus different values of β.

when T increases, as larger segments can better accommodate the randomness in NC estimation. When T is too small (such as 1 second), the segment is too small to accommodate the random or statistical fluctuation in the number of NC packets received, leading to high residual loss rate. On the other hand, a higher T does not reduce the residual loss rate much. Therefore a good value of T is around 2 s, which is the value we use in our simulations.

We investigate the impact of number of substreams N to the delay performance in Fig. 11. With T and L are fixed, we increase N from 2 to 16. We observe that both the average packet delay and playback delay are high when N is too small. This is because the substreams are too massive in unit so that it does not allow flexibility for good substream assignment. When N is large enough (such as 8), low delays are achieved, by the optimal substream assignment of SA solution. As the delay reductions are not significant with more substreams $(N > 8)$, we choose N to be 8 in our simulations to save the FRNC solution complexity.

The surplus of estimating the number of NC packets is controlled by parameter β. We show in Fig. 12 the effects of residual loss rate versus β. When β is small (i.e., 2), the residual loss rate is high (around 10%). This is because the low surplus cannot accommodate the random fluctuations of packet loss, leading to poor NC recovery. Moreover, the high loss would propagates along overlay path. On the other hand, when β is large (i.e., 3), the surplus is able to allow good NC recovery. Given by (2), larger β leads to more relax estimation and higher bandwidth dilation. Therefore, $\beta = 3$ is a good value in our simulations.

We next study the effect of user dynamic on the performance of SPANC. In Fig. 13 we illustrate the system dynamics when one of the streaming parents of a child dies at the tenth second. The number of NC packets requested X, averaged $\overline{X}$, and actually required X' are plotted against time in Fig. 13(a). We also plot the corresponding bitrate after recovery in Fig. 13(b). At the tenth second, a substream is lost due to parent failure, so there is a sudden rise in the number of NC packets required. The NC recovery therefore failed, leading to a drop in the good packets received. The child adapts to this situation quite fast in SPANC by computing a new schedule with the remaining parents so that the subsequent stream is not affected. As is clear from (2), X is usually higher than X'. Therefore the NC recovery is successful in recovering the full stream. This figure shows that SPANC is able to react quite quickly upon a peer churn to maintain high stream continuity.

VII. Conclusion

In this paper, we study scheduling optimization for P2P live streaming. Given a mesh overlay, we study how to minimize scheduling delay of a child. Our scheme, termed SPANC, achieves low delay by pushing video packets in substreams and recovering packet loss using network coding. Given a set of

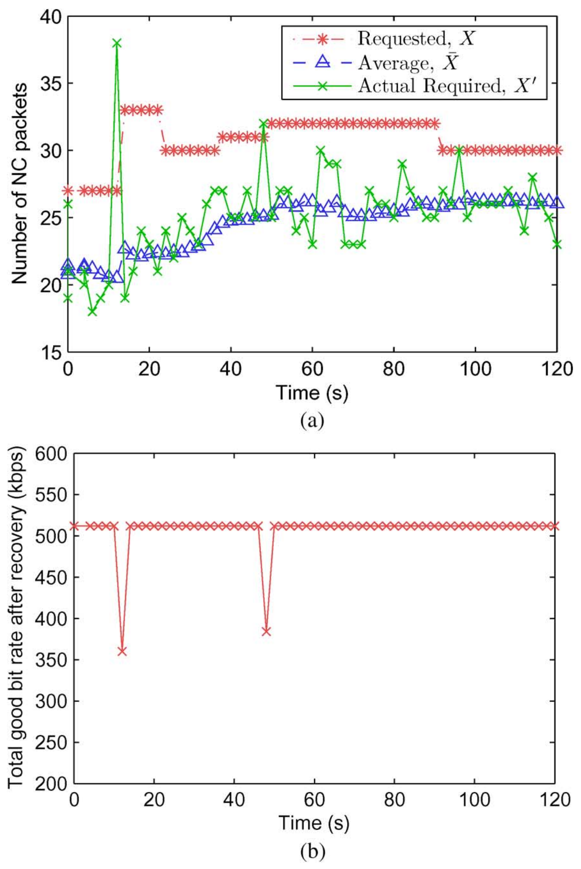


Fig. 13. Scenario where a streaming parent dies at the tenth second. (a) Estimation of X. (b) Total good bit rate after recovery.

parents, the child computes an optimal push schedule for its parents.

We optimize the design of SPANC by addressing two of its subproblems, the SA and FRNC problems . Given heterogeneous contents, delays, and bandwidths of parents, we formulate the SA problem to assign substreams to parents to achieve minimum delay. The SA problem can be optimally and efficiently solved in polynomial time by transforming to a MWBM (max-weighted bipartite matching) problem. In order to assign NC packets to each parent to achieve minimum recovery delay, we also formulate the FRNC problem that assigns NC packets to different parents. The FRNC problem can also be solved exactly and efficiently in polynomial time with dynamic programming.

Simulation results show that SPANC achieves substantially lower delay with comparable playback quality (in terms of residual loss rate) and little cost in bandwidth, as compared with current pull, network coding and hybrid pull–push approaches. SPANC is also reactive to network dynamic and adapts to peer churns to maintain high stream quality.

REFERENCES

[1] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, “A measurement study of a large-scale P2P IPTV system,” *IEEE Trans. Multimedia*, vol. 9, no. 8, pp. 1672–1687, Dec. 2007.

[2] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, “CoolStreaming/DONet: A data-driven overlay network for live media streaming,” in *Proc. IEEE INFOCOM*, Miami, FL, Mar. 2005, pp. 2102–2111.

[3] M. Zhang, Q. Zhang, L. Sun, and S. Yang, “Understanding the power of pull-based streaming protocol: Can we do better?,” *IEEE J. Sel. Areas Commun.*, vol. 25, no. 9, pp. 1678–1694, Dec. 2007.

[4] Y. H. Chu, S. G. Rao, S. Seshan, and H. Zhang, “A case for end system multicast,” *IEEE J. Sel. Areas Commun.*, vol. 20, no. 8, pp. 1456–1471, Oct. 2002.

[5] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, “Distributing streaming media content using cooperative networking,” in *Proc. 12th ACM Int. Workshop NNOSSDAV*, Miami Beach, FL, May 2002, pp. 177–186.

[6] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, “SplitStream: High-bandwidth multicast in cooperative environments,” in *Proc. 19th ACM Symp. Operating Syst. Principles*, Lake George, NY, Oct. 2003, pp. 298–313.

[7] Y. Liu, “On the minimum delay peer-to-peer video streaming: How realtime can it be?,” in *Proc. 15th Int. Conf. Multimedia*, 2007, pp. 127–136.

[8] V. N. Padmanabhan, H. J. Wang, and P. A. Chou, “Resilient peer-to-peer streaming,” in *Proc. 11th IEEE Int. Conf. Network Protocols*, Atlanta, GA, Nov. 2003, pp. 16–27.

[9] N. Magharei, R. Rejaie, and Y. Guo, “Mesh or multiple-tree, a comparative study of live P2P streaming approaches,” in *Proc. IEEE INFOCOM*, Anchorage, AL, May 2007, pp. 1424–1432.

[10] X. Jin, W.-P. K. Yiu, S.-H. Chan, and Y. Wang, “On maximizing tree bandwidth for topology-aware peer-to-peer streaming,” *IEEE Trans. Multimedia*, vol. 9, Special Issue on Content Storage and Delivery in Peer-to-Peer Network, no. 8, pp. 1580–1592, Dec. 2007.

[11] K.-W. Kwong and H. K. Tsang, “Building heterogeneous peer-to-peer networks: Protocol and analysis,” *IEEE/ACM Trans. Netw.*, vol. 16, no. 2, pp. 281–292, Apr. 2008.

[12] D. Ren, Y.-T. H. Li, and S.-H. G. Chan, “On reducing mesh delay for peer-to-peer live streaming,” in *Proc. IEEE INFOCOM*, Phoenix, AZ, Apr. 2008, pp. 1058–1066.

[13] T. Small, B. Li, and B. Liang, “Outreach: Peer-to-peer topology construction towards minimized server bandwidth costs,” *IEEE J. Sel. Areas Commun.*, vol. 25, no. 1, pp. 35–45, Jan. 2007.

[14] M. Meo and F. Milan, “A rational model for service rate allocation in peer-to-peer networks,” in *Proc. IEEE INFOCOM*, Miami, FL, Mar. 2005, pp. 2798–2802.

[15] M. Zhang, Y. Xiong, and Q. Zhang, “On the optimal scheduling for media streaming in data-driven overlay networks,” in *Proc. IEEE GLOBECOM*, New York, Nov. 2006.

[16] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, “Anysee: Peer-to-peer live streaming,” in *Proc. IEEE INFOCOM*, Barcelona, Catalunya, Spain, Apr. 2006, pp. 2411–2420.

[17] N. Magharei and R. Rejaie, “PRIME: Peer-to-peer receiver-driven MEsh-based streaming,” in *Proc. IEEE INFOCOM*, Anchorage, AL, May 2007, pp. 1415–1423.

[18] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr, “Chainsaw: Eliminating trees from overlay multicast,” in *Proc. 4th Int. Workshop on Peer-to-Peer Syst.*, Ithaca, NY, Feb. 2005, pp. 127–140.

[19] B. Li, S. Xie, G. Y. Keung, J. Liu, I. Stoica, H. Zhang, and X. Zhang, “An empirical study of the coolstreaming+ system,” *IEEE J. Sel. Areas Commun.*, vol. 25, no. 9, pp. 1627–1639, Dec. 2007.

[20] G. Zheng, S.-H. G. Chan, X. Luo, and A. C. Begen, “Pattern-push: A low-delay mesh-push scheduling for live peer-to-peer streaming,” in *Proc. IEEE Int. Conf. Multimedia and Expo (ICME)*, New York, 28 Jun.–3 Jul. 2009, pp. 1158–1161.

[21] Z. Liu, Y. Shen, K. W. Ross, S. Panwar, and Y. Wang, “Substream trading: Towards an open P2P live streaming system,” in *Proc. 16th IEEE Int. Conf. Network Protocols*, Oct. 2008, pp. 94–103.

[22] Z. Li, Y. Yu, X. Hei, and D. H.-K. Tsang, “A unified framework for sub-stream scheduling in P2P hybrid streaming systems and how to do better?,” in *Proc. NETWORKING*, 2009, pp. 728–741.

[23] K.-H. K. Chan, S.-H. G. Chan, and A. Begen, “Optimizing substream scheduling for peer-to-peer live streaming,” in *Proc. IEEE Consum. Commun. Netw. Conf.*, Jan. 2010.

[24] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, “Network information flow,” *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216, Jul./Aug. 2000.

[25] C. Gkantsidis and P. Rodriguez, “Network coding for large scale content distribution,” in *Proc. IEEE INFOCOM*, Miami, FL, Mar. 2005, pp. 2235–2245.

[26] Y. W. P. Chou and K. Jain, “Practical network coding,” in *Proc. Allerton Conf. Commun., Control Comput.*, Oct. 2003.

[27] C. Feng and B. Li, "On large-scale peer-to-peer streaming systems with network coding," in *Proc. ACM Multimedia*, Vancouver, BC, Canada, 2008, pp. 269–278.

[28] M. Wang and B. Li, "Network coding in live peer-to-peer streaming," *IEEE Trans. Multimedia*, vol. 9, no. 8, pp. 1554–1567, Dec. 2007.

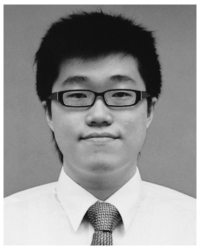
[29] M. Wang and B. Li, "R2: Random push with random network coding in live peer-to-peer streaming," *IEEE J. Sel. Areas Commun.*, vol. 25, no. 9, pp. 1655–1666, Dec. 2007.

[30] C. H. Papadimitriou and K. Steiglitz*, Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice-Hall, 1982.

[31] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An approach to universal topology generation," in *Proc. Int. Workshop MASCOTS*, Cincinnati, OH, Aug. 2001, pp. 346–353.

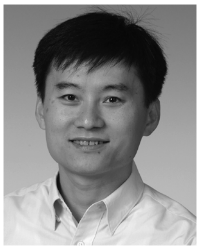
[32] E. Veloso, V. Almeida, W. M. , Jr., A. Bestavros, and S. Jin, "A hierarchical characterization of a live streaming media workload," *IEEE/ACM Trans. Netw.*, vol. 14, no. 1, pp. 133–146, Feb. 2006.

[33] J. Sommers, P. Barford, N. Duffield, and A. Ron, "Improving accuracy in end-to-end packet loss measurement," in *Proc. ACM SIGCOMM*, New York, 2005, pp. 157–168.



K.-H. Kelvin Chan received the B.Eng. degree in computer engineering, and M.Phil. degree in computer science and engineering from the Hong Kong University of Science and Technology (HKUST), Kowloon, in 2007 and 2009, respectively.

His research interests include computer networks, multimedia networking, and peer-to-peer systems.



S.-H. Gary Chan (S'89–M'98–SM'03) received the B.S.E. degree (highest honors) Princeton University, Princeton, NJ, in 1993, and the M.S.E. and Ph.D. degrees from Stanford University, Stanford, CA, in 1994 and 1999, respectively, all in electrical engineering.

He is currently an Associate Professor with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology (HKUST), Hong Kong. He is the Director of Sino Software Research Institute (SSRI), HKUST. He was a Visiting Research Collaborator with Princeton University in 2009, a Visiting Associate Professor with Stanford University (2008–2009), the Director of the Computer Engineering Program at HKUST (2006–2008), Visiting Assistant Professor in networking with the University of California at Davis (1998–1999), and Research Intern with the NEC Research Institute, Princeton, NJ (1992–1993). He was a William and Leila Fellow at Stanford University (1993–94). He was a cochair of the workshop on "Advances in Peer-to-Peer Multimedia Streaming" at the ACM Multimedia Conference (2005). His research interest includes multimedia networking, peer-to-peer technologies and streaming, and wireless communication networks.

Dr. Chan is a member of Tau Beta Pi, Sigma Xi, and Phi Beta Kappa. He is an associate editor of the IEEE TRANSACTIONS ON MULTIMEDIA and a Vice-Chair of Peer-to-Peer Networking and Communications Technical Sub-Committee, IEEE Comsoc Emerging Technologies Committee. He is also a guest editor of the Special Issue on Interactive Multimedia of the IEEE TRANSACTIONS ON MULTIMEDIA (2011) and the Special Issue on Distributed Image Processing and Communications of the IEEE SIGNAL PROCESSING MAGAZINE (2011). He was the TPC chair of IEEE Consumer Communications and Networking Conference (CCNC) in 2010. From 2003 to 2010, he served as a Vice-Chair of IEEE COMSOC Multimedia Communications Technical Committee (MMTC). He was a guest editor of the Special Issue on Peer-to-Peer Multimedia Streaming of the *IEEE Communications Magazine* (2007) and "Advances in Consumer Communications and Networking" in Springer Multimedia Tools and Applications (2007). He was a cochair of multimedia symposium in IEEE Globecom (2007 and 2006) and IEEE ICC (2007 and 2005). At Princeton, he was the 1993 recipient of the Charles Ira Young Memorial Tablet and Medal and the POEM Newport Award of Excellence.

Ali C. Begen (M'07) received the Ph.D. degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta.

He is currently with the Video and Content Platforms Research and Advanced Development Group at Cisco Systems, Inc., San Jose, CA. His interests include networked entertainment, Internet multimedia, transport protocols, and content distribution. He is currently working on architectures for next-generation video transport and distribution over IP networks, and he is an active contributor in the IETF in these areas.

Dr. Begen is a member of the Association for Computing Machinery. He was the recipient of the Best Student Paper Award at the IEEE ICIP 2003 and the Most Cited Paper Award from Elsevier Signal Processing: Image Communication in 2008.