Toward continuous push-based P2P live streaming

Dongni Ren *Wangkit Wong* *S.-H. Gary Chan*
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology, Kowloon, Hong Kong, China
Email: {tonyren, jerrywong, gchan} @cse.ust.hk

***Abstract*—Due to unpredictable peer churns (joins, leaves and failures), it is challenging to offer video continuity in peer-to-peer (P2P) live streaming. In this paper, we study a push-based P2P network formed by unreliable nodes (i.e., nodes which may churn at any time). To achieve high stream continuity, the video is encoded into k MDC (Multiple-Description Coded) streams and t FEC (Forward Error Correction) streams. To achieve low delay and reduce error correlation between streams, the $k+t$ streams are *pushed* to the nodes in *parent-disjoint* spanning trees. The issue is how to construct these trees minimizing the worst-case node delay.**

We address the optimization of the spanning trees through problem analysis and algorithmic design. After presenting a model capturing important system parameters and delay components, we formulate the problem and prove that it is NP-hard. We then propose SUN (Streaming with Unreliable Nodes), a simple, adaptive and distributed algorithm which continuously reduces delay through overlay adaptation. Through extensive simulation on real Internet and Internet-like topologies, we show that stream continuity can be achieved with push-based P2P streaming. SUN is effective, achieving low delay and high continuity in the presence of node churns for P2P live streaming.

I. Introduction

A peer-to-peer (P2P) live streaming network is often formed by unreliable nodes, i.e., nodes which may churn (i.e., join, leave or fail) at any time. Whenever there is a churn, the stream of the descendant nodes is disrupted. In order to offer high continuity, such disruption has to be mitigated. In this paper, we study push-based P2P live streaming with unreliable nodes. The challenge is to minimize overlay delay while achieving high stream continuity and meeting a certain streaming rate requirement (say, in excess of 1 Mbps).

Despite much work on P2P live streaming, there has not been sufficient consideration on how to design *push-based* live streaming to achieve *stream QoS* (in terms of continuity, bitrate and delay) with unreliable nodes. To the best of our knowledge, this is the first body of work addressing this. Previous approach is based on a pull-based mesh, where a node continuously searches for neighbors (using gossip) and pulls content from them. This results in rather *ad-hoc* connectivity, which adversely affects the end-to-end streaming rate that can be supported in the network. Furthermore, as the major objective of the pulling process is to aggregate a full video, it seldom optimizes source-to-end delay, leading to unsatisfactory delay performance.

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 Google Mobile 2014 and Faculty Research Awards.

In this paper, we consider a push-based P2P network achieving high stream quality, and study how to *minimize* source-to-end delay meeting a streaming rate requirement in the presence of node churns. We address the issue through problem formulation and algorithmic optimization. Tree-push systems have been shown to be effective in achieving low delay [1], [2]. However, there has been insufficient work on how to *optimize* trees under node unreliability. To address node unreliability, the video in our network is encoded into k MDC (Multiple Description Coding) streams of similar bandwidth ($k \geq 1$). To address node churns, t FEC (forward-error correction) streams of the same bandwidth are generated ($t \geq 0$), so that full video can be recovered so long as k out of these $k+t$ streams are received. (Clearly, a cost of that is bandwidth dilation due to the FEC streams.) Because of the use of MDC, video quality will be only partially affected if some of the k source streams are received.

To minimize delay, all the $(k+t)$ streams are *pushed* from the source to nodes in spanning trees. To further reduce the disruption on descendants due to node churns, we require that each node is served by $(k+t)$ *distinct* parents, each of which corresponds to a spanning tree (i.e., parent-disjoint spanning trees). Note that the use of MDC and FEC streams of much lower bandwidth than the original source effectively overcomes network bottlenecks to meet a certain streaming rate requirement.

Figure 1 shows an example of the streaming network with two MDC source streams and one FEC stream, i.e., $k = 2$ and $t = 1$. All streams are distributed from the server to nodes A, B, C, D and E. A, B and C are directly connected to the streaming server and they receive all their streams from it. Because node D and E are not served by the streaming source, they should connect to distinct parents in their spanning trees in order to achieve robustness against churns, i.e., D receives two MDC streams from A, B, and FEC stream from C, E receives source streams from A, C and FEC stream from D.

Because a node can decode the video only when any k out of the $(k+t)$ streams are received, its delay from the source is the *slowest path* out of all its $(k+t)$ trees (i.e., the maximum-delay path). Such delay increases quickly with the number of nodes if the trees are not constructed properly. The challenge is that how to construct the trees to achieve minimum delay.

In this paper we propose a novel and effective algorithm called SUN (Streaming with Unreliable Nodes) to construct highly efficient trees achieving low delay and high video quality (i.e., high continuity while meeting a streaming rate

978-1-4673-0921-9/12/$31.00 ©2012 IEEE

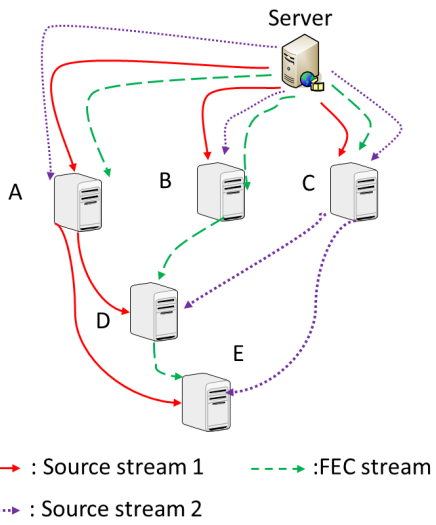

Fig. 1: An example of push-based P2P live streaming with unreliable nodes and its constituent underlying spanning trees.

requirement). Our study consists of the following:

- *Problem formulation and its complexity analysis:* Given k MDC streams and t FEC streams, we present a realistic model which captures scheduling delay, edge bandwidth, propagation delay, etc. We formulate the delay optimization problem which is to design spanning trees that minimize the diameter (i.e., worst-case delay) of the overlay network. We prove that the problem is NP-hard.
- *SUN: A distributed P2P streaming algorithm to achieve high video quality:* We propose SUN, a novel algorithm for each node to search for $(k + t)$ distinct parents to achieve low delay. Our algorithm is simple, fully distributed and efficient (i.e., low overhead), and is robust to unreliable nodes (i.e., maintains high stream continuity in the presence of node churns). It adapts to dynamic network conditions with nodes continuously adjusting their positions in the trees to reduce delay while meets streaming rate requirement.
- *Simulation studies:* We conduct extensive simulation study on topologies from real Internet and Internet generator. Our results show that SUN achieves low delay and high continuity in the presence of unreliable nodes.

The organization of the paper is as follows. After discussing related work in Section II, we present the optimization problem and its complexity analysis in Sections III. The distributed algorithm of SUN is discussed in Sections IV. Illustrative simulation results are presented in Section V. We conclude in Section VI.

II. Related Work

Single tree structure was first proposed to distribute streams among peers, where all peers are arranged into a tree rooted at the source [3], [4]. Although these approaches are simple and achieve low delay, the failure of a node will lead to stream disruption of all its descendants. To address the weaknesses of single tree structure, multiple trees (or forest) approach has been proposed [5], [6]. Most of the work, however, has not investigated the *optimization* of the forest structure and how to achieve high continuity in the presence of peer churns. To the best of our knowledge, this is the first body of work addressing the design of push-based multi-tree P2P streaming to achieve QoS (in continuity and bitrate). We formulate the construction of multiple trees as an optimization problem. Our proposed scheme leads to a highly optimized and adaptive structure with high stream continuity. Another body of work on streaming mesh uses pull-based approach for data exchange [7]. In this approach, peers usually connect to their closest neighbours using gossip to pull data. Despite its simplicity, this approach often leads to high delay (due to its random connections and buffermap) and high control overhead (due to messaging and bitmap exchange). As compared to pull-based mesh, our push-based network achieves much lower source-to-end delay.

Multiple description coding (MDC) has been widely used in media streaming to address the bandwidth heterogeneity issue. The video is encoded into multiple descriptions. At the receiver end, the streaming quality is proportional to the number of descriptions received [8]. In forward error correction (FEC), the sender generates redundant data to its messages. With the redundancy the receiver can recover all the source packets if it receives a certain number of coded packets [9]. In this work we use MDC to encode the video stream so that they are delivered by unique spanning trees. We also adopt the FEC method to generate repair streams against churns.

III. Problem Formulation and Complexity Analysis

Consider a P2P network modeled as a directed graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices representing the nodes in the overlay (including the server) with $n = |\mathcal{V}|$ and $\mathcal{E} = \mathcal{V} \times \mathcal{V}$ the set of overlay edges between nodes. Let $s \in \mathcal{V}$ represents the source. The edge cost of $\langle i, j \rangle$, denoted as d_{ij}, represents the underlay unicast path delay from node i to node j, which is equal to the sum of the propagation delay d^p_{ij} and the scheduling delay d^s_{ij} from node i to node j, i.e., $d_{ij} = d^p_{ij} + d^s_{ij}$.

The stream is split into k MDC source streams of similar bandwidth. The k source streams also generates t FEC streams for recovery purpose. To mitigate the adverse effect of node churn, all the $(k+t)$ streams are distributed in distinct delivery trees in such a way that each peer is served by $k + t$ *distinct* parents. Note that peers directly connected to the source receive all the streams from the source (as the source is assumed to be reliable). A delivery tree is a spanning tree for a stream containing all the nodes in $\mathcal{V}$ rooted at s. Denote the set of k MDC trees consisting of source-stream parents as $\mathcal{K}$ and the set of t FEC trees consisting of recovery parents as $\mathcal{T}$. Further let $\mathcal{P}_i$ be the set of all $(k + t)$ parents of node i.

Let the streaming rate of the MDC-encoded video be B kbps, and hence the bitrate of one MDC or FEC stream is $b =$

B/k kbps. We refer b as the basic unit of network bandwidth, which is the bandwidth reserved by the parent for a single end-to-end connection (i.e., parents stream to each of their children at rate b).

For every node i in V, it has an uplink bandwidth of u_i units, $u_i \in Z^+$, which represents the maximum total number of children it can serve in all spanning trees. The end-to-end throughput of the edge $\langle i, j\rangle$ is denoted as $w_{ij} \in Z^+$, which is the maximum number of substreams that can simultaneously accommodate in edge $\langle i, j\rangle$. For any node in V , if it gets an aggregate of k out of the $k + t$ streams from its parents, we call the node *fully served*. In other words, if node i receives k streams from all $k + t$ spanning trees, it is *fully served* and can play back the video with continuity. Note that s has an uplink bandwidth of u_s units and has no parent.

The worst-case scheduling delay from node j to node i, denoted as d^s_{ji}, is given by

$$d^s_{ji} = \sum_{k\in\mathcal{P}_j} \frac{L}{\min(w_{jk}, u_j)b/t_{jk}}, \tag{1}$$

where L (bits) is the segment size used in streaming, and t_{jk} is the number of concurrent substreams on edge $\langle j, k\rangle$ (we have make the usual assumption that bottleneck is at the edge).

The problem is to minimize the worst-case delay of the network. Consider a packet transmitted from s at time 0 via a delivery tree l, where $l \in \mathcal{K} \cup \mathcal{T}$. Denote D^l_i the maximum delay (including packet recovery from FEC parents) for the packet to arrive at node i. By definition, $D^l_s = 0$. Node i first gets its packet from its $k + t$ parents. Consider the tree $l \in \mathcal{K} \cup \mathcal{T}$. Denote the delay of the packet to node i from its parent j as d^l_i, which is obviously given by

$$d^l_i = D^l_j + d_{ji} \tag{2}$$

The overall delay of node i, i.e. the time taken for a peer to aggregate the entire stream before playback, is determined by the slowest path, i.e.,

$$D_i = \max_{l\in K\cup T} D^l_i. \tag{3}$$

Minimum Delay Robust Streaming Problem (MDRS): The MDRS problem is to construct a multi-tree overlay out of unreliable nodes with $k+t$ spanning trees rooted at the source and each node having $k + t$ distinct parents (k MDC parents and t FEC parents) so that the worst-case delay of the peers is minimized, i.e.,

$$\min \max_{i\in\mathcal{V}} D_i. \tag{4}$$

MDRS problem is NP-Hard. The Travelling salesman problem (TSP) is reducible to MDRS in polynomial time. The proof of a similar problem can be found in [10].

IV. SUN: A Distributed Tree Construction Algorithm to Achieve Stream QoS

In this section, we present SUN, a simple and fully distributed algorithm scalable to large group to reduce peer delay. SUN acheives high stream QoS in terms of continuity and bandwidth. There are four operations of the algorithm:

A. Peer Arrival

A new arrival, say peer i, has to have $k + t$ parents to assemble a full stream in a dynamic network environment. To achieve that, it contacts a rendezvous point (RP) which returns a number of peers in the overlay as the pool of candidate parents. It may enlarge the pool by requesting neighbors from these nodes.

Peer i then checks the delay d_{ji} with each candidate j. In addition to network distance, peer i also asks j for source-to-end delay D^l_j in each spanning tree l and the amount of its available bandwidth (which is simply u_j minus the number of children that j has). In other words, the nodes in the overlay compute their delays according to Equation 3. Given a delivery tree l, peer i selects the node with the minimum delay among the candidates (i.e., $\min_j(D^l_j + d_{ji})$) and connects to it to retrieve the substream. The selected parent is removed from the pool and peer i repeats the process for the remaining delivery trees. According to the above, it joins all k delivery trees to fulfill the streaming rate requirement.

B. Peer Departure and Failure

Each peer in the network periodically sends its "heartbeat" to its parents and children. Upon detecting that a parent of tree l has left, the child node contacts RP to retrieve a new list of peers, and tries to rejoin tree l by selecting a new parent from the peer list. During the rejoin process, In the mean time if the lost parent serves one of the k MDC streams, the affected nodes use the FEC streams to recovery the lost data.

C. Adaptation

In a distributed environment, there is no specific *a priori* joining and leaving order of peers. As a result, a peer may need to adapt to the dynamic network condition to move to a better position in the overlay to reduce delay. This is especially important when a high-bandwidth node joins the network at a later time. Node movement to better position is the design objective of adaptation.

The adaptation consists of three steps:

- Request: A child i periodically inspects its residual bandwidth. If this is greater than the streaming rate, i sends its parents a REQUEST for adaptation. The REQUEST message contains its available bandwidth and a time-to-live field (TTL) indicating the scope of the REQUEST is to be flooded. When a node receives a REQUEST message, if the TTL field is greater than zero, it decrements TTL and forwards the REQUEST message to its neighbors including parents and children (except the one from which the message comes from).
- Grant: Upon receiving a REQUEST message, the candidate checks whether its uplink bandwidth is less than the residual bandwidth of the requester i, the request originator. If this is the case, the candidate sends the requester i a GRANT response which contains its delays

(which may be in hops) in each delivery tree. The GRANT message indicates that the adaptation between the requester and the candidate is permitted.

- Accept: Child i then chooses the slowest tree l from all $k+t$ spanning trees to adapt. At this stage, child i may have received a number of GRANT messages from different ancestors. Among the ancestors (upstream nodes) who have sent GRANT messages to it, child i accepts the ancestor j that satisfies the following conditions: i) $D_j^l < D_h^l + d_{hi} < D_i^l$, where h is the parent of j in tree l (i.e., i can improve its delay by changing parent to h); and ii) available uplink bandwidth of peer j is less than i. Obviously, h-i-j does not form any parent-child relationship in the original tree. After such j is found, i replaces its existing parent in the tree with h. In addition, j connects to i to get the substream l. This is possible because the conditions guarantee both i and j still retains the property of distinct parents. Moreover, the second condition makes sure that we are moving high-bandwidth peers to a position above the low bandwidth ones. In summary, child i takes up the position of the ancestor j in tree l and in turn provides the substream l to j. It is clear that after the process, the delay of the slowest tree is reduced.

V. Simulation Environment and Illustrative Results

We carry out simulations to evaluate the performance of the proposed algorithm. For comparison purpose, we also simulate two traditional scheme, namely, *closest parent* and *random parent*. The *closest parent* scheme is distance-based, in which peers look for the closest parents for streaming. Since this scheme captures locality among peers, it (or its variant) is widely adopted for P2P streaming with satisfactory performance [3], [11]. While the *random parent* scheme randomly chooses parents for newly-arrived nodes, which is used in many state-of-art commercial applications [12]. This scheme does not capture the locality of the nodes, thus streaming network constructed in this way is often of high delay.

A. Simulation Setup and Metrics

In simulations, peers arrive according to a Poisson process at rate λ (requests/second) and then remains in the overlay for an exponential length of time (seconds). During their life-time, peers follow the proposed algorithms to search for parents. The user holding time is according to an exponential distribution with mean $1/\mu$ (seconds). We have used other distribution and the results are qualitatively the same. To be fair in comparison, peers in all the schemes look for the same number of parent to achieve the same streaming quality.

Our simulation is carried out on a real Internet topology provided by CAIDA, which was collected on June 12th, 2011 and contains 1,747 routers and 3,732 links. The round trip times (RTTs) between inter-connected routers are also given in the topology. We use Distance-vector routing to compute the latencies between any two router nodes in the network.

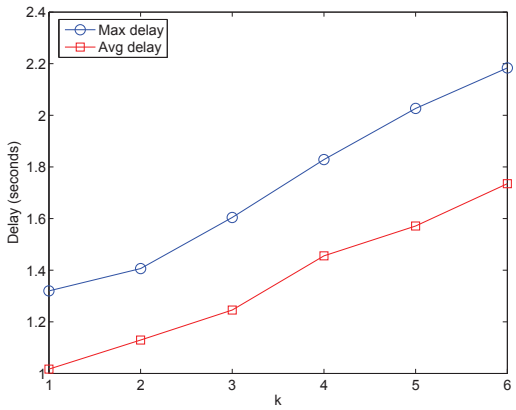

Fig. 2: Delay of SUN versus k.

Peers are randomly attached to the routers. Unless otherwise stated, we use $k = 2$, $t = 1$, $\lambda = 1$ req/s and $1/\mu = 480$ seconds as our baseline parameters. Note that u_i is normally distributed with mean 2 Mbps and standard deviation 1 Mbps (We only take the positive values). The streaming rate B is 1Mbps. The packet size C is 100 kbits. The search time for a new parent is uniformly distributed from 1 second to 15 seconds. Our target worst-case loss rate of all peers is 5% The performance metrics of interest are:

- *Delay:* The primary concern of our protocol is the source-to-end delay of peers. It is measured by summing all the link delays on the overlay path from the source to the peer. Because there are multiple paths to reach a peer, the maximum delay of all the paths is taken as the measure. Delay is measured according to Equations (2)-(3). We are interested in the average, maximum and distribution of the source-to-end delay in the overlay.
- *Loss rate:* When ancestors of a node leave the network, the peer may experience data loss. In our scheme, k MDC stream and t FEC stream are pushed to all nodes in the streaming overlay, so long as k out of these k+t streams are received by the node, it is able to recovered full video and will not be affected by the peer leave. If a node receives less than k stream in total, its video quality will be partially affected. Its residual video quality equals to number of MDC streams received after recovery divided by k; and the loss rate equals to one minus residual video quality. The overall loss rate of a node is equal to its average loss rate over time.

B. Illustrative Simulation Results

Figure 2 shows the performance of SUN versus the number of MDC streams k. To keep the total streaming rate consistent, each MDC stream has a lower rate when k increases. Both maximum delay and average delay increase with k. The reason is that each node in the overlay needs to obtain streams from more parents. This leads to a higher scheduling delay. The overall delay is also increased because we calculate it as the maximum delay in all trees.

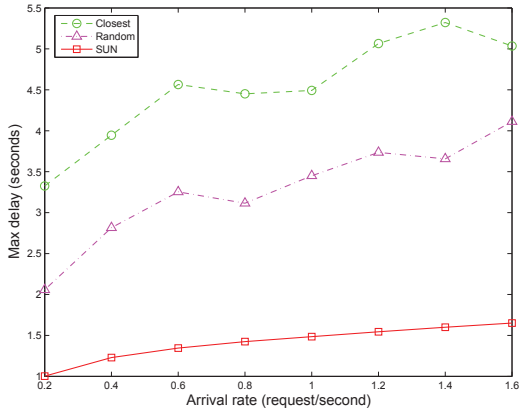

Fig. 3: Maximum delay versus arrival rate.

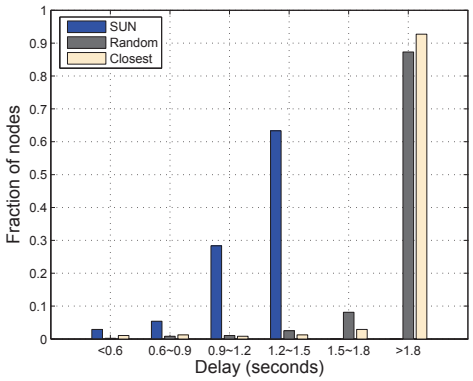

Fig. 4: Delay distribution for different schemes.

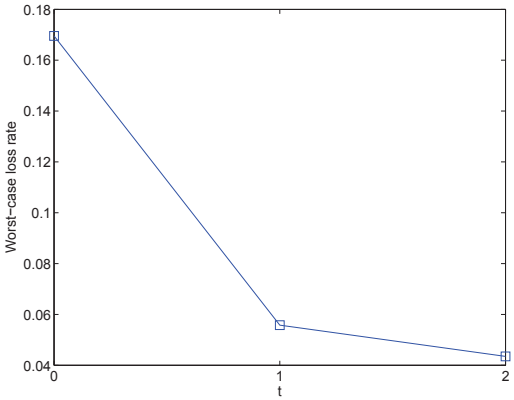

Fig. 5: Worst-case loss rate vesus t.

We compare SUN with other schemes by plotting the maximum delays versus peer arrival rate λ in Figure 3. In general, the delay increases with the arrival rate. This is because the system population increases with arrival rate, which leads to longer overlay diameter and hence delay. Clearly, SUN performs significantly better than the traditional Closest parents scheme and random parents scheme. It can position the peers effectively and achieves much better delay, which stays low even when the number of peers increases. The Closest parent scheme, despite of its forming close-neighbor groups among peers, performs the worst. There are two reasons for this. First, it has not considered source-to-peer delay. Each peer only greedily selects parents with shortest RTT to it. On the other hand, SUN achieves low delay by putting peers closer to the source. Second, Closet neighbor does not consider how many children a node is serving, thus often leads to a parent node overwhelmed and with high scheduling delay. Random parents scheme does not have good delay performance either because it has not considered the locality of the nodes.

In Figure 4 we show the comparison of delay distribution between SUN and other schemes. Clearly, SUN outperforms other schemes with more low-delay nodes. The performance of SUN is very well when most peers achieve low delay (0.9 to 1.5s). The worst-case delay is not much larger than the other peers. It demonstrates that SUN is able to arrange the overlay in a way that most of the nodes in the overlay share rather similar delays, and the worst-case delay is optimized.

To illustrate the benefit of FEC stream, we show in Figure 5 the worst-case loss rate against the number of FEC stream t. Clearly, adding one FEC stream can significantly reduce the loss rate. As the number of FEC streams increases, we can achieve lower loss rate. This is because more FEC streams means that there are more redundancy against node churns. The decrease of loss rate, however, diminishes as t increases. Therefore we use one FEC stream as benchmark to provide robustness to nodes against network dynamics and peer churns.

We show in Figure 6 the loss rate distribution of SUN-Distributed. We observe that there are more peers with higher loss rate when FEC recovery is not used (More than 60% of the peers have a loss rate larger than 2%). With one FEC stream, the loss rate of the peers is significantly improved when most of the peers suffer from nearly no loss (less than 2%). It also shows that the number of FEC streams does not need to be large. One redundant stream is enough to efficiently reduce the loss rate in streaming.

From Figure 7, we observe that the loss rate varies with k given a specific number of FEC stream (t=1). The loss rate first

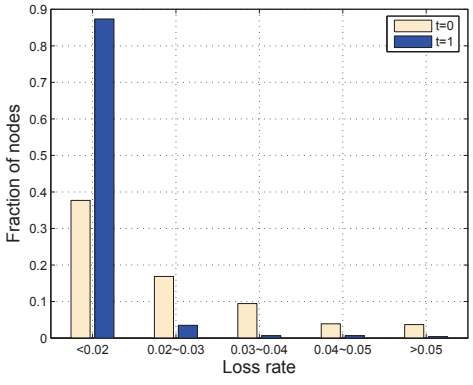

Fig. 6: Loss rate distribution.

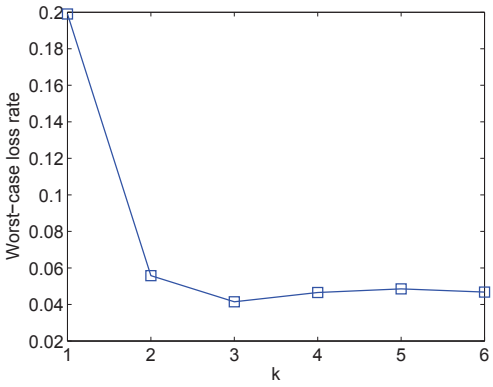

Fig. 7: Worst-case loss rate vesus k.

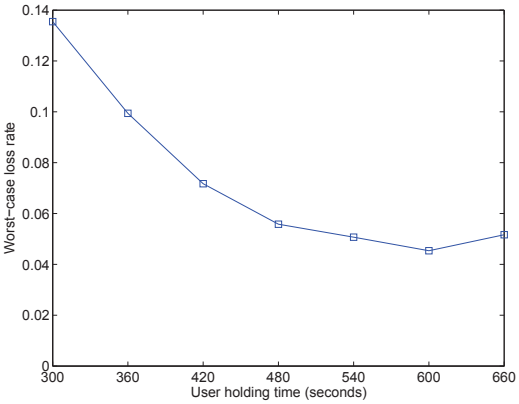

Fig. 8: Worst-case loss rate versus user holding time.

decreases, and then flattens off as k increases. This is because in Multiple Description Coding, each received MDC stream can be decoded independently. More MDC streams provides robustness to node churns. However, if the number of MDC streams is too large, the nodes have to connect to more distinct parents and they are more likely to suffer from an ancestor failure. Therefore k does not need to be very large since more MDC streams does not necessarily lead to low loss rate. This is also the reason we choose $k = 2$ as our baseline in simulation.

Figure 8 shows the worst-case loss rate versus average holding time. The loss rate decreases as the holding time increases. This is because when the holding time is larger, nodes stay connected to their parents longer and hence experience less interruption, and loss rate therefore improves. However when the holding time is greater than some value (550 seconds), there is no obvious improvement in loss rate. This is because the number of peers in the streaming network increases with holding time and thus the tree depth also gets higher. Each peer will have a larger hop count to the source and hence much easier to be affected by the churns of its ancestors.

VI. Conclusion

We have studied how to design a P2P streaming network achieving high video quality (in continuity and bitrate) and low delay with unreliable nodes. To mitigate peer churns, we consider that the stream is divided into k MDC source streams and t FEC recovery streams. Each peer has $k + t$ distinct streaming parents delivering the streams. We have formulated the overlay design problem as Minimum Delay Robust Streaming (MDRS) problem, which is to form a minimum-delay overlay given k and t. We prove the problem is NP-hard, and propose a simple, adaptive and distributed algorithm called SUN that constructs a low-delay network in the presence of node churns to achieve low delay and high video continuity.

We have conducted extensive simulation on real Internet topologies to study the performance of our algorithm. The results show that SUN achieves much lower source-to-peer delay as compared with the representative closest parent and random schemes. It also achieves high stream continuity (above 95%) despite the dynamic behavior of the network.

References

[1] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or multiple-tree, A comparative study of live P2P streaming approaches," in *IEEE INFOCOM*, Anchorage, Alaska, USA, May 2007, IEEE, pp. 1424–1432.

[2] Wenjie Jiang, S.-H. Gary Chan, Mung Chiang, Jennifer Rexford, K.-F. Simon Wong, and C.-H. Philip Yuen, "Proxy-P2P streaming under the microscope: Fine-grain measurement of a configurable platform," in *Proceedings of the 19th International Conference on Computer Communications and Networks (ICCCN) (Invited paper)*, 2-5 Aug. 2010.

[3] Vidhyashankar Venkatraman, Kaoru Yoshida, and Paul Francis, "Chunkyspread: Heterogeneous unstructured end system multicast," in *The 14th IEEE International Conference on Network Protocols*, Nov. 2006.

[4] Hao Yin, Xuening Liu, and Zhan, "Livesky: Enhancing CDN with P2P," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 6, pp. 16:1–16:19, Aug. 2010.

[5] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth multicast in cooperative environments," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, The Sagamore, Bolton Landing (Lake George), New York, Oct. 2003, pp. 298–313.

[6] Jiancong Chen, S.-H. Gary Chan, and Victor O. K. Li, "Multipath routing for video delivery over bandwidth-limited networks," *IEEE Journal on Selected Areas in Communications Special Issue on Design, Implementation and Analysis of Communication Protocols*, vol. 22, no. 10, pp. 1920–1932, Dec. 2004.

[7] Nazanin Magharei and Reza Rejaie, "PRIME: peer-to-peer receiver-driven mesh-based streaming," *IEEE/ACM Trans. Netw.*, vol. 17, pp. 1052–1065, Aug. 2009.

[8] Chia-Wei Hsiao and Wen-Jiin Tsai, "Hybrid multiple description coding based on H.264," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, Jan. 2010.

[9] D. Jurca, P. Frossard, and A. Jovanovic, "Forward error correction for multipath media streaming," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, Sept. 2009.

[10] Dongni Ren, Y.-T. Hillman Li, and S.-H. Gary Chan, "On reducing mesh delay for peer-to-peer live streaming," in *IEEE INFOCOM*, Phoenix, Arizona, Apr. 2008, IEEE.

[11] Xiaofei Liao, Hai Jin, Yunhao Liu, Lionel M. Ni, and Dafu Deng, "Anysee: Peer-to-peer live streaming," in *Proc. IEEE Infocom*, 2006, pp. 2411–20.

[12] Xiaojun Hei and Chao Liang, "A measurement study of a large-scale P2P IPTV system," *IEEE Transactions on Multimedia*, vol. 9, no. 8, 2007.