PRACTICAL ALGORITHM FOR MINIMUM DELAY PEER-TO-PEER MEDIA STREAMING

Jun Luo

School of Computer Engineering, Nanyang Technological University (NTU), Singapore
Email: junluo@ntu.edu.sg

ABSTRACT

Though the **existence** of a minimum delay peer-to-peer media streaming scheme has been shown (under the name of *snow-ball streaming*), **no** actual algorithm has ever been designed so far, due to the lack of a systematic way to construct the chunk scheduling that achieves the minimum delay bound. Inspired by the growth of interest in building hybrid streaming systems that consist of backbone trees supplemented by other overlay structures, we revisit the minimum delay streaming problem and design practical min-delay algorithms to support the streaming in the backbone trees. What mainly distinguishes our multi-tree push from the conventional ones is an unbalanced tree design guided by the snow-ball streaming, which has a provable minimum delay. We design algorithms to construct and maintain our *SNowbAll multi-tree Pushing* (SNAP) overlay. Our simulations in *ns*-2 indicate that our approach outperforms other tree-based mechanisms.

Keywords— P2P streaming, delay, tree-based push

1. INTRODUCTION

We have been witnessing a tremendous growth of peer-to-peer (P2P) streaming applications on the Internet. In the networking research community, this momentum transforms into many challenging problems. Among the major technique problems, improving the **delay** performance of P2P streaming appears to be a recurring issue, as pointed out by [1, 2, 3]. Earlier designs for P2P streaming are mainly *tree-based push* (e.g., [4, 5]). Though they achieve satisfactory delay performance, they do suffer higher maintenance complexity and data outage upon peer dynamics [6]. Later designs that make use of *mesh-based pull* (e.g., [7, 8]) are the mainstream implementations that are widely deployed. They generally provide a higher robustness against peer dynamics but have to strike a compromise between efficiency and latency [3]. In order to obtain the best from both worlds, several proposals recently started to promote a hybrid approach [9, 10]. Such a hybrid approach usually has a tree-based backbone that actively pushes media data to stable and resourceful peers, as well as a second-tier overlay that accommodates dynamic or resource-scarce peers.

This work is supported in part by AcRF Tier 1 grant RG 32/09.

In order to reduce delay within a backbone tree, one would desire to achieve the minimum delay bound for disseminating every media chunk to (say) N peers. This bound, given a homogeneous uploading capacity, is shown to be $1 + \lceil \log_2 N \rceil$ [11], where the **existence** of a minimum delay streaming scheme has been shown, under the name *snow-ball streaming*. Unfortunately, due to the lack of a systematic way to construct a distributed scheduling for chunk uploading, no algorithm has ever been designed so far. Existing proposals that aim at minimizing delay are approximations of the snow-ball streaming, through either deterministic pull [3] or randomized push [12].

In this paper, we focus on the design of practical streaming algorithms that achieve the minimum delay. In particular, we propose algorithms to construct and maintain a multi-tree overlay, called *SNowbAll multi-tree Pushing* (SNAP). The scheduling policy guided by SNAP trees guarantees a minimum delay for continuous chunk streaming, as promised by the snow-ball streaming principle. We also implement SNAP in *ns*-2, our simulation results show that SNAP outperforms its up-to-date competitors. As we are advocating a hierarchical structure where stable peers [10, 13] are organized into a multi-tree backbone while the remaining peers are accommodated by another tier of overlay, we believe that our SNAP can exactly be used for streaming in the backbone.

The remainder of this paper is organized as follows. We briefly explain the idea of snow-ball streaming in Section 2, and we also put forward a graph (tree in particular) representation for disseminating **single** media chunk using the snow-ball principle. We then describe in Section 3 the algorithms that construct and maintain SNAP, which consists of multiple trees for continuous streaming. We report the experiment results in Section 4 and conclude our paper in Section 5.

2. SNOW-BALL DISSEMINATION FOR SINGLE CHUNK

Literally, the basic idea of snow-ball streaming for single-chunk dissemination can be summarized as follows [11]:

> *After receiving a new chunk, a peer keeps pushing that chunk to other peers who have not received it, until every peer receives the chunk.*

978-1-4244-7493-6/10/$26.00 ©2010 IEEE

Mathematically, if we assume that time is sloted and all the uploadings are aligned to the time slots, we could represent the number of peers that receive that chunk as a function of the sequence number of the time slots. Let 0-th slot be the period during which the media serve uploads the chunk to the first peer, then the number of peers that receive the trunk at the end of kth is

$$R_k = \begin{cases} 2^k & N - R_{k-1} \geq 2^k \\ N - R_{k-1} & N - R_{k-1} < 2^k \end{cases}$$

where N is the total number of peers. The optimality of this algorithm is proven in [11]: as every peer keeps busy in uploading immediately after its own reception, there is no room to make further improvement.

It is pretty trivial to transform this algorithm into an unbalanced tree structure that we call *snow-ball tree* (SBT). We show such a transformation for $N = 16$ in Fig. 1. Basically,

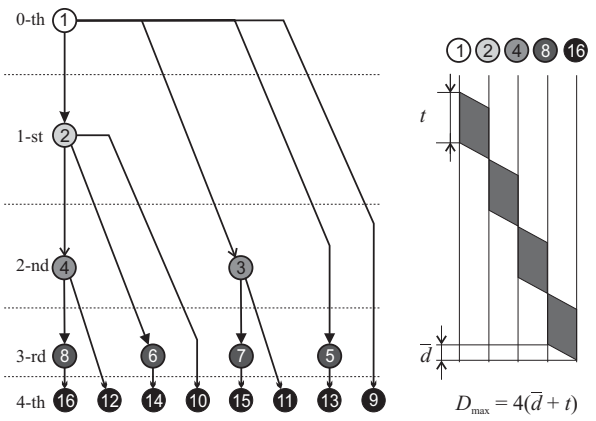


Fig. 1. The *snow-ball tree* (SBT) of single chunk dissemination. We illustrate both maximum delay calculation and the definition of *peer level set* in the figure.

the peers that receive the chunk in the k-th time slot are put at the k-th level of the tree. The geometric progression nature of R_k guarantees that each peer in the k-th level definitely has a parent in the previous levels.

This tree representation greatly facilitates the calculation of chunk delay. It is clear that the delay of each uploading consists of two parts: *transmission delay* t and *average queueing/propagation delay* $\bar{d}$. In general, a media chunk consists of several packets (up to several MBs in volume), so we usually have $\bar{d} \ll t$, suggesting that the delay is dominated by t. In Internet data transmission, the queueing/propagation delay is often counted once due to the well known *pipelining* effect. However, as a peer has to fully receive a chunk before sending it out, the extent that pipelining effect appears strongly depends on the path from the root to a certain peer in an SBT, which leads to the following maximum delay $D_{\max}$ and average delay $\bar{D}$.

$$D_{\max}^{\text{SBT}} = \lceil \log_2 N \rceil \left(\bar{d} + t \right) \quad (1)$$

$$\bar{D}^{\text{SBT}} \leq \frac{\lceil \log_2 N \rceil}{2} \bar{d} + \left(\lceil \log_2 N \rceil - 1 \right) t + o(N) \quad (2)$$

The maximum delay obviously comes from the path that has no pipelining effect, the path $1 \rightarrowtail 2 \rightarrowtail 4 \rightarrowtail 8 \rightarrowtail 16$ shown in Fig. 1. The derivation of the average delay is a bit tricky and we refer readers to our technical report. Note that our derivation differs from the one presented in [11] in that we explicitly separate the two delay components. If we let $\bar{d} = 0$, we get the same expression presented in [11].

3. SNOWBALL STREAMING BY A MULTI-SBT STRUCTURE

Although SBT is shown to be equivalent to the snow-ball algorithm, the extension of SBT for multi-chunk dissemination (hence media streaming) is far less trivial. In the original proposal of the snowball streaming algorithm [11], the existence of an algorithm for chunk streaming is proven by an induction, which cannot be used to derive a multi-tree structure. Recently, Feng et al. [3] apply Trellis graphs to represent the snowball streaming algorithm, but they fail to suggest a distributed chunk scheduling policy. Therefore, we need to find a multi-tree extension for SBT that provides the same delay guarantee for chunk streaming. To this end, we first propose a centralized construction algorithm, then we show how to make it operate in a distributed manner.

3.1. Multi-Tree Representation of The Snowball Algorithm

Let $\mathcal{N}$ be the set of all peers and $|\mathcal{N}| = N$. Our design goal can be described as follows:

> *A set of SBTs such that, if the server takes turn to push chunks to their roots in a round-robin fashion, the minimum delay streaming is achieved.*

For practical purposes, we require the multi-tree structure to consists of a finite number of SBTs. In other words, the chunk dissemination follows a repetitive pattern of trees with a period of P. Note that, as each chunk has a corresponding SBT and we know that SBT is delay optimal for single chunk dissemination, the challenge we are facing is to resolve the parallelism among all these P SBTs.

We refer to the i-the SBT in this P-tree pattern as T_i. We denote by $L_{k,i}^p$ the *peer level set* containing the set of peers in the k-th level of T_i, and by $L_{k,i}^e$ the *edge level set* containing the edges whose ends are in $L_{k,i}^p$. We illustrate the concept of peer level set in Fig. 1, where we only show an arbitrary T_i with $\{L_{0,i}^p, L_{1,i}^p, L_{2,i}^p, L_{3,i}^p, L_{4,i}^p\}$ and with $L_{0,i}^p$ containing the peer that directly receives a chunk from the server. Finally,

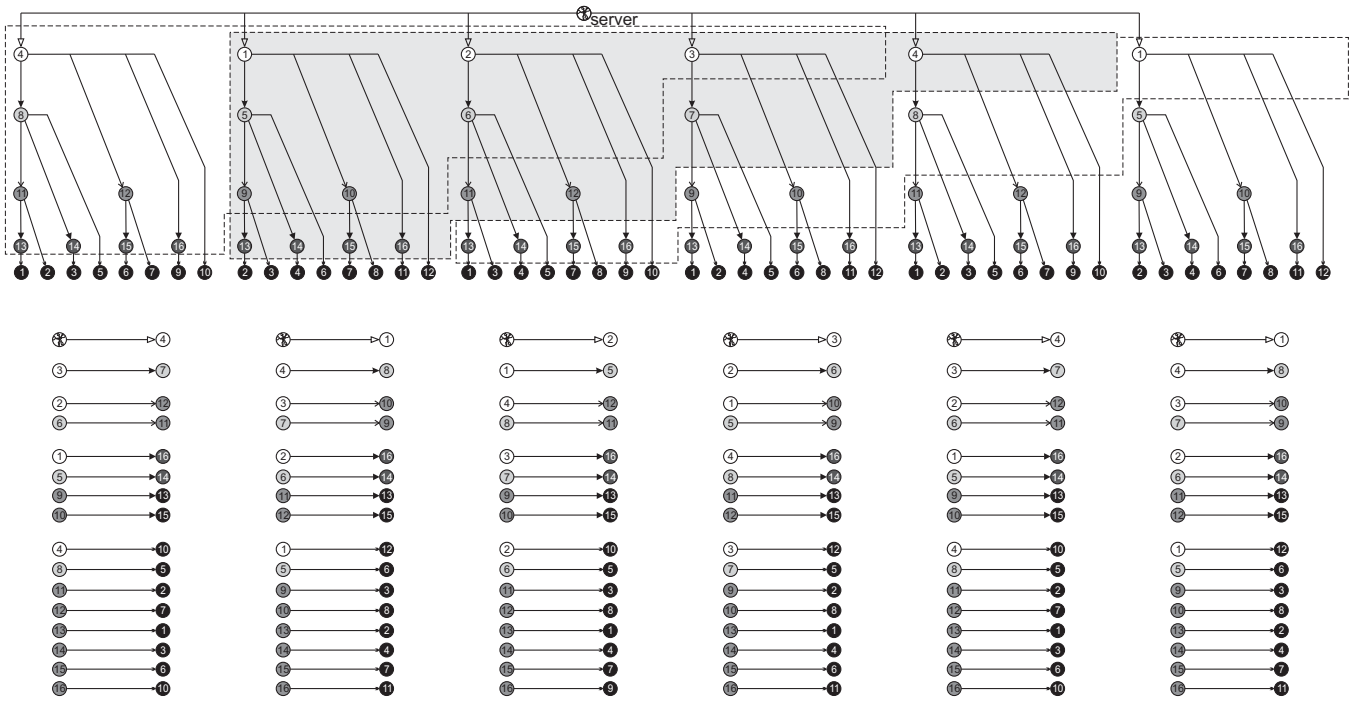


Fig. 2. The multi-SBT structure, K-ISet, and corresponding uploading schedules for $N = 16$. The scheduled uploadings below each SBT take place concurrently when the server pushes a chunk to the root of this tree.

let $K = \lceil \log_2 N \rceil$ be the maximum depth of every SBT. As the minimum delay can be guaranteed for every SBT only if the maximum parallelism is achieved in terms of the uploading scheduling, we have a sufficient condition for a multi-tree structure to yield the minimum delay for chunk streaming.

Proposition 1 *The following condition guarantees a multi-tree structure to yield the minimum delay: If the edges in $L^e_{1,i}$ is scheduled, then all edges in $L^e_{k,i-k+1} : k \in (2, K)$, as well as the $N - 2^{K-1}$ edges in $L^e_{K,i-K+1}$, should be able to be scheduled together.*

The proof is omitted and the readers are referred to our technical report. Basically, this scheduling squeezes the sequence of a geometric progression (from 1 to 2^K) into one time slot, though the peers involved in the progression come from different trees. To facilitate further discussion, we use the following definition to characterize the **origins** of the edges involved in the schedule described in *Proposition 1*.

Definition 1 *A set of peers is an independent set (ISet) if they are the origins of all the edges involved in the schedule described in Proposition 1. In particular, for some integer K, a K-ISet refers to an ISet in a system of $N = 2^K$ peers.*

It is clear that an ISet should not contain duplicate peers, as a peer can only perform one uploading within one time slot. In the next section, we will explain how we can construct a Multi-SBT structure that satisfies the requirement set in *Proposition 1*. We use Fig. 2 to give a more tangible illustration of the concepts of ISets, as well as the sufficient scheduling required by *Proposition 1*.

3.2. Constructing the Multi-SBT Structure

Given the sufficient condition to achieve the minimum delay described in *Proposition 1*, our design goal is achieved if we could construct a multi-tree structure satisfying that condition. We first give, in Fig. 3, the algorithm that performs the construction for $N = 2^K$, then we show its extension to an arbitrary N. We omit the correctness proof for brevity.

Algorithm *Multi-SBT Construction for $N = 2^K$*
1. $\mathcal{S} \leftarrow \mathcal{N}$; $s_k \leftarrow \emptyset, k = 0, 1, \cdots, K$; $k \leftarrow 0$
2. **repeat**
3. $s_k \leftarrow (K-k)2^{(k-1)^+}$ distinct peers in $\mathcal{S}$
4. **do** assign the peers in s_k to the k-th level of the SBTs in a periodic fashion.
5. $\mathcal{S} \leftarrow \mathcal{S} \backslash s_k$; $k \leftarrow k + 1$
6. **until** $k = K$
7. Fill the $L^p_{K,i}$ with the peers that have not appeared in T_i.
8. **return** P distinct SBTs and $\{s_0, s_1, \cdots, s_K\}$.

Fig. 3. The multi-tree construction algorithm for $N = 2^K$

Basically, the algorithm starts with empty "skeleton" of trees whose vertices need to be indexed. The data struc-

ture used to represent the trees is $\{s_k\}_{k=0,1,\cdots,K}$, with $s_k = \bigcup_i L^p_{k,i}$ (i.e., s_k includes the peer level sets of all the trees). According to the 3rd and 4th steps of the algorithm, the peers in the k-th level of a multi-SBT structure repeat in a period of $P_k = K - k$, given the number $|s_k|$ of peers to be put in the k-th level of the multi-SBT structure and the fact that there are $2^{(k-1)^+}$ peers in $L^p_{k,i}$ of a tree T_i. Consequently, the **maximum** number of required SBTs is the *least common multiple* (LCM) of a set of periods $\{K, K-1, \cdots, 1\}$. Fortunately, the period P of the tree pattern can actually be much smaller than that number. As detailed calculation shows that the total number of peers required to complete the algorithm is only $2^K - 1 = N - 1$, we can make use of this extra peer to increase the period of the first level to K, which has the potential to greatly reduce P. We illustrate the outcome of this algorithm for $N = 16$ in Fig. 2: the ISets (in particular 4-ISets) are shown as the sets of peers that encompassed in the staircase-shaped frames. Although the LCM of the set of periods $\{4, 3, 2, 1\}$ is 12, the actual period is just 4, as we use that extra peer to increase P_1 from 3 to 4.

For an arbitrary N where $2^{K-1} < N < 2^K$, the size of an ISet is between 2^{K-1} and 2^K. Therefore, we simply put $N - 2^{K-1}$ peers in arbitrary positions in the relative complement of (K-1)-ISet in K-ISet, which is effectively equivalent to changing the periods at certain levels, as shown by the steps from 4 to 7 of the algorithm in Fig. 4 (which is an extension of the basic algorithm in Fig. 3). These extra peers are then

Algorithm *Multi-SBT Extension for Arbitrary N*
1. Choose arbitrary $\widetilde{\mathcal{N}} \subseteq \mathcal{N}$ s.t. $|\widetilde{\mathcal{N}}| = 2^{\lfloor \log_2 N \rfloor}$
2. Run Multi-SBT Construction for $\widetilde{\mathcal{N}}$
3. $\mathcal{S} \leftarrow \mathcal{N} \backslash \widetilde{\mathcal{N}}$; $s \leftarrow \emptyset$; $k \leftarrow 0$
4. Choose an ascendingly ordered set $\mathcal{K} = \{k_1, k_2, \cdots, k_m\}$ s.t. $|\mathcal{S}| = \sum_{k \in \mathcal{K}} 2^{(k-1)^+}$
5. **for all** $k \in \mathcal{K}$
6. $s \leftarrow 2^{(k-1)^+}$ distinct peers in $\mathcal{S}$; $s_k \leftarrow s_k \cup s$
7. **do** assign the peers in s_k to the k-th level of the SBTs in a periodic fashion; $\mathcal{S} \leftarrow \mathcal{S} \backslash s$
8. Fill the $L^p_{K,i}$ with the peers that have not appeared in T_i.
9. **return** P distinct SBTs and $\{s_0, s_1, \cdots, s_K\}$.

Fig. 4. The multi-tree extension algorithm for an arbitrary N

used to further upload chunks to the incomplete $L^p_{K,i}$ (which contains exactly $N - 2^{K-1}$ peers) in every T_i. For example, given $N = 20$, we have many strategies to extend the case of $N = 16$. These may include:

1. Increase P_0, P_1, and P_2 all by 1, i.e., $\mathcal{K} = \{0, 1, 2\}$.
2. Increase P_3 to 2: i.e., $\mathcal{K} = \{3\}$.
3. Replace any 4 peers in the fourth level by a repetitive pattern: $\mathcal{K} = \{4\}$.

as well as any arbitrary combinations of the above strategies that increase the size of an ISet by 4 peers. The 4 extra peers added to an ISet are then used to further upload chunks to the partial $L^p_{5,i}$ (including exactly 4 peers) appended to every T_i in the 8th step of the algorithm.

3.3. Distributed Pushing with SNAP

The centralized multi-SBT construction algorithms can only be executed by the server. To be practical for P2P media streaming, our *SNowball multi-tree Pushing* (SNAP), based on the multi-SBT, should operate without each peer knowing the global information about all the SBT trees. More precisely, we need to answer the following three questions:

Q1: How big the neighbor table of a peer can be?
Q2: How to efficiently reconstruct the SBTs upon peer joining or leaving?
Q3: How to deal with bandwidth heterogeneity?

3.3.1. Sizing the Neighbor Table

A complete neighbor table of a peer should contain the children of this peer in different SBTs; it consists of subsets of peers for different trees. For example, as shown in Fig. 2, the neighbor table of peer 1, $\{[5, 10, 16, 12]\}$, has only one subset of peers, while that of peer 11 contains two subsets, $\{[13, 3], [13, 2]\}$. Having such a complete neighbor table at every peer, SNAP can proceed in a distributed way: each peer, upon receiving a new chunk, chooses a subset of peers in a **round-robin** fashion and pushes the chunk to the children in the subset **sequentially**. It is straightforward to see that, for a single SBT (as shown in Fig. 1), the size of a neighbor table is bounded by $\lceil \log_2 N \rceil$, where the bound is obtained at the root. The periodic rotation of peers in every level of the multi-SBT structure inevitably increases this size, but, as shown by the following proposition, this increase is not drastic.

Proposition 2 *The size of the largest neighbor table (owned by the root of each SBT) is $\mathcal{O}\left(\lceil \log_2 N \rceil^2\right)$.*

In practice, peers may have a neighbor table of much smaller size. For example, peers in the $(K-2)$-th and $(K-1)$-th levels only have a neighbor table of size bounded by 3. This shows that SNAP runs correctly with a neighbor table much smaller than $\mathcal{N}$. In other words, its scalability is guaranteed.

3.3.2. Coping with Peer Dynamics

From the server point of view, peer joining and leaving simply reshape the multi-SBT overlay. For peer joining, this reshaping is conducted by an algorithm similar to what is shown in Fig. 4: it basically adapts to the variation in N by adjusting the periods at certain levels. The reshaped multi-SBT is then conveyed to certain peers by updating their neighbor tables.

If a peer leaves SNAP, its starved children in certain SBTs will alert the server. Upon receiving such alerts, the server will assign a parent for these peers at a lower level and also (locally) update their neighbor tables, which actually reduces the level of the corresponding peers. Let us consider the

SNAP shown in Fig. 2, if peer 5 leaves, peers 9, 14, 6 in the second SBT will be starved. As a response to the alerts from them, the server will replace 5 by 9 in the second tree and replace 9 by 13 in both second and fourth trees. Of course, all the edges (uploadings) ending at 5 are removed. It can be easily shown that such local replacements can always maintain the optimality of the multi-SBT structure. It is true that the delay will be increased during this repairing phase, which actually accounts for the fact that the delay obtained in our simulations is not optimal (see Sec. 4). However, the same simulation results also show that SNAP performs much better than the most up-to-date competitor [10], due to the use of the optimal pushing tree and the fact that repairing is always needed for a structured streaming overlay. Also, as SNAP is supposed to act as the backbone of certain hybrid push-pull streaming systems, we expect the repairing to be rare events because a backbone consists of only stable peers.

3.3.3. Heterogeneous Cases

The biggest disadvantage of snowball streaming is its limited compatibility with the heterogeneity in peers' uploading bandwidth. As shown in [11], apart from two very special cases, it is generally impossible to perform scheduling for heterogeneous cases.

To get around this limitation, we propose to unify the bandwidth by "slicing" (through time sharing) the uploading capacity of individual peers. More precisely, for a peer i to joint SNAP, it must at least offer a *baseline uploading bandwidth* B_{base}, i.e., $B_i \geq B_{\text{base}}$, where B_i is the uploading bandwidth of peer i; otherwise the peer has to stay out of the SNAP and be accommodated by a second tier overlay. The value of B_{base} is usually defined by the feature of the streamed media. For example, $B_{\text{base}} = 300$Kbps if the streaming rate is 300Kbps. After joining SNAP, B_{base} is sliced from B_i, while the remaining bandwidth $B_i - B_{\text{base}}$ (if still non-zero) can be used, for example, to upload chunks in the second tier overlay.

We note that our design entails a hybrid and hierarchical system: as not all peers are able to offer B_{base}, those whose uploading bandwidths are scarce need to joint the second tier. They may pull chunks from the server, peers in the SNAP backbone, or among each other, or they may join (sub)tree streaming but receive media contents with lower rate. Those peers inevitably suffer the efficiency-latency tradeoff (for mesh pull, due to the reason explained in [3]) or lower quality (for tree push, due to the lower bandwidth offered by such peers), but this is the price that a system has to pay for accommodating heterogeneity.

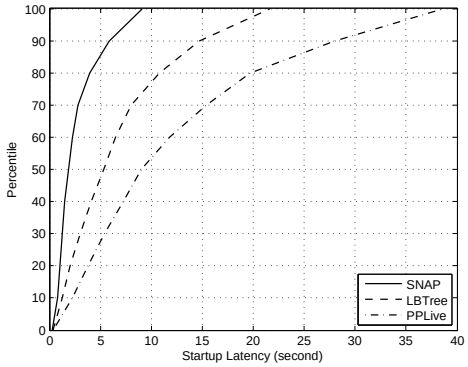
4. PERFORMANCE EVALUATION

We have implemented SNAP along with two other systems, PPLive-like system [14] and LBTree [10], in *ns*-2. While PPLive is a typical mesh-based pull system, LBTree is a tree-based backbone. We use the same parameter settings as those in [10]. We apply two metrics, *startup latency* and *control overhead*, for the comparison. We refer to [10] for the definitions of these metrics.

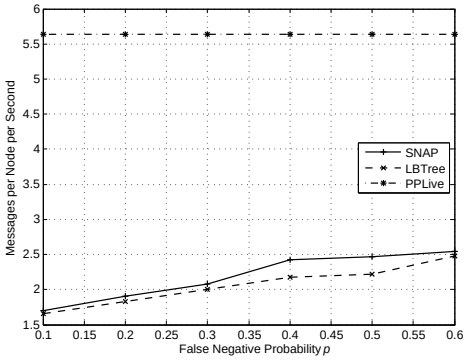
We generate the underlying topology using GT-ITM [15], where the intra-transit bandwidths are 2Gbps and the transit-stub/intra-stub bandwidths are set to 2Mbps, 5Mbps, and 10Mbps with equal probability. The link delay varies from 50ms to 500ms. Only nodes in stub-domains may participate P2P streaming. The session length is 3600 seconds and each chunk is 300Kb (1-second video), and we run 30 sessions for each system with 500 peers. To focus more on the streaming performance, we did not implement the algorithm to identify stable peers, but rather use a probability p to characterize the false negative of an identification algorithm. We use Pareto distribution (mean 100 seconds, $\alpha = 1$) to model the durations of peers that are falsely identified as stable, whereas real stable peers stay until the end.

In Fig. 5 (a), we plot the empirical *cumulative distribution function* (CDF) of startup latency of all the three systems: SNAP, LBTree, and PPLive. It shows that, though LBTree has already achieved a great improvement over PPLive (already shown by [10]), SNAP obtains further improvement over LBTree. We attribute this further improvement to the optimality of the SBT trees used by SNAP. One may complain that the actually delay obtained through simulations are worse than the optimal case (which should lead to a maximum delay of only a few seconds). This is due to the more realistic assumptions in our simulation: certain SNAP peers are falsely identified; they will leave after a short period and these peer leavings will lead to tree repairing (as explained in Sec.3.3.2), which may potentially increase the delay.

We also evaluate the control overhead for the three systems. The overhead of PPLive comes from (i) regular gossiping to maintain the mesh overlay, (ii) regular chunk availability advertisements, and (iii) chunk requests. Both SNAP and LBTree do not have such control overhead, but they need to maintain the backbone, whose overhead varies with the stability of the backbone. In our simulation, we vary the value of p to emulate the stability changes of the backbone: the smaller the value, the more stable the backbone is. As shown in Fig. 5 (b), the control overhead of PPLive remains constant with different values of p, while that of SNAP and LBTree naturally increases with p. Although LBTree performs constantly better than SNAP, the discrepancy in overhead is negligible for small values of p. This is reasonable as SNAP does entail more efforts to maintain its multi-SBT overlay, compared with the loosely coupled multi-tree structure of LBTree. In fact, as the large discrepancy takes place only in very pessimistic cases (more than 50% of the backbone peers are incorrectly identified), we deem SNAP an excellent mechanism to trade overhead for better delay performance.



(a) CDF of startup latency.



(b) Control overhead as a function of p.

Fig. 5. Comparing SNAP with LBTree and PPLive. We only plot the mean values in (b), as the 95% confidence interval is very narrow for every point in the figure due to the large amount of data being collected.

5. CONCLUSION

In this paper, we have focused on the design of the streaming backbone that consists of stable peers. Based on the theoretical results of [11], our SNowbAll multi-tree Pushing (SNAP) applies a distributed chunk scheduling policy guided by a multi-tree overlay, and it guarantees minimum delay of chunk streaming. Using simulations with *ns*-2, we have demonstrated the effectiveness and efficiency of SNAP.

6. REFERENCES

[1] J. Liu, S.G. Rao, B. Li, and H. Zhang, "Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast," *Proceedings of the IEEE*, vol. 96, no. 1, pp. 11–24, 2007.

[2] X. Hei, Y. Liu, and K.W. Ross, "IPTV over P2P Streaming Networks: The Mesh-Pull Approach," *IEEE Comm. Mag.*, vol. 46, no. 2, 2008.

[3] C. Feng, B. Li, and B. Li, "Understanding the Performance Gap between Pull-based Mesh Streaming Protocols and Fundamental Limits," in *Proc. of the 28th IEEE INFOCOM*, 2009.

[4] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowston, and A. Singh, "SplitStream: High-Bandwidth Multicast in Coorperative Environments," in *Proc. of the 19th ACM SOSP*, 2003.

[5] V. Venkataraman, K. Yoshida, and P. Francis, "Chunkyspread: Heterogeneous Unstructured Tree-based Peer-to-Peer Multicast," in *Proc. of the 14th IEEE ICNP*, 2006.

[6] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or Multiple-Tree: A Comarative Study of Live P2P Streaming Approaches," in *Proc. of the 26th IEEE INFOCOM*, 2007.

[7] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming," in *Proc. of the 24th IEEE INFOCOM*, 2005.

[8] B. Li, S. Xie, Y. Qu, G.Y. Keung, C. Lin, J. Liu, and X. Zhang, "Inside the New Coolstreaming: Principles, Measurements and Performance Implications," in *Proc. of the 27th IEEE INFOCOM*, 2008.

[9] M. Zhang, Q. Zhang, L. Sun, and S. Yang, "Understanding the Power of Pull-Based Streaming Protocols: Can We Do Better?," *IEEE J. on Sel. Areas in Communications*, vol. 25, no. 9, pp. 1678–1694, 2007.

[10] F. Wang, J. Liu, and Y. Xiong, "Stalbe Peers: Existence, Importance, and Application in Peer-to-Peer Live Video Streaming," in *Proc. of the 27th IEEE INFOCOM*, 2008.

[11] Y. Liu, "On the Minimum Delay Peer-to-Peer Video Streaming: how Realtime can it be?," in *Proc. of the 15th ACM Multimedia*, 2007.

[12] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, and A. Twigg, "Epidemic Live Streaming: Optimal Performance Trade-Offs," in *Proc. of the 32nd ACM SIGMETRICS*, 2008.

[13] Z. Liu, C. Wu, B. Li, and S. Zhao, "Distilling Superior Peers in Large-Scale P2P Streaming Systems," in *Proc. of the 28th IEEE INFOCOM*, 2009.

[14] X. Hei, C. Liang, J. Liang, Y. Liu, and K.W. Ross, "Insights into PPLive: A Measurement Study of a Large-Scale P2P IPTV System," in *Proc. of the IW3C2/ACM WWW-IPTV*, 2006.

[15] "GT-ITM," http://www.cc.gatech.edu/projects/gtitm/.