NSYNC: Network Synchronization for Peer-to-Peer Streaming Overlay Construction

Hongbo Jiang
Division of Computer Science
Case Western Reserve University, Cleveland, OH 44106-7071
hongbo.jiang@case.edu

Shudong Jin
Division of Computer Science
Case Western Reserve University, Cleveland, OH 44106-7071
shudong.jin@case.edu

ABSTRACT

In peer-to-peer streaming applications such as IP television and live shows, a key problem is how to construct an overlay network to provide high-quality, almost real-time media relay in an efficient and scalable manner. Much work has focused on the construction of tree and graph network topology, often based on the inference of network characteristics such as delay and bandwidth. Less attention has been paid to improving the liveness of media delivery, and to exploiting the flexibility of applications to construct better overlay networks. We propose the NSYNC, an ongoing work on constructing low-latency overlay networks for live streaming. It aims at solving the following problems. In typical applications, peers must buffer a portion of a real-time event, e.g., for at least a few seconds, to limit the impact of adversary network conditions. Thus, it introduces both (1) delay, especially long delay for peers that are many hops away from the origin servers, and (2) partial ordering between the peers. With NSYNC, the application media players can slightly increase or decrease the speed of playing media. Thus, the peers in a network can be synchronized to achieve two effects. First, late peers can catch early peers and the origin server such that the entire peer networks improve liveness. Second, the client/server roles between a pair of neighboring peers can be reversed, allowing opportunities for constructing more efficient overlay networks. NSYNC can be used in various peer-to-peer streaming systems.

1. INTRODUCTION

Peer-to-peer computing paradigm has provided scalable solutions to many network applications, and a large portion of Internet traffic is from peer-to-peer applications [14]. One of them is peer-to-peer streaming, i.e., the delivery of continuous media to a large number of users simultaneously. The media objects can be live or stored media, for example, Internet television and movies, live shows, and conference broadcasting. With peer-to-peer streaming, the peers (usually end hosts) in the network can relay media data to other peers. This computing paradigm capitalizes on the up-link bandwidth of the peers, and effectively alleviates the potential bottleneck problem at the origin servers of the media objects. One key problem in peer-to-peer streaming is how to construct an overlay network to provide high-quality, low-latency media relay, in particular for live streaming applications [15]. Early peer-to-peer streaming systems and techniques [3, 1] were focused on constructing application layer multicast trees. For example, the construction of multicast trees is often based on the inference of network characteristics such as end-to-end delay, loss rate, and bandwidth, and based on the relative proximity between peers [12]. Such practices are very beneficial to providing high-quality, real-time media stream in an efficient manner.

The limitations of tree overlay topology have been addressed in several recent studies [5, 13, 18, 10, 7]. It is vulnerable to peer failures and prone to disrupted services. In particular, if the peers are more dynamic and they leave/join the network more frequently, many other peers in the subtrees could be affected. To overcome these limitations, new multi-parent approaches were proposed. A peer can fetch media data from multiple peers (i.e., multiple parents). This method usually improves the quality of media stream. If one of the parents fails, the peer can still retrieve media data from other parents, as long as the aggregate rate is roughly equal to the stream playback rate.

However, this approach also introduces new problems, for example, how to choose multiple parents for a peer. While many of these problems have been addressed, we find one key problem is largely neglected. In many streaming media applications, a receiver often buffers a portion of a media stream, say for a few seconds, before the application continuously plays it. This is necessary in order to limit the impacts of packet losses, end-to-end delays and delay jitter. In peer-to-peer streaming, a peer's startup latency is decided by the latency of its parent(s), plus the latency necessary to buffer a portion of the media stream. If the overlay network becomes large and there are many hops between a peer and the origin server, then the peer may observe a huge latency. It has been observed, in real peer-to-peer streaming systems such as CoolStreaming and PPLive, the latency can be up to several minutes. While a long latency rarely reduces a user's interests in watching a movie, it does cause user dissatisfactions if for a live sports event, a peer-to-peer streaming system lags well behind a Web-based score reporting system.

Such undesirable characteristics are mainly due to one key assumption: the startup latency of a peer (the lag behind the origin server) cannot be simply reduced once introduced. There are several consequences. First, the entire network may observe a longer and longer average latency. To elaborate on this, just imagine that some early peers can leave and late peers (with long latency) will become the parents of new arriving peers. Second, there is a fixed partial ordering within the set of peers in the system. This limits the design space for overlay construction, and the network may

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
NOSSDAV '06 Newport, Rhode Island USA
Copyright 2006 ACM 1-59593-285-2/06/0005 ...$5.00.

not be optimized. We will also elaborate on this in the next section. On the other hand, if we can break this above assumption, we may greatly improve the overlay network construction and provide good liveness of events to the peers. In this paper, we show this is possible and is beneficial to do so. We propose NSYNC in which, peers can alternate their playback speeds slightly to reduce the lag behind the origin server, and multiple neighboring peers can be synchronized before the client/server roles are reversed. In this way, NSYNC can help reduce the average latency of peers and can result in more efficient overlay network construction. While this paper shows only NSYNC's two operations, catching and reversal, based on alternating playback speeds, it is possible to build more complex operations to further improve overlay construction. Our preliminary results reveal that NSYNC can be effective. The remainder of the paper is organized as follows. In the next section, we first illustrate motivations behind NSYNC. In section 3 we describe the algorithms in NSYNC. In Section 4 we provide an evaluation and the preliminary results. Section 5 briefly describes related work before we conclude this paper with open questions.

2. MOTIVATION

Consider in a peer-to-peer streaming system, peer B receives media stream from its parent, peer A. If a peer can have multiple parents, our arguments can be generalized too. To tolerate network delays and delay jitter, a peer first prefetches a portion of the media stream before continuously playing it. Thus, both peers maintain buffers to keep prefetched media data, as shown in Figure 1. The shaded area indicates the prefetched and buffered media data. Let T denote the current time. Let $T_A < T$ be the latest portion of the media stream that has been prefetched by A, and let $T_B < T$ be the latest portion of the media stream that has been prefetched by B. Since A is the parent of B, we have also $T_B < T_A$. Let δ_A be A's lag behind the origin server, and let δ_B be B's lag behind the origin server.

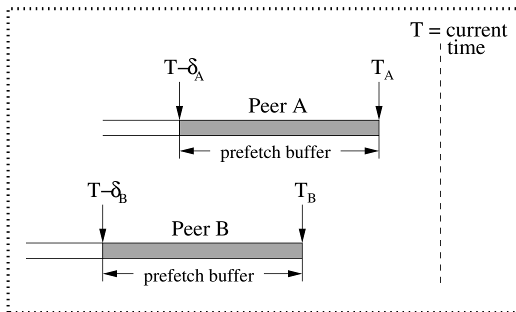


Figure 1: In peer-to-peer streaming, peers need to buffer a portion of the media stream and this introduces startup latency.

2.1 Long Latencies

Initially, B will prefetch data from A and buffer it. For the simplicity of discussion, assume both peers need to buffer the same amount of data. Often the prefetching and buffering can introduce additional delay to B no matter what strategy is used. Let us consider two cases: (1) peer A starts to relay the bytes to B immediately when they are played, and (2) peer A starts to relay the bytes to B immediately when they are received.

In the first case, the difference between δ_A and δ_B depends on how fast B can receive data from A at the time. For example, in order to buffer a 5-second segment and the available bandwidth between the two peers is equal to the nominal media rate, then $\delta_B \geq \delta_A + 5$ seconds. Once peer B starts to play the media stream while peer A continues to play it, the difference between δ_A and δ_B will remain constant. However, assume before peer B joins, peer A had two other child peers who consumed most of A's up-link bandwidth. When B joins, network bandwidth is so scarce that a long latency is introduced by B. At some time later, A's other children have left the system, but B still lags way behind A.

Things could be even worse if the peers can join and leave the system dynamically. Assume another peer C joins the system and chooses B as its parent. This new peer will observe an even longer latency, $\delta_C > \delta_B$. Furthermore, if A leaves and joins the system later again, this new A may choose C as its parent. In this way, early peers leave the system, and new peers introduce longer latencies. This process may repeat, and to the worst case, may cause the average latency of the entire system to increase without being bounded. Finally, one may argue that, in practice a new peer may choose other peers with low latency as the parents. Although the problem of unbounded latency can be alleviated, low-latency peers are more likely to be overloaded.

In the second case, peer A starts to relay the bytes that are just received. If the network conditions are perfect, i.e., there is no packet loss and the available bandwidth is always higher than the nominal media rate, then the two peers observe only a slight difference between their latencies, the propagation delay from A to B. This would provide good liveness, although theoretically dynamic joining/departure of peers may still cause an unbounded latency. However, there are at least two more reasons why additional delay must be introduced. First, packet losses exist. If the peers relay the packets immediately when they are received, packet losses are cumulative. A peer far from the origin server might observe an extremely high loss rate and the quality of media would degrade significantly. Therefore, retransmission (and the delay introduced by retransmission) should be considered. Buffering and repairing a media segment (i.e., adding delay) before relaying it seems to be a reasonable design choice. Second, available bandwidth is limited. When peer A relays the media to B, peer A's up-link bandwidth might be limited at the moment such that longer time is required to relay a fixed-length segment. For example, assume both A and B buffer a 5-second segment. While A is downloading the media from its own parents, it must also relay it to B at the nominal rate. However, it is likely that there is not enough up-link bandwidth, noticing that A could be downloading the media at its full link capacity and now a new connection is created. Let us say, it takes 10 seconds to fill peer B's 5-second buffer. When B can eventually start to play the media, it is already 5 seconds behind A.

2.2 Inefficient Overlay Construction

While long latencies are not desirable for users in some applications, e.g., streaming of live sports events, the partial ordering for the set of all peers are not good for efficient overlay network construction. By partial ordering, we mean for any pair of peers, one of them always precedes the other. It is possible for the later peer to fetch data from the earlier peer, but not the other way around. Consider the scenario shown in Figure 2. Assume peer S joins the system first and peer A chooses S as its parent. At a later time, peer B also joins the system. Although B is very close S, e.g., in term of peer-to-peer delay, for some reason B chooses A as its parent. This could happen for various reasons. For example, B was not able to discover S, or S has no abundant bandwidth to serve another peer in addition to A. As the result, the media stream may be transmitted coast-to-coast from S to A, and then coast-to-coast back from A to B. Clearly there is a more cost-efficient overlay

construction: S can send the media stream to B, who relays the stream to A. This second construction not only reduces the average latency observed by the peers, but also will likely reduce network resource consumption. However, the partial ordering among the peers makes this impossible.

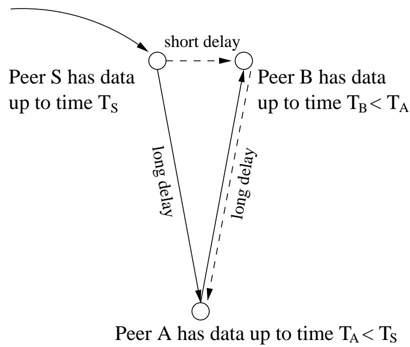


Figure 2: Due to the partial ordering among the peers, $T_B < T_A < T_S$, they cannot have a more efficient overlay construction as shown by the dashed line.

Above weaknesses of typical peer-to-peer streaming systems are due to the following assumption: *the startup latency introduced by a peer can be increased but cannot be decreased*. Adversary network conditions may cause the latency to be increased, e.g., when there is a sudden network congestion and the buffer is drained. However, a user wants to continuously and smoothly play a media stream, so the latency cannot be simply reduced, for example, by skipping a segment of the media stream. Therefore, providing some mechanisms to reduce the startup latency while allowing the user to smoothly play the media stream is a challenge. If this problem is solved, we can explore new design space for constructing more efficient overlay network and designing better peer-to-peer streaming systems.

3. NSYNC

NSYNC uses a simple idea to break the assumption of fixed playback latency for a peer. It assumes the applications such as media players have flexibilities. The user associated with NSYNC-enabled peer can slightly increase the speed of playing the media stream. This increased speed is $(1 + \epsilon)$ times the nominal media playback rate, where ϵ is usually a small fraction. For example we may set $\epsilon = 5\%$, such that the user will hardly perceive any difference. The feasibility of this idea is as follows. First, technically it is easy to increase or decrease the playback speed. For example, commercial media players such as Windows Media Player allow users to configure/modify the software to achieve this goal. Second, the slight change in speed will unlikely affect the quality of media stream. For example, in the context of video-on-demand application, piggybacking techniques [4] also slightly increase the playback speed (see also ample evidence therein)

When a peer is playing the media stream in this higher speed, we say it is in the *accelerated* mode. The peer can stay in this mode for a while to reduce the latency between itself and other peers (e.g., its parents). Although the increased speed is only slightly higher, after a while the peer can effectively reduce the playback latency. Eventually the peer will resume the normal operation with a smaller latency.

NSYNC provides a set of primitives to the peers such that they can simply call the primitives to enter or leave the accelerated mode. To make NSYNC useful, we need to consider in what scenarios, NSYNC should be used to improve overlay network construction. Furthermore, once a scenario is identified, there still remain many other questions. For example, first, when and how should the peers enter and leave the accelerated mode? Second, how can a set of peers coordinate with each other if they are in the accelerated mode? For example, the buffer can be drained quickly in the accelerated mode. Third, what if one of multiple cooperating peers in the accelerated mode leaves the system? To answer these questions, let us consider two simple scenarios where NSYNC is useful, and we consider how the peers cooperate with each other to achieve the goals.

3.1 NSYNC for Catching

In the simpler scenario, a peer will enter the accelerated mode to reduce its lag behind its parent(s). Figure 3 intuitively shows how it works with two peers A and B. Peer B receives media stream from peer A. Initially, the difference between their playback latencies δ_A and δ_B is large enough to guarantee a smooth play by peer B. However, at some later time the peers discover that this difference (and the buffer) is too large, and peer B likes to have better liveness. Therefore, B will enter the accelerated mode. Assume B is still receiving media data from A at the nominal rate. At some time later, B finds the buffer contains only a small portion of the media stream. Further decreasing it would risk a low-quality playout the media stream. Hence, B cancels the accelerated mode, see Figure 3(b).

This catching process is easy for implementation. From above description, peer B does not request any assistance from A. It needs only to decide by itself if a portion of media stream has arrived in time for playout. Even if this scenario becomes more complex, this caching process still works well. Consider two more complex cases. First, assume B is the parent of another peer C, and B has reduced its playback latency after catching. C can still receive media data from B as long as B keeps the played portion of media stream in its local memory (which is inexpensive). This works well even if both B and C are in the accelerated mode. Second, assume B has multiple parents. B will see if it can receive media data from all parents in time for playout. The potential of the caching process is therefore dependent on the slowest parent.

By slightly modifying this catching process, we can achieve even better results. We have assumed that B always receives media data from A at the nominal rate. In the catching process, B can receive data at $(1+\epsilon)$ times the nominal rate. Thus, the buffer of B will not be drained, but it will contain more up-to-date media data. After a while, δ_B will be closer to δ_A and T_B will be closer to T_A too. See Figure 3(c). This second catching process will ensure B always has abundant media data in the buffer, but it requires there is also higher bandwidth between A and B.

3.2 NSYNC for Reversal

In the second scenario, two peers will reverse their client/server roles using NSYNC operations. Figure 2 shows two peers A and B to perform this *reversal*. Assume at time $T = 0$, A has media data up to time T_A and its playback latency is δ_A. B has media data up to time T_B and its playback latency is δ_B. Figure 4(a) shows the initial state. At time 0, although it is impossible for S to send media data to both A and B, the peers discover there is a better overlay construction: S sends media data to B and then B relays it to A.

Peer B will enter the accelerated mode to catch A first. It receives media data from A at a rate equal to $(1+\epsilon)$ times the nominal rate. In the meanwhile, peer A will run at usual. Assume there is a propagation delay d between A and B, and for the simplicity of de-

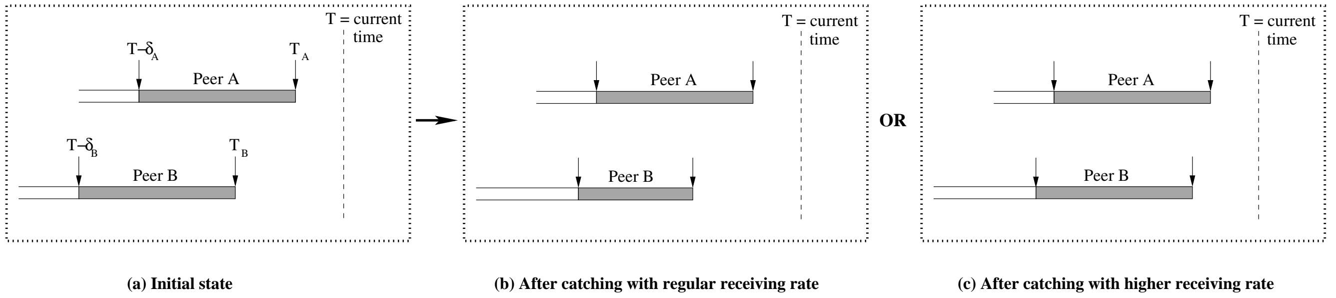


Figure 3: Peer B catches its parent, peer A by entering the accelerated mode. B may either receive data at the nominal rate to reach the state shown in (b), or receive data at a higher rate to reach the state shown in (c).

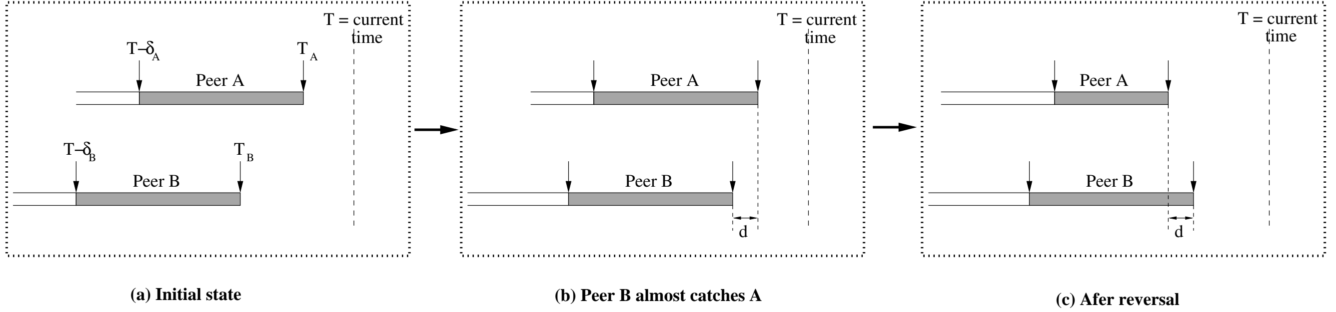


Figure 4: Peer A and peer B reverse their client/server roles. Peer B first enters the accelerated mode to catch A.

scription, we assume path symmetry. At time $(T_B - T_A - d)/(\epsilon R)$, where R is the nominal media rate, peer B *almost* catches A except there is a propagation delay d. See Figure 4(b). At this point of time, A stops fetching data from S. B also stops fetching data from A, but instead it resorts to A for the next portion of the media stream. Assume B receives data from S at the nominal media rate. At time d later, B virtually catches A. Within another delay d, A will be able to receive new media data from B. The client/server roles are reversed. Both A and B resume their normal operations. See Figure 4(c).

The above reversal process assumes that peer A buffers a portion of media stream whose duration is much larger than $2d$. With this assumption, peer A will be able to play the buffered media data while switching from S to B as its parent. In many peer-to-peer systems, a peer usually keeps a segment whose duration is at least a few seconds, i.e., it is long enough. However, if A has not kept enough data, we can still solve the problem. The idea is, A can enter a *decelerated* mode: it plays the media stream at a slightly lower speed. For example, the speed can be $(1 - \epsilon)$ times the nominal rate. If necessary, A can enter the decelerated mode before the reversal process starts, so that the duration of the buffered media stream will be long enough.

Several other problems remain and here we briefly discuss how to solve them. (1) First, we need to decide when to start the reversal process and who will initiate it. This can be done periodically, and decided by the peers locally. A peer can send queries to its neighbors to check if any two neighbors should be connected directly (if that is more efficient). Peer-to-peer delay is a possible metric in deciding the peering relationships. The neighbors can exchange probe packets to estimate the delay. The peer who sends out the initial queries will collect the delay information and initiate the reversal process if applicable. (2) Second, if some peers are already involved in the reversal process, we need ensure the correctness of this process. To simplify it, we can *lock* the involved peers so that one peer can participate in only one reversal process at the same time. If any peer leaves during this process, the process is immediately aborted and all other peers resume the normal operations. (3) Third, when peers leave and join the systems frequently, the reversal process may have to be canceled often. Thus, it incurs some unnecessary overhead for no good. For this reason, we can start the reversal process only after the peers have stayed online for some reasonably long period. This method will likely reduce the chance of aborted reversal. For example, characterization of live streaming workload has revealed that the users tend to stick to the live events when their interests have grown [17]. Therefore, if the peers have stayed online for a longer period, they are increasingly unlikely to leave the system soon.

4. PRELIMINARY EVALUATION

In this section we focus on the preliminary evaluation of using NSYNC for constructing locality-aware peer-to-peer streaming systems, and demonstrate the effectiveness of NSYNC. We have implemented a simulator of NSYNC. For comparison, in the simulator we also implement other peer selection techniques without using NSYNC. We compare the performance of various techniques in terms of the average parent-child delay, which indicates on average how far a peer is from its parents. In this preliminary evaluation, NSYNC uses only the reversal of client/server roles to let peers receive media stream preferably from close parents.

4.1 Simulation Setup

We use the BRITE topology generator [8] to generate a set of one level network topologies with between 500 and 2000 nodes. The default setting of the topology generator is used. We have found on average each node can have 4 to 10 neighbors at any scale. In each topology we use all-pairs shortest path algorithm to compute

the end-to-end delay between the peers, and normalize the delay to a maximum of 1000ms. In addition, we set a delay threshold of 600ms. If the delay between two peers is larger than this value, the system presumes it is unlikely to have the desired QoS characteristics. As a result, these two peers will not establish a peering relationship.

The peer behavior in our simulation is modeled as a birth-death process, although this may not the most accurate. Initially, no peer is turned on and connected to the origin server. A peer may be turned on and off by its user. A non-operational peer is turned on with probability 1% on every time tick, and, an operational peer is turned off with probability 0.2% on every tick. The length of time tick in the system is 10 seconds. The average duration of a peer is therefore over one hour. In order to obtain accurate results, the simulator ran for longer than 1000 time ticks, or about three hours.

When a peer joins the overlay network, it obtains 10 connections information received from the server or by other means, any of which may or may not satisfy the QoS in terms of delay. In addition, we set each peer can have up to 5 parents and can support up to 5 children. Without overlay reorganization, peers only select nearby up to 5 parents from the first 10 possible connections with the lowest delay. With NSYNC, peers can reverse client/server roles and improve the overlay connectivity periodically as described in Section 3. The message overhead is neglected.

4.2 Simulation Results

We begin by examining the effects of different techniques in building locality-aware overlay networks. Figure 5(a) shows that first, compared with random parent selection, the reorganization can noticeably improve the locality of the result overlay. In this 1000-peer scenario, NSYNC reversal achieves more than 10% improvement compared with random selection. This is because after each reversal process, the new neighbors of a peer likely become closer neighbors. We should note here that while this improvement is still limited, it is mainly due to that we only attempt to reverse the client/server roles of immediate peers. If NSYNC implements more complicated overlay construction, we expect the improvement will be more obvious.

In the next experiments, we investigate the performance in overlay networks with different scales. Here we use averages of 500 simulation runs after the 500th time tick where the system performance becomes stable. Figure 5(b) shows that even with different scales, the relative improvement of NSYNC reversal technique is consistent. One might notice that the absolute performance of the two techniques are different when the network size varies, e.g., the average delay is the lowest with 1000 nodes. This is due to the randomness of the simulation, for example, the BRITE topology generator may generate links with lower parent-child delay. In summary, we suspect if the network further scales up and the delay becomes more heterogeneous as in the Internet, then the advantages of using NSYNC reversal will be much more obvious. We will exploit this in the future.

Finally, we study the impact of the delay threshold on the performance of the techniques. When a peer joins the overlay network, it selects its parents using a delay threshold to decide whether link can be established. Figure 6 shows the effect of the delay threshold on the average parent-child delay. Again, the figure shows the average quantities of 500 simulation runs. We can see that in general, when the delay threshold is larger, the peers will observe long initial delay. Therefore, NSYNC will potentially reduce the average parent-child delay more drastically.

To summarize, the use of NSYNC can be effective in providing low-latency peer-to-peer streaming. Our preliminary results show some improvement, although we we have exploited only the NSYNC reversal operations. It is important to notice that, NSYNC primitives do not conflict with the techniques in current peer-to-peer streaming techniques and systems. Hence, NSYNC can be adopted together with them.

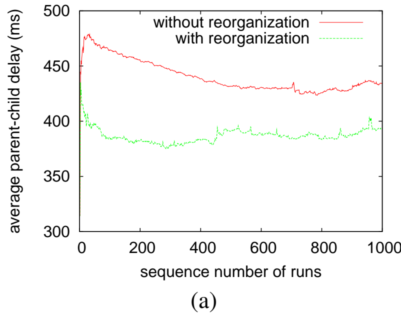
5. RELATED WORK

Early work on peer-to-peer media streaming was based on a single multicast tree. Representative systems include End-System Multicast [3], NICE [1], and ZIGZAG [16]. In End-System Multicast, in order to build a multicast tree, a complete virtual graph is constructed. This graph consists of all participants, with a virtual link (with distance) between any pair of vertexes. A minimum spanning tree is then constructed from this complete virtual graph. In NICE, the peers are first organized into a hierarchy to support a large large receiver sets. ZIGZAG also organizes the peers into an appropriate tree. This tree has a height logarithmic with the number of clients. However, when peers in the tree do not have sufficient available capacities due to transient congestion, or when they depart or fail, the streaming session is interrupted and requires expensive reconciliation work. To address this problem, streaming based on multiple multicast trees has been proposed [9, 2]. The media can be split into multiple sub-streams, each delivered along a multicast tree. As a result, it is more robust against peer departures and failures. An affected receiving peer may still be able to continuously play the media stream at a degraded quality, while waiting for the tree to be repaired. These advantages come with an expensive cost, however, as all the trees need to be maintained in highly dynamic peer-to-peer networks.

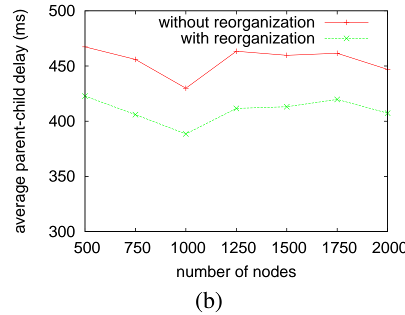
More recent systems such as DONet [18], PRO [13], Chainsaw [10], and DagStream [7] construct networks with stronger connectivity. Meshes and directed acyclic graphs are constructed to for overlay networks. They allows a multi-parent approach to limiting the impacts of peer dynamics and network dynamics. Several new systems have taken this multi-parent approach. For example, in CoolStreaming [18], each peer periodically exchanges the availability (and timeliness) information of the media stream with multiple neighbors. The media segments to be retrieved from each neighbor are dependent upon the number of potential suppliers for each segment and the available upload capacities of neighbors. In PeerStreaming [6] and PPlive [11], the media workload is distributed among the set of supplying peers in proportion to their upload capacities. These heuristics fall short of achieving global optimality, and thus may starve the peers with high download demands. Furthermore, the construction and maintenance of an overlay network (with multiple parents) require often much higher overhead. It was also found that in both CoolStreaming and PPLive, peers experience extremely long latency, presumably due to the buffering requirement. To the best of our knowledge, no prior peer-to-peer streaming techniques or systems have attempted to use peer synchronization to improve liveness of streaming and to improve efficiency of overlay network.

6. CONCLUSION

We have illustrated the weaknesses of current peer-to-peer streaming techniques and systems to support delivery of live stream media. We have proposed the NSYNC to construct low-latency overlay networks for live streaming. With NSYNC, the applications (such as Windows media player) can slightly increase or decrease the speed of playing media stream. Thus, the peers in a network can be synchronized by calling our NSYNC primitives. We have shown how to use NSYNC to construct more efficient overlay net-



(a)



(b)

Figure 5: Average parent-child delay in networks (a) with 1000 nodes, and (b) with different numbers of nodes.

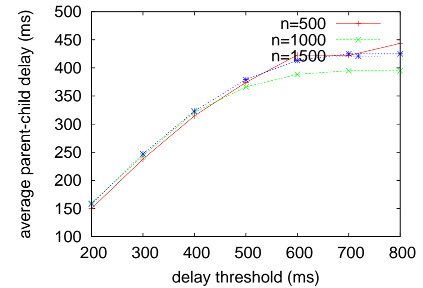


Figure 6: Effect of delay threshold d on the performance of NSYNC reversal.

works via two examples: *catching* and *reversal*. With catching, a peer can accelerate its play-out to reduce its lag behind its parent (as well the origin server). With reversal, a pair of peers can reverse their client/server roles, if both increase and decrease the play-out speed. The use of both catching and reversal can improve the liveness of the entire peer-to-peer network, and make the overlay network more efficient. Our preliminary results show NSYNC can be effective.

NSYNC is generally useful for two reasons. First, NSYNC provides a set of primitives to build more complex operations. Catching and reversal are two example operations, and we believe peer synchronization can be useful for other purposes. Second, NSYNC can be used together with various peer-to-peer streaming techniques, and can be adopted by various systems.

NSYNC is an effort to explore the new design space for peer-to-peer streaming systems. There are many open questions related to the algorithms, implementation, and deployment of NSYNC which we would like to further exploit and discuss. From the algorithm perspective, for example, given an overlay network, how can we use NSYNC operations (such as catching and reversal) to reorganize the overlay network? What is the complexity of the optimal solutions and are there good heuristics for approximation? From the implementation perspective, what primitives should be provided by NSYNC, and what operations can be developed from the primitives? From the deployment perspective, are there any incentives for the users to enable NSYNC, and are there any technical/non-technical barriers for its deployment?

7. REFERENCES

[1] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proceedings of ACM SIGCOMM*, August 2002.

[2] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. I. T. Rowstron, and A. Singh. SplitStream: high-bandwidth multicast in cooperative environments. In *Proceedings of ACM SOSP*, October 2003.

[3] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS*, June 2000.

[4] L. Golubchik, J. C. S. Liu, and R. R. Muntz. Adaptive piggybacking: A novel technique for data sharing in video-on-demand storage servers. *ACM Multimedia Systems Journal*, 4(3):140–155, 1996.

[5] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. PROMISE: Peer to peer media streaming using collectcast. In *Proceedings of ACM MULTIMEDIA*, 2003.

[6] J. Li. PeerStreaming: A practical receiver-driven peer-to-peer media streaming system. Technical Report MSR-TR-2004-101, Microsoft Research, September 2004.

[7] J. Liang and K. Nahrstedt. Dagstream: Locaity aware and failure resilient peer-to-peer streaming. In *Proceedings of S&T/SPIE Conference on Multimedia Computing and Networking (MMCN)*, January 2006.

[8] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An approach to universal topology generation. In *Proceedings of IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, August 2001.

[9] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *Proceedings of International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, May 2002.

[10] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *Proceedings of International Workshop on Peer-to-Peer Systems (IPTPS)*, February 2005.

[11] PPLive. http://www.pplive.com/en/index.shtml.

[12] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proceedings of IEEE INFOCOM*, June 2002.

[13] R. Rejaie and S. Stafford. A framework for architecting peer-to-peer receiver-driven overlays. In *Proceedings of International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, June 2004.

[14] S. Saroiu, P. K. Gummadi, and S. D. Gribble. Measuring and analyzing the characteristics of Napster and Gnutella hosts. *ACM Multimedia Systems Journal*, 8(5), 2002.

[15] K. Sripanidkulchai, A. Ganjam, B. Maggs, , and H. Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. In *Proceedings of ACM SIGCOMM*, August 2004.

[16] D. A. Tran, K. A. Hua, and T. T. Do. ZIGZAG: An efficient peer-to-peer scheme for media streaming. In *Proceedings of IEEE INFOCOM*, April 2003.

[17] E. Veloso, V. Almeida, W. Meira, A. Bestavros, and S. Jin. A hierarchical characterization of a live streaming media workload. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop (IMW)*, November 2002.

[18] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. Coolstreaming / DONet: A data-driven overlay network for live media streaming. In *Proceedings of IEEE INFOCOM*, March 2005.