Tweeting Videos: Coordinate Live Streaming and Storage Sharing*

Xu Cheng
School of Computing Science
Simon Fraser University, BC, Canada
xuc@cs.sfu.ca

Jiangchuan Liu
School of Computing Science
Simon Fraser University, BC, Canada
jcliu@cs.sfu.ca

ABSTRACT

User generated video sharing (e.g., YouTube) and social-networked micro-blogging (e.g., Twitter) are among the most popular Internet applications in the Web 2.0 era. It is known that these two applications are now tightly coupled, with many new videos being tweeted among Twitter users. Unfortunately, video sharing sites are facing critical server bottlenecks and the surges created by Twitter followers would make the situation even worse. To better understand the challenges and opportunities therein, we have conducted an online user survey on their personal preference and social interest of Internet video sharing. Our data analysis reveals an interesting coexistence of live streaming and storage sharing, and that the users are generally more interested in watching their friend's videos. It further suggests that the users are willing to share their resources to assist others with close relations, implying node collaboration is a rationale choice in this context.

In this paper, we present COOLS (Coordinated Live Streaming and Storage Sharing), an initial attempt toward efficient peer-to-peer tweeting of user-generated videos. Through a novel ID code design that embeds nodes' locations in an overlay, COOLS leverages stable storage users and yet inherently prioritizes living streaming flows. Preliminary evaluation results show that, as compared to other state-of-the-art solutions, COOLS successfully takes advantage of the coexistence of live streaming and storage sharing, providing better scalability, robustness, and streaming quality.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications

General Terms

Design, Human Factors, Performance

*This research is supported by a Canada NSERC Strategic Project Grant, an NSERC Discovery Grant, an NSERC DAS Grant, and an MITACS Project Grant.

NOSSDAV'10, June 2–4, 2010, Amsterdam, The Netherlands.

Keywords

Live Streaming, Storage Sharing, Social Network

1. INTRODUCTION

In the recent years, Web 2.0 has been dominating the Internet as well as mobile networks. Many such applications are backed by vast user-generated content (UGC); typical examples include YouTube[1] for video sharing and Facebook[2], Twitter[3] for social networking. The pervasive penetration of wireless LANs and 3G networks further stimulates the development of such applications, where the content can be generated at anytime and anywhere.

YouTube, established in 2005, is now serving well over a billion views a day [11]. Watching online video has no doubt become one of the most important entertainments in people's daily life. Through Twitter, a user can post text-based status called *tweet*, and the user's subscribers called *followers* can receive the tweet update in real-time. So far, people have already posted more than 10 billion tweets[4] and the number is still fast growing. More importantly, the two Web 2.0 applications have close interactions, e.g., YouTube enables an automatic post through Twitter, so that the followers can quickly be notified and then watch a new video. New feed applications such as FriendFeed[5] (by Facebook) and recently introduced Google Buzz [10] are trying to further couple video sharing and social networking.

Unfortunately, YouTube-like sites are facing critical server bottlenecks and the surges created by Twitter followers would only make the situation worse [4]. In fact, even the text-based Twitter has encountered system-wide outages during some critical events, e.g., the Obama's inauguration and Michael Jackson's tragical death [1, 9]. While peer-to-peer has long been advocated as a solution for TV or movie content streaming, it remains unclear whether it is doable for the short user-generated videos with independent asynchronous viewers.

To better understand the challenges and opportunities therein, we have conducted a user questionnaire survey on their personal preference and social interest of Internet video sharing. Our data analysis reveals an interesting coexistence

[1]http://www.youtube.com
[2]http://www.facebook.com
[3]http://twitter.com
[4]The IDs of tweets are assigned sequentially. With the Twitter API (http://twitter.com/statuses/show/10000000000.xml), we can find the latest posted tweet.
[5]http://friendfeed.com

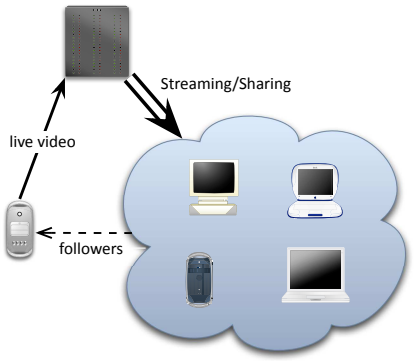

Figure 1: Application Scenario

of live streaming and storage sharing; that is, some users expect to watch the video immediately while some expect later. As illustrated in Figure 1, a user could use a digital camcorder or mobile phone to record video, and the live video is then sent to a server. Through Twitter-like services, the server can broadcast the live video to the user's followers, who can be wired Internet user or mobile device users. A follower has three options:

- Watch the live video, and thus have stringent requirement on streaming quality;
- Do not watch the live video, but expect to watch it later. Such a user is delay-tolerant, and some of them may also switch to the first option at some time during the live streaming;
- Not interested in the video at all. Some celebrities or organizations have thousands of followers and their followers probably do not know each other. In this case, if such a user does not want to watch the video now or later, he/she may not want to share the resources with other followers, either.

The coexistence clearly makes a system design more complicated. It however also suggests that semi-synchronized user for live streaming may reach a critical mass for collaborative streaming, and that the relatively stable storage nodes could be leveraged to combat node churns. More importantly, our survey reveals that many of the users are willing to share their resource to assist others with close relations. Also, although people are not necessarily fully satisfied with the playback quality provided by most of the current video streaming services, their concern is more about the video content, which largely determines the viewing duration. Consequently, if their friends upload videos, they will be more interested and likely watch the entire video. All these imply collaborative peer-to-peer is a rationale and promising choice in this context.

In this paper, we present COOLS (Coordinated Live Streaming and Storage Sharing), an initial attempt toward efficient peer-to-peer tweeting of user-generated videos. Through a novel ID code design that embeds nodes' locations in a tree overlay, COOLS leverages stable storage users and yet inherently prioritizes living streaming flows with short startup delay. It also gracefully accommodates users' switch from the second option (storage) to the first one (live streaming), as well as node dynamics.

Preliminary evaluation results show that, as compared to other state-of-the-art solutions, COOLS successfully takes advantage of the coexistence of live streaming and storage sharing, providing better scalability, robustness, and streaming quality.

2. RELATED WORK

Web 2.0 applications are emerging in the recent years, and there have been quite a few related measurement studies, particularly on understanding YouTube for video sharing [3, 4, 5, 8], Facebook, MySpace and LinkedIn for social networking [2, 8], and etc. Different from these measurements that indirectly infer user behavior and interests, we have conducted an online questionnaire survey that directly obtains such data. The survey also motivates our study on the coexistence of live streaming and storage sharing, and implies that peer-to-peer is a rationale choice in this new context.

Numerous peer-to-peer protocols have been developed for live or on-demand video streaming, which can be broadly classified into two categories according to their overlay structures [7], namely, tree-based (e.g., ChunkySpread [13]) and mesh-based (e.g., CoolStreaming [14]). We have seen earlier attempts toward joint live and on-demand peer-to-peer streaming. For example, BitTorrent has enabled a streaming mode, so that user could watch on-demand video while downloading it [12]; also, peer-to-peer streaming platforms such as PPLive now provide both live and VoD modes [6]. However, efficient implementation and more importantly, seamless integration of the two types of users, particularly in the social network context, remains a great challenge.

3. A USER QUESTIONNAIRE SURVEY

Most of existing studies on video sharing measure the network usage to derive user-related statistics. Trying to directly understand the Internet users' preference and social interests on viewing and sharing online videos, we instead create a Web survey and invite people worldwide to fill it in. The survey contains 13 single-choice questions plus several questions on personal information. So far, 114 people have participated the survey. 58.8% of them are from North America, 33.3% from Asia, and 7.0% from Europe, with various network connections, refer to Figure 2. Most of them are of ages 19 and 30, which is exactly the core generation of YouTube and Twitter users.

The survey questions and detailed results can be found at `http://netsg.cs.sfu.ca/survey.html`. We now summarize the key observations we have seen from the survey results. We find that 62% participants usually leave a video streaming session after selecting it and return after a while, rather than stay and wait, and most of them (84.2%) consider playback quality as the key factor of this behavior (55.2% and 61.4% are satisfied with the startup delay and playback continuity, respectively). Then, if they leave, 46.3% of them will come back after a quarter of the video has been downloaded, and 68.3% will come back after half of the video has been downloaded; 22.0% of them will wait until the entire video is downloaded. Regarding waiting time, most of them (69.2%) will spend less than 5 minutes for waiting, and no one will wait more than 30 minutes. In short, for the same video content, viewers of live streaming and that store-and-play both exist.

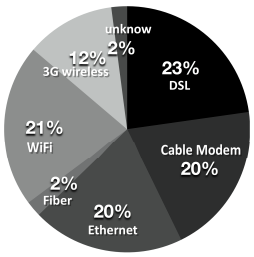

Figure 2: Breakdown of user's network connections

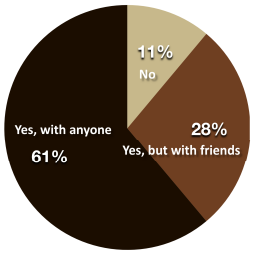

Figure 3: User's willingness of contribute

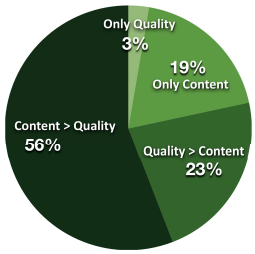

Figure 4: Breakdown of user's concern on videos

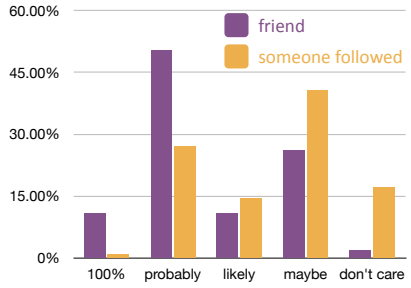

Figure 5: Comparison of the possibility of watching the entire video

Second, the survey asks users if they are willing to share their resources while streaming or downloading, regardless any particular implementation. The result, as shown in Figure 3, is gratified that only 11% of users do not want to contribute. Over 60% of the users do not care who they are sharing with, and 28% users only want to share the resource with close relations, e.g., in the same Twitter follower set.

Third, in general, only 55.9% of the people tend to watch the entire video. Not surprisingly, this behavior is affected by the video content, as people concern more about the video content than the playback quality, as shown in Figure 4. Interestingly, when a video is uploaded by a friend, a user are more likely to watch the entire video. Figure 5 shows the comparison of the possibility of watching the entire video, uploaded by a friend and someone the user followed. We consider a weighted possibility that "100%", "probably", "likely", "maybe" and "don't care" are 1, 0.9, 0.7, 0.5 and 0, respectively. The figures then will be 76.8% and 55.6% to watch the entire video.

4. SYSTEM OVERVIEW

We consider a video tweeting system, in which a server accepts user-generated video from networked camcorders or multimedia-ready mobile phones, and distributes the video to the uploader's followers.

4.1 Streaming User and Storage User

As suggested by the survey, there exist two types of followers interested in the tweeted video, namely, *streaming users* and *storage users*. The streaming users expect to watch the live video, and the storage users expect to download and then watch the video at a different time, due to the presence of other concurrent events.

A streaming user would leave after a while if finding the video is out of its interest. On the other hand, a storage user that downloads the video asynchronously does not have the concern of interest nor playback quality, until he/she starts to watch the live video. Therefore, such users are relatively stable, though they could switch their options. There is another significant difference between this video tweeting and conventional streaming: conventionally, users may join and leave at any time, while in our scenario, all the users join the system at the beginning but can leave at any time, i.e., the initial number of users is the upper-bound.

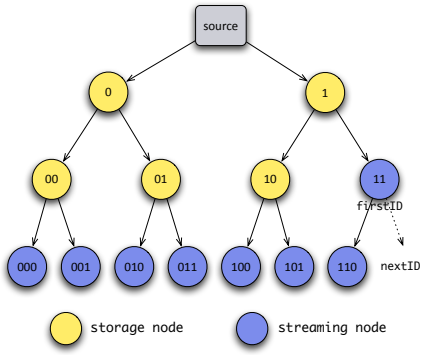

Figure 6: Example of overlay tree with IDs

4.2 Overlay Tree

Considering the above factors, we advocate a tree overlay design for video tweeting. It is known that a tree with data push is more efficient than a mesh with data pull, but constructing and maintaining the tree with peer churns is a daunting task. Fortunately, since the users are followers of the tweeted video, they are generally interested in watching the whole video as our survey suggests, and the existence of storage users implies that their churns are much less frequent, which can thus be strategically placed to improve the robustness of a tree.

To efficiently coordinate the two types of users, our COOLS implements a labeled tree that embeds node locations in the overlay. For ease of exposition, we explain it with a binary tree, in which each node is assigned an ID, represented by a binary code. The two children of the server node (the source) have IDs 0 and 1, respectively, and, for a given node, its left child's ID is the node's ID appended by a 0, and the right child's ID is that appended by a 1. As such, the ID embeds the location of a node and also that of its all ancestors, and the number of digits (*length*) indicates its depth in the tree.

We now define a partial order of the IDs: if two IDs are of identical length, the one with greater value is considered greater, e.g., 010 is greater than 001; otherwise, the longer ID is greater, e.g., 000 is greater than 11. We then define an *increment* operation of the ID: if not all the bits of the ID is 1, an increment operation will increase the ID value

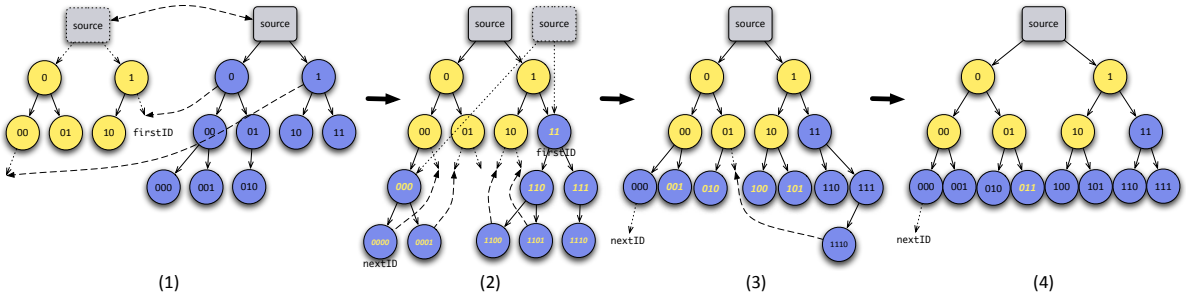

Figure 7: Example of overlay construction: creating, merging and promotion

by 1; otherwise, the length of ID will be increased by 1 and all the bits are set to 0. The operation of *decrement* can be defined similarly.

Since the storage nodes are relatively more stable, we expect that the storage nodes' IDs are smaller than that of streaming nodes after the tree is stabilized; in other words, the storage nodes will be placed at more critical locations of the tree. Figure 6 shows an example of such an overlay tree. We will detail the construction and maintenance of the overlay in the next section, particularly on both achieving robustness with storage users and minimizing delay for streaming users.

5. DESIGN DETAILS

5.1 Overlay Construction

5.1.1 Creating Storage Tree and Streaming Tree

As mentioned, the storage nodes are expected to be close to the source. However, we also need to guarantee short startup for the streaming nodes, which requires them to be close to the source, too. Fortunately, since the storage users are delay-tolerant, the dilemma can be eliminated by prioritizing the streaming nodes in the initial stage.

Specifically, COOLS first constructs two trees, one contains all the streaming nodes, referred to as *streaming tree*, and the other contains all the storage nodes, referred to as *storage tree*. After the two trees are established, and the streaming nodes have buffered enough data, the two trees will be merged to one overlay tree.

For each tree, we let source record the next value of the current maximum ID, which is initially set to 0. To make it clear, the value for the streaming tree is $nextID$, and that for the storage tree is $firstID$ (the meaning will be explained later). The source adds nodes to the trees sequentially. Each newly added streaming user will be assigned an ID of value $nextID$, and $nextID$ is then increased by one. The node thus knows its parent's location by checking the prefix of the ID. If the source has enough children, it will provide the address of one of its children whose ID is the same as the first digit of the node's ID, i.e., which branch should the new node go; otherwise, the new node will be source's child. The construction for the storage tree is similar, except that $firstID$ will be used.

It is worth noting that each node only keeps the local information of the parent, two children, and the source, while the source only keeps the information of the four depth 1 children as well as the $nextID$ and $firstID$.

5.1.2 Tree Merging and Node Promotion

At the beginning, the source dedicates to the streaming tree. When the streaming nodes have buffered enough data for starting playback, the source will start to push data to the storage tree. In the meantime, the source searches for the node(s) that should be the parent(s) of two new nodes, and then performs a merge. Specifically, it stops pushing data to the streaming tree and notifies the two streaming tree children to connect to the parent(s). Since the streaming nodes have sufficient amount of the video data, they will seamlessly join the storage tree without interruption. The first step in Figure 7 shows the procedure of merging two trees.

Since the source records the next value of the current maximum ID in the storage tree as $firstID$, the new ID of the left streaming child node will change to $firstID$, and this is the reason we name this value; the right child node is assigned the next value of $firstID$. Note that the value of $firstID$ is not changed. Moreover, the source also recomputes a new $nextID$, which is the next value of the maximum ID in the storage tree, e.g., 0000 in Figure 7. Then the source disseminates this value in the tree.

After the two trees are merged, the overlay tree is probably not a complete tree, as some streaming nodes may locate deeper than expected, referring to the value of $nextID$. These nodes are in an *unsteady state*, e.g., node 0000, 0001, 1100, 1101 and 1110 in the second step of Figure 7. Some leaf storage nodes are also unsteady if they should have children but haven't yet, also based on $nextID$, e.g., nodes 00, 01 and 10; other nodes are in a *steady state*. Since most of the unsteady streaming nodes will move upwards, we call this procedure as *node promotion*.

The unsteady nodes will send control messages toward the source. If it finds that its ID is no smaller than $nextID$, it sends a *promotion message*; if its potential children's ID is smaller than $nextID$, it send a *child requiring message*. A rendezvous (not necessary the source) receiving such messages will match them, notifying two senders to connect with each other. For example, in the figure, node 00 matches itself with node 0000, node 0 matches node 01 with node 0001, node 1 matches node 10 with nodes 1100 and 1101, and at last the source matches node 01 with node 1110.

Suppose the heights of the original two trees are H_l and H_s, respectively. To merge and promote, in the worst case, all the promotion and child requiring messages are matched at the source. Thus in each round, the nodes in the highest depth send promotion message and get matched, which takes $(H_s + H'_l)$ time, where H_l is decreased by 1 each round. The

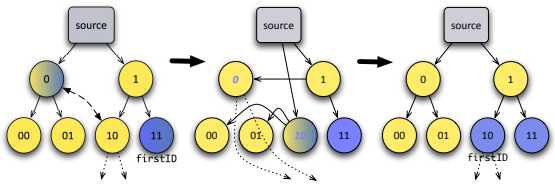

Figure 8: Example of node demotion

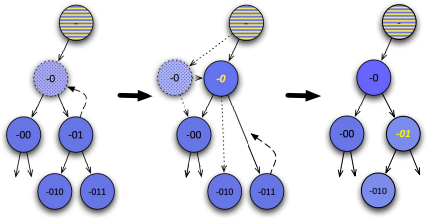

Figure 9: Example of node leave

tree's height will eventually become H, and all the nodes between depth H_s and depth H are streaming node. There will be $(H_l + H_s - H)$ rounds. For complete trees, all the three heights are bounded by $O(\log N)$, and the time to complete the promotion is thus bounded by $O((\log N)^2)$.

5.2 Handling Node Dynamics

A storage user may have time to watch the live video after a while, and thus becomes a streaming node. There is also possibility that a streaming user finds the video of no interest, and thus stops watching and leaves the system. Given that the users are more likely to watch the entire video being uploaded by their friends, such events are relatively rare in our application scenario; yet proper handling is still necessary, as addressed below.

5.2.1 Node Demotion

For switching from storage to streaming, we need to demote the node in the tree. It is worth clarifying that the demotion will not degrade the playback quality, as it only lowers the depth of the node in the tree, since it becomes more possible to leave the overlay.

Figure 8 shows an example of the demotion process. Suppose user 0 starts to watch the live video, it first informs the source and gets the value of $firstID$ from the source, and the source then decreases the value of $firstID$ by one. In order to minimize the overhead, only the demoting node (node 0) and the last storage node (node 10) will perform a switch, so that node 0 will become the first streaming node. Then the two nodes exchange the IDs, as well as their connections of children and parents. The demotion is then completed. Since the demoting node have already downloaded sufficient data, it can immediately start watching without any startup delay. Obviously, the demotion will not affect other nodes' playback.

5.2.2 Node Leave and Crash

Certain streaming users may stop watching after finding that the video, though uploaded by a friend, is of no interest. Node crash is also possible. Storage nodes however will not leave unless crash.

When a node gracefully leaves the system, it will notify the source, its parent and its children about its current information of the connections. When the source receives the notification, it will compute a new $nextID$ and disseminate the new value. All the right child nodes along the path will be promoted, and all the left child nodes remain unchanged. An example is shown in Figure 9.

When a node crashes, neither the source nor its connected node will be notified. With our ID design, the crashed node's children can quickly locate their grandparent, and the grandparent can thus knows the connection information of its grandchildren. The source will also be notified by these children. Then these nodes will repair the tree as if the crashed node graceful leaves.

Note that the crash of a storage node may cause that the new tree violates the requirement that all the storage nodes' IDs should be smaller than that of the streaming nodes. This can be addressed by the source through computing a new $firstID$ value, and switching the last storage node and the streaming node that should be demoted.

6. PERFORMANCE EVALUATION

6.1 Simulation Settings

We now present our preliminary evaluation results for COOLS. We have also implemented ChunkySpread- [13] and CoolStreaming-like [14] overlays for comparison, which are typical tree and mesh designs, respectively. In our evaluation and comparison, we use the following three typical metrics, which together reflect the quality of service experienced by end users.

Data loss rate. It is defined as the fraction of the data blocks missing their playback deadlines;

Startup delay. It is the time taken by a node between its requesting to join the overlay and receiving enough data blocks to start playing;

Playback delay. It is the time for a data block from being sent out by the source to being played at a node.

We also examine the control overhead, and record the number of the control messages sent by each individual node over time.

Unless otherwise specified, the following default parameters are used in our simulation: the session length is set to 3600 seconds and each data block is of 1-second video data; there are 2000 overlay nodes, 60% of which are storage nodes at the beginning. All the nodes join the overlay at the beginning. To emulate node dynamics, loosely based on our survey results, a storage node will switch to streaming with a probability of 0.02 per second after 5 minutes, and a streaming node will leave the system with a probability of 0.02 after viewing 10 minutes of the video.

6.2 Sample Results

Figure 10 shows the cumulative distribution function (CDF) of data loss rate. The pure tree-based solution performs the worst. The mesh-based solution is resilient to node dynamics, and thus performs better than the pure tree-based approach. While our COOLS is also tree-based, through differentiating the two types of nodes and making the stable

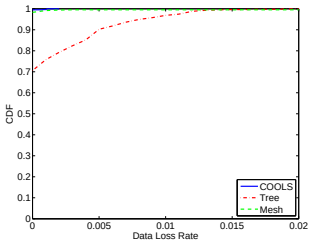

Figure 10: CDF of data loss rate

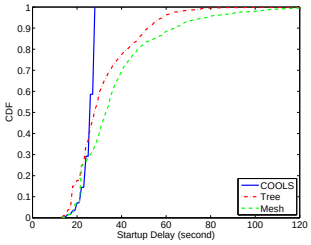

Figure 11: CDF of startup delay

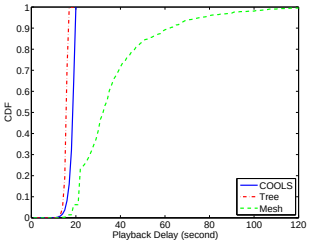

Figure 12: CDF of playback delay

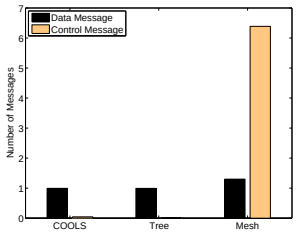

Figure 13: Comparison of messages transimitted

storage nodes close to the root, it performs even better than mesh in this application scenario.

Figure 11 shows the CDF of startup delay. The pure tree-based solution performs slightly better than the mesh-based, but our COOLS takes much shorter time to start playing. This is because the mesh needs a longer time to search for partners, and the pure tree-based solution is unaware of the coexistence of the two types of nodes and thus does not explicitly optimize the service to the streaming nodes.

Figure 12 shows the CDF of playback delay. Again, our COOLS and the pure tree perform better than mesh, showing the superiority of data push in terms of delay minimization. Our COOLS performs slightly worse than the pure tree, mainly due to the computing and updating of node IDs. Nevertheless, our ID design helps nodes quickly locate their positions, improving the system robustness as demonstrated by the data loss evaluation.

Finally, Figure 13 compares the control message overhead of the three approaches. Not surprisingly, with data pull, the mesh suffers from much higher overhead than the other two. Our COOLS is tree-based and hence experiences similar overhead as the pure tree, though slightly higher because of the switching operations.

7. CONCLUSION AND FUTURE WORK

This paper presented Coordinated Live Streaming and Storage Sharing (COOLS). The COOLS design was motivated by a questionnaire survey on real users of Internet video sharing, which reveals a coexistence of live streaming and storage sharing for social-networked video and the interest of different users. Through a novel ID code design that inherently reflects nodes' locations in a peer-to-peer overlay, COOLS leverages stable storage users and yet inherently prioritizes living streaming flows, providing better scalability, robustness, and streaming quality.

COOLS offers an initial attempt to promote the coexistence of the two types of users for peer-to-peer streaming. There are many possible avenues to explore in this framework. We expect to extend the tree structure with flexible degrees, so as to further reduce its height and hence delay. Since there are wired Internet user and wireless mobile user, their heterogeneity may also need to be addressed in the overlay construction and maintenance. Moreover, we are now conducting a systematical measurement to further understand and utilize the interest and behavior Twitter users.

8. REFERENCES

[1] AppScout. Inauguration: Twitter Reports 5 Times Normal Tweets Per Second. http://www.appscout.com/2009/01/inauguration_twitter_reports_5.php, 2009.

[2] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida. Characterizing User Behavior in Online Social Networks. In *Proc. of ACM IMC*, 2009.

[3] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *Proc. of ACM IMC*, 2007.

[4] X. Cheng, J. Liu, and C. Dale. Understanding the Characteristics of Internet Short Video Sharing: A YouTube-based Measurement Study. *IEEE Transactions on Multimedia*, 2010.

[5] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. YouTube Traffic Characterization: A View From the Edge. In *Proc. of ACM IMC*, 2007.

[6] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, and C. Huang. Challenges, Design and Analysis of a Large-scale P2P-VoD System. In *Proc. of ACM SIGCOMM*, 2008.

[7] J. Liu, S. G. Rao, B. Li, and H. Zhang. Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast. *Proceedings of the IEEE*, 96(1):11–24, 2008.

[8] A. Mislove, M. Marcon, K. Gummadi, P. Dreschel, and B. Bhattacharjee. Measurement and Analysis of Online Social Networks. In *Proc. of ACM IMC*, 2007.

[9] paidContent. Twitter Search Fails Under Thursday's Celebrity News Rush. http://paidcontent.org/article/419-twitter-search-fails-under-thursdays-celebrity-news-rush, 2009.

[10] The Official Google Blog. Introducing Google Buzz. http://googleblog.blogspot.com/2010/02/introducing-google-buzz.html, 2010.

[11] The Official YouTube Blog. Y,000,000,000uTube. http://youtube-global.blogspot.com/2009/10/y000000000utube.html, 2009.

[12] TorrentFreak. BitTorrent Launches Ad Supported Streaming. http://torrentfreak.com/bittorrent-launches-ad-supported-streaming-071218, 2007.

[13] V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast. In *Proc. of IEEE ICNP*, 2006.

[14] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. CoolStreaming/DONet: A Data-Driven Overlay Network for Peer-to-Peer Live Media Streaming. In *Proc. of IEEE INFOCOM*, 2005.