SonicStream: A Network Coding Based Live P2P Media Streaming System With Rich User Experiences

Xiaogang Chen, Ning Ren, Xiaochen Zhang, Xin Wang, and Jin Zhao

Abstract: **Recent studies have convinced that network coding can improve the performance of live media streaming in terms of startup delay, resilience to peer dynamics, as well as reduced bandwidth cost on dedicated streaming servers. However, there still exist some strategy drawbacks and neglected problems which need to be further researched. In addition to the commonly used evaluation parameters of the network and user experiences mentioned above, we focus on additional key factors, playback lag and switch lag, which have not been fully explored in previous work. In this paper, we present SonicStream, a novel and fully implemented live peer to peer (P2P) media streaming system with consideration of rich user experiences, including startup delay, playback continuity, playback lag, switch lag, etc. In pursuit of a further enhanced user experience, we revise traditional peer selection/data scheduling methods. Through a series of experimental evaluations and a cautious comparison with the latest similar work R^2, the superior performance of SonicStream has been preliminarily verified.**

Index Terms: **Live media streaming, network coding, playback lag, switch lag, user experience.**

I. INTRODUCTION

Live media streaming has emerged to be immensely popular along with the entertainment demand, such as live sport matches, from millions of Internet users. Recent studies have shown that peer to peer (P2P) structure in media streaming systems could be efficient to solve original tough problems [1], [2] due to its natural properties such as scalability, autonomy and resilience to dynamics. According to the type of overlay structures, the work related to P2P media streaming systems could be summarized into three categories: (1) Single-tree structure [3]–[5]; (2) multi-tree structure [6]–[8]; (3) Mesh based structure with the use of gossip-based protocol [9]–[11]. Due to the user experience requirement for smooth playback, live media streaming is typically sensitive to network congestion. Buffering is a typical mechanism to smooth out the congestion jitter. However, buffering also introduces the possibility of user experience degradation in playback lag among peers at the same time.

Because of the instantaneity of live media streaming systems, playback lag, which is defined as the time difference between playback time point of a client and the server, is very important. However, only a few works focus on source-to-end delay, the delay to receive the first media data block directly or indirectly from the source. It only has partial influence on the playback lag of a peer. The work [10] is the most encouraging work on source-to-end delay, which adopts a "pull-push" hybrid scheduling strategy and multi-tree overlay structure to improve the average source-to-end of the system. It uses the "pull" strategy as a highly efficient bandwidth-aware multicast routing protocol and the "push" strategy as the transmitting method, which is delay and control overhead efficient. While AnySee [11] adopts a hop measurement strategy to choose the nearest path to the content server to minimize the source-to-end delay in a tree-based overlay structure.

Recently, to eliminate drawbacks of previous work, such as scheduling difficulties, vulnerable to peer dynamics and network congestion, etc., an innovative method, network coding, has been considered in recent work. Network coding was first proposed by Ahlswede *et al.* [12], the core idea of which is to allow mediate peers in an overlay network to have extra capability of coding/decoding received data. Many literatures have demonstrated that network coding could help the system achieve a theoretically maximum throughput [13], [14], and some recent work has persuasively brought this concept into P2P live media streaming systems [15]–[18]. Due to the unavoidable increase of data delay in pull-based designs, R^2 [19] adopted an innovative random push method, which actively pushes encoded data blocks to peers with inadequate resources based on their data availability index.

As shown in R^2, push strategy, which is delay and control overhead efficient, can be introduced to mesh-based P2P live media streaming systems because of the networking coding. To our best knowledge, there is no existing work on optimization of the playback lag over mesh based live media streaming systems. And the switch lag, which is measured as the time difference between the moment when a user issues request for another channel and the moment the first frame could be viewed by the user, is also missing in the previous studies. R^2 deals with the case that the peer's bandwidth merely meets the stream rate, but in our work it shows that R^2 doesn't work well under an environment with more bandwidth available by introducing more control overhead. To reduce the playback lag, better peering strategies and scheduling schemes, in addition to source-to-end optimization, are also needed [20]. Peering strategy is mostly responsible for the structure construction of P2P network, and thus involves the source-to-end factor. Scheduling scheme is responsible for the data transfer in the P2P network. The faster it is, the shorter the playback lag will be.

In this paper, we focus on rich user experiences, including

Manuscript received on July 20, 2008; approved for publication on November 11, 2008.

This work was supported in part by NSFC under Grant No. 60702054, 863 program of China under Grant No. 2006AA01Z203, Shanghai Municipal R&D Foundation under Grant No. 07dz15004-1, the Shanghai Rising-Star Program under Grant No. 08QA14009, and State Key Laboratory of Integrated Service Networks under Grant No. ISN-9-06.

The authors are with the School of Computer Science, Fudan University, Shanghai, China. email: xinw, 052021178, 06300720197, jzhao@fudan.edu.cn, and zxcxczhang@gmail.com. Corresponding author: X. Wang.

startup delay, playback continuity, playback lag, switch lag, etc., in mesh-based P2P live media streaming by employing a distributed playback lag adaptive peering strategy and a push strategy, rather than pull or pull-push hybrid strategy. This strategy is encouraged by network coding and will significantly decrease the scheduling complexity. The system is called SonicStream, which means users can watch the channel very fast due to the better playback lag and switch lag. In our previous work [21], we have presented preliminary results of SonicStream. In this paper, we designed and implemented a full-function real-world P2P live streaming system with network coding and we analyzed major metrics relative to the network and user experiences.

As described above, the main contribution of this paper could be summarized into: (1) An actual P2P live media streaming system is implemented (2) a method to provide rich user experiences in a mesh-based environment is introduced, including a better push strategy and peering strategy. With all the details we take into consideration, the satisfactory of users could be more acceptable.

The remainder of this paper is organized as follows: Section II presents the implementation overview and describes how the peering strategies and scheduling schemes mentioned above are adopted and how they work in our system. Also, some special details of the system are considered. Section III demonstrates the performance evaluation of our system. Conclusion and possible future work are finally outlined in Section IV.

II. SCHEME DESIGN OF SONICSTREAM

A. *Implementation Overview*

Our system is a live media streaming system based on a data-driven unstructured overlay. The media source is referred to as the streaming server and the others (receivers) as peers. The term "peer" is referred to each of the receivers. The system consists of three major components:

1. Content server: The bootstrapping and content publisher peer of the system;
2. Access server: Access portal of the system, which returns a channel list to peers;
3. Peer: The core component executed on each peer, gathering most functions of the system.

As discussed above, the main object of the paper is to provide rich user experiences, including startup delay, playback continuity, playback lag, switch lag, etc. To our best knowledge, this is the pretty comprehensive optimization of the metrics. Here, the architecture of the peer is presented, showing how it is constructed and what has been done to achieve the goal described above.

The software architecture of a peer could be best depicted in Fig. 1.

A.1 Overlay Management Module

This module is basically responsible for peer discovery and overlay structure construction and maintenance. As illustrated in Fig. 1, the overlay module consists of a member management module and a partner management module.

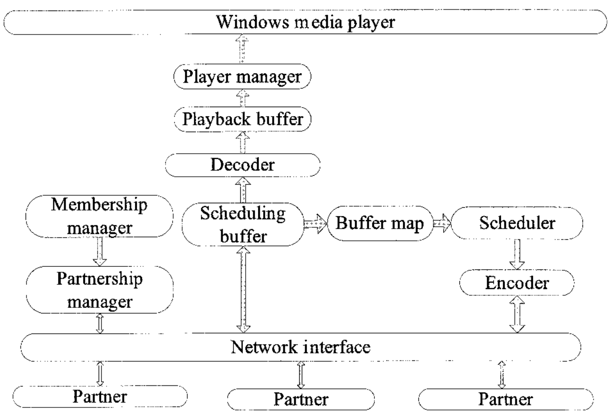


Fig. 1. Peer structure of the system [21].

Member management module maintains the member list, which is the current existing peers of a channel. It uses a gossip-like protocol to do this.

The partner management module maintains the partner list of a peer. Peers periodically exchange neighborhood information, such as IP message, data availability, etc., within their own partner clusters.

Media content will be actually exchanged among peers in the same partnership cluster. The initial selection of partners for each peer, therefore, could have great influence on the playback performance, such as startup delay, playback lag, etc. To guarantee an acceptable playback lag, the system is designed as follows:

When a peer joins a channel, the bootstrapping peer (i.e., content server) will return the number of the segment it is playing and the current capacity of the system. The term capacity is the total number of the peers in the channel. Let's assume that d is the maximum number of father neighbors, u is the maximum number of child neighbors, c is the capacity of the system at that time and s is the maximum number of the peers that the server can serve (3 in the case of SonicStream). Note that u is greater than d for the system to work well. So the capacity of the system can be calculated as [21]:

$$c = \frac{us[1-(\frac{u}{d})^{(h-1)}]}{d-u} + s. \tag{1}$$

After the peer received the message, it will run a function to set its start point h segment after it. And when u is not equal to d, h is calculated as:

$$h = \log_{\frac{u}{d}}\left(\frac{c(u-d)}{su}\right) + 1. \tag{2}$$

When u is equal to d, h is calculated as c divided by s:

$$h = \frac{c}{s}. \tag{3}$$

When the start point is set, a peer will choose partners from the member list that was established. Peers will only accept this requirement when their buffer maps are overlapped with each other.

It may be learned from EQ2 that the peer won't increase its start point h unless the former peers exhaust all the bandwidth available. So as a preferred implementation, the system can provide better playback lag than what the random peering strategy could do.

Peer updates priority of its neighbors according to the result of data scheduling. In addition, peers deliver quit message to the peers in their member list when they are ready to leave, on receiving such message, a peer would delete this peer from its member list automatically and flood the message. In abnormal situations such as power-off, the updating function would guarantee that disappearing peers be deleted from system in no more than P interval. Also, in this way, the Content server could maintain the system capacity to be a respective explicit value.

A.2 Data Scheduling Algorithm

The system adopts a random push algorithm for data scheduling similar to R^2 [19]. The difference is that a peer pushes data blocks to its partners only when it has received all blocks of a certain segment, which will save great amount of coding complexity with a delay of one segment duration. In addition, a peer only sends limited number of blocks to its partners to avoid unacceptable data overhead in the experiment environment. Unlike R^2, where the buffer map is updated on demand, the buffer map is exchanged periodically to avoid unacceptable data overhead. Also, the schedule process runs periodically to guarantee timely scheduling of media content, and the scheduling frequency has been carefully chosen to decrease the consumption of resources, which will be discussed in Section III. To achieve a better user experience, following improvements of scheduling algorithm have been made in the system:

First, after a synchronized comparison of local buffer-map and fresh buffer-map of each partner, a positive offset will be returned representing the start point of their overlapping part, base on which, data blocks could be orderly scheduled.

Second, to avoid redundant data dissemination, the number of data blocks each peer delivers to its neighbors is confined. After sequential comparison, the number of blocks each peer should send to its partner each time is set to be the ratio of the partner's residual demanding block number in a certain segment and the maximum partner number predefined. So that all the neighbors could help the peer get sufficient blocks in a collaborative but restrictive manner.

A.3 Network Coding and Decoding Modules

As illustrated in Fig. 1, network coding and decoding modules are employed in the system, shown as encoder and decoder in Figure 1 respectively. The network coding is adopted to enable the scheduling algorithm, push strategy work well, by making streaming packets scheduled faster. This will cause better playback lag.

Network coding/decoding modules adopt a typical random linear network coding method and a traditional Gauss elimination method, respectively. The network coding finite field is 2^8, which is enough for the relevancy consideration.

A.4 Other Technical Solutions

Windows media encoder is used to generate successive ASF streaming packets (advanced streaming format, a media format defined by Microsoft, suitable to be played over network).

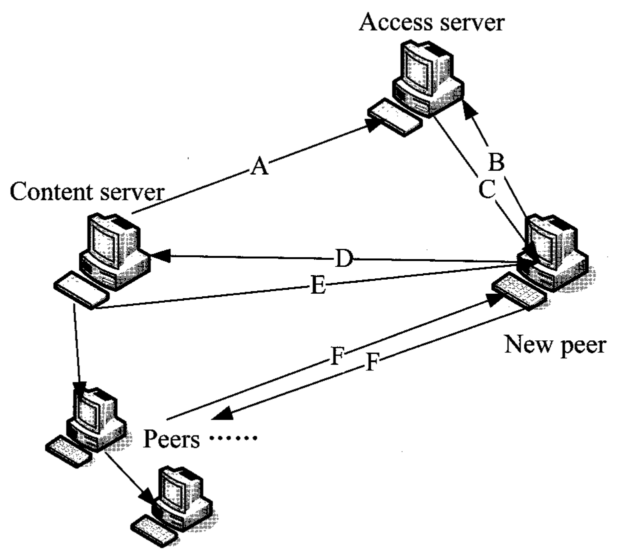


Fig. 2. Operational overview of the system.

A program module is designed for sending stream to windows media player based on streaming http protocol. This functionality is performed by player management module (shown in Fig. 1). In this way, the coupling problem between the module and player is simplified, which is different from other Live P2P media streaming systems with self-designed player.

B. Operational Overview

The system is generally implemented according to the following descriptions:

Original media content (generated by windows media encoder in the case of SonicStream) is first divided into several data segments, each segment is labeled with a sequence number and further divided into several blocks (we call the number of blocks contained in a segment as segment size).

The operational flow is shown in Fig. 2.

The steps of operation may be described as:

1. Content Server contacts with Access Server and registers its information on Access Server.
2. Peer connects to the access server for the channel list.
3. The access server will return a channel list to the peer.
4. The peer chooses a channel,
5. Return a channel capacity to the peer.
6. Peer starts peering function and streaming function.

The peering function can be described as follows:

The peer will run a gossip-like protocol to establish and maintain a member (peers that view the same channel) list for each peer based on a random chosen manner.

The new peer will choose its partner from the member list.

After the confirmation of each peer's partner list using a partner selection method described above, media data could be flexibly exchanged among peers in the same partnership based on each peer's data availability index (referred to as buffer map in the case of SonicStream). The streaming function can be described as two parts, receiving and sending. The receiving and sending parts are shown in Figs. 3 and 4, respectively.

As discussed above, network coding/decoding is deployed on

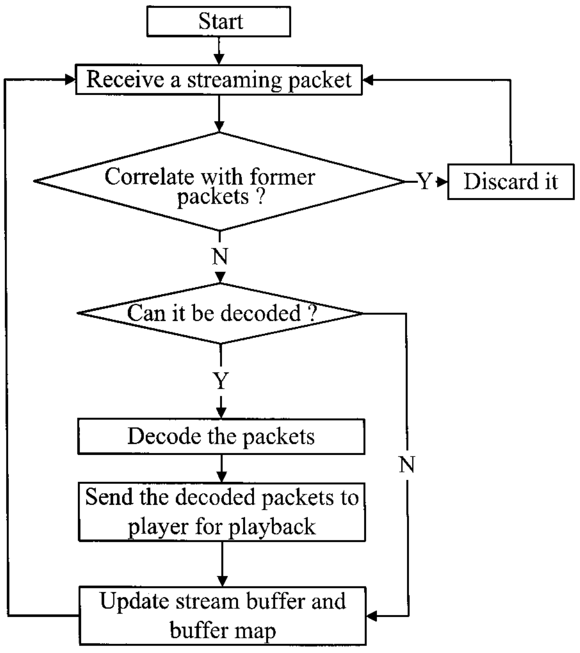

Start
Receive a streaming packet
Correlate with former packets ?
Y
Discard it
N
Can it be decoded ?
Y
Decode the packets
N
Send the decoded packets to player for playback
Update stream buffer and buffer map

Fig. 3. Streaming flow part 1 (receiving flow).

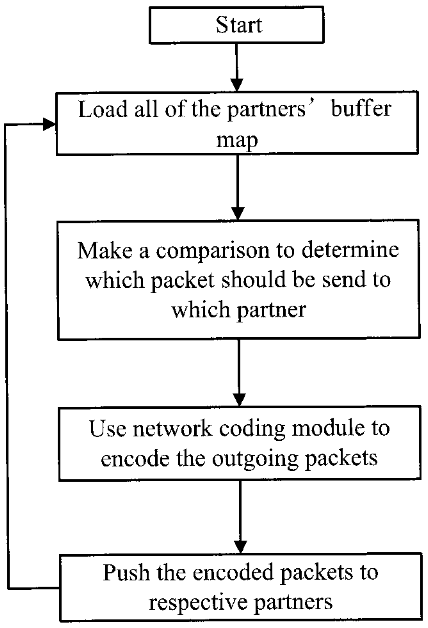

Start
Load all of the partners' buffer map
Make a comparison to determine which packet should be send to which partner
Use network coding module to encode the outgoing packets
Push the encoded packets to respective partners

Fig. 4. Streaming flow part 2 (sending flow).

all peers in the system for data dissemination, adopting a typical random linear network coding method and a traditional Gauss elimination method. A random push mechanism is adopted for data scheduling process inspired by R^2 [19]. A peer will not start to play until its playback buffer is fulfilled.

III. EXPERIMENT AND PERFORMANCE EVALUATION

Our testing scale involves a group size of 25 computers with similar deployment and performance in the same LAN. The network is constructed in a typical "star" structure. The resource of those computers could be fully used with an approximate upload capacity of 100Mb/second (with proper restriction of such capacity, the improvement of our work compared to previous ones could be maximally observed). We run 8 peers on each computer and the bandwidth available for each peer is 10 Mbps. Each peer acts as a single fully functional node in the system.

As a real implementation and for better performance, enormous experiments have been done to define the proper values for various parameters. Because of the paper restriction, however, only segment size will be illustrated in figures separately to show that how it is set. The other parameters are discussed along with the other metrics and given as follows:

As described above, original media content (generated by windows media encoder in the case of SonicStream) is first divided into several data segments, each segment is labeled with a sequence number and further divided into 32 blocks, i.e. the segment size is 32 and each block is 2 KB, i.e. the block size is 2 KB. unless specified, the maximum number of father neighbors (which ones send streaming packets to the peer) is D (3 in the case of SonicStream), the maximum number of child neighbors (which ones the peer sends streaming packets to) is U (6 in the case of SonicStream), the buffer size is 5 seconds (containing 5 seconds data for playback at most), the streaming rate is 284 Kbps (CBR) and the server serves only 3 peers.

To evaluate the performance of our system, we mainly focus on the following metrics:

1. *Startup delay:* Measured by the period of time lasting from the click of a certain channel to the first frame which could be actually viewed by the user.
2. *Playback continuity:* Representing the number of blocks that arrive before their playback deadlines over the total number of blocks.
3. *Playback lag:* (Synchronization with content server): Measured by the difference of the current playback segment between each peer and content server.
4. *Bandwidth overhead:* Measured as the total amount of control overhead (such as BM exchange) and wasted data (data packets discarded due to correlation or excess of existing packets).
5. *Switch lag:* the period of time consumed lasting from the click of another channel, which is either the same as or different from the former channel, to the first frame which could be actually viewed by the user.

Also, to check the system works better, R^2 system and a pull based P2P media streaming system adopting the same overlay management policy without network coding (illustrated as P2P Live in the figures) are constructed.

Test results and analysis.

A. Segment Size

Segment size is a network coding parameter that can affect the performance of the system. By way of example, when segment size is 1, the network coding system degrades to P2P system. Then the push strategy can't work for causing too much redundancy data. On the other hand, too large segment size will lead to increased network coding/decoding time, and the efficiency

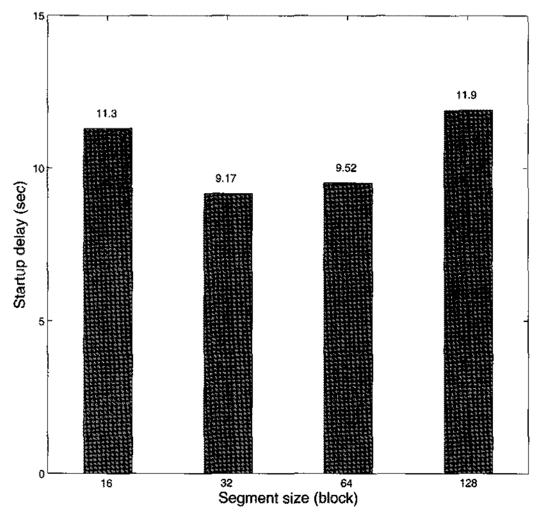


Fig. 5. Segment size.

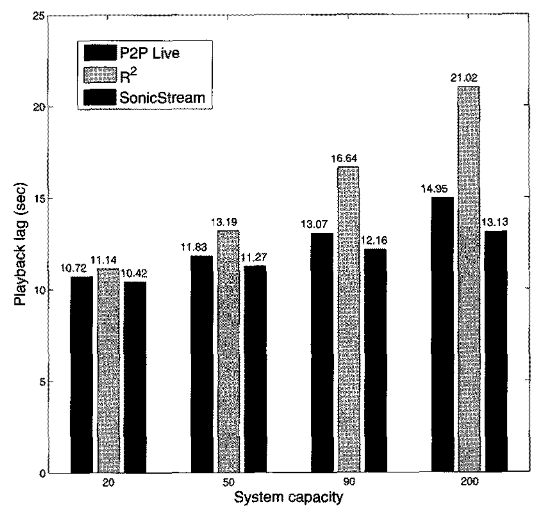


Fig. 6. Average playback lag of different systems [21].

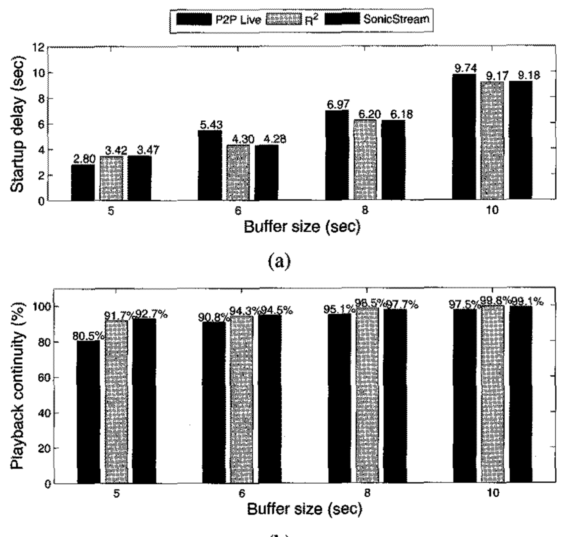


Fig. 7. (a) Startup delay and (b) playback continuity.

of the push strategy will be counteracted. The result of the experiment is shown in Fig. 5.

As illustrated in Fig. 5, the best system performance appears at 32 blocks per segment, showing that in live streaming systems, too small segment size won't take good advantage of the benefit of network coding, while too large segment size will take too long time to decode so that it counteracts the benefit of network coding.

B. Playback Lag

We test our playback lag optimization policy with a comparison of SonicStream, R^2 and P2P Live. The result is shown below in Fig. 6.

The playback lag is calculated as the average playback lag of all peers of the system. That is the sum of playback lags of all peers divided by the system capacity.

The result shows that SonicStream can perform better on playback lag in compare with R^2. The playback lag is pretty high because the content server can only serve 3 peers. As the system capacity grows, the average playback lag gets higher because of the limitation of the service ability of the server. Notice that the difference between R^2 and SonicStream gets larger as the capacity grows. This is because as the system gets larger, the possibility to choose a partner with higher playback lag gets higher.

It also shows that the push policy has lower transmitting delay from the comparison with P2P Live. Notice that the average playback lag is higher than the theoretical value (generally equals the sum of buffer size, h, and source-to-end delay) because of the start time of the player.

A possible drawback of synchronized playback adopted by R^2 is that, the time between the occurrence of a live event in the stream and its playback is the same across the board in all peers in the entire channel. This is harmful [19]. With the overlay management policy, SonicStream can easily avoid this situation.

C. Startup Delay and Playback Continuity

In a real-world P2P network scenario, peers are always facing a challenging dynamical atmosphere which involves arrival/departure behaviors with high frequency. We run 200 copies of our system on 25 computers to simulate 200 peers. To emulate such a volatile situation, a dynamical simulation is conducted based on the following assumption: At the start, peers arrive as a Poisson distribution, with the strength of 10. When the on-line peers reach a number of 100, a Poisson departure of peers is additionally introduced with strength of 5. A Poisson distribution control module is implemented on access server.

As startup delay and playback continuity are key factors which could affect user experience, we intend to evaluate how the system performs in terms of these two parameters. Experimental results are shown in Fig. 7.

The result shows that SonicStream and R^2 perform better

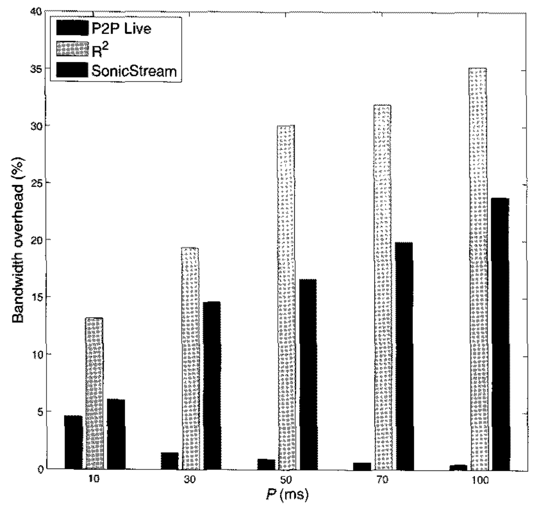


Fig. 8. Data overhead.

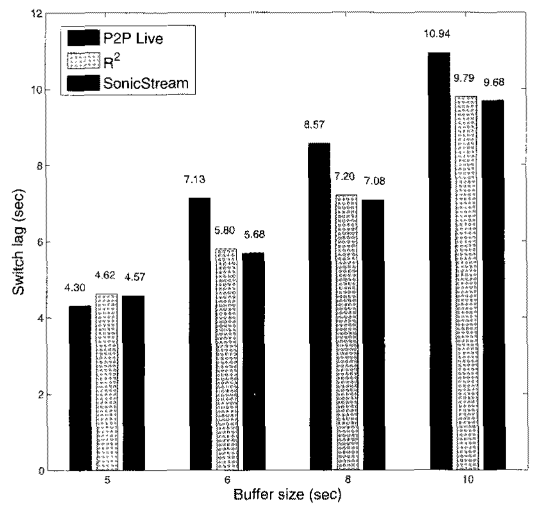


Fig. 9. Switch lag.

than P2P Live because of network coding and the different scheduling policy. And the startup delay is also improved due to the introduction of scheduling algorithm.

D. Bandwidth Overhead

AS defined above, bandwidth overhead includes control overhead and data bandwidth overhead. The internal time of every partner message exchanging is referred to as P. The internal time of every scheduling is referred to as T and G represents the generation size. The following equation should hold:

$$T/G \leq P \leq T. \tag{4}$$

The exchange frequency of buffer-map among the same partner cluster should be cautiously prescribed, since too long interval would cause delayed implication of buffer-map updating to partners and thus lead to uncontrolled dissemination of data blocks, ending as huge amount of overhead; and too short interval would also be devastating due to the unsustainable overhead of extra bandwidth consumption. To achieve an optimum system performance, we should make a trade-off on this parameter.

In R^2, this problem was partially solved by no longer sending buffer-maps periodically, instead, they are sent only when their status get changed, this solution, however, is only suitable in narrow-bandwidth situations when upload bandwidth barely exceed the stream rate of playback. In our system, this method could cause much overhead since large amount of data blocks would be pushed to saturated peers before it is actually notified by updated buffer-map to stop. In the experiment we set T to be 50 milliseconds, the statistic overhead result is present in Fig. 8.

Result shows that R^2, as designed for low bandwidth network, causes more data overhead in the environment with more bandwidth available. Because pull strategy can utilize bandwidth efficiently, all the data overhead is caused by control overhead. So P2P Live has a contrary tendency of data overhead changing with the other two. We also can see that even SonicStream causes about 6% data overhead. We argue that using a mesh-based push algorithm in an environment with more bandwidth available, as a real implementation, there is always cooperating problem between threads, so the data overhead always exists.

E. Switch Lag

In a real streaming system, fast channel switch is necessary for a user. In the former studies, this problem is hardly involved. As a real implementation, the switch lag is implemented as a key feature of our system, and also carefully tested. The result is shown in Fig. 9.

Notice that our system performs better than R^2 in various buffer size settings. Also, P2P Live works better than SonicStream when the buffer size is set to 5 seconds. That's because SonicStream waits a small period of time for better playback continuity, which can be seen in Fig. 7.

As is illustrated, switch lag is greater than the startup delay, because a peer will quit from the former channel, which will take some time, and after this, the peer will start the steps of joining a channel. This explains the greater switch lag.

The result also tells that our system only performs a little better than R^2, that is because R^2 wastes more bandwidth and this compensates the strategy flaw.

IV. CONCLUSION AND FUTURE WORK

The objective of this paper is to present a rich and better user experiences design and implementation of a P2P live media streaming system combining random push algorithm and random linear network coding inspired by R^2. Note that since our system deals with the playback lag and other user experience metrics, and when comparing to R^2, it will perform no better than R^2 for the scene of a flood of peers joining the system simultaneously.

In particular, the contribution of this paper involves of: (1) Implementing an actual P2P live media streaming system, thus making it feasible and promising in realistic application scenarios. (2) Introducing a method to provide rich and better user experiences, including startup delay, playback continuity, playback lag, switch lag, etc., in a mesh-based environment, including overlay management strategy and scheduling strategy. (3) Showing that our system enjoys great advantage with comparison of R^2, in terms of startup delay, playback continuity, and playback lag, through strict and meticulous experiments. With all the details we take into consideration, the satisfactory of users could be more acceptable. A user can enjoy better playback lag and switch lag in our system. Due to limitation of our work, there still exist problems which have not been deeply researched (such as the heterogeneity of peer bandwidth, mo efficient buffer management strategy for better playback del: [20], how network coding will work in high definition system etc.). Innovative solutions to such problems should be propos in the future as an extension to work mentioned in this paper.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful advice

REFERENCES

[1] Y. Tang, L. Sun, M Zhang, S Yang, and Y. Zhong, "Live video streaming service over peer to peer network: Design, implementation and experience", *IJCSNS*, vol.6, no.3B, Mar. 2006.

[2] D. A. Tran, K. A. Hua, T. T. Do, "A peer-to-peer architecture for media streaming," *IEEE J. Sel. Areas commun.*, vol. 22, no. 1, Jan. 2004.

[3] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proc. ACM SIGCOMM*, 2002.

[4] D. A. Tran and K. A. Hua, "Zigzag: An efficient peer-to-peer scheme for media streaming," in *Proc. IEEE INFOCOM*, 2003.

[5] J. Li, P. Chou, and C. Zhang, "Mutualcast: An efficient mechanism 1 one-to-many content distribution," in *Proc. SIGCOMM ASIA*, Apr. 200

[6] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and Singh, "SplitStream: High-bandwidth content distribution in cooperati environments," in *Proc. 19th ACM SOSP*, 2003.

[7] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High bar width data dissemination using an overlay mesh," in *Proc. 19th AC Symp. Operating Systems Principles* (SOSP), 2003.

[8] V. Venkataraman, P. Francis, and J. Calandrino, "Chunkyspread: Mul tree unstructured peer-to-peer multicast," in *Proc. the 5th Int. Worksh on Peer-to-Peer Systems*, Feb. 2006.

[9] X. Zhang, J. Liu, B. Li, TS. Yum, "CoolStreaming/DONet: A data-driv overlay network for efficient live media streaming," in *Proc. IEEE INFO COM*, Mar. 2005.

[10] M Zhang, J. Luo, L. Zhao, and S. Q. Yang, "A peer-to-peer network fo live media streaming-using a push-pull approach," in *Proc. MM 2005*, Nov 6–11, 2005.

[11] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, "Anysee: Scalable liv streaming service based on inter-overlay optimization," in *Proc. IEEE IN FOCOM*, Apr. 2006.

[12] R. Ahlswede, N. Cai, S.-Y. R. Li, and W. Yeung, "Network informatio flow," In *Proc. IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216 Apr. 2000.

[13] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. 41 Annu. Allerton Conf. Commun., Control, and Comput., Monticello, I* Oct. 2003.

[14] Z. Li, B. Li, D. Jiang, and L. C. Lau, "On achieving optimal throughp with network coding," in *Proc. IEEE INFOCOM*, vol. 3, Miami, FL, M 2005, pp. 2184-2194.

[15] J. Guo, Y. Zhu, and B. Li, "Codedstream: Live media streaming with ove lay coded multicast," in *Proc. SPIE/ACM Conf. Multimedia Computi and Networking*, 2004.

[16] Y. Liu, Y. Peng, W. Dou, and B. Guo, "Network coding for peer-to-pe live media streaming," in *Proc. fifth Int. Conf. GCC*, 2006.

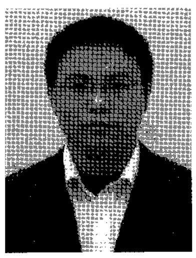
[17] J. Zhao, F. Yang, Q. Zhang, and Z. Zhang, "On improving the throughput of media delivery applications in heterogeneous overlay network," in *Proc. GLOBECOM*, 2006.

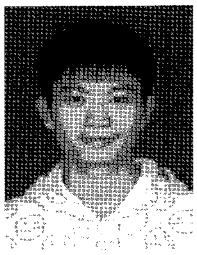
[18] M. Wang and B. Li, "Lava: A reality check of network coding in peer-to-peer live streaming," in *Proc. IEEE INFOCOM*, 2007.

[19] M. Wang and B. Li, "R2: Random push with random network coding in live peer-to-peer streaming," *IEEE J. Sel. Areas Commun.*, 2007.

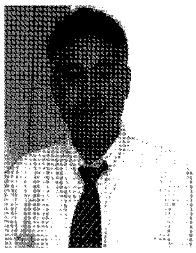
[20] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "A measurement study of a large-scale P2P IPTV system," *IEEE Trans. Multimedia*, 2007.

[21] X. Zhang, X. Chen, N. Ren, J. Zhao, and X. Wang, "SonicStream: An implementation of a live P2P media streaming system with improved playback lag," in *Proc. IEEE ICCE*, 2009.


Xiaogang Chen was born in YanTai, China, on December 14, 1981. He received the B.Sc. degree in Electronics Engineering and the M.Sc. degrees in Computer science from the Fudan University of Shanghai, China, in 2005 and 2008, respectively. Presently, he is a patent engineer assiatant. His major research interests are network architecture, P2P streaming, and network coding.


Ning Ren is the undergraduate of Fudan University, he will receive a B.Sc. in Computer Science from Fudan University of Shanghai, China in 2010. He is visiting Microsoft Research Asia as an intern in 2008 and 2009. His research interests include peer-to-peer network, data mining of program.

Xiaochen Zhang was born in Xi'an, China, on August 30, 1984. She received the B.Sc. degree in Computer Science and Technology from Xi'an Jiaotong University, China in 2006. She is now a master student working at Self-Organized Network and Information Coding (SONIC) laboratory with the major of Computer Science in Fudan University, China. Her major research interests are network coding, QoS guarantees and live streaming services.


Xin Wang is currently an Associate Professor in the School of Computer Science at Fudan University. He received his B.S. degree in information theory and M.S. degree in communication and electronic systems from Xidian University, China, in 1994 and 1997, respectively. He received his Ph.D. degree in computer science from Shizuoka University, Japan, in 2002. His research interests include wireless networks, peer-to-peer networks, and network coding.

Jin Zhao received the B.Eng. degree in computer communications from Nanjing University of Posts and Telecommunications, China, in 2001, and the Ph.D. degree in computer science from Nanjing University, China, in 2006. He joined Fudan University as an Assistant Professor in 2006. He visited Microsoft Research Asia as an intern in 2004. His research interests include P2P networks, media streaming and network coding theory.