Network coding based live Peer-to-Peer streaming towards minimizing buffering delay

Conference Paper · November 2010

DOI: 10.1109/ICCASM.2010.5619295 · Source: IEEE Xplore

CITATIONS
5

READS
100

5 authors, including:

Jin Zhou
ShenHuang Technology Ltd., Beijing, China
12 PUBLICATIONS **56** CITATIONS
SEE PROFILE

Jun Li
Tsinghua University
172 PUBLICATIONS **781** CITATIONS
SEE PROFILE

Network Coding Based Live Peer-to-Peer Streaming Towards Minimizing Buffering Delay*

Zhiming Zhang[1, 2], Ranran Hou[1], Hao Chen[1], Jin Zhou[2] and Jun Li[2, 3]
[1] Department of Automation, Tsinghua University, Beijing, China
[2] Research Institute of Information Technology, Tsinghua University, Beijing, China
[3] Tsinghua National Lab for Information Science and Technology, Beijing, China
Email: zzm02@mails.tsinghua.edu.cn, sakurazuka_hou@163.com, c-h06@mails.tsinghua.edu.cn, {zhoujin, junl}@tsinghua.edu.cn

***Abstract*--In current live Peer-to-Peer (P2P) streaming systems, buffering delay is too large compared with the channel switching delay of traditional TV service provided by cable companies. In this paper, network coding is applied to a live P2P streaming system to minimize the buffering delay that users experience. A new scheduling algorithm is designed, with server-push mechanism and intelligent method for determining initial playback point, to make full use of the advantage of network coding. The simulation results show that buffering delay can be reduced to as much as 5 seconds, compared to more than 15 seconds in current systems.**

Keywords-P2P networks; multimedia streaming; network coding

I. INTRODUCTION

During recent years, many commercial live Peer-to-Peer (P2P) streaming systems emerged after successful combination of P2P technology with video streaming systems [1]. P2P technology can make up for the shortage of server bandwidth by utilizing the uplink bandwidth of users in the same streaming session, and add scalability to live streaming applications. While compared with traditional TV services provided by cable companies, the buffering delay of live P2P streaming is too large. When users chooses a live P2P steaming, they usually have to wait for more than 15 seconds before the video starts [2, 3], which badly affects users' quality of experience.

Network coding was proposed in information theory to achieve the maximum throughput of multicast in an acyclic network [4]. Y.-h. Chu et al pointed out that linear network coding is enough for most situations [5]. T. Ho et al further proposed random linear network coding [6] which makes network coding practical. The first important application of network coding is Avalanche, a new protocol for P2P file sharing, which has shown promising results for file sharing systems with network coding [7, 8].

Network coding has the advantage to reduce buffering delay of live P2P streaming. In real system, random linear network coding needs to segment the continuous data packets with same size. Each segment has the same number of original data packets. Network coding operation is limited in the segment, such that nearly all coded data packets have some information of the original data packets in that segment. All coded data packets in the same segment have no difference from users' point of view. In other words, when a user wants to request some data packets from a segment from his neighbors, all data packets of that segment in his neighbors' buffers are considered equal. This characteristic increases the number of data packets the user needs, leading to higher successful request ratio and shorter fetching time.

In this paper, we used several designs to make full use of the advantage of random linear network coding for reducing buffering delay of live P2P streaming. Our designs were checked by simulations. The results indicated that buffering delay of live P2P streaming could be reduced to as much as 5 seconds.

The remainder of this paper is organized as follows. Section II introduces related works. Section III elaborates the details of our system design. Section IV validates our design. Section V concludes this paper and provides some open problems to be solved in the future.

II. RELATED WORK

Live P2P streaming has two kinds of delay. One is playback delay, which is the time difference between user's playback point and server's playback point. Another one is buffering delay, which is the lag between a channel is chosen by a user and the video comes up on user's screen. D. Ren et al [9] tries to decrease the playback delay by constructing a power-based topology. In this paper, we focus on buffering delay.

Network coding has been used in live P2P streaming due to the similarity between live P2P streaming and multicast. Mea Wang et al [10, 11] designed a pull-based live P2P streaming system with random linear network coding. They evaluated the system under different parameter settings. Experiment results show that network coding can improve the performance of live P2P streaming system, especially when the bandwidth supply barely meets the streaming demand. They focus on the fair comparison between live P2P streaming systems with and without network coding. We believe that network coding can be further explored to reduce the buffering delay of live P2P streaming.

Mea Wang et al [12] further employs random push streaming algorithm to utilize the advantage of network coding. Simulation results show that this kind of design can improve the performance of live P2P streaming greatly in terms of buffering delay, resistance to peer (user) dynamics and utilization ratio of peer uplink bandwidth. However, all

* Supported by project from NEC Labs China

commercial live P2P streaming systems use pull-based or push-pull based scheduling algorithm, so their results still needs to be validated in real world system.

III. SYSTEM DESIGN

In this section, we will give our design of pull-based live P2P streaming system with random linear network coding. Some definitions are given in table I.

Table I
Definitions used in this paper

Definition	Explanation
r_s	Streaming rate
u_i	Uplink bandwidth capacity of peer *i*
u_s	Uplink bandwidth capacity of server
R_i	Packets number that peer *i* should request in one request cycle
Data packet	The minimum transmission unit
Segment	A group of continuous data packets which is the operation scope of network coding
B	The original number of data packets in one segment
Redundant	We say received data packet is redundant if the data packet is linear dependent with the data packets of the same segment this node has received.
Aggressiveness (AGG)	The number of coded data packets one peer should receive before it starts to serve others.
Density	The percentage of nonzero coefficients when the linear network coding is performed
Buffer map	Segment availability information
decodable	We say one segment is decodable if and only if this segment has AGG data packets that are linear independent.
Skipped segment	The segment that is not decodable when the segment arrives at playback point.
UUR	Uplink bandwidth Utilization Ratio
N	Node count in a streaming system

A. Design of Architecture

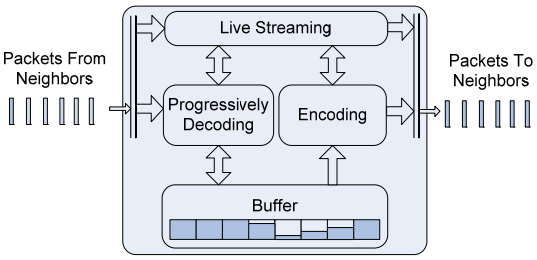

Figure 1. The architecture of a Node

The architecture of a node (peer) is shown in figure 1. Compared with live P2P streaming system without network coding, network coding based live P2P streaming system added two modules to the node architecture, i.e. Progressingly Decoding module and Encoding module. When node receives a data packet, Progressingly Decoding module will read the data packet from the corresponding segment the node has received and progressively decode the segment with the newly received one [11]. If the newly received data packet is not linearly dependent with the data packets in the buffer, the progressively decoded result will be put into the buffer; otherwise, the newly received data packet will be dropped. The function of Encoding module is to generate new coded data packets for the neighbors by combining the received data packets in the corresponding segment. All calculation of random linear network coding is realized in Galois field. The field size is 16 (2^4).

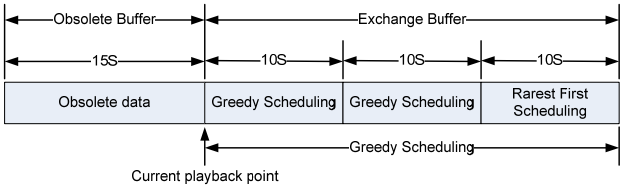

Figure 2. Buffer and the corresponding scheduling algorithm

Figure 2 shows the buffer of a node and the corresponding scheduling algorithm. There is a 30-second exchange buffer before the playback point. In order to provide service for other nodes with later playback point, 15 seconds of played data is stored in the obsolete buffer.

UDP protocol is employed to transmit data packets. Its advantage is the predictable transmission delay which is the absence of TCP protocol. Duplicate data packets will be produced in live P2P streaming system if the arrival time of data packet could not be predicted. However, we can estimate the arrival time of the requested data packets with corresponding streaming scheduling algorithm by using UDP protocol [13]. Because once one peer receives a request packet from others, it will send all the requested data packets it has in the following request interval time uniformly.

The use of UDP protocol brings a new problem on end-to-end bandwidth estimation. Lack of end-to-end bandwidth estimation will lead to that node's streaming scheduling algorithm has no ability to estimate the utilization ratio of each neighbor's uplink bandwidth, which will result in that some nodes' uplink are congested and others are starved. In our system, the scheduling algorithm requests three data packets at most from each neighboring peer per request cycle at the beginning. After that, the limit is set as the data packet rate received in last five seconds plus a constant.

B. Scheduling Algorithm

As shown in Figure 2, the exchange buffer is partitioned into three equal parts. Every part is given one third of R_i in each request cycle, which is the number of data packets that peer i should request per cycle. The first and the second part use the greedy scheduling algorithm to request the data packets from others. The third part uses rarest first scheduling algorithm. The "greedy" scheduling algorithm means the data packet closest to the playback point has the highest priority. If one segment is not decodable but is the closest one to the playback point compared to other

segments in that part, this segment will be requested first. However, "rarest first" gives higher priority to the newest ones [14]. To ensure each node requests enough data packets in each cycle, if the number of requested data packets in the last three stages is less than the number of data packets that the peer should request, the remaining number of data packets will be assigned to the whole exchange buffer by using greedy algorithm. This scheduling algorithm can keep all the segments in the exchange buffer from being empty, promoting the nodes with different playback points to cooperate with each other perfectly.

The cooperation requires each node to access the buffer information of its neighbors as soon as possible. This involves two parameters. One is AGG and the other is the time to send buffer map packet. As for AGG, we set it as two that means that once one node has received two data packets of one segment, this node is ready to serve others in terms of the corresponding segment. While its neighbors have no ideas about the segment before they received buffer map packet. Therefore, we let the node send buffer map packet to its neighbors immediately after it received AGG data packets of one segment. Because the video stream is partitioned into segments, the overhead introduced by buffer map packet is bounded by the neighbor count and the reciprocal of the time that one segment spans. This mechanism can let the neighbors acquire the newest information immediately, reducing the delay further.

The following question is how to request each segment from neighbors. Because coded data packets in the same segment have no difference, previous literatures [10, 11] request data packet based on granularity of segment. One segment stands for one-second video stream, so the granularity is a slightly coarse. In our design, the granularity of scheduling is reduced to data packet level. Once one segment is requested, the pair (segment_number, number_of_packets) will be filled in the request packet for the target neighbor. After the neighbor received the request packet, this node will only transmit the number of data packets it has in its own buffer to its neighbor to avoid redundancy if the requested data packets in (segment_number, number_of_packets) is bigger than that in its own corresponding segment. Otherwise, this neighbor will transmit the requested number of data packets back to it.

The number of data packet requested per cycle needs adjusting according to network situation. One segment is composed of one second's data packets. The request interval is set as half second to tradeoff overhead and delay. Normally, one node can request half segment at most one time, but the node may not be able to achieve streaming rate due to data packet loss. Therefore, the requested number of data packets per cycle is set as 65% of the segment. Moreover, once there is one segment which can not be decoded in the most urgent 10-second buffer, the requested number of data packets in one cycle is increased to (0.65 + (10 - i) * 0.2) times of normal count, where i refers to the number of seconds between current playback point and the non-decodable segment with the most urgent segment number. Therefore, the worst case is that node requests about five times of the normal count per cycle. The worst case is also limited by the sum of end-to-end bandwidth under this node gains.

C. Minimizing Server Bandwidth Costs

Buffering delay and playback delay can be a tradeoff with server bandwidth cost [15]. The higher the server bandwidths costs are, the shorter the buffering delay and playback delay are. Therefore, minimizing server bandwidth costs can reduce buffering delay and playback delay.

The capacity of server uplink bandwidth is an important factor affecting streaming quality. Due to the characteristics of random linear network coding, one node cannot decode the coded data packets before it acquires B data packets with no dependency. Therefore, the server should insert at least B data packets per segment into the streaming system. However, the pull-based scheduling algorithm only cares about the peer's own requirement and all the neighbors of the server compete for the uplink bandwidth of server, which leads to that server may not be able to insert B data packets of some segments into the streaming system. Moreover, the situation deteriorates with the decrease of server uplink bandwidth.

In our system, server-push mechanism is used to minimize the use of server uplink bandwidth. The first question is how many data packets should be pushed? On the one hand, the server should push more than B data packets per segment to the system due to the data packet loss and linear dependency of data packets. On the other hand, the number of pushed data packets cannot be too large. This is because the pushed data will compete for the server uplink bandwidth, which will lead to the loss of data packets and increase of possibility that less than B data packets of some segments are inserted into the system when the server uplink bandwidth is less than the rate of pushed data packets. So in our system, the number of pushed data packets per segment is set to 1.2 times of the number of data packets in one segment, i.e. $1.2 * B$. The extra uplink bandwidth of server is used by pull-based scheduling if any.

The remaining question is to whom the data should be pushed. If all the data packets are pushed to one neighbor of the server, the same situation that happened in the server without push-based scheduling will appear again to this neighboring peer. Therefore, the data should be pushed to different neighbors to make the data as dispersive as possible. However, each neighbors' peer should gain at least AGG data packets of each segment to make them have the ability to relay the data packets to others.

D. Entry Points to Data Requests

How to determine the initial playback point of each peer is a non-trivial question, especially in a large-scale system. Let us look at two extreme conditions. If all uplink bandwidths of users are bigger than streaming rate, i.e. $u_i \geq r_s$ for all i, every node can find two neighbors, one parent and one child, and the playback point can be simply set as slightly latter than its parent. In this way, the buffering delay can be very small, but the largest playback delay is proportional to the number of nodes. Moreover, the extra

uplink bandwidth of peers $(u_i - r_s)$ will be wasted. Another example is that all peers have the same playback point. Although the peers can cooperate with each other in terms of uplink bandwidth, this system is not scalable, because some peers will not be able to get the data before deadline with the increase of peer number due to the transmission delay of data packets between peers.

In our design, the playback point is determined by most-popular mechanism. Each newly joined node requests peers' information from server first then it sends the connection packets to each of them and waits for responses. Once two third of the neighbors' response has been received, the playback point will be determined based on the popularity of segments in its neighbors. The playback point is set as the segment number that is most popular with highest segment number in its neighbors in order to fully utilize the neighbors' uplinks bandwidth and reduce the buffering delay at the same time. Moreover, only the greedy algorithm is applied during buffering process in the scope of exchange buffer to fill in the most urgent segment as soon as possible, leading to a short buffering delay. Once the playback point is determined, data packets will be requested from the playback point.

IV. EXPERIMENTS AND VALIDATIONS

Table II
Parameter settings of simulations

Parameters	Values
Streaming rate	500 kbps
Number of data packets per segment	50
Data packet size	1250 bytes
Number of nodes	600
Simulation time	600 seconds
Peer arriving rate	5 per second
Neighbor count of each peer	25
Peer Resource Index (PRI)	1.2
Uplink bandwidth capacity of server (u_s)	700 kbps
Galois field size	16 (2^4)

Table III
One example for the sources of overhead and the corresponding fraction

Source of Overhead	Overhead Percentage (Compared with streaming rate)
Network coding coefficients	1.8
Redundant packets	2.3
Skipped segments	0.2
Control packets	2.9
Total	7.2

In this section, we check our design by doing simulations. The simulator P2PStrmSim [16] is employed to evaluate the performance of our design. Table II provides main parameter settings used in the following simulations.

In terms of end-to-end delay, we use the measurement result of latency used in [17] to simulate the real under-layer Internet topology. End-to-end bandwidth is distributed uniformly between 50kbps and 2Mbps.

We use the peers with different uplink bandwidth to simulate the heterogeneity of the networks. Three types of peers are employed and their uplink bandwidths are 1Mbps, 384kbps and 128kbps respectively, which simulates three typical DSL peers. Once the peer joined the system, they would stay in the system to the end of simulation. Different Peer Resource Index (*PRI*) can be achieved by adjusting the fraction of each type of peers. PRI is defined as the ratio of total uplink bandwidth to the minimum requirement of download bandwidth, i.e.

$$PRI = \frac{u_s + N * u_i}{N * r_s}$$

In the following simulations, we set $PRI = 1.2$. The exact fractions of each type of peers (1Mbps, 384kbps and 128kbps) are (0.42, 0.42, 0.16).

The uplink bandwidth of server is set to $u_s = 1.4 * r_s = 700kbps$. Table III gives a typical simulation result of overhead. We can see that the overhead caused by all kinds of sources is less than 10%. The pushed data rate is equal to $1.2 * r_s = 600kbps$. Therefore, $u_s = 1.3 * r_s = 650kbps$ is enough for server to stream video. In order to ensure that enough data packets of each segment are streamed to system from server, we further give additional 10% uplink bandwidth to server, making the uplink bandwidth of server equal to $700kbps$.

A. *Simulations for Validating Buffering Delay*

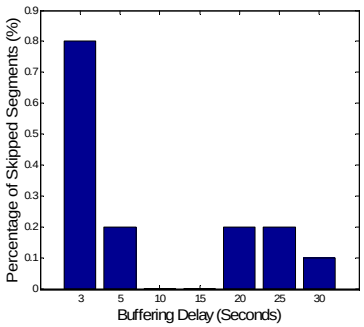

Figure 3. Average streaming quality with different buffering delay

Figure 3 shows the average streaming quality with different buffering delay. A segment will be skipped when the segment cannot get enough data packets to be decoded before its deadline. The shorter buffering delay is, the more segments will be skipped. We can see from Figure 3 that average skipped segments are only 0.2% even when the buffering delay is 5 seconds. The peer can still get healthy streaming quality with 5 seconds of buffering delay.

B. *Simulations for Checking System Scalability and the Ability to Resist Flash Crowd*

In order to make our designs applied in real system, other aspects of network coding based live P2P streaming needs to

be checked, such as system's ability to resist peers' flash crowd, the performance stability when the number of nodes are increased.

Figure 4 shows the streaming qualities when the arriving rate of nodes is increased. The system can maintain good performance when the arriving rate reaches ten nodes per second. The ability to resist peers' flash crowd is proportional to the scale of system. When the scale of our system reaches one million nodes, the system can support more than ten thousand new nodes per second. Furthermore, the uplink bandwidth of server in our system can be set as very slow, only barely meeting the minimum requirement.

As shown in Figure 5, skipped segment fraction keeps low and stable when the node count varies from 600 to 3,000 with buffering delay of 5 seconds. We believe that the streaming quality can also hold good and stable when the node count reaches more than 3,000, basing on the trend.

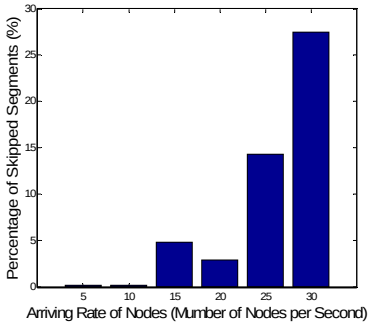

Figure 4. Average streaming quality with different nodes arriving rate

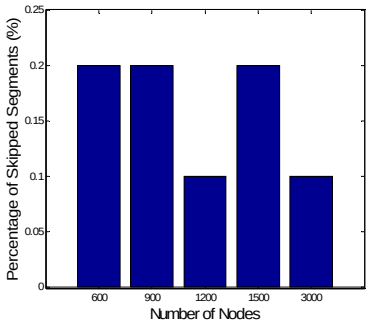

Figure 5. Average streaming quality with different number of nodes

V. CONCLUSIONS AND FUTURE WORKS

Compared to traditional TV services provided by cable companies, the delay of live P2P streaming system is currently too large. In this paper, we applied network coding to live P2P streaming system to reduce buffering delay with our design. The methods used are data packet granularity based scheduling algorithm, server-push method, and intelligent method for determining initial playback point. The simulations results showed that this system could make the buffering delay as low as five seconds when server uplink bandwidth is only 1.4 times of streaming rate while maintaining a healthy streaming quality and ability to resist flash crowd.

For future works, we will use the real trace data to check the performance of our system. Also we will study the playback delay to increase users' quality of experience, especially during live broadcast, such as World Cup. Because the playback time difference between users can be up to several minutes in real live streaming system with large number of users.

VI. REFERENCES

[1] X. Zhang, J. Liu, B. Li, and T.-S. P. Yu, "CoolStreaming/DONet: A Data-driven Overlay Network for Peer-to-Peer Live Media Streaming," in IEEE INFOCOM, Miami, FL, USA, 2005.

[2] "PPLive," http://www.pplive.com/en/.

[3] "PPStream," http://www.ppstream.com/.

[4] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung, "Network Information Flow," IEEE Transactions on Information Theory, vol. 46. No. 4, pp. 1204–1216, July 2000.

[5] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear Network Coding," IEEE Transactions on Information Theory, vol. 49, No. 2, Feburary 2003.

[6] T. Ho, R. Koetter, M. M´edard, D. R. Karger, and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting," in IEEE International Symposium on Information Theory, Yokohama, Japan, 2003.

[7] C. Gkantsidis and P. R. Rodriguez, "Network Coding for Large Scale Content Distribution," in IEEE INFOCOM, Miami, FL, USA, 2005.

[8] C. Gkantsidis, J. Miller, and P. Rodriguez, "Anatomy of a P2P Content Distribution System with Network Coding," in International Workshop on Peer-to-Peer Systems, Santa Barbara, CA, USA, 2006.

[9] D. Ren, Y.-T. H. Li, and S.-H. G. Chan, "On Reducing Mesh Delay for Peer-to-Peer Live Streaming," in IEEE INFOCOM, Phoenix, AZ, USA, 2008.

[10] M. Wang and B. Li, "Lava: A Reality Check of Network Coding in Peer-to-Peer Live Streaming," in IEEE INFOCOM, Anchorage, AK, USA, 2007.

[11] M. Wang and B. Li, "Network Coding in Live Peer-to-Peer Streaming," IEEE Transactions on Multimedia, Special Issue on Content Storage and Delivery in Peer-to-Peer Networks, vol. 9, No. 8, pp. 1554-1567, December 2007.

[12] M. Wang and B. Li, "R2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming," IEEE Journal on Selected Areas in Communications, Special Issue on Advances in Peer-to-Peer Streaming Systems, vol. 25, No. 9, pp. 1655-1666, December 2007.

[13] M. Zhang, L. Sun, and S. Yang, "iGridMedia: Providing Delay-Guaranteed Peer-to-Peer Live Streaming Service on Internet," in IEEE GLOBECOM, 2008.

[14] Y. Zhou, D. M. Chiu, and J. C. S. Lui, "A Simple Model for Analyzing P2P Streaming Protocols," in IEEE International Conference on Network Protocols, Beijing, China, 2007.

[15] T. Small, B. Liang, and B. Li, "Scaling Laws and Tradeoffs of Peer-to-Peer Live Multimedia Streaming," in ACM Multimedia, Santa Barbara, California, USA, October 23-27, 2006, pp. 539-548.

[16] "P2PStrmSim," http://media.cs.tsinghua.edu.cn/~zhangm/.

[17] B. Wong, A. Slivkins, and E. G. Sirer, "Meridian: A Lightweight Network Location Service without Virtual Coordinates," in ACM SIGCOMM, Philadelphia, PA, USA, 2005.