A Study on Reducing Chunk Scheduling Delay for Mesh-based P2P Live Streaming

Zhengjun Chen, Kaiping Xue, Peilin Hong*
Department of Electronic Engineering and Information Science
University of Science and Technology of China (USTC) Hefei, China, 230027
*plhong@ustc.edu.cn

***Abstract*—P2P streaming services have been gaining much success in recent years. In this paper, we address the design of chunk scheduling algorithm which achieves low delay for chunk distribution. We propose a distributed priority-based chunk scheduling algorithm (DPC algorithm). The proposed scheme can approach the minimum delay bound in homogeneous environment. We also extend our DPC algorithm to a general heterogeneous case where peers have different upload bandwidth. Simulation results show the priority-base chunk scheduling algorithm performs close to the theoretical bound, and suits continuity requirement of P2P streaming application.**

Keywords-Peer-to-Peer streaming; Delay Bound; Priority-based Chunk Sceduling

1 INTRODUCTION

P2P technologies have been gaining much success in the application of video streaming. Comparing with traditional Client/Server and IP multicast, the most conspicuous characteristic of P2P technology is that such technology can utilize peers' upload bandwidth effectively to reduce servers' workload. Some P2P multicast systems have been developed to provide live streaming service over the Internet, such as PPLive[1], UUsee[2], ESM[3], etc.

Although the initial successes of P2P streaming are exciting, compared with IP multicast, P2P streaming systems suffer from longer channel startup delay [4], namely, the time a user has to wait before video playback starts. The measurement study [5] on PPLive shows that startup delays can reach as high as a couple of minutes for some channels. In mesh-based P2P streaming, the total delay is composed of two components: 1) **propagation delay**, the path delay from a peer to its neighbors; and 2) **chunk scheduling delay**, due to chunk scheduling policy of a peer with its neighbors. Much work has been focus on constructing overlay topology to reduce propagation delay. In contrast, few studies have addressed the optimization of chunk scheduling delay for P2P streaming.

In this paper, our focus is on the design of a chunk scheduling algorithm that can achieve low delay for chunk distribution. Motivated by snow-ball algorithm [6], we propose a distributed priority-based chunk scheduling algorithm (DPC algorithm). We theoretically demonstrate that DPC algorithm can approach the minimum delay bound in homogeneous environment. For general heterogeneous case, we derive the minimum bound for chunk distribution and show that the performance of DPC algorithm is close to the theoretical bound. At the same time, DPC algorithm can deliver chunks "pseudo sequentially", which is well-suited for the continuity requirement of P2P streaming. The insight of our study can be used to design new P2P systems with shorter startup delays.

The rest of the paper is organized as follows. In Section 2, we briefly present the related works. In Section 3, we present basic model for P2P streaming system. Section 4 describes our distributed priority-based algorithm for homogeneous environment. Section 5 extends our DPC algorithm for heterogeneous case. In Section 6 simulation results are presented. The paper is concluded in Section 7.

2 RELATED WORKS

In this section, we briefly review previous work. Because peers in P2P streaming systems are widely spread across ISPs and countries, there has been much study on optimizing overlay topology to reduce propagation delay. [7], [8] adopt a hierarchically clustered architecture to reduce delivery time. [9], [10] construct the locality-aware overlay network to decrease propagation delay. Our work is orthogonal to them, and able to combine with overlay construction algorithm to achieve low-delay streaming.

Chunk scheduling issue has also been widely studied in mesh-based P2P streaming system. ChainSaw [11] employs a random strategy to decide which packets to request from neighbors. Coolstreaming [12] uses a rare-first strategy to request a segment with less potential suppliers firstly. [13] models the scheduling problem as a classical min-cost network flow problem, and proposes a distributed heuristic algorithm to maximize system throughout. In order to maximize the streaming rate, [14] proposes a simple queue-based chunk scheduling method to achieve high bandwidth utilization in P2P streaming system. However, none of them have addressed minimizing chunk scheduling delay.

The most related work is [6], in which the author derives the minimum delay bound for chunk distribution in P2P streaming systems, and develops a centralized snow-ball algorithm to achieve the minimum bound in homogeneous case. A global coordinator is needed to monitor peers' status and decide chunk dissemination. The snow-ball algorithm can not be applied to general heterogeneous case. In contrast, the proposed DPC algorithm is a decentralized protocol where peers only need to exchange information with its neighbors and make local decision. Global status information does not

Table 1: NOTATIONS

$P_i, i = 0,...,N$	Nodes in the overlay and P_0 is the video source
$U_i, i = 0,...,N$	The upload bandwidth of node P_i
$D_{\min}$	The minimum delay bound for single chunk distribution in homogeneous case
$D^*_{\min}$	The minimum delay bound for single chunk distribution in heterogeneous case
R	The streaming rate

need to be collected. Moreover, DPC algorithm can be easily extended to generous heterogeneous case.

3 System Model

In P2P live streaming, video content is divided into different chunks. Without loss of generality, we normalize chunk size to be one. Each chunk is labeled an increasing sequence number. In order to qualify the time to distribute chunks to all peers in the system, a slotted system is adopted. We consider network core has enough capacity, and hence the bottleneck lies on the network edge. We assume the propagation delay between any two peers is negligible as compare to chunk transmission delay. The video source only uploads one copy of the video content and won't participate in chunk dissemination afterward. We further assume that a fully connected mesh topology is formed among the source and all peers. This assumption will be relaxed in section 6. Table 1 shows some notations that are used in the succeeding sections.

4 DPC Algorithm for Homogeneous Case

In this section, we consider a homogeneous case where the source and all peers have the same upload bandwidth of 1. Communication messages among peers are introduced in Table 2. Peers rely on the information exchanged among them to carry out chunk scheduling. Note that these communication messages are very small in size. Therefore, comparing with chunk transmission, the overhead of exchanging communication messages is negligible.

Figure 2 illustrates the interaction among peers at the beginning of one time slot. The source informs all peers the availability of chunk 3. p_1 and p_2 exchange BM messages (Fig 2(a)). All peers send INTEREST (3) to the source. In addition, p_1 sends INTEREST (2) to p_2 (Fig 2(b)). The source chooses one peer (for example p_1) to send OFFER (3). At the same time, p_2 sends OFFER (2) to p_1 (Fig 2(c)). After receiving OFFER messages, p_1 replies by sending REQUEST (2) to p_2, and sends DECLINE (3) to p_0 (Fig 2(d)). The real chunk transmission will begin subsequently.

4.1 Priority assignment

We now introduce two rules for priority assignment and motivation behind them:

Rule 1: A chunk with lower sequence number has a higher priority. A lower sequence number means this chunk is more stringent to playback. In order to satisfy the continuity requirement, peer should request this chunk first.

Rule 2: A peer with lower LS has a higher priority. A lower LS means this peer falls behind other peers. This peer may be useless to its neighbors. In order to achieve high bandwidth utilization, every peer should serve the lagging peer first.

Based on the above rules, our DPC algorithm contains ***supplier side scheduler*** and ***receiver side scheduler***. As a supplier, a peer receives INTEREST messages from multiple neighbors. The peer must determine to which neighbor it sends OFFER message. As a receiver, a peer might receive multiple OFFER messages. The peer must determine to which supplier it sends REQUEST message.

4.2 Supplier side scheduler

In our design, each peer maintains an ***interest-list*** containing INTEREST messages received in the current slot. Due to upload bandwidth constraint, the supplier has to choose only one peer to provide service. The basic scheduling process can be described as follows: the supplier first ranges peers sending INTEREST messages according to their LS. The peer with lower LS has a higher priority to be served. The supplier will send OFFER message to the first peer in the ***interest-list***. If the selected peer sends back DECLINE message, the supplier will remove the peer from the ***interest-list*** and select the next peer to send a new OFFER message immediately.

The pseudo code of the supplier side scheduling rule is shown in Table 3.

Table 3: SUPPLIER SIDE RULE

```
while ($P_s$.interest-list $\neq \phi$):
{
    m ← the INTEREST message with the lowest sender's LS,
        break tie randomly;
    $P_i$ ← the peer sending m;
    k ← the sequence number of the chunk that $P_i$ interests;
    send OFFER (k) to $P_i$;
    $m^*$ ← the message received from peer $P_i$;
    if $m^*$ == REQUEST (k):
        send chunk k to $P_i$;
        return;
    else if $m^*$ == DECLINE (k):
        delete m from interest-list;
}
```

Table 2: COMMUNICATION MESSAGES AMONG PEERS

HAVE (k)	The video source informs all participating peers that chunk k is available
BM	Buffer map, a fixed length of bit array to reflect the availability of chunks. Each peer exchange BM with its neighbors at the beginning of time slot
INTEREST(k)	If p_j has chunk k that p_i does not have, p_i sends this message to p_j
OFFER (k)	If p_j permits p_i to get chunk k from it, p_j sends this message to p_i
DECLINE (k)	p_i sends this message to p_j to indicate it will not get chunk k from p_j
REQUEST (k)	p_i sends this message to p_j, then p_j will upload chunk k to p_i
LS	The largest sequence number of the chunk possessed by a peer. It can be acquired from the buffer map. LS serves as an indicator of peer's priority. The lower LS of a peer, the higher priority it has

4.3 Receiver side scheduler

A peer may receive multiple OFFER messages within one time slot. OFFER messages providing the same chunk belong to the same set, which is called offer set. Each peer uses an offer-list to record different offer sets. The peer first ranks offer sets based on chunk sequence number. Chunk with lower sequence number has a higher priority to be requested. Then, the receiver ranges the suppliers providing the same OFFER messages. The supplier with lower LS has a higher priority to be chosen. The receiver will send REQUEST message to the selected supplier, and send DECLINE message to other suppliers providing the same OFFER message. A peer can download at most one chunk whose sequence number is larger than its LS. The reason is that the number of chunks a peer can download within one time slot should be equal with the streaming rate in a long time.

The pseudo code of the receiver side scheduling rule is shown in Table 4.

Table 4: RECEIVER SIDE RULE

```
count = 0;
while ($P_i$. offer-list $\neq \phi$):
{
  $M \leftarrow$ the offer set has the lowest chunk sequence number;
  $k \leftarrow$ the lowest chunk sequence number;
  $m \leftarrow$ the OFFER message with the lowest sender's
      LS in $M$, break tie randomly;
  $P_s \leftarrow$ the peer sending $m$;
  send REQUEST ($k$) to $P_s$;
  send DECLINE ($k$) to other peers providing OFFER ($k$);
  delete $M$ from offer-list;
  if $k > P_i.LS$:
      count = count + 1;
  /* a peer can download at most one chunk whose
     sequence number > its LS */
  if count > 0:
      break;
}
```

In Figure 3, we use a sequence of five subfigures to illustrate the process of chunk. Each subfigure shows the buffer status of peers at the end of time slot. There are five peers in the system. and four chunks needed to be distributed to all peers. Each peer does not own any chunk initially. The source uploads one chunk per slot. Black block indicates that peer possesses the corresponding chunk. The source uploads chunk 1 to p_1 in slot 1. In slot 2, the source uploads chunk 2 to p_5. At the same time, p_1 uploads chunk 1 to p_2. Following the same argument, chunk 1 is available in all peers by the end of slot 4. All peers will have chunk 2 by the end of slot5 and so on. One can observe that chunks are distributed to all peers in sequential order, which indicates our DPC algorithm is suitable for continuity requirement of P2P streaming application.

4.4 Optimality of DPC algorithm

We now prove that DPC scheduling algorithm can approach the minimum delay bound for chunk distribution. Given a homogeneous case, the minimum delay for single chunk dissemination is governed by the following formula [6]:

$$D_{\min} = 1 + \lceil \log_2 N \rceil \tag{1}$$

N denotes the number of participating peers.

According to equation 1, it takes at least T time slots to distribute M chunks to all peers.

$$T = M + \lceil \log_2 N \rceil \tag{2}$$

The equation 2 can be illustrated as follows: it takes $M-1$ time slots for the source to upload $M-1$ chunks into the system. For the last chunk, it needs at least $D_{\min}$ time slots to disseminate this chunk to all peers in the system. Thus, we can get equation 2.

Theorem1: Assuming that the transfer time of communication messages can be negligible, DPC scheduling algorithm can achieve the minimum delay bound for chunk distribution.

Proof: We assume that the source uploads chunk k to some peer at time slot $k\,(k \geq 1)$. We call chunk that are not available in all peers processing chunk. We use S_k to denote the set of peers who has processing chunk k. An optimal scheduling should satisfy the sufficient condition $\triangle$ [6]:

$$|S_k(i)| = \begin{cases} 2^{i-k} & k \le i < k + D_{\min} - 1 \\ N & i \ge k + D_{\min} - 1 \end{cases}$$

$|S_k(i)|$ denotes the number of peers who has processing chunk k at the end of time slot i. We show DPC algorithm can satisfy the sufficient condition for the following two cases:

Case 1: $N = 2^{D_{\min}-1}$

Initial condition: before time slot $D_{\min}$, the sufficient condition △ is satisfied.

Based on **supplier side rule**, empty peers have the highest priority. There are enough empty peers in the system before time slot $D_{\min}$. Therefore, the source and non-empty peers can always find empty peers to upload. Based on **receiver side rule**, a peer can only download one chunk whose sequence number is larger than its LS at one time slot. Thus, an empty peer can only obtain one chunk within one time slot. It can be easily verified that the sufficient condition is satisfied before the time slot $D_{\min}$. In addition, we have $\{S_k, k > 0\}$ are pairwise disjoint.

Induction: if at time slot $D_{\min}$, the sufficient condition △ is satisfied, DPC algorithm can guarantee that △ is still satisfied at time slot $D_{\min} + 1$.

At the beginning of time slot $D_{\min}$, according to △, the number of peers with chunk 1 is $N/2$, that is equal with the number of peers without it. According to **receiver side rule**, a peer will first try to download the chunk with lower sequence number. As a result, peers without chunk 1 will accept OFFER messages from S_1. Therefore, chunk 1 will be available in all peers at the end of time slot $D_{\min}$. At the same time, there are $N/2$ OFFER messages from the source and $S_2 \cup ... \cup S_{D_{\min}-1}$. According to **supplier side rule**, Peers in S_1 have higher priority than peers from $S_2 \cup ... \cup S_{D_{\min}-1}$. Thus, every peer in S_1 will get a new chunk with larger sequence number. It is easy to check the sufficient condition △ is still satisfied.

At time slot $D_{\min} + 1$, chunk 2 will be uploaded for the last time. One can treat chunk 2 as chunk 1 in previous time slot. Repeat the same argument above, △ is still satisfied at time slot $D_{\min} + 1$.

Conclusion: DPC algorithm can achieve the minimum delay bound in Case 1.

Case 2: $2^{D_{\min}-2} < N < 2^{D_{\min}-1}$

The difference between **case 1** and **case 2** is that $\{S_k, k > 0\}$ may overlap, that is, some peers can hold two processing chunk simultaneously in **case 2**. However, these peers do not violate the sufficient condition. According to **receiver side rule**, if a peer receives multiple OFFER messages providing the same chunk, it will send REQUEST message to peer with lower LS. For example, in Figure 3, p_2 and p_5 both have two processing chunks at the end of time slot 4. At the beginning of time slot 5, p_1 may receive OFFER (2) from p_3, p_4 and p_5. p_3 and p_4 have a lower LS than p_5. So p_1 will choose p_3 or p_4 to request chunk 2, and while send DECLINE message to p_5. Then, p_5 can upload chunk 4 to p_2. Following similar argument in **Case 1**, △ is satisfied in **Case 2**.

5 DPC Algorithm for Homogeneous Case

In real network environment, different peers may have different uploading bandwidth. In order to adapt to realistic environment, we extend our DPC algorithm to heterogeneous case. We first present the lower delay bound for heterogeneous case, then we show the differences between DPC algorithm in homogeneous and heterogeneous case.

5.1 Lower delay bound for heterogeneous case

In heterogeneous case, one can get the minimum delay bound for single chunk dissemination:

$$D^*_{\min} = 1 + \left\lceil \log_{(U_{\max}+1)} \frac{N}{U_0} \right\rceil \quad (3)$$

Here $U_{\max} = \max\{U_i, i = 0,1,...,N\}$. The equation 3 can be illustrated as follows: the source uploads the chunk to U_0 peers at the first time slot. These peers have upload bandwidth of $U_{\max}$. At the end of second time slot, at most $(U_{\max}+1)U_0$ peers have the chunk. Following the same argument, there will be at most $(U_{\max}+1)^{i-1}U_0$ peers at the end of time slot i. Thus, we can get equation 3.

From equation 3, it takes at least T^* time slots to distribute M chunks to all peers.

$$T^* = \frac{M-1}{U_0} + D^*_{\min} \quad (4)$$

Following similar argument in equation 2, equation 4 can be easily demonstrated.

Note that the delay bound for heterogeneous case cannot be achieved. The purpose of presenting such a lower bound is to provide a benchmark for comparison.

5.2 DPC algorithm for hetergenous case

We need do some small changes to **supplier side scheduler** and **receiver side scheduler** respectively**.** For **supplier side scheduler**, a peer can upload more chunks per slot in heterogeneous case. Therefore, a peer will serve multiple neighbors with lowest LS**,** and while a peer can only serve one neighbor in homogeneous case. For **receiver side scheduler**, a peer can download more than one chunk whose sequence number is larger than its LS. The number of downloaded chunks with larger sequence number is still confined to the

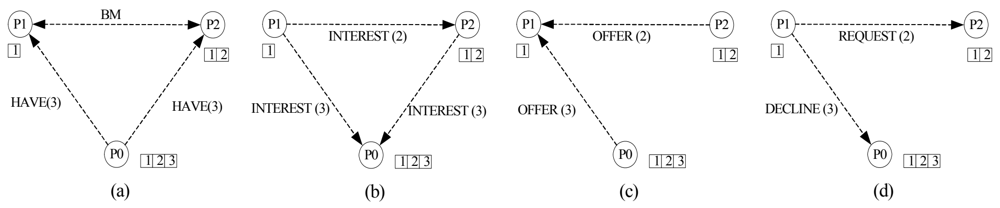

Figure 2: The exchange of communication messages within one time slot

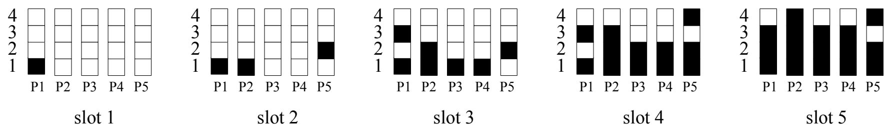

Figure 3: the buffer status of five peers at the end of five time slots

streaming rate. We will evaluate the performance of DPC algorithm in next section.

6 Simulation Results

In this section, we compare DPC algorithm and Rare-first algorithm [12] under homogeneous and heterogeneous environment respectively. The video content is divided into 100 chunks, starting from 1 to 100. A real P2P streaming system may have thousands of peers, and therefore it is not realistic for a peer to maintain too many concurrent connections. In order to simulate a realistic scenario, we relax the assumption of a fully connected topology and choose the following logical network: each peer has L neighbors and these neighbors are randomly selected from all N peers in the system. In our experiments, L is set 20. All our simulations were repeated 20 times and the average values are plotted.

6.1 Homogeneous enviroment

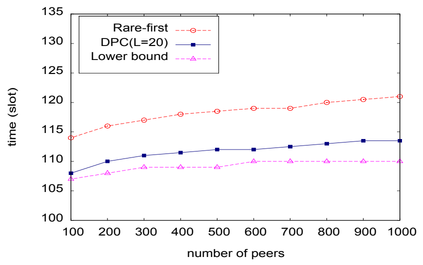

Figure 4: transmission time for 100 chunks to N peers

We first compare transmission time of the whole video content under different scheduling algorithm. The source and peers have upload capacity of 1. Figure 4 illustrates transmission time for each scheduling algorithm as we vary N. The lower bound is given by equation 2. One can observe that the performance of DPC scheduling algorithm is close to the lower bound even with limited neighbors, and its performance is better than Rare-first scheduling method.

6.2 Homogeneous enviroment

In heterogeneous case, the upload bandwidth of the source is set $U_0 = 4$. The streaming rate is $R = 4\ chunks / slot$. The upload bandwidth of a peer is chosen randomly between 2 and 6. The average upload bandwidth of all peers is 4, which is equal to the streaming rate. Figure 5 shows transmission time for each scheduling strategy as we vary N. The lower bound is given by equation 4. One can observe that our DPC algorithm is still close to the lower bound, and perform better than Rare-first algorithm.

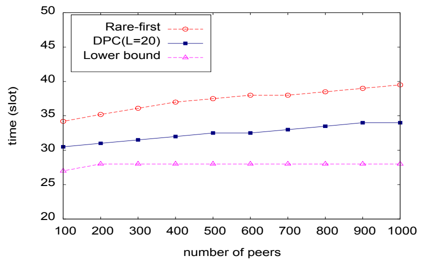

Figure 5: transmission time for 100 chunks to N peers

In the context of P2P streaming, every chunk associates with a deadline, which is the time the peer needs to render this chunk. If a peer receives a chunk after its corresponding deadline, this chunk is useless to playback. Chunks arriving after playback deadline are called late chunks. Figure 6 shows percentage of late chunks for each scheduling algorithm as we vary start-up latency. The number of peers is 1000 in this experiment. One can observe that the percentage of late chunks in DPC algorithm decreases much faster than Rare-first policy. This is attributed to two reasons. On the one hand, our DPC algorithm can disseminate chunks to all peers faster. On the other hand, DPC algorithm can assure chunks are disseminated to all peers in pseudo sequential order, which is very important for P2P streaming application.

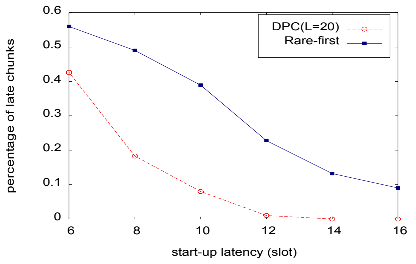

Figure 6: percentage of late chunks as we vary start-up latency

7 Conclusion

In this paper, we propose a distributed priority-based chunk scheduling algorithm for mesh-based P2P live streaming. We theoretically demonstrate DPC algorithm can approach the minimum delay bound in homogeneous environment. Then, we derive the theoretical bound for heterogeneous case and show that the performance of our DPC algorithm is close to the theoretical bound. The inherent property of DPC algorithm is well-suited for streaming application. Results from simulation demonstrate the effectiveness of DPC algorithm. The next step is to test DPC algorithm in real network environment.

8 Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant No. 60772033 and China Next Generation Internet Project NO.CNGI-04-12-ID.

Reference

[1] PPLive. PPLive Homepage. http:// www.pplive.com/.

[2] UUSee inc. http://www.uusee.com/.

[3] Y. hua Chu, S. G. Rao, S. Seshan, and H. Zhang,"A case for end system multicast," IEEE Journal on Selected Areas in Communication (JSAC), Special Issue on Networking Support for Multicast, vol. 20, pp. 1456-1471, Oct. 2002.

[4] Sachin Agarwal, Jatinder Pal Singh, and et al, "Performance and Quality of-Service Analysis of a Live P2P Video Multicast Session on the Internet," in 16th International Workshop on Quality of Service, June 2008.

[5] Xiaojun Hei, Chao Liang, Jian Liang, Yong Liu and Keith Ross, "A Measurement Study of a Large-Scale P2P IPTV System," in IEEE Transactions on Multimedia, vol. 9, pp. 1672-1687, December, 2007.

[6] Yong Liu, "On the Minimum Delay Peer-to-Peer Video Streaming: how Realtime can it be?" in the Proceedings of ACM Multimedia, September, 2007.

[7] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in Proc. ACM SIGCOMM'02, Aug. 2002.

[8] D. A. Tran, K. Hua, and T. Do, "A peer-to-peer architecture for media streaming," IEEE Journal on Selected Areas in Service Overlay Networks, vol. 22, pp. 121–133, Jan 2004.

[9] X. Zhang, Q. Zhang, Z. Zhang, G. Song, W. Zhu, "A construction of locality-aware overlay network: mOverlay and its performance," IEEE J. Select. Areas in Communication, vol. 22, pp. 18-28, Jan. 2004.

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

[11] Vinay Pai, Kapil Kumar, and et al, "Chainsaw: Eliminating Trees from Overlay Multicast," In 4th International Workshop on Peer-to-Peer Systems, February, 2005.

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

[13] Meng Zhang, Yongqiang Xiong, Qian Zhang, and Shiqiang Yang, "On the Optimal Scheduling for Media Streaming in Data-driven Overlay Networks," In the Proceedings of IEEE GLOBECOM 2006, November 2006.

[14] Yang Guo,Chao Liang, Yong Liu, "Adaptive Queue-based Chunk Scheduling for P2P Live Streaming", In the Proceedings of IFIP Networking, May, 2008.