Segment-sending Schedule in Data-driven Overlay Network

Dan Li, Yong Cui, Ke Xu, Jianping Wu
Department of Computer Science and Technology, Tsinghua University,
Beijing, China
{lidan, cy, xuke}@csnet1.cs.tsinghua.edu.cn, jianping@cernet.edu.cn

***Abstract*—Data-driven overlay network is suitable for live-event streaming, because it can provide relatively-continuous streaming even in dynamic environment. In terms of improving streaming quality, prior work covered membership management, buffer map exchange, segment requesting schedule, etc. In this paper, we address the problem of segment-sending schedule on the segment-providing node, which may also affect the streaming quality. The schedule methods we discuss include FIFO schedule, lower-sequence favored schedule, and higher-sequence favored schedule. Simulation results show that if users care playing continuity much more than playing delay, the higher-sequence favored schedule brings the best streaming quality; however, if users care playing delay much more than playing continuity, lower-sequence favored schedule is the preferred choice. Through this work, we find another way to improve streaming quality in data-driven overlay network.**

I. Introduction

With the more and more popular use of Internet, multimedia applications, especially live-event streaming have been growing at fast speeds [1]. In live-event streaming, large amounts of users are interested in the real-time data from a common source. Compared to other applications, live-event streaming demands higher network bandwidth as well as node forwarding capacity.

IP multicast seems to be the ideal technology for live-event streaming [2]. However, IP multicast changes the "unicast" principle of the traditional Internet, and a lot of problems with it, such as multicast management, congestion control, and pricing model, haven't yet been solved well. All these lead to the difficulty of deploying Internet-scale IP multicast.

Researchers subsequently turn to tree-based overlay multicast [4~13], which constructs the multicast trees on hosts instead of on routers. Tree-based overlay multicast overcomes the deployment problem of IP multicast and also has better flexibility. However, overlay nodes are usually unstable. There is a high frequency of node crashing, node joining and node leaving, which leads the overlay topology to change from time to time. This will severely impact the quality of live-event streaming, which has a stringent demand on continuity.

Lately, non-structured overlay network have been put forward, which no longer forwards streaming data along a tree [14~19]. Instead, each node forwards available data to some other nodes that are expecting the data. Data-driven overlay network [14] belongs to this kind of protocols. There are no predefined structures such as parent/children, or upstreaming/ downstreaming in data-driven overlay network. Each node exchanges the data-availability information with partner nodes, and requests data from a suitable partner that can provide the data. If a node receives requests from partner nodes, it should reply by sending the corresponding data. In this way, it is the availability of data that guides the flow directions, not relying on any fixed structures. Data-driven overlay network is especially suitable for live-event streaming because data can be propagated in a relatively continuous way even with node dynamics.

A stream is divided into many segments. Each segment flows from the source node to all the nodes in the data-driven overlay network, by segment-requesting and segment-sending between partner nodes. The segment-requesting and segment-sending between partner nodes is composed of three steps: the segment-requesting node selects a "best" partner node to send the request for a certain segment; the segment request arrives at the requesting queue of the segment-providing node; and the segment-providing node replies the request in the requesting queue by sending the corresponding segment. To improve the streaming quality, prior work has covered aspects including membership management, buffer map exchange, segment-requesting schedule, etc [14]. But no work addressed how a segment-providing node schedules the segment- sending when replying the requests in the requesting queue. Different ways to schedule the segment-sending may result in different streaming qualities.

We discuss the problem of segment-sending schedule in this paper, which primarily includes three methods: FIFO schedule, lower-sequence favored schedule, and higher-sequence favored schedule. Extensive simulations are conducted to compare the streaming quality under each of these three schedule methods. By this work, another way can be found to improve the streaming quality in data-driven overlay network.

The remainder of this paper is organized as follows. In Section II related work is introduced. Section III focuses on the streaming quality of data-driven overlay network. The segment-sending schedule methods are discussed in Section IV. We conduct simulations to compare the streaming quality under FIFO schedule, lower-sequence favored schedule, and higher-sequence favored schedule in Section V. Finally,

1-4244-0355-3/06/$20.00 (c) 2006 IEEE

conclusion and future work are presented in Section VI.

II. Related Work

Because of the deploying difficulty of IP multicast, live-event streaming is primarily realized in the overlay network currently, including both the tree-based overlay multicast and the non-structured overlay network.

Tree-based overlay multicast can be classified into three categories, namely mesh-first, tree-first and implicit approaches [3]. In the mesh-first approach, multicast members first organize themselves into an overlay mesh topology, and then compute the forwarding tree based on the mesh. Examples of this category of overlay multicast protocols include End-System Multicast [4], Scattercast [5], Kudos [6], etc. In contrast, protocols based on the tree-first approach first construct a shared data delivery tree, and subsequently each member discovers a few other members of the multicast group that are not its neighbors on the overlay tree, then establishes and maintains additional control links to these members. The data delivery tree and the additional control links form the control topology. Representations of this category of overlay multicast protocols include Yoid [7], Host Multicast [8], ALMI [9], Switch Tree [10], etc. Protocols using the implicit approach create a control topology with some specific properties. The data delivery path is implicitly defined on this control topology by some packet forwarding rule which leverages the specific properties of the control topology to create loop-free multicast paths. overlay multicast protocols belonging to this category include NICE [11], Scribe [12], Bayeux [13], and so on.

Non-structured overlay network does not rely on a tree to distribute data. This kind of network includes gossip-based protocols and data-driven overlay network. In gossip-based protocols [15~19], each node forwards available data to a set of randomly selected nodes. But in data-driven overlay network [14], each node maintains several partner nodes, and data is transmitted among partner nodes, eventually to the whole overlay network. Compared to gossip-based protocols, the advantage of data-driven overlay network is that data is flowing in a request-reply way, thus there is not any redundant data consuming the network bandwidth. To improve streaming quality, membership management, buffer map exchange, and segment-requesting schedule algorithm was discussed in [14]. However, how to schedule the segment-sending on the segment-providing node is not covered in previous literature. In fact, segment-sending schedule algorithm is also important for improving streaming quality, which is addressed in this paper.

III. Streaming Quality

The streaming data transmitted in the data-driven overlay network can be divided into segments with uniform length. A buffer map can represent the information of available segments on a node. Each node periodically exchanges its buffer map with its partner nodes, and decides from which partner node to fetch a certain segment. If there are multiple partner nodes holding the same expected segment, various ways can be chosen to select the segment-providing node, for instance, the one with the highest bandwidth. This is called the segment-requesting schedule.

Each node maintains a buffer window, which is used to store a number of continuous segments of the stream. A node can only request for segments that are within the buffer window. After having played a segment, the segment could be discarded and the buffer window is moved upwards. Suppose the buffer window size of each node is B, which is usually much less than the total segment number of the stream. To make the streaming more continuous, each node usually doesn't begin to play the stream immediately after receiving the first segment. Instead, it waits for the arrival of the first several segments and then begins to play the stream. The number of segments each node waits for before playing the stream is called the playing waiting segment number, denoted by W. Obviously, there should be the relationship $W \le B$.

For ease of discussion, some denotations are made here.

Segment Arrival Time $Ar(s,i)$—The time slot of the last bit of segment s (where s also represents the sequence number of the segment) arriving at node i.

Segment Playing Time $Pl(s,i)$—The time slot of beginning playing segment s on node i.

Segment Playing Deadline $De(s,i)$—The time slot of the end of playing segment $s-1$ on node i.

We define the following concepts to express the streaming quality each node i observes from receiving the segments of a stream.

Definition 1. *Playing Continuity*

Suppose the set of all segments of the stream is A, the segment arrival time of segment s on node i is $Ar(s,i)$, the segment playing deadline of segment s on node i is $De(s,i)$. The playing continuity on node i is defined as $Pc(i) = \frac{|\{s \mid s \in A, and\ Ar(s,i) \le De(s,i)\}|}{|A|}$.

Definition 2. *Playing Delay*

Suppose the set of all segments of the stream is A, the source node of the stream is src, and the playing time of segment s on node i is $Pl(s,i)$. The playing delay on node i is defined as $Pd(i) = \frac{\sum_{s \in A}(Pl(s,i) - Pl(s,src))}{|A|}$.

Definition 3. *Playing Delay Ratio*

Suppose the playing delay of the stream on node i is $Pd(i)$, the playing time of the whole stream is T. The playing delay ratio on node i is defined as $Pdr(i) = \frac{Pd(i)}{T}$.

The overall streaming quality a user observes is determined by both the playing continuity and the playing delay ratio (or playing delay). In reality, users may care the playing continuity more than the playing delay ratio, or the contrary. Let $\alpha (0 \le \alpha \le 1)$ be the user preference index to playing continuity, and $1-\alpha$ be the user preference index to playing

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.

delay. $\alpha > 0.5$ means users care playing continuity more than playing delay ratio, $\alpha < 0.5$ means users care playing delay ratio more than playing continuity, and $\alpha = 0.5$ means users care playing continuity the same as playing delay ratio.

Definition 4. *User Satisfaction Degree*

Suppose the playing continuity on node i is $Pc(i)$, the playing delay ratio on node i is $Pdr(i)$, and the user preference index to playing continuity is $\alpha(0 \le \alpha \le 1)$. The user satisfaction degree of the stream on node i is defined as $U(i) = \frac{(Pc(i))^{3\alpha-2\alpha^2}}{(1+Pdr(i))^{1+\alpha-2\alpha^2}} * 100\%$.

The streaming quality of the stream on node i can be directly quantified by $U(i)$. If α varies as 1, 0.75, 0.5, 0.25, and 0, we get $U(i)$ as follows.

$$U(i) = \begin{cases} Pc(i) * 100\%, & \alpha = 1 \\ \frac{Pc(i)^{\frac{9}{8}}}{(1+Pdr(i))^{\frac{5}{8}}} * 100\%, & \alpha = 0.75 \\ \frac{Pc(i)}{1+Pdr(i)} * 100\%, & \alpha = 0.5 \\ \frac{Pc(i)^{\frac{5}{8}}}{(1+Pdr(i))^{\frac{9}{8}}} * 100\%, & \alpha = 0.25 \\ \frac{1}{1+Pdr(i)} * 100\%, & \alpha = 0 \end{cases}$$

It is easy to see that if $\alpha = 1$, $U(i)$ equals to 100% only when $Pc(i) = 1$; if $\alpha = 0$, $U(i)$ equals to 100% only when $Pdr(i) = 0$; in other cases, $U(i)$ equals to 100% when $Pc(i) = 1$ and $Pdr(i) = 0$.

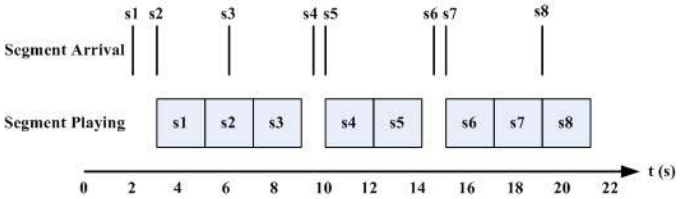

Fig. 1. Segment arrival and segment playing

Figure 1 shows an example where a node receives a stream of 8 segments. The playing length of each segment is 2 seconds and the playing waiting segment number is 2. According to the definitions above, the playing continuity is 0.75, the playing delay is 4s, and the playing delay ratio is 0.25. If the user preference index to playing continuity is 0.5, the user satisfaction degree is 60%.

IV. Segment-sending Schedule Methods

In data-driven overlay network, each node exchanges buffer map with partner nodes and decides from which partner node to request a certain segment. The request arrives at the requesting queue on the segment-providing node. At the time when the segment-providing node replies to the requests, if there are multiple requests in the requesting queue, it has to schedule the segment-sending. There are two issues to consider: how to send the replying segments if the priority of each request is assigned, and how to assign the priority of each request in the requesting queue.

There are two ordinary methods to schedule the segment sending, both in favor of higher-priority requests. One method is to send one segment after another according to the priorities of the requests with the whole sending bandwidth, which is called the one-after-another method. The other method is to allocate the sending bandwidth proportional to the priorities of the requests and send the segments simultaneously, which is called the proportional-bandwidth method.

A segment s is available on node i only after the segment arrival time $Ar(s,i)$, either for playing or for forwarding. By queuing theory, the one-after-another method is better than the proportional-bandwidth method in that the requests with higher priorities will receive better services without sacrifice to requests with lower priorities. If there are new requests coming when the segment-providing node is sending a segment to some partner node, the new requests are inserted into the requesting queue, and the next-turn segment-sending schedule should be conducted only after the current sending is completed.

The key problem here is how to assign the priorities of the requests in the requesting queue. The simplest way is to assign the requests arriving earlier with higher priorities, which we call FIFO schedule. However, in data-driven overlay network which transmits live-event stream, the sequence number of the segment each node requests for should be considered. Higher-sequence favored or lower-sequence favored streaming may result in different streaming qualities. The method of assigning lower-sequence segments with higher priorities is called lower-sequence favored schedule, and the method of assigning higher-sequence segments with higher priorities is called higher-sequence favored schedule.

The procedure of segment-sending schedule on the segment-providing node can be shown in Figure 2. Requests arrive at the segment-providing node, and the providing node assigns the priorities of these requests by one of the three priority assignment methods we suggest. After that, the segment-providing node replies to the requests by sending the

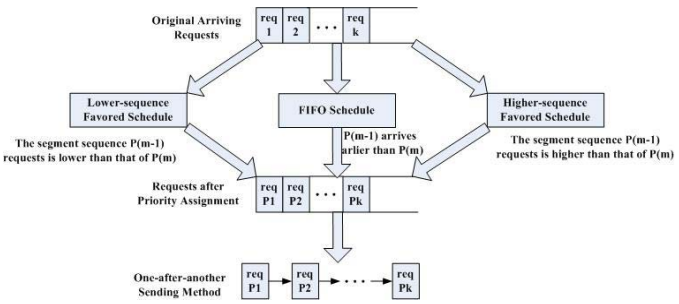

Fig. 2. Segment-sending Schedule

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.

TABLE I. Segment Sending

```
    // send the corresponding segments to satisfy the requests
1   void segmentSend (){
2     while (true)
        // there is request in the requesting queue
3       if !queueEmpty()
          // get the requesting node and the requesting segment
          // of the request at the queue head
4         queueOutHead( &seg , &node )
          // send the segment to the requesting node
5         segmentSend( seg , node )
6       end if
7     end while
8     return
9   }
```

TABLE II. Priority Assignment of FIFO Schedule

```
    // receive a segment-request from a partner node
1   void reqMsgRcv ( reqMsg ){
2     k ← reqMsg.source
3     s ← reqMsg.segment
      // insert the request at the tail of the queue
4     queueInTail ( k , s )
5     return
6   }
```

TABLE III. Priority Assignment of Lower-sequence Favored Schedule

```
    // receive a segment-request from a partner node
1   void reqMsgRcv ( reqMsg ){
2     k ← reqMsg.source
3     s ← reqMsg.segment
4     Pr(k,s) = 1/(s+1)    // priority assignment
      // insert the request into the queue based on its priority,
      // higher priority nearer to the queue head
5     queueInPr( k , s , Pr(k,s) )
6     return
7   }
```

TABLE IV. Priority Assignment of Higher-sequence Favored Schedule

```
    // receive a segment-request from a partner node
1   void reqMsgRcv ( reqMsg ){
2     k ← reqMsg.source
3     s ← reqMsg.segment
4     Pr(k,s) = s+1    // priority assignment
      // insert the request into the queue based on the priority,
      // higher priority nearer to the queue head
5     queueInPr ( k , s , Pr(k,s) )
6     return
7   }
```

corresponding segments in a one-after-another way.

The segment sending algorithm is illustrated in TABLE I. It just gets the request at the head of the requesting queue, and sends the corresponding segment. If there are n requests in the requesting queue, the computing complexity is $O(n)$.

The priority assignment algorithm of FIFO schedule is illustrated in TABLE II. We can see that it needn't really assign the priorities to the arriving requests, but simply puts the newly-arriving request at the tail of the requesting queue. If there are n requests arriving, the computing complexity is $O(n)$.

Both lower-sequence favored schedule and higher-sequence favored schedule should assign the priority of the newly-arriving request and insert the request into the requesting queue by its priority, higher priority nearer to the queue head. The sequence-priority mapping functions are arbitrary if only the request for higher-sequence segments are assigned with higher priority in higher-sequence favored schedule and it is contrary in lower-sequence favored schedule. Let $Pr(i,s)$ be the priority assigned to request from node i for segment s. Example mapping functions are $Pr(i,s)=1/(s+1)$ for lower-sequence favored schedule, and $Pr(i,s)=s+1$ for higher-sequence favored schedule. The priority assignment algorithms of lower-sequence favored schedule and higher-sequence favored schedule are shown in TABLE III and TABLE IV, respectively. If there are n requests arriving, the computing complexity of both the algorithms is $O(n\log n)$.

The computing complexity of FIFO schedule is the lightest, but since it doesn't consider the sequence number of the requesting segments, it may not achieve the optimal streaming quality as users care.

Since the stream is played after the first several segments arrive, if lower-sequence segments are favored when streaming, the beginning time of playing the stream on many nodes may be earlier. However, the earlier arrival of higher-sequence segments may improve the playing continuity to some extent, because after the lower-sequence segments arrive, the stream can be played in a relatively continuous way. Further comparison of these three schedule methods are made in the section below.

V. Simulations

We conduct extensive simulations to compare the streaming quality of FIFO schedule, lower-sequence favored schedule, and higher-sequence favored schedule. The scale of the simulated data-driven overlay network is of 500 nodes. Based on the suggestion in [14], we here let each node has 4 partner nodes. The stream propagated in the network is composed of 5000 segments. We observe the average user satisfaction degree of all nodes in different cases, each under FIFO schedule, lower-sequence favored schedule, and higher-sequence favored schedule.

Firstly, let the buffer window size be 60 ($B=60$), and the playing waiting segment number vary as 10, 20, 30, 40, 50 and 60. If the user preference index to playing continuity $\alpha=1$, the user satisfaction degree is illustrated in Figure 3.1; if $\alpha=0$, the user satisfaction degree is illustrated in Figure

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.

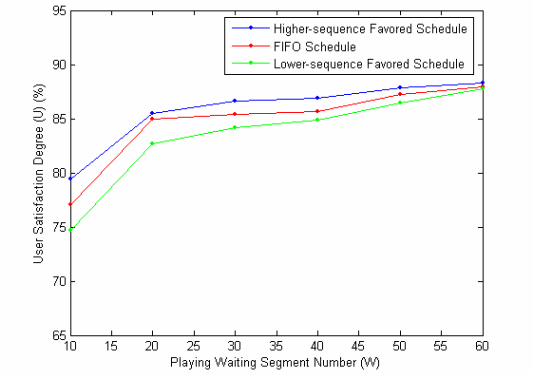

Fig. 3.1. User satisfaction degree (U) over different playing waiting segment number (W) (B=60, α=1)

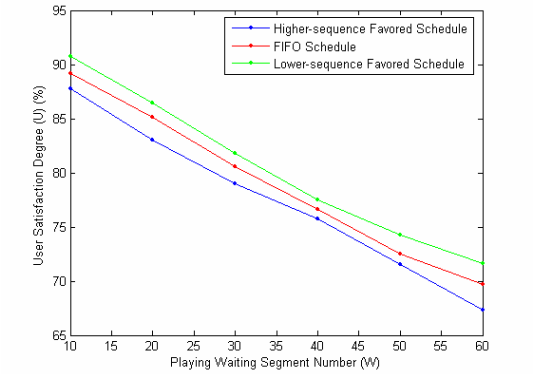

Fig. 3.2. User satisfaction degree (U) over different playing waiting segment number (W) (B=60, α=0)

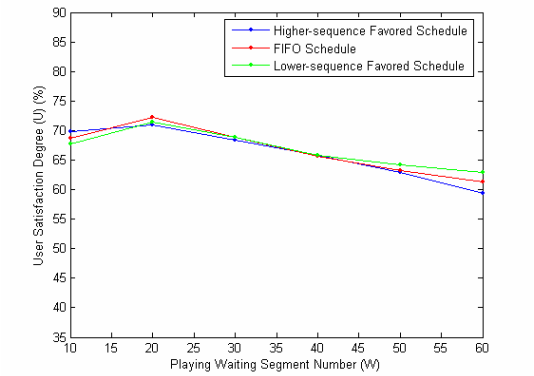

Fig. 3.3. User satisfaction degree (U) over different playing waiting segment number (W) (B=60, α=0.5)

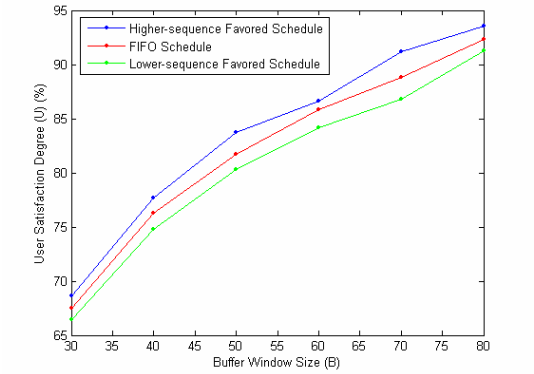

Fig. 4.1. User satisfaction degree (U) over different buffer window size (B) (W=30, α=1)

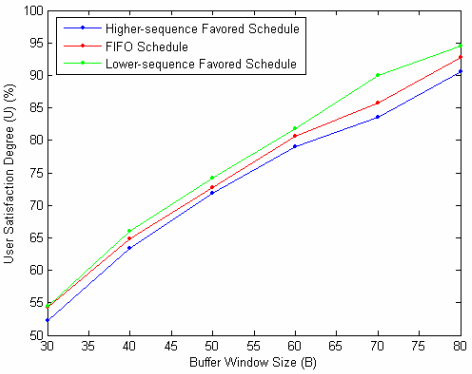

Fig. 4.2. User satisfaction degree (U) over different buffer window size (B) (W=30, α=0)

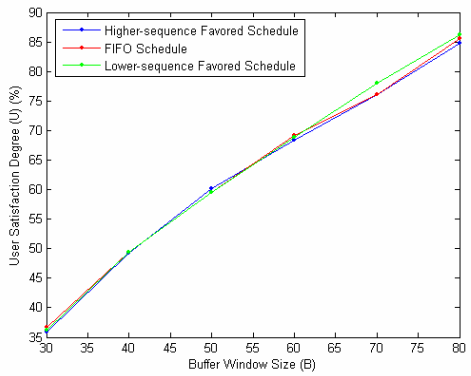

Fig. 4.3. User satisfaction degree (U) over different buffer window size (B) (W=30, α=0.5)

3.2; and if $\alpha = 0.5$, the user satisfaction degree is illustrated in Figure 3.3.

From Figure 3.1, we see that if users care playing continuity only, the higher-sequence favored schedule brings the highest user satisfaction degree among the three methods, the next is the FIFO schedule, and the worst is lower-sequence favored schedule. In this case, the user satisfaction degree increases with the growth of playing waiting segment number. Figure 3.2 shows that if users care playing delay only, the lower-sequence favored schedule brings the highest user satisfaction degree, the next is the FIFO schedule, and the worst is higher-sequence favored schedule. Under this situation, the user satisfaction degree decreases with the growth of the playing waiting segment number. Figure 3.3 illustrates that if users care playing continuity as much as playing delay, there seems not obvious difference among these three schedule methods.

Now we let the playing waiting segment number be 30

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.

($W = 30$), and the window buffer size vary as 30, 40, 50, 60, 70 and 80. If the user preference index to playing continuity $\alpha = 1$, the user satisfaction degree is illustrated in Figure 4.1; if $\alpha = 0$, the user satisfaction degree is illustrated in Figure 4.2; and if $\alpha = 0.5$, the user satisfaction degree is illustrated in Figure 4.3.

Figure 4.1 demonstrates that if users care playing continuity only, the higher-sequence favored schedule brings the highest user satisfaction degree among these three methods, the next is FIFO schedule, and the worst is lower-sequence favored schedule. On this occasion, the user satisfaction degree increases with the growth of buffer window size. Figure 4.2 shows that if users care playing delay only, the lower-sequence favored schedule brings the highest user satisfaction degree, the next is FIFO schedule, and the worst is higher-sequence favored schedule. The user satisfaction degree also increases with the growth of the buffer window size in this case. From Figure 4.3, we still can't see any obvious difference among these three schedule methods.

As a whole, through simulations, we get the results that if users care playing continuity much more than playing delay, the higher-sequence favored schedule method brings the highest streaming quality; however, if users care playing delay much more than playing continuity, the lower-sequence favored schedule method is the best.

Therefore, in addition to increasing the buffer window size or changing the playing waiting segment number, selecting the suitable segment-sending schedule method can also improve the streaming quality users care in data-driven overlay network.

VI. Conclusion and Future Work

Data-driven overlay network is especially suitable for live-event streaming because it can tolerant the node dynamics. In data-driven overlay network, each node exchanges data-availability information with partner nodes and fetches a certain segment from a suitable partner node. This paper discusses the problem of segment-sending schedule when the segment-providing node receives multiple segment requests, which is not considered by prior work. We compare the streaming quality under FIFO schedule, lower-sequence favored schedule, and higher-sequence favored schedule. Simulation results show that if users care playing continuity much more than playing delay, higher-sequence favored schedule brings the highest streaming quality; and if users care playing delay much more than playing continuity, lower-sequence favored schedule is the preferred choice. Thus, we find another way to improve the streaming quality in data-driven overlay network.

Live-event streaming is used more and more widely. How to ensure efficient, reliable and truthful streaming and improve the user satisfaction degree is a hot topic. It will still be the aims of our future work.

Reference

[1] J. Liu, B. Li, and Y. Q. Zhang, "Adaptive video multicast over the Internet," IEEE Multimedia, 10(1):22-31, Jan/Feb, 2003

[2] S. E. Deering, "Multicast Routing in Internetworks and Extended LANs", Proc. of ACM SIGCOMM 1988, Stanford, CA, USA, Aug 1988

[3] S. Banerjee, and B. Bhattacharjee, "A Comparative Study of Application Layer Multicast Protocols", http://www.cs.wisc.edu/~suman/pubs.html

[4] Y. H. Chu, S. G. Rao, and H. Zhang, "A Case for End System Multicast", Proc. of ACM Sigmetrics 2000, Santa Clara, CA, USA, Jun 2000

[5] Y. Chawathe, "Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service", Ph.D. Thesis, University of California, Berkekey, Dec 2000

[6] S. Jain, R. Mahajan, D. Wetherall, and G. Borriello, "Scalable Self-Orgnizing Overlays", Technical report, Washington University, 2000

[7] P. Francis, "Yoid: Extending the Internet Multicast Architecture", White Paper, http://www.icir.org/yoid

[8] B. Zhang, S. Jamin, and L. Zhang, "Host Multicast: A Framework for Delivering Multicast to End Users", Proc. of IEEE INFOCOM 2002, New York, USA, Jun 2002

[9] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: An Application Level Multicast Infrastructure", Proc. of USENIX Symposium on Internet Technologies and Systems, San Francisco, California, USA, Mar 2001

[10] D. A. Helder, and S. Jamin, "End-host Multicast Communication using Switch-trees Protocols", Proc. of Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems, Berlin, Germany, May 2002

[11] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast", Proc. of ACM SIGCOMM 2002, Pittsburgh, PA, USA, Aug 2002

[12] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "SCRIBE: A Large-scale and Decentralized Application-level Multicast Infrastructure", IEEE Journal on Selected Areas in Communications, 20(8):100-110, 2002

[13] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, and R. H. Katz, "Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination", Proc. of International Workshop on Network and Operating Systems Support for Digital Audio and Video, Port Jefferson, NY, USA, Jun 2001

[14] X. Zhang, J. Liu, B. Li, and T. P. Yum, "Data-Driven Overlay Streaming: Design, Implementation, and Experience", Proc. of IEEE INFOCOM 2005, Miami, Florida, USA, Mar 2005

[15] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan, "Resilient multicast using overlays," Proc. ACM SIGMETRICS 2003, San Diego, CA, USA, Jun 2003

[16] P. Eugster, R. Guerraoui, A.M. Kermarrec, and L. Massoulie, "From epidemics to distributed computing," IEEE Computer, 37(5):60-67, 2004

[17] S. Jin and A. Bestavros, "Cache-and-relay streaming media delivery for asynchronous clients," Proc. of NGC 2002, Boston, MA, USA, Oct 2002

[18] Y. Cui, B. Li, and K. Nahrstedt, "oStream: asynchronous streaming multicast," IEEE Journal on Selected Areas in Communications, 22(1): 91-106, 2004

[19] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2Cast: peer-to-peer patching scheme for VoD service," Proc. of WWW 2003, Budapest, Hungary, May 2003

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.