I.J. Intelligent Systems and Applications, **2011, 3, 1-10**
Published Online May 2011 in MECS (http://www.mecs-press.org/)

A Dynamic Self-Adjusted Buffering Mechanism for Peer-to-Peer Real-Time Streaming

Jun-Li Kuo, Chen-Hua Shih and Yaw-Chung Chen
Computer Science, National Chiao Tung University, HsinChu, Taiwan
Email: estar.cs95g@nctu.edu.tw; shihch@csie.nctu.edu.tw; ycchen@cs.nctu.edu.tw

***Abstract*—Multimedia live stream multicasting and on-line real-time applications are popular recently. Real-time multicast system can use peer-to-peer technology to keep stability and scalability without any additional support from the underneath network or a server. Our proposed scheme focuses on the mesh architecture of peer-to-peer live streaming system and experiments with the buffering mechanisms. We design the dynamic buffer to substitute the traditional fixed buffer.**

According to the existing measurements and our simulation results, using the traditional static buffer in a dynamic peer-to-peer environment has a limit of improving quality of service. In our proposed method, the buffering mechanism can adjust buffer to avoid the frozen or reboot of streaming based on the input data rate. A self-adjusted buffer control can be suitable for the violently dynamic peer-to-peer environment. Without any support of infrastructure and modification of peer-to-peer protocols, our proposed scheme can be workable in any chunk-based peer-to-peer streaming delivery. Hence, our proposed dynamic buffering mechanism varies the existing peer-to-peer live streaming system less to improve quality of experience more.

***Index Terms*—multimedia, live streaming, real-time service, peer-to-peer, buffer control, IPTV**

I. Introduction

With advancement and development of network technology, more and more on-line applications appended to the Internet are popular among many people in the recent years. In addition, some services limited to the bandwidth challenges become practicable and feasible because of an increase of broadband access. For example, the service of multimedia multicast satisfies hundreds or thousands of users simultaneously. More and more people enjoy multimedia services easy from youTube or many Web 2.0 websites. It is easy that users just click a mouse to access Internet as well as press a remote controller to watch Internet television.

The Internet television owns more outstanding advantages than traditional television, such as rich and various programs as well as watching television anytime and everywhere via wireless mobile device. Even a successful IPTV (Internet Protocol TeleVision) system can allow arbitrary users to create the user-generated content. Several researchers forecast and expect that IPTV will be the television in the next generation [1].

However, the traditional server-client system cannot afford the high bandwidth of multimedia streaming for IPTV; hence, many researchers considered that peer-to-peer (P2P) architecture should be a good network overlay to broadcast multimedia streaming based on four reasons:

(1) P2P technology has been developed successfully for file sharing and has been demonstrated to heal with the accessed utilization for thousands of users simultaneously. P2P technology also owns the scalability, such as BitTorrent and eDonkey.

(2) P2P can use the resource of each peer adequately, especially bandwidth resource. Peers can cooperate with each other to share available bandwidth for the scalable applications.

(3) P2P content distribution uses an application layer multicast without any additional support from the underneath network.

(4) P2P servicing model greatly reduces server cost. In summary, using P2P overlay has become an increasingly popular approach for live streaming over the Internet.

Presently, the live media distribution is called “P2P live streaming” and is different from video on demand (VoD). P2P live streaming service usually provides in-time programs, such as live ball game, first-hand stock information, or the latest news. On the other hand, VoD service often provides the rebroadcasted movies or series. Obviously, the limit of real-time matters must be considered more in live streaming than in VoD. Therefore, various kinds of discussed issues in the P2P live streaming system were derived and referred to quality of service (QoS). In this paper, we consider in the users’ opinions instead of the developers’ opinions.

The structures of P2P live streaming system can be roughly differentiated between tree-based structure and mesh-based structure. According to the data-driven way (i.e. how to acquire data), we can call them *tree-push* and *mesh-pull* respectively. Tree-push means that the ancestors usually push available source to their descendants. On the contrary, mesh-pull means that the peer collects available source by itself and competes against other peers. Additionally, many recent studies addressed the concept about hybrid; hence, push and pull already have combined each other or worked together [2]. In our practice and experience, tree and mesh own their

individual advantages respectively, maybe the programming developers take trade-off into account to select one structure among them to implement a P2P live streaming system:

(1) Single-tree structure is simple and efficient but vulnerable to dynamics;
(2) Multiple-tree structure is more resilience but more complex than single tree;
(3) Mesh structure is more robust but occurs to longer delay and control overhead.

So far, most of the previous research modifies application multicast to improve the quality of experience (QoE) [12] for the audiences and almost probes into (1) Overlay structure, (2) Peer adaptation, (3) Data scheduling, and (4) Hybrid reconstruction.

(1) Overlay structure. Briefly, overlay structure includes basic architecture and data delivery strategy. Basic architecture of mesh consists of swarming and gossip. Swarming originates from the swarm of BitTorrent, means the group of peers that all share the same torrent. Therefore, swarming is like as the BT-liked protocol. Besides, gossip is similar with swarming, and peer selection of gossip protocol is freer than swarming.

    Data delivery strategy varies according to above-mentioned basic architecture. In general, the centralized delivery strategy has the higher efficiency but the less scalability than the distributed delivery strategy. The developers try to find the most suitable overlay structure for P2P live streaming.

(2) Peer adaptation. Peers join or leave arbitrarily, also called peer churn. No matter what overlay structure is used, P2P streaming system needs a recovery mechanism to ensure a reliable work after peer churn. The overlay is reconstructed to maintain an adequate amount of source for all peers. In addition, peers can change partners to get the better QoS. These behaviors are generally called peer adaptation.

(3) Data scheduling. In mesh-pull scheme, a successful download must decide who to be requested, which one chunk[1] to get, and how to get available chunk. A good data schedule can select a workable peer to get a useful chunk to maintain QoS.

    The well-known data scheduling includes rarest first and optimistic unchoke in BitTorret, which is much efficient and nice in P2P file sharing. However, the data schedule of BitTorret is not suitable for live streaming service. A data schedule for P2P live streaming takes basically the prompt delivery and buffered queue into account.

(4) Hybrid reconstruction. Several inherent shortcomings of mesh overlay are difficult to overcome, so some researchers consider that an integration of mesh and tree is a resolution for the challenges from mesh [11].

Making these four alterations mentioned above in an old and an existing system is resource-consuming because it is necessary to modify the original multicast, to alter the real-time protocol, or to increase the overhead of each peer. Hence, these issues can only be considered when developing a new P2P streaming system. Furthermore, so far, previous studies never aimed at buffer control to improve QoS.

To vary the existing peer-to-peer live streaming system less but improves quality of experience more, our proposed scheme focuses on the mesh architecture of P2P live streaming system and experiments with the buffering mechanisms. P2P swarming[2] based on mesh must have a component to buffer video data for the smoothness. We enlarge or shorten the buffer size with the variation of data rate to prevent from the unstable source due to peer churn. Only buffer process of client is modified to improve the global efficiency of streaming system without any additional overhead and modification.

The rest of this paper is organized as follows. In Section 2, we present the buffer introduction of PPLive and Cool-Streaming. In Section 3, we discuss our proposed scheme in detail. Simulation results are presented in Section 4, and we conclude the work in Section 5.

II. Related Work

Buffer control is indispensable to a P2P network, and it is also one component of P2P live streaming system to avoid jitter, especially in swarming delivery under randomly mesh architecture. The buffer of P2P live streaming is smaller than 10 MB of each peer according to the measurement, and the buffer size equally reserves the video data for 200 seconds around [3]. Two reasons why mesh structure use buffer mechanism are necessary. First, buffer keeps video playing smoothly. As a result, every peer must buffer a sufficient quantity of video data and ensure the stable source to begin playing back. Second, buffer mitigates the possible challenges because of P2P churns. In general, peer churn often causes an interruption or a break of data delivery; hence, the buffered data can deal with the emergency before the overlay recovery.

A. The Buffer Observation of PPLive

PPLive is a popular Internet application for P2P streaming, but it is a pity that PPLive software is not open to academia. Therefore, we only deduce its principles from the monitoring measurements [3].

[1] A chunk is a unit of video data block.

[2] Swarming means peers all cooperate to get the same work in a group.

Depending on Figure 1, PPLive starts to establish and set up buffer after opening to execute. And then PPLive program requires the memory space to buffer data, the initial buffer size equals approximately 100 chunks. Buffer size grows with time until up to the size of 1000 chunks, and then it stops growing or grows slowly. The reason of growing buffer size slowly is that 1000 chunks are enough sufficient to play back; thus, enlarging buffer space is not important to do immediately. The coordination with Figure 1 and Figure 2, we can know that buffer size stops growing entirely when it is the size of 1100 chunks. In addition, the change of network parameters will not vary the buffer size; thus, the buffer control of PPLive is the type of static fixed buffer.

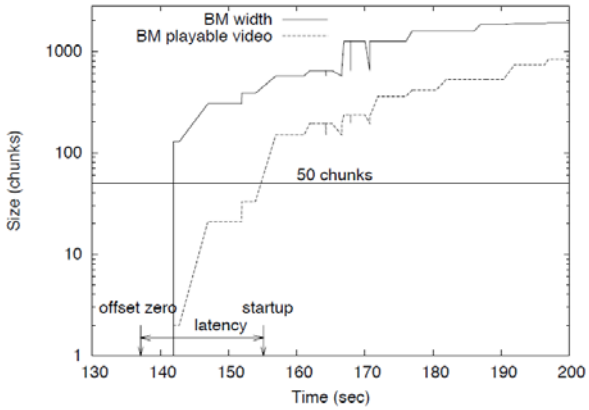

Figure 1. Buffer size vs. buffer saturation at initial time

However, Figure 2 reveals a drawback in fixed buffer mechanism. When the incoming streaming is less or unstably provided to buffer, the buffer still holds a fixed size. This inflexible method may lead to video frozen, video skipping, and program rebooting, and then further impact the users' QoE. These three situations all result from a lack of buffered data; in other words, the chunks cannot arrive at the destination before the playback deadline.

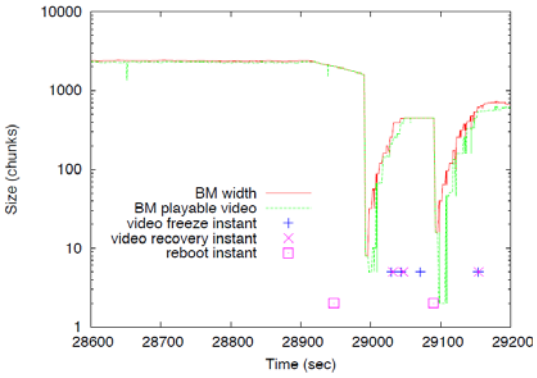

Figure 2. Buffer variation

As Figure 3 illustrates, the occasional nulls cause the small gaps of the in-order buffer, and the gaps result in video skipping. If a block of chunks cannot be collected but the communication of peers still keep well, the video player is frozen until the available chunks meet the playable region of buffer. The reason of video frozen is the less availability of stringent chunks due to the network congestion or a bad algorithm of chunk selection. However, if peer leaving leads to a starvation and peer adaptation cannot handle the trouble immediately, the user's streaming programming cannot but reboot.

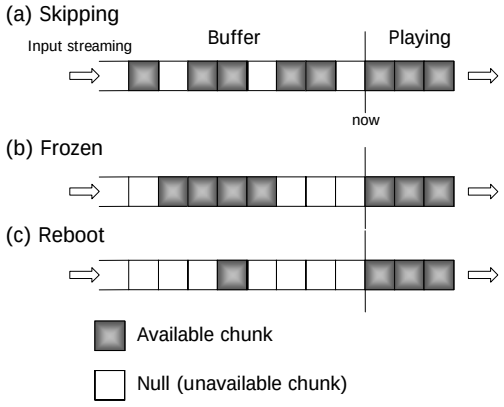

Figure 3. Buffer accidental events

Three accidental events are discussed via the buffer situation in some paper, i.e. *buffer occupancy* decides to enter which one mode as Figure 3 shown. In our research, we also simplify the subject by a discussion of buffer occupancy. Buffer is like a bridge between peer connections and application player. When QoS of network starts to degrade or some peer detects insufficient resource, client-program should adjust the data scheduling or execute peer adaptation early to deal possibly with the imminent predicament.

B. The Buffer Introduction of CoolStreaming

CoolStreaming is one successful, famous, and open P2P live streaming system. The history of CoolStreaming as well as its basic architecture and designing concepts are in the academic report [4]. On the other hand, one paper about CoolStreaming [5] specifies obviously how to design components of its system and procedures of its algorithm in detail. In addition, another paper about CoolStreaming [6] illustrates the mathematical and numerical analysis, the simulative and evaluative results, and the test in real world. Especially, buffer mechanism is mentioned and described in the article about theory of CoolStreaming as Figure 4 and Figure 5 shown [6]. Importantly, CoolStreaming still lives, so it keeps updating, correcting, and modifying. The latest design for P2P live streaming and the investigation report of users' true experience are introduced: Coolstreaming+ system modifies buffer management and data scheduling to improve the efficiency of hybrid pull-push and multiple sub-streams [7], and analyzes online statistics to find a way for NAT traversal [8].

As Figure 4 shown, a video stream is encoded to a division into many chunks. Every chunk has the sequence number to identify itself and buffer can sort the chunk in order according to their sequence numbers. A video stream is *decomposed* into many sub-streams to avoid a point failure. All sub-streams *combine* to the complete video stream. When some one delivery path (sub-stream)

is disconnected, peer can still receive other sub-streams to combine a part of video stream.

As Figure 5 shown, the structure of CoolStreaming buffer includes *synchronization buffer* and *cache buffer*. Synchronization buffer receives incoming chunks from every sub-stream to synchronize these chunks in the correct memory space and then push them to cache buffer. Cache buffer sort chunks with sequence numbers in order to prepare for playback.

The buffer of CoolStreaming is also fixed as like as PPLive, and there is not any mechanism for dealing with emergency. The designers considered that if peers or neighbors are selected well, consequently the stable and sufficient data source can be provided well. Focusing on the optimal overlay construction or peer adaptation is a current method popularly.

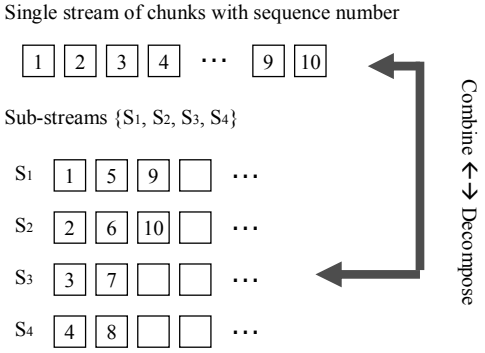

Figure 4. Example of stream decomposition

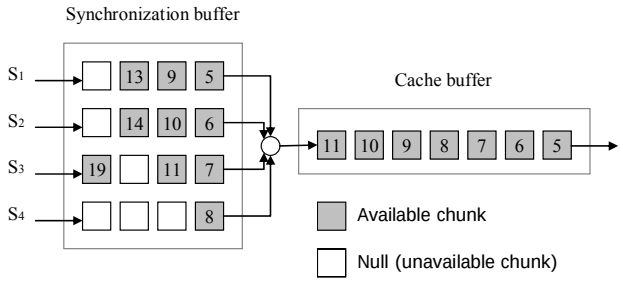

Figure 5. Structure of buffers and chunk combination process in a peer

III. Proposed Scheme

A. Algorithm for Proposed Scheme

As we mentioned above, the partner leaving can break a delivery path, and a lack of data resource temporarily leads to a reboot. Once the reboot happens, the user be compelled to execute peer selection again and waits for a long time to watch the frames again. The users must complain about the long waiting time. However, the reboot can be avoided via the method that peer monitors the variation of its data rate, and adjusts the buffer size based on the data rate. The decrease of data rate often indicates a warning of partner leaving. When the data rate of the peer decreases, the size of buffer is enlarged. And thus the buffer early reserves some un-continuous data. In this duration of overlay recovery, collecting sequential data is not the top priority in the schedule of our proposed algorithm. A lack of data source must result in bad QoS during recovery time. Although users watch the un-continuous frames, the matter avoids the reboot. In summary, we want to sacrifice playback quality of video to decrease the probability or frequency of reboot.

The decrease of data rate also indicates a sudden jitter or bottleneck of network bandwidth. This situation is different from the above peer leaving. Peers still keep and maintain the delivery paths or multicast streams to share data chunks between each others. In this situation, peers do not need to recover overlay, peers can choose to wait for a quality restoration passively or handle peer adaptation actively. The size of buffer is enlarged to pass the corner for unstable network bandwidth. After peer churn, peer adaptation, overlay recovery, or etc., the buffer shortens to the original size when coming back in the stable quality.

Figure 6 illustrates the final states of a peers' life cycle in the mesh scheme. The black lines denote the regular and success processes, and the grey dotted lines denote the accidental failures with error exceptions. A peer corresponds directly with the server when new joining. In new joining process, the server gives new peer an identification and executes the authentication. Then the second process is an initiation of preconditioned parameters, including the playback time synchronization, a set of candidate partners, the information of TV programs, other arguments of player, etc. Next, after the initiation, start-up process is responsible for selecting the partners and initializing the buffer to start gathering the video data.

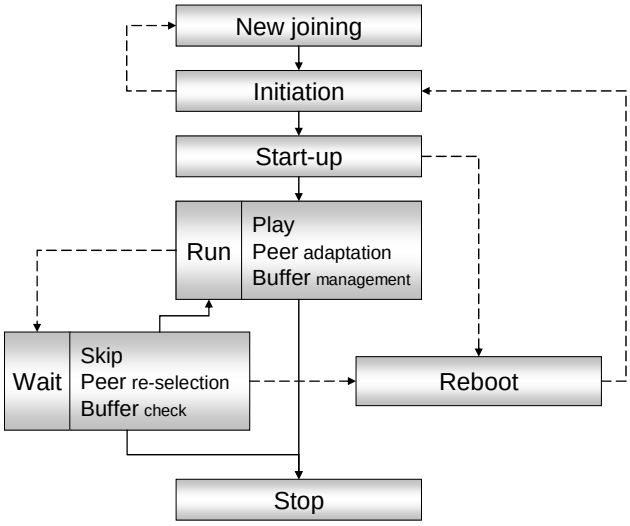

Figure 6. Final states of peers' life cycle

The importance of our algorithm put emphasis on three threads including video play, peer adaptation, and buffer management in the run process. The threads execute nonstop in the background. Buffer management of our algorithm monitors and computes received data rate to judge whether buffer must be enlarged or shortened. When received data rate decreases, buffer size is enlarged to gather and reserve non-continuous chunks beforehand

to reduce the probability of reboot. However, QoS degradation is inevitable consequently because the gathered chunks are non-continuous.

As Figure 6 illustrated, when a problem happens in playback, player is not able to show any image and then it enters into wait process. The player waits for available video data from buffer to play frames again. In wait process, the system can judge how to deal with the problem. In general, the controller of system can check buffer to decide that the video should be frozen, skipped, or rebooted as Figure 3 shown. The worst case is rebooting due to player system will return to the initiation state after rebooting; hence, the expected time of playing video again is very much long.

B. Client Program Flow

In summary, we list these processes and their responsibilities in Table I:

TABLE I
THE RESPONSIBILITIES AND FUNCTIONS OF CLIENT'S PROCESS IN OUR PROPOSED SCHEME

Process	Responsibility	Functions
New join	Identification	Bind with server Ensure identification and authentication
Initiation	Configuration	Setup network parameters Load and link media player Synchronize playback information Get candidate partners from server
Start-up	Preparation	Build local P2P overlay Initialize buffer Open decode stream
Run	Play	Get data from buffer to playback Execute decode stream Update information to server Interact with user
	Peer adaptation	Reselect partners Maintain local P2P overlay
	Buffer management	Monitor and compute incoming bit rate Enlarge or shorten size of buffer Schedule chunk selection
Wait	Interruption	Stop to playback
	Buffer check	Decide playback skip
	Peer reselection	Reselect partners
Reboot	Renewal	Close decode stream and clear buffer Close connections with peers Prepare for initiation and start-up again
Stop	Exit	Exit P2P network Close media player Disconnect all connections

1. New join: A peer connects to the server and logins for identification and authentication in the first process. And then the peer binds the server to keep alive.
2. Initiation: A peer must configure the software to work successfully in P2P network. This configuration includes two parts, one is about network, and another is about media application. Under the asymmetrical incoming/outgoing bandwidth, each peer sets up network parameters to adapt itself in P2P network. Peer gets the information of the playback time synchronization and a set of candidate partners from server. Media player is linked and loaded in this process.
3. Start-up: After the processes of new-join and initiation, peer starts up the preparation for smoothness before playback. It selects partners from candidates to build local P2P overlay. It also initializes the buffer and opens decode stream. These actions are done for starting to gather the video data.

   The duration since start-up process to run process is called as start-up delay. Users are always intolerable for long start-up delay. The good P2P overlay, good data scheduling, good network capacity, and good buffer mechanism can shorten the start-up delay to improve QoE.
4. Run: This process is the most important part of our proposed scheme. It includes video play, peer adaptation, and buffer management; buffer management is the most significant contribution among them. In traditional schemes, buffer management just gives assistance to data scheduling and sorts data to extract from buffer for playback. In our proposed scheme, besides working above jobs, buffer management monitors and computes incoming data bitrates. According to the incoming data bitrates, buffer size and data schedule can be adjusted into dynamic P2P environment.

   Peer adaptation always finds and evaluates suitable candidate partners to improve P2P efficiency of data delivery, i.e. to build a robust overlay for locality. In other words, peer adaptation can reselect partners to maintain local P2P overlay. Peer adaptation handles the assignments of lower network layer, and play handles the assignments of higher application layer to interact with users. Its important work is the media playback including decoding and screening, play must update and report relative information to server periodically.
5. Wait: When any error interrupts playback leading to that player shows incorrect images, the program flows into wait process. We focus the kind of P2P network error, such that wait triggers interruption, buffer check, and peer reselection sequentially.

   First, interruption stops playing to avoid decode error of play in run process (usually means the null error). Interruption notices the player waits for available video data from buffer to play again. Second, buffer is checked to decide that the video should be skipped, frozen, or rebooted. This decision influences peer reselection. Third, peer reselection is executed if it is necessary.

   Peer reselection is similar with peer adaptation, but reselection has two differences from adaptation. (1) Peer reselection works in play-off time, it builds new connections between other peers; peer adaptation works in play-on runtime, it maintains old connections

for local overlay. (2) Peer reselection can require server to fetch a new list of candidate partners for overlay recovery; peer adaptation knows new partners by itself via exchanging overlay information with each other.

6. Reboot: This process deals with renewal and reset to prepare to initialize and start up again. Therefore, reboot needs to close video stream and transmission stream, and clear buffer.
7. Stop: Users end up this application, and thus clients disconnect all transmissions and streams to exit P2P network.

We can discover this client program flow influence the performance of P2P system. For examples, start-up process produces start-up delay; run process produces playback delay; wait process produces recovery time. However, these performance metrics are subjective. An objective discussion is in next section.

C. Parameters of System Performance

In P2P applications for file sharing or VoD service, the utilization of download bandwidth is as high as possible. However, for live streaming, the incoming bit rate equals to the encoding rate of playback quality. Because the users (peers) cannot get the future data and do not need the past data in the live shows. If the incoming bit rate is stable and approximate to the playback quality, the users should enjoy the smooth playback. We define an *efficiency ratio* (R) of the incoming bit rate to the playback quality rate (q).

$$\text{Efficiency ratio } (R) = \frac{\text{Incoming bit rate (kbps)}}{\text{Playback quality rate (kbps)}} \quad (1)$$

If $R > 1$, the packet duplication or the impermanent remedy for packet loss happens.
If $R = 1$, the user can watch smoothly.
If $R < 1$, the available data is insufficient to play.

We also define R_t as the efficiency ratio of some time t, so the average throughput of an incoming stream is surely $\int^t R_t\, q\, dt$. A lower efficiency ratio usually indicates a more dynamic overlay with frequent peer churn (leaving). We also define a chunk available rate meant that the number of the collected availably chunks is divided by the number of the total accessible chunks. The chunk available rate is higher, and the source has the higher availability to delivery efficiently. The chunk available rate is always smaller than the efficiency ratio; however, the chunk available rate converges toward the efficiency ratio approximately in long time.

$$\text{Chunk available rate} = \frac{\text{\# of available chunks}}{\text{\# of total chunks}} \le 1.0 \quad (2)$$

A QoE criterion is evaluated by the smoothness of playback called playable rate, also called continuity [5]. The playable rate is defined as the playable time over the total playback duration. In another opinion, the playable rate equals to the number of video chunks that arrive before playback deadlines over the total number of video chunks. Of course, the playable rate is approximate to the chunk available rate.

$$\text{Playable rate} = \frac{\text{\# of chunks in buffer before deadline}}{\text{\# of total chunks}} = \frac{\text{Playable time}}{\text{Total playback duration}} \quad (3)$$

We can deduce that the average throughput of an incoming stream should equal to the total number of chunks, i.e. chunk available rate × t. In general, efficiency ratio is often smaller than 1.0, and efficiency ratio ≥ chunk available rate ≥ playable rate, because the challenges of peer competition, end-to-end delay, chunk unavailability, in-time deadline, overlay reconstruction, etc. exist in P2P network. While the playback duration is very large, R = efficiency ratio ≒ chunk available rate ≒ playable rate. In addition, if the playable rate decreases, the probability of reboot increases. If the reboot happens when a continuous nulls n in buffer, the probability of reboot is $(1 - R)^n$. We call n the *threshold of reboot*, represented how long the frozen time the streaming service can tolerate.

IV. Simulation

A. Simulation Environment

We implement a media simulator including server-program, swarm-program, and client-program in Java language. The server-program plays a data source server in a swarm to provide video data, and the swarm-program creates five threads to simulate five peers individually in a swarm. The peers sometimes cooperate to complement the media chunks, but sometimes compete for rare source. When providing data to clients, swarm-program simulates the peer competition to bring chunks into clients' buffer out-of-order and the peer churn to vary the incoming bit rate violently. Peer churn breaks the stable incoming stream and leads to a degradation of quality. The client-program plays video for users and gathers statistics of packet level to estimate QoE. We can observe all simulation results from the client-program with many dynamic network metrics.

Figure 7 provides an overview of our simulation system. Server-program can open a video and multicast to client through swarming. At the same time of multicasting, the experiments can handle the rate variation via the swarm-program. In addition, server and client both have a media player to observe three delays (i.e. the start-up delay, end-to-end delay, and buffer preparation latency between server and client) leading to the lag of IPTV. On the screen of the client-program, users can experience the quality via their real eyes, this describes the subjective QoE. On the other hand, under the background of the client-program, we can get statistics from packet level via the occupation of buffer.

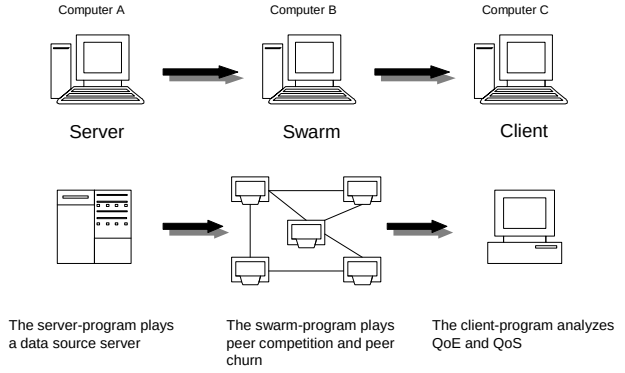

Figure 7. The simulated system

In the experiments, we can vary three arguments to test and compare: efficiency ratio, start-up time, and buffer size. For example, we assume that the playback rate is 300 kbps, and the efficiency ratio R is 0.4 when the incoming bit rate decreases from 300 kbps to 120 kbps. We denote buffer size as B_{size}, and B_{size} = 8 means that buffer can contain the content for 8 seconds. In order to understand easily, we assume that the size of a chunk is 300 kilobytes, and the video rate is 300 kbps; hence, a chunk can be played a second.

B. Simulation Results

Our estimation for QoE includes the probability of reboot and the playable rate to infer the possible interruption of streaming service. First, we define a probability function denoted as P (B_{size}, n, R) meaning the probability of reboot under the conditions when buffer size, threshold of reboot, *and* efficiency ratio are B_{size}, n, *and* R respectively. A penalty of reboot hurts QoE more heavily than a frozen process or a skipping process. As a result, a live streaming service should reduce the probability of reboot. Second, we use the playable rate to discuss the smoothness of playback. The high playable rate indicates that few interrupted time is in playback time and has a nice QoE.

1) The probability of reboot

The efficiency ratio of live streaming equals 1.0 usually; however, we decrease the efficiency ratio to simulate the condition of neighbors leaving which leads to the insufficient incoming source. First, when we assume B_{size} is 8 and n is 4, Figure 8 shows the correlation between probability of reboot and efficiency ratio. Our proposed dynamic buffer scheme can reduce the probability of reboot by 6 % than fixed buffer scheme, and more than half of users can watch a video when the efficiency ratio decreases to 0.3 in our proposed scheme. Compared with the fixed buffer, our proposed scheme has a low probability of reboot even if the efficiency ratio is very low. Beside, when the efficiency ratio is approximate to 0.0 and 1.0, the difference between the fixed buffer scheme and the dynamic buffer scheme is very small. When the efficiency ratio is approximate to 1.0, the incoming data is sufficient and the buffer does not need to adjust its size in our proposed scheme, so both schemes have the similar performance. When the efficiency ratio is approximate to 0.0, there is not any incoming data to be able to buffer, so both schemes have the similar performance, too.

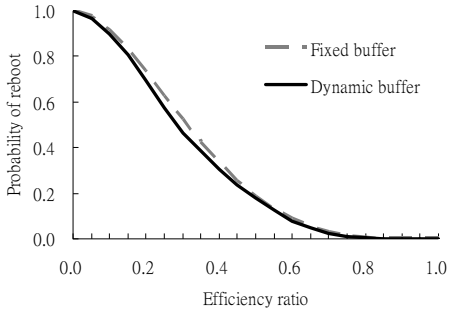

Figure 8. Probability of reboot P is correlated with efficiency ratio R when B_{size} is 8 and n is 4, i.e. P (8, 4, R)

Second, we assume B_{size} is 8 seconds and R is 0.3 as Figure 9 illustrated. As we can see, the probability of reboot will become smaller when the threshold of reboot n increase in both schemes. However, our proposed dynamic buffer scheme improves more when the threshold of reboot is larger, because the incoming bit rate is monitored in our proposed scheme, this leading to a low probability of long continuous nulls. Furthermore, the threshold of reboot n is larger, the tolerance for streaming skips is higher to hurt users' QoE. An assumption of n = 1 or 2 illustrates that the streaming reboots immediately when a skip happens. In fact, the threshold of reboot n is smaller than the start-up time; hence, n = 4 ~ 6 is the reasonable range in the experiment.

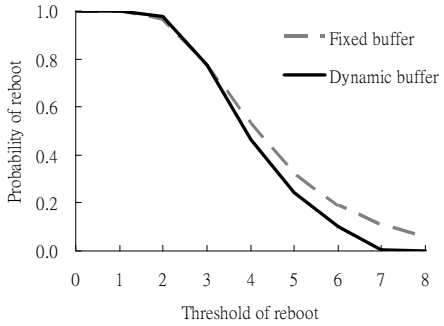

Figure 9. Probability of reboot P is correlated with threshold of reboot n when B_{size} is 8 and R is 0.3, i.e. P (8, n, 0.3)

Third, the efficiency ratio is given 0.3 in the experiment. As Figure 10 shown, the variation of buffer size does not affect the performance between the fixed buffer scheme and our proposed scheme. The larger buffer size can prepare more data, and thus the probability of reboot is lower surely. But the large buffer size leads to a high memory cost and a heavy overhead of data scheduling. Our proposed scheme is always 5 % better than the fixed buffer scheme.

From above charts, we demonstrate that the efficiency ratio affects directly the probability of reboot, and our

proposed scheme with unfixed buffer can be more suitable than the scheme with fixed buffer in the dynamic overlay. On the other hand, the threshold of reboot influences the buffer scheme less and may be a tradeoff for users' QoE. In addition, our proposed scheme for dynamic buffer can use a self-adjusted buffer control regardless of the buffer size.

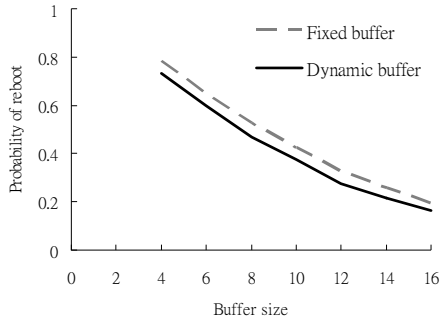

Figure 10. Probability of reboot P is correlated with size of buffer B_{size} when R is 0.3, i.e. P (B_{size}, $B_{size}/2$, 0.3)

2) The playable rate

In this section, we regularly and periodically vary rate to experiment as Figure 11 shown. The incoming bit rate just equals the playback rate when it reaches the peak. Therefore, the efficiency of buffer is obviously revealed when data rate declines. We use the playable rate to estimate the QoE. In addition, we can demonstrate briefly what to influence QoE deeply by varying the arguments for repeating experiments again and again.

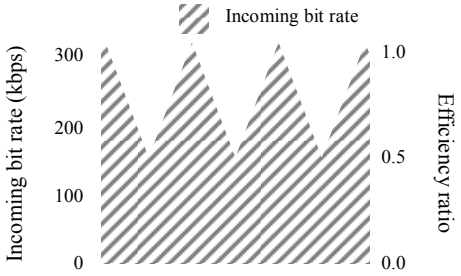

Figure 11. The variation of incoming bit rate in the simulated time

First, we vary the client's start-up time to observe the playable rate of streaming. On the best effort Internet, start-up buffer always has been a useful mechanism to deal with the rate variations of streaming sessions. Small buffer often means a shorter startup delay [9]. As Figure 12 illustrated, the playable rate is well improved when the start-up time increases. The reason for this is that the start-up time is a preparing time for buffering; hence, the longer time results in the fuller buffer preparation. However, the enhance of the playable rate increases lightly when the start-up time is more than a certain value (maybe five seconds in our experiment) because no more chunks is needed to preserve initially. Moreover, if the time of start preparation is long, the waiting time of the initial video increases and the playback delay of the streaming increases as compared with the traditional television. Live baseball game for example, other audiences already cheered for a home run, but the pitcher did not throw the ball on the screen of P2P streaming.

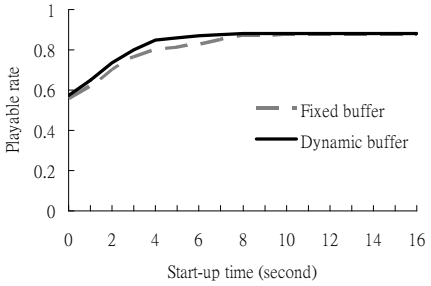

Figure 12. The playable rate via start-up time

Second, we vary the client's buffer size to observe the playable rate of streaming. The size of buffer and its rate of drainage and replenishment determine the smoothness of the playback [10]. As Figure 13 illustrated, the playable rate is well improved when the buffer size increases. The size of buffer can be larger to contain more content, but it has a limit to improve the playable rate because only in-time content rather than the future or the oldness one is needed to buffer. The small buffer is insufficient for the jitter from network or the peer churn from P2P overlay. However, the larger buffer leads to the more complex algorithm of chunk selection and the more overhead of memory.

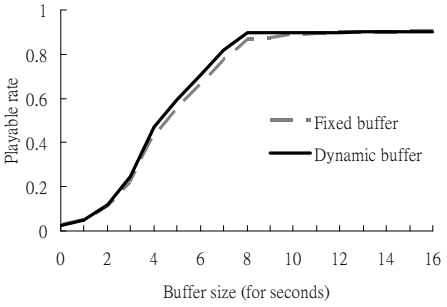

Figure 13. The playable rate via buffer size

Third, we vary the connection's efficiency ratio to observe the playable rate of streaming. As Figure 11 illustrated, the variation of incoming bit rate (efficiency ratio) is similar to a regular sierra. The bit rate just equals the playback rate when bit rate reaches the peak. Therefore, the efficiency of buffer is always smaller than 1.0 and is obviously revealed when data rate declines. For the entire simulated time t, the total throughput of incoming stream equals:

$$\int^{t} R_t \, q \, dt = \text{the grey area in Figure 11} = (0.5\, t + 0.5\, R_t\, t)\, q \quad (4)$$

And, the average of efficiency ratio = the total throughput of incoming stream divides by q and t:

$$\text{average of } R = \int^{t} R_t \, q \, dt / q\, t = (0.5\, t + 0.5\, R_t\, t)\, q / q\, t = 0.5 + 0.5\, R \quad (5)$$

Because of peer competition, end-to-end delay, chunk unavailability, and limitation of in-time deadline, the playable rate must be smaller than the average of efficiency ratio. In the other hand, the foot is R equaled to the minimum of R_t, so the playable rate should be larger than R. Therefore, the playable rate should be in the interval [R, 0.5 + 0.5 R]:

$$R \leq \text{playable rate} \leq 0.5 + 0.5\,R \qquad (6)$$

As Figure 14 illustrated, the performance of playable rate raises with the increase of efficiency ratio. When the efficiency ratio is more approximate to 1.0, the playable rate is closer to the up bound, because the wave of efficiency ratio is small to maintain a stable incoming bit rate well. As a result, a high and stable efficiency ratio can bring a good QoE. On the other hand, when the efficiency ratio is more approximate to 0.0, the playable rate is closer to the low bound, because the wave of efficiency ratio is large to experience an unstable incoming bit rate. As a result, a low and unstable efficiency ratio brings a bad QoE. We can imagine that the low efficiency ratio represents a dynamic P2P network with frequent peer churn; hence, a peer cannot get any chunk even if it has sufficient bandwidth, vice versa.

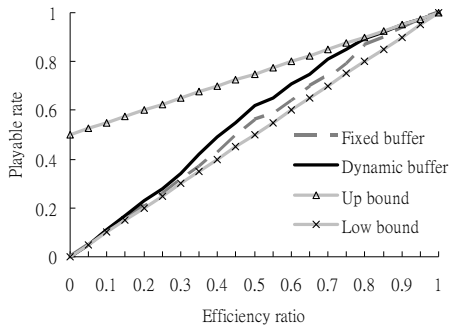

Figure 14. The playable rate via efficiency ratio

The performance of our proposed dynamic buffer scheme is always better than the performance of fixed buffer scheme. The gap between the performance of dynamic buffer scheme and the performance of fixed buffer scheme is obvious when the efficiency ratio is in [0.3, 08], because the buffer is timely enlarged or shortened depending on the incoming bit rate to get the in-time chunks. The improvement illustrates that the self-adjusted buffer control can be used to avoid the temporary starvation of source and to avoid the reboot. We demonstrate that rather our proposed scheme can be suitable for the P2P network with unstable bit rate, than the fixed buffer scheme. In addition, the gap is unobvious when the efficiency ratio is high, because the buffer shouldn't be adjusted in our proposed scheme. The gap is also unobvious when the efficiency ratio is low because the incoming bit rate is too low to support buffer.

In summary, we cannot find that buffer size or start-up preparation is more important than another. The goal of start-up waiting is to fill the buffer; hence, the start-up time is influenced not only by receiving rate, but also by buffer size. On the other hand, the buffer management is independent, this is one reason why we would like to modify buffer dynamically.

Buffer enlarged and start-up time lengthened can improve QoS. However, when encountering peer churn, the sudden shortage of video source leads to the quality degradation. As a result, using fixed buffer in dynamic P2P environment has a limit of improving QoS. The original design of fixed buffer may make video frames more continuous than the dynamic design, but when content bottleneck happens, the waiting time of reboot recovery is longer. On the other hand, video is played off and on in our proposed scheme, but long waiting time is less experienced.

Our proposed scheme of dynamic buffer is better than the fixed buffer when incoming bit rate decreases. When the efficiency rate is much high, the performance is good in both schemes, because there are any difference between the dynamic buffer and the fixed buffer when the incoming bit rate is stable. When the efficiency rate is much low, the performance is bad in both schemes, any buffer mechanism cannot work because of a lack of the incoming resource. However, our proposed scheme is superior to the fixed buffer when efficiency rate decreases. Because the self-adjusted buffer control can adjust the size of buffer according to dynamic incoming bit rate, our proposed scheme can improve the playable rate when the incoming bit rate drops down.

V. Conclusion

Our works discuss in the QoS of P2P live streaming and improve QoE. We proposed a concept about the self-adjusted buffer control to reduce the probability of rebooting. This buffering mechanism is suitable for mesh structure. Control mechanism monitors and computes the incoming data rate and checks the occupation of buffer to modify dynamically the size of buffer. This modification improves the efficiency of collecting data depending on network dynamic bitrates. Finally, we demonstrated that the method can reduce reboots to improve playback QoS via the simulation results. In addition, we point out some unavoidable drawbacks or limitations of innate P2P network.

The performance of IPTV is influenced by too many factors; hence a comprehensive comparison is difficult to discuss integration. We focus on the design of buffer forming to prevent from network jitter. We break up the traditional static buffer and design a dynamic buffer control to improve a part of performance. Through the evaluation of simulation results, we can analyze that our proposed scheme is better than traditional scheme and improves QoE a little.

This buffering mechanism only need to modify a little on client terminals and it is unnecessary to modify the total P2P system. It is suitable for chunk-based buffer

design, and it can work on all existing P2P live streaming system nowadays.

In the future, we will put the self-adjusted buffer control in the current P2P live streaming system to experiment. In advanced studies, we will integrate buffer control with peer adaptation, churn recovery and chunk scheduling together to design an integrated implementation for P2P live streaming system.

APPENDIX A DEMO

REFERENCES

[1] Yarali, A. and A. Cherry, *Internet Protocol Television (IPTV).* IEEE TENCON, 2005.

[2] Locher, T., et al., *Push-to-Pull Peer-to-Peer Live Streaming.* Lecture Notes in Computer Science, 2007.

[3] Hei, X., Y. Liu, and K.W. Ross, *Inferring Network-Wide Quality in P2P Live Streaming Systems.* Selected Areas in Communications, 2007.

[4] Zhang, X., J. Liu, and B. Li, *On large-scale peer-to-peer live video distribution: Coolstreaming and its preliminary experimental Results.* IEEE International Workshop on Multimedia Signal Processing, 2005.

[5] Zhang, X., et al., *CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming.* IEEE International Conference on Computer Communications, 2005.

[6] Xie, S., et al., *Coolstreaming: Design, Theory, and Practice.* IEEE Transactions on Multimedia, 2007.

[7] Li, B., et al., *Inside the new coolstreaming: Principles, measurements and performance implications.* IEEE Journal on Selected Areas in Communications, 2007.

[8] Li, B., et al., *An empirical study of the Coolstreaming+ system.* IEEE Journal on Selected Areas in Communications, 2007.

[9] Liao, X., et al., *AnySee: Peer-to-peer live streaming.* IEEE International Conference on Computer Communications, 2006: p. 2411-2420, 3337.

[10] Tang, Y., et al., *Deploying P2P networks for large-scale live video-streaming service.* IEEE Communications Magazine, 2007. **45**(6): p. 100-106.

[11] Hei, X., Y. Liu, and K. Ross, *IPTV over P2P streaming networks: The mesh-pull approach.* IEEE Communications Magazine, 2008. **46**(2): p. 86-92.

[12] Agboma, F., M. Smy, and A. Liotta, *QOE ANALYSIS OF A PEER-TO-PEER TELEVISION SYSTEM.* IADIS International Conference on Information Systems, 2008.

Jun-Li Kuo (estar.cs95g@nctu.edu.tw) received his M.S. degree in Computer Science and Information Engineering from National Central University, Taiwan in 2006. He is a Ph.D. student in Computer Science at National Chiao Tung University, Taiwan. His research interests include peer-to-peer network and WiMax multimedia, cloud computing.

Chen-Hua Shih (shihch@csie.nctu.edu.tw) is currently a Ph.D. candidate in computer science at National Chiao Tung University, Hsinchu City, Taiwan. He received his B.S. degree in computer science and information engineering at National Central University, Jhongli City, Taiwan. His research interests include QoS, mobile IP, wireless networks, and network protocols.

Yaw-Chung Chen (ycchen@cs.nctu.edu.tw) received his Ph.D. degree in computer science from Northwestern University, Evanston, Illionis, USA in 1987. During 1987-1990, he worked at AT&T Bell Laboratories. Now he is a professor in the Department of Computer Science of National Chiao Tung University, and the Director of NCTU/III Joint Research Center. His research interests include multimedia communications, high speed networking, and wireless networks.

