Hierarchical Architecture for Peer-to-Peer Video on Demand Systems with the Notion of Dynamic Swarms*

Yasuaki YUJI†, Nonmember and Satoshi FUJITA†a, Member

SUMMARY This paper proposes a method to reduce the playback suspension in a Video-on-Demand system based on the Peer-to-Peer technology (P2P VoD). Our main contribution is twofold. The first is the proposal of a hierarchical P2P architecture with the notion of dynamic swarms. Swarm is a group of peers to have similar playback position and those swarms are connected with an overlay so that requested pieces are forwarded from a swarm to another swarm in a bucket brigade manner, where the forward of pieces is regulated by the super-peer (SP) of each swarm. The second contribution is the proposal of a match making scheme between requests and uploaders. The simulation result indicates that the proposed scheme reduces the total waiting time of a randomized scheme by 24% and the load of the media server by 76%.

key words: Peer-to-Peer, video-on-demand, playback suspension, match-making

1. Introduction

With a widespread of broadband accesses to the Internet, video streaming services have attracted many network users in recent years. In fact, YouTube attracted more than one trillion views in 2011 and it is forecasted that the video streaming will occupy 55% of the global consumer traffic by 2016 [5]. In general video streaming services, the playback of a video stream concurrently proceeds with an acquisition of the stream from media servers. This indicates that the heavy load of media servers will easily degrade the quality of streaming services in terms of the delay, jitter and the temporary suspension of the playback. With such a background, video-on-demand using peer-to-peer technology (P2P VoD, for short) has become a promising approach to reduce the load of the servers by distributing it to all participants [3], [6], [9], [13], [14], [16].

In P2P VoDs, each video file is divided into small fragments called pieces and those pieces are disseminated to the client peers over a logical network called P2P overlay. Each peer acquires pieces by repeating local communication among nearby peers so that pieces close to the playback position are given a higher priority than others. When a piece cannot be acquired by the time of playback, the playback of the stream is suspended until the piece is acquired. Such a suspension gives considerable stress to the users and will significantly degrade the satisfaction of users concerned with the overall streaming service.

Such a playback suspension could be avoided by regulating the behavior of each uploader so that it selects peers requesting pieces close to the deadline as the downloaders with high priority. In addition, it would also be crucial for each downloader to become adjacent with peers to have enough pieces to be acquired. In the literature, there are several proposals to realize such a deadline-driven selection of uploaders [1], [18], [19], but if each uploader independently conducts such a selection, we could not avoid the duplicated uploads of the same pieces to a specific peer, which causes the waste of the upload bandwidth of the overall network and the degradation of the upload performance. Of course, we could remove such redundant uploads by allowing each downloader to reject the upload of duplicated pieces, but such a reactive approach causes an overhead which increases the delay of the streaming, i.e., it is meaningful to seek a proactive approach so that uploaders conduct the upload of pieces in a collaborative manner.

In order to realize such a collaborative upload, in this paper, we propose a hierarchical P2P architecture based on the notion of dynamic swarms. In the proposed architecture, swarm is a group of peers to have similar playback position, which is designed to effectively share and acquire pieces close to the playback position. With this notion, peers in the P2P system can organize a logical structure such that requested pieces are forwarded from a swarm to another swarm in a bucket brigade manner. The forward of pieces is regulated by the super-peer (SP) of the swarm, so that the performance degradation due to the independent selection of downloaders could be avoided. More specifically, each SP conducts a match making between the set of requests and the set of uploaders in such a way that the set of requests associated with an uploader fully utilizes the upload bandwidth of the uploader. The reader should note that the suspension of a video stream, which is the primary objective function to be minimized in the proposed scheme, occurs due to several reasons in P2P VoDs, which includes the low utilization of the upload bandwidth, the concentration of requests received from many downloaders, and the incorrect ordering of the uploaded pieces. Among such issues, we will focus on the increase of the utilization of the upload bandwidth and the balance of the load of uploaders as a concrete way to reduce the suspension time in the overall streaming service.
The performance of the proposed method is evaluated by simulation. Since the concentration of requests can not be resolved by any of previous schemes except for a randomized scheme, we used a combination of a random selection of downloaders with the aid of SPs as the competitor of the proposed scheme. The simulation result indicates that the proposed scheme reduces the waiting time of the randomized scheme by 24% and the load of the media server by 76%.

The remainder of this paper is organized as follows. Section 2 describes the model of P2P VoD. Section 3 reviews related work. Sections 4 and 5 describe the proposed swarm architecture and the scheduling algorithm, respectively. Section 6 describes the result of simulation. Finally, Sect. 7 concludes the paper with future work.

2. Model

2.1 P2P Network

Let \( \mathcal{P} \) be a P2P network consisting of a media server, a tracker and \( N \) peers. Peers in \( \mathcal{P} \) are directly connected with the media server and are mutually connected with a logical network called P2P overlay. Peers and the media server can communicate with each other by sending messages over links connecting them. To simplify the discussion, we assume that the capacity of links and the download capacity of links can communicate with each other by sending messages over links connecting them. Each peer can leave the system at any point in time. In the following, we assume that peers can leave the system without conducting normal procedure except for specific peers such as super peers.

In order to realize a continuous playback without suspension, each peer must acquire every missing piece before its playback time. As a concrete method to be aware of the set of pieces acquired by its neighbors, we assume the existence of buffer map (BM) which is commonly used in many P2P VoDs. BM for file \( F \) is a bit array of length \( n \), and the \( j \)-th element in the array represents whether the \( j \)-th piece is acquired (value 1) or not (value 0). BM is periodically exchanged between neighboring peers so that each peer can learn the set of pieces acquired by each neighbor in almost real-time.

2.3 Behavior of Each Peer

Suppose that peer \( p \) newly joins the system. The behavior of \( p \) before leaving the system is as follows.

**Preparation:** At first, peer \( p \) asks the tracker to send back a (random) list of peers in the overlay. After receiving the list, \( p \) contacts each peer \( q \) in the list to become a neighbor of \( q \). It then generates its own BM and exchanges the latest copy of BM with its neighbors. The BM of \( p \) is updated when it acquires a new piece, and the exchange of a copy of BM is periodically conducted until it leaves the system.

**Playback:** Peer \( p \) acquires the first few pieces of file \( F \) from the media server. After that, it starts the playback from piece 1. In the succeeding steps, \( p \) executes the acquisition and the playback in a concurrent manner. Pieces to be acquired next are determined by the piece selection rule. The request for a selected piece \( j \) is sent to an adjacent peer holding the piece, by referring to the latest copy of BMs received from neighbors. A request received from a neighbor can be forwarded to another neighbor if necessary. Suppose that \( p \) holds several pieces requested by other peers. If the number of requests exceeds \( u_p \), then \( p \) selects \( u_p \) requests using an appropriate peer selection rule.

**Departure:** Each peer can leave the system at any point in time. In the following, we assume that peers can leave the system without conducting normal procedure except for specific peers such as super peers.

3. Related Work

BiToS is a P2P VoD which adopts a piece selection rule described as follows [15]. Each peer divides the set of unacquired pieces into high priority set and low priority set by the closeness to the PP, and conducts a piece selection in the following two steps: 1) select one set with a certain probability, and 2) select a piece from the selected set according to the rarest first rule [2]. Although the piece selection rule adopted in BiToS is widely used in the literature [21], it is pointed out that in order to attain a satisfactory performance, the selection of pieces from the high priority set should be conducted so that both of the rareness and the closeness to the PP (within the high priority set) are taken into account [12].

Yang et al. proposed a way of organizing an overlay
so that each peer \( p \) is adjacent with peers to have as many un-acquired pieces of \( p \) as possible [18]. The key idea of the scheme is that: 1) for each neighbor \( q \), \( p \) maintains the number of pieces which are held by \( q \) but are not held by \( p \), and 2) when this number becomes less than a threshold, \( p \) asks the tracker to recommend a new neighbor \( q^* \) to have PP similar to \( p \). An apparent drawback of this approach is that the tracker should keep track of the PP for all peers participating in the system, which will be relaxed in the proposed scheme by delegating the role of management to super peers.

In [19], BitTorrent is extended so that each request carries the deadline and upon receiving such requests from downloaders, each peer determines the target of upload in the descending order of the criticalness of the deadline. A similar idea has been proposed in [1] so that the set of neighbors is dynamically divided into high priority group and low priority group, and in selecting the target of upload, it first selects a group with a certain probability and then selects requests in the descending order of the criticalness of the deadline. Appropriate selection of uploaders is also considered in the literature [12], [19]. In the method proposed in [19], each peer keeps the number of un-responded requests for each neighbor, and issues a request to a neighbor with the least un-responded requests. Such a selective issue of requests considering the load of uploaders could effectively avoid the concentration of requests to a specific uploader compared with other schemes in which downloaders issue a request to all of the relevant peers.

Finally, we should remind that the idea of adopting a hierarchical architecture to improve the performance of P2P video streaming is not new [4], [7], [10], [11]. However, most of previous works focus on the load balancing of the media server in organizing a hierarchical overlay, which is different from our approach in which the role of match making is delegated to super peers.

4. Swarm Architecture

4.1 Overview

As was described previously, in order to realize a continuous playback of video streams, each peer \( p \) must collect pieces close to its playback position (PP) as early as possible, where the PP of \( p \) continuously goes ahead as the playback of the stream proceeds. An efficient way to realize such a continuous piece acquisition is to contact a peer \( q \) whose PP is slightly ahead of \( p \) and to ask \( q \) to directly upload pieces to \( p \). In addition, if there are several candidates for the uploader, the upload from those candidates should be appropriately regulated so as not to cause a duplicated upload nor the missing of urgent pieces.

To solve such crucial issues, we propose a hierarchical P2P in which peers to have similar PP dynamically organize a group of peers called swarm. In this architecture, each peer belongs to exactly one swarm generated by the tracker\(^1\) and each swarm contains exactly one super peer (SP) selected from the members of the swarm. Those SPs are connected with an overlay, and any two peers participating in the system can communicate with each other via SPs corresponding to them. Each SP manages the information on all peers in the corresponding swarm such as the residual upload capacity and the latest BM to regulate uploads conducted by the managed peers. Each request (for a piece) issued by a peer is sent to the SP of the same swarm, and it will be forwarded to other SPs and the media server if necessary.

The reader should note that such an overlay of SPs can effectively support VCR operations such as the fast forward and the jump to a specific chapter [14]. In fact, since each swarm consists of peers to have similar PPs, each peer \( p \) can quickly identify an appropriate uploader after (or during) conducting a VCR operation, if \( p \) could identify an SP whose PP is close to the PP of \( p \) through the overlay of SPs.

4.2 Generation of Swarms

Swarms generated by the tracker are given a sequence number starting from 0. When a new swarm \( X \) is generated by the tracker, the first peer in \( X \) is selected as the (first) SP of \( X \). More concretely, the tracker keeps two global variables \( x \) and \( y \), where \( x \) indicates the sequence number of the latest swarm and \( y \) indicates the number of ordinary (i.e., non-super) peers in the latest swarm, and updates them as follows:

- When it receives a request to join the system from a new peer \( p \), variable \( y \) is incremented by one in modulo \( x \); where \( x \) is a parameter indicating the intended swarm size;
- If \( y = 0 \) after the increment, it generates the \( x \)th swarm consisting of peer \( p \), makes \( p \) to be the first SP of the swarm, and increments \( x \) by one;
- Otherwise, it makes \( p \) to be a member of the current swarm with sequence number \( x \), and notifies the fact to the SP and the tracker.

In the proposed architecture, each peer including SP is allowed to leave the swarm at any point in time. However, if an SP \( p \) wishes to leave, it should select a successor \( q^* \) from the set of peers in the swarm managed by \( p \), and notify the fact to the tracker and all members of the swarm before leaving. The reader should note that the overhead due to such a maintenance of the overlay is a common disadvantage of hierarchical P2P systems as in the proposed architecture, whose impact becomes large as the churn rate becomes high. Let \( \sigma_q \) denote the piece ID associated with peer \( q \) such that: 1) all pieces from \( \eta_q \) to \( \sigma_q \) are acquired by \( q \) and 2) the next piece of \( \sigma_q \) is not acquired by \( q \) or \( \sigma_q \) is the last piece of the given video file. In the following, we call \( \sigma_q \) the acquisition position of \( q \), which indicates the status

---

\(^1\)The reader should note that in the proposed architecture, the tracker plays the role of managing swarms in addition to the control of the overlay as in BitTorrent-like systems. A similar extension of the role of the tracker has been conducted in the scheme proposed by Yang et al. [18].
of piece acquisition by \( q \) ahead of \( \eta_p \). By using the notion of the acquisition position, the procedure for selecting the successor of \( p \) is described as follows.

**procedure** Select_Successor

- Let \( q' \) be a peer in the swarm managed by \( p \) such that \( "\prec q \) is the most ahead among all peers in the swarm."

- Then, \( p \) selects \( q' \) as the successor of \( p \).

A reason of why we use \( \sigma_q \) instead of \( \eta_q \) as the indicator is that we wish to take into account the status of piece acquisition rather than the status of playback. In other words, we expect that the SP of a swarm acquires more pieces than any other peers in the swarm.

**4.3 Overlay of SPs**

The overlay of SPs is constructed in such a way that for each swarm \( X \), the SP of \( X \) is directly connected with the SP of another swarm \( Y \) satisfying relation \( X \prec Y \), where a formal definition of relation \( \prec \) will be given later. The role of the overlay is to realize a continuous upload of pieces from swarm \( Y \) to swarm \( X \) under the regulation by the SP of \( Y \), as well as a timely notification of necessary information such as the list of peers and the set of latest BMs from \( Y \) to \( X \). See Fig. 1 for illustration. If \( X \prec Y \), we say that \( Y \) is the precursor of \( X \) and \( X \) is a follower of \( Y \), where each swarm can have several followers whereas the number of precursors is restricted to be at most one. In the following, we use symbol \( X \) to denote the set of peers in swarm \( X \). In addition, we denote the set of peers in the precursor of \( X \) as \( \text{pre}(X) \) and the set of peers in the followers of \( X \) as \( \text{fol}(X) \).

When swarm \( X \) is generated by the tracker, the precursor of \( X \) is determined by the tracker according to the following procedure:

- if there is a (non-empty) swarm generated earlier than \( X \) by more than \( \tau \) time units, then among such swarms, it selects a swarm \( Y \) with the smallest number of followers as the precursor, where \( \tau \) is an appropriate parameter,

- if there is no such swarm but there is a swarm generated earlier than \( X \), then it selects the oldest swarm as the precursor of \( X \), and

- otherwise, \( X \) has no precursor.

In this method, the tracker determines the precursor of \( X \) by merely referring to the generation time and the number of followers of existing swarms without communicating with SPs\(^1\). A critical point is that a swarm \( Y \) which is generated “just before” \( X \) may not have enough pieces for \( X \). For example, consider the case in which many peers arrive at the system almost at the same time. In such a case, swarms \( X \) and \( Y \) have a similar status on piece acquisition, i.e., it is highly probable that \( Y \) does not have enough pieces for \( X \). This suggests that we should select a swarm which stays in the system for a sufficiently long time, as the precursor of swarm \( X \).

**4.4 Support of VCR Operations**

This subsection describes how to support VCR operations in the proposed architecture. Suppose that peer \( p \) in swarm \( X \) updates its PP to \( \eta_p \) by conducting a VCR operation. Then, \( p \) moves itself to a swarm which has enough pieces ahead of \( \eta_p \), in the following manner.

1. Starting from swarm \( X \), \( p \) sequentially traverses precursor of precursors to identify: 1) the first SP \( q \) such that the acquisition position \( \sigma_q \) of \( q \) is ahead of \( \eta_p \) and 2) the last SP \( r \) such that \( \sigma_r \) is behind of \( \eta_p \). Note that \( q \) may not exist in general.
2. Peer \( p \) moves to the swarm managed by \( q \) if it contains a peer holding the piece with ID \( \eta_p \) and moves to the swarm managed by \( r \) otherwise.

If we merely consider the benefit of the moving peer \( p \), it would be better to select the oldest swarm as the target of the move. However, as the “age” of the target swarm increases, it becomes harder for \( p \) to contribute to the other peers as an uploader after the move. Thus, in the proposed method, we select a swarm to have an acquisition position sufficiently close to \( \eta_p \), as the target of move.

The reader should note that the cost of the above procedure in terms of the time and the communication cost is proportional to the number of swarms in the system, since it sequentially checks all swarms starting from \( X \), in the worst case. Such a flaw can be overcome by adopting a more sophisticated data structure such as DSL (Dynamic Skip List)\(^1\) and VMesh\(^2\), but we leave the detailed description of the improved scheme to another paper because it is out of scope of the current paper.

---

\(^1\)In other words, it assumes that the time after the generation precisely reflects the status of piece acquisition. However, such an assumption does not hold in general particularly when the playback is frequently suspended. How to compensate this issue is left as a future work.
5. Assignment of Uploaders

This section describes a way of assigning uploaders to the requests received from downloaders. As was described previously, each peer in swarm $X$ can acquire pieces from peers in $X \cup \text{pre}(X)$. In the proposed scheme, it first partitions received requests into two classes, i.e., priority ones and normal ones, and conducts the assignment of uploaders for each class so as not to cause the waste of resources nor the missing of urgent pieces. The reader should note that although the proposed algorithm described below does not take into account the heterogeneity of peers, we can easily extend the algorithm to heterogeneous environments by considering the ability of upload in determining the matching.

5.1 Preliminaries

In the proposed method, each peer acquires pieces from other peers in the ascending order of the piece ID. However, since the acquisition of a piece generally takes a longer time than the playback in P2P VoDs, we design the scheme so that peer $i$ directly acquires a piece with ID $i$ from the media server if $i - \eta_p \leq \alpha$, i.e., $p$ reserves at least $\alpha$ (≥ 1) pieces ahead of $\eta_p$ in its local buffer\(^1\). Requests for the other pieces with ID $i$ are classified into the following two types:

- it is a priority request if $i - \eta_p \leq \beta$ for a predetermined threshold $\beta$ ($> \alpha$), and
- it is a normal request, otherwise.

In the following, we assume that each request is identified by a pair of peer ID and piece ID. Let $R$ be the set of requests received by the SP $q$ of swarm $X$. The reader should note that $R$ consists of requests issued by peers in $X \cup \text{fol}(X)$. As was described above, $q$ is aware of the upload bandwidth and the latest BM of the peers in swarm $X$. The objective of the assignment algorithm is to determine $R$ and $X$, to calculate a matching between $R$ and $X$. If such a matching is obtained, each peer in $X$ can autonomously decide the target of uploads and pieces to be uploaded. More concretely, we consider the problem of maximizing the number of requests assigned to the uploader, subject to the following constraints:

- The number of requests assigned to peer $p$ does not exceed $u_p$ (constraint on the upload capacity).
- Each request is assigned to at most one peer (avoidance of duplicated upload).
- If a peer is assigned to a normal request and it has a piece requested by a priority request, then the priority request must be assigned to some peer by the assignment (preference of priority requests).

This problem is equivalent to the problem of finding a maximum matching in a bipartite graph with vertex set $R \cup X$ if we neglect the priority of requests. It is known that the maximum matching problem can be solved in $O(n^{1/2}m)$ time if the given bipartite graph has $n$ vertices and $m$ edges\([8]\). However, in P2P VoDs considered in this paper, each SP (the scheduler) must complete an assignment of requests to the set of uploaders within a very short time (e.g., less than 100ms), which means that the running time of the above optimum algorithm is not satisfactory for our purpose. Hence in the proposed scheme, we will take an approach to calculate a quasi-optimal solution using heuristic method.

5.2 Matching Algorithm

Let $R$ be a set of requests and $P$ be a set of uploading peers. Let $G$ be a bipartite graph with vertex set $R \cup P$ and edge set $E$, where vertices $r \in R$ and $p \in P$ are connected by edge $e \in E$, if and only if the piece requested in $r$ is held by $p$. A (maximal) matching between $R$ and $P$ is calculated by the following procedure:

1. Let $p^*$ be a vertex in $P$ with the maximum degree. Let $R^*$ be the set of neighbors of $p^*$ in $G$, and $r^*$ be a request in $R$ with the minimum degree.
2. Add edge $(r^*, p^*)$ to the solution, i.e., determine the assignment of request $r^*$ to peer $p^*$, and remove $r^*$ from $G$ with all incident edges.
3. If the number of edges incident on $p$ in the solution reaches the upload bandwidth $u_p$ of $p$, then remove $p$ from $G$ with all incident edges.
4. If $E = \emptyset$, then terminate the algorithm. Otherwise, go to Step 1.

In the proposed assignment scheme, the above procedure is applied to the set of priority requests and the set of normal requests in this order. More concretely, we first run the procedure by letting $R$ be the set of priority requests and $P := X \cup \text{pre}(X)$, and then run the procedure by letting $R$ be the set of normal requests and $P := X$. Thus the upload port of peers in $X$ are available for normal requests after completing the first assignment, and in the first assignment, the SP of $X$ assigns peers in $\text{pre}(X)$ (i.e., peers not in $X$) to the priority requests issued by the peers in $X$ (a way of resolving such “conflicts” of assignments will be discussed later). The running time of the assignment scheme is evaluated as follows. At first, since the size of $R$ decreases by at least one in each iteration and a minimum (or maximum) element in a set can be found in logarithmic time if we implement it using a heap, the running time of the matching algorithm is bounded as $O(|R| \log |P| + \log |R|)$. In our swarm architecture, the number of peers in each swarm is bounded by $S_{\text{max}}$ on average, and the number of requests issued by the peers in a swarm can be bounded by a constant. Thus, we can conclude that given a set of requests and a set of peers, each SP can generate a quasi-optimal solution in a short computation time.

\(^{1}\)If the number of direct requests is sufficiently small, the media server can respond to all of such requests since the upload bandwidth of the media server is generally much wider than that of normal peers.
5.3 Conflict Resolution

In the proposed scheme, the assignment of requests to uploaders is conducted by all SPs in parallel. Hence each peer in swarm \( X \) might be regarded as an uploader by several SPs, i.e., the SP of swarm \( X \) and its followers. Such a conflict of assignments observed in swarm \( X \) is resolved by the following rule:

1. Priority requests assigned by the SP of \( X \) is given the highest priority.
2. The other priority requests assigned by the followers is given the second highest priority.
3. Normal requests assigned by the SP of \( X \) is given the lowest priority.

In other words, each SP tries to use resources in its own swarm \( X \) as much as possible, and resources in \( \text{pre}(X) \) can be used by the peers in \( X \) only when there remains a room in \( \text{pre}(X) \) after the first assignment. The reader should note that if there remain unassigned (urgent) requests, they are forwarded to the media server to meet the deadline, as long as the server has enough residual upload capacity.

6. Simulation

6.1 Preliminaries

We evaluate the performance of the proposed scheme by simulation. In the simulation, we assume that 200 peers consecutively join the system according to the Poisson distribution with mean 12 sec, start playback after collecting the first 3 pieces and leave the system after completing the playback of the last piece. No VCR operation is conducted by the peers. The media server is the last resort for peers to acquire pieces by the deadline, although in each round, it can respond to at most \( u_q \) requests from received requests which are selected in the descending order of the piece ID.

Since any of previous schemes [1], [12], [15], [18], [19] concerned with match making do not use the overlay of SPs, we cannot directly compare them with the proposed scheme. Recall that the key idea of the proposed scheme is that: 1) it explicitly uses the data structure reflecting the difference of the status of piece acquisition, and 2) it explicitly conducts match making between requests and uploaders by using a hierarchical approach. The effect of such techniques is obvious. Thus in the following, we evaluate the quality of the match making algorithm by comparing it with a simple but practical scheme based on randomization\(^1\). In the randomized scheme, peer \( p \) in swarm \( X \) requests \( \beta' \) pieces ahead of \( \eta_p \) to peers in swarm \( X \) and if the requested piece is within distance \( \beta (< \beta') \) from \( \eta_p \), it is also sent to swarm \( \text{pre}(X) \) through corresponding SPs. Peer \( q \) receiving requests selects at most \( u_q \) requests independently and randomly, and responds to them.

Those two schemes are evaluated with respect to the following metrics.

- The waiting time of each peer during the suspension of the playback. We denote the waiting time of peer \( p \) as \( W(p) \), which includes the startup time \( W_s(p) \) before starting the playback.
- The amount of data uploaded by the media server per second. Note that it does not exceed the upload capacity of the media server.

The other parameters are fixed as follows. We consider a video file of length 1600 sec with playback bit rate of 512 Kbps, which is divided into 534 pieces of size 192 KB each, so that the playback of a piece takes 3 sec. The upload capacity of the server is 20 Mbps and that of each peer is 3 Mbps; i.e., the server (resp. a peer) can upload at most 13 (resp. 2) pieces during the playback of a piece. Two thresholds used in the proposed scheme are fixed as \( \alpha = 1 \) and \( \beta = 3 \), and the threshold used in the randomized scheme is fixed as \( \beta' = 4 \). That is, one piece ahead of the PP is acquired from the server, two pieces are acquired using priority request, and additional one piece is acquired by the randomized scheme. The size of each swarm is fixed to \( S_{\text{max}} = 20 \) and parameter \( \tau \) is fixed to 10 sec.

6.2 Waiting Time

The average waiting time of the proposed and the randomized schemes are 3.05 sec and 3.95 sec, respectively, i.e., it attains a reduction of 24%. Figure 2 illustrates the waiting time and the startup time for each peer. In the proposed scheme, the waiting time equals to the startup time except for the first three peers, i.e., they do not encounter the playback suspension after starting the playback. This is best possible for our setting, since it needs at least 3 sec to acquire a piece, where the acquisition from the server might take longer time if the server becomes busy. In other words, the above result indicates that the server has enough residual capacity so that new peers can certainly acquire pieces from the server.

On the other hand, in the randomized scheme, the frequent playback suspension occurs in early steps of the simulation. Those peers are members of the first swarm. They should acquire pieces from the media server because they have no enough precursors, but since the server should continuously upload the first few pieces to newly joined peers, it increases the waiting time as the number of participants increases. In addition to peers in the first swarm, several peers encounter a long startup time of more than 3 sec, which indicates that the load of the server keeps high during the simulation.

\(^1\)The reader should note that in the simulation, we do not use a deterministic scheme in which requests to be responded are selected by referring to the priority, since it easily causes a duplicated upload and the missing of low priority pieces.
6.3 Server Stress

Playback suspension occurs if a piece is not acquired from the media server by the deadline. Since newly arrived peers rely on the download from the media server, if there remains enough capacity at the media server, we could simultaneously reduce the frequency of playback suspension and the startup time of each peer. In the proposed scheme, the upload rate is bounded by 15 Mbps and a peak occurs only when the server simultaneously uploads to a peer with the most ahead playback position and newly joined peers. On the other hand, in the randomized scheme, many peers continuously rely on the server and after the 130th round, the upload rate frequently reaches the upload capacity 20 Mbps. The number of pieces $U_p$ actually uploaded from the media server is 2577 in the proposed scheme and 8618 in the randomized scheme, which indicates that the proposed scheme reduces the load of the server by 76%.

6.4 Impact of Upload Bandwidth of Server

Finally, we evaluate the impact of the upload capacity of the media server to the performance of the schemes by varying it from 26 to 18 Mbps. The average waiting time of the proposed scheme is around 3.05 sec regardless of the value of $u_s$. However, in the randomized scheme, while it attains 3.05 sec when $u_s$ is 26 Mbps, it significantly degrades as $u_s$ decreases. In fact, although the difference to the proposed scheme is kept small from 26 to 20 Mbps, it rapidly increases to more than 45 sec for 18 Mbps, which is because the “chain of playback suspension” violates the continuous flow of pieces from precursors to the followers. Similar phenomena could be observed for the number of uploaded pieces. The above results indicate that the superiority of the proposed scheme is enhanced when the upload bandwidth of the media server is not large compared with the number of participant peers.

7. Concluding Remarks

This paper proposes a hierarchical P2P architecture with the notion of swarms for P2P VoDs. The result of simulations indicates that the proposed scheme reduces the total waiting time of a randomized scheme by 24% and the load of the media server by 76%. A future work is to evaluate the performance of the proposed scheme in an environment in which several peers conduct VCR operation such as pause, jump and fast forward.

References


Yasuaki Yuji received his B.E. and M.E. degrees in information engineering from Hiroshima University in 2011 and 2013 respectively. His current research interests include peer-to-peer networks and distributed systems.

Satoshi Fujita received the B.E. degree in electrical engineering, M.E. degree in systems engineering, and Dr.E. degree in information engineering from Hiroshima University in 1985, 1987, and 1990, respectively. He is a Professor at Graduate School of Engineering, Hiroshima University. His research interests include communication algorithms, parallel algorithms, graph algorithms, and parallel computer systems. He is a member of the Information Processing Society of Japan, SIAM Japan, IEEE Computer Society, and SIAM.