Semi-Structured BitTorrent Protocol with Application to Efficient P2P Video Streaming

SUMMARY In this paper, we propose a method to enhance the download efficiency of BitTorrent protocol with the notion of structures in the set of pieces generated from a shared file and the swarm of peers downloading the same shared file. More specifically, as for the set of pieces, we introduce the notion of super-pieces called clusters, which is aimed to enlarge the granularity of the management of request-and-reply of pieces, and as for the swarm of peers, we organize a clique consisting of several peers with similar upload capacity, to improve the smoothness of the flow of pieces associated with a cluster. As is shown in the simulation results, the proposed extensions significantly reduce the download time of the first 75% of the downloaders, and thereby improve the performance of P2P-assisted video streaming such as Akamai NetSession and BitTorrent DNA.

key words: P2P-assisted video streaming, BitTorrent, chunk scheduling, P2P overlay

1. Introduction

BitTorrent is a distributed file sharing system released in 2001 [4]. In 2003, BitTorrent attracted considerable attentions by the developers in the OSS community as it distributed 30,000 ISO images of Red Hat Linux 9 in only three days, and nowadays, it is widely used for distributing major free softwares without overloading the server. In July 2018, Rainberry, the developer of BitTorrent, was taken over by TRON Foundation, which is the developer of content entertainment systems with a virtual currency called TRON, and as the result, it becomes to be able to explicitly reward contributing users with BTT tokens, to provide strong incentives.

BitTorrent client installed on a user’s machine proceeds the download of shared files in the following steps:

1. At first, it contacts a tracker listed in the torrent file corresponding to the target file to acquire a list of peers involved in uploading/downloading of the file. Such a group of peers, called a swarm, is created for each file, where the list returned by the tracker contains only a part of the entire swarm.
2. It then contacts individual peers in the list to establish a link with them. After that, the client becomes a part of the overlay network corresponding to the shared file.
3. The client sends a request to neighbors in the overlay to acquire a fragment of the shared file called piece, where the authenticity of acquired pieces is verified by checking the hash value of each piece with the values received from the tracker beforehand.
4. After receiving all pieces correctly, it restores the original file from them.

In the above protocol, each peer is forced to upload pieces as requested by the other peers while conducting the download, which implies that the upload capacity of the overall system increases as the swarm size increases. On the other hand, the number of pieces to be acquired from neighbors is proportional to the size of the shared file, since each piece has a fixed size such as 256 KB or a few MB. As such, pieces generated from a single shared file are sequentially acquired from neighbors in the overlay. The order of piece acquisition is determined by a piece selection policy, such as random, the rarest first, and the endgame mode, which are adaptively switched depending on the status of piece acquisition. Finally, in order to properly recognize the set of pieces currently held by its neighbors, each peer periodically exchanges the status of piece acquisition with neighbors in the overlay in the form of a bit vector called bitfield.

In this paper, we are going to enhance the download efficiency of the BitTorrent protocol by introducing the notion of structures into: 1) the set of pieces generated from a shared file and 2) the swarm of peers corresponding to the shared file. More specifically, as for the set of pieces, we introduce the notion of super-pieces called clusters, which is aimed to enlarge the granularity of the management of request-and-reply of pieces, thereby reducing the overhead associated with the management. The cluster size can be dynamically adjusted according to the environment (similar to the window control mechanism used in TCP), and if an acquired piece was contaminated for some reason, individual request is issued for resending the piece after discarding it. On the other hand, concerned with peers in a swarm, we organize a clique consisting of several peers with similar upload capacity, to improve the efficiency of the flow of piece stream associated with a piece cluster. With the notion of cliques, the behavior of peers can be seen from three different aspects so that:

1. the injection of a piece stream from outside the clique,
2. the circulation of acquired piece stream within the clique, and
3. the emission of the piece stream towards the outside of the clique.
With respect to the first and the third aspects, a clique can be seen as a big (super-)peer, and the second aspect is realized by a complete peer-connection inside the clique.

The above extensions could improve the efficiency of mesh-based P2P video streaming since those systems share several techniques such as swarm and un-choke policies with the BitTorrent protocol. However, since video streaming differs from simple file sharing in the sense that the acquisition of pieces close to the current playback position in the video should be given as high priority as rare pieces, we need to develop a specific method to properly reflect such features to the BitTorrent-based video streaming. In this paper, we discuss how the proposed semi-structured BitTorrent protocol can improve the performance of mesh-based P2P video streaming systems.

The remainder of this paper is organized as follows. Section 2 overviews related work with the details of BitTorrent protocol. Sections 3 and 4 propose a semi-structured BitTorrent protocol and an application to the mesh-based video streaming, respectively. Section 5 summarizes the results of simulations. Finally, Sect. 6 concludes the paper with future work.

2. Related Work

2.1 BitTorrent Protocol

In BitTorrent protocol, each peer maintains adjacency with a hundred of peers in a P2P overlay which dynamically changes over time, while the number of un-choked peers, which are the target of upload, is limited to be small to enable peers with poor communication environments to participate in the upload. The set of un-choked peers is determined by an un-choke algorithm, including the selection of a random peer which is called optimistic un-choke.

Each peer periodically exchanges a bitfield representing the set of acquired pieces with the adjacent peers, and during the download of a shared file, the availability of pieces in the neighborhood is estimated with the collected bitfields. When peer p is un-choked by a neighbor q, p informs q which piece it wants to receive from q, and after receiving the request, q transfers the requested piece to p. In the following, we call a peer which holds all pieces a seeder and the other peers leechers. Note that the request to the neighborhood is always issued by a leecher.

2.1.1 Peer Protocol

In BitTorrent, a number of messages are exchanged between peers over HTTP/TCP connections with port numbers from 6881 to 6889. Each message contains 4-byte integer representing the message length, message type of 1-byte length, and the payload, where messages of length 0 are used for the keep alive notification. All message types used in BitTorrent are summarized as follows:

0 Notify that the receiver is choked (choke)
1 The receiver is un-choked (un-choke)
2 Notify that the sender is interested in a piece held by the receiver (interested)
3 The sender is not interested in any piece held by the receiver (not interested)
4 The sender completes the download of requested piece and the check of hash values (have).
5 Indices of pieces which have been acquired by the sender in the form of a bit vector (bitfield).
6 Request for sending a designated piece (request).
7 Transfer the requested piece (piece).
8 Notify the cancel of request (cancel).

For example, if a message received from a neighbor is 00 00 02 05 f8, then the message header of 5-byte length indicates that the message length is 2 bytes and the message type is 5 (i.e., bitfield). Since the payload f8 of the message is 1111 1000 in binary representation, it indicates that the sender of this message holds the first five pieces generated from a shared file.

2.1.2 Transfer of Pieces

In BitTorrent, the piece transfer is realized as a sequence of sub-pieces of 16 KB length each, which are generated by dividing a piece into small fragments. Sub-pieces generated from a single piece can be received from either a single peer or multiple peers to support a kind of parallel download, and an upload of sub-pieces to other leechers can only be started after receiving all sub-pieces and completing the integrity check of the original piece by SHA-1 hash. In fact, it frequently occurs the discard of sub-pieces due to the failure of verification in BitTorrent, which suggests the limitation of the copyright protection through poisoning.

2.2 Related Work as a File Sharing System

After few years of the proposal of BitTorrent, a lot of research have been done on evaluating its performance as a distributed file sharing system and comparing it with other file sharing systems such as Gnutella. For example, Bharambe et al. showed that the performance of BitTorrent is nearly optimal with respect to the uplink bandwidth utilization and the download time, and Legout et al. demonstrated that the un-choke algorithms and the piece selection policy are certainly effective under typical settings. Similar evaluations have been conducted in [11], [22], including the modeling of the behavior of BitTorrent as a fluid.

However, it is pointed out that Tit-for-Tat used in BitTorrent is vulnerable to free riding, which motivated a lot of work on analyzing and improving the robustness against

---

1Optimistic un-choke is mainly used for bootstrapping. In the default configuration, one of the four un-choke destinations for each peer is the optimistic un-choke destination.

11Verification by hashing is similar to the role of I-frames in HTTP streaming, such as HLS and MPEG DASH.
selfish peers [16], [30]. For example, Manoharan et al. [20] proposed a demerit point strategy to suppress free riding, and Atlidakis et al. [1] extended the optimistic un-choke so that the strength of interest can be taken into account (recall that conventional optimistic un-choke selects a random peer as the target of upload). Azzedin et al. [2] introduced the notion of reciprocity used in the game theory to the optimistic un-choke to mitigate the influence of free riders.

As another direction of research, the efficient utilization of the upload bandwidth of peers has been a key issue for many researchers. For example, as a proposal from the attacker’s side, a client called BitTyrant [26] was proposed to exploit the weakness of Tit-for-Tat, and BitThief [17] was proposed to exploit the weakness of the optimistic un-choke. To overcome these weaknesses, Levin et al. proposed an auction-based un-choke algorithm to guarantee the fairness of the allocation of the upload bandwidth [13]. GA-BT [28] is an un-choke algorithm based on the genetic algorithm. GA-BT dynamically predicts an optimal fitness value by using the divisible load theory, to accelerate the convergence process for generating (pseudo-)optimal selection of un-choked peers. Rafit’s doctoral dissertation [25] proposed three enhancements of BitTorrent protocol. The first is a team-enhanced protocol which dynamically organizes peers with similar upload bandwidth into teams with the aid of a central entity, the second is a buddy-enhanced protocol which is based on pairs of peers with similar upload bandwidth, and the third is the acquisition of an optimal un-choke algorithm through the reinforcement learning. In response to the third proposal, [21] attempted to apply DQN in combination with the reinforcement learning and deep neural networks to acquire optimal un-choke algorithms.

2.3 Related Work as a Video Streaming Infrastructure

Several techniques used in BitTorrent have been applied to the P2P video streaming. For example, in mesh-based video streaming systems such as CoolStreaming/DONet [31], PRIME [19], and PPLive [27], it specifies which video chunks (pieces) are to be acquired from which neighbor, and such a selection is effectively done by periodically exchanging the current status of input buffer with its neighbors, similar to the exchange of bitfields in BitTorrent. However, a direct application of basic mechanisms in BitTorrent to video streaming might cause an unacceptable delay when the round trip time (RTT) to the neighboring peers is large. Therefore, many mesh-based systems take a pull/push approach, in which some important chunks are pushed automatically, while the rest are dispatched individually in response to the pull request.

The behavior of peers in mesh-based systems is generally controlled with a scheduling window (SW). With respect to SW, existing scheduling schemes are classified into three types, i.e., urgency-based, rarity-based, and their hybrid. Urgency-based schemes prioritize the streaming continuity by requesting chunks closer to the playback position of the user [7], [9], and rarity-based schemes try to improve the overall system performance as in CoolStreaming/DONet [31]. Although rarity-based schemes are not as effective as in file sharing systems since SWs of peers in a swarm are not synchronized, it certainly accelerates the redistribution of pieces within the common part of SWs [14]. PPLive [23] takes a hybrid approach so that the highest priority is assigned to the first chunk in the SW, and after completing the acquisition, it requests the rarest chunk in the SW. An optimal size of SW would vary depending on the type of scheduler. It is short for rarity-based schemes, but it can be large for urgency-based schemes, as long as it firstly acquires a chunk close to the playback position.

From the practical aspect, many works have been done to improve the performance of mesh-based systems in combination with Content Delivery Networks (CDNs). In fact, it is effective to divide the buffer into two regions so that chunks in the start-up region are supplied by CDN and the remaining part is filled with the neighboring peers, where the relative size of each region is determined by the upload capacity of CDN and the number of neighboring peers. Lu et al. [18] extended this idea so that the playback buffer is divided into three regions. Specifically, in addition to the above two regions, it introduces an emergency region which will be used when the necessary chunks cannot be acquired from neighbors by the deadline. In the same paper, Lu et al. proposed a method to determine the target of upload by taking into account the upload bandwidth of each peer as well as the stability and the contribution of the peer.

Hu et al. [10] proposed a request scheduler with three types of FCFS queues to reduce the startup delay. In this scheduler, requests issued by a peer are put into the first queue of the peer, which is forwarded to either the CDN edge server or a subset of adjacent peers depending on the region relevant to the request. If the requested chunk is not received by the deadline, an emergency chunk request is put into the second queue, and if the requested chunk can only be received from the edge server, the request is put in the third queue.

3. Semi-Structured BitTorrent Protocol

This section describes the details of the proposed method. The proposed method is an extension of BitTorrent and can be mixed with the original protocol so as to partially accelerate the piece distribution among peers supporting the extended protocol. However, to make the exposition clear, we will merely consider a partial overlay consisting of peers supporting the extended protocol. In addition, we assume that the number of pieces generated from a shared file is equal to a power of two, and that the system contains at least one seeder. An application of the scheme to the mesh-based video streaming will be described in the next section.

3.1 How to Organize Piece Clusters

Since pieces in BitTorrent are given sequential numbers starting from 0 as their unique IDs, we can naturally orga-
nize a piece cluster in the following way:

1. Regard the prefix of piece ID of an appropriate length as the cluster ID, and  
2. Extend the request message designating the piece ID so as to specify the prefix length as an option.

Suppose that peer $p$ wishes to acquire a piece from $q$. In the original protocol, such a piece acquisition proceeds as follows (see Fig. 1 (a) for illustration):

**Piece acquisition in the original BitTorrent:**

1. At first, $p$ sends interest message to $q$ to notify the possibility of sending a piece request.  
2. After being un-choked, $p$ sends request(ID) to $q$.  
3. $q$ returns the piece with designated ID to $p$.  
4. $p$ checks the hash of received piece, and after completing it, $p$ sends have(ID) to $q$.

A semi-structured version of the protocol, on the other hand, repeats the following steps (see Fig. 1 (b) for illustration):

**Extended piece acquisition flow:**

1. At first, $p$ sends interest message to $q$ to notify the possibility of sending a piece request.  
2. $p$ sends request(ID, prefix length) to $q$.  
3. If $q$ does not hold the whole cluster requested by $p$, $q$ declines the request. Otherwise, $q$ returns the requested cluster to $p$ in the form of a piece stream.  
4. $p$ checks the hash of received pieces in concurrent with the downloading, and after completing the check of all pieces in the cluster, $p$ sends have(ID, prefix length) to $q$. If it finds a mis-match of hash values during the check, $p$ sends have(ID) to $q$ for each piece passing the check.  
5. For large clusters, it might occur a situation in which $q$ wishes to choke $p$ during the transfer of piece stream to $p$. In such a case, the choke of $p$ is postponed until the end of the transfer.

We restrict the size of each cluster to a power of two since it enables us to change the cluster size by simply incrementing/decrementing the prefix length. Such a change of the cluster size dynamically occurs so that the increase is invoked by the sender $q$ and the decrease is invoked by the receiver $p$. More concretely, the sender $q$ of a piece stream continuously predicts the duration of un-choke period of $p$ by taking into account the past behavior of the neighboring peers, and gradually increases the cluster size. On the other hand, the receiver of the stream (i.e., requester) decreases the cluster size when the request for a cluster is rejected or the acquired cluster contains contaminated pieces. In both cases, the updated value of the cluster size is appropriately shared by peers $p$ and $q$.

### 3.2 Rafit’s Enhancement of BitTorrent Protocol

In [25], Rafit enhanced BitTorrent so as to dynamically organize team of peers with similar upload bandwidths. It is similar to the notion of cliques in our proposal, but is different from ours since the main objective of [25] is to exclude free riders by providing an alternative mechanism for the optimistic un-choke. Team organization is the responsibility of the tracker, and peers which falsify their upload bandwidth or do not behave as expected are blacklisted by the tracker. In addition, it protects whitewash and sybil attacks by limiting the movement to other teams.

Our proposal aims to further enhance the team-enhanced BitTorrent protocol with the notion of piece clusters. We follow the implementation shown in [25], which uses the following additional messages for the team participation and approvals:

- join-team request is sent from a leecher $r$ to the tracker with the upload bandwidth of $r$ as the parameter.  
- join-team response is returned from the tracker to $r$ with team ID and a list of team members.  
- join-team announcement is sent from the leecher to each team member.  
- join-verification request is sent from each team member to the tracker to request the approval of join of the leecher to the team.  
- join-verification response is returned from the tracker to team members to notify the approval of join of the leecher to the team.

### 3.3 How to Organize Cliques from Peers in a Swarm

Let $u(p)$ denote the maximum number of peers which can be simultaneously un-choked by peer $p$, which is determined by the upload bandwidth of $p$ and its stability\(^1\). In the following, we call $u(p)$ the upload capacity of peer $p$. Peer $p$ is said to be of class $i$ if it satisfies $2^i \leq u(p) < 2^{i+1}$ for some integer $i$.

In the proposed scheme, peers in the same class organize a clique, so that the maximum number of peers in a clique of class $i$ is bounded by $2^i$. Let $C$ be a clique of class $i$ consisting of $K$ peers, where $1 \leq K \leq 2^i$ should hold by definition. At first, clique $C$ is connected to a set of seeders, and after being un-choked by the seeders, it requests $K$ individual piece streams to these seeders.

---

\(^1\)In the default setting of BitTorrent, $u(p)$ is set to four.
of selecting piece clusters will be described later). Received streams are distributed within the clique, and at the same time, is forwarded to other cliques since each peer in the clique has at least one residual capacity after forwarding received stream to $K - 1$ members in the clique (note that it holds $K - 1 \leq 2^i - 1 \leq u(p) - 1$). See Fig. 2 for illustration. In other words, at the time of being created by the tracker (i.e., when the clique size is one), the upload capacity of the clique to the outside of the clique is at least $2^i$, and when the clique size becomes $K(>1)$, the upload capacity of the clique becomes at least $K \times (2^i - K + 1)$ which reaches $2^i$ when $K = 2^i$.

4. Application to Video Streaming

Next, we apply the above semi-structured BitTorrent to the mesh-based P2P video streaming. Key issues we need to consider are: 1) organization of cliques with the notion of scheduling window, and 2) the separation of a given video stream into sub-streams called stripes.

4.1 The Trigger for Organizing Cliques

In video streaming systems, the playback of a video proceeds at the same speed for all peers, unless video cassette recording (VCR) operation such as pausing and fast-forwarding takes place. This implies that it is reasonable to configure peer cliques at a certain point in time until the end of the playback, while to fully enjoy the effect of cliques, the timing of organizing a clique should be as early as possible and peers in the clique share as large portion of the scheduling window as possible. Thus in the proposed method, each peer explores clique members via trackers in parallel to starting to acquire the start-up area from the CDN, and gradually moves to the delivery through the clique after completing the organization. When a peer invokes a VCR operation, on the other hand, the peer tries to find an appropriate clique corresponding to the new scheduling window through tracker again, and becomes a new member of it. Yuji and Fujita proposed a concrete procedure for such a warm-based P2P video streaming [29].

4.2 Split a Video Stream into Stripes

The second point we consider is the way of splitting a piece stream into stripes. The notion of stripe-based video streaming was firstly proposed in Splitstream [5], and is known to enhance the performance of video streaming in terms of: 1) the efficient use of the upload bandwidth of peers and 2) a high tolerance against the fluctuation in the transmission rate. While the piece clustering method proposed in the last section focuses on the prefix of piece IDs, in the splitting of a given piece stream, we focus on the suffix of piece IDs, which could be used with the prefix-based clustering in a combined manner.
For example, consider the case in which the length of piece ID is 10 bits, the prefix length is 4 bits and the suffix length is 2 bits. Then a piece stream corresponding to a cluster consists of 64 (= $2^6$) pieces starting with a fixed prefix of 4 bits such as 0010, for example. Since the length of suffix determines the number of stripes, those 64 pieces are delivered in four (= $2^2$) stripes, as shown in Fig. 3, where those stripes could be delivered through different routes; e.g., if the distribution is realized through a clique consisting of four peers a, b, c, and d, the stripe with suffix 00 is received by a and is forwarded to the rest of the members, the stripe with suffix 01 is received by b and is forwarded to the rest of the members, and so on.

5. Evaluation

We evaluate the performance of the proposed method by simulation. In the simulation, we assume that the exchange of control messages between peers completes within one simulation step which roughly corresponds to 10 ms in the real-world. In addition, the transfer of a piece to the receiver takes $T_c$ time which is a random value drawn from 20 ms to 200 ms. Each peer periodically updates the set of un-choked peers for every 3000 steps (i.e., 30 s), which might not be synchronized with the neighbors since the start time of each peer is randomly shifted within the range of 80 steps.

In the experiment, we fix a random network consisting of $N = 128$ peers with edge probability 0.1 (i.e., each peer has 12.7 neighbors on average) and a designated seeder, and evaluate the download time of each peer by conducting 100 runs on the same network. The number of pieces generated from a shared file is fixed to $P = 1024$. The validity of this approach is verified by comparing the average download time over such 100 runs with the download time averaged over 100 different networks, as the preliminary experiment (namely, we randomly generated 100 different networks with the same parameters and took an average over those networks, to certify that the evaluation with a single random network is sufficient). Figure 4 summarizes the results. The horizontal axis is the list of peers sorted in an ascending order of the download time and dotted lines indicate the results for 100 different networks. It can be seen that for the first 75% of the peers, the download times are concentrated on a very narrow time range around 35000 steps for both cases. Figure 5 illustrates such a concentration in the form of the frequency distribution. This shows that the difference between the average and the standard deviation of two distributions is less than one second, and such a phenomenon could be observed for other $N$s such as 512 and 1024.

5.1 Effect of Piece Clustering

Next, we evaluate the effect of piece clustering by assuming that the transfer of a cluster takes 20 ms per piece in addition to constant $T_c$, where the actual transfer time per piece depends on the piece size and the network latency. The result for a single run illustrated in Fig. 6 suggests that an increase in the cluster size does not necessarily lead to the reduction in the download time. Figure 7 shows the download time averaged over 100 runs and the first 75% of the peers, when
the transfer time per piece is 20 ms and 100 ms, respectively. Although the download time gradually decreases as the cluster size, denoted DUP, increases for small DUPs, it takes a minimum value at a certain DUP, e.g., at DUP=64 for 20 ms and at DUP=16 for 100 ms, respectively. The reason why the download time does not decrease monotonically with the cluster size is as follows: 1) the requested peer might not have all of the pieces in a designated cluster, and 2) missing pieces must be retrieved by individual requests.

5.2 Effect of Peer Clique

The effect of peer cliques is evaluated as follows. We fix the clique size as \( c = 4 \) and consider a virtual network consisting of \( 32(= N/c) \) super-peers, where due to the contraction of peers into a super-peer, the edge density between super-peers becomes larger than 0.1. Figure 8 shows the impact of such a contraction to the download time, where the maximum number of un-chokes done by each super-peer is given as a parameter CAP. Note that the least case of CAP=4 (i.e., normal) occurs when all peers in each super-peer are leechers and the most case of CAP=16 occurs when all peers in the super-peer are seeders. This result indicates that the increase of CAP significantly reduces the download time. The reader should note that although the above experiments evaluate the impact of two proposed techniques independently, they could be applied to the original BitTorrent simultaneously since those approaches are orthogonal to each other.

5.3 Effect as a Part of Video Streaming

We have shown that the proposed method significantly reduces the average download time in BitTorrent protocol. Such an advantage is meaningful even for the mesh-based video streaming since the media-player should acquire all pieces before completing the overall playback, while the video streaming is different from simple file sharing in the sense of the existence of urgent pieces due to the playback and VCR operations. With respect to the urgent piece acquisition, the proposed method has a limitation such that it could not bound the maximum download time. However, such a flaw can be overcome with the support of CDN servers, and the proposed method is effective to reduce the load of such CDN servers since it realizes a short download time for at least 75% of the downloaders, while the magnitude of reduction depends on the bitrate of the video stream.

6. Concluding Remarks

This paper proposes a method to improve the download efficiency of BitTorrent protocol. Experimental results show that the proposed method significantly reduces the download time of the first 75% of the downloaders. A future work is to conduct performance evaluation in actual mesh-based video streaming systems.

\footnote{DUP means the number of requests which could be issued by the requester in a duplicated manner.}
References

[1] V. Attilakos, M. Roussopoulos, and A. Delis, “EnhancedBit: unleashing the potential of the unchocking policy in the BitTorrent protocol,” JPDC, vol.74, no.1, pp.1959–1970, 2014.

[2] F. Azzedin and M. Yahay, “Modeling BitTorrent choking algorithm using game theory,” Future Generation Computer Systems, vol.55, pp.255–265, Feb. 2016.

[3] A. Bharatme, C. Herley, and V.N. Padmanabhan, “Analyzing and improving a BitTorrent network’s performance mechanisms,” Proc. IEEE INFOCOM, pp.1–12, 2006.

[4] BitTorrent. 2008. http://www.bittorrent.org/

[5] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, “SplitStream: high-bandwidth multicast in cooperative environments,” ACM SIGOPS Operating Systems Review, vol.37, no.5, Dec. 2003.

[6] B. Cohen, “Incentives build robustness in BitTorrent,” Proc. the 1st Workshop on Economics of Peer-to-Peer Systems, 2003.

[7] J. Feng, “A research on scheduling strategy in peer-to-peer streaming media,” Proc. Pacific-Asia Conf. on Circuits, Communications and Systems, pp.439–442, 2009.

[8] S. Fujita, “Flash crowd absorber for P2P video streaming,” IEICE Trans. Inf. Syst., vol.E102-D, no.2, pp.261–268, Feb. 2019.

[9] Y. Guo, S. Mathur, K. Ramaswamy, S. Yu, and B. Patel, “PONDER: performance aware P2P video-on-demand service,” Proc. IEEE GLOBECOM, pp.225–230, 2007.

[10] C. Hu, M. Chen, C. Xing, and B. Xu, “EUE principle of resource scheduling for live streaming systems underlying CDN-P2P hybrid architecture,” Peer-to-Peer Netw. Appl., vol.5, no.4, pp.312–322, 2012.

[11] M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garcés-Erice, “Dissecting BitTorrent: five months in a torrent’s lifetime,” Proc. PAM, 2004.

[12] A. Legout, G. Urvoy-Keller, and P. Michiardi, “Rarest first and choke algorithms are enough,” Proc. IMC pp.203–216, Oct. 2006.

[13] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee, “BitTorrent is an auction: Analyzing and improving BitTorrent’s incentives,” Proc. SIGCOMM, vol.38, no.4, pp.243–254, Oct. 2008.

[14] J. Li, C. Yeo, and B. Lee, “Peer-to-peer streaming scheduling to improve real-time latency,” Proc. IEEE ICME, pp.36–39, 2007.

[15] X. Liao, H. Jin, Y. Liu, L.M. Ni, and D. Deng, “AnySee: peer-to-peer live streaming,” Proc. 25th INFOCOM, 2006.

[16] N. Liogkas, R. Nelson, and E. Kohler, “Exploiting BitTorrent for fun (but not profit),” Proc. IPTPS, 2006.

[17] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer, “Free riding in BitTorrent is cheap,” Proc. HotNets, 2006.

[18] Z. Lu, X. Gao, S. Huang, and Y. Huang, “Scalable and reliable live streaming service through coordinating CDN and P2P,” Proc. 17th ICPADS, pp.581–588, 2011.

[19] N. Magharei and R. Rejaie, “PRIME: peer-to-peer receiver-driven mesh-based streaming,” IEEE/ACM Trans Netw. vol.17, no.4, pp.1052–1065, Aug. 2009.

[20] S. Manoharan and T. Ge, “A demerit point strategy to reduce freeriding in BitTorrent,” Computer Communications, vol.36, no.8, pp.875–880, May 2013.

[21] T. Naito and S. Fujita, “Acquiring nearly optimal peer selection strategy through deep Q-network,” Proc. CANDAR, pp.120–125, 2018.

[22] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, “The BitTorrent P2P file-sharing system: measurements and analysis,” Proc. IPTPS, 2005.

[23] PPLive. http://www.pptv.com.

[24] D. Qiu and R. Srikant, “Modeling and performance analysis of BitTorrent-like peer-to-peer networks,” Proc. SIGCOMM, vol.34, no.4, pp.367–378, Oct. 2004.

[25] I.R. Raiff, Improving the BitTorrent protocol using different incentive techniques (Ph.D. thesis), University of California, Los Angeles, 2010.

[26] J. Shneidman, D.C. Parkes, and L. Massoulié, “Faithfulness in internet networks,” Proc. PINS, pp.220–227, Sept. 2004.

[27] M. Wang, L. Xu, and B. Ramamurthy, “Exploring the design space of multichannel peer-to-peer live video streaming systems,” IEEE/ACM Trans Netw., vol.21, no.1, pp.162–175, Feb. 2013.

[28] T. Wu, M. Li, and M. Qi, “Optimizing peer selection in BitTorrent networks with genetic algorithms,” Future Generation Computer Systems, vol.26, no.8, pp.1151–1156, Oct. 2010.

[29] Y. Yuji and S. Fujita, “Hierarchical architecture for peer-to-peer video on demand systems with the notion of dynamic swarms,” IEICE Trans. Inf. & Syst. vol.E97-D, no.12, pp.3025–3032, Dec. 2014.

[30] M. Zghiabeh and F. C Harmanztis, “Revisiting free riding and the Tit-for-Tat in BitTorrent: a measurement study,” Peer-to-Peer Netw. Appl., vol.1, pp.162–173, 2008.

[31] X. Zhang, J. Liu, B. Li, and Y.-S. Yum, “CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming,” Proc. 24th Annual Joint Conf. of the IEEE Computer and Communications Societies, pp.2102–2111, 2005.

Satoshi Fujita
received the B.E. degree in electrical engineering, M.E. degree in systems engineering, and Dr. 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, and IEEE Computer Society.