An Optimal Peer Selection Algorithm for Mesh-based Peer-to-Peer Networks

Seung Chul Han[1] and Ki Won Nam[2]
[1] Department of Computer Engineering, Myongji University, Korea
[e-mail: dr.seungchul@gmail.com]
[2] College of Education, Department of Early Childhood Education, Chungang University, Korea
[e-mail: bongbong@mju.ac.kr]
*Corresponding author: Ki Won Nam

Received January 25, 2018; revised June 3, 2018; accepted July 2, 2018; published January 31, 2019

Abstract

In order to achieve faster content distribution speed and stronger fault tolerance, a P2P peer can connect to multiple peers in parallel and receive chunks of the data simultaneously. A critical issue in this environment is selecting a set of nodes participating in swarming sessions. Previous related researches only focus on performance metrics, such as downloading time or the round-trip time, but in this paper, we consider a new performance metric which is closely related to the network and propose a peer selection algorithm that produces the set of peers generating optimal worst link stress. We prove that the optimal algorithm is practicable and has the advantages with the experiments on *PlanetLab*. The algorithm optimizes the congestion level of the bottleneck link. It means the algorithm can maximize the affordable throughput. Second, the network load is well balanced. A balanced network improves the utilization of resources and leads to the fast content distribution. We also notice that if every client follows our algorithm in selecting peers, the probability is high that all sessions could benefit. We expect that the algorithm in this paper can be used complementary to existing methods to derive new and valuable insights in peer-to-peer networking.

Keywords: Peer-to-peer, content distribution, peer selection, swarming, performance metric, link stress

This research was supported by the NRF grant funded by the Korea government (MEST) NRF-2017R1A2B1005285.

http://doi.org/10.3837/tiis.2019.01.008 **ISSN : 1976-7277**

1. Introduction

The conventional way of content distribution is based on the client-server architecture where all contents are stored at sever, and transmitted to client upon requests [1]. This model is simple and easy; but it exposes the limit for large number of users since the server frequently becomes a bottleneck as more clients join with more requests. And furthermore, the costs of deploying and maintaining server facility and service network can be significant.

An approach that is increasingly visible in content distribution is peer-to-peer (P2P) networks in which the source peer divides a content into pieces (or chunks) and sctters them across the network. A peer subsequently takes part in distributing and consuming of the content by exchanging data with other peers. When a user wants to download a file, the peer searches nodes with the file chunks, then connect with the peers and receive the file chunks simultaneously. This technique, called *swarming*, achieves faster content distribution speed, higher throughput, and resilience to errors and traffic fluctuation [2]. Moreover, as a P2P model pushes the computing and transmission cost toward the network edge, it enhances scalability for large number of users on a global scale without extra cost. Therefore, P2P network appears to be a promising approach for content distribution.

The P2P networks are generally catergorized into *tree-based* and *mesh-based* categories, based on the structures and functions they implement for content distribution. The tree-based P2P networks construct a tree at the beginning of the session. Peers are the nodes of the tree and push the data downward leaves. This approach requires little message overhead, but is not suitable to cope with the high rate of peer joining/leaving (i.e., churn). In the case of churn, the tree must be continuously rebuilt, which costs considerable cost and time. Moreover, this approach does not incorporate with the swarming mechanism which achieves higher throughput by allowing connected peers to exchange data in both directions. In a tree-based architecture, data always flows in one direction from parent to child peers, since there is only parent-child relationship between connected peers. In a mesh-based architecture, on the other hand, a user contacts tracker nodes to retrieve information of peers (i.e., candidates) which contain the file chunks, and receives from each peer on the list a buffer map, a map of the chunks of file they own. Then, the peer does construct a subset of peers and downloads data from the selected peers. Though it involves some overhead due to the exchange of buffer maps between peers, it usually offers good resilience to frequent peer joining/leaving and is congruous with the swarming mechanism [4].

However, we identify one fundamental problem. The problem is selecting a set of nodes participating in swarming sessions. The purpose of peer selection is to construct the best subset of peers so that the service quality is maximized and the network congestion is minimized. Because the swarming performance depends on the participating peers, a selection algorithm is essential for the swarming technique [5].

There are many researches proposed for the peer selction issue. But, most of them only deal with the user-centric metrics, such as the downloading or the round-trip time of each connections, little research has been done about bottleneck level of swarming sessions. For instance, in many existing systems, a peer periodically calculates the throughput of each connection of a swarming session and reconstructs a set of peers which can generate better performance (See Section 2 for a review.). These myopic strategies do not consider concurrent connections of a swarming session separately nor their possible interactions. However, we

know that network congestion level is important in the large service deployment network, such as IPTV, the quality of service of a streaming application is heavily influenced by bandwidth availability and network congestion.

This fact leads us to concentrate on a new network-centric metric, the WLS (*worst link stress*) which is the largest number of concurrent data flows on a edge. It reflects to the congestion level of bottleneck of the network. The minimum WLS improves utilization of the network capacity and gives faster content distribution.

In this paper, we consider a new performance metric, the worst link stress (WLS), and propose a peer selection algorithm that produces the set of peers generating optimal WLS. We assume that a user can get a peer-list which contains the desired file chunks (a.k.a. *candidates)*, and the route information from each candidate to the user. The user can then select peers from the candidates which will take part in the swarming session. We theoretically prove the optimality and analyze the complexity.

With the experiments on the *PlanetLab* [6], we show that our algorithm is effective in improving network performance in the real Internet. It has several advantages following; the algorithm optimizes the congestion level of the bottleneck link, it enables to provide larger number of users and service sessions. Second, the congestion of the network is distributed. It improves not only the utilization of network but also the speed of content distribution. Furthermore, we also notice that if every swarming session uses the algorithm, the overall network load is well balanced, which ultimately benefits all sessions.

The paper reviews related works on the peer selection problem and P2P swarming applications. In Section 3, we describe some definitions and assumptions which help us to understand the algorithm. In Section 4, we propose a new selection algorithm which minimizes the worst link stress, and analyze the time complexity. In Section 5, we compare the algorithm with existing algorithms in the real Internet test-bed, PlanetLab. In Section 6, we draw the conclusions.

2. Related Work

2.1 Peer Selection

There are many existing content distribution networks which solve the selection problem in a wide variety of ways. The traditional mirrors server-based content distributions usually select the nearest peers [3]. Many other systems employ various ranking functions. A user initially selects some random peers and keeps updating the peer set with better ranks. The selection process may use the RTT (*round-trip time*) [9] or the uploead/downloading capacity [10], or the overlap degree of content [11]. In [12], a peer in a structured locality-aware P2P network sends/receives more with closer peers. This helps reducing the total network traffic of the session.

While many previous related works have been directed to the selection problem, only a small number of them has been considered the bandwidth congestion in a swarming session. Important issues relative to the optimization of bottleneck and network load balance are not systematically covered by previous works.

2.2 P2P Swarming Applications

Many existing P2P systems, such as PPLive [15], TVAnts [16], SopCast [17], MiMeG [18], UUSee [19], PPStream [20], MySee [21], Pando [22], and Red Swoosh [23], are using the swarming technique.

PPLive is the largest chunk-driven streaming P2P overlay in the world. It supports video streaming distribution with a gossip protocol managing users and channels. PPLive prefers to download from geographically closer peers with high bandwidth. The selection policy has a strong greedy behavior; it continuously repeats switching peers with better ranks. TVAnts and SopCast are mesh-based P2P networks similar to BitTorrent. Super nodes keep track of nodes and file chunks. When a user contacts with peers directed by a super node, it receives buffer maps from each of them. After receiving the maps, the user selects peers to download the video chunks from. TVAnts implements a selection policy that chooses peers by selecting with high probability those within the same geographical region while in SopCast the choice is completely independent of the peer's location. Both systems largely prefer to download from high bandwidth peers. UUSee is a pull-based protocol in mesh-based P2P networks. Each peer calculates maximum upload capacity, and informs tracking servers its sending throughput. The servers manage these peers, and assign them on requests from users. In MiMeG, servers monitor the current uplink and downlink bandwidth consumption by each peer, and decide when and how to share videos among peers. UUSee and MiMeG select peers heuristically. They attempt to choose the peers with sufficient aggregate uplink capacity for serving the stream. Some other streaming services which are currently offered through P2P technology include: PPStream, MySee, Pando. We believe that most P2P swarming applications use the strategies described above in practice.

3. Preliminaries

3.1 Performance Metrics

Definition 1 Let $\mathcal{T}$ be a tree, and $S = \{s_1, \ldots, s_n\}$ is a subset of nodes in the tree and $E = \{e_1, \ldots, e_l\}$ is a set of edges used by the flows from a node in S to root. $LS(e)$, which denotes the link stress of a link e, $e \in E$, is the number of flows on e. The **WLS (worst link stress)** of nodes s_1, …, s_n is defined as,

$$WLS(s_1,\ldots,s_n) = \max_{e \in E} LS(e)$$

For example, in **Fig. 1**, if $s_1 = 1$, $s_2 = 2$, and $s_3 = 3$, then $WLS(s_1,s_2,s_3) = 3$ because the number of flows of edge (0,1) is 3.

The user locates at the root of $\mathcal{T}$ and the nodes in S are peers of a swarming session. The worst link stress (WLS) is the max number of paths on an edge and indicates the degree of congestion, it reflects how many data flows from different sources are overlapped on the most congested link. It represents the worst link stress loaded by concurrent connections. It also indicates the maximum number of users in as swarming session because the WLS represents how many individual connections can be added to the session. We found that swarming sessions following our algorithm in selecting peers improves the swarming performance across the network, and it leads to less overlap to other connections. The experimental results

show that if all the swarming session choose the WLS-min algorithm, the probability is high that all sessions can get higher performance (See Section 5.2).

Definition 2 Let $S = \{s_1, \ldots, s_n\}$ be a subset of nodes in a tree and $E = \{e_1, \ldots, e_l\}$ be the set of edges of the paths from $\{s_1, \ldots, s_n\}$ to root. The **DOI (degree of interference)** of nodes $s_1,\ldots,s_n$ is defined as,

$$DOI(s_1,\ldots,s_n) = \sum_{e \in E} (LS(e) - 1)$$

For example, in **Fig. 1**, if $s_1 = 1$, $s_2 = 2$, and $s_3 = 3$, then $DOI(s_1,s_2,s_3) = 3$.

The DOI is the sum of the path-lengths from all nodes of $S = \{s_1, \ldots, s_n\}$ minus the sum of the number of links. By its definition, the DOI represents the bandwidth usage of a swarming session [24], because the bandwidth usage of a swarming session is directly reflected by the average path length of each connection. It means it is related to the performance metrics, *work* [12]. The optimal peer selection algorithm which generates miminum DOI in a hypercube is presented in [24].

According to our experimental results in Section 5, the WLS and the DOI are highly orrelated because the DOI is mimimized when none of the edges have many data flows on them. Therefore, minimizing the WLS tends to reduce the DOI.

3.2 Constructing a Tree Rooted at User

In a mesh-based P2P network, the source breaks a file into pieces (chunks) and scattered them over the network. A peer contacts a tracker server and obtain information of peers (candidates) which contains the file chunks, and receives buffer maps. It then selects some peers from the candidates to establish concurrent connections.

3.2.1 Finding candidates

Each user can construct a set of candidate peers by exploiting underlying P2P substrates. In a structured P2P network, centralized tracker nodes generally manage the information of all the peers and a user can get the information from the tracker nodes. In a unstructured P2P network, a user can use some gossip protocol [10] or the distributed hash table (DHT) [25] to find candidates.

3.2.2 Inferencing network topology

In this paper, we assume that a use already has the topology information of the peers which can provide data at the rate required by the user. However, in practice, most routes/gateways do not reveal their network information such as packet loss rate, delays, bandwidth, or routing information. Thus, we need other methods to obtain necessary network information at the end users. [26] summarizes several useful methods to conjecture hidden network information at the end users. For example, network topology can be inferred by sending passive/active packets[27, 28]

However, in many cases, *traceroute* (*tracert* in Windows) has been proved to be the most practicable method to infer the network topology. The deterministic problem is proved to be a

NP-hard problem when intermediate nodes will not provide information [29]. Therefore, we use algorithms of [26, 30] to infer the network topology.

4. Optimal Peer Selection Algorithm

4.1 Problem Statement

The definition of selection problem stated in this section can be described as follows. Let a tree $\mathcal{T}$ have the root as the user and all the leaves and some internal nodes as the candidates. The user selects k nodes (i.e., peers) from the candidates to download the desired file chunk so that the swarming session has the minimum WLS.

First, we conduct the *depth-first search* algorithm on the given tree $\mathcal{T}$ and label each node of the tree by the order of traversing. Let $I(u)$ be the label of node u and $\mathcal{T}_u$ be the subtree rooted at node u. **Table 1** summarizes the notations and definitions which are needed in understanding the optimal algorithm.

Table 1. Notations and Definitions

$\mathcal{T}$	Tree whose root is user and candidates are located at some non-leaf nodes and all the leaf nodes
$\mathcal{T}_v$	Subtree of $\mathcal{T}$ whose root is v
S	Candidate set
S_v	Candidate set of $\mathcal{T}_v$
$\hat{\mathcal{T}}$	Virtual tree constructed by *SLCA(S)*. See Definition 6.
M_v	Candidate set, v is the immediate ancestor in $\hat{\mathcal{T}}$ without any descendants in S
$\Pi(v)$	Children of v in $\hat{\mathcal{T}}$.
k	Number of nodes to be selected
$q(v)$	Number of nodes to yet be selected from $\mathcal{T}_v$
$LCA(S)$	The lowest common ancestor of S. See Definition 4.

Definition 3 Let $\mathcal{T}$ be a rooted tree with node set, $S = \{s_1, \ldots, s_m\}$, $m > 1$. A node u is a **common ancestor** of S if all the paths of $\{s_1, \ldots, s_m\}$ to root contain u.

Definition 4 Let $\mathcal{T}$ be a rooted tree with node set, $S = \{s_1, \ldots, s_m\}$, $m > 1$. The **lowest common ancestor** of S is denoted by $\boldsymbol{LCA(S)}$ or $\boldsymbol{LCA(s_1, \ldots, s_m)}$.

Definition 5 Let $\mathcal{T}$ be a rooted tree with node set, $S = \{s_1, \ldots, s_m\}$, $m > 1$. Let denote the set of LCAs of all the subsets of S as $\boldsymbol{SLCA(S)}$.

4.2 Sketch of the Algorithm

The key observation is that $LS(u,v)$, where v is a child of u, is more than that of any link in $\mathcal{T}_v$. Hence, one of the edges connected to the root must have the worst link stress. We assume w is a children of root and W is the set of children of root node. We denote a set of candidates in $\mathcal{T}_w$ as S_w. Let $|S_w|$ be the number of nodes in S_w and we label each $w \in W$ with $|S_w|$. After labeling, *Algorithm 1* (see Section 4.4) is called with parameters $(l, (b_1, \ldots, b_l), q)$, l is the number of sets; b_j, is the number of items in set j, $1 \le j \le l$; q is the total number of items to be selected.

Then, the *Algorithm 1* returns a list $(c_1, \ldots, c_l)$, c_j is the number of items selected from each each $\mathcal{T}_w$.

Based upon the above description, the main algorithm (*Algorithm 2*) optimizes the WLS recursively when it optimizes the WLS of $\mathcal{T}_v$, $v \in SLCA(S)$. In Section 4.3, we provide descriptions of *SLCA* so as to make the *Algorithm 2* clear. We present descriptions of *Algorithm 1* and *Algorithm 2* in Section 4.4 and 4.5, and analyze the time complexity of them. In Section 4.6, we completely describe the WLS-min algorithm and discuss the total time complexity.

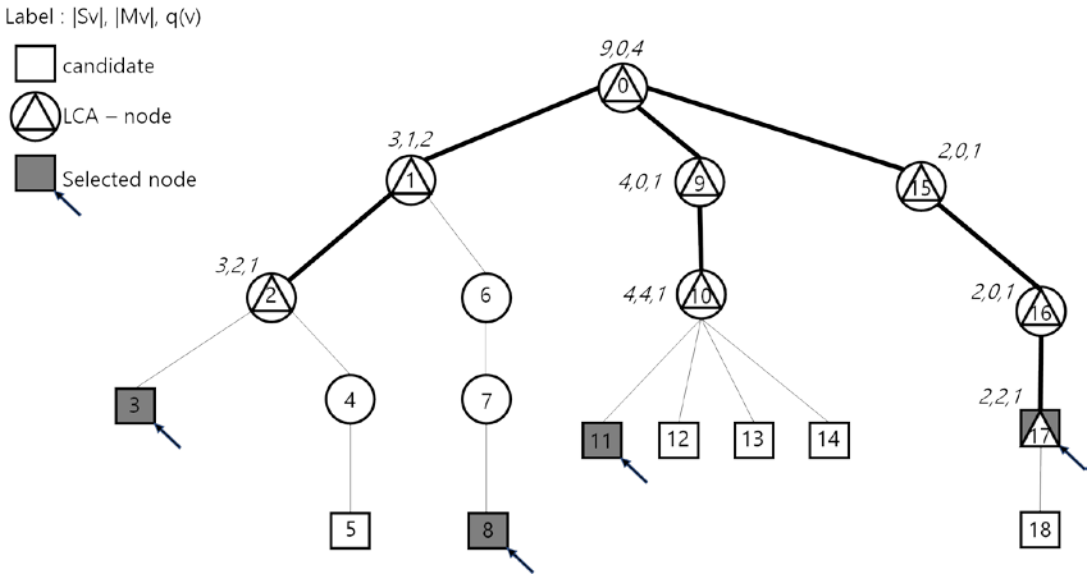

Fig. 1. Peer selection example.

4.3 *SLCA(S)*: set of LCAs of all subsets of *S*

Lemma 1 *Let $\mathcal{T}$ be a rooted tree, traverse $\mathcal{T}$ by using the DFS (depth-first search) algorithm and label each node of the tree by the order of traversing. Let u and v be nodes of $\mathcal{T}$, and $I(u) < I(v)$. Then, the relation of u and v must be*

$$LCA(u, v) = u, \text{ or } \mathcal{T}_u \cap \mathcal{T}_v = \Phi$$

Proof. The proof is too simple due to the property of the depth-first search algorithm itself, we omit it to keep page limitation.■

Lemma 2 *Let $\mathcal{T}$ be a rooted tree with node set, $S = \{s_1, \ldots, s_m\}$, $m > 1$. Let $W_1, \ldots, W_n$ be a covering of S, which is, $W_i \subseteq S$ for $1 \leq i \leq n$, and $\cup_{i=1}^{n} W_i = S$.*

$$LCA(S) = LCA(LCA(W_1), \ldots, LCA(W_n))$$

Proof. Let $u = LCA(S)$, by defnination, u is a common ancestor of all the nodes of W_i, $1 \leq i \leq n$. It means that u is the lowest common ancestor of all the $LCA(W_i)$'s. If $v \neq u$ and is the LCA of all the $LCA(W_i)$'s, then v must be a descendant of u as well as a common ancestor of S. This is contradictory to the assumption.

Lemma 3 *Supposet* (s_1, s_2, s_3) *is a list of nodes of tree* $\mathcal{T}$.

$$LCA(s_1, s_2, s_3) = LCA(s_1, s_3)$$

Proof

$$LCA(s_1,s_2,s_3) = LCA(LCA(s_1,s_2), LCA(s_2,s_3))$$

The equality above is by Lemma 2. By Lemma 1, the relation of (s_1,s_2) and (s_2,s_3) must be following cases.

- Case 1: $LCA(s_1,s_2) = s_1$ and $LCA(s_2,s_3)=s_2$
  $LCA(s_1, s_2, s_3) \rightarrow LCA(LCA(s_1, s_2), s_3) \rightarrow LCA(s_1, s_3)$
- Case 2: $LCA(s_1,s_2) = s_1$ and $\mathcal{T}_{s2} \cap \mathcal{T}_{v3} = \Phi$
  Same as Case 1.
- Case 3: $\mathcal{T}_{s1} \cap \mathcal{T}_{s2} = \emptyset$ and $LCA(s_2, s_3) = s_2$
  By Definition 4, $LCA(s_1,s_2)=LCA(s_1,s_3)$.
  $LCA(s_1, s_2, s_3) \rightarrow LCA(s_1, LCA(s_2, s_3)) \rightarrow LCA(s_1, s_2) \rightarrow LCA(s_1, s_3)$
- Case 4: $\mathcal{T}_{s1} \cap \mathcal{T}_{s2} = \emptyset$ and $\mathcal{T}_{s2} \cap \mathcal{T}_{s3} = \emptyset$
  Assume $LCA(s_1,s_2,s_3) \neq LCA(s_1, s_3)$ and let $v=LCA(s_1,s_2,s_3)$. By the definition of the depth first search, $LCA(s_1,s_3)$ is an ancestor of s_1, s_2, and s_3. If $LCA(s_1,s_3) \neq v$, $LCA(s_1,s_3)$ must be an ancestor of v, which contradicts with $LCA(s_1,s_3)$. ■

Lemma 4 *Supposet* (s_1, s_2, s_3) *is a list of nodes of tree* $\mathcal{T}$.

$$LCA(s_1, \ldots, s_n) = LCA(s_1, s_n)$$

Proof We use the induction on n. The base case n=3 has been proven in Lemma 3. Then, we assume that it is true for $(s_1, \ldots, s_l)$, $3 < l < n$. The next step is to prove it is true for $(s_1, \ldots, s_{l+1})$.

$$\begin{aligned} LCA(s_1, \ldots, s_{l+1}) &= LCA(LCA(s_1, \ldots, s_l), s_{l+1}) && (1)\\ &= LCA(LCA(s_1, s_l), s_{l+1}) && (2)\\ &= LCA(s_1, s_l, s_{l+1}) && (3)\\ &= LCA(s_1, s_{l+1}) && (4) \end{aligned}$$

(1) and (3) are by Lemma 2, (2) is by assumption, (4) is by Lemma 3.

Lemma 5 *Supposet* (s_1, s_2,s_3) *is a list of nodes of tree* $\mathcal{T}$.

$$SLCA(s_1,\dots,s_n) = \bigcup_{i=1}^{n-1} LCA(s_i,s_{i+1})$$

Proof Let $S = \{s_1, \dots, s_n\}$. By Lemma 2 and Definition 3,

$$SLCA(S) = \cup_{1\le i<j\le n} LCA(s_i,s_j)$$

We prove by induction. Suppose $S^i = \{s_1, \dots, s_i\}$, for $2 \le i \le n$. The base case, $i=2$, it is simple to see the lemma is true. We assume that it is true for S^l, where $2 \le l<n$.

$$SLCA(S^{l+1}) = \left(\bigcup_{i=1}^{l-1} LCA(s_i,s_{i+1})\right) \cup \left(\bigcup_{i=1}^{l} LCA(s_i,s_{l+1})\right) \quad (5)$$

Consider $LCA(s_i, s_{l+1})$ for some $1 \le i \le l$-1. Then,

$$\begin{aligned} LCA(s_i, s_{l+1}) &= LCA(s_i, s_l, s_{l+1}) && \text{by Lemma 4} \\ &= LCA(LCA(s_i,s_l), LCA(s_l,s_{l+1})) && \text{by Lemma 2} \\ &= LCA(s_i,s_l) \text{ or } LCA(s_l,s_{l+1}) \end{aligned}$$

The first equality is by Lemma 4, the next equality is by Lemma 2, and the third equality is true because $(LCA(s_i,s_l) = s_l)$ *or* $(LCA(s_l,s_{l+1}) = s_l)$ *or* $(LCA(s_i,s_l) = LCA(s_l,s_{l+1}) = s_l$.
We already showed that $LCA(s_i, s_{l+1})$ is either the same as $LCA(s_l, s_{l+1})$, or $LCA(s_i, s_l)$, as in $SLCA(S^l)$. Therefore, by (5)

$$SLCA(S^{l+1}) = \bigcup_{i=1}^{l} LCA(s_i,s_{i+1})$$

Theorem 1 *The time complexity of line 5 in Algorithm 2 is* $O(n^2)$.

Proof By Lemma 5, the running time for constructing the set of LCAs of all pairs of consecutive nodes in S is $O(n^2)$. ■

4.4 Optimal allocation (Algorithm 1)

The *Algorithm 1* is called with arguments, $(l, (b_1, \dots, b_l), q)$, where $(b_1, \dots, b_l)$ is a list of integers, and l and q are integers, and returns a list of l integers.

Assume l sets and set j has b_j elements, and we select total q elements from these l sets. The *Algorithm 1* returns the list of the number of elements taken from each set c_j, for $1 \le j \le l$, so as to minimize the maximum c_j and $\sum_{j=1}^{l} c_j = q$. In other words, it minimizes the maximum number of elements to be taken from each set. *Algorithm 2* calls *Algorithm 1* to decide the number of peers to be selected from each subtrees.

ALGORITHM 1: MIN-MAX$(l, (b_1, \ldots, b_l), q)$

1: **input**: l and q are integers, l is the number of sets and q is the total number of elements to be selected from l sets.

2: **output**: list of $(c_1, \ldots, c_l)$, c_j ($1 \le j \le l$) is the number of elements to be chosen from set j, $\sum_{j=1}^{l} c_j = q$ *and* $max_{1 \le j \le l}\ c_j$ is minimized.

3: **if** $\sum_{j=1}^{l} b_j \le q$ **then**

4: **return** $(b_1, \ldots, b_l)$

5: **end if**

6: $\mathrm{sort}(b_j)_{j=1}^{l}$ by ascending order and reindex as $b_1 \le b_1 \le \cdots \le b_l$

7: $t \leftarrow 0$

8: $r \leftarrow q$

9: **for** $j = 1$ to l **do**

10: $\delta \leftarrow b_j - t$

11: **if** $\delta(l - j + 1) < r$ **then**

12: $t \leftarrow b_j$

13: $r \leftarrow r - \delta(l - j + 1)$

14: **else**

15: $\delta \leftarrow \lfloor r \,/\, (l - j + 1) \rfloor$

16: $t \leftarrow t + \delta$

17: $r \leftarrow r - \delta(l - j + 1)$

18: **break**

19: **end if**

20: **end for**

21: **for** $j = 1$ to l **do**

22: $c_j \leftarrow \min(b_j, t)$

23: **end for**

24: **for** $j = l$ down to $l - r + 1$ **do**

25: $c_j \leftarrow c_j + 1$

26: **end for**

27: recover the original set indices

28: return $(c_1, \ldots, c_l)$

4.5 Construction of $\hat{\mathcal{T}}$

The *Algorithm 2* (Section 4.6) traverses all the nodes of the virtual tree $\hat{\mathcal{T}}$ which is formed by the nodes $\hat{V} = SLCA(S) \cup S$. The virtual tree is constructed at line *6* of *Algorithm 2*.

Definition 6 Let *S* be a set containing multiple nodes of tree $\mathcal{T}$, the virtual tree $\hat{\mathcal{T}} = (\hat{V}, \hat{E})$ consists of the nodes $\hat{V} = SLCA(S) \cup S$. If *u* and *v* are in $\hat{V}$ and *u* is the parent of *v*, then $(u, v) \in \hat{E}$. *LCA(S)* is the root of $\hat{\mathcal{T}}$.

Fig. 1 shows an example of the virtual tree $\hat{\mathcal{T}}$. All the nodes in triangle are in $\hat{V}$ and the edges of $\hat{\mathcal{T}}$are highlighted. The the virtual tree $\hat{\mathcal{T}}$ of $\mathcal{T}$ can be easily derived by Lemma 5. In line 6, the nodes of *S* are sorted by their IDs. Suppose $S = (s_1, \ldots, s_n)$ is a sorted list, then $\hat{V} = \{LCA(s_1, s_2), LCA(s_2, s_3), \ldots, LCA(s_{n-1}, s_n)\}$. Hence, $r = LCA(s_1, s_n)$ is the root of $\hat{\mathcal{T}}$ due to Lemma 4. All the edges in $\hat{E}$ can be identified by the routes from all the nodes of $\hat{V}$ to the root. Thus, the time complexity of constructing $\hat{\mathcal{T}}$ is $O(n^2)$.

Lemma 7 *The time complexit of line 6 in Algorithm 2 is* $O(n^2)$.

4.6 Peer Selection with minimum WLS

Algorithm 2 traverses all the nodes, $\hat{V}$, in the virtual tree $\hat{\mathcal{T}}$ by the *breadth-first* manner. It has a queue, *Q*, which stores nodes to visit. Suppose M_v is the set of nodes whose parent in $\hat{\mathcal{T}}$ is *v* and have no descendants in *S*. Suppose S_v , $v \in \hat{V}$, is a set of candidates of $\hat{\mathcal{T}}_v$. Then, $M_v \subset S_v$, and for each node $w \in M_v$, $w \neq v$, *w* has no descendants and *v* is the only virtual node which is on the routes from *w*.

4.6.1 A peer selection example

Fig. 1 present an example of peer selection. Suppose a tree whose root node is the user and the squared nodes, $S = \{3,5,8,11,12,13,14,17,18\}$, are candidates. We select $k=4$ nodes to connect from the candidates with the minimum worst link stress. The virtual nodes, $\hat{V}$, are triangled, and the virtual edges of $\hat{\mathcal{T}}$ are highlighted. The optimal peer set generated by Algorithm 2 are pointed by arrow. The root node selects 4 peers from its 3 subtrees. The *Algorithm 1* allocates (2, 1, 1) nodes to be selected from subtree from $\mathcal{T}_1$, $\mathcal{T}_9$, and $\mathcal{T}_{15}$. Then, the algorithm is called recursively at $\mathcal{T}_1$, $\mathcal{T}_9$, and $\mathcal{T}_{15}$, respectively.

4.6.2 Extending virtual tree $\hat{\mathcal{T}}$

The *Algorithm 2* constructs an extended virtual tree $\hat{\mathcal{T}}^e$ which has nodes *u*, $u \in SLCA(S)$ or $u \in S$ and *edges (u, v)*, path from *v* to root includes *u* without any node of $\hat{\mathcal{T}}^e$ is on the route from *v* to *u*. In line 7, $\hat{\mathcal{T}}^e$ is built by adding nodes and edges in *S*. For each node *v* which is in *SLAC(S)*, *Algorithm 2* counts the number of nodes, $|S_v|$, under the subtree $\mathcal{T}_v$. After building $\hat{\mathcal{T}}^e$, the set M_v and its size $|M_v|$ are stored by node *v*. Thus, the time complexity of line 7 is $O(n^2)$, it is same to result of Lemma 7.

ALGORITHM 2: WLS-MIN (*S*, *T*, *k*)

1: **Input**:
$S = \{s_1, \ldots, s_n\}$, candidate nodes
T : Tree whose root is user and candidates are located at some non-leaf nodes and all the leaf nodes
k: number of nodes to be selected, $1 \le k \le n$
2: **Output**:
G: selected *k* nodes with optimal WLS
3: $\mathrm{G} \leftarrow \emptyset$
4: traverse *T* by using DFS (depth-first search) algorithm and label each node of the tree by the order of traversing
5: $\mathrm{SLCA}(S) \leftarrow \{LCA(s_1, s_2), LCA(s_2, s_3), \ldots, LCA(s_{n-1}, s_n)\}$
6: Construct $\hat{T}, r \leftarrow$ root of $\hat{T}$
7: At each $\upsilon \in \mathrm{SLCA}(S), record\ |S_v|, M_v, k(v) \triangleq |M_v|$
8: $Q \leftarrow \{r\}$
9: $q(r) \leftarrow k$
10: **while** $\mathrm{Q} \neq \emptyset$ **do**
11: $u \leftarrow \mathrm{haed}[Q]$
12: **if** $q(u) > k(u)$ **then**
13: $\mathrm{G} \leftarrow \mathrm{G} \cup M_v$
14: $q(u) \leftarrow q(u) - k(u)$
15: **else if** $0 < q(u) \le k(u)$ **then**
16: Insert subset of M_v with $|q(u)|$ items to *G*
17: $\mathrm{q}(u) \leftarrow 0$
18: **end if**
19: **if** $\mathrm{q}(u) \neq 0$ **then**
20: {Optimally allocate $q(u)$ *to* $v \in \Pi(u)$}
21: $\left(q(v)\right)_{v \in \Pi(u)} \leftarrow \mathrm{MIN-MAX}\left(|\Pi(u)|, (|S_v|)_{v \in \Pi(v)}, q(u)\right)$
22: **for** $\upsilon \in \Pi(u)$ **do**
23: **if** $\mathrm{q}(v) > 0$ **then**
24: Enqueue(Q, v)
25: **end if**
26: **end for**
27: **end if**
28: Dequeue(Q)
29: **end while**
30: **return** *G*

4.6.3 Description of lines *10 ~ 29*

Suppose *q(u)*, $u \in \mathcal{T}_u$, is number of peers yet to be selected. In lines 12 through 18, *Algorithm 2* tries to select the nodes which are in M_u. These nodes are either leaf nodes of $\hat{\mathcal{T}}^e$ or nodes of *S*. In line 21, *Algorithm 1* is called to optimally allocate the number of peers to be selected at each of subtree rooted at the children of *u* in $\hat{\mathcal{T}}$ and inserts the value of allocation into the queue, *Q* (lines 23 through 25).

4.6.4 Time complexity of Algorithm 2

The time complexity of the *while* loop can be analyzed as below.

Lemma 8 *The lines 10 through 29 of Algorithm 2 can be done in O(n log n).*
Proof The worst case scenario is when the root of the virtual tree $\hat{\mathcal{T}}$ has *O(n)* children. In line 16, it inserts a node to the set *G* at most *n* times since the loop iterations can not exceed |*G*|. In lines 22 through 26, the *for* loop iterations can not exceed *O(n)*, since it visits every nodes of *SLCA(S)* one by one.

We assume that the size of the candidate set *S* is *n* with virtual tree $\hat{\mathcal{T}}$ containing *l* non-leaf nodes which are denoted by $v_1,\ldots,v_l$. Then, the size of node set $\hat{\mathcal{V}}$ can not exceed *n-1*, and *l* is no greater than *n-2*. We assume node v_i has n_i immediate descendants in $\hat{\mathcal{T}}$, for $i = 1, \ldots, l$. Then,

$$\sum_{i=1}^{l} n_i = (|\hat{\mathcal{V}}| - 1) \leq (n-2)$$

In lin 21, the *Algorithm 1* is called *l* times, and the number of operations taken is $n_i \log n_i$ for each v_i. If *T(n,l)* is the number of operations of the worst scenario, then

$$T(n,l) \leq (n-2) \log ((n-2)/l) \qquad (6)$$

T(n,l) is maximized when *l* is *1*. Therefore, the number of operations for a tree containing *n* candidates is of *O(n log n)*.

Theorem 2 *The time complexity of Algorithm 2 is* $O(n^2)$.
Proof By Theorem 1 and Lemma 7 and 8, the time complexity of *Algorithm 2* is $O(n^2)$.

5. Evaluation

5.1 Case of single user

The *PlanetLab* [6] is an open platform for developing Internet technology on a global scale. It is composed of 1353 nodes at 717 sites worldwide in current.

We randomly selected three nodes from United States, Japan and France as the users[1]. We assume each user found *90* nodes as the candidates. These candidates were randomly selected with different random seeds in each experiment. Each user selects 15 nodes from 90 candidates for a swarming session. We ran *traceroute* and collected routing information to construct network topology. We checked each routing paths and calculate the RTTs, DOIs and WLSs. For each experiment, different candidate nodes were randomly selected and we averaged the measures of the experiments.

We present the comparison results of the *WLS-min* algorithm with existing selection algorithms, the random algorithm where the user randomly selects peers whereas the closest algorithm selects the nearest peers with the RTT criterion. They are the most popular methods in current P2P applications. (See Section 2.)

[1] They are *planetlab1.eecs.ucf.edu*, *planet0.jaist.ac.jp* and *planetlab1.eurecom.fr*.

We use three performance metrics to compare these algorithms.

- **Worst link stress (WLS)**: We count the number of flows on each link to calculate the worst link stress of a link and average 10 worst link stresses.
- **Degree of interference (DOI)**: We calculate the DOI generated by the concurrent connections to measure the burden of the swarming session.
- **Average number of flows on a link**: We calculate the average number of flows of a link to measure the load-balance of a network.

In **Fig. 2**, we measure the number of routes on each link and average the 10 largest for the random algorithm, the closest algorithm and the *WLS*-min algorithm, respectively. The *WLS*-min algorithm clearly shows the best performances in all cases. If a user in *planetlab1.eecs.ucf.edu* used the *WLS*-min algorithm rather than the closest algorithm, the average of 10 worst stresses was reduced about *44%*. Though the closest algorithm minimizes the RTTs of connections, it may cause congestion if the selected peers are nearby. The *WLS*-min algorithm effectively reduces the bottleneck stress of a swarming session.

We measure the DOI of a swarming session in **Fig. 3**. The results show similar trends to the results in **Fig. 2**. We see that the *WLS-min* algorithm offers much lower DOI than others. For instance, the *WLS*-min algorithm generated *35%* less DOI than the random algorithm and the closest algorithm in average. It is interesting that the closest algorithm is not remarkably better than the random algorithm in the DOI criterion.

Fig. 4 compares the average number of data flows on a link. The *WLS-min* algorithm performs best, and the closest is the worst in all cases. The average numbers of the random algorithm, the closest algorithm and the *WLS-min* algorithm are *1.83, 2.27* and *1.48*. The *WLS-min* algorithm is best in distributing the link stress or network load.

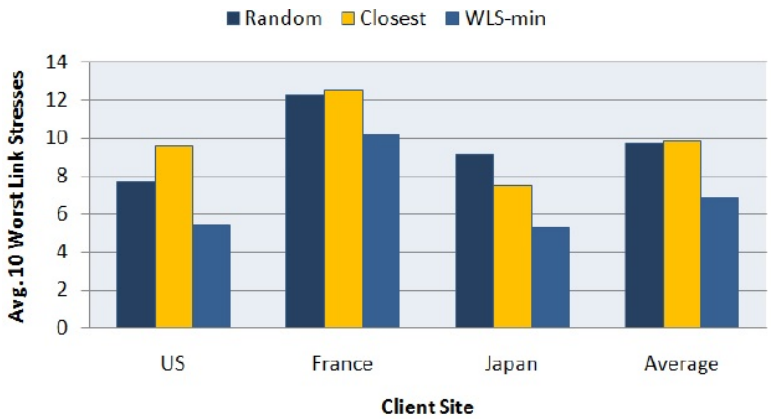

Fig. 2. The average link stress of 10 most congested links

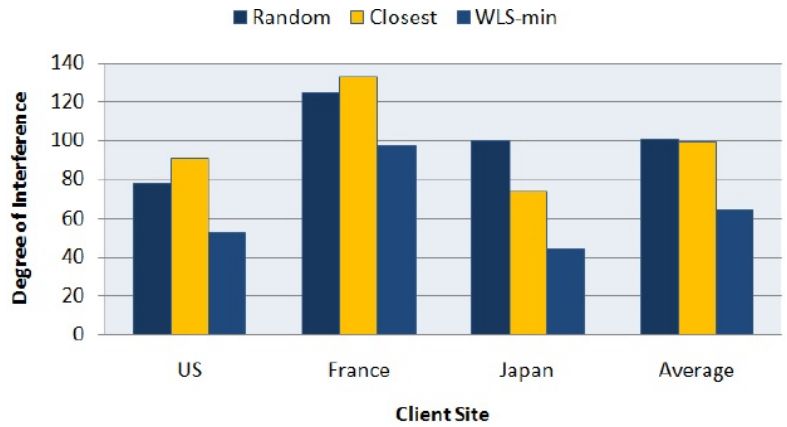

Fig. 3. The DOI measures

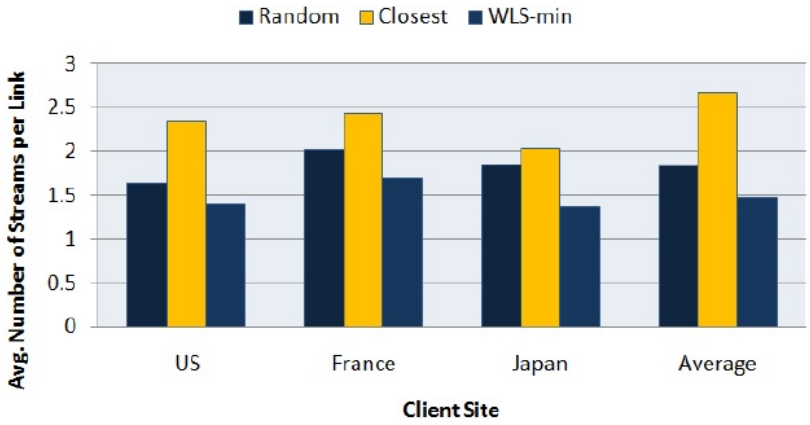

Fig. 4. The average of flows on a link

5.2 Case of multiple users

In this experiment, we show the benefits of the proposed algorithm for multiple concurrent swarming sessions. When every session uses the *WLS-min* algorithm in selecting peers, the probability is high that all sessions can benefit.

Our experiments were simulated on the *Transit-Stub* model[2] [31] where domains are classified into two types: transit domains and stub domains. Nodes in a stub domain are typically an endpoint in a network flow and nodes in a transit domain are typically intermediate routes. Nodes within a domain tend to be fairly interconnected but rarely connect to nodes outside of the domain. Our model consists of *2100* nodes with two level layers and the average diameter is about *12.0*. We assume all the links have similar delay and bandwidth. We do not consider either packet loss or nodal delay. We conduct simulations with different random seeds and present the averages.

There are *20* concurrent swarming sessions, each has a user and *100* candidates which are randomly scattered over the transit-stub model. Each user chooses $k=20$ peers from its

[2] We conducted this experiment on the *Transit-Stub* model instead of PlanetLab because the experiments were misunderstood as DDOS attacks.

candidates using same selection scheme.

We calculate the average number of routes on the 10 worst congested links for the three selection algorithms in **Fig. 5**. The random and the *WLS-min* algorithm generate better results as compared to the closest algorithm. The WLS-min algorithm works well in improving the bottleneck level, the average number of routes on the 10 most congested links was reduced from *31.4* to *22*.

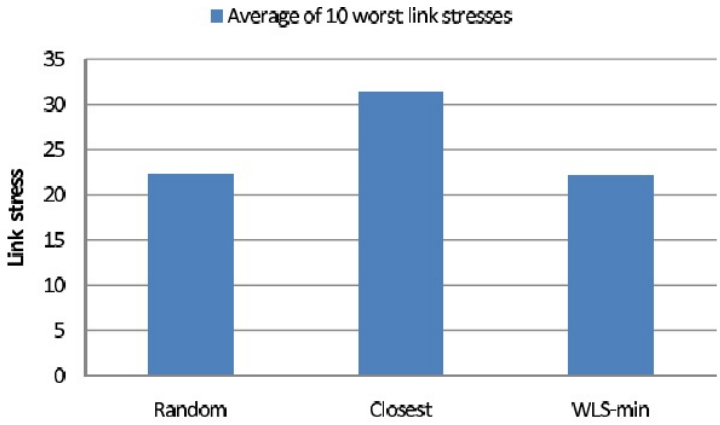

Fig. 5. Average of flows on the *10* most congested links with 20 concurrent swarming sessions

In **Fig. 6**, we present the DOI of all the swarming sessions. The results show similar trends to the results in **Fig. 5.**

In **Fig. 7,** we show the average of flows on a link. We see that the *WLS-min* algorithm offers lower result than the others. The average numbers of the random algorithm, the closest algorithm and the *WLS-min* algorithm are *3.1*, *3.32* and *2.81*, respectively.

Fig. 8 shows the distribution of the link stress. It is remarkable that the slope of the curves of the closest algorithm is relatively moderate and longer than the random algorithm and the *WLS-min* algorithm. If every user follows the *WLS-min* algorithm in selecting peers, all the links will have less than *24* routes on it. It means the *WLS-min* algorithm works well in balancing network stress for multiple concurrent swarming sessions.

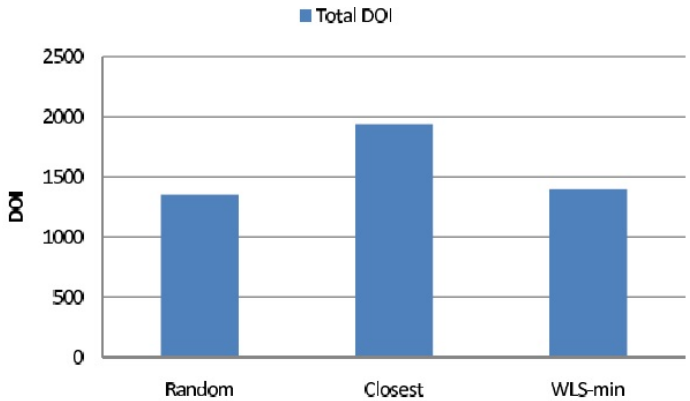

Fig. 6. The DOI of all connections for the case of 20 concurrent swarming sessions

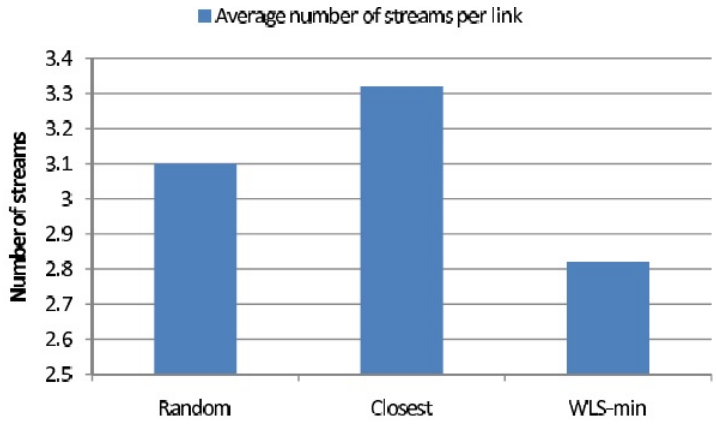

Fig. 7. The average of flows on a link for the case of 20 concurrent swarming sessions

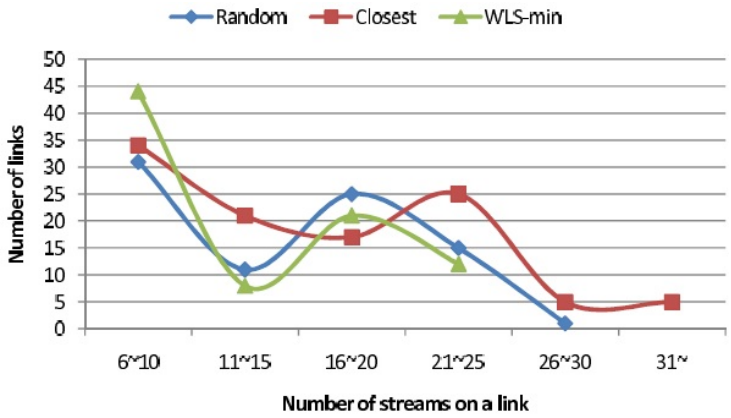

Fig. 8. The link stress distribution for the case of 20 concurrent swarming sessions

6. Conclusion

We conduct a research on the issue of optimal peer selection, which is critical in the mesh-based P2P networks. In this paper, we consider a new performance metric, the worst link stress (WLS), and propose a peer selection algorithm that produces the set of peers generating optimal WLS. Then we theoretically prove the optimality and discuss the complexity.

We have presented experimental results that show the advantages of our algorithm over the existing algorithms. It is remarkable that our algorithm outperforms existing popular algorithms, the random algorithm and the closest algorithm, in all performance metrics.

We now summarize the benefits of the proposed algorithm in the following aspects. the algorithm optimizes the congestion level of the bottleneck link, it enables to provide larger number of users and service sessions. Second, the congestion of the network is distributed. It improves not only the utilization of network but also the speed of content distribution. Furthermore, we also notice that if every swarming session uses the algorithm, the overall network load is well balanced, which ultimately benefits all sessions.

For future works, when describing the worst link stress, we assume a homogeneous network where all the links have same bandwidth, but if there is a big difference in the link bandwidths, our formulation and algorithm should consider this heterogeneity. We expect that our ongoing research will derive new and valuable insights in peer-to-peer networking.

References

[1] D.Coilean and D.O'mahony, "Accounting and Accountability in Content Distribution Architectures: A Survey," *Journal ACM Computing Survey*, vol. 47, no. 4, 2015. Article (CrossRef Link)

[2] R.Rejaie and N.Mahgarei, "On performance evaluation of swarm-based live peer-to-peer streaming applications," *Multimedia Systems*, vol. 20, pp. 415-427, 2014. Article (CrossRef Link)

[3] M.Zhao, P.Aditya, A.Chen, Y.Lin, B.Maggs, B.Wishon, and M.Ponec, "Peer-assisted content distribution in Akamai netsession," in *Proc. of the 2013 conference on Internet measurement conference*, pp. 31-42, 2013. Article (CrossRef Link)

[4] C.Y.Gho, H.S.Yeo,H.Lim, P.Hoong, and I.I.T.Tan, "A Comparative Study of Tree-based and Mesh-based Overlay P2P Media Streaming," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 8, no. 4, 2013. Article (CrossRef Link)

[5] K.Xu, X.Liu, Z.Ma, Y.Zhong, and W.Chen, "Exploring the policy selection of the P2P VoD system: A simulation-based research," *Peer-to-Peer Networking and Applications*, vol. 8, no. 3, pp.459-473, 2015. Article (CrossRef Link)

[6] PlanetLab website, http://www.planet-lab.org

[7] V.S.Pai, "Next-Generation CDN: A CB Perspective," *Advanced Content Delivery, Streaming, and Cloud Services*, 2014. Article (CrossRef Link)

[8] I.C.Yen, "Deploying Virtual Clusters through P2P-based Content Distribution," in *Proc. of 11th IEEE International Symposium on Network Computing and Applications*, pp.167-170, 2012. Article (CrossRef Link)

[9] BitTorrent website, http://www.bittorrent.com

[10] D.Kosti, R.Braud, C.Killian, E.Vandekieft, J.W.Anderson, A.C.Snoeren, and A.Vahdat, "Maintaining high bandwidth under dynamic network conditions," in *Proc. of USENIX Annual Technical Conference*, 2005. https://infoscience.epfl.ch/record/99653
D.Bickson, D.Malkhi, and D.Rabinowitz, "Efficient large scale content distribution," in *Proc. of the 6th Workshop on Distributed Data and Structures*, 2004. Article (CrossRef Link)

[11] X.Zheng, C.Cho, and Y.Xia, "Optimal Swarming for Massive Content Distribution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 6, 2010. Article (CrossRef Link)

[12] Y.Xia, S.Chen, C.Cho, and V.Korgaonkar, "Algorithms and Performance of Load Balancing with Multiple Hash Functions in Massive Content Distribution," *Computer Networks*, vol. 53, no. 1, pp.110-125, 2009. Article (CrossRef Link)

[13] PPLive website, http://www.pplive.com

[14] TVAnts website, http://www.tvants.com

[15] SopCast website, http://www.sopcast.com

[16] Mixed Media Grid (MiMeg) – Distringuishing Features and Functions, http://eprints.ncrm.ac.uk/821

[17] UUSee website, http://www.uusee.com

[18] PPStream website, http://www.ppstream.com

[19] MySee website, http://mysee.com

[20] Pando website, http://www.pando.com

[21] RedSwoosh website, http://www.akamai.com/redswoosh

[22] S.C.Han and Y.Xia, "Constructing an optimal server set in structured peer-to-peer network," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 1, 2007. Article (CrossRef Link)

[23] P.Felber, P.Kroopf, and E.Schiller, "Survey on Load Balancing in Peer-to-Peer Distributed Hash Tables," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, 2014.
Article (CrossRef Link)
[24] B.Eriksson, G.Dasarathy, P.Barford, and R.Nowak, "Toward the Practical Use of Network Tomography for Internet Topology Discovery," in *Proc. of IEEE INFOCOM*, 2010.
Article (CrossRef Link)
[25] X.Zhang and C.Phillips, "A Survey on Selective Routing Topology Inference Through Active Probing," *IEEE Communications Surveys and Tutorials*, vol. 14, no. 4, 2012.
Article (CrossRef Link)
[26] P.Sattari, C.Fragouli, and A.Markopoulou, "Active topology inference using network coding," *Physical Communication*, vol. 6, pp.142-163, 2012. Article (CrossRef Link)
[27] B.Yao, R.Viswanathan, F.Chang, and D.Waddington, "Topology inference in the presence of anonymous routers," in *Proc. of IEEE INFOCOM*, 2003. Article (CrossRef Link)
[28] J.Ni, H.Xie, S.Tatikonda, and Y.R.Yang, "Efficient and dynamic routing topology inference from end-to-end measurements," *IEEE/ACM Transactions on Networking*, vol. 18, no. 1, 2010.
Article (CrossRef Link)
[29] K.Calvert, M.Doar, and E.Zegura, "Modeling internet topology," *IEEE Communications Magazineg*, vol. 35, no. 6, 1997. Article (CrossRef Link)

Seung Chul Han is a head professor at the Dept. of Computer Science and Engineering, Myongji University, Seoul, Korea. He has his Ph.D. from Dept. Computer Science, University of Florida, M.S. from Purdue University. His primary research interests include P2P networks, graph theory, computer security, and OS.

Ki Won Nam is an assistant professor at the Department of Early Childhood Education at the Chungang University, Seoul, Korea. She has a Ph.D from the Chungang University in 2013, and M.S degree in 2001 and a B.S. degree in 1999 from Chungang University, Seoul, Korea. Her primary research iterests include Computer education, computational thinking, and physical computing.