Distributing Network Loads in Tree-based Content Distribution System

Seung Chul Han[1], Sungwook Chung[2], Kwang-Sik Lee[1], Hyunmin Park[1] and *Minho Shin[1]
[1] Department of Computer Engineering,
Myongji University, Seoul, Korea
[2] Department of Computer Engineering,
Changwon National University,Changwon, Korea
[dr.seungchul@gmail.com, swchung@changwon.ac.kr, hush@naver.com, hpark@mju.ac.kr,
shinminho@gmail.com]
*Correspondingauthor: Minho Shin

Received September 2, 2012; revised November 30, 2012; accepted January 3, 2012; published January 29, 2013

Abstract

Content distribution to a large number of concurrent clients stresses both server and network. While the server limitation can be circumvented by deploying server clusters, the network limitation is far less easy to cope with, due to the difficulty in measuring and balancing network load. In this paper, we use two useful network load metrics, the worst link stress (WLS) and the degree of interference (DOI), and formulate the problem as partitioning the clients into disjoint subsets subject to the server capacity constraint so that the WLS and the DOI are reduced for each session and also well balanced across the sessions. We present a network load-aware partition algorithm, which is practicable and effective in achieving the design goals. Through experiments on PlanetLab, we show that the proposed scheme has the remarkable advantages over existing schemes in reducing and balancing the network load. We expect the algorithm and performance metrics can be easily applied to various Internet applications, such as media streaming, multicast group member selection.

Keywords: Content distribution, load balance, optimization

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST)(2012-0005552,2011-0015187,2011-0003930,2010-0016970,2009-0069191)

http://dx.doi.org/10.3837/tiis.2013.01.002

Copyright ⓒ 2013 KSII

1. Introduction

Network load balancing is a methodology to distribute workload across network to achieve optimal resource utilization, maximize throughput, minimize congestion, and avoid bottleneck. It is being received significant attention because one of the distinct trends related to the Internet is that it is being used to distribute content on a more and more massive scale. Some servers that provide content have become tremendously popular and have millions of clients.[2]

The service requests from a large number of concurrent clients can lead to server and network that are frequently overloaded. While the server limitation can be circumvented by deploying server farms or server clusters, the network limitation is far less easy to cope with, due to the difficulty in determining the cause and location of congestion and in provisioning extra resources. If many concurrent connections pass through the same network links, they may overload the links. A likely consequence is poorly balanced network load, which hurts the overall service quality of all clients, as well as causes inefficient utilization of the network resources. Even when a well-provisioned server has been placed, the quality of service at the receiving side is not satisfactory because of the unbalanced network load; therefore, successful deployment of content distribution systems requires a scheme that is conducive to mitigate this problem.

In this paper, we consider the problem of client partitioning in content distribution system with the objective of balancing the network load. One can imagine that there exist a server containing the data and a large number of clients requesting all or portions of the data.[3] The server may not be able to serve all the clients simultaneously due to its insufficient resources, such as outbound bandwidth or computational power. The server could be allowed to partition the clients into disjoint subsets and serve each subset at a time. We call a subset of clients a *session*. The problem addressed in this paper is how to partition the clients into sessions subject to the server capacity constraint so that the network load is reduced for each session and also well balanced across the sessions. A balanced use of the network resources allows it to accommodate more clients and service channels, the most important concern for content providers.

For illustration of the problem, consider the toy example in Fig**. 1**. Suppose a server is connected with four clients and can serve at most two clients at a time. Suppose the links have roughly identical bandwidth, we wish to find the optimal partition that balances the network load. The content distribution paths should be a tree rooted at the server and covering all clients as **Fig. 1(a)**. Two possible partitions are {(A,B),(C,D)} and {(A,C),(B,D)} which are shown in **Fig. 1(b)** and **Fig. 1(c)**, respectively. Intuitively, it is easy to see that (c) is better than (b) from the point of view of network load balance.[4]

---

[2] YouTube is a prime example. Around a quarter of the Internet users on any given day are estimated to visit YouTube.
[3] It is not required that the clients request identical data, e.g., the same file.
[4] The reason is that IP multicast is not broadly available on the Internet. Some ISPs and networks support it internally, but it is usually not available across network boundaries as is needed forcontent distribution.

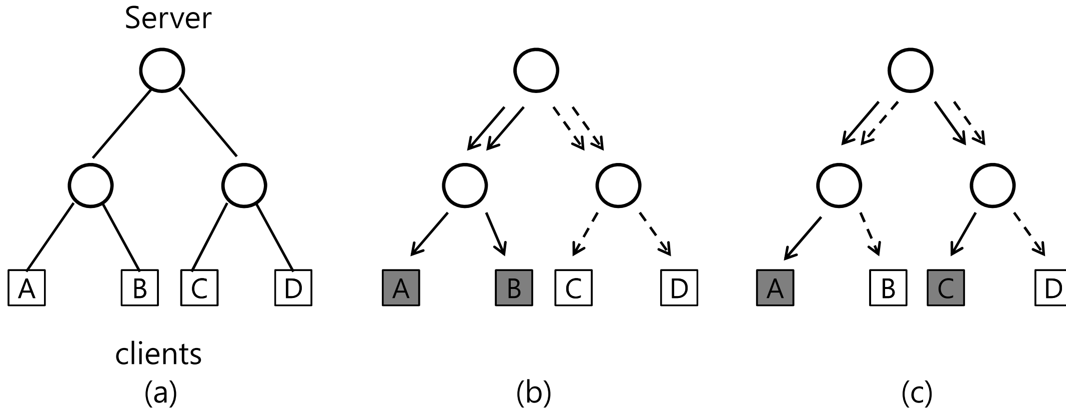


Fig. 1. A toy example of client partition problem. A server(root) distributes streaming contents to four clients (a). Partition of (c) achieves better load balance than (b).

In the client partition problem, we identify two important issues. First, how a server should partition its clients? With thousands of clients, finding the optimal solution is not trivial and the obtained performance can dramatically vary depending on it. When done properly, it provides improved utilization of the network capacity, a remedy for congestion, or gives the fast distribution. If done improperly, on the other hand, unbalanced load has made capacity shortage in the network a genuine possibility, which will become more serious as the number of clients increases.

The second issue is what to measure in the evaluation and comparison of different client partitions from the network perspective. While there have been a number of schemes developed, prior works mostly focus on user-centric performance metrics, such as the round-trip time (RTT), downloading time or packet loss rate of individual connection, but tend to ignore the network-centric objectives, such as the network load balance, congestion or bottleneck. The latter performance concerns are more important for the content provider because network congestion and load balance are critical when many data flows are transmitted simultaneously. In particular, the bandwidth sensitive services, such as media streaming, are characterized by network congestion and bandwidth usage. Therefore, it is necessary to consider network-centric metrics that is able to effectively measure the loads on the network resources.

This paper is fairly unique in emphasizing two network-centric metrics, the worst link stress (WLS)[1] and the degree of interference (DOI)[2], for the measurement of the network load. The WLS is the largest number of downloading streams on any link and directly related to the worst congestion level in the network. The DOI measures the total number of downloading streams seen by all network links. The DOI is useful to measure the total network resource usage, including the total bandwidth and the number of links used by the session.

In this paper, we present a greedy client partition algorithm that minimizes the WLS and DOI for each session, and balances them across sessions. The experimental results from the actual Internet testbed, *PlanetLab*[3], show that the proposed algorithm is simple yet effective in achieving the design goals.

The rest of the paper is organized as follows. In Section 2, we review previous works on related problems. In Section 3, we introduce two network-centric performance metrics. In Section 4, we present the network load balancing client partition algorithm and analyze its

Copyright ⓒ 2013 KSII

running time. We evaluate the algorithm and compare it with other algorithms through experiments on PlanetLab in Section 5. Finally, the conclusions are drawn in Section 6.

2. Related Works

The literature on content distribution is vast. We will mainly review the most relevant studies in content distribution system, which handles its clients in a variety of (usually ad-hoc) ways[5]. We can roughly classify the content distribution systems into three categories, which are likely to continue their coexistence.

In the first category, the infrastructure-based content distribution systems (e.g., Akamai [4][5]) and web caches generally gather neighbor nodes. The second category is tree-based end-system multicast (e.g., [6][7]) where all clients are typically served by the common tree root. The third category is mesh-based P2P systems, which typically employ the techniques of file striping and collaborative download (e.g., BitTorrent[8][9]). Their client management algorithms vary a lot. In PPStream[10][11] and FastReplica[12][13], client selection is essentially done randomly. Other systems employ a node ranking function. A node favors other nodes with high ranking. The ranking function may be the nodal load (CoBlitz[14]), the round-trip time (RTT) (ChunkCast[15]), the sending and/or receiving bandwidth to and from each node[8][9], and the degree of content overlap between the client and the server (Bullet [16][17]). One common practice is that a node initially selects some random nodes, but gradually probes other nodes and dynamically switches to those with better ranking over the course of data transmission.

While a lot of research efforts have been directed to the problem, important issues relative to the overall network congestion and load balance are not systematically covered by the literature. Network resource usage efficiency is always among the most important issues in networking systems. In fact, a main concern for content providers would be to optimize it in order to maximize the overall throughput. A scheme that is conducive to mitigating congestion or balancing network resource usage is valuable. From this point of view, our paper is fairly unique in emphasizing the network-centric performance metrics.

3. Preliminaries

3.1 Network Load Metrics

Suppose $S = \{s_1, ..., s_n\}$ is a subset of nodes in a tree, $\mathcal{T}$, and E is the set of edges used by the paths from root to the nodes in *S*, called *S*-paths.

Definition 1 The link stress of an edge *e* in $\mathcal{T}$, denoted by *LS(e)*, is the number of *S*-paths via *e*. Let *E* be the set of all edges in $\mathcal{T}$. The **worst link stress (WLS)** is defined as,

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

[5]Not all systems frame or handle the problem explicitly, but all should have at least an implicit client handling algorithm.

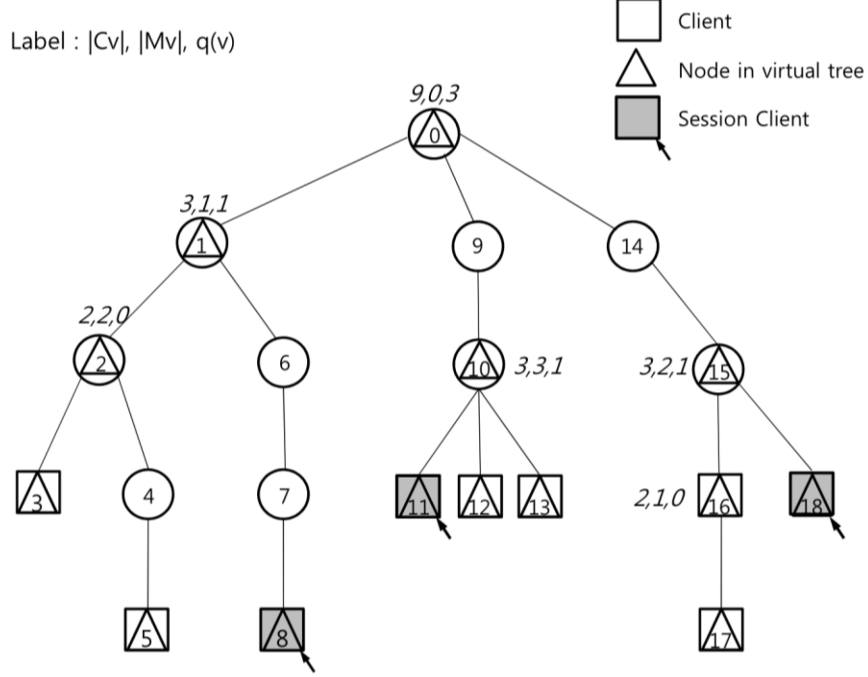


Fig. 2. An example of partitioning algorithm (Algorithm 1). The tree contains a server (node 0) and 9 clients (node 3,5,8,11,12,13,16,17,18). The grey nodes comprise a partition formed by Algorithm 1.

The root of the tree represents the content server, and the leaves and some intermediate nodes are clients. The WLS is the largest number of data streams on any link and is an indication of how well the load is balanced in the network and of how much connections to different clients interfere with each other at the bottleneck link, assuming the links have roughly identical bandwidth. It is both a measure of the burden placed of the point-to-point connections on the network and a measure of the quality of stream delivery. The reciprocal of the WLS tells how many times each stream can be increased without causing network congestion. It gives the maximum affordable number of subscribers and service channels. The issue of reducing link stress has also been considered in several other works[18][19][20]. We found that content distribution sessions following our client partition algorithm make a balanced use of the network bandwidth, which tends to cause the least interference to other clients.

Definition 2 The **degree of interference (DOI)** 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. 2**, if $s_1 = 3, s_2 = 5$ and $s_3 = 8$, then $DOI(s_1, s_2, s_3) = 3$ because the number of S-paths on link (1,2) is 2 and on link (0,1) is 3.

The DOI is closely related to the network resource usage and the degree of congestion in the network[21]. Suppose the nodes in S are clients, each receiving a data stream from the server (root). Let the base case for comparison be that every edge involved in the data

Copyright ⓒ 2013 KSII

transmission session sees exactly one stream. The DOI measures the difference between the total number of streams seen by all edges and the base case. From a slightly different viewpoint, suppose there is one unit of cost associated with a stream traversing an edge, and suppose every client receives one stream from the server. The DOI is the difference between the actual total cost and the cost of the base case where each edge sees exactly one stream. Therefore, the DOI is useful to measure the total network bandwidth usage by the point-to-point connections.

Also, as will be shown by our experimental results in Section 5, the DOI tends to be correlated with the WLS. Hence, minimizing the WLS usually also reduces the DOI. The reason is that the minimum DOI usually occurs when none of the links have many data streams on them, that is, none of the links are highly congested.

3.2 Build A Content Distribution Tree

In a content distribution system, the client management is usually centralized and the server manages the list of connecting clients. The algorithm proposed in this paper assumes that a tree topology is known for the server where the point-to-point connections established between the server and each client.

In many cases, Internet routers are reluctant or unable to provide internal network information such as topology, bandwidth, delay, or packet loss rates. Thus, there needs to be a way to discover unknown network states by using measurements available at the end hosts. [22] refers to a set of techniques to infer internal network properties using measurements available at the end hosts and the edge routers as “Internet tomography.” For instance, active or passive packet probes are commonly used with multicast or unicast traffic to obtain the internal network topology [23]. [24] introduces an interesting scheme where the entropy of inter-packet spacing is used to find the bottleneck.

Whereas there are several inference schemes, in practice, *traceroute* has been popular in finding the network topology. Even though [25] reports that the deterministic topology inference problem is NP-hard if there are anonymous routers that won't reveal their identity to *traceroute* probes, by using a heuristic algorithm in [25] we can deduce a reasonably accurate network topology. Thus, we assume *traceroute* and the algorithm in [25] is used when less than 10% of routers are anonymous. If more than 10% of the routers are anonymous, we can apply a statistical inference method described in [26] to discover mission links. Therefore, we rely on *traceroute* and the inference schemes discussed above to build a tree, T, rooted at content server for our algorithm.

4. Client Partition Algorithm

Suppose that there exists a server and a large number of clients requesting data. Due to the server capacity limitation (e.g., computational power, outbound bandwidth), server can service a subset of all the clients at a time. The problem is how to partition the clients into disjoint subsets (or sessions) subject to the server capacity constraints so that the WLS and the DOI are reduced for each session and also well balanced across the sessions.

As the preliminary step, we use *traceroute* and the topology inference schemes discussed in Section 3.2 to build a tree $\mathcal{T}$ rooted at content server. Then, we do a depth-first search (DFS) on $\mathcal{T}$ and label all the nodes in the order that they are first visited. Let us denote the label of node u in $\mathcal{T}$ by $I(u)$. When there is no confusion, we will use u and $I(u)$ interchangeably to denote node u. Let S_u be the subtree rooted at node u in $\mathcal{T}$.

4.1 Problem Statement

Given a tree $\mathcal{T}$ where the server is the root and all the leaf nodes and some internal nodes are clients, $C = \{c_1, \dots, c_n\}$. Suppose the server can serve a maximum of $\left\lceil \frac{n}{m} \right\rceil (m \leq n)$ clients simultaneously due to its capacity limitation. In order to efficiently serve all the clients, the server partitions the clients into *m* disjoint groups, $C_1, \dots, C_m$, and serves each group at a time. Our goal is to find partitions of the client set so as to reduce and balance the network loads across the sessions.

Definition 3 The lowest common ancestor (LCA) of a set of nodes $\boldsymbol{S} = \{\boldsymbol{s_1}, \dots, \boldsymbol{s_n}\}$, where $\boldsymbol{n} \geq \boldsymbol{2}$, in a tree is the deepest node in the tree that is a common ancestor of all nodes in $\boldsymbol{S}$. Itis denoted by $\boldsymbol{LCA(S)}$ or $\boldsymbol{LCA(s_1, \dots, s_n)}$.

Lemma 1 *Given a tree $\mathcal{T}$, do depth-first search(DFS) and label all the nodes in the order that they were first visited. Suppose u and v are two nodes in $\mathcal{T}$, and $I(u) < I(v)$. Then, the relation of (u,v) must be one of the following cases.*

$$LCA(u, v) = u, \text{ or } S_u \cap S_v = \emptyset$$

Proof The proof of Lemma 1 is rather easy due to the property of the DFS algorithm, we omit it for brevity.■

Lemma 2 *Suppose S is a set of nodes in an arbitrary tree. Let $W_1, \dots, W_n$ be a covering of S. That is, $W_i \subseteq S$ for $1 \leq i \leq n$, and $\cup_{i=1}^{n} W_i = S$. Then,*

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

Proof The proof of Lemma 2 is rather easy due to the definition of LCA, we omit it for brevity. ■

Lemma 3 *Suppose (s_1, s_2, s_3) is a sorted list of distinct nodes in $\mathcal{T}$ by their IDs. Then,*

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

Proof The relation of (s_1,s_2) and (s_2,s_3) must be one of the following cases by Lemma 1.

- Case 1: $LCA(s_1,s_2) = s_1$ and $LCA(s_2,s_3)=s_2$ *or* $LCA(s_1, s_2) = s_1$ and $\mathcal{T}_{s2} \cap \mathcal{T}_{s3} = \emptyset$
  $LCA(s_1, s_2, s_3) \rightarrow LCA(LCA(s_1, s_2), s_3) \rightarrow LCA(s_1, s_3)$
- Case 2: $\mathcal{T}_{s1} \cap \mathcal{T}_{s2} = \emptyset$ and $LCA(s_2, s_3) = s_2$
  By property of LCA, $LCA(s_1,s_2)=LCA(s_1,s_3)$. Therefore, $LCA(s_1, s_2, s_3) \rightarrow LCA(s_1, LCA(s_2, s_3)) \rightarrow LCA(s_1, s_2) \rightarrow LCA(s_1, s_3)$
- Case 3: $\mathcal{T}_{s1} \cap \mathcal{T}_{s2} = \emptyset$ and $\mathcal{T}_{s2} \cap \mathcal{T}_{s3} = \emptyset$
  Suppose $LCA(s_1,s_2,s_3) \neq LCA(s_1, s_3)$ and let $v=LCA(s_1,s_2,s_3)$. By the nature of DFS, $LCA(s_1,s_3)$ 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*, it contradicts that $LCA(s_1,s_3)$ is the lowest common ancestor of s_1 and s_3. ■

Copyright ⓒ 2013 KSII

Lemma 4 *Suppose* ($s_1, \dots ,s_n$) *is a sorted list of distinct nodes in* $\mathcal{T}$ *by their IDs. Then,*

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

Proof The proof will be based on induction on n. The case of n=2 is trivial. The base case n=3 is proven in Lemma 3. Let us make the induction hypothesis that the lemma is true for the list ($s_1, \dots ,s_l$), where $3 < l < n$. We will show it is true for ($s_1, \dots ,s_{l+1}$).

$$LCA(s_1, \dots, s_{l+1}) = LCA(s_1, s_{l+1})$$

Definition 4 Given a set, S, of two or more nodes in a tree, let us denote the set of LCAs of all subsets of S with two or more nodes by ***SLCA(S)***.

For example, in **Fig. 2**, if $S = \{3, 5, 8, 11\}$, then $SLCA(S) = \{0, 1, 2\}$.

Lemma 5 Suppose ($s_1, \dots ,s_n$) is a sorted list of distinct nodes in $\mathcal{T}$ by their IDs. Then,

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

Proof Let $S = \{s_1, \dots ,s_n\}$. By Lemma 2, we only need to focus on the LCAs of node pairs, because they form a covering for every subset of S. Thus, by definition, $SLCA(S) = \cup_{1 \le i < j \le n} LCA(s_i, s_j)$. The proof is based on induction. Let us define the sets Let $S^i = \{s_1, \dots ,s_i\}$, for $2 \le i \le$n. As the base case, the lemma is trivially true for S^2. We make the induction hypothesis that the lemma is true for S^l, where $2 \le l < n$. Then,

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

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

$$LCA(s_i, s_{l+1}) = LCA(s_i, s_l) \text{ or } LCA(s_l, s_{l+1}).$$

$LCA(s_i, s_{l+1})$ is either the same as $LCA(s_l, s_{l+1})$, or the same as $LCA(s_i, s_l)$, which is already in $SLCA(S^l)$. Hence,

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

Definition 5 Given a set S with two or more nodes in $\mathcal{T}$, the virtual tree $\hat{\mathcal{T}} = (\hat{V}, \hat{E})$ is formed by the nodes $\hat{V} = SLCA(S) \cup S$. For any two nodes $u, v \in \hat{V}$, $(u, v) \in \hat{E}$if u is the immediate ancestor of v in $\hat{V}$. That is, u is on the path from v to the root of $\mathcal{T}$, and there are no other nodes in $\hat{V}$ on the path segment from v to u. The root of $\hat{\mathcal{T}}$ is $LCA(S)$.

An example of the virtual tree $\hat{\mathcal{T}}$ is shown in **Fig. 2**. An efficient algorithm for constructing $\hat{\mathcal{T}}$ is based on Lemma 5. First, we sort the nodes in S in increasing order of the node ID. This takes $O(n \log n)$. Now, assume $S = (s_1, \dots ,s_n)$ is a sorted list. The set of nodes in $\hat{\mathcal{T}}$ is $\hat{V} = \{LCA(s_1, s_2), LCA(s_2, s_3), \dots, LCA(s_{n-1}, s_n)\}$. The root is $r = LCA(s_1, s_n)$. The edges in $\hat{E}$ can be identified by traversing the paths in $\mathcal{T}$ from each node, say v, in $\hat{V}$ toward the root of $\mathcal{T}$. In each step along the way, we inspect if the current node, say u, is in $\hat{V}$. If so, the

edge (u, v) is added to $\hat{E}$, and the path traversal originating from node v is stopped. It can be done in $O(n^2)$. This scheme will work on any general network provided the server has collected the topology information of the network.

Lemma 6 The running time for the construction of the virtual tree $\hat{\mathcal{T}}$ is $O(n^2)$.
Proof By Lemma 5. ■

4.1.1 Algorithm Description

Our client partition algorithm, Algorithm 1, takes two arguments as input, a set of clients and the size of session. Inside the *for* loop between line 4 and 30, it selects clients whose WLS is the minimum to build a session. The loop repeats until the client set becomes empty.

Let us consider the example shown in Fig. 2, where $\mathcal{T}$ has nine clients, C = {3,5,8,11,12,13,16,17,18}, indicated as squared nodes. The objective is to partition C into three sessions. The set of nodes in $\hat{\mathcal{T}}$ are triangled. Each node in $\hat{\mathcal{T}}$ is also labeled with $|C_v|$, $|M_v|$ and $q(v)$, where C_w is the set of clients in S_w, M_v is the set that contains all nodes in C for which v is the immediate ancestor in $\hat{\mathcal{T}}$ (i.e., in $SLCA(C)$) and that have no descendants in C, and $q(v)$ is the number of clients to be selected from the subtree of $\mathcal{T}$ rooted at v, S_v. The first session of clients is indicated with arrows. The first node to be visited is the root of $\hat{\mathcal{T}}$, node

Algorithm 1 CLIENT-PARTITION(C, m)

```
1:  Input:
      C = {s_1, ..., s_n}, client set
      m: size of session, 1 ≤ m ≤ n
2:  Output:
      G_1, ..., G_⌈n/m⌉: partitions of client set C
3:  for i = 0 to ⌈n/m⌉ do
4:    G_i ← ∅
5:    Construct SLCA(C) using Lemma 5
6:    Construct T̂, r ← root of T̂
7:    At each v ∈ SLCA(C), record |C_v|, M_v, κ(v) ≤ |M_v|
8:    Q ← {r}
9:    q(r) ← m
10:   while Q ≠ ∅ do
11:     u ← head[Q]
12:     if q(u) > κ(u) then
13:       G_i ← G_i ∪ M_u
14:       q(u) ← q(u) − κ(u)
15:     else if 0 < q(u) ≤ κ(u) then
16:       Add an arbitrary subset of M_v with |q(u)| elements to G_i
17:       q(u) ← 0
18:     end if
19:     if q(u) ≠ 0 then
20:       (q(v))_{v∈Π(u)} ← ALLOCATION(|Π(u)|, (|C_v|)_{v∈Π(v)}, q(u))
21:       for v ∈ Π(u) do
22:         if q(v) > 0 then
23:           Enqueue(Q, v)
24:         end if
25:       end for
26:     end if
27:     Dequeue(Q)
28:   end while
29:   C ← C − G_i
30: end for
31: return G_1, ..., G_⌈n/m⌉
```

Copyright ⓒ 2013 KSII

0, denoted by u. It is understood that three clients are to be selected from S_u. The motivation is that the nodes in M_u generate no more than one unit of link stress on the edges from u. The three clients to be selected are allocated to the subtrees of $\mathcal{T}$ rooted at the children of u in $\hat{\mathcal{T}}$, according to Algorithm 2. The resulting allocation is that 1, 1, and 1 clients are to be selected from the subtrees rooted at node 1, 9, and 14, respectively. Then, it runs recursively for each subtree, S_1, S_9, S_{14}. Finally, we get 3 partitions, G_1={8,11,18}, G_2={5,12,16}, G_3={3,13,17}. **Table 1** compares with the random algorithm in terms of the network load metrics.

Table 1. Comparison of partition algorithms

Scheme	Sessions	WLS	DOI
Algorithm 1	{8,11,18},{5,12,16},{3,13,17}	1,1,1	0,0,0
Random algorithm	{3,8,17},{5,16,18},{11,12,13}	2,2,3	1,2,4

The key idea of the *for* loop between line 4 and 30 is that the link stress of any edge (u, v), where u and v are nodes in a tree, $\mathcal{T}$, and u is the parent of v, is no less than that of any edge in the subtree S_v. Hence, the edge with the worst link stress must be connected to the root of $\mathcal{T}$. One possible algorithm for minimizing the WLS is as follows. Let W be the set of children of the root. For $w \in W$, let C_w be the set of clients in S_w. Suppose each w in W is labeled with $|C_w|$, i.e., the number of nodes in C_w. Let M_v be the set that contains all nodes in C for which v is the immediate ancestor in $\hat{\mathcal{T}}$ (i.e., in $SLCA(C)$), and that have no descendants in C. Then, we call Algorithm 2 with the arguments $l = |W|$, $(b_1, \dots, b_l)$ equal to the list of $|C_w|$'s and $q = k$. The returned list of numbers from Algorithm 2 is an optimal allocation, with respect to the WLS-minimization criterion, of the number of clients to be selected in each subtree S_w, for all $w \in W$. The actual algorithm builds on top of this basic idea. It minimizes the WLS recursively in the sense that it minimizes the WLS for every subtree rooted at every $v \in SLCA(C)$. The WLS for each such subtree, S_v, is defined over only the edges in S_v. That is, the WLS over S_v is $\max_{e \in E_v} LS(e)$, where E_v is the set of edges in S_v.

Line 5 can be accomplished in $O(n)$ by Lemma 5, line 6 is $O(n^2)$ by Lemma 6, and line 7 can be done in the process of building virtual tree (line 6). Let us now analyze the running time of the *while* loop. In each entry into the *while* loop, we only need to consider the running time of Algorithm 2 called in line 21. In line 16, the total number of operations by the completion of the *while* loop cannot be more than the size of Gi, which is at most n. Similarly, the ***for*** loop in line 22 through 26 cannot take more than $O(n)$ over all entries into the *while* loop, because this part simply visits all nodes in $SLCA(C)$ one at a time. Therefore, the running time for Algorithm 1 is $O(n^3)$.

4.1.2 Allocation Subroutine

Algorithm 2 takes three arguments, (l, $(b_1, \dots, b_l)$, q), where l and q are positive integers, and $(b_1, \dots, b_l)$ is a vector of l positive integers. It returns a vector of l positive integers.

Consider l sets, 1 through l, where $l \geq 1$. Set j contains b_j items ($1 \leq j \leq l$). Suppose we must choose q items from these l sets. The objective is to decide the number of items chosen from set j, denoted by c_j, for $1 \leq j \leq l$, so that $\sum_{j=1}^{l} c_j = q$ and that the largest c_j is minimized. That is, $\max_{1 \leq j \leq l} c_j$ is minimized. Any vector $(c_j)_{j=1}^{l}$ such that $\sum_{j=1}^{l} c_j = q$ will be called a feasible allocation. In our setting of client partition in a tree $\mathcal{T}$, as an example, the sets may

correspond to the subtrees rooted at each of the children of the root, and the items may be the nodes in C.

It is interesting to point out that Algorithm 2 maximizes the minimum c_j over all j. When each item is infinitely divisible, there is a classic notion of max-min allocation or max-min fairness[27]. We say a feasible $(c_j)_{j=1}^{l}$ is a max-min allocation if, for any other feasible allocation $(\bar{c}_j)_{j=1}^{l}$, $c_j < \bar{c}_j$ for some j implies that there exists some i with $c_j > c_i$ and $c_i > \bar{c}_i$. In other words, in the max-min allocation $(c_j)_{j=1}^{l}$, we cannot increase c_j without reducing some c_i, which is already smaller than or equal to c_j. This definition in fact applies to more general settings such as network bandwidth allocation to different connections subject to the capacity constraints at the network links. In our simple setting, it has an alternative characterization. Let us denote $c_{(j)}$ to be the jth smallest number in $(c_j)_{j=1}^{l}$. Repeated elements are ordered arbitrarily among themselves.

Algorithm 2 first finds the maximum of the min-max allocation. In line 6 of Algorithm 2, the list $(b_1,\ldots, b_l)$ is sorted and the set indices are redefined so that the list is in non-decreasing order. In an allocation $(c_j)_{j=1}^{l}$, if $c_j = b_j$, we say set j is *saturated*. Otherwise, it is said to be *non-saturated*. If $(c_j)_{j=1}^{l}$ were the min-max allocation for the infinitely divisible case, the non-saturated sets would all have the same allocation, which would be the maximum. In the discrete case, it is possible that some non-saturated sets are allocated fewer Than the maximum number of items. In lines 7 through 20, the algorithm computes the number t by trying the sequence $t = b_1$, $t = b_2,\ldots$. The eventual value of t is equal to the maximum of a min-max allocation if all sets are saturated or allnon-saturated sets have identical allocation. Otherwise, t isequal to one less than the maximum, and the number r is equal to the number of sets with the maximum number of items[6]. In lines 21 through 23,

Algorithm 2 ALLOCATION(l, $(b_1,\ldots,b_l)$, q)

```
1: input: l, number of sets; b_j, the number of items in set j, 1 ≤ j ≤ l; q > 0, the total number of items
   to be selected
2: output: (c_1,...,c_l). Each c_j is the number of items selected from set j, 1 ≤ j ≤ l, satisfying Σ_{j=1}^l c_j =
   q and max_{1≤j≤l} c_j is minimized.
3: if Σ_{j=1}^l b_j ≤ q then
4:     return (b_1,...,b_l)
5: end if
6: sort (b_j)_{j=1}^l and re-index the sets so that b_1 ≤ b_2 ≤ ... ≤ b_l
7: r ← q
8: t ← 0
9: for j = 1 to l do
10:     δ ← b_j − t
11:     if δ(l − j + 1) < r then
12:         t ← b_j
13:         r ← r − δ(l − j + 1)
14:     else
15:         δ ← ⌊r/(l − j + 1)⌋
16:         t ← t + δ
17:         r ← r − δ(l − j + 1)
18:         break the for loop
19:     end if
20: end for
21: for j = 1 to l do
22:     c_j ← min(b_j, t)
23: end for
24: for j = l down to l − r + 1 do
25:     c_j ← c_j + 1
26: end for
27: revert to the original set indices
28: return (c_1,...,c_l) {with respect to the original set indices}
```

Copyright ⓒ 2013 KSII

each set j selects min(b_j, t) items, for $1 \leq j \leq l$. In the case where t is not the maximum (indicated by $r > 0$), the last r sets each select an additional item (lines 24through 26). Therefore, the running time for Algorithm2 is $O(n^2)$.

5. Evaluation

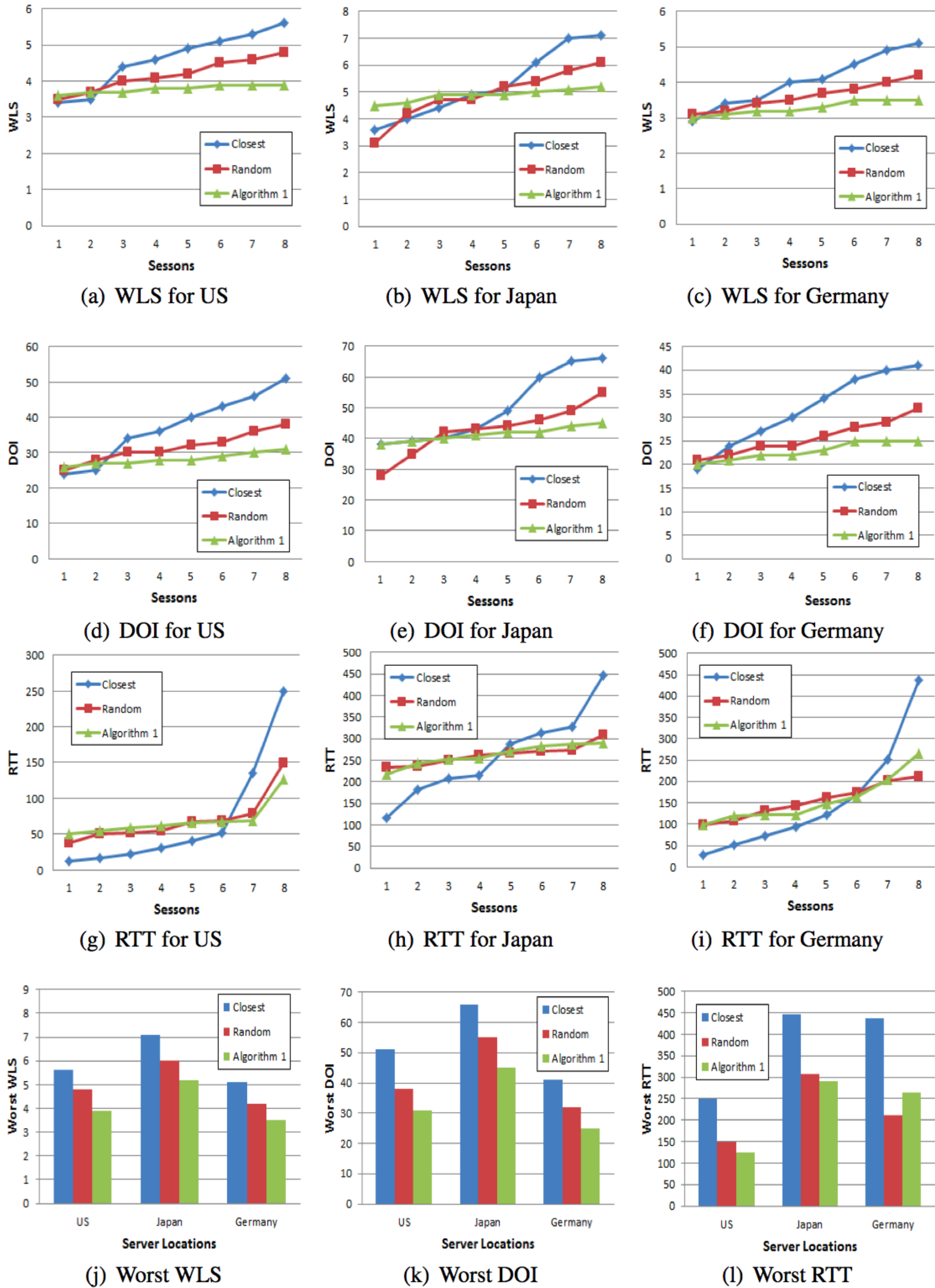
In this section, we present experimental results demonstrating the benefits of Algorithm 1 on an actual global Internet testbed, *PlanetLab*[3]. The PlanetLab network currently consists of 1133 nodes at 515 sites. We found 614 nodes available across the Internet at the time of our experiments. We collected the *traceroute* data between the PlanetLab nodes, which was used to calculate the WLS, DOI, physical paths, and round-trip time (RTT). More specifically, we randomly selected 3 nodes as content servers from the US, Europe, and Asia (*planet2.cs.rochester.edu,planet01.hhi.fraunhofer.de, planet0.jaist.ac.jp*), respectively. In each experiment, 64 nodes (about 10% of all nodes) were randomly chosen as the client nodes. Then, the server partitioned the client nodes into 8 sessions and served each session at a time. While serving each session, we counted the number of streams on each physical link for measuring the WLS and the DOI. For each server, the experiment was repeated with different client sets and we present the average of the results obtained.

We compare our algorithm with (1) the *Random* scheme where each session, the server chooses $\left\lceil \frac{n}{m} \right\rceil (m \leq n)$ clients uniformly at random from the client pool toconstruct the session, (2) the *Closest* scheme, where the server groups clients by the RTT from the server. These two schemes are the most typical strategies in the related works[4][15].

Fig. 3(a)-3(c) compare the distribution of the WLS of the sessions among the Closest scheme, the Random scheme, and Algorithm 1. For each run, we sort the measured WLS for the sessions in increasing order. The more gradual the slope is, the better WLSs are balanced among sessions. In all cases, Algorithm 1 yields the most balanced WLSs among the sessions. This is because Algorithm 1 strategically partitions clients to minimize the WLS. This suggests that, if we identify "load" with the number of streams on a link, Algorithm 1 is the best from the load balancing point of view. Throughout the experiments, the Closest scheme yields the worst balanced loads among sessions.

Fig. 3(d)-3(f) plots the distribution of the DOI of the sessions. For each run, we sort the measured DOI for the sessions in increasing order. We see that Algorithm 1 yields the most uniformly distributed DOI for all servers, leading to very well balanced bandwidth usage. The saving in network bandwidth is substantial, too. Moreover, the average DOI of all sessions is much lower than the other schemes. This indicates that WLS is a good metric for the network performance.

Fig. 3(g)-3(i) plots the distribution of the average RTTs of the sessions. For each run, we sort the average RTTs in increasing order. We see that Algorithm 1 produces the most balanced results for all servers. For example, in **Fig. 3(h)**, the maximum difference of the RTT between sessions is about 300ms for the Closest scheme, while Algorithm 1 yields only 170ms.



(a) WLS for US (b) WLS for Japan (c) WLS for Germany

(d) DOI for US (e) DOI for Japan (f) DOI for Germany

(g) RTT for US (h) RTT for Japan (i) RTT for Germany

(j) Worst WLS (k) Worst DOI (l) Worst RTT

Fig. 3. WLS, DOI, and RTT are compared among the Closest, Random, and Algorithm 1 for three different servers. It is clear that Algorithm 1 outperforms the other methods in balancing WLS and DOI, and it also balances RTT comparable to Random.

Fig. 3(j)-3(l) compares the worst WLS, the worst DOI, and the worst RTT of each method for each server, respectively. In **Fig. 3(j)**, Algorithm 1 yields the smallest worst WLSs compared to other methods in all servers. Improvement of up to 31% was observed. Likewise, **Fig. 3(k)** shows that Algorithm 1 outperforms the Closest and the Random schemes by 20 and 10 DOIs on average, respectively. Lastly, **Fig. 3(l)** compares the RTT of each method. In all servers, Algorithm 1 yields significantly smaller RTTs than the Closest scheme. Algorithm 1 also outperformed the Random scheme in the US and Japan servers, while the Random scheme yields less RTT than Algorithm 1 in the Germany server. This is because Random scheme may balance RTT very well depending on the RTT distribution among clients. However, as shown in **Fig. 3**, the balanced RTT not necessarily yields balanced network performance.

6. Conclusions

In this paper, we make an in-depth investigation on the issue of distributing network loads, which is a fundamental problem in massive content distribution systems. We introduce two useful network load metrics, WLS and DOI, and formulate the problem as partitioning clients into disjoint subsets according to the WLS criterion. Then, we present a partition algorithm in which the network loads of each session is reduced and also well-balanced across the sessions.

Using simulations on PlanetLab, we show that our scheme is practicable and effective in achieving the design goals. It is noticeable that our algorithm performs significantly better than the random algorithm and the closest algorithm, which are the most commonly used schemes.

Due to the nature of the problem, our problem formulation, algorithm, and performance metrics in this paper are relevant and can be applied to various Internet applications, including IPTV, VoD, cloud computing, P2P networks, or edge-mapping in content distribution networks.

References

[1] S. C. Han and Y. Xia, "Optimal node selection algorithm for parallel download in overlay content," *Computer Networks,* vol. 53, pp. 1480-1496, 2009. Article (CrossRef Link)

[2] S. C. Han and Y. Xia, "Constructing an optimal server set in structured peer-to-peer network," *IEEE JSAC,* vol. 25, pp. 170-178, 2007. Article (CrossRef Link)

[3] "PlanetLab," [Online]. Available: http://www.planet-lab.org.

[4] "Akamai," [Online]. Available: http://www.akamai.com.

[5] E. Nygren, R. K. Sitaraman and J. Sun, "The Akamai Network: A platform for high-performance Internet applications," *ACM SIGOPS Operating Systems Review,* vol. 44, 2010. Article (CrossRef Link)

[6] C. X. Zheng and Y. Xia, "Optimal swarming for massive content distribution," *IEEE Transactions on Parallel and Distributed Systems,* vol. 21, no. 6, 2010. Article (CrossRef Link)

[7] 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, 2009. Article (CrossRef Link)

[8] "BitTorrent," [Online]. Available: http://www.bittorrent.com.

[9] J. Pouwelse, P. Garbacki, D. Epema and H. Sips, "The BitTorrent P2P file-sharing system: Measurements and analysis," *Peer-to-Peer Systems IV,* vol. 3640, 2005. Article (CrossRef Link)

[10] "PPStream," [Online]. Available: http://www.ppstream.com.

[11] J. Jia, C. Li and C. Chens, "Characterizing PPStream across Internet," in *NPC Workhops*, 2007. Article (CrossRef Link)

[12] L. Cherkasova and J. Lee, "FastReplica: Efficient large file distribution within content delivery networks," in *USITS*, 2003.

[13] J. Lee and G. Veciana, "On application-level load balancing in FastReplica," *Computer Communications,* vol. 30, 2007. Article (CrossRef Link)

[14] K. Park and V. S. Pai, "Scale and performance in the CoBlitz largefile distribution service," in *USENIX/ACM NSDI*, 2006.

[15] B. Chun, P. Wu, H. Weatherspoon and J. Kubiatowicz, "ChunkCast: An anycast service for large content distribution," in *USENIX IPTPS*, 2006.

[16] D. Kosti, A. Rodriguez, J. Albrecht and A. Vahdat, "Bullet: high bandwidth data dissemination using an overlay mesh," in *SOSP*, 2003. Article (CrossRef Link)

[17] D. Kostic, A. Rodriguez, J. Albrecht and A. Vahdat, "Bullet: high bandwidth data dissemination using an overlay mesh," *ACM SIGOPS Operating Systems Review,* vol. 37, 2003. Article (CrossRef Link)

[18] M. Castro, M. Jones, H. Wang and A. Wolman, "An evaluation of scalable application-level multicast built using peer-to-peer overlays," in *INFOCOM*, 2003. Article (CrossRef Link)

[19] S. Ratnasamy, M. Handley, R. M. Karp and S. Shenker, "Application-level multicast using content-addressable networks," *Networked Group Communications,* vol. 2233, 2001.

[20] Y. Chu, "A case for end system multicast," *IEEE JSAC,* vol. 20, 2002. Article (CrossRef Link)

[21] S. C. Han and Y. Xia, "Network load-aware content distribution in overlay networks," *Computer Communications,* vol. 32, 2009. Article (CrossRef Link)

[22] N. Knight, H. X. Nguyen and M. Roughan, "The Internet Topology Zoo," *IEEE JSAC,* vol. 29, Oct 2011. Article (CrossRef Link)

[23] N. Duffield and F. Presti, "Network tomography from measured end-to-end delay covariance," *IEEE Transactions on Networking,* vol. 12, 2004. Article (CrossRef Link)

[24] I. Bazzi, D. Katabi and X. Yang, "A passive approach for detecting shared bottlenecks," in *ICCCN*, 2002.

[25] F. Chang, B. Yao, R. Viswanathan and D. Waddington, "Topology inference in the presence of anonymous routers," in *INFOCOM*, 2003. Article (CrossRef Link)

[26] R. Castro, M. Coates and R. Nowak, "Maximum likelihood network topology identification from edge-based unicast measurements," in *SIGMETRICS*, 2002. Article (CrossRef Link)

[27] D. Bertsekas and R. Gallager, Data Networks, Prentice-Hall, 1995.

Copyright ⓒ 2013 KSII

Seung Chul Han is an assistant professor at the Computer Engineering department at the Myongji University, Seoul, Korea. He has a PhD degree in Computer Science from the University of Florida in 2007, an MS degree in 2003 from Purdue University, and a BS degree from Sogang University, Seoul, Korea. His primary research interests include networks, security, and Android-linux systems.

Sungwook Chung is an assistant professor in the Department of Computer Engineering at Changwon National University, Korea. He received M.S. and Ph.D degrees in Computer and Information Science and Engineering from the Universityof Florida, USA, in 2005 and 2010, respectively. He worked for smart IPTV developments at the KT Central R&D Lab. in Korea from 2010 to 2012. His research interests include distributed multimedia systmes and community area networks.

Kwang-Sik Lee received the BS degree from the Hanyang University and the MS degree from Yonsei University, Seoul, Korea. He is currently working toward the Ph.D degree in Computer Engineering at the Myongji University. His primary research interests include network protocols and security.

Hyunmin Park received the BS degree in electronics engineering from the Seoul National University, Seoul, Korea, in 1985; and the MS and PhD degrees in computer engineering from North Carolina State University, Raleigh, in 1988 and 1995. He is a professor in the Department of Computer Engineering at Myongji University in South Korea. His research interests include computer network architecture, routing protocols, cloud computing and network security.

Minho Shin is an Assistant Professor in the Department of Computer Engineering at Myongji University, Korea. His research interests include Wireless Networks, Mobile Computing, and Network Security. He received a BS in Computer Science from the Seoul National University, Korea, an MS and Ph.D in Computer Science from the University of Maryland, College Park.