# A Simple and Efficient One-to-Many Large File Distribution Method Exploiting Asynchronous Joins

Soojeon Lee, Kyungran Kang, Dongman Lee, and Jae-Hoon Kim

**In this paper, we suggest a simple and efficient multiple-forwarder-based file distribution method which can work with a tree-based application layer multicast. Existing multiple-forwarder approaches require high control overhead. The proposed method exploits the assumption that receivers join a session at different times. In tree-based application layer multicast, a set of data packets is delivered from its parent after a receiver has joined but before the next receiver joins without overlapping that of other receivers. The proposed method selects forwarders from among the preceding receivers and the forwarder forwards data packets from the non-overlapping data packet set. Three variations of forwarder selection algorithms are proposed. The impact of the proposed algorithms is evaluated using numerical analysis. A performance evaluation using PlanetLab, a global area overlay testbed, shows that the proposed method enhances throughput while maintaining the data packet duplication ratio and control overhead significantly lower than the existing method, Bullet.**

**Keywords: Application layer multicast, P2P, file distribution, asynchronous joins.**

Manuscript received Apr. 18, 2006; revised June 29, 2006.

Soojeon Lee (phone: + 82 42 860 1079, email: soojeonlee@etri.re.kr) and Jae-Hoon Kim (email: jhkim@etri.re.kr) are with Radio & Broadcasting Research Division, ETRI, Daejeon, Korea.

Kyungran Kang (email: korykang@ajou.ac.kr) is with College of Information Technology, Ajou University, Suwon, Korea.

Dongman Lee (email: dlee@icu.ac.kr) is with School of Engineering, Information and Communications University, Daejeon, Korea.

## I. Introduction

Application layer multicast has emerged as an efficient and feasible delivery method for one-to-many file distribution. Most application layer multicast methods [1]-[5] deliver data using a delivery tree similar to an IP multicast routing tree. Using a delivery tree can save bandwidth and reduce the file reception delay compared with multiple unicasts. However, it has a limitation. Receivers cannot fully utilize network resources since they cannot have a higher data reception rate than their ancestors in a given delivery tree. To overcome this limitation of the tree-based approaches, multiple delivery path methods have been proposed such as Bullet [6], SplitStream [7], informed delivery [8], and the redundant mesh method [9]. They exploit multiple delivery paths to allow a receiver to receive data packets from multiple forwarders. Receivers can exploit available network resources more and thus reduce file reception delay.

SplitStream divides original data into multiple strands, which are forwarded along a forest of interior-node-disjoint subtrees. It is assumed that receivers join a session at the same time, which is not obvious to globally distributed Internet users. The redundant mesh method requires the intermediate end systems in a delivery tree to re-encode data to avoid duplicate packets and to enhance reliability. It does not require additional control message overhead but it incurs processing overhead at the intermediate end systems. Bullet and informed delivery propose multiple forwarder selection algorithms to enhance the performance of legacy tree-based application layer multicast methods. It is assumed that a source encodes data using an erasure-resilient code. Receivers can rebuild an original file out of necessary data packets regardless of their reception order. By increasing the number of data sources and surpassing the constraint of in-order delivery, they support higher throughput compared with the tree-

based methods. However, they have noticeable control overheads. First of all, more than one forwarder may forward the same data packet. To avoid duplications, all the receivers have to exchange extra data packet reception state messages. Second, the size of a data packet reception state message increases proportionally to the number of data packets. To reduce the size of a state message, several reception state summary methods such as bloom filter [6], [8] have been proposed. However, these methods can cause a false positive and result in data packet duplication. Third, in order to find appropriate forwarders more quickly, data packet reception state messages should be exchanged as frequently as possible.

In this paper, we propose a simple and efficient multiple-forwarder-based file distribution method that significantly reduces the overhead costs mentioned above. The proposed method exploits an assumption that receivers join a session at different times. Analysis of data on multicast session join patterns [10] supports our assumption. Thus, preceding receivers always have data packets that a succeeding receiver has not received from its parent. Furthermore, the data packets delivered from a parent after the receiver joined and before the next receiver joins do not overlap with those of other receivers. By exploiting this, a receiver, as a forwarder, is able to forward data packets to its succeeding receivers without duplication. A list of the all receivers in a session is managed by a source in the order of the sequence number of the first data packet each receiver received from its parent node.

We propose three algorithms for forwarder selection: moving backward linearly, moving backward with skipping, and moving backward with probing. A set of preceding receivers is nominated by the source and a receiver selects its forwarders out of the nominees according to certain criteria. The most basic criterion is the reverse order of the sequence numbers of the nominees' first data packets received from their parents. Secondly, some nominees are skipped and the size of the data packet set is extended to cover the data packet set of skipped nominees. It reduces the overhead of a negotiation with the nominees by decreasing the frequency of message exchanges. Thirdly, to pick out the best forwarders, the nominees are probed. In this study, the efficiency of the selection algorithms has been analyzed numerically. The performance of the proposed method has also been evaluated on the global testbed PlanetLab [11] spanning over many countries. The evaluation results show that the proposed method significantly enhances throughput compared with Bullet and has a far lower data packet duplication ratio and control overhead.

The rest of this paper is organized as follows. Section II describes related work and section III presents the detailed algorithm of the proposed method. The performance evaluation is presented in section IV and we conclude our work in section V with some thoughts on future works.

## II. Related Work

Bullet [6], informed delivery [8], SplitStream [7], and redundant mesh [9] address the problems of the tree-based application layer multicast methods. First, bandwidth is monotonically decreasing along the tree from the root to the leaf nodes. Thus, expensive bandwidth probing is continuously needed to find a better parent. Second, since each receiver is connected to only one parent, it is vulnerable to node and/or link failures.

Bullet builds a high-bandwidth mesh on top of an arbitrary overlay tree in a peer-to-peer manner. Each receiver locates proper peers containing the required data packets. Thus, each receiver needs to exchange a summary of its own data packet reception state with others but this results in control overhead. Furthermore, a summary cannot express an exact data packet reception state, thus duplicate data packets may be received.

Informed delivery attempts to optimize the delivery throughput across high-bandwidth networks. It focuses on collaborative transfers between peers. By using the digital fountain encoding approach, each host can reconstruct original content with low overhead. Each host approximates reconciliation of different sets of symbols between collaborating peers and tries to reduce the data packet duplication ratio. In addition, each receiver continuously exchanges a summary ticket to be aware of the reception state of other receivers. However, these efforts do not exclude duplicate data packets completely and they incur additional control overhead.

SplitStream encodes data into multiple strands and those strands are delivered across a forest of interior-node-disjoint multiple trees. The goal is to share the distribution overhead among as many participants as possible. In addition, it aims to provide reliable delivery by way of multiple delivery paths. To build the interior-node-disjoint multiple trees, it assumes that the participants join the session at the same time. However, users are distributed over the global network and they are not likely to join the session simultaneously. SplitStream is not general enough to be applied to normal data distribution cases.

The redundant mesh method suggests building a forest. Any given receiver belongs to multiple trees and parent nodes will differ from tree to tree. It avoids duplicate data packets and enhances reliability by data encoding and decoding at the intermediate nodes in the delivery tree. It can achieve high data redundancy but the computing delay incurred by encoding and decoding results in high end-to-end delay.

Peer-to-peer methods such as BitTorrent [12] propose data acquisition from multiple forwarders. BitTorrent suggests

dividing a file into multiple pieces so that a receiver can acquire each piece from different forwarders and finish file reception in a short time. It is quite similar to multiple forwarder-based application layer multicast methods but the granularity of data is much coarser. The efficiency achieved by the multiple forwarders will be lower than that of the application layer multicast methods.

In summary, despite their advantages, the multiple-forwarder-based file distribution methods realized so far incur considerable control overheads to maintain delivery topology and/or to exchange data packet reception states.

## III. Proposed Method

### 1. Overview

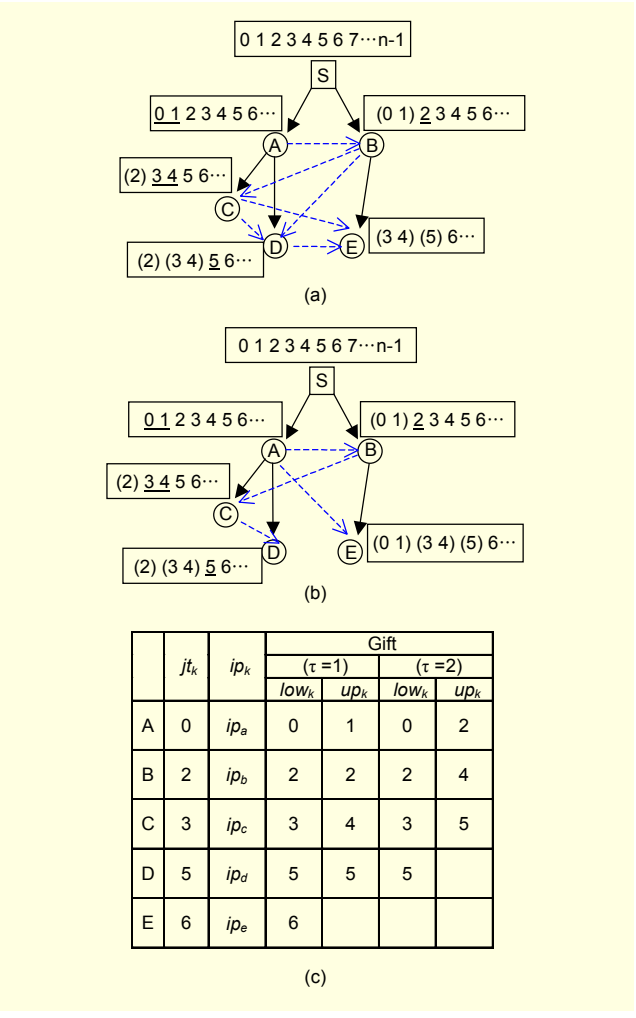
As mentioned in the previous section, multiple forwarder approaches require high control overheads. The proposed method takes advantage of the fact that receivers join a session at intervals. In a delivery tree, the set of data delivered from a parent to each receiver since a receiver has joined until the next receiver joins does not overlap that of other receivers. By using this property, a receiver can forward data packets to the late-comers without duplicates.

We assume that a session is initiated by a source and the source is well-known to the would-be receivers. The source encodes a file and generates a series of data packets by using tornado codes [13]. Data packets are propagated through a delivery tree. Suppose that the number of data packets required for transferring a file is $m$. By encoding $m$ data packets, $n$ different packets are to be generated and we call $n$ an *encoding interval*. The source starts transmission with the first data packet in an encoding interval. While transmitting, the sequence number of transmitted data packet increases one by one. When the $n$th data packet is transmitted, the source repeats data transmission from the first data packet. A receiver can rebuild the original file by receiving a sufficient number of different data packets in the encoding interval.

Figure 1 illustrates how data packets are acquired by a receiver with the proposed method. The circular nodes represent receivers and the rectangular node represents a source. At first, a delivery tree rooted at a source is built and data packets flow along the tree. From the view point of a target receiver, additional forwarders other than its parent node are selected from the other receivers in the tree. The solid arrows represent parent-child relations established by the application layer multicast tree construction method, which is beyond the scope of this paper. The dotted arrows represent forwarder-sink relations established by the proposed method. Note that for a given receiver, another receiver that appoints it as a forwarder is called a *sink*. In Fig. 1(a), for example, receiver $B$ is a child of the source and the parent of $E$ in the tree. Simultaneously, it is the forwarder of receiver $C$ and $D$, and the sink of receiver $A$.

A forwarder forwards a specific data packet out of the data packet set received from its parent. We call a specific set of data packets a *gift*. In Fig. 1(a) and (b), the box beside a node represents a set of the data packets delivered to the receiver. The numbers underlined by a solid line represent the gift of the receiver itself. The plain numbers represent the data packet delivered from the parent node. Those enclosed in parentheses represent the gift forwarded from the forwarders. To avoid data packet duplications, a receiver as a forwarder forwards its sinks data packets, which are not delivered from its forwarders but from its parent. A receiver will reselect its forwarders when it finishes receiving the gift of the forwarders. Until rebuilding the original file, a receiver keeps selecting new forwarders if available.

As Fig. 1(c) shows, a source maintains a list of current




| | $jt_k$ | $ip_k$ | Gift ($\tau$ =1) | | Gift ($\tau$ =2) | |
|---|---|---|---|---|---|---|
| | | | $low_k$ | $up_k$ | $low_k$ | $up_k$ |
| A | 0 | $ip_a$ | 0 | 1 | 0 | 2 |
| B | 2 | $ip_b$ | 2 | 2 | 2 | 4 |
| C | 3 | $ip_c$ | 3 | 4 | 3 | 5 |
| D | 5 | $ip_d$ | 5 | 5 | 5 | |
| E | 6 | $ip_e$ | 6 | | | |

(c)

Fig. 1. Examples of data delivery paths determined by the proposed method (when a forwarder has at most two sinks).

receivers in a session. For any integer $k$ larger than 1, the $k$th entry in the list consists of a tuple $<jt_k, ip_k, low_k, up_k>$, where $jt_k$ is the join time of the $k$th receiver and $ip_k$ is the access address of the receiver like an IP address. The sequence number of the receiver's first data packet received from its parent and set by a report from the $k$th receiver is denoted by $low_k$; $low_{k+1}$ is that of the next receiver and set by the $(k+1)$th receiver's report; while $low_k$ and $up_k$ are the lower and upper boundaries of the gift of the $k$th member, respectively. The details of how they are determined are explained with the forwarder selection algorithms later in this section.

## 2. Control Message Exchanges

Figure 2 shows the example procedure of control message exchanges from the viewpoint of a receiver. When a receiver joins a file distribution session, it sends a *join_request* message to the source. Then the source creates an entry for the receiver in the receiver list and sends a *join_reply* message back to the receiver. If the receiver receives the first data packet from its parent, it notifies the sequence number to the source by sending a *first_packet* message so that the source determines the range of a gift of the preceding receiver.

To request the entries of preceding receivers who will be recruited for the forwarders, the receiver sends *preceding_request* message to the source by specifying its address (*ip*), the maximum number of entries (*offset*), and the sequence number (*seq*). The value of an *offset* will be determined by the maximum number of simultaneous forwarders, which will be limited by incoming link bandwidth. It is hard to determine an exact number but we can have a rough number according to the system configuration. As long as the number of simultaneous forwarders does not exceed the capacity of an incoming link, the receiver recruits new forwarders. The receiver can send a *preceding_request* multiple times and the *sequence_number* shows the sequence of the requests.

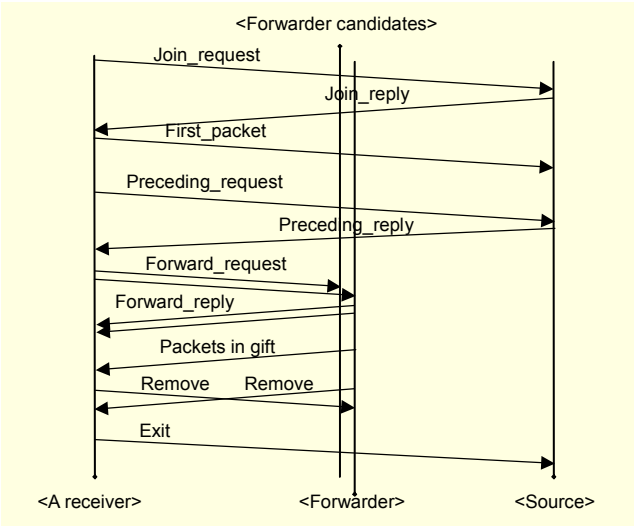



Fig. 2. Example of control message exchanges from joining to leaving.

The source replies with the access address and range of the gift of the preceding receivers using a *preceding_reply* message. For example, in Fig. 1(c), receiver $D$ sends a *preceding_request* message with $ip = ip_d$, *offset* = 2, and *seq*=0. The source will provide the information of receiver $B$ and $C$.

A receiver requests the data packets in the gift to each preceding receiver by sending a *forward_request* message. The request will be rejected if the requested preceding receiver already has as many sinks as its outgoing link bandwidth allows. The maximum number of simultaneous sinks will be determined according to the system configuration like that of forwarders. The preceding receivers reply whether to accept the request by responding with a *forward_reply* message.

Upon acceptance of the request, the forwarder forwards packets in its gift. As soon as the gift of a forwarder runs out, it means that the connection is broken and a new forwarder needs to be sought. From the list of the forwarder candidates in the *preceding_reply* message, the receiver selects forwarders using one of the forwarder selection algorithms explained later in this section. If the list is exhausted, the receiver re-requests the information of additional preceding receivers from the source by sending a *preceding_request* message with increased *seq*.

When a sink no longer needs to receive data packets from a forwarder, it sends a *remove* message to the forwarder so that the forwarder does not transfer anymore. When a receiver leaves a session, it sends an *exit* message to the source so that the corresponding entry in the receiver list can be deleted. It also sends a *remove* messages to its forwarders and sinks to be disconnected from them.

## 3. Forwarder Selection Algorithms

We have three variations of forwarder selection algorithms. Usually, a forwarder is selected from the preceding receivers that have joined the session earlier than a target receiver. The preceding receiver information is acquired via a *preceding_reply* message from the source and some of the preceding receivers are tried according to the selection algorithm. As Fig. 3 shows, we propose three algorithms according to selection criteria: moving backward linearly, moving backward with skipping, and moving backward with probing. The dark rectangle represents a gift from forwarders and the circled number represents a selection order. For simplicity, we assume that an encoding interval is sufficiently large that all data packets have different sequence numbers.

### A. Moving Backward Linearly Algorithm

The moving backward linearly (MBL) algorithm is the basic forwarder selection algorithm. A forwarder is selected from the preceding receivers in the reverse order of the first received data packet's sequence number, that is, *low* in the receiver list. Note that the sequence number of each data packet is unique in a session. In this algorithm, $up_k = low_{k+1}$. Thus a gift of the $(k-1)$th receiver is set to $[low_{k-1}, low_k-1]$. It means a set of the data packets that a receiver receives until the next receiver receives the first data packet. Figure 3(a) illustrates the procedure. The $k$th receiver selects forwarders sequentially by moving backward from the $(k-1)$th receiver to $(k-3)$th receivers.

There are two practical reasons for selecting the forwarders in the reverse order from the most recent preceding member. First, it reduces the data packet duplication ratio. We assume that each receiver joins a session at different times and the gifts of the preceding receivers do not overlap within an encoding interval. Therefore, the recent preceding receivers are more likely to belong to the same encoding interval compared with the old preceding members and their gifts will not overlap with one another. Second, from the viewpoint of the target receiver, it is more probable that the recent preceding members will remain in the session, compared with the old preceding members, while the target receiver is receiving data packets from the preceding members.

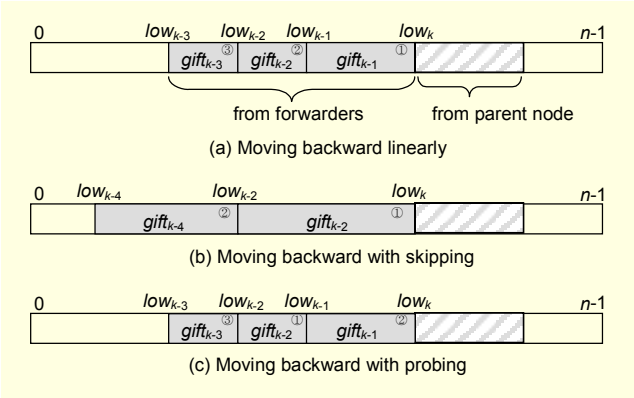



Fig. 3. Three forwarder selection algorithms.

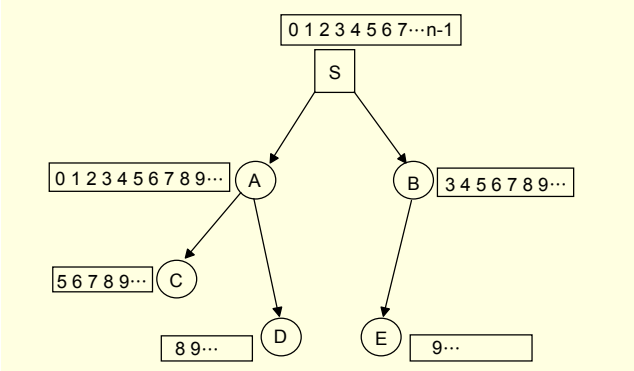



Fig. 4. Example of delivery tree and a snapshot of the packet reception state.

Figure 4 shows an example of the delivery tree. The node represents the receiver of the session. The source is S and $A$, $B$, $C$, $D$, and $E$ are the receivers. The rectangle beside a node represents the sequence numbers of the data packets that are delivered along the delivery tree. Receiver $B$ selects receiver $A$ as its forwarder. Receiver $D$ may select receivers $C$ and $B$ as its forwarders. Receiver E selects its forwarders out of three candidates: receivers $D$, $C$, and $A$. With the proposed method, late joiners have more forwarder candidates and more chances to reduce the delay to acquire the required size of data.

### B. Moving Backward with Skipping Algorithm

The moving backward with skipping (MBS) algorithm is a variation of the moving backward algorithm in order to support highly dense sessions where receivers join a session with a short join time interval. A short join time interval results in a small gift. It results in frequent forwarder selections and incurs control overhead to the receiver. To reduce the control overhead and increase throughput in a highly dense session, the moving backward with skipping algorithm can be applied.

The gift of the $k$-th receiver is set to $[low_k, low_{k+\tau}-1]$, where $\tau \geq 2$. A *skipping interval* is denoted by τ. The value of $\tau$ may be determined by the source since the source knows the join pattern and can decide appropriate values. The gift is extended to cover the gift of $\tau$ receivers. Thus, the $k$-th receiver selects the $(k-\tau \times h)$th receivers as its forwarders by moving backward where $h \geq 1$. Large $\tau$ results in a large gift per forwarder and reduces the forwarder selection overhead. MBL can be considered a special case of MBS where $\tau=1$.

In Fig. 3(b), with $\tau = 2$, the forwarder selection sequence will be the $(k-2)$th, $(k-4)$th, and $(k-6)$th receivers. The gift of each receiver will be about twice as large as that of the MBL algorithm.

### C. Moving Backward with Probing Algorithm

The previous algorithms determine the forwarders only according to the order of *low*. It will take a long time for a forwarder of low performance to forward a gift to a receiver, while preventing the receiver from having another forwarder of high performance. To avoid such a problematic situation, we propose the moving backward with probing (MBP) algorithm which considers variable network traffic parameters as the forwarder selection criteria. Various peer selection metrics such as RTT, bottleneck bandwidth, or 10 kB probing [4] can be combined. The most important difference from the previous algorithms is that the forwarder selection order can be different

from that of the reverse order of *low*. The probing results are the most important selection criteria in this algorithm.

Initially, $\omega$ preceding receivers are probed at a time where $\omega \geq 2$ and the results are stored in the candidate queue and sorted in the performance order. The *probing* coefficient is denoted by $\omega$ and it determines the probing overhead. The best one is selected as a forwarder. To select another forwarder, $\omega$ new preceding receivers are probed. The preceding receivers are selected from the forwarder candidates in the reverse order of *low*. The probing results are stored and the best one is selected as a new forwarder. In other words, $(\omega-1) + \omega$ records are compared to select the best one. The MBL algorithm can be considered a special case of MBP, where $\omega=1$.

After repeating the procedure $\rho$ times, the length of the candidate queue, $len_{cq}$, becomes $(\omega-1)\times\rho$. It means that each forwarder is the best one among $(\omega-1)\times\rho+1$ candidates. The queue cannot grow longer than $max_{cq}$, which is a predefined value. We propose $max_{cq}$ should be larger than 5. According to [14], an 80% optimal peer can typically be found by trying less than 5 random candidates. It seems enough to keep $max_{cq}$ 5. However, the candidates in the queue are those excluded from the previous forwarder selection procedures and their performance will not be so high as to apply the 80% property directly. Thus a larger $max_{cq}$ is required.

If a receiver wants to have simultaneous $\delta$ forwarders, it will set the *offset* of a *preceding_request* message to $\delta\times\omega$. It repeats probing and selecting $\delta$ times consecutively. Figure 3(c) shows an example of this algorithm. It is the case with $\omega=2$ and $\delta=3$. In the first probing step of the round, the ($k$–1)th and ($k$–2)th receivers are probed and the results are stored in the candidate queue. The ($k$–2)th receiver is selected since it shows the best performance. In the second step, the ($k$–3)th and ($k$–4)th receivers are probed and the results are stored in the candidate queue. Let us assume that their performance is lower than that of the ($k$–1)th receiver. Then, the ($k$–1)th receiver is selected as the next forwarder. In the third step, the ($k$–5)th and ($k$–6)th receivers are probed and the results are stored in the candidate queue. The ($k$–3)th receiver shows better performance than the ($k$–4)th, ($k$–5)th, and ($k$–6)th receivers. Thus it is selected as the next forwarder. In conclusion, the $k$th receiver selects the ($k$–2)th, ($k$–1)th, and ($k$–3)th receivers as its forwarders, sequentially. After the gift of the forwarders runs out, the receiver resumes probing by acquiring new forwarder candidates and exploiting the probing results stored in the candidate queue.

## IV. Performance Evaluation

### 1. Numerical Analysis

We evaluate the impact of the proposed forwarder selection algorithms by observing the *completion delay*, which is defined as the time taken to receive the required data packets to rebuild the original file. We also observe the network bandwidth utilization enhancement achieved by multiple forwarders. Before proceeding with the analysis, we assume there are enough preceding receivers and the session is in the steady state. Furthermore, neither link stress nor performance degradation is caused by receiving data packets from multiple forwarders. We do not consider network dynamics, neither.

#### *A. Impact of Skipping*

To simplify and clarify the analysis, we define several parameters and make certain assumptions. First, the join time interval of receivers is constant as $\lambda$. Therefore, the time length of the gift of a receiver will be constant as $\lambda\tau$, where $\tau$ is the skipping interval. Second, the time that is required to determine a proper parent is constant as $\alpha$. The time required to negotiate with a new forwarder is also constant as $\theta$. Third, the available download bandwidth from a forwarder and the available download bandwidth from a parent are constant as $v_f$ and $v_p$, respectively. In fact, the connection with the parent and that with a forwarder are established via TCP, which grows additively. It may take several round trip times to converge to a stable congestion window. The expression becomes long and complex if the steps are included. To simplify the expression, we assume a constant data transmission rate. Finally, every receiver has maximum forwarders as $\delta$. Since the join time interval and forwarder selection delay are constant, the receivers select new $\delta$ forwarders per $(\lambda\tau+\theta)$ time after the first forwarder selection trial.

The total size of data packets required to recover the original file is $M$. The receiver may acquire the required $M$ data packets from the parent node and $\delta$ forwarders. It is expressed roughly like (1), where $t$ is the completion delay.

$$(t-\alpha)\times v_p + c\times\delta\times(\lambda\times\tau)\times v_f = M \tag{1}$$

The first term of (1), $(t-\alpha)\times v_p$, represents the size of data packets delivered by the parent. The second term, $c\times\delta\times(\lambda\times\tau)\times v_f$, denotes that of the data packets forwarded by the forwarders, where $c$ represents the number of chances to select the forwarders. From (1), we can get (2) to show the impact of the skipping interval $\tau$ more clearly.

$$\begin{aligned} t &= \frac{(M+\alpha\cdot v_p - c\cdot\delta\cdot\lambda\cdot\tau\cdot v_f)}{v_p} \\ &= \frac{M}{v_p} + \alpha - c\cdot\delta\cdot\frac{v_f}{v_p}\cdot\lambda\cdot\tau \end{aligned} \tag{2}$$

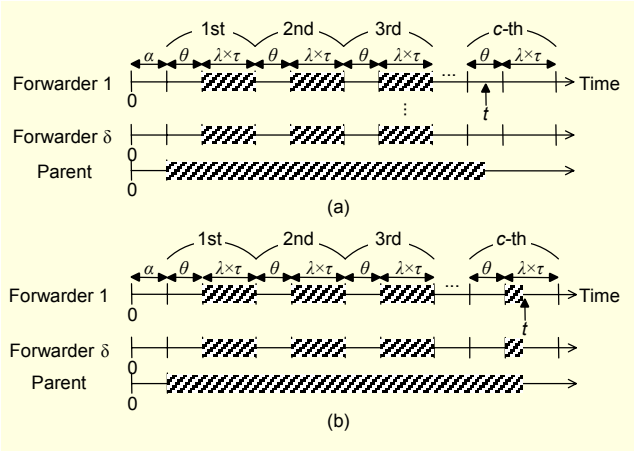



Fig. 5. Data packet reception from forwarders and a parent with MBS algorithm.

With the fixed value of $c$, $\tau$ affects the completion delay negatively. In other words, the completion delay decreases as $\tau$ increases. Especially with small $\lambda$, a large $\tau$ is required to reduce the completion delay sufficiently.

Equation (2) is a rough expression, but we will clarify the cases and show the impact of the skipping interval more thoroughly. If we try to exploit the forwarders as long as it takes for the file reception to complete, the file reception may complete while negotiating with the $c$th forwarders group as shown in Fig. 5(a) or while receiving the gift from the $c$th forwarder group as shown in Fig. 5(b).

For the case depicted in Fig. 5(a), $M$ can be expressed like (3-1) and $t$ will be expressed like (3-2). To simplify the expression, we assume $v_f$ and $v_p$ are the same as $v$.

$$\{\delta(\lambda\tau)(c-1)+(t-\alpha)\}v=M \tag{3-1}$$

$$t=[(\lambda\tau+\theta)(c-1)+\theta+\alpha,(\lambda\tau+\theta)c+\alpha] \tag{3-2}$$

From (3-1) and (3-2), we can obtain an expression for determining $c$ like (3-3) and $c$ will be an integer that satisfies (3-3).

$$\frac{M+v\{\delta(\lambda\tau)+\lambda\tau\}}{v\{\delta(\lambda\tau)+(\lambda\tau+\theta)\}}\le c<\frac{M+v\{\delta(\lambda\tau)+(\lambda\tau+\theta)\}}{v\{\delta(\lambda\tau)+(\lambda\tau+\theta)\}} \tag{3-3}$$

For the case shown in Fig. 5(b), to satisfy the equality in (2), $c$ may not be an integer and another expression (4-1) is used to express $M$ to determine $c$ as an integer.

$$\{(t-i\theta-\alpha)\delta+(t-\alpha)\}v=M \tag{4-1}$$

Since $t$ lies within the $c$th forwarder group's gift forwarding, $t$ will be expressed as (4-2).

$$t=[(\lambda\tau+\theta)(c-1)+\theta+\alpha,(\lambda\tau+\theta)c+\alpha] \tag{4-2}$$

From (4-1) and (4-2), we can get an expression to determine $c$. It will be an integer that satisfies (4-3).

$$\frac{M}{v\{\delta(\lambda\tau)+(\lambda\tau+\theta)\}}\le c<\frac{M+v\{\delta(\lambda\tau)+\lambda\tau\}}{v\{\delta(\lambda\tau)+(\lambda\tau+\theta)\}} \tag{4-3}$$

Table 1. Value of the variables and description.

| Variable | Value | Description |
|---|---|---|
| $\theta$ | 0.5 s | Delay required to find proper parent node |
| $M$ | 21 MB | Size of original file = 20 MB<br>FEC decoding overhead = 0.05 |
| $\delta$ | 10 | Maximum number of simultaneous forwarders |
| $v$ | 1 Mbps | Data reception rate from a forwarder ($v_f$)<br>Data rate from the parent node ($v_p$) |
| $\tau$ | 1, 2, 3, 4, 5 | Skipping interval |
| $\lambda$ | 0.1, 0.5, 1, 5, 10 | Join time interval (in seconds) |

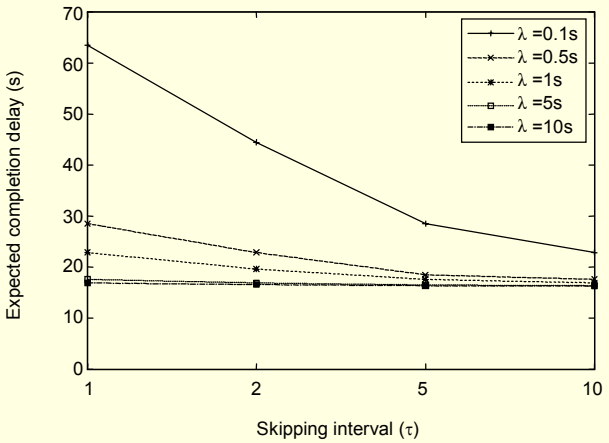



Fig. 6. Impact of skipping.

With the constant parameters explained in Table 1, we can determine an integer $c$ which satisfies either (3-3) or (4-3) and resulting completion delay $t$.

Figure 6 shows the expected completion delay as varying the skipping interval $\tau$ and join interval $\lambda$. Note that it is the same as the MBL algorithm when $\tau = 1$. Following the figure, when $\lambda$ is large, skipping provides no benefit. This is because there are not sufficient preceding receivers and skipping utilizes only some portion of them. However, when $\lambda$ is small, it shows a significant performance gain as $\tau$ grows because skipping reduces the forwarder selection overhead by decreasing the frequency of selection.

*B. Impact of Probing*

It is assumed that receivers are classified into $K$ classes depending on their upstream bandwidth in decreasing order. Class $k$ is denoted as $C_k$ and the probability that a receiver belongs to $C_k$ is $\varphi_k$. The mean upstream bandwidth of the receivers in $C_k$ is denoted as $v_k$. $v_a > v_b$ if $a < b$.

Assume that $q$ forwarders are utilized during the session. To simplify, we set $K$ to 3. Then, the probability distribution function $P$ of $x$, $y$, and $z$, which are the numbers of forwarder candidates classified into $C_1$, $C_2$, and $C_3$ respectively, follows multinomial distribution as

$$P(x,y,z,q) = \frac{q!}{x!y!z!}\varphi_1{}^x\varphi_2{}^y\varphi_3{}^z, \tag{5}$$

where $x+y+z=q$.

• *MBL Algorithm*

The total upstream bandwidth of a receiver's forwarders, in other words, the total bandwidth that is used by the forwarders' data packet forwarding, is determined by

$$U_m(x,y,z) = xv_1 + yv_2 + zv_3, \tag{6}$$

where $x+y+z=q$.

With the MBL algorithm, the number of the forwarder candidates will be the same as $q$, since the forwarder is selected in the reverse of their *low* order. Thus, the expected total upstream bandwidth of the forwarder candidates is acquired as

$$F_m(q) = \sum_{y=0}^{q}\sum_{x=0}^{q-y} P(x,y,q-(x+y),q)\times U_m(x,y,q-(x+y)). \tag{7}$$

• *MBP Algorithm*

The MBP algorithm selects a forwarder by probing $\omega$ preceding receivers. To simplify the analysis, let us assume that $q$ preceding receivers are already probed before selecting $q$ forwarders. Then, for a given set of forwarders, the total upstream bandwidth of a receiver's forwarders is determined by

$$U_p(x,y,z) = \begin{cases} qv_1, & x \ge q \\ xv_1 + (q-x)v_2, & x+y \ge q, \quad x < q \\ xv_1 + yv_2 + (q-(x+y))v_3, & x+y < q \end{cases}, \tag{8}$$

where $x+y+z=\omega q$.

As (6) shows, with the MBL algorithm, there is no choice of high performance forwarders. The expected total bandwidth is just a summation of the provided performance of the forwarders. However, as (8) shows, with a given number of forwarders, the MBP algorithm tries to select the best one among the forwarder candidates and enhances network bandwidth utilization. The bandwidth is mostly consumed by the high rank forwarders. With the fixed capability of the forwarder candidates, small $q$ results in the forwarders of high rank and will provide high performance gain compared with the MBL algorithm.

Since the forwarders are selected out of $\omega q$ forwarder candidates, the expected total upstream bandwidth of the forwarder candidates is acquired as

$$F_p(q') = \sum_{y=0}^{q'}\sum_{x=0}^{q'-y} P(x,y,q'-(x+y),q')\times U_p(x,y,q'-(x+y)), \tag{9}$$

where $q' = \omega q$ .

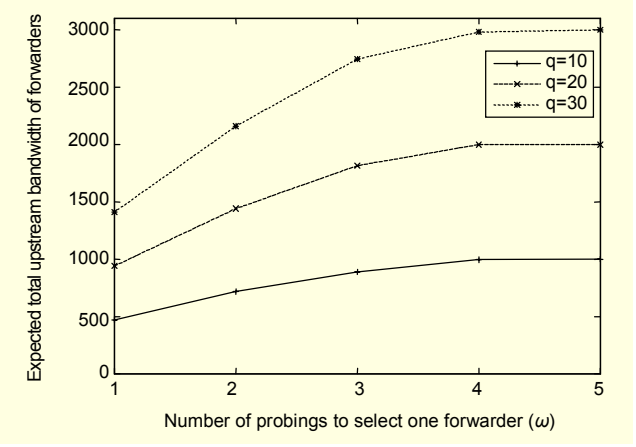



Fig. 7. Impact of probing.

Figure 7 shows an advantage of probing in terms of the expected total upstream bandwidth of forwarders. We use parameters on the analogy of [15]. The variables $\varphi_1$, $\varphi_2$, and $\varphi_3$ are 0.3, 0.5, and 0.2, respectively, while $v_1$, $v_2$, and $v_3$ are 100, 30, and 10, respectively. It is same as the MBL algorithm when $\omega$ = 1. As $\omega$ becomes larger, with fixed $q$, the expected total upstream bandwidth of forwarders increases and rapidly converges to the maximum capacity. With the same conditions as the join interval, negotiation interval, and number of simultaneous forwarders, the high total upstream bandwidth results in high utilization of network resources and reduction of the completion delay.

## 2. PlanetLab Experiments

We implemented the proposed method including the

forwarder selection algorithms and the control message exchange procedure using MACEDON [14], by which Bullet was implemented as well. We ran the implementations on PlanetLab[1)] [10] and compare the performance evaluation results with those of Bullet.

We used 50 nodes for the experiments; 5 nodes in Asia, 4 in Europe, 39 in North America, 1 in South America, and 1 in Australia. No node came from the same site. The file size was 20 MB, and the stretch factor and decoding overhead for FEC coding were 10 and 0.05, respectively, as in [17]. The maximum number of forwarders that a sink could have and the maximum number of sinks that a forwarder could have were set to 10. The sending rate of a source and the maximum receiving rate of a receiver were set to 1Mbps, as in [6]. RTT was used as the probing metric for the MBP algorithm. The join time interval of each receiver was set to 10 seconds and TCP was used as a transport layer protocol. The join sequence of receivers was determined once, and used throughout all of the experiments. For simplicity, we assumed that there would be no failure of receivers and if a receiver received enough data packets to recover a file, it would leave the session immediately.

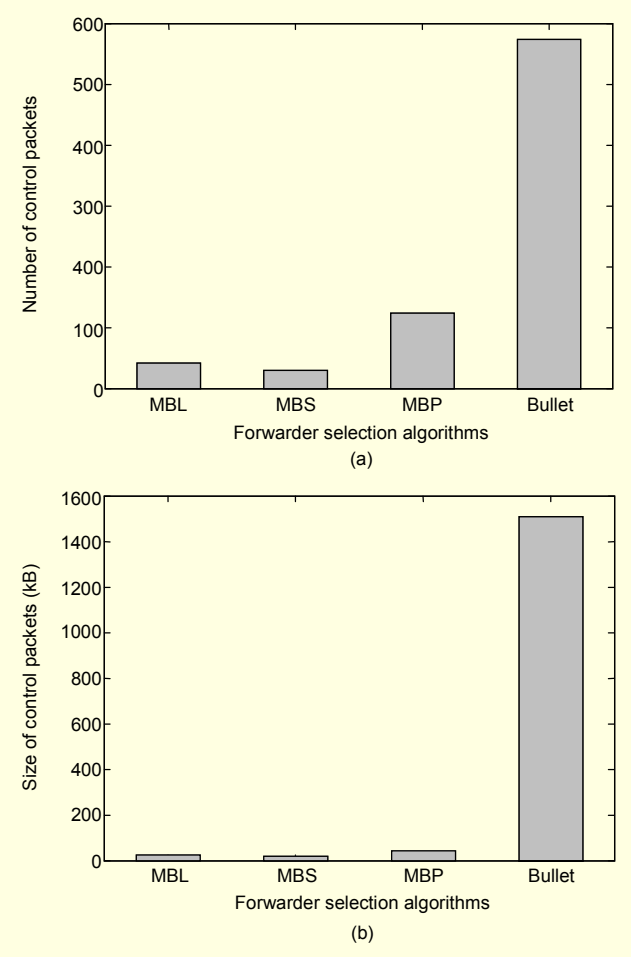



Fig. 8. Comparison of control overheads.

1) PlanetLab is a collection of about 700 machines hosted by 330 sites, spanning over 25 countries as of June 2006. One of PlanetLab's main purposes is to serve as a testbed for overlay networks. By applying our implementation on PlanetLab, we can acquire more practical experiment results.

### *A. Data Packet Duplication and Control Overhead*

In terms of data packet duplication, Bullet shows 0.9 MB of duplicate data packets due to the false positive and 4% data packet duplication ratio. However, the proposed forwarder selection algorithms incur no data packet duplication because each gift does not overlap with others.

Figure 8 compares the number and size of control packets of Bullet and the proposed method. In terms of the number of control packets, Bullet requires 5 to 10 times more than the proposed method. In terms of the size of control packets, a receiver in Bullet requires 1.5 MB, which is hundreds of times larger than the requirement of a receiver in the proposed method. In Bullet, most of control overheads come from the exchanges of summary tickets [6], which are not necessary in the proposed method. Furthermore, even a source that acts as a coordinator in the proposed method incurs control overheads about hundred times less than normal receivers in Bullet in terms of the size of control packets. Among the three algorithms of the proposed method, the moving backward with probing algorithm causes more control overhead than the others due to the probing packets. In brief, the proposed method is much more efficient than Bullet in terms of data packet duplication and control overhead.

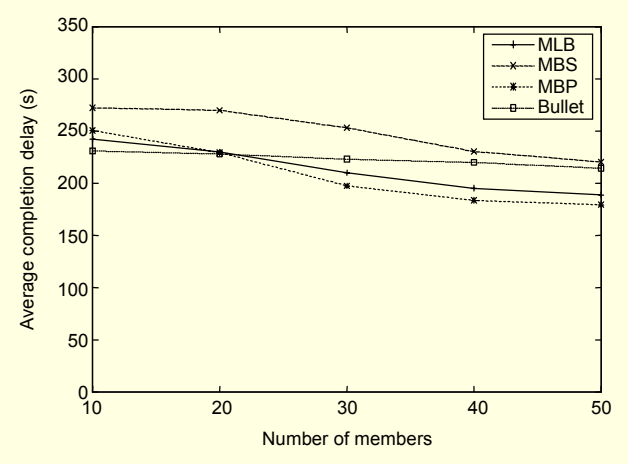



Fig. 9. Average completion delays.

### *B. Completion Delay*

Figure 9 shows an average completion delay of the algorithms in the proposed method and Bullet while varying

the number of receivers. Ten to fifty receivers were used for the experiments. A receiver in the proposed method selects forwarders only from the preceding receivers while a receiver in Bullet can select forwarders from all other receivers.

When the number of receivers was small, the proposed method showed larger completion delay than Bullet. This is because, in the proposed method, there is a smaller number of available forwarders than the number available in Bullet. When the number of receivers is 10, for example, the fifth receiver can have at most 4 forwarders in the proposed method. However, the fifth receiver can have at most 9 receivers in Bullet.

However, as the number of receiver gets larger, the completion delays of the proposed method become smaller because there are enough preceding receivers to be used until completing file reception. Thus, for example, when the number of receivers is 50, the MBL and MBP algorithms show shorter completion delay than Bullet. As shown in previous numerical analysis and Fig. 6, the MBS algorithm works better with a short join interval. Therefore, since this experiment uses a long join interval, the MBS algorithm is less efficient, compared with the other proposed algorithms.

Figure 10 illustrates the benefits of MBP over MBL by showing the ratio of the completion delays of the MBL algorithm to that of the MBP algorithm. The higher ratio indicates a higher performance gain with the MBP algorithm. In the experiments, we varied both $\delta$ and $\gamma$ to 2, 5, and 10 with 50 nodes without changing the other parameters. As $\delta$ increases, the completion delay ratio becomes smaller. This is because the size of the forwarder candidate pool becomes smaller and most forwarder candidates are selected as forwarders regardless of their ranks. With small $\delta$ such as 2, a receiver can have a high probing coefficient and has more chances to select its forwarders from higher ranks.

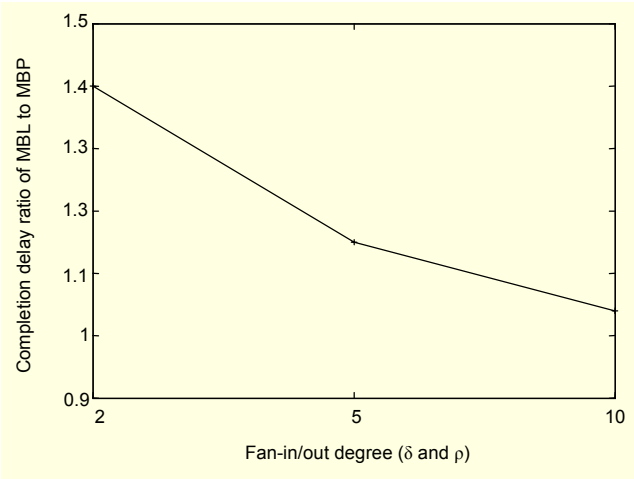



Fig. 10. Benefits of MBP algorithm over MBL algorithm.

In practice, the average number of forwarders that a receiver could use was less than 17 in the evaluation environment. This means that even though the probing-based method is used, if $\delta$ is 10, most of the candidates in the candidate queue are eventually used as forwarders regardless of their rank. For this reason, ranking the candidates and selecting the highly ranked ones does not significantly enhance the throughput.

## V. Conclusion

In this paper, we proposed a simple and efficient application layer multicast method for file distribution. Existing multiple forwarder-based methods require high control overheads to avoid packet duplication resulting from multiple forwarders. The proposed method exploits the fact that receivers join a session at different times. We proposed an interaction procedure between the source and the receivers to select proper forwarders. In addition, we suggested three algorithms for forwarder selection: moving backward linearly (MBL), moving backward with skipping (MBS), and moving backward with probing (MBP). The MBL algorithm is the basic one which selects the forwarders in the reverse order of the first data packet sequence number. The MBS and MBP algorithms are variations of the MBL algorithm designed to enhance the gain of multiple forwarders. For highly dense sessions with a short join interval, the MBS algorithm is preferred. The MBP algorithm is preferred when the maximum number of simultaneous forwarders is small and the network resources of the other receivers are heterogeneous.

We numerically analyzed the impact of the forwarder selection algorithms. In addition, we ran the proposed method on PlanetLab and proved the superior performance of the proposed method compared with the existing Bullet method, in terms of packet duplication, control overhead, and throughput.

We are currently working on the following issues. First, fair sharing of the network bandwidth will be studied to support the case that a receiver participates in multiple sessions at the same time. Second, a dynamic adaptation method will be investigated in the case that the join time interval is highly variable.

## References

[1] G. Kwon and J. Byers, "ROMA: Reliable Overlay Multicast with Loosely Coupled TCP Connections," *Proc. of IEEE INFOCOM,* vol. 1, 2004, pp. 385-395.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," *Proc. of ACM SIGCOMM*, 2002, pp. 205-217.

[3] Y. Chu, S. Rao, and H. Zhang, "A Case for End System Multicast," *Proc. of ACM Sigmetrics*, 2000, pp. 1-12.

[4] J. Jannotti, D.K. Gifford, K.L. Johnson, M.F. Kaashoek, and J.W. O'Toole Jr., "Overcast: Reliable Multicasting with an Overlay Network," *Proc. of the Fourth Symposium on Operating System Design and Implementation (OSDI)*, 2000, pp. 197-212.

[5] S. Banerjee and B. Bhattacharjee, "A Comparative Study of Application Layer Multicast Protocols," *Proc. of IEEE INFOCOM,* vol. 4, 2005, pp. 2809-2814.

[6] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," *Proc. of ACM Symp. on Operating Systems Principles (SOSP)*, 2003, pp. 282-297.

[7] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Content Distribution in a Cooperative Environment," *Proc. of Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, Feb. 2003, pp. 298-313.

[8] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed Content Delivery Across Adaptive Overlay Networks," *Proc. of ACM SIGCOMM*, 2002, pp. 47-60.

[9] Y. Zhu, B. Li, and J. Guo, "Multicast with Network Coding in Application-Layer Overlay Networks," *IEEE J. of Selected Areas in Commun. (JSAC)*, vol. 22, no. 1, Jan. 2004, pp. 107-120.

[10] K.C. Almeroth and M.H. Ammar, "Multicast Group Behavior in the Internet's Multicast Backbone (MBone)," *IEEE Communications*, June 1997, vol. 35, pp. 224-229.

[11] http://www.planet-lab.org

[12] B. Cohen, "Incentives Build Robustness in BitTorrent," Proc. of Workshop on Economics of Peer-to-Peer Systems, May 2003, http://www.bittorrent.com/bittorrentecon.pdf

[13] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data," *Proc. of ACM SIGCOMM*, vol. 28, no. 4, Oct. 1998, pp. 56-57.

[14] T.S.E. Ng, Y.H. Chu, S.G. Rao, K. Sripanidkulchai, and H. Zhang, "Measurement-Based Optimization Techniques for Bandwidth-Demanding Peer-to-Peer Systems," *Proc. of IEEE INFOCOM,* vol. 3, 2003, pp. 2199-2209.

[15] S. Saroiu, K.P. Gummandi, and S.D. Gribble, "Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts," *Multimedia Systems*, vol. 9, no. 2, Aug. 2003, pp. 170-184.

[16] A. Rodriguez, S. Bhat, C. Killian, D. Kostić, and A.Vahdat, "MACEDON: Methodology for Automatically Creating, Evaluating, and Designing Overlay Networks," *Proc. of NSDI*, 2004, pp. 267-280.

[17] S. Rost, J. Byers, and A. Bestavros, "The Cyclone Server Architecture: Streamlining Delivery of Popular Content," *Computer Communications Journal*, vol. 25, no. 4, Mar. 2002, pp. 403-412.

**Soojeon Lee** received the BS degree in computer science from Korea University, Korea in 2003, and the MS degree in computer engineering from Information and Communications University (ICU), Korea in 2005. He has worked as a Researcher in Electronics and Telecommunications Research Institute (ETRI) from 2005. His research interests include operation and mission planning of satellite ground control systems.

**Kyungran Kang** received the BS degree in computer engineering from Seoul National University, Korea in 1992, and the MS degree and PhD degrees in computer science from KAIST, Korea in 1994 and 1999, respectively. From 1999 to 2000, she worked as a Senior Researcher at ETRI. From 2000 to 2002, she worked as a Principal Researcher at DigitalWave, Inc. From 2002 to 2003, she worked as an Assistant Research Professor at ICU. She has been an Assistant Professor at Ajou University since 2004. Her research interests include computer networks, wireless networks, and community computing. Dr. Kang is a member of KISS, IEEE, and ACM.

**Dongman Lee** received the BS degree in computer engineering from Seoul National University, Korea in 1982, and the MS and PhD degrees in computer science from KAIST, Korea in 1984 and 1987, respectively. From 1988 to 1997, he worked as a Technical Contributor at Hewlett-Packard. From 1998 to Nov. 2003, he was an Associate Professor in the School of Engineering, Information and Communications University (ICU), Daejon, Korea. Since Dec. 2003, he has been a Professor at ICU. He has been actively participating in Korean and International Internet Number and Name Committees since 1998. He received a Prime Minister Award as recognition in the advancement of the Korean Internet in 2000. His laboratory, Collaborative Distributed Systems and Networks Lab (http://cds.icu.ac.kr), was appointed as a National Research Laboratory in 2001. He has served as a member of the program committee of numerous international conferences including IEEE Multimedia, COMPSAC, PDCS, PRDC, VSMM, ICAT, and so on. He is Editor of JCN. His research interests include distributed systems, computer networks, mobile computing and pervasive computing. Dr. Lee is a member of KISS, IEEE, and ACM.

**Jae-Hoon Kim** received the PhD degree in computer engineering from Chungbuk National University, Cheongju, Korea in 2001. He joined ETRI in 1983, where he was involved in developing the Intelligent Network and KOREASAT Projects. From 1992 to 1994, he was an OJT Engineer in Martra-Marconi Space in the U.K. for the KOREASAT Project. From 1995 to 1999, he participated in the KOMPSAT-1 Ground Mission Control Project as a Principle Member of engineering staff in system engineering. From 2000 to 2005, he participated in the KOMPSAT-2 Ground Mission Control Project as a Team Leader. He is now working for the COMS-1, KOMPSAT-3 and KOMPSAT-5 Ground Mission Control Projects as a Team Leader. His research interests are security in satellite communications, fault diagnosis of satellites using AI technologies, and system modeling using objected-oriented technologies.