Broadcast with Tree Selection on An Overlay Network

Takeshi Kaneko, and Kazuyuki Shudo
Tokyo Institute of Technology

***Abstract*—On an overlay network where a number of nodes work autonomously in a decentralized way, the efficiency of broadcasts has a significant impact on the performance of a distributed system built on the network. While a broadcast method using a spanning tree produces a small number of messages, the routing path lengths are prone to be relatively large. Moreover, when multiple nodes can be source nodes, inefficient broadcasts often occur because the efficient tree topology differs for each node. To address this problem, we propose a novel protocol in which a source node selects an efficient tree from multiple spanning trees when broadcasting. This technique reduces the frequency of inefficient broadcasts for multiple source nodes, and shortens routing paths while maintaining a small number of messages. We examined path lengths and the number of messages for broadcasts on various topologies. As a result, especially for a random graph, our proposed method shortened path lengths by approximately 28% compared with a method using a spanning tree, with almost the same number of messages.**

I. Introduction

An overlay network is an application-level logical network built on an existing network such as the Internet. It is applied to various distributed systems: e.g. distributed key/value stores [1], video streaming [2], and online games [3]. In recent years, application to the fields of IoT and blockchain is also expected [4].

On an overlay network where a number of nodes work autonomously in a decentralized way, the large-scale distributed system requests frequent information exchange with low latency to achieve high scalability and high reliability. Therefore, the efficiency of broadcasts has a significant impact on the performance of the distributed system.

A variety of methods to realize efficient broadcasts have been proposed [5]. In particular, while tree-based methods using a spanning tree built on the network produce a small number of messages, the routing path lengths are prone to be larger than those of flooding and gossip-based methods. Moreover, when multiple nodes issue broadcasts on a single spanning tree, inefficient broadcasts often occur because the appropriate topology of the tree differs for each node.

To address this problem, we propose a novel protocol in which a source node selects an efficient tree from multiple spanning trees when broadcasting. This method reduces the frequency of inefficient broadcasts for multiple source nodes, even if most participating nodes can be source nodes. It thereby shortens routing path lengths while maintaining the advantage of tree-based methods, which is producing a small number of messages. We adopt Plumtree [6] as the spanning tree construction protocol.

The remainder of this paper is organized as follows. Section II presents the related work on broadcasts using spanning trees. Section III defines notations used in this paper. Section IV presents the proposed method. Section V presents the evaluation results. Finally, Section VI presents the conclusion.

II. Related Work

A. Plumtree

Leitao et al. proposed a broadcast method called Plumtree [6], which combines a gossip protocol and a tree-based protocol. This achieves both a small number of messages and high reliability. In addition, the routing path lengths of each broadcast are relatively small for multiple source nodes due to the optimization process of the tree topology. However, as with other tree-based broadcasts, the lengths are still prone to be larger than those of flooding because broadcast messages are sent only on a single spanning tree topology.

B. Methods Using Multiple Spanning Trees

SplitStream [7] and Chunkyspread [8] are multicast methods that construct multiple spanning trees on a network. Because the purpose is streaming, the content is divided into some strips and they are sent with all trees simultaneously; the load is thereby balanced for each tree. On the other hand, the proposed method selects an appropriate tree instead of multiple trees and shortens the routing paths. SplitStream also differs from the proposed method in that it constructs spanning trees on a structured overlay Pastry [9], rather than an unstructured overlay.

Thicket [10] is a multicast method that constructs and manages multiple spanning trees on an unstructured overlay. As with other existing methods, the purpose of using multiple spanning trees is load balancing. On the other hand, the proposed method selects one tree from multiple spanning trees when broadcasting to shorten the routing paths. It is possible to partially apply the proposed method to Thicket because the idea for the tree selection is simple.

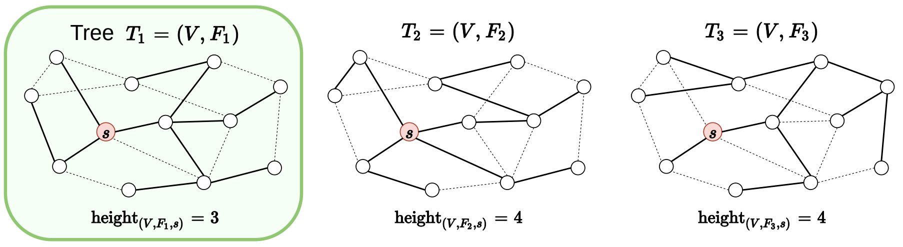
III. Notation

This section defines notations used in this paper.

An undirected graph G is represented by a pair (V, E), where V denotes a set of vertices and E denotes a set of edges. We define the following for $G = (V, E)$:

- $\mathrm{adj}_G \colon V \ni v \mapsto$ (the set of all neighbors of v) $\in 2^V$
- $\deg_G \colon V \ni v \mapsto |\mathrm{adj}_G(v)| \in \mathbb{N}$
- $\mathrm{dist}_G \colon\ V \times V \ni (u, v) \mapsto$
  (the distance between u and v) $\in \mathbb{N} \cup \{\infty\}$
  - Especially, $^\forall v \in V, \mathrm{dist}_G(v, v) = 0$.

An undirected rooted tree T is represented by a triple (V, F, r); where V denotes a set of vertices, F denotes a set of edges, and r denotes the root node. It can also be regarded simply as an undirected graph $T = (V, F)$. We define the following for $T = (V, F, r)$:



Tree $T_1 = (V, F_1)$
$T_2 = (V, F_2)$
$T_3 = (V, F_3)$
s
$\text{height}_{(V,F_1,s)} = 3$
$\text{height}_{(V,F_2,s)} = 4$
$\text{height}_{(V,F_3,s)} = 4$

Fig. 1. An example of tree selection from multiple spanning trees for a source node s.

- $\text{parent}_T \colon V - \{ r \} \ni v \mapsto (\text{the parent of } v) \in V$
- $\text{children}_T \colon V \ni v \mapsto$ $(\text{the set of all children of } v) \in 2^V$
- $\text{subtree}_T \colon V \ni v \mapsto (\text{the rooted subtree induced by } v)$
- $\text{height}_T \coloneqq \max \{ \text{dist}_T(r, v) \mid v \in V \} \in \mathbb{N}$

IV. Proposed Method

In the proposed method, multiple spanning trees are constructed on an overlay network (Section IV-A), and a source node selects an appropriate tree from them (Section IV-B) and issues a broadcast (Section IV-C).

Each spanning tree is embedded in an unstructured overlay network managed by a peer sampling service [11], [12]. A spanning tree is maintained along the Plumtree protocol [6]. In Plumtree, a node has two types of neighbors in *eagerPushPeers* and *lazyPushPeers* sets. *EagerPushPeers* include neighbors along tree edges and *lazyPushPeers* include other neighbors. A broadcast is basically performed over a tree, that is, forwarding to *eagerPushPeers*. A node sometimes forwards a notification of arrival of a message to *lazyPushPeers*. The notified node starts repairing a tree if it has not received the message along the tree. Tree construction is as follows. A node has all its neighbors in its *eagerPushPeers* at the start of the construction. The initial tree is lengthy and there are multiple paths to a single node. If a node receives the same message from multiple neighbors, the node replies a PRUNE message to the sender and both the sending and receiving nodes move each other from *eagerPushPeers* to *lazyPushPeers*.

Since messages for each broadcast are sent on a single tree, the number of messages is approximately $\Theta(n)$ for a stable overlay network where n denotes the number of participating nodes.

Moreover, our aim is to minimize the maximum path length from the source node to each node for each broadcast by the following process:

1) Suppose that a finite number of spanning trees $T_1 = (V, F_1), T_2 = (V, F_2), \ldots, T_k = (V, F_k)$ are managed on a graph $G = (V, E)$[1] of the network topology.
2) A source node $s \in V$ selects the tree T_λ $(\lambda \in \{ 1, 2, \ldots, k \})$ that minimizes
$$\text{height}_{(V,F_\lambda,s)} = \max \{ \text{dist}_{T_\lambda}(s, v) \mid v \in V \}, \quad (1)$$
which is the maximum path length of each simple path whose one end is the node s on T_λ.

[1]Strictly speaking, an overlay network with a peer sampling service forms a topology of a directed graph. However, for simplicity, the network topology is regarded as an undirected graph in this paper because each managed spanning tree behaves like an undirected graph.

Algorithm 1: Fields and Initialization in a Node $v_{current}$

```
1  data structure Tree
2      field eagerPushPeers: Set⟨Node⟩
3      field lazyPushPeers: Set⟨Node⟩
4      field distFor: Map⟨Node, Int⟩
5      field parent: Node
6  upon receiving ⟨INIT⟩ then
7      peers ← getPeers()
8      trees ← new Map⟨Int, Tree⟩
9      receivedMsgs ← new Map⟨Int, Message⟩
10     missing ← ∅
```

Fig. 1 shows an example of tree selection. In this example, the network manages three spanning trees T_1, T_2, and T_3, the source node is a node s, and then $\text{height}_{(V,F_1,s)} = 3$, $\text{height}_{(V,F_2,s)} = 4$, and $\text{height}_{(V,F_3,s)} = 4$. Since T_1 minimizes (1), the source node s selects T_1 from them.

Note that, however, since each node does not have global information on the network, it selects a tree based on a value estimated only from information it knows rather than the exact value of (1).

A. Construction of spanning trees

The proposed method constructs multiple spanning trees on the network at the beginning of the protocol. The number of trees, a positive integer k, is a system parameter determined before running the protocol. We randomly select k start nodes from the network[2] and construct a spanning tree for each start node based on the delivery tree by flooding.

Algorithm. 1 and Algorithm. 2 are pseudocodes on initialization and spanning tree construction, respectively. For simplicity, the latter pseudocode assumes that no messages are lost and no nodes join or leave during the construction of a spanning tree. In practice, it is necessary to address such situations by using timeout timers.

Each start node v_{start} constructs a new spanning tree by issuing an event $\langle \texttt{ConstructTree}, \mathit{treeId}_{new} \rangle$ to itself v_{start}. Herein, we generate treeId_{new}, the ID corresponding to the tree to be constructed, e.g., by a pseudorandom number to avoid duplicate IDs.

[2]For example, we can sample some start nodes by doing a random walk from a node and selecting nodes on every certain number of hops. However, because sampling with a simple random walk is biased by the degree of each node, if it is preferred to sample nodes with a uniform distribution, e.g., we should use a random walk using Metropolis-Hastings algorithm [13].

Algorithm 2: Tree Construction in a Node $v_{current}$

```
upon receiving $\langle$ConstructTree, $treeId\rangle$ from $v_{sender}$ then
    if $treeId \notin trees.keys()$ then
        $t \leftarrow$ new $Tree$
        $trees[treeId] \leftarrow t$
        $t.eagerPushPeers \leftarrow peers$
        $t.parent \leftarrow v_{sender}$
        for $v_{adj} \in t.eagerPushPeers$ do
            if $v_{adj} = v_{sender}$ then
                continue
            send $\langle$ConstructTree, $treeId\rangle$ to $v_{adj}$
    else
        send $\langle$UpdateDistTowardRoot, $treeId, \infty\rangle$ to $v_{sender}$

upon receiving
  $\langle$UpdateDistTowardRoot, $treeId, distForSender\rangle$ from
  $v_{sender}$ then
    $t \leftarrow trees[treeId]$
    if $distForSender < \infty$ then
        $t.distFor[v_{sender}] \leftarrow distForSender$
    else
        $t.eagerPushPeers \leftarrow t.eagerPushPeers - \{v_{sender}\}$
        $t.lazyPushPeers \leftarrow t.lazyPushPeers \cup \{v_{sender}\}$
    if $t.eagerPushPeers - \{ t.parent \} = t.distFor.keys()$ then
        if $t.parent = v_{current}$ then
            // when $v_{current}$ is the root node of $t$
            send $\langle$UpdateDistTowardLeaves, $treeId, 0\rangle$ to
              $v_{current}$
        else
            $maxDist \leftarrow$
              getMaxDistExceptOne($t.distFor, t.parent$)
            send
              $\langle$UpdateDistTowardRoot, $treeId, maxDist + 1\rangle$
              to $t.parent$

upon receiving
  $\langle$UpdateDistTowardLeaves, $treeId, distForSender\rangle$ from
  $v_{sender}$ then
    $t \leftarrow trees[treeId]$
    if $v_{sender} \neq v_{current}$ then
        $t.distFor[t.sender] \leftarrow distForSender$
    for $v_{adj} \in t.eagerPushPeers$ do
        if $v_{adj} = v_{sender}$ then
            continue
        $maxDist \leftarrow$ getMaxDistExceptOne($t.distFor, v_{adj}$)
        send
          $\langle$UpdateDistTowardLeaves, $treeId, maxDist + 1\rangle$
          to $v_{adj}$

function getMaxDistExceptOne($distFor, v_{excepted}$)
    $maxDist \leftarrow 0$
    for $(v_{adj}, d) \in distFor$ do
        if $v_{adj} = v_{excepted}$ then
            continue
        $maxDist \leftarrow \max\{maxDist, d\}$
    return $maxDist$
```

When each node $v_{current}$ receives an event $\langle\texttt{ConstructTree}, treeId\rangle$, if it does not have the corresponding tree data, then it generates a new tree t and initializes $t.eagerPushPeers$ with its neighbor nodes and sends the same event $\langle\texttt{ConstructTree}, treeId\rangle$ to each of the nodes. If it has the corresponding tree data, then it sends an event $\langle$UpdateDistTowardRoot, $treeId, \infty\rangle$ to the sender to indicate that $v_{current}$ has already received the event for the $treeId$. The sender that receives it excludes $v_{current}$ from $t.eagerPushPeers$. Through these processes, if each node is considered to have an edge with each node in $t.eagerPushPeers$, a spanning tree forms on the network where the start node v_{start} is the root node.

A node that sent an event $\langle\texttt{ConstructTree}, treeId\rangle$ to its neighbor nodes, after receiving an event $\langle$UpdateDistTowardRoot, $treeId, distForSender\rangle$ from each of them, sends an event $\langle$UpdateDistTowardRoot, $treeId, maxDist + 1\rangle$ to the parent node $t.parent$ where $maxDist$ is

$$maxDist = \max\left(\{\,0\,\} \cup \left\{ \begin{array}{l} t.distFor[v] \mid \\ \quad v \in t.eagerPushPeers - \{\, t.parent \,\} \end{array} \right\}\right). \tag{2}$$

In general, the following holds for any node $v \in V$ of a rooted tree $T = (V, F, v_{start})$:

$$\text{height}_{\text{subtree}_T(v)} = \max\left(\begin{array}{l}\{\,0\,\} \cup \\ \left\{ \text{height}_{\text{subtree}_T(u)} + 1 \;\middle|\; u \in \text{children}_T(v) \right\}\end{array}\right). \tag{3}$$

Thus, this process means computing the height of the tree by dynamic programming through the propagation of "distance" information into the root node v_{start}. Therefore, Each $t.distFor[v_{adj}]$ represents $(\text{height}_{\text{subtree}_T(v_{adj})} + 1)$.

After completing the propagation of "distance" information into the root node, "distance" information propagates into leaf nodes such that each node sends an event $\langle$UpdateDistTowardLeaves, $treeId, distForSender\rangle$ to each of the children. In general, the following holds for any $u \in V$ and any internal node $v \in V - \{\, v_{start} \,\}$ of a rooted tree $T = (V, F, v_{start})$:

$$\begin{aligned} &\text{height}_{\text{subtree}_{(V,F,v)}(u)} \\ &= \max\left(\{\,0\,\} \cup \left\{ \begin{array}{c} \text{height}_{\text{subtree}_{(V,F,v)}(w)} + 1 \mid \\ w \in \text{children}_{(V,F,v)}(u) \end{array} \right\} \right) \\ &= \max\left(\begin{array}{l} \{\,0\,\} \cup \\ \left\{ \begin{array}{l} \text{height}_{\text{subtree}_{(V,F,v)}(w)} + 1 \mid \\ \quad w \in \text{adj}_T(u) - \{\text{parent}_{(V,F,v)}(u)\} \end{array} \right\} \end{array} \right) \end{aligned} \tag{4}$$

If

$$\text{subtree}_{(V,F,v)}(u) = \text{subtree}_T(u), \tag{5}$$

Algorithm 3: A Broadcast with Tree Selection in a Node $v_{current}$

```
1  upon receiving ⟨BROADCAST, $msg$⟩ from $v_{sender}$ then
2      $msgId \leftarrow \mathtt{hash}(msg \parallel v_{current})$
3      $treeId \leftarrow \mathtt{selectTree}()$
4      send ⟨GOSSIP, $msg, msgId, treeId, 0, 0$⟩ to $v_{current}$
5  function selectTree()
6      $minTreeId \leftarrow \infty$
7      $minDist \leftarrow \infty$
8      for $(treeId, t) \in trees$ do
9          $distToEnd \leftarrow \max(\{0\} \cup t.distFor.values())$
10         if $distToEnd < minDist$ then
11             $minDist \leftarrow distToEnd$
12             $minTreeId \leftarrow treeId$
13     return $minTreeId$
```

then

$$\begin{aligned}&\mathrm{height}_{\mathrm{subtree}_{(V,F,v)}(u)}\\ =&\max\left(\{0\}\cup\left\{\begin{aligned}&\mathrm{height}_{\mathrm{subtree}_T(w)}+1\mid\\ &\quad w\in\mathrm{adj}_T(u)-\{\mathrm{parent}_T(u)\}\end{aligned}\right\}\right)\\ =&\max\left(\begin{aligned}&\{0\}\cup\\ &\left\{\mathrm{height}_{\mathrm{subtree}_T(w)}+1\;\middle|\;w\in\mathrm{children}_T(u)\right\}\end{aligned}\right)\\ =&\mathrm{height}_{\mathrm{subtree}_T(u)}.\end{aligned} \tag{6}$$

Moreover,

$$\mathrm{parent}_{(V,F,v)}(u)\in\mathrm{children}_T(u) \tag{7}$$

is a necessary condition for

$$u\neq v\wedge\mathrm{subtree}_{(V,F,v)}(u)\neq\mathrm{subtree}_T(u). \tag{8}$$

Thus, when completing the propagation into leaf nodes, each $t.distFor(v_{adj})$ represents $(\mathrm{height}_{\mathrm{subtree}_{(V,F,v_{current})}(v_{adj})}+1)$.

Through the above processes, a spanning tree is constructed on the overlay network. In a one-and-a-half round-trip broadcast by the start node v_{start}, every $t.distFor$ is computed by dynamic programming with piggyback of the "distance" information.

B. Tree selection

A source node $v_{current}$ broadcasts by issuing an event $\langle\text{BROADCAST}, msg\rangle$ for a message body msg to itself $v_{current}$. Here, it selects a tree from $trees$ and makes each node send messages on the tree.

Algorithm. 3 is a pseudocode on the tree selection. A function `selectTree` returns the ID of the tree t in $trees$ such that (9) is minimum.

$$\max(\{0\}\cup\{\,distFor[v_{adj}]\mid v_{adj}\in t.eagerPushPeers\,\}) \tag{9}$$

If t represents a rooted tree $(V,F,v_{current})$, $distFor[v_{adj}]$ represents $(\mathrm{height}_{\mathrm{subtree}_{(V,F,v_{current})}(v_{adj})}+1)$. Thus, (9) represents

$$\begin{aligned}&\max\left(\{0\}\cup\left\{\begin{aligned}&\mathrm{height}_{\mathrm{subtree}_{(V,F,v_{current})}(v)}+1\mid\\ &\quad v\in\mathrm{adj}_{(V,F,v_{current})}(v_{current})\end{aligned}\right\}\right)\\ =&\mathrm{height}_{(V,F,v_{current})}.\end{aligned} \tag{10}$$

This is equivalent to (1) where $v_{current}=s$. Therefore, this process intuitively means tree selection minimizing the maximum path length for the broadcast.

In practice, tree selection does not always minimize the maximum path length because the selection is based on local information and the network topology is dynamic. How the values $t.distFor$ are updated is important for improving the accuracy of the tree selection.

C. Message propagation and value update

Message propagation and topology management during a broadcast mostly follow the Plumtree protocol for a tree determined by the tree selection algorithm. For message propagation, a node provides a function receiving a `Gossip` message, a function receiving a `Prune` message. For topology management and optimization, a node provides a function receiving `Ihave` message, a function receiving a `Timeup` message, a function receiving `Graft` message. Pseudocode for them can be derived intuitively from the Plumtree protocol. An extended version of this paper will provide the pseudocode.

To support dynamics of the network for tree selection, each node updates the values of $t.distFor$ (line 4 in Algorithm 1) through the message propagation with piggybacked "distance" information. Since the data size of the piggyback is very small, the additional cost of the communication is insignificant.

The more frequent broadcasts are, the faster the values of $distFor$ follow the exact values for the dynamic network, and the more accurate tree selection becomes. On the other hand, the more the network changes, the more there is a delay in following the values, and the less accurate tree selection becomes.

Finally, for joining and leaving of nodes, a node provides a function receiving `NeighborDown`, a function receiving `NeighborUp`, and a function receiving `DistUpdate`. `NeighborUp` and `NeighborDown` are events issued by a peer sampling service and update information for neighbor nodes. Moreover, the protocol also locally computes the values of $t.distFor$. An extended version of this paper will provide a pseudocode.

V. Evaluation

We conducted experiments by simulating broadcasts on an overlay network, and measured the path lengths and the number of messages. We developed the simulator by ourselves.

A. Experiment settings

The simulating procedure for a combination of a method and an overlay network is as follows:

1) We generate an overlay network $G=(V,E)$, on which spanning trees are built.

2) We construct spanning trees if a method requires them.
3) Let the following procedure be 1 cycle, and we execute 1000 cycles.
   a) We determine a source node $v_{start} \in V$ randomly.
   b) We issue a broadcast on the source node v_{start}.
   c) We measure the path lengths and the number of messages.
   d) In Plumtree and the proposed method, we update tree topologies by following each method.

We use the following graphs as the overlay network, on which spanning trees are built. The number of nodes for each graph is $|V| = 10000$.

`Random Graph`
: A random graph generated by Erdös-Rényi model [14]. In this experiment, we generate a graph of $\Gamma_{10000,50000}$ where $|V| = 10000$ and $|E| = 50000$. $\Gamma_{n,N}$ is a graph generation model whose N edges are randomly selected from $\binom{n}{2}$ node pairs for the fixed number of nodes n.

`BA Model Graph`
: A graph generated by Barabási-Albert model [15]. The model generates a scale-free graph. In this experiment, $|V| = 10000$, and the number of edges added at each step of the generation is 5. Thus, $|E| \approx 50000$.

We use the following broadcast methods for comparison.

`Plumtree`

`proposed`
: The proposed method. The number of trees is 10.

`ideal proposed`
: A broadcast method where every tree selection for the proposed method is ideal. This means that a source node always selects a tree minimizing the maximum path length. The number of trees is 10.

`multiple Plumtrees`
: A method that manages 10 Plumtrees and broadcasts to all of them. The path length is the minimum path length among all the trees. The number of messages is approximately 10 times larger than `Plumtree`.

`flooding`

Since we assume that there are multiple source nodes for broadcasts, we set the value *threshold* used in the topology optimization of Plumtree protocol to 7.

B. Experiment result

Fig. 2 and Fig. 3 show the transitions of the maximum path lengths and the average path lengths of broadcasts for `Random Graph`, respectively. The translucent lines represent the measured values, and the opaque lines represent the moving average value for 50 cycles. The same representation is used in the following figures showing the transition of path lengths. We use moving averages to facilitate comparison of the relative values for each broadcast method. As a result of Fig. 2, the proposed method `proposed` reduces the maximum path length by approximately 28% compared with `Plumtree`.

TABLE I
REDUCTION RATIO OF MAXIMUM PATH LENGTH BY PROPOSED METHOD COMPARED TO PLUMTREE.

Overlay network	Reduction ratio
`Random Graph`	28%
`BA Model Graph`	7%

However, it is not as short as that of `ideal proposed`, which suggests that the update of values *distFor* does not sufficiently follow the topology changes by Plumtree protocol. In addition, the maximum path length of `ideal proposed` is almost the same as that of `multiple Plumtrees`. This suggests that the proposed method can potentially achieve the same length as the maximum path length when using multiple Plumtrees despite the number of messages on using a single Plumtree if the values *distFor* follows the topology changes. Furthermore, as a result of Fig. 3, the magnitude relationship of the average path length of each broadcast method tends to be almost the same as that of the maximum path length.

Fig. 4 shows the relation between the maximum path length and the number of messages for `Random Graph`. We plotted the results for 901–1000 simulation cycles. As a result, the numbers of messages of `Plumtree`, `proposed`, and `ideal proposed` are approximately 10000 ($\approx |V|$), which are much smaller than those of `flooding` and `multiple Plumtrees`, and especially, are approximately 11% of that of `flooding`. This is because, in the former, most messages are sent on a single spanning tree and there are few duplicate messages, while in the latter, many duplicate messages occur due to the property of the methods. Therefore, the experimental results for `Random Graph` show that the proposed method shortens path lengths compared to `Plumtree` while maintaining the small number of messages.

Fig. 5 and Fig. 6 show the transitions of the maximum path lengths and the average path lengths of broadcasts for `BA Model Graph`, respectively. As a result of Fig. 5, the proposed method `proposed` reduces the maximum path length by approximately 7% compared to `Plumtree`. In addition, it achieves almost the same maximum path length as that of `ideal proposed`, which suggests that the update of values *distFor* significantly follows the topology changes by Plumtree protocol for `BA Model Graph`.

Fig. 7 shows the relation between the maximum path length and the number of messages for `BA Model Graph`. We plotted the results for 901–1000 simulation cycles. It presents a similar result to Fig. 4 on the number of messages.

Table I summarizes the reduction ratio of maximum path length by the proposed method compared to Plumtree. Contribution of the proposed method could be much on a random graph, and moderate on a scale-free network generated by a BA model, that is natively efficient for broadcasting due to its small diameter.

VI. CONCLUSION

In this paper, we proposed a novel broadcast method. It constructs multiple spanning trees on the overlay network, and a source node selects an appropriate tree from them

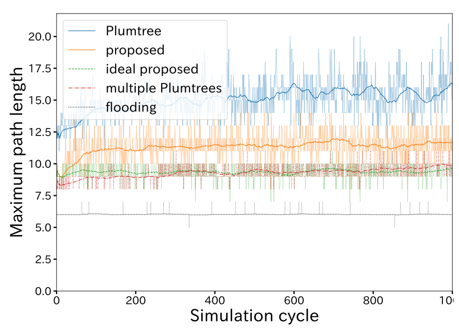


Fig. 2. Maximum path length for `Random Graph`.

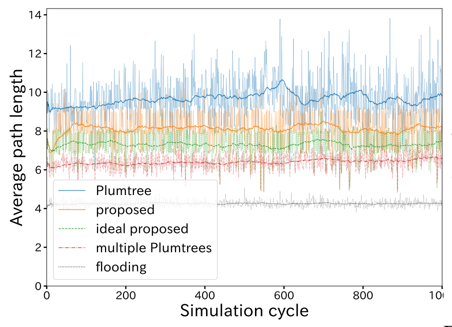


Fig. 3. Average path length for `Random Graph`.

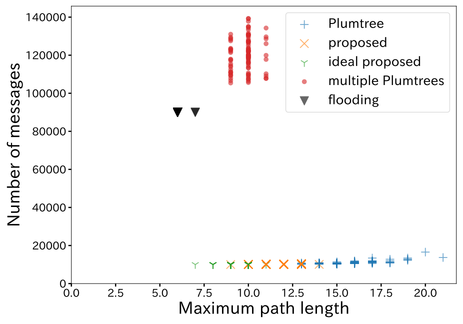


Fig. 4. Relation between the maximum path length and the number of messages for `Random Graph`.

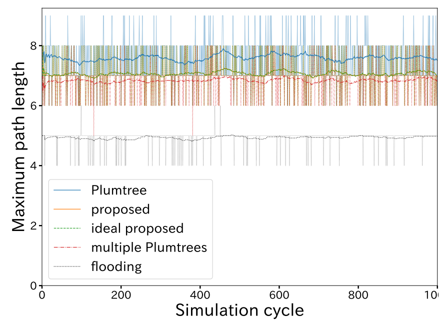


Fig. 5. Maximum path length for `BA Model Graph`.

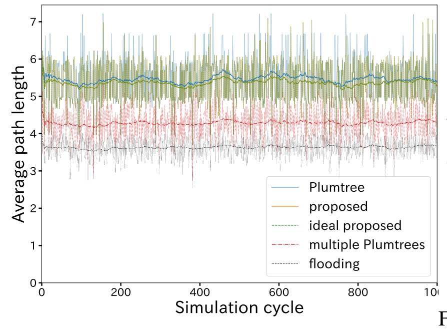


Fig. 6. Average path length for `BA Model Graph`.

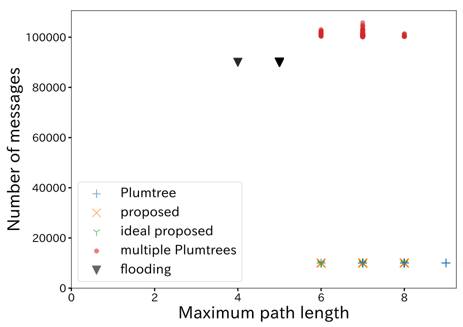


Fig. 7. Relation between the maximum path length and the number of messages for `BA Model Graph`.

when broadcasting. This reduces the frequency of inefficient broadcasts for multiple source nodes. It thereby achieves shortening routing path lengths while maintaining a small number of messages.

The evaluation experiments show that the effect of the proposed method on the path lengths is dependent on the topology of the overlay network. For graphs that closely resemble realistic network topologies, the paths tend to be shortened, especially for a random graph, with a reduction ratio of approximately 28% compared to Plumtree. Moreover, the number of messages was almost the same as the number of nodes. This shows that the proposed method shortens path lengths while maintaining the small number of messages.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number JP21H04872.

References

[1] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's Highly Available Key-value Store," in *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP '07. New York, NY, USA: ACM, 2007, pp. 205–220.

[2] N. Ramzan, H. Park, and E. Izquierdo, "Video streaming over P2P networks: Challenges and opportunities," *Signal Processing: Image Communication*, vol. 27, no. 5, pp. 401–411, May 2012.

[3] A. Yahyavi and B. Kemme, "Peer-to-peer Architectures for Massively Multiplayer Online Games: A Survey," *ACM Comput. Surv.*, vol. 46, no. 1, pp. 9:1–9:51, Jul. 2013.

[4] A. Dorri, S. S. Kanhere, and R. Jurdak, "Towards an Optimized BlockChain for IoT," in *Proceedings of the Second IEEE/ACM International Conference on Internet-of-Things Design and Implementation*, ser. IoTDI '17. New York, NY, USA: ACM, 2017, pp. 173–178.

[5] P. Ruiz and P. Bouvry, "Survey on Broadcast Algorithms for Mobile Ad Hoc Networks," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, pp. 8:1–8:35, Jul. 2015.

[6] J. Leitao, J. Pereira, and L. Rodrigues, "Epidemic Broadcast Trees," in *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*, Oct. 2007, pp. 301–310.

[7] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth multicast in cooperative environments," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 298–313, Oct. 2003.

[8] V. Venkataraman, K. Yoshida, and P. Francis, "Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast," in *Proceedings of the 2006 IEEE International Conference on Network Protocols*, Nov. 2006, pp. 2–11.

[9] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Middleware 2001*, ser. Lecture Notes in Computer Science, R. Guerraoui, Ed. Springer Berlin Heidelberg, 2001, pp. 329–350.

[10] M. Ferreira, J. Leitão, and L. Rodrigues, "Thicket: A Protocol for Building and Maintaining Multiple Trees in a P2P Overlay," in *2010 29th IEEE Symposium on Reliable Distributed Systems*, Oct. 2010, pp. 293–302.

[11] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Transactions on Computer Systems*, vol. 25, no. 3, pp. 8–es, Aug. 2007.

[12] J. Leitao, J. Pereira, and L. Rodrigues, "HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, Jun. 2007, pp. 419–429.

[13] M. Al Hasan, "Methods and Applications of Network Sampling," in *Optimization Challenges in Complex, Networked and Risky Systems*, ser. INFORMS TutORials in Operations Research. INFORMS, Oct. 2016, ch. 5, pp. 115–139.

[14] P. Erdös and A. Rényi, "On random graphs I." *Publicationes mathematicae*, vol. 6, no. 26, pp. 290–297, 1959.

[15] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.