# $K$-tree Trunk and a Distributed Algorithm for Effective Overlay Multicast on Mobile Ad Hoc Networks

Yamin Li and Shietung Peng
Department of Computer Science
Hosei University
Tokyo 184-8584 Japan
{yamin, speng}@k.hosei.ac.jp

Wanming Chu
Department of Computer Hardware
University of Aizu
Aizu-Wakamatsu 965-8580 Japan
w-chu@u-aizu.ac.jp

## Abstract

*Overlay multicast protocols construct a virtual mesh spanning all member nodes of a multicast group. It employs standard unicast routing and forwarding to fulfill multicast functionality. The advantages of this approach are robustness and low overhead. However, efficiency and stability are the issues that must be addressed in the mobile ad hoc network (MANET) environment. In this paper, we propose an effective structure for overlay multicast to solve these problems in MANET. Instead of using a spanning tree on the virtual mesh, we introduce a simple structure called $k$-tree trunk for multicast. A $k$-tree trunk of a tree is a subtree with $k$ leaves that minimizes the sum of the distances of all vertices to the subtree plus the size of the subtree. The $k$-tree trunk is more stable and easier to maintain than the spanning tree in MANET. The simulation results show that our approach handles the flexibility and mobility issues in an overlay multicast protocol effectively, especially when the group size is large.*

**Keywords.** *Mobile ad hoc network (MANET), multicast, overlay mesh, tree-core, efficiency, stability.*

## 1 Introduction

Mobile ad hoc network (MANET) refers to a form of infrastructureless network connecting mobile devices with wireless communication capacity. Each node in MANET behaves as a router as well as an end host, so that the connection between any two nodes is a multi-hop path supported by other nodes. In MANET, the multicast support is critical since the close cooperation among team members is required for many MANET applications.

Multicasting in MANET faces many challenges due to the continuous changes in network topology (mobility) and limited channel bandwidth. Many multicast routing protocols have been proposed for MANET [1, 7, 6, 2, 4, 9, 10, 11, 12, 3]. For multicast protocols, robustness and overhead are key issues since the protocols maintain state information at all nodes involved — both member nodes and non-member nodes that act as routers for supporting the multicast session.

Most multicast research for ad hoc networks has focused on IP layer multicast protocols. Such protocols require the cooperation of all the nodes of the network. Application layer multicasting (overlay multicasting) is an alternative approach to IP layer multicasting. The overlay multicast has the following advantages: First, it does not require changes at the network layer; second, routing complications are hidden; and third, intermediate nodes do not have to maintain per group state for each multicast group. However, the use of application layer multicast can result in the transmission of multiple copies of multicast messages over each physical link. This effect is especially visible when the number of multicast group members is large.

In the overlay multicast approach for MANET, a virtual infrastructure is built to form an overlay network on top of the physical network. Each link in the virtual topology is a unicast path in the physical network. The overlay network implements multicast functionalities such as dynamic membership maintenance, packet duplication and multicast routing. AMRoute [3] is an ad hoc multicast protocol that uses the overlay multicast approach. The protocol does not need to track the network mobility since it is handled by the underlying unicast protocols. Thus, it can operate seamlessly on multiple domains that use different unicast routing protocols [9].

To handle the efficiency issue in overlay multicast approach, minimum cost spanning tree on the virtual mesh is built. The cost of constructing and maintaining the tree depends very much on the size of the tree. For this reason, the overlay multicast approach works well for small groups but the performance degrades rapidly when the group size grows. In this paper, we propose an effective structure called $k$-tree trunk, for the overlay multicast on the virtual mesh. The selection of the value $k$ largely depends on the size of the spanning tree at hand. A small $k$ is enough practically for most of the networks and the communication groups. The $k$-tree trunk significantly reduces the cost for the maintenance and provides higher stability under the mobile environment.

The rest of the paper is organized as follows. Section 2 presents $k$-tree trunk, a new structure for the multicast on virtual mesh. Section 3 gives an distributed algorithm for finding a $k$-tree trunk. Section 4 provides simulation results on the performance of multicasting on the $k$-tree trunk and compares these results to those on the minimum spanning tree. Section 5 concludes this paper.

## 2 A $K$-Tree Trunk for Overlay Multicast

As mentioned in the previous sections, overlay multicasting protocol is an application layer protocol that constructs an overlay multicast tree of logical links among the group members. For small group this approach works well. However, as the size of the group grows, the maintenance and update of the multicasting tree will become costly and inefficient. Instead of using a spanning tree, we use $k$-tree trunk for multicasting on the virtual mesh. This approach is beneficial when the multicast group is large.

A $k$-tree trunk of a tree is a subtree with exactly $k$ leaves in the tree that minimizes the total cost of the multicast communication (to be defined later) among all subtrees with $k$ leaves. The $k$-tree trunk is a simple infrastructure to support the overlay multicast in mobile ad hoc networks at application level. A structure called $k$-tree core in tree networks had been proposed and studied by Peng et al [5]. However, in $k$-tree core, the object function to be minimized is different with the $k$-tree trunk defined in this paper.

Let $G$ be an edge-weighted graph with vertex set $V(G)$. Each edge $e = (u, v)$ has a weight $w(e)$, or $w(u, v)$, where nodes $u$ and $v$ are neighbors connected with edge $e$. Let $G'$ be a connected subgraph of $G$, we define the $w(G') = \sum_{e \in G'} w(e)$. For an any node $u \in V(G)$, we define $d(u, G') = \min\{d(u, v) | v \in V(G')\}$ and $\delta(G') = \sum_{u \in V(G)} d(u, G')$ where $d(u, v)$ is the distance between nodes $u$ and $v$. Then, we define $\gamma(G') = w(G') + \delta(G')$. In this paper, we consider $\gamma(G')$ as the object function that we want to minimize.

In the wireless ad hoc networks, we can consider $w(G')$ as an *inner cost* and $\delta(G')$ as an *outer cost* for the overlay multicast using $G'$ as infrastructure. For example, if Steiner tree $T$ of the virtual mesh is used then $\delta(T) = 0$, and in the case of stateless networks (no infrastructure), we have $G' = \{u\}$ and $w(G') = 0$. The $k$-tree trunk has a simpler structure than the spanning tree (only $k$ leaves) and the total cost $\gamma(G')$ for the multicast is minimized among subtrees with $k$ leaves.

Let $T$ be an edge-weighted tree with vertex set $V(T)$ and edge set $E(T)$. Each $e \in E(T)$ has a weight $w(e) > 0$. Let $T_k$ be a subtree of $T$ with $k$ leaves. Let $F$ be the set of all $T_k$ of $T$. Then a $k$-tree trunk is a $T_k$ that minimizes $\gamma(T_k) = w(T_k) + \delta(T_k)$ among all $T_k$ in $F$, where $w(T_k) = \sum_{e \in E(T_k)} w(e)$, $\delta(T_k) = \sum_{u \in V(T)} d(u, T_k)$, and $d(u, T_k) = d(u, v) | v \in V(T_k)$. A 2-tree trunk is a path called tree trunk shortly. Finding a tree trunk is the base for finding $k$-tree trunk for $k > 2$.

The multicast can be performed using $k$-trunk as follows. If the source node is on the trunk, it broadcast the message to all the nodes on the trunk. Then, each node on the trunk send the message to its non-trunk nodes by unicasting. If the source node is not on the trunk, it unicasts the message to its trunk node first. The $k$-trunk can be more stable than the tree. For example, if a non-trunk node quits from the group, there is no affect on the connectedness of the $k$-trunk; but for the tree, a node quit may separate the tree to two unconnected parts.

To find a tree trunk, we first orient tree $T$ into a rooted tree $T_r$ with root $r$. For any vertex $v \in T_r$, we denote the parent of $v$ as $p(v)$, the subtree rooted at $v$ as $T_v$, and the number of vertices in $T_v$ as $|T_v|$. Let a rooted trunk $P(r, l_0)$ be a path from root $r$ to leaf $l_0$ which minimizes $\delta(P(r, l)) + w(P(r, l))$ among all paths from $r$ to leaf $l$ in $T_r$. We show that the problem of constructing a tree trunk in $T$ can be reduced to the problem of constructing a rooted trunk in a rooted tree $T_r$. The following lemma forms the theoretical background for the reduction.

**Lemma 1** Let rooted tree $T_r$ be an orientation of $T$ and $P(r, l_0)$ a rooted trunk in $T_r$. Then $P(r, l_0) \cap P(l_1, l_2) \neq \emptyset$ for any trunk $P(l_1, l_2)$ in $T$.

**Proof:** Assume that $P(r, l_0) \cap P(l_1, l_2) = \emptyset$ for a trunk $P(l_1, l_2)$. Let $i$ be the closest vertex in $P(r, l_0)$ to $P(l_1, l_2)$ and $j$ the closest vertex in $P(l_1, l_2)$ to $P(r, l_0)$. Let path $C = P(l_0, i) \cup P(i, j) \cup P(j, l_2)$. Since $P(r, l_0)$ is a rooted trunk, $\delta(P(l_0, i)) + w(P(l_0, i))) \leq \delta(P(l_1, i)) + w(P(l_1, i))$. Since $i$ is not a leaf, we have $\delta(P(l_1, i)) + w(P(l_1, i)) < \delta(P(l_1, j)) + w(P(l_1, j))$. Similar, we have $\delta(P(l_0, j)) + w(P(l_0, j)) < \delta(P(l_0, i)) + w(P(l_0, i))$. From these equations, we get $\delta(P(l_0, j)) + w(P(l_0, j)) < \delta(P(l_1, j)) + w(P(l_1, j))$. This implies $\delta(C) + w(C) < \delta(P(l_1, l_2)) + w(P(l_1, l_2))$, a contradiction to the fact that $P(l_1, l_2))$ is a trunk. Therefore, the lemma must be true. ❏

**Theorem 1** Let rooted tree $T_r$ be an orientation of $T$ and $P(r, l_0)$ a rooted trunk in $T_r$. Then a rooted trunk in rooted tree $T_{l_0}$, a new orientation of $T$, is a trunk in $T$.

**Proof:** Let $P(l_0, l_0')$ be a rooted trunk in $T_{l_0}$. Assume that $P(l_1, l_2)$ is a trunk in $T$. From Lemma 1, $P(l_0, l_0') \cap P(l_1, l_2) \neq \emptyset$. Let $P(i, j) = P(l_0, l_0') \cap P(l_1, l_2)$, where

$i$ is the vertex in $P(i,j)$ closest to vertices $l_0$ and $l_1$. Since $P(r,l_0)$ is a rooted trunk, we have $\delta(P(l_0,i)) + w(P(l_0,i)) \leq \delta(P(l_1,i)) + w(P(l_1,i))$. Similarly, Since $P(l_0,l_0')$ is a rooted trunk, we have $\delta(P(l_0',j)) + w(P(l_0',j)) \leq \delta(P(l_2,j))+w(P(l_2,j))$. Therefore, we get $\delta(P(l_0,l_0')) + w(P(l_0,l_0')) \leq \delta(P(l_1,l_2)) + w(P(l_1,l_2))$. We conclude that $P(l_0,l_0')$ is a trunk in $T$. ❑

Next, we introduce trunk partition of a rooted tree $T_r$. A *trunk partition* of $T_r$, denoted as $\Gamma(T_r)$, can be defined recursively as follows. For a node $v \in T_r$, let $v_1,\ldots,v_q$ be the children of $v$. Let $P_{v_i,l_i}$ be a rooted trunk of $T_{v_i}$. We define the trunk partition of $T_v$, $\Gamma(T_v)$ as follows:

$$\Gamma(T_v) = \cup_{i=1}^{q}(\{P_{v,l_i}\} \cup \Gamma(T_{v_i} - \{P_{v_i,l_i}\})$$

. From the definition, $\Gamma(T_v)$ can be found if rooted trunk of node $u$, for all $u \in T_v$, has been computed. It can be verified easily that trunk partition of $T_r$ is a partition of $T$ into edge-disjoint paths $P_i,\ 1 \leq i \leq m$. That is,

$$\Gamma(T_r) = \{P_i,\ 1 \leq i \leq m\},\ \cup_{i=1}^{m} P_i = T$$

**Theorem 2** Let rooted tree $T_r$ be an orientation of $T$ and $P(r,l_0)$ a rooted trunk in $T_r$. Let $\Gamma(T_{l_0}) = \cup_{i=1}^{m}\{P_i\}$, where $\gamma(P_i) \geq \gamma(P_j)$ if $i \geq j$, be a trunk partition of the rooted tree $T_{l_0}$, a new orientation of $T$. Then, $T_k = \cup_{i=1}^{k-1} P_i$ is a $k$-tree trunk in $T$.

**Proof:** We prove the theorem by induction on $k$. For $k=2$, from Theorem 1, the theorem is true. Let $P_1 = P(l_0,l_1)$ and $P_i = P(v_i,l_i)$ for $i > 1$. Let $T_k'$ be a $k$-tree trunk, $k > 2$, in $T$. By the induction assumption, for any subtree $T_{k-1}'$ of $T_k'$ with $k-1$ leaves, we have $\gamma(T_{k-1}) \geq \gamma(T_{k-1}')$. Without loss of generality, we can assume that $l_i,\ 0 \leq i \leq k-2$, are the leaves of $T_k'$. Let $l_s \neq l_i,\ 0 \leq i \leq k-2$, be the leaf in $T_k'$. Assume that $P(l_s,w)$ be the path that is edge-disjoint with $T_{k-1}$. From the definition of trunk partition, we have $\gamma(P(l_{k-1},v_{k-1})) \leq \gamma(P(l_s,w))$. Therefore, $\gamma(T_k) \leq \gamma(T_k')$. We conclude that $T_k$ is a $k$-tree trunk in $T$. ❑

In the next section, we will introduce an efficient distributed algorithm that can perform tree orientation, finding rooted trunk, and a trunk partition altogether. The algorithm uses only local information and communication is done by asynchronous message passing.

## 3 A Distributed Algorithm for Finding a $K$-Tree Trunk

We propose a distributed algorithm for finding a $k$-tree trunk of a tree $T$ in this section. The algorithm is based on *branch-cut* operation.

For a given root node $r$, the branch-cut operation works inward from leaves ($\neq r$). The branch-cut operation first identifies *candidates*. A node $u \neq r$ is a candidate if the following conditions are satisfied: (1) $u$ is a nonleaf node; and (2) if $r \notin N(u)$ exactly one nonleaf $v \in N(u)$, otherwise, $N(u) - \{r\}$ are all leaves. The root $r$ is a candidate if all its neighbors are leaves. If $u$ becomes a candidate then branch-cut is performed on $u$; the neighbors of $u$ that are leaves are cut-off from the tree and $u$ becomes a leaf (for tree orientation, we set all edges connecting $u$ and its leaf neighbors the direction toward $u$). The *branch* $B(u)$ is a subtree in $T$ that includes all edges oriented toward $u$ or its descendants through branch-cut. Figure 1 depicts a tree $T_r$ that contains a candidate $u$. Note that $w$ is not a candidate due to $r \in N(w)$, although it has only one nonleaf neighbor $v$.

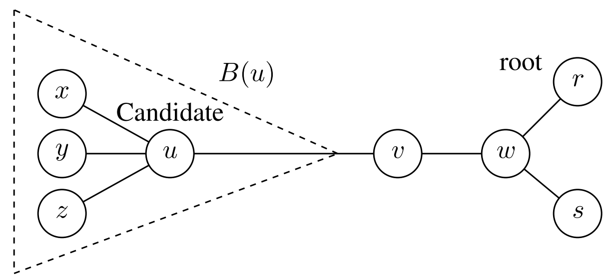


**Figure 1. Candidates in tree $T_r$**

Through branch-cut operation, the rooted trunk and trunk partition of the branch $B(u)$ with root $u$ are calculated and saved in $u$. Since all candidates that are not root $r$ calculate the disjoint local trunks for different branches at the same time, the algorithm inherits natural parallelism. In a distributed environment, global clock and global information are not available, so branch-cut operation should be done asynchronously, and based on the local information only.

To find the rooted trunk based on branch-cut, if we use the formula $w(P) + \delta(P)$ directly, $\delta(l)$, for all leaves $l$ of tree $T$, should be calculated first. However, calculating the value of $\delta(l)$ requires global information. To overcome this problem, we define *cost saving* that needs local information only. The cost saving of a path from a leaf $l$ to node $v$, denoted as $C_s(P(l,v))$, is defined as follows:

$$C_s(P(l,v)) = \delta(v) - \delta(P(l,v)) - w(P(l,v)) \quad (1)$$

Now, from the definition of rooted trunk, to find a rooted trunk in branch $B(u)$ equals to find a path $P(l,u)$ in $B(u)$ such that $C_s(P(l,u))$ is maximized. It is the key in the design of distributed algorithm for finding rooted trunk based on branch-cut in which $C_s(P(l,u))$ can be computed using local information only. The formula for computing cost saving while extending path from $v$ to $u$ is

$$C_s(P(l,u)) = C_s(P(l,v)) + (|B(u)| - 1) \times w(u,v) \quad (2)$$

where $v$ is a child of $u$ and $v \in P(l,u))$.

The cost saving of a local rooted trunk in $B(u)$, denoted as $C_s(u)$ is defined as

$$C_s(u) = \max_{l \in B(u)} C_s(P(l, u)) \tag{3}$$

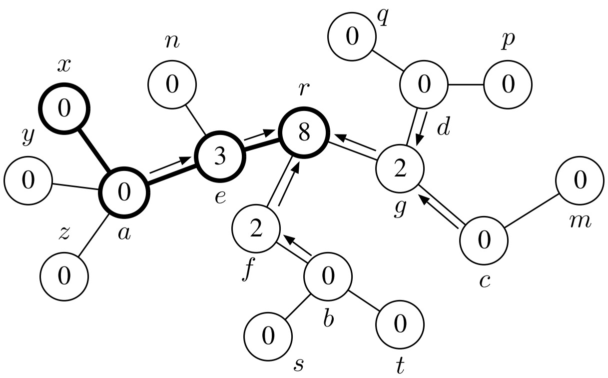


**Figure 2. Calculation of cost saving during branch-cut**

As shown in Figure 2, given tree $T$ and root $r$, four nodes $a$, $b$, $c$, and $d$ are identified as candidates. We perform branch-cut and these four nodes become leaves with $C_s(a) = C_s(b) = C_s(c) = C_s(d) = 0$. Next, nodes $e$, $g$, and $f$ are identified as candidates. We perform branch-cut and calculate the cost savings $C_s(e) = 0 + (4-1) \times 1 = 3$, and $C_s(f) = C_s(g) = 2$ in parallel by using Equations 2 and 3. Finally, since all three neighbors of node $r$ are leaves, we perform branch-cut at $r$ and calculate three cost saving from three branches. They are $C_s(P(x,r)) = 8$, $C_s(P(s,r)) = 5$, and $C_s(P(q,r)) = 7$, respectively. Therefore $P(x,r)$ is selected as rooted trunk.

---

Algorithm 1: Finding_K_Tree_Trunk
**Input:** A weighted tree $T$ and $k$
**Output:** A $k$-tree trunk $T_k$
**begin**
1. Orient tree $T$ into a rooted tree $T_r$ with an arbitrary vertex $r$;
2. Construct a rooted trunk $P(r, l_0)$ in $T_r$;
3. Re-orient $T$ into $T_{l_0}$;
4. Construct a trunk partition of $T_{l_0}$, $\Gamma(T_{l_0})$;
5. Find the path $P_{k-1}$ such that $C_s(P_{k-1})$ is the $(k-1)$th largest among the paths in $\Gamma(T_{l_0})$;
6. Return $T_k = \cup_{i=1}^{k-1} P_i$, where $P_i \in \Gamma(T_{l_0})$ and $C_s(P_i) \geq C_s(P_j)$ if $i < j$

**end**

---

The algorithm for finding a $k$-tree trunk is given in Algorithm 1. Algorithm 2 is a procedure for finding a rooted trunk and a trunk partition (and tree orientation) in which we use the local information to compute the following four variables in each node $u$:

---

Algorithm 2: Trunk_Partition
**Input:** A weighted tree $T$ and a root $r$
**Output:** A rooted trunk and a trunk partition of $T_r$
**begin**
  $u$ = my_node_id;
  $u.size = 1$;
  $u.saving = 0$;
  $u.path = \{u\}$;
  $u.parti = \emptyset$;
  $n = degree(u)$;
  $L = N(u)$;   /* $N(u)$ *is the set of neighbor nodes of* $u$ */
  **if** $(n = 1)$ and $(u \neq r)$   /* *a leaf* */
    **send** Message($u.size, u.saving, u.path, u.parti$) to $v \in L$;
    **exit**();
  **else**
    **while** (true)
      **receive** Message $(v.size, v.saving, v.path, v.parti)$ from $v \in L$;
      $n = n - 1$;
      $L = L - \{v\}$;
      $u.size = u.size + v.size$;
      **if** $(u.saving < v.saving + (v.size - 1) \times w((u : v)))$
        $u.saving = v.saving + (v.size - 1) \times w((v : u))$;
        $u.path = u.path \cup (u : v)$;
        $u.parti = u.parti \cup v.parti \cup \{v.path \cup (u : v)\} - \{v.path\}$;
      **endif**
      **if** $(n = 1)$ and $(u \neq r)$   /* *branch-cut* */
        **send** Message($u.size$, $u.saving$, $u.path$, $u.parti$) to $v \in L$;
        **exit**();
      **endif**   /* *my_node_id finish* */
      **if** $(n = 0)$   /* *u is root* */
        **return** $(u.path, u.parti)$;
        **exit**();   /* *Trunk partition found* */
      **endif**
    **endwhile**
  **endif**
**end**

---

1. The number of nodes in $B(u)$, denoted as $u.size$.
2. The cost saving $C_s(u)$, denoted as $u.saving$.
3. The local rooted trunk in $B(u)$, denoted as $u.path$ ($C_s(u.path) = C_s(u)$).
4. The trunk partition of branch $B(u)$, denoted as $u.parti$.

**Theorem 3** Algorithm 2 finds a rooted trunk and a trunk partition of $T$ with root $r$ in $T_r$ in $O(d)$ time, where $d$ is the diameter of $T$, assuming that the degree of node $v \in T$, $deg(v) = O(1)$.

**Proof:** From the initial values assigned to leaves and the iteratively extending formula 2, it can be verified easily that the $u.saving = C_s(u)$. Next, When node $u \neq r$ becomes a leaf ($|L| = n = 1$), it sends a message to the only node $v$ left in $L$ that is either a nonleaf node or the root and exits. Therefore, the message from $u$ to $v$ is sent only once and no message will be sent from $v$ to $u$. That is, there is no conflict during asynchronous communication. Since the message is sent from $u$ ($u$ is cut-off) only if $u \neq r$, root $r$ will receive message from its neighbor only and the edges are oriented from its neighbors toward $r$. Therefore, root $r$ must be the last node remained during the branch-cut process.

Next, when node $u$ sends message to node $v$, $u$ is cut-off and the trunk partition of $T_u$ should be extended to $v$. Therefore, the algorithm updates the current $v.parti$ by adding $u.parti \cup \{u.path \cup (u : v)\} - \{u.path\}$. It can be seen easily that when $v \neq r$ becomes a leaf, $v.parti$ will be the trunk partition of $T_v$. When $r$ becomes a single node ($n = 0$), $r.parti$ is a trunk partition of $T_r$. The running time of the algorithm is $O(h) = O(d)$, $h$ is the height of the rooted tree $T_r$, since computing and communication time for each node $u$ is a constant assuming that $deg(u) = O(1)$. ❑

Next, we shows in Theorem 4 that finding a $k$-tree trunk can be done efficiently in a distributed environment using local information only.

**Theorem 4** Given a weighted tree $T$, there exists a distributed algorithm that finds a $k$-tree trunk in $T$ in $O(d)$ time, assuming that the degree of node $v \in T$, $deg(v) = O(1)$ and local computations take $O(1)$ time.

**Proof:** From Theorems 1 and 2, we know that algorithm 1 finds a trunk and a $k$-tree trunk correctly. From Theorem 3, steps 1 - 4 of algorithm 1 can be done efficeintly in $O(d)$ time. Next, steps 5 - 6 of Algorithm 1 that find the $(k-1)$th largest number in $P_i$, $1 \leq i \leq m$, and the union of the largest $k-1$ paths in the trunk partition can be done locally at root node. Therefore, the running time of Algorithm 1 is $O(d + n)$. ❑

## 4 Performance Analysis and Simulations

The network for the performance simulation is configured as below. There are 200 nodes randomly roaming within a 2000m × 1500m area. The radio transmission range of each node is set to be 350m, 450m, and 550m. The group size is chosen to be 10 to 100, stepped by 10. Each configuration runs 100 trials.

Figure 3 shows the trunk size — the number of nodes of trunk. The trunk size is relatively small compared to the multicast group size. Also, increasing the radio transmission range reduces the trunk size.

Figure 4 shows the message delivery cost. The message delivery cost here is simply defined as the sum of physical hop length of virtual links of the trunk when a message is

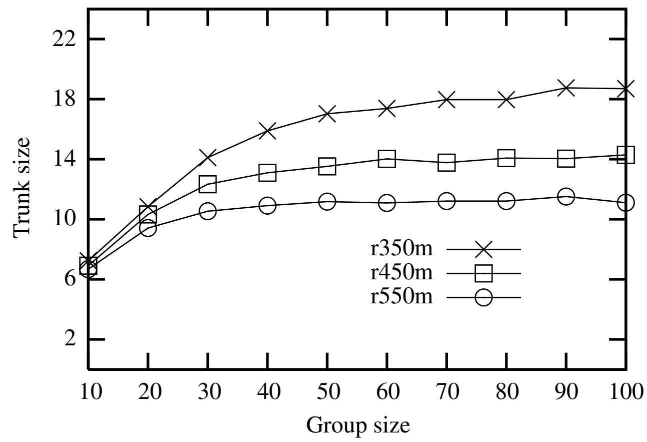


**Figure 3. Average trunk size**

multicasted to all the group members. Increasing the radio transmission range will decrease the cost but the effect is not obvious.

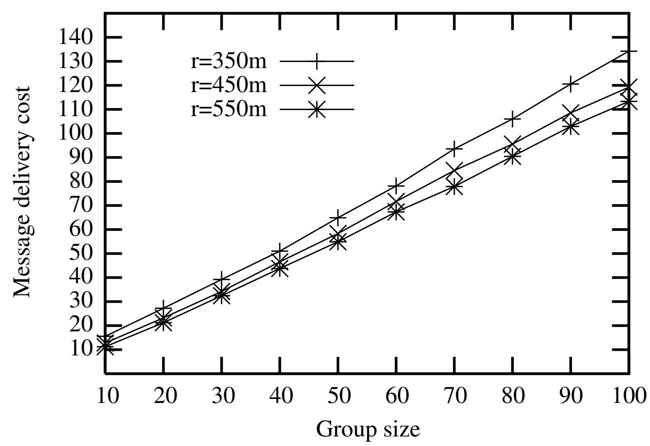


**Figure 4. Average cost**

Table 1 lists the message delivery costs of trunk, 3-tree trunk, 4-tree trunk, 5-tree trunk, and AMRoute with the radio transmission range of 350m. The table also lists the cost for stateless transformation in which the message is sent to every member individually by unicast routing.

Figure 5 depicts the costs listed in the table. Trunk structure maintains fewer nodes than AMRoute. By simply increasing $k$, the message delivery cost is closed to the optimal cost of tree. From our simulation, we conclude that the $k$-tree trunk with $k = 3$ or $4$ provides better maintenance-cost/performance.

The virtual trunk remains static even though the underlying physical topology is changing. We also investigated the mobility effect on the message delivery cost. The movement of each node follows the random waypoint

**Table 1. Average cost**

| #Mem | Stateless | Trunk | Tree | 3-Trunk | 4-Trunk | 5-Trunk |
|---|---|---|---|---|---|---|
| 10 | 30.14 | 15.55 | 14.95 | 14.91 | 14.66 | 14.43 |
| 20 | 63.69 | 27.19 | 24.42 | 25.18 | 24.58 | 24.43 |
| 30 | 101.0 | 39.24 | 32.90 | 34.97 | 33.42 | 33.07 |
| 40 | 135.2 | 50.95 | 40.82 | 44.82 | 42.48 | 41.44 |
| 50 | 167.6 | 64.94 | 49.93 | 56.04 | 52.89 | 51.30 |
| 60 | 192.3 | 78.15 | 59.50 | 68.40 | 64.10 | 61.86 |
| 70 | 231.1 | 93.56 | 69.28 | 79.98 | 75.08 | 72.52 |
| 80 | 260.2 | 106.0 | 79.15 | 92.70 | 86.28 | 83.06 |
| 90 | 308.6 | 120.5 | 89.11 | 103.9 | 97.06 | 93.82 |
| 100 | 333.9 | 134.2 | 99.09 | 116.6 | 108.7 | 104.8 |

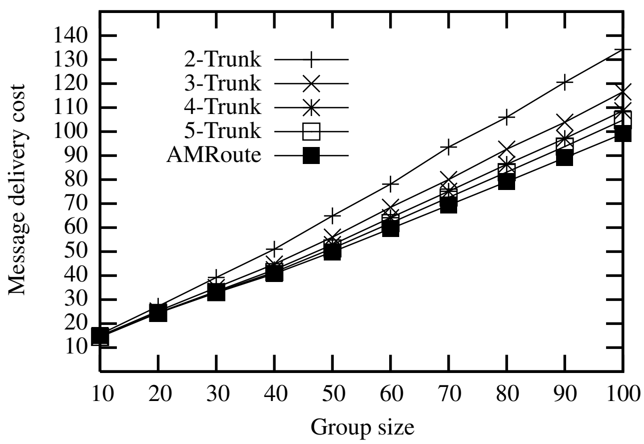


**Figure 5. Average cost**

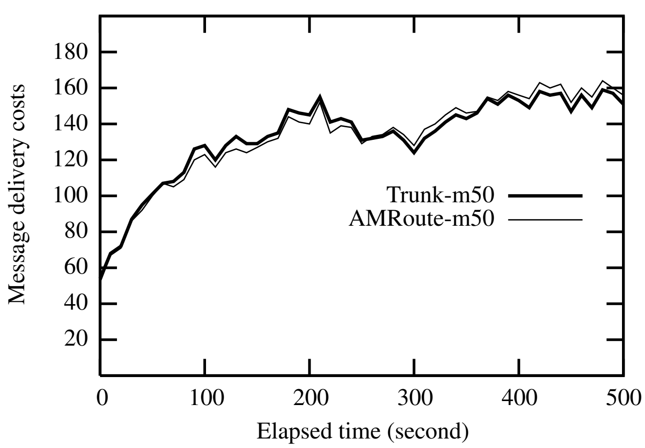


**Figure 6. Cost with mobility**

model [8]: Each node selects a destination location randomly and moves straight toward the destination with a constant speed which is uniformly distributed over [0,20] meters/second. After arrival, the node pauses for 10 second and then moves to another location, and so on.

Figure 6 shows the time-line of the costs of the AMRoute and the trunk for multicast group size 50 at the radio transmission range of 350m. As the member node moves, the message delivery costs of both the AMRoute and trunk increase. In practice, the trunk structure must be re-constructed periodically, like the AMRoute does.

## 5 Concluding Remarks

A new infrastructure called $k$-tree trunk for overlay multicasting on mobile ad hoc network was proposed and an efficient distributed algorithm for finding a $k$-tree trunk were given. Then we evaluated its performance through simulations. Our future work includes the investigating more precisely the influences of using the new structure on the performance under more realistic environments or larger networks as well as multicast groups. Other applications for $k$-tree trunk will also be a possible direction for future work.

## References


[1] K. Chen and K. Nahrstedt. Effective location-guided tree construction algorithm for small group multicast in manet. In *Proc. of IEEE INFOCOM'02*, June 2002.

[2] D. Janotti et al. Overcast: reliable multicasting with an overlay network. In *Proc. of the 4th Symposium on Operating System Design and Implementation*, Oct. 2000.

[3] J. Xie et al. Amroute: ad hoc multicast routing protocol. *ACM Mobile Networks and Applications*, 7(6), Dec. 2002.

[4] S. J. Lee et al. On-demand multicast routing protocol in multihop wireless mobile networks. *ACM Mobile Networks and Applications*, 7(6), Dec. 2002.

[5] S. Peng et al. Algorithms for a core and $k$-tree core of a tree. *Journal of Algorithms*, 15:143–159, 1993.

[6] M. Ge, S. V. Krishnamurthy, and M. Faloutsos. Overlay multicasting for ad hoc networks. In *Proc. of the Third Annual Mediterranean Ad Hoc Networking Workshop (Med-HocNet 2004)*, pages 131–143, June 2004.

[7] C. Gui and P. Mahapatra. Efficient overlay multicast for mobile ad hoc networks. In *Proc. of IEEE WCNC2003*, March 2003.

[8] David B. Johnson and David A. Maltz. *Dynamic source routing in ad hoc wireless networks*. Kluwer Academic Publishers, 1996.

[9] S. J. Lee and W. Su. Performance comparison study of ad hoc wireless multicast protocols. In *Proc. of IEEE INFOCOM'00*, Mar. 2000.

[10] R. Novak, J. Ruge, and G. Kandus. Steiner tree based distributed multicast routing in networks. *Steiner Trees in Industries*, 8(5):1–25, 2000.

[11] E. Royer and C. E. Perkins. Multicast operations of the adhoc on-demand distance vector routing protocol. In *Proc. of ACM MOBICOM'99*, Aug. 1999.

[12] C. W. Wu and Y. C. Tay. Amris: a multicast protocol for ad hoc wireless networks. In *Proc. of IEEE NILCOM'99*, Nov. 1999.