A Distributed and Efficient Flooding Scheme Using 1-Hop Information in Mobile Ad Hoc Networks

Hai Liu, Xiaohua Jia, *Senior Member*, *IEEE*, Peng-Jun Wan, Xinxin Liu, and Frances F. Yao

Abstract—Flooding is one of the most fundamental operations in mobile ad hoc networks. Traditional implementation of flooding suffers from the problems of excessive redundancy of messages, resource contention, and signal collision. This causes high protocol overhead and interference with the existing traffic in the networks. Some efficient flooding algorithms were proposed to avoid these problems. However, these algorithms either perform poorly in reducing redundant transmissions or require each node to maintain 2-hop (or more) neighbors information. In the paper, we study the sufficient and necessary condition of 100 percent deliverability for flooding schemes that are based on only 1-hop neighbors information. We further propose an efficient flooding algorithm that achieves the local optimality in two senses: 1) The number of forwarding nodes in each step is minimal and 2) the time complexity for computing forwarding nodes is the lowest, which is $O(nlogn)$, where n is the number of neighbors of a node. Extensive simulations have been conducted and simulation results have shown the excellent performance of our algorithm.

Index Terms—Flooding, broadcasting, mobile ad hoc networks, wireless networks.

1 Introductions

Flooding is one of the most fundamental operations in mobile ad hoc networks (MANETs). Most of the major routing protocols, such as DSR [1], AODV [2], ZRP [3], LAR [4], etc., rely on flooding for disseminating route discovery, route maintenance, or topology update packets. Flooding is a very frequently invoked utility function in MANETs. Therefore, an efficient implementation of flooding scheme is crucial in reducing the overhead of routing protocols and improving the throughput of networks.

Pure flooding, or blind flooding, was first discussed in [5], [6], where every node in the network retransmits the flooding message when it is its first time to receive it. This simple scheme guarantees that a flooding message can reach all nodes if there is no collision and the network is connected. However, it generates an excessive amount of redundant network traffic because all nodes in the network transmit the flooding message. This will consume a lot of the energy resources of mobile nodes and cause congestion of the network. Furthermore, due to the broadcast nature of radio transmissions, there is a very high probability of signal collisions when all nodes flood the message in the network at the same time, which would cause more retransmissions or some nodes failing to receive the message. This is the so-called broadcast storm problem [7]. Sinha et al. claimed that "in moderately sparse graphs, the expected number of nodes in the network that will receive a broadcast message was shown to be as low as 80 percent" in [8].

To solve the broadcast storm problem, several schemes have been proposed to reduce the redundancy in flooding operations. The most notable works are [9], [10], and [11]. However, these algorithms either perform poorly in reducing redundant transmissions or require each node to maintain 2-hop neighbor information. Maintaining 2-hop neighbor information for each node incurs extra overhead of the system and the information can hardly be accurate when the mobility of the system is high. In the paper, we propose an efficient flooding algorithm that is only based on 1-hop neighbors information, which makes the protocol easy to be implement and light weight in overhead. Our proposed algorithm also achieves local optimality in two senses: 1) The number of forwarding nodes is minimal and 2) the time complexity is the lowest. The time complexity for computing the forwarding nodes in each step is $O(nlogn)$, which is the lower bound (n is the number of neighbors of a node).

The efficient flooding scheme is different from the broadcast mechanisms discussed in [12], [13]. The broadcast mechanism is used for transmission of a large amount data or stream media data, which requires a broadcast routing to find an efficient route before the actual transmission of data so that data can be transmitted efficiently along the prefound route. In contrast, flooding is usually used for dissemination of control packets, which is a one-off operation. It does not need routing before hand.

2 Related Work

The existing efficient flooding schemes can be classified into three categories based on the information each node keeps:

- H. Lu, X. Jia, P.-J. Wan, and F.F. Yao are with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong. E-mail: {liuhai, jia, pwan}@cs.cityu.edu.hk, csfyao@cityu.edu.hk.
- X. Liu is with the School of Computer Science, Wuhan University, China. E-mail: whulxx@cityu.edu.hk.

Manuscript received 18 Nov. 2005; revised 7 Apr. 2006; accepted 5 Sept. 2006; published online 9 Jan. 2007.
Recommended for acceptance by J. Hou.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0481-1105.
Digital Object Identifier no. 10.1109/TPDS.2007.1023.

1) no need of neighbor information, 2) 1-hop neighbor information, and 3) 2-hop or more neighbor information.

Schemes in the first category do not need information on neighbors. A pure flooding scheme is a typical example in this category. The authors of [7], [14] showed the serious problem that pure flooding causes through analysis and simulations. A probabilistic-based scheme was further proposed to reduce redundant rebroadcasts and differentiate timing of rebroadcasts to avoid collisions. Upon receiving a flooding message for the first time, a node will forward it with probability P. Clearly, when $P = 1$, this scheme is equivalent to pure flooding. The probabilistic scheme includes counter-based, distance-based, location-based, and cluster-based flooding schemes. Simulation results showed different levels of improvement over pure flooding. This probabilistic scheme was further investigated in [15]. It showed that the success rate curve for probabilistic flooding tends to become linear for the network with low average node degree and resembles a bell curve for the network with high average node degree. In these schemes, a nonredundant transmission might be dropped out, without being forwarded further. This will cause some nodes in the network to fail to receive the flooding message (i.e., these nodes are not reached by the flooding). Besides this deliverability problem, another major concern of these techniques is the difficulties in setting the right threshold value (e.g., retransmission probability, etc.) in various network situations [16].

Schemes in the second category assume that each node keeps information of 1-hop neighbors. One-hop neighbor information can be obtained by exchanging the HELLO message in MAC layer protocols. A major issue in the schemes that use 1-hop or 2-hop information is the selection of a subset of neighbors for forwarding the flooding message. There are two strategies for choosing forwarding nodes: sender-based, where each sender nominates a subset of its neighbors to be the next hop forwarding nodes, and receiver-based, where each receiver of a flooding message makes its own decision on whether it should forward the message. Several flooding schemes that use 1-hop information and guarantee 100 percent deliverability were discussed in [10]. This work also analyzed the performance of the two strategies for choosing forwarding nodes. To avoid transmission collision, it also proposed a simple transmission order for forwarding nodes: A farther neighbor waits for a shorter time to forward a message after it receives it. The flooding with self-pruning (FSP) scheme proposed in [17] is a receiver-based scheme that uses 1-hop information. In this scheme, a sender forwards a flooding message by attaching all of its 1-hop neighbors to the message. A receiver compares its own 1-hop neighbors with the node list in the message. If all of its 1-hop neighbors are already included in the list, it will not forward the message; otherwise, it forwards the message as its sender. The work in [18] compared the performance of several flooding schemes. It showed that the improvement of FSP is very limited in most network conditions. Another notable work of efficient flooding that uses 1-hop neighbor information is Edge Forwarding [9]. For each node, its transmission coverage is partitioned into six equal-size sectors. A node,

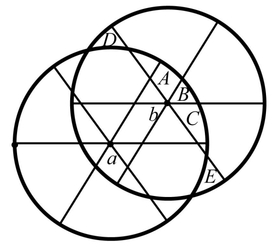


Fig. 1. Example of edge forwarding.

upon receiving a flooding message, makes its own decision whether it should forward the message based on the availability of other forwarding nodes in the overlapped areas. Taking an example in Fig. 1, node a, whose coverage disk is partitioned into six sectors, floods a message that is received by its neighbor b. Node b does not need to forward the message if and only if 1) there exist nodes in the small enclosed areas A, B and C, and 2) any nodes in areas D and E can be reached by the nodes in A and C, respectively. This is because the coverage disk of b can be covered by either a or the nodes in areas A, B, and C. By doing so, it reduces the forwarding nodes in flooding.

Most existing flooding schemes that use neighborhood knowledge are based on information of 2-hop neighbors. To obtain the information about 2-hop neighbors, one solution is that each node attaches the list of its own neighbor information to the HELLO message for exchange. The schemes proposed in [17], [19], [20], [21] are sender-based, while the schemes in [11], [22], [23], [24], [25], [26] are receiver-based. In the schemes that use 2-hop neighbor information, each node knows the network topology (connectivity) of 2-hop neighbors. To forward messages efficiently, the task for each node is to select the minimal subset of its 1-hop neighbors that can reach all of its 2-hop neighbors. A multipoint relaying method was proposed in [19], [20] which tries to find the minimal number of forwarding nodes among the neighbors. Finding the minimal number of forwarding nodes was proved to be NP-complete [20]. Authors proposed a heuristic algorithm that selects forwarding nodes at each step such that the number of newly covered neighbors is maximized. The approximation ratio of this heuristic algorithm was proved to be at most $logn$, where n is the number of 2-hop neighbors. Notice that this performance ratio is only for each step (i.e., for 2-hop neighbors), not for the entire network. Another important technique is the use of *connected dominating set* (CDS) [11], [27]. A *dominating set* (DS) is a subset of nodes such that every node in the graph is either in the set or is adjacent to a node in the set. A CDS is a connected DS. Any routing in MANETs can be done efficiently via CDS [11]. Although finding minimal CDS (MCDS) is NP-hard even in unit disk graph (UDG) [28], some distributed algorithms for computing MCDS with approximation ratio have been proposed in [27], [29]. However, maintaining a CDS in the network is costly, which is not suitable for flooding operations in highly mobile situations. Generally, the schemes that use 2-hop neighbor information incur high protocol overhead in the network with high mobility and high node density and they cannot be easily fitted into a network that does not support 2-hop neighbor information exchange.

Our flooding scheme requires each node to keep only 1-hop neighbor information, including their IDs and their geographic locations. The location information of each node can be obtained via GPS or some distributed localization methods [30] when GPS service is not available. We prove that our flooding scheme not only guarantees 100 percent deliverability, but also achieves *local optimality* in terms of the number of forwarding nodes and computational complexity. In this paper, we will not discuss the scheduling of transmissions of forwarding nodes. Interested readers can refer to the related work in [31], [32].

The rest of the paper is organized as follows: We propose an efficient flooding scheme in Section 3. Section 4 discusses the handling of mobility. In Section 5, we discuss the simulation of our flooding scheme by using the ns-2 test bed and compare its performance with other flooding algorithms. Finally, we conclude the work in Section 6.

3 Efficient Flooding Scheme Based on 1-Hop Information

3.1 System Model and Overview of Method

We assume all nodes in the network have the same transmission range R. Thus, the network can be represented as a unit disk graph $G(V, E)$. We assume the network is connected. Each node v in V has a unique ID, denoted by $id(v)$. Let $N(v)$ denote the set of neighbor nodes of v. That is, nodes in $N(v)$ are within the transmission range of v and can receive signals transmitted by v. Node v needs to know the information of its direct neighbors, including their IDs and their geographic locations. The 1-hop neighbor information can be easily obtained from the HELLO messages periodically broadcast by each node. For the rest of the paper, we simply use *neighbors* to mean 1-*hop neighbors*.

The basic idea of our flooding scheme is as follows: When a node (called the source) has a message to be flooded out, it computes a subset of its neighbors as forwarding nodes and attaches the list of the forwarding nodes to the message. Then, it transmits (broadcasts) the message out. After that, every node in the network does the same as follows: Upon receiving a flooding message, if the message has been received before, it is discarded; otherwise, the message is delivered to the application layer, and the receiver checks if itself is in the forwarding list. If yes, it computes the next hop forwarding nodes among its neighbors and transmits the message out in the same way as the source. The message will eventually reach all the nodes.

We discuss our method in three parts: 1) forwarding node selection, where a node selects a subset of its 1-hop neighbors to forward the flooding message, 2) forwarding node optimization, which further reduces the size of forwarding nodes by removing the nodes that are already covered, and 3) mobility handling, where each node incrementally updates its forwarding set in response to topology changes.

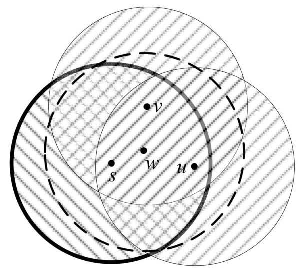


Fig. 2. Neighbor's area of node s.

3.2 Theoretical Foundations of Minimal Forwarding Nodes

We aim at designing a 1-hop flooding scheme. Flooding schemes in [9], [10], and [17] are all 1-hop flooding schemes that guarantee 100 percent deliverability of flooding messages. To achieve the optimal efficiency, we need to study the sufficient and necessary condition of 100 percent deliverability for flooding schemes that are based on 1-hop information. We introduce the following definitions:

Definition 1 (Coverage disk of a node). *The coverage disk of node s, denoted by $d(s)$, is a disk that is centered at s and whose radius is the transmission range of s.*

Since all neighbors of node s should be covered by $d(s)$, in this paper, we call "s covers u" or "u is covered by s" when u is a neighbor of s.

Definition 2 (Coverage area of a node-set). *The coverage area of a set of nodes A, denoted by $C(A)$, is the union of coverage disks of nodes in A.*

We simply state "the area is covered by A" if the area is within $C(A)$.

Definition 3 (Neighbor's coverage area). *The neighbor's coverage area of node s is the union of coverage disks of all s's neighbors plus s itself, i.e., $C(N(s) \cup \{s\})$.*

Definition 4 (Boundary of neighbor's area). *The boundary of neighbor's area of node s is the boundary of the area of $C(N(s) \cup \{s\})$.*

For simplicity, the neighbor's coverage area is called the *neighbor's area* and the boundary of neighbor's area called the *neighbor's boundary* for the rest of the paper. For example, in Fig. 2, the set of neighbors of s $N(s) = \{u, v, w\}$. Thus, the neighbor's area of s is $C(\{s, u, v, w\})$, i.e., the whole shadow area. The neighbor's boundary of s is the outside boundary of this shadow area.

Definition 5 (Forwarding set). *The set of forwarding nodes of s, denoted by $F(s)$, is a subset of s's neighbors that are selected for forwarding the flooding message ($F(s)$ includes s itself).*

Definition 6 (Minimum forwarding set $F_{min}(s)$). *The minimal forwarding set of s, denoted by $F_{min}(s)$, is the smallest $F(s)$ that covers the neighbor's area of s.*

Definition 7 (100 percent deliverability). *A flooding scheme is said to be 100 percent deliverable if and only if, FOR ANY NETWORK TOPOLOGY, all the nodes in the network should*

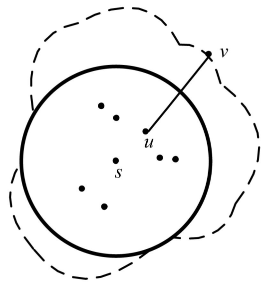


Fig. 3. Construct a new topology.

be able to receive flooding messages, letting every node execute the flooding scheme.

Theorem 1. *A 1-hop flooding scheme achieves 100 percent deliverability if and only if, for each node s, the neighbor's area of s is covered by $F(s)$.*

Proof. Sufficient condition ($\leftarrow$). Suppose, for each node s, the neighbor's area of s is covered by $F(s)$. We need to prove that the flooding scheme achieves 100 percent deliverability.

For each transmission node s, since all 2-hop neighbors of s are within the neighbor's area of s, they are sure to be covered by nodes in $F(s)$. Thus, all nodes that are 2-hop away from the source s are sure to be covered by $F(s)$. Notice that s's 3-hop neighbors are neighbors of s's 2-hop neighbors. There must exist some transmission nodes in $F(s)$ such that s's 3-hop neighbors are 2-hop neighbors of these transmission nodes. Thus, s's 3-hop neighbors are sure to be covered by forwarding sets of these transmission nodes. Nodes that are 4-hop and more from the source can be proved in a similar way. Therefore, the flooding message will be forwarded hop by hop throughout the whole network.

Necessary condition ($\rightarrow$). Suppose a flooding scheme A achieves 100 percent deliverability. Let $F_A(s)$ denote the set of forwarding nodes of s that is computed by A. We need to prove that, for each node s, the neighbor's area of s is covered by $F_A(s)$. We prove it by contradiction.

Suppose scheme A does not guarantee that, for each node, the neighbor's area of the node is covered by $F_A(s)$. There must exist node s in some network and the neighbor's area of s is not fully covered by $F_A(s)$. Since $F_{min}(s)$ is the smallest forwarding set that covers the neighbor's area of s, we have $F_{min}(s) \not\subset F_A(s)$. In other words, there exists node $u \in F_{min}(s)$ and $u \notin F_A(s)$. Notice that the coverage disks of all nodes in $F_{min}(s)$ are sure to contribute to the neighbor's boundary of s (if not, it can be removed from $F_{min}(s)$). We construct a new topology based on the current network as follows. We only keep node s and its neighbors and remove all other nodes from the network. Then, a new node, say v, is added and placed on the boundary that is contributed *only* by u (the dashed line in Fig. 3 is the neighbor's boundary of s). From s's view, everything remains the same since s has information of only 1-hop neighbors. So, $F_A(s)$ remains the same on this new topology. Notice that u is the only neighbor of s that can reach v and $u \notin F_A(s)$. Thus, v cannot be covered by any node in $F_A(s)$.

On the other hand, since there are no other nodes outside the coverage disk of s, node v can neither be covered by forwarding set of other nodes. That is, node v will eventually miss the flooding message. It contradicts the assumption that flooding scheme A achieves 100 percent deliverability. Theorem 1 is proved. □

Theorem 1 tells us that the sufficient and necessary condition of 100 percent deliverability for any 1-hop flooding scheme is that, for each node s, the neighbor's area of s should be covered by $F(s)$. Otherwise, some nodes in the network may miss the flooding message. Theorem 1 gives the theoretical guideline for computing $F(s)$ in our flooding scheme.

3.3 Computing Minimal Forwarding Nodes

Suppose s is a node that receives a flooding message for the first time and s appears in the forwarding list attached to the message (s could be the original source of the message). s is designated as a forwarding node and it computes the next hop forwarding nodes from its neighbors. Since s only has 1-hop neighbor information, it does not know who are the 2-hop neighbors. To achieve 100 percent deliverability, according to Theorem 1, $F(s)$ must cover the entire neighbor's area of s. Our task can be formally defined as:

$$\text{Minimize } F(s) \text{ such that } \bigcup_{v \in F(s)} d(v) = \bigcup_{u \in N(s)} d(u).$$

Taking the example in Fig. 2 again, s has three neighbors: u, v and w. Since $d(u) \cup d(v) \cup d(s)$ makes up the neighbor's area of s, it is enough to cover all of s's 2-hop neighbors if only u and v forward the message. In other words, $d(w) \subseteq d(u) \cup d(v) \cup d(s)$, there is no need for w to forward the message.

To minimize $F(s)$, every node in $F(s)$ must contribute to the neighbor's boundary of s; otherwise, this node can be removed from $F(s)$ without affecting the coverage area of $F(s)$. Therefore, computing the minimal $F(s)$ is to find a subset of $N(s)$ such that every node in the subset contributes to the neighbor's boundary of s.

We first give a simple $O(n^2)$ algorithm to compute $F(s)$ as follows, where $n = |N(s)|$: Since the outside nodes of $N(s)$, i.e., the nodes further away from s, are usually the nodes that contribute to the neighbor's boundary of s, all nodes in $N(s)$ are sorted in descending order into a list according to their euclidean distances to s. The first node in the list (that is the farthest away from s) is included in $F(s)$. Each time, the next node in the list is considered. If its coverage disk is not fully covered by the so far constructed $F(s)$, it is added into $F(s)$. This operation is repeated until all nodes in the list are considered. It is not difficult to see that $F(s)$ covers the neighbor's area of s, i.e., $F_{min}(s) \subseteq F(s)$. Notice that every node in $F(s)$ contributes to the neighbor's boundary of s. Any node in $F(s)$ cannot be removed since other nodes cannot take over its duty. That is, $F(s) \subseteq F_{min}(s)$. Therefore, $F(s)$ is the minimum. It is easy to see that the time complexity of this algorithm is $O(n^2)$.

Next, we present an algorithm with time complexity $O(nlogn)$. The strategy of this method is to compute the neighbor's boundary of s and, thus, the nodes that

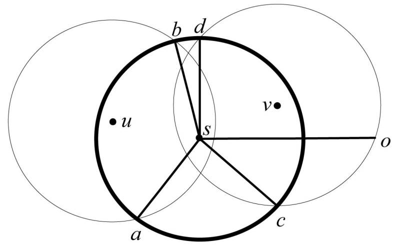


Fig. 4. Example of arcs.

contribute to this boundary are the nodes in $F(s)$. We use the pair wise boundary merging method to compute the boundary efficiently. Initially, each node is arbitrarily paired with another node to merge their coverage boundaries. Then, the merged pair's boundary is further merged with another pair's boundary. This merge operation is repeated until eventually there is only one big merged boundary, which is the neighbor's boundary of s. The minimal $F(s)$ consists of the nodes that contribute to this boundary.

Before considering the procedure for merging boundaries, we introduce data structures to represent arcs and boundaries. A boundary consists of a sequence of arcs. If we use the location of s as the reference point, any arc in the neighbor's boundary of s can be uniquely defined by a 3-tuple (θ^s, u, θ^e), where θ^s, u, and θ^e are the starting angle, the center and the ending angle of the arc, respectively. θ^s and θ^e are relative to the horizontal line going through s counting in the counterclockwise direction. For example, in Fig. 4, line os is the horizontal line from s, which is used as the reference line in counting the starting and ending angles of arcs. Arc $\widehat{ab}$ of disk u is represented by $(\angle osb, u, \angle osa)$, where $\angle osb$ is the starting angle and $\angle osa$ the ending angle of $\widehat{ab}$.

A boundary consists of a sequence of arcs. Thus, a boundary is represented by an array of arcs, denoted by $B[\,]$. The ith element in $B[\,]$, $B[i] = (\theta_i^s, u_i, \theta_i^e)$, $i = 1, \ldots, m$, denotes the ith arc in the boundary. The arcs in $B[\,]$ are sorted in nondescending order according to their starting angles. That is, $B[1].\theta_1^s \leq B[2].\theta_2^s \leq \ldots \leq B[m].\theta_m^s$. This sorted feature of arcs in B is critical to the efficient merging algorithm to be presented below. Because of this feature, the arcs in two boundaries can be merged in the same sequential order as the progress of their array indices without backtracking. To make the ending angle greater than the starting angle, any arc that crosses the horizontal line from s is split into two arcs. For the same example in Fig. 4, arc $\widehat{cd}$ is split into arcs $\widehat{od}$ and $\widehat{co}$ and they are represented by $(0°, v, \angle osd)$, $(\angle osc, v, 360°)$, respectively. The neighbor's boundary of s in Fig. 4 can be represented as

$$
\begin{aligned}
B[\,] &= \{\widehat{od}, \widehat{db}, \widehat{ba}, \widehat{ac}, \widehat{co}\} \\
&= \{(0°, v, \angle osd), (\angle osd, s, \angle osb), (\angle osb, u, \angle osa), \\
&\quad (\angle osa, s, \angle osc), (\angle osc, v, 360°)\}.
\end{aligned}
$$

Considering merging two boundaries B_i and B_j into a new one B, we start from the first arcs in B_i and B_j, respectively, merge them, and store the merged arc in B. Suppose now we are at the point of merging the kth arc of B_i (i.e., $B_i[k]$) with the lth arc of B_j (i.e., $B_j[l]$) and storing the merged arc in $B[h]$. Notice that two arcs intersect each other at no more than two points (because any two different disks intersect with each other at no more than two points). There are three possible cases of the intersection: 1) no intersection, 2) only one intersecting point, and 3) two intersecting points. We discuss the cases one by one.

If arcs $B_i[k]$ and $B_j[l]$ have no intersection, it can be further divided into three subcases: 1) The sectors of two arcs overlap each other, as in Fig. 5a, 2) the sector of one arc is contained by the other, as in Fig. 5b, and 3) there is no overlapping of the sectors of two arcs, as in Fig. 5c. For case 1, arc $B_i[k]$ contributes to the resulting boundary B. Thus, the resulting arc in B is $B_i[k]$, i.e., $B[h] = B_i[k]$. Then, we move to next arc in B_i and compare $B_i[k+1]$ with $B_j[l]$. For case 2, segment $\widehat{ab}$ of $B_i[k]$ is for sure to contribute to B, but segment $\widehat{bc}$ may intersect with the arc of $B_j[l+1]$. Therefore, $B[h]$ is set to $\widehat{ab}$. Then, we move to the next arc in B_j, i.e., $B_j[l+1]$, to compare it with segment $\widehat{bc}$. For case 3, without loss of generality, assuming $B_i[k].\theta^s < B_j[l].\theta^s$, arc $B_i[k]$ is sure to contribute to B, but $B_j[l]$ may intersect with $B_i[k+1]$. We set $B[h] = B_i[k]$ and move the next to compare $B_i[k+1]$ with $B_j[l]$. Notice that, in the above merging operation, we use the important feature that arcs in B_i and B_j are sorted according to their starting angles and the merging operation can be done in sorted order.

Cases 1-3 are exhaustive if two arcs $B_i[k] = (\theta_u^s, u, \theta_u^e)$ and $B_j[l] = (\theta_v^s, v, \theta_v^e)$ have no intersection. Since $B_i[k]$ and $B_j[l]$

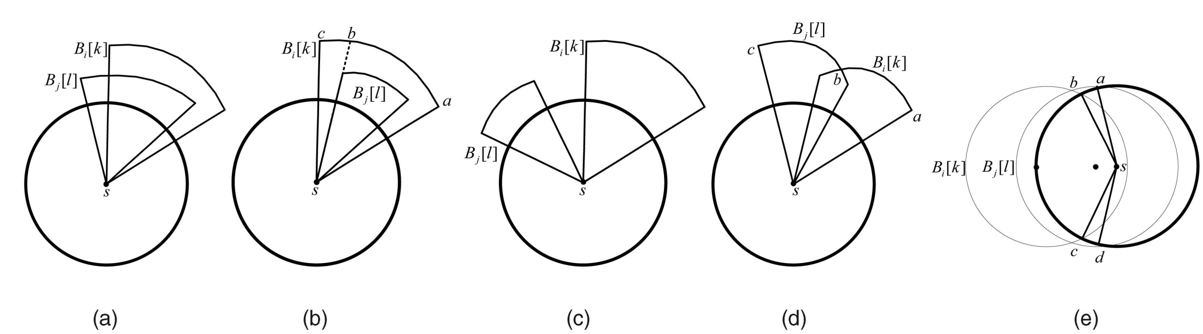


Fig. 5. Relationship between two arcs. (a) Case 1.1. (b) Case 1.2. (c) Case 1.3. (d) Case 2. (e) Case 3.

are symmetrical, we only need to consider one of the symmetrical cases. Notice that θ_v^s is either inside (θ_u^s, θ_u^e), or is outside (θ_u^s, θ_u^e). If θ_v^s is inside (θ_u^s, θ_u^e), θ_v^e is either inside (θ_u^s, θ_u^e) (Case 2) or is outside (θ_u^s, θ_u^e) (Case 1). If θ_v^s is outside (θ_u^s, θ_u^e), we only need consider the case θ_v^e is outside (θ_u^s, θ_u^e) (Case 3). This is because the other case is symmetrical as in case 1, where θ_v^s is inside (θ_u^s, θ_u^e) and θ_v^e is outside (θ_u^s, θ_u^e).

For the case where arcs $B_i[k]$ and $B_j[l]$ have only one intersecting point at b, as shown in Fig. 5d, segment $\widehat{ab}$ of $B_i[k]$ contributes to B and segment $\widehat{bc}$ of $B_j[l]$ may intersect with $B_i[k+1]$. Thus, we set $B[h] = \widehat{ab}$ and then move to the next to compare $B_i[k+1]$ with the segment $\widehat{bc}$ of $B_j[l]$.

If arcs $B_i[k]$ and $B_j[l]$ have two intersecting points at b and c, as shown in Fig. 5e, both segments $\widehat{ab}$ of $B_j[l]$ and $\widehat{bc}$ of $B_i[k]$ contribute to B. We set $B[h] = \widehat{ab}$ and $B[h+1] = \widehat{bc}$. Then, we move to compare $B_i[k+1]$ with segment $\widehat{cd}$ of $B_j[l]$.

Given two arcs $B_i[k] = (\theta_u^s, u, \theta_u^e)$ and $B_j[l] = (\theta_v^s, v, \theta_v^e)$, the remaining problem is how to determine which case these two arcs belong to. First, it is easy to compute two intersecting points of disk u and disk v. The basic idea is to determine how many intersecting points of the disks are contained in both arcs $B_i[k]$ and $B_j[l]$. Similarly to counting the starting (ending) angle of an arc, we compute the relative angles of these two points to the reference point s in the counterclockwise direction. $B_i[k]$ and $B_j[l]$ have one (two or zero) intersecting point(s) if and only if the relative angle of one (two or zero) intersecting point(s) is (are) *both* contained in the starting angle and ending angle of $B_i[k]$ and $B_j[l]$. For example, if the relative angle of one intersecting point of both disks is contained in (θ_u^s, θ_u^e) and (θ_v^s, θ_v^e), $B_i[k]$ and $B_j[l]$ are in cases $1 \sim 3$. To further define cases $1 \sim 3$, we simply compare the starting angles and ending angles of $B_i[k]$ and $B_j[l]$. For example, if $\theta_u^s \leq \theta_v^s \leq \theta_u^e$ and $\theta_u^e \leq \theta_v^e$, they are in case 1. Notice that all computations can be done in constant time.

Following the above discussion of merging the two arcs in B_i and B_j and moving the pointer to the next arc for merging, this operation can be repeated until all arcs in B_i are merged with B_j into the new boundary B. The following is the detailed algorithm:

BoundaryMerge Algorithm
Input: B_i and B_j.
Output: B.
Begin
  $k = 1$; // pointer to the current arc in B_i.
  $l = 1$; // pointer to the current arc in B_j.
  $h = 1$; // pointer to the current arc in B.
  while (there is unmerged arc in both B_i and B_j) **do**
    Merge $B_i[k]$ and $B_j[l]$ to B according to cases 1-3;
    Adjust k, l, and h accordingly;
  Return B.
End

Theorem 2. *The time complexity of BoundaryMerge algorithm is $O(n_1 + n_2)$, where n_1 and n_2 are the numbers of arcs in B_i and B_j, respectively.*

Proof. Notice that, in BoundaryMerge algorithm, we always move to the next arc of B_i or B_j after comparison and no backtracking is needed. So, the total running time is $O(n_1 + n_2)$, where n_1 and n_2 are the number of arcs in B_i and B_j, respectively. Theorem proved. □

Now, we consider the forwarding node selection algorithm. Initially, for each node i, $1 \leq i \leq |N(s)|$, its arc outside of the area of $d(s)$ is represented by a boundary array $B_i[\,]$. Then, the arcs are merged pair wise by using the BoundaryMerge algorithm until it becomes a single boundary of the coverage area of $N(s)$. $F(s)$ consists of the nodes that contribute to this boundary.

FwdNodes Algorithm
Input: s and $N(s)$.
Output: $F(s)$.
Begin
  $j = n$; // $n = |N(s)|$.
  while $j > 1$ **do**
    for $(i = 1; i < j; i = i + 2)$
      $B_{(i+1)/2}[\,] = \text{BoundaryMerge}(B_i[\,], B_{i+1}[\,])$;
    $j = j/2$;
  Output $F(s) = \{B[i].u_i | i = 1, 2, \ldots, k\}$; //$B$: the final boundary.
End

Notice that if n is odd in the above algorithm, we add a virtual arc whose starting angle and ending angle are both $0°$. It does not affect the correctness of the output.

Theorem 3. *The time complexity of FwdNodes algorithm is $O(nlogn)$, where $n = |N(s)|$.*

Proof. In the FwdNodes algorithm, each time we partition the current n boundaries into an $n/2$ group and run the BoundaryMerge algorithm to merge two boundaries in each group. According to Theorem 2, it takes $O(n)$ to complete boundary merge in all groups. Since each time the number of groups is reduced by half, it costs $O(logn)$ to obtain the final group, i.e., coverage's boundary. So, the total time complexity is $O(nlogn)$. □

The FwdNodes algorithm requires that each node has 1-hop information. According to Theorem 1, it guarantees that all nodes can receive the flooding message. Based on these conditions, the following theorem states that our algorithm is optimal:

Theorem 4. *The FwdNodes algorithm achieves local optimality in terms of: 1) the number of forwarding nodes is minimal, i.e., $F(s) = F_{min}(s)$; 2) the time complexity is the lowest.*

Proof. In the FwdNodes algorithm, each node only has 1-hop information. To cover 2-hop neighbors that are beyond its view, each node should select some 1-hop neighbors to relay the message such that these selected neighbors can sufficiently cover the neighbor's area of the node. Thus, all 2-hop neighbors are guaranteed to be covered. In the algorithm, each node s selects the minimal set of nodes $F(s)$ to forward the message by computing the neighbor's boundary of s. Notice that any node in $F(s)$ contributes to the final boundary. If a node in $F(s)$ does not relay the message, other nodes cannot take over its duty. It means that all nodes in $F(s)$ should

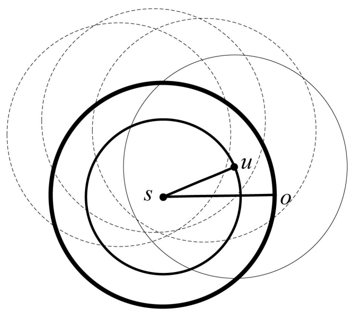
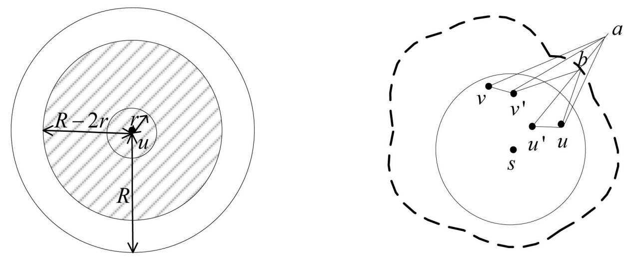


Fig. 6. The sorting problem is reduced to a boundary computing problem.



(a) (b)

Fig. 7. Reducing the coverage disk to compensate the location error.

forward the message to guarantee 2-hop neighbors are covered. Thus, we have $F(s) = F_{min}(s)$.

According to Theorem 3, the time complexity of computing $F(s)$ is $O(nlogn)$, where $n = |N(s)|$. Notice that computing $F(s)$ is equivalent to computing the neighbor's boundary of s. We will prove that sorting problem can be reduced to a boundary computing problem.

Given n real numbers to be sorted, we first scale them to the numbers in $[0, 2\pi]$. For each scaled number $a \in [0, 2\pi]$, point u is placed on the border of a small circle, such that $\angle uso = a$ (see Fig. 6, the center of the small circle is s, line os is parallel to the X-axis and angles are counted in the counterclockwise direction). Notice that these nodes are in $N(s)$ and the distance from s are the same. So, each disk of node contributes to the neighbor's boundary of s. After running FwdNodes algorithm, starting angles $B[i].\theta_i^s$, $i = 1, 2, \ldots, n$, of the final boundary are in nondescending order. It is equivalent to sorting the given n nodes. That is, the sorting problem can be reduced to boundary computing problem. We know that the fastest sorting algorithm costs $O(nlogn)$ time. So, the FwdNodes algorithm is the fastest algorithm to compute the neighbor's boundary of s. □

The local optimality of the algorithm is based on the accurate location information of nodes. However, the location information of nodes may not be 100 percent accurate in real applications. Our method can be slightly modified to deal with the situation where the location information is not 100 percent accurate. Suppose the error of location information is $\pm r$. For example, in Fig. 7a, the position of u is u's self-estimated location and u's actual location can be anywhere within the small circle with radius r. Thus, the largest variation of u's actual location is the diameter of the circle, i.e., $2r$. To compensate for this location error, we can simply make the valid coverage area of each node a disk with radius $(R - 2r)$ (the gray disk in Fig. 7a). By doing so, we ensure that the *actual* neighbor's area of s will be covered by nodes in $F(s)$ with radius R in the real system and 100 percent deliverability can be guaranteed. The following theorem gives a formal proof of the 100 percent deliverability of this modified method:

Theorem 5. *Given location error $\pm r$ and transmission range R, by using $(R - 2r)$ as the radius of coverage disk for computing $F(s)$, the algorithm guarantees 100 percent deliverability in the real system, provided the unit disk graph with radius $(R - 2r)$ is connected.*

Proof. According to Theorem 1, it is equivalent to prove that the actual neighbor's area of s, using $(R - 2r)$ as the radius of coverage disk, is covered by nodes in $F(s)$ with radius R in the real system. We prove it by contradiction.

Suppose the actual neighbor's area of s cannot be covered by nodes in $F(s)$ with radius R in the real system. There must exist node a that is covered by actual location u with radius $(R - 2r)$. But, a cannot be covered by any node in $F(s)$ at its actual location with radius R. Taking an example in Fig. 7b, the outside dash line is the theoretical neighbor's boundary of s by using $(R - 2r)$ as the radius of coverage disk. Suppose the position of u' is u's self-estimated location. We know $d(u, u') \le r$, where $d(u, u')$ is the euclidean distance between u and u'. Since $d(u, a) \le R - 2r$, we have $d(u', a) \le d(u, u') + d(u, a) \le R - r$.

If $u' \in F(s)$, then a can be covered by u' at its actual location u with radius R since $d(u, a) \le R - 2r$. It contradicts the assumption that a cannot be covered by any node in $F(s)$ at its actual location with radius R.

If $u' \notin F(s)$, we have $d(u', b) \ge R - 2r$ (otherwise, u' contributes to the theoretical neighbor's boundary of s, i.e., $u' \in F(s)$). Thus, we have

$$d(a, b) = d(u', a) - d(u', b) \le (R - r) - (R - 2r) \le r. \quad (1)$$

Notice that b is the theoretical neighbor's boundary of s. There must exist node $v' \in F(s)$ such that b is covered by v' with radius $(R - 2r)$. That is,

$$d(v', b) \le R - 2r. \quad (2)$$

Combining (1) with (2), we have $d(v', a) \le d(v', b) + d(a, b) \le R - r$. Suppose position v is the actual location of v'. We have $d(v, v') \le r$. Thus, $d(v, a) \le d(v, v') + d(v', a) \le R$. That is, a can be covered by v' at its actual location v with radius R. It also gets contradicted. Theorem 5 is proved. □

Noticing that r is often very small compared with R, this modification will not significantly degrade the performance of our algorithm.

After computing $F(s)$, s attaches IDs of nodes in $F(s)$ to the flooding message and broadcasts it out. When receiving this message, a neighbor node of s, say u, checks if its own ID is in the forwarding list attached to the message. If yes, it will compute $F(u)$ and broadcast the message in the same

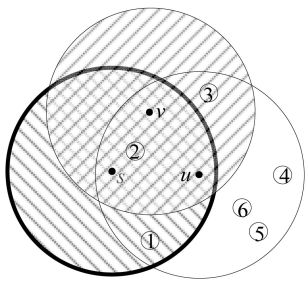


Fig. 8. An example of optimizing $F(u)$.

way as s. In this way, the message is forwarded hop by hop until all of the nodes in the network receive it.

The proposed method is based on the model that the network is represented as a unit disk graph. The methodology is applicable to the cases where the coverage area of each node is not a perfect disk. According to Theorem 1, so long as the neighbor's area is covered by $F(s)$, 100 percent deliverability is always guaranteed. The deliverability guarantee has nothing to do with the shape of the node's coverage area. Therefore, if node s knows the coverage areas of its neighbors (even they are not in the shape of a perfect disk), s can still compute $F(s)$ to cover its neighbor's area and 100 percent deliverability can be achieved. If the exact shape of coverage area is costly or impossible to obtain, one solution is to approximate the actual coverage area with some regular shape such that this regular shape is surely covered by the actual coverage area. Similarly to Theorem 5, we can prove that 100 percent deliverability is still guaranteed when using these regular shapes to compute $F(s)$.

3.4 Forwarding Node Optimization

The $F(s)$ computed above is only locally optimal based on the 1-hop information of s. When a node u receives the flooding message from s (we call s the parent of u) and u is a forwarding node nominated by s (i.e., $u \in F(s)$), the computing of $F(u)$ can be further optimized based on the information of $F(s)$, which is attached to the flooding message from s. This is because some nodes in $F(u)$ may already be covered by node s or node-set $F(s)$ and, thus, $F(u)$ could be further reduced by removing out those nodes.

Consider the example given in Fig. 8, where nodes u and v are neighbors of s and $F(s) = \{u, v\}$. The coverage area $d(u)$ overlaps with $d(s)$ and $d(v)$ (notice node v is also in $F(s)$). The nodes in the overlapped area of $d(u)$ with $d(s)$ were already considered by s when computing $F(s)$. Thus, these nodes can be removed from $F(u)$. For the overlapped area of $d(u)$ with other nodes in $F(s)$, for example, node v in Fig. 8, we use node ID as the priority for forwarding messages. That is, the node with the smaller ID has to forward the message if its coverage disk overlaps with another node. Therefore, the nodes of $F(u)$ that fall into the coverage area of the following node-set can be removed from $F(u)$:

$$\{s\} \cup \{v | \ v \in (F(s) \cap N(u)) \text{ and } id(v) \leq id(u)\}. \qquad (3)$$

Notice that, in node-set (3), we only consider set $F(s) \cap N(u)$. That is, nodes in node-set (3) are all neighbors of node u. Thus, u knows the geographic locations of nodes in the above node-set. This location information is necessary when u checks whether nodes in $F(u)$ fall into the coverage area of the node-set (3). We can see this optimization is still based on 1-hop information of a node.

Taking the example in Fig. 8 again, suppose $id(v) \leq id(u)$ and $F(u) = \{1, 2, 3, 4, 5\}$. Since nodes 1 and 2 are in $N(s)$, they are already covered by s and can be removed from $F(u)$. Node 3 is covered by v and v is also a forwarding node and $id(v) \leq id(u)$. Thus, node 3 can also be removed from $F(u)$. Finally, $F(u) = \{4, 5\}$. That is, node u only needs to nominate the nodes of $F(u)$ in a clear area.

The significance of this optimization is that it prevents the flooding message from going backward. The message is always propagated forward toward the uncovered area, which reduces the redundant transmissions greatly.

The following is the optimized forwarding node selection algorithm. It is executed whenever a node receives a flooding message:

OptFwdNodes Algorithm
Input: message m from s.
Begin
  if m was received before, **then** discard m;
  else
    Deliver m to upper layer;
    if this node, say u, is in forward-list in m
      Compute $F(u)$;
      Remove from $F(u)$ the nodes that are covered by node-set (3);
      Attach $F(u)$ to m and transmit m out.
End

According to Theorem 4, we know that the FwdNodes algorithm can guarantee that all nodes receive a flooding message. After optimization in the OptFwdNodes algorithm, the 100 percent deliverability feature is still preserved. The following theorem states this feature:

Theorem 6. *The OptFwdNodes algorithm guarantees that all nodes can receive a flooding message.*

Proof. According to Theorem 4, if all forwarding nodes run the FwdNodes algorithm, 100 percent deliverability is guaranteed. So, to prove Theorem 6, we need to prove that removing nodes from $F(u)$ in the OptFwdNodes algorithm does not affect 100 percent deliverability of our scheme.

We suppose $\{u, v\} \subseteq F(s)$ and $id(v) \leq id(u)$. If some nodes in $F(u)$ are neighbors of s and v, the coverage disks of these nodes are sure to be within the coverage area of $F(s)$ and $F(v)$. So, there is no need to let these nodes forward the message. Thus, removing these nodes from $F(u)$ does not affect 100 percent deliverability of our scheme. The theorem is proved. □

The time complexity of OptFwdNodes algorithm is given below.

Theorem 7. *The time complexity of the OptFwdNodes algorithm is $O(nlogn)$, where $n = |N(s)|$.*

Proof. It is not difficult to see that the time complexity of the OptFwdNodes algorithm is the same as that of the FwdNodes algorithm. Theorem 7 is proved. □

The source node first floods a message by running FwdNodes algorithm. Each forwarding node forwards the message by running OptFwdNodes algorithm. Finally, all nodes in the network can receive the message and the redundant transmission can be significantly reduced. However, transmission failures often occur in MANETs due to the unpredictable environments. A nodes that misses a flooding message still has a chance to receive the message from other nodes by "redundant" transmissions. Thus, further optimization of the forwarding set is needed or not is based on the practical situations of networks. If transmissions are reliable, further optimization can save more redundant messages. Otherwise, a few redundant messages may be helpful to the fault tolerance of the system.

4 Mobility Handling

In MANETs, nodes may be mobile, which causes dynamic changes of the network topology. For the flooding scheme, each node, say s, maintains its neighbor information and computes $F(s)$. To cope with the dynamic topology changes, there are two strategies to maintain the flooding scheme: 1) no update, each node recomputes its forwarding node set for each flooding request, or 2) incremental update. Each node incrementally updates its forwarding node set upon each topology change. For strategy 1, we do not need to do anything. In this section, we propose an efficient algorithm that can incrementally update the forwarding node set as the topology changes. By using this method, nodes do not need to recompute the forwarding node set when it needs to flood a message. The forwarding node set is maintained at each node and is always ready for use.

For each node u, there are three cases that require updating $F(u)$: 1) a neighbor of u moves, but still in $N(u)$, 2) a neighbor of u moves out of $N(u)$, and 3) a node moves in and becomes the new neighbor of u. We assume that only one update is handled at a time. We concentrate on updating $F(u)$ for these three cases and discuss them case by case.

Case 1. We consider the first case that a neighbor of u, say v, moves but is still in $N(u)$. There are two subcases.

Case 1.1. If $v \notin F(u)$, we need to check whether the coverage disk of v exceeds the neighbor's boundary of u. If it happens, the disk of v will contribute to the final boundary B and $F(u)$ will be updated. We first compute how many arcs in B are affected by the movement of v. It can be done by locating the starting angle and ending angle of the current location of disk v in B by binary search. Suppose that k arcs in B are affected by the arc of disk v. It means that the sectors of these arcs overlap with the sector of disk v. Notice that these k arcs form a continuous segment of B in nondecreasing order according to their starting angles. Then, we run BoundaryMerge algorithm to merge this segment and the arc of disk v to update the new boundary B and $F(u)$.

Case 1.2. If $v \in F(u)$, the final boundary B not only may be affected by the *current* location of v, but also may be affected by the *former* location of v. Notice that $v \in F(u)$ and the location of v changes. Some nodes in $N(u) - F(u)$ may contribute to B because v leaves its former place. On the other hand, some nodes in $F(u)$ may become invalid because v moves to the current place. So, it has two steps to update. We first compute how many arcs in $N(u)$ may contribute to the new boundary because of leaving v. Since there is no order in $N(u)$, we find k arcs that may contribute to B one by one. We compute the new boundary of these k arcs. Second, similarly to Case 1.1, we still need to compute how many arcs in B are affected by the new location of v. Suppose l continuous arcs in B are affected. We update B and $F(u)$ again by merging these l continuous arcs and the arc of disk v in current place.

Case 2. Node v is a neighbor of u and v moves out of $N(u)$. If $v \notin F(u)$, there is no need to update. If $v \in F(u)$, some nodes in $N(u) - F(u)$ may contribute to B due to the leaving of v. This is similar to the first step of Case 1.2. We can update $F(u)$ for this case.

Case 3. Node v moves into the coverage disk of u and becomes a new neighbor of u. Similar to Case 1.1. We can update $F(u)$ for this case. A detailed algorithm is given below.

```
TopologyUpdate Algorithm
Input: v that changes its location to u.
Output: updated F(u).
Begin
  if v ∉ F(u) and v is now in N(u) //case 1.1 or case 3.
      Find arcs in B that are affected by disk v;
      //suppose k arcs B[i], B[i+1], ..., B[i+k-1] are
         affected.
      BoundaryMerge({B[i], B[i+1], ..., B[i+k-1]}, d(v));
  if v ∈ F(u) //case 1.2 or case 2.
      Find arcs in N(u) that are affected by v's leaving;
      //suppose k arcs are affected.
      Compute the boundary of the affected k arcs;
      Find arcs in B that are affected by v's current place;
      //suppose l arcs B[i], B[i+1], ..., B[i+l-1] are
         affected.
      BoundaryMerge({B[i], B[i+1], ..., B[i+l-1]}, d(v));
  Update F(u) based on the new boundary B.
End
```

Theorem 8. *The time complexities of the update for Case 1.1, Case 3, Case 1.2, and Case 2 are $O(k + logn)$ and $O(n + klogk)$, respectively, where $n = |N(s)|$ and k is the number of nodes that are affected by topology change.*

Proof. For Case 1.1 and Case 3, it costs $O(logn)$ to locate the arc of disk v in B by binary search. It further costs $O(k)$ to merge $\{B[i], B[i+1], \ldots, B[i+k-1]\}$ and disk v by the BoundaryMerge algorithm. So, the total time cost of update for Case 1.1 and Case 3 is $O(k + logn)$.

For Case 1.2 and Case 2, it costs $O(n)$ to find k disks of nodes in $N(u)$ that are affected by the movement of v. Similarly to boundary computing in the FwdNodes algorithm, computing the new boundary of these k disks costs $O(klogk)$. It further costs $O(l + logn)$ to compute the new boundary of l disks in the second step. So, the total time cost of update for Case 1.2 and Case 2 is $O(n + klogk)$. Theorem 8 is proved. □

From Theorem 8, we can see that update for Case 1.1 and Case 3 is very efficient compared to recomputing $F(u)$. Update for Case 1.2 and Case 2 is also efficient when k is not

TABLE 1
Four Flooding Schemes in Simulation

Algorithms	Information Required	Strategies	Time Complexity
Pure Flooding	None	Sender-based	$O(1)$
Edge Forwarding	1-hop information	Sender-based	$O(p \times q)$*
CDS-based	2-hop information	Receiver-based	$O(n^2)$
Our Scheme	1-hop information	Sender-based	$O(nlogn)$

*Taking the example in Fig. 1, p is the number of nodes in area D and q is the number of nodes in area A.

large. If $k = \Theta(n)$, the time complexity of the TopologyUpdate algorithm is the same as that of FwdNodes algorithm.

5 Simulation

To analyze the performance of our flooding scheme, we compare it with three deliverability-guaranteed schemes: *Pure flooding*, *Edge Forwarding* [9], and *CDS-based flooding* [11]. The information of the schemes is listed in Table 1. Edge Forwarding is picked because that it has the best performance among existing 1-hop flooding schemes [9]. It is a good comparative criterion to inspect our flooding scheme. CDS is one of the most important techniques to flooding operation in MANETs. So, CDS-based flooding is further selected for comparisons. In the CDS-based scheme, a node marks itself as belonging to the CDS if there exist two unconnected neighbors. A marked node can quit the CDS later if its neighbors are covered by two CDS neighbors and they have greater IDs. It was proved that the marked nodes form a CDS [11]. Notice that all forwarding nodes in a flooding operation form a CDS in the network. It means that the number of forwarding nodes is no less than the number of MCDS (Minimum CDS) in the network. So, the number of MCDS is the lower bound of the number of forwarding nodes. Although computing MCDS is NP-hard, there exists a ratio-8 approximation algorithm [29]. This lower bound is computed and is used as a benchmark for comparison with the simulated flooding schemes.

We study the performance of flooding schemes against four parameters: *number of nodes*, *transmission range*, *network size*, and *network load*. We run simulations under the ns-2 testbed with the CMU wireless extension. The simulator parameters are listed in Table 2. The popular two-ray ground reflection model is adopted as the radio propagation model. The MAC layer scheme follows the IEEE 802.11 MAC specification. We use the broadcast mode with no RTS/CTS/ACK mechanisms for all message transmissions. Each data packet with attached information has a constant length of 256 bytes. The bandwidth of a wireless channel is set to 2M b/s as the default. Some of the schemes require nodes to send a HELLO message to their 1-hop neighbors periodically. This cost of the HELLO message is ignored in our performance study.

The main objective of those efficient flooding schemes is to reduce the number of forwarding nodes as much as possible such that the redundant transmission is minimized. So, we use the metric *ratio of forwarding nodes* to evaluate the efficiency of flooding schemes. The ratio of forwarding nodes is defined to be the ratio of the total number of nodes involved in the packet forwarding in a flooding operation over the total number of nodes in the network, such as:

$$\text{ratio of forwarding nodes} = \frac{\text{the number of forwarding nodes}}{\text{the number of total nodes}}.$$

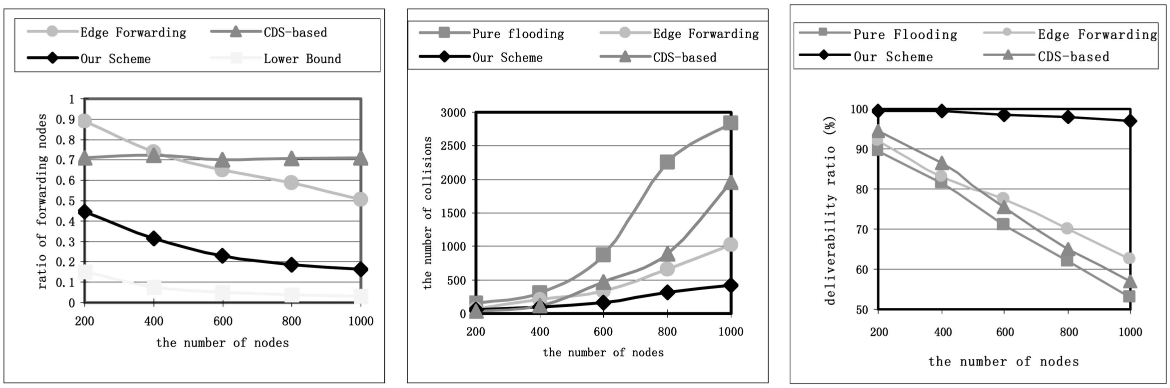
Reducing the forwarding nodes in flooding would effectively reduce the signal collision in the network. The MAC layer of IEEE 802.11 in ns-2 can check the occurrence of collisions. If the number of collisions is high, it would result in more packet loss or more retransmissions. We also use the metric *number of collisions* to evaluate the efficiency of flooding schemes. The number of collisions is defined to be the sum of collisions that each node experiences before it receives the flooding message correctly.

Signal collisions will eventually affect the deliverability of flooding messages. Some nodes in the network miss flooding messages due to the large number of collisions. The metric *deliverability ratio* is used to further study the efficiency of algorithms. The deliverability ratio is defined by the number of nodes that successfully receive the flooding messages over the total number of nodes in the network.

In each simulation run, we generate a certain number of nodes and randomly place them on a square area. There is a link between two nodes if and only if their Euclidean distance is not greater than transmission range R. The source which initiates a flooding message is randomly picked from nodes in the network. Only one flooding occurs at any one time (except for the experiments on deliverability ratio). Three flooding schemes and the theoretical lower bound that are mentioned above are simulated and

TABLE 2
Simulation Parameters

Parameter	Value
Simulator	*ns*-2 (version 2.28)
MAC Layer	IEEE 802.11
Data Packet Size	256 bytes
Bandwidth	2 Mb/s
Transmission Range	100~300 meter
Number of Nodes	200~1000
Size of Square Area	200,000~1,000,000 meter^2
Network Load	1Pkt/s~25Pkt/s
Number of Trails	100



(a) (b) (c)

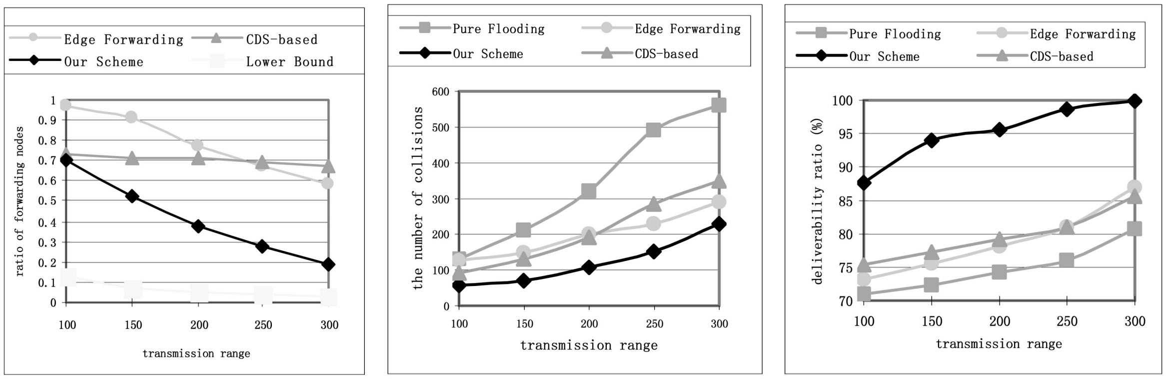
Fig. 9. Performance versus the number of nodes.

compared with our scheme under the same environment. We study how the ratio of forwarding nodes, the number of collisions, and the deliverability ratio are affected by four parameters: the number of nodes, transmission range, network size, and network load, respectively. The results presented in the following figures are the means of 100 separate runs. Any case where the network is not connected is discarded.

5.1 Performance versus Number of Nodes

In this simulation, a certain number of nodes, from 200 to 1000, are randomly placed on a $1,000 \times 1,000\ m^2$ area. The transmission range is fixed at 250 m. In the experiment of deliverability ratio in Fig. 9c, the network load is set to 10 Pkt/s. It means that the network generates 10 flooding messages per second on average. The deliverability ratio is calculated for 100 seconds. Since every node is a forwarding node in the pure flooding scheme, its curve was dropped out in Fig. 9a. The simulation results are plotted in Fig. 9a, Fig. 9b, and Fig. 9c. We have following observations:

1. The performance of our flooding scheme is significantly better than performance of Edge Forwarding and CDS-based schemes showed in Fig. 8. This is because that each forwarding node u selects the minimal $F(u)$ to cover all 2-hop neighbors in our scheme. It guarantees that the number of forwarding nodes is minimized at each step, while the Edge Forwarding and CDS-based schemes do not.
2. The curve of our scheme becomes closer to the curve of lower bound when the number of nodes increases. In Fig. 9a, when the number of nodes reaches 1,000, only 16.5 percent of nodes participate in forwarding in our scheme while ratios of Edge Forwarding and CDS-based schemes are 50.7 percent and 71 percent, respectively. This is because, as the increase of network density (resulting from the increase of nodes), $N(u)$ becomes larger, but $F(u)$ is saturated. That is, the number of nodes required to cover the same area (i.e., the neighbor's area) will not increase that much, because each node has a fixed coverage disk. Therefore, the ratio $F(u)/N(u)$ decreases as the increase of nodes in the network. We can conclude that our flooding scheme is more suitable for networks with high density.
3. Both the curves of our scheme and of Edge Forwarding fall down when the number of nodes increases in Fig. 9a. But, the number of nodes has little effect on the result of CDS-based scheme. Notice that when network density increases, there is more chance for u's neighbors being connected. At the same time, high density also causes increase of $N(u)$. It means there is a high chance that there exist two unconnected neighbors. So, these two conflicting factors make the result of CDS-based scheme not sensitive to the change of number of nodes.
4. Our scheme, Edge Forwarding and CDS-based scheme have much lower collisions comparing with pure flooding. The reason is that every node forwards flooding messages in pure flooding and it results in large number of collisions in the network. Collisions of pure flooding and CDS-based schemes increase quickly while the number of nodes increases. Performance of our scheme is the best among all schemes. For example, in Fig. 9b, when the number of nodes reaches 600, the number of collisions of our scheme is only 211 while that of Edge Forwarding and CDS-based schemes are 335 and 469, respectively. After that, their collisions are more than 100 percent higher than our scheme.
5. Deliverability ratio of our scheme is significantly higher than the ratios of Edge Forwarding, CDS-based and pure flooding schemes. In Fig. 9c, our scheme guarantees 100 percent deliverability when the number of nodes varies from 200 to 400, while deliverability ratios of other schemes are only 75 percent-92 percent around. Although collisions occur in our scheme even the number of nodes is small, a node that misses flooding messages from a forwarding node still has chance to receive messages from another forwarding node. So the value of our scheme can almost reach 100 percent if the number of nodes is between 200 to 600 (the number of collisions is low). The performance of pure flooding is the worst among the three schemes. It is caused by the broadcast storm problem since every node retransmits the flooding message in the network.



(a) (b) (c)

Fig. 10. Performance versus transmission range.

5.2 Performance versus Transmission Range

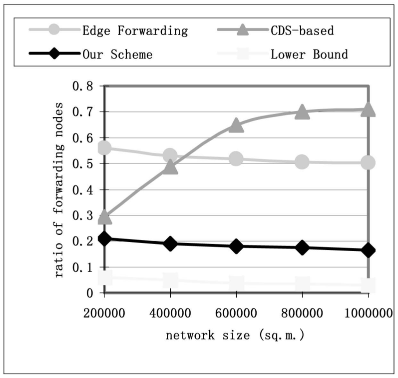
In this simulation, 500 nodes are randomly placed on a $1000 \times 1000\ m^2$ area. The network load is set to 10Pkt/s and each simulation is run for 100 seconds in Fig. 10c. We study the performance against the transmission range of each node. The simulation results are plotted in Fig. 10a, Fig. 10b and Fig. 10c. We have following observations:

1. Performance of our scheme is significantly better than performance of Edge Forwarding and CDS-based schemes shown in Fig. 10a. The reason is similar to the results in Fig. 9a. As the increase of transmission range, each node has more neighbors. It has the same effect on the increase of network density as the increase of nodes in a fixed square area.
2. The curve of our scheme becomes closer to the curve of lower bound when transmission range increases in Fig. 10a. This trend is more significant than that in Fig. 9a. See Fig. 10a, when the transmission range reaches 300 m, only 19 percent of nodes participate in forwarding in our flooding scheme while values of Edge Forwarding and CDS-based schemes are 58 percent and 67 percent, respectively. This is because that increase of transmission range not only results in higher density of network, but also makes flooding faster in the network. It means that flooding operation can be done in less steps due to the large transmission range of nodes. Notice that our scheme achieves that the number of forwarding nodes is minimal at each step. So, less steps to complete flooding makes our results closer to the lower bound when transmission range increases.
3. Both the curves of our scheme and Edge Forwarding fall down when transmission range increases in Fig. 10a. The curve of CDS-based scheme does not change much when R increases from 100 m to 250 m. The reason has been discussed before. But, further increase of R makes the curve fall down. It is because that when R reaches a certain value, such as 250 m in Fig. 10a, further increase of R will slightly increase $N(u)$ due to the fixed number of nodes. But, an increase of R makes nodes have more chance to be connected. So, the curve of the CDS-based scheme falls down when R is more than 250 m.
4. Curves in Fig. 10b show the similar trend as those in Fig. 9b. When the transmission range increases (i.e., a node has more neighbors), there are more chances for nodes to experience collisions. Since our scheme minimizes the number of forwarding nodes in each step, its performance is much better than that of pure flooding, Edge Forwarding and CDS-based schemes.
5. Deliverability ratios of three schemes all increase when transmission range increases in Fig. 10c. It is because that increase of transmission range not only causes more collisions, but also provides more chances for nodes to receive flooding messages from different forwarding nodes. See Fig. 10c, the ratio of our scheme becomes very close to 100 percent when transmission range reaches 300. Our scheme performs best among four schemes.

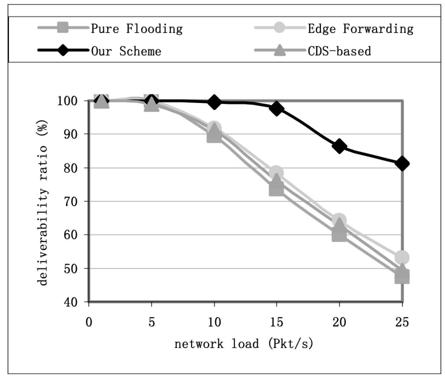
5.3 Performance versus Network Size and Network Load

In the simulation of Fig. 11a, we increase the area of the network region, from 200,000 to 1,000,000 m^2. The node density is fixed at 1,000 $m^2/node$. That is, there are 200 nodes in the network with size 200,000 m^2. The generated nodes are randomly placed on the network square domain. We fix the transmission range at 250 m. We observe that our scheme and Edge Forwarding are both highly scalable with respect to the network size. In contrast, performance of CDS-based scheme is better in a smaller network, but becomes worse when network size increases. Thus, it is not a scalable flooding scheme. Once again, our scheme performs the best among all flooding schemes.

In the simulation of Fig. 11b, 1,000 nodes are randomly placed on a $1,000 \times 1,000\ m^2$ area. The transmission range is fixed at 250 m. We vary the network load from 1Pkt/s to 25Pkt/s and each simulation is run for 100 seconds. We observe that performance of our scheme is significantly better than that of pure flooding, Edge Forwarding and CDS-based schemes. The ratio of our scheme keeps almost 100 percent when network load is less than 15Pkt/s. Further increase of network load causes the curve to quickly fall down. It is because that more frequently flooding message are generated, larger number of collisions nodes experience. In contrast, the curves of pure flooding, Edge Forwarding



(a)



(b)

Fig. 11. Performance versus network size and network load.

and CDS-based schemes fall down when network load is just over 5Pkt/s.

6 Conclusions

The paper addressed the efficient flooding problem in MANETs. We have presented an efficient flooding scheme that uses only 1-hop neighbor information. We have proved that our proposed scheme achieves the local optimality in terms of: 1) the number of forwarding nodes is the minimal and 2) the time complexity $O(nlogn)$ is the lowest. Extensive simulations have been conducted to compare our scheme with pure flooding, Edge Forwarding and CDS-based schemes. Simulation results have shown that our proposed scheme uses less forwarding nodes, incurs less collision, obtains high deliverability ratio and is highly scalable, compared with the existing schemes.

Acknowledgments

This work is supported in part by the Research Grants Council of Hong Kong under grant numbers CityU 1165/04E, CityU 114505, and NSF China 60633020.

References

[1] D. Johnson and D.A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," *Mobile Computing,* T. Imielinski and H.F. Korth, eds., pp. 153-181, Kluwer Academic Publishers, 1996.
[2] C.E. Perkins, "Ad Hoc On-Demand Distance Vector (AODV) Routing," *INTERNET DRAFT—Mobile Ad Hoc NETworking (MONET) Working Group of the Internet Eng. Task Force (IETF),* Nov. 1997.
[3] Z.J. Haas and M.R. Pearlman, "The Zone Routing Protocol (ZRP) for Ad Hoc Networks," Mobile Ad Hoc NETworking (MONET) Working Group of the Internet Eng. Task Force (IETF), Nov. 1997.
[4] Y. Ko and N. Yaidya, "Location-Aided Routing (LAR) in Mobile Ad Hoc Networks," *Proc. MOBICOM '98,* pp. 66-75, 1998.
[5] C. Ho, K. Obraczka, G. Tsudik, and K. Viswanath, "Flooding for Reliable Multicast in Multi-Hop Ad Hoc Networks," *Proc. Int'l Workshop Discrete Algorithms and Methods for Mobile Computing and Comm.,* pp. 64-71, 1999.
[6] J. Jetcheva, Y. Hu, D. Maltz, and D. Johnson, "A Simple Protocol for Multicast and Broadcast in Mobile Ad Hoc Networks," Internet Draft: draft-ietf-manet-simple-mbcast-01.txt, July 2001.
[7] S. Ni, Y. Tseng, Y. Chen, and J. Sheu, "The Broadcast Storm Problem in a Mobile Ad Hoc Network," *Proc. ACM/IEEE MOBICOM '99,* pp. 151-162, Aug. 1999.
[8] P. Sinha, R. Sivakumar, and V. Bharghavan, "Enhancing Ad Hoc Routing with Dynamic Virtual Infrastructures," *Proc. IEEE INFOCOM '01,* pp. 1763-1772, 2001.
[9] Y. Cai, K.A. Hua, and A. Phillips, "Leveraging 1-Hop Neighborhood Knowledge for Efficient Flooding in Wireless Ad Hoc Networks," *Proc. 24th IEEE Int'l Performance Computing and Comm. Conf. (IPCCC),* Apr. 2005.
[10] C.C. Yang and C.Y. Chen, "A Reachability-Guaranteed Approach for Reducing the Broadcast Storms in MANETs," *Proc. IEEE Semiann. Vehicular Technology Conf. (VTC '02),* Sept. 2002.
[11] J. Wu and H. Li, "On Calculating Connected Dominating Set for Efficient Routing in Ad Hoc Wireless Networks," *Proc. Third Int'l Workshop Discrete Algorithms and Methods for Mobile Computing and Comm. (DiaLM),* pp. 7-14, 1999.
[12] J.E. Wieselthier, G.D. Nguyen, and A. Ephremides, "On the Construction of Energy-Efficient Broadcast and Multicast Trees in Wireless Networks," *Proc. IEEE INFOCOM '00,* 2000.
[13] D. Li, X. Jia, and H. Liu, "Energy Efficient Broadcast Routing in Ad Hoc Wireless Networks," *IEEE Trans. Mobile Computing,* vol. 3, no. 2, pp. 144-151, Apr.-June 2004.
[14] Y. Tseng, S. Ni, and E.Y. Shih, "Adaptive Approaches to Relieving Broadcast Storms in a Wireless Multihop Mobile Ad Hoc Networks," *Proc. Int'l Conf. Distributed Computing Systems (ICDCS '01),* pp. 481-488, 2001.
[15] Y. Sasson, D. Cavin, and A. Schiper, "Probabilistic Broadcast for Flooding in Wireless Mobile Ad Hoc Networks," Technical Report IC/2002/54, Swiss Federal Inst. of Technology, 2002.
[16] M.T. Sun, W.C. Feng, and T.H. Lai, "Location Aided Broadcast in Wireless Ad Hoc Networks," *Proc. GLOBECOM '01,* 2001.
[17] H. Lim and C. Kim, "Multicast Tree Construction and Flooding in Wireless Ad Hoc Networks," *Proc. ACM Int'l Workshop Modeling, Analysis, and Simulation of Wireless and Mobile System (MSWIM),* pp. 61-68, Aug. 2000.
[18] B. Williams and T. Camp, "Comparison of Broadcasting Techniques for Mobile Ad Hoc Networks," *Proc. MOBIHOC '02,* pp. 914-205, 2002.
[19] A. Laouiti, A. Qayyum, and L. Viennot, "Multipoint Relaying: An Efficient Technique for Flooding in Mobile Wireless Networks," *Proc. 35th Ann. Hawaii Int'l Conf. System Sciences (HICSS '01),* 2001.
[20] A. Qayyum, L. Viennot, and A. Laouiti, "Multipoint Relaying for Flooding Broadcast Messages in Mobile Wireless Networks," *Proc. 35th Hawaii Int'l Conf. System Sciences,* 2002.
[21] W. Lou and J. Wu, "Double-Covered Broadcast (DCB): A Simple Reliable Broadcast Algorithm in MANETs," *Proc. INFOCOM '04,* 2004.
[22] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, "Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks," *Proc. MOBICOM '01,* pp. 85-96, July 2001.
[23] W. Peng and X. Lu, "On the Reduction of Broadcast Redundancy in Mobile Ad Hoc Networks," *Proc. MOBIHOC '00,* 2000.
[24] I. Stojmenovic, M. Seddigh, and J. Zunic, "Dominating Sets and Neighbor Elimination Based Broadcasting Algorithms in Wireless Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 1, pp. 14-25, Jan. 2002.

[25] J. Sucec and I. Marsic, "An Efficient Distributed Network-Wide Broadcast Algorithm for Mobile Ad Hoc Networks," CAIP Technical Report 248, Rutgers Univ., Sept. 2000.
[26] J. Wu and F. Dai, "Broadcasting in Ad Hoc Networks Based on Self-Pruning," *Proc. INFOCOM '03,* Mar. 2003.
[27] F. Dai and J. Wu, "An Extended Localized Algorithm for Connected Dominating Set Formation in Ad Hoc Wireless Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 10, pp. 908-920, 2004.
[28] M.V. Marathe, H. Breu, H.B. Hunt III, S.S. Ravi, and D.J. Rosenkrantz, "Simple Heuristics for Unit Disk Graphs," *Networks,* vol. 25, pp. 59-68, 1995.
[29] P.J. Wan, K. Alzoubi, and O. Frieder, "Distributed Construction of Connected Dominating Set in Wireless Ad Hoc Networks," *Proc. IEEE INFOCOM '02,* vol. 3, pp. 1597-1604, June 2002.
[30] K. Langendone and N. Reijers, "Distributed Localization in Wireless Sensor Networks: A Quantitative Comparison," *Computer Networks,* vol. 43, pp. 499-518, 2003.
[31] D.R. Kowalski and A. Pelc, "Deterministic Broadcasting Time in Radio Networks of Unknown Topoloby," *Proc. 43rd Ann. IEEE Symp. Foundations of Computer Science (FOCS '02),* 2002.
[32] R. Gandhi, S. Parthasarathy, and A. Mishra, "Minimizing Broadcast Latency and Redundancy in Ad Hoc Networks," *Proc. Fourth ACM Int'l Symp. Mobile Ad Hoc Networking and Computing (MOBIHOC '03),* pp. 222-232, June 2003.

Hai Liu received the BSc (1999) and MSc (2002) degrees in applied mathematics from the South China University of Technology, China. He is currently a PhD candidate in the Department of Computer Science at the City University of Hong Kong. His research interests include distributed systems, wireless networks, and mobile computing.

Xiaohua Jia received the BSc (1984) and MEng (1987) degrees from the University of Science and Technology of China and received the DSc (1991) degree in information science from the University of Tokyo, Japan. Professor Jia is currently associated with the Department of Computer Science at the City University of Hong Kong. His research interests include distributed systems, computer networks, WDM optical networks, and Internet and mobile computing. He is a senior member of the IEEE.

Peng-Jun Wan received the PhD degree (1997) from the University of Minnesota, the MS degree (1993) from the Chinese Academy of Science, and the BS degree (1990) from Tsinghua University. His research interests include wireless networks and optical networks.

Xinxin Liu received the BSc (1998) degree from the Huazhong University of Science and Technology and the MEng (2003) degree from the China Ship Research & Development Academy. Currently, he is a PhD student at Wuhan University. His research interests include embedded systems and computer networks.

Frances F. Yao received the BSc (1969) degree from National Taiwan University and the PhD (1973) degree in mathematics from the Massachusetts Institute of Technology. She is currently head of the Department of Computer Science at the City University of Hong Kong. Her research interests include combinatorial and geometric algorithms, energy-efficient computing, and sensor networks. She is a fellow of the AAAS.