SOS: A Safe, Ordered, and Speedy Emergency Navigation Algorithm in Wireless Sensor Networks

Andong Zhan*† Fan Wu* Guihai Chen*
*Shanghai Key Laboratory of Scalable Computing and Systems,
Department of Computer Science and Engineering, Shanghai Jiao Tong University, China
†Department of Computer Science, Johns Hopkins University, USA
azhan2@jhu.edu, {fwu,gchen}@cs.sjtu.edu.cn

***Abstract*—Wireless sensor networks can play an important role in detecting disasters and navigating the personnel out of dangerous areas. In emergency navigation applications, personal safety and evacuation time of human beings are commonly considered criteria. Consequently, existing works mainly focus on finding the shortest safe path for each person. However, in practice, the number of people may be much larger than the safety capacities of the computed evacuation paths. This may lead to congestion on the evacuation paths, and result in longer evacuation time and even casualty increment. In this paper, we model the problem of emergency navigation as a network flow scheduling problem and present a distributed algorithm, namely SOS, to solve it. We also conduct extensive and large-scale simulations to evaluate the performance of SOS. Numerical results show that SOS achieves superior performance to existing approaches in terms of average and last evacuation time, with low overhead.**

Keywords: wireless sensor networks, emergency navigation, network flow

I. Introduction

More than 250,000 people were killed by natural disasters worldwide last year, making 2010 one of the deadliest years in more than a generation, UN Secretary-General Ban Ki-moon said in a report on Feb. 10, 2011 [1]. In addition, above ninety terrorist attacks worldwide brought great danger to public security last year [2]. Both natural and man-made disasters cause heavy casualties, great economic losses, and panics. Evacuation techniques are highly needed to navigate the personnel out of danger quickly. Wireless sensor networks (WSNs) can be used to save human-beings by detecting emergencies in a large-scale area and alarming the people inside. Furthermore, sensor networks can provide navigation services so that internal users can evacuate safely from the dangerous areas as soon as possible.

Previous emergency navigation algorithms in WSNs [5], [8], [9], [12] attempt to search the shortest safe paths for users. However, they cannot guarantee safe evacuation for all the people in crowd areas, e.g., at downtown, airport, rail station, etc. Furthermore, existing algorithms only utilize the information of dangerous areas, while ignoring the users' information, like locations and quantities. This may cause congestion and even more casualties, when too many people are navigated to the same path. Therefore, it is critical to efficiently schedule people to achieve ordered evacuation. Without order, safety and speediness can not be guaranteed.

In this paper, we study the problem of achieving safe, ordered, and speedy emergency navigation using WSNs. WSNs can not only detect danger, but also sense the people through their communication devices. Our main idea is to integrate the information of dangerous areas and users to provide a best effort emergency navigation schedule. Therefore, we propose SOS (Safe, Ordered, and Speedy) emergency navigation algorithm. SOS can compute an optimal schedule for all the people based on their locations and situation of dangerous areas. The major contributions of our work can be summarized as follows:

- We are the first to take people's information into account in studying the problem of emergency evacuation and formulate it as a network flow problem.
- We propose SOS, a distributed network flow algorithm to compute an optimal schedule to evacuate people out of danger. SOS is fit for large-scale sensor deployment with high efficiency and scalability.
- We extensively evaluate SOS's performance. Numerical results show that SOS achieves superior performance to existing works in terms of average and last evacuation time with low overhead.

The rest of the paper is organized as follows.In Section II, we discuss related work. In Section III, we describe the problem specification. In Section IV, we show our design principles. Finally, we present evaluation results of our algorithm in Section V and conclude in Section VI.

II. Related Work

Several emergency navigation algorithms have been proposed by utilizing large-scale sensor networks to monitor disasters and navigate people out of danger.

Li et al. [9] were the first to propose distributed algorithms for guiding navigation across a sensor network. They establish an artificial potential field by flooding from dangerous nodes. Based on that, a minimum exposure path to the goal, which has the least sum of the safe values of nodes in this path, can be maintained. But the flooding brings heavy network load especially for large-scale sensor networks. Tseng et al. [12] proposed an emergency navigation algorithm based on TORA [11]. It increases the weight of safety in path selection so as to avoid the navigation path entering hazardous areas. But it also depend on flooding all over the network. Furthermore, every sensor node has to know their own location. In order to lower

978-1-4577-0638-7 /11/$26.00 ©2011 IEEE

TABLE I
THE COMPARISON OF NAVIGATION ALGORITHMS

Navigation Algorithm	Path Establishment	Path Update	Node Location	Experiment	Network Load	Scalability
Qun Li et al. [9]	artificial potential field	overall & periodic	unneeded	simulation & implementation	high	low
Tseng et al. [12]	TORA [11]	overall & periodic	required	simulation	medium	low
Buragohain et al. [5]	skeleton graph	N/A	required	simulation	medium	medium
Mo Li et al. [8]	road map	local & dynamic	unneeded	simulation & implementation	low	high

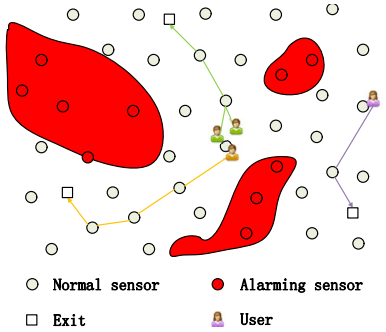


Fig. 1. Emergency navigation via a sensor network.

the network load, Buragohain et al. [5] abstracted the network by establishing a skeleton graph based shortest path and minimum exposure path. In addition, inspired by MAP routing algorithm [4], Li et al. [8] proposed road map based navigation algorithm which does not need location information. Their algorithm provides the user the safest path, which means its most dangerous node is safer than ones in other paths, and design a dynamic updating algorithm to decrease the update load when emergency changes. Table I compares the current emergency navigation algorithms.

However, the related works above ignore the number of people on the path and the path's safety capacities. In some emergency, excessive pedestrians walking or running in a narrow path may cause congestion and even stampedes, which largely increases casualties.

III. Problem Specification

When an emergency (such as fire, poison gas, explosive, etc.) happens, people should be navigated to safe zones/exits as soon as possible. Fig. 1 shows such a sample scenario – people are navigated out of danger by a sensor network. Once the sensor network detects a danger (e.g., some sensors detect their surrounding temperatures exceed a safety threshold), it searches safe paths based on the situation of the emergency and people's information, and schedule all the people out of danger as soon as possible. Meanwhile, the schedule should adapt to the ever-changing dangerous areas and current evacuation status. In this paper, we model the emergency evacuation problem as a joint path planning and pedestrian flow scheduling problem and leave the navigation adaptation based on dynamic changes of dangerous areas as our future work.

A. Assumptions

We assume a wireless sensor network can detect dangerous areas (red regions in Fig. 1).A node triggers a "Yes" alarm if it resides in a dangerous area (red nodes), or triggers a "No" signal if it is in a safe area (gray nodes). We assume users carry wireless communication devices, which enable them to "talk" with nearby sensors, e.g., 802.15.4 compatible PDA or smart phone. We also assume that the communication devices can track surrounding sensor nodes by measuring the strength and direction of their wireless signals [10].

For a user, a navigation path is a sequence of sensor nodes and must be safe. Existing works ignore the *safety capacity* of a path, i.e., the number of people passing through it safely in a unit of time. This weakness undermines the usage of existing solutions in practical settings that have a large number of pedestrian users in a given area. We propose a more practical definition for the safe path and its safety capacity:

Definition 1 (Safe Path): A path $P = \{s_1, s_2, \ldots, s_n\}$ is a *safe path* if and only if $\forall s_i \in P,\ d_i \geq d_\Gamma$, where d_i is the distance between sensor node s_i and its nearest alarming neighbor node, and d_Γ is the *safe distance threshold*.

So P can be described as a node sequence from s_1 to s_n and d_Γ is determined by a specific scenario.

Definition 2 (Safety Capacity): For each sensor node s_i on a safe path, its *safety capacity* u_i is the maximum number of people passing through it safely per unit time.

The safety capacity u_i for a node s_i can be determined by d_i and the property of the ground around the node s_i. In this work, we use two representative functions to describe nodes' safety capacities:

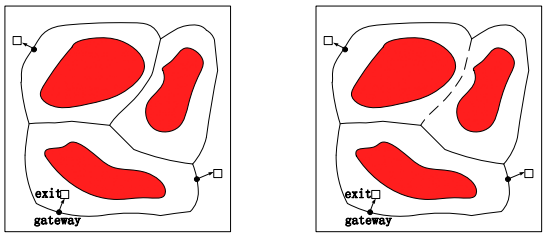
- Constant function: $u_i = c$, which is the same for all the nodes, e.g., nodes in a passageway.
- Linear function: $u_i = kd_i, k > 0$, which is linear with the the distance between s_i and its nearest alarming neighbor node, e.g., sensors deployed in a grassland.

More complicated functions can also be applied to describe the safety capacity function. However, our proposed algorithm can always computes the optimal evacuation schedule. Therefore, we focus on the two functions in this paper.

B. Objective and Requirements

The objective of a successful navigation is to schedule all the users to bypass the dangerous areas safely, and finally evacuate to pre-known exits as soon as possible. Fig. 1 shows such an example schedule that leads all users to the exits. The scheduling process is in a fully distributed manner without any centralized authority, such as a base station. Each user is hand-off guided by sensor nodes along the scheduled path.

We mainly have the following requirements on the emergency navigation algorithm:



(a) Medial axis graph. (b) Safe medial axis graph.

Fig. 2. Graph model

- All the escape paths given by the navigation algorithm should be safe paths.
- All the users should evacuate from the emergency area *orderly* without congestion, which means for any node s_i on escape paths, $p_i \leq u_i$, where p_i is the number of people around s_i in any time unit.
- The navigation algorithm should minimize the *total evacuation time*. Here a user's evacuation time is the duration from the emergency occurrence to the time when he or she reaches an exit.
- To lower the cost, sensor nodes do not require geographical location information, i.e., the sensor nodes are not equipped with GPS.
- The navigation algorithm should be efficient and scalable, i.e., the communication overhead should be small.

IV. Design Principles

Similar to Li et al.'s work [8], SOS is based on a medial axis graph [4] giving the emergency areas. Using it as the backbone, we formulate the navigation schedule problem as a network flow problem and design a distributed algorithm to solve it. In the following, we present the design principles in three steps: constructing the medial axis graph, formulating the navigation schedule problem, and designing the distributed algorithm.

A. Constructing the Medial Axis Graph

We assume that the whole area E is covered by a sensor network, and the dangerous area is denoted by D. In our navigation, the users can only move in the area $R = E \backslash D$. Fig. 2(a) shows a medial axis graph in R. In the graph, red zones are dangerous areas, and black lines between the dangerous areas and the terrain border form a medial axis graph[1]. We define the *safe medial axis graph* as follow.

Definition 3 (Safe Medial Axis Graph): A sensor node s_i is in a safe medial axis graph if and only if $d_i \geq d_\Gamma$ and the distances (in terms of number of hops) from s_i to its two closest dangerous areas are the same (e.g., solid line in Fig. 2(b)).

Let N_S denote the set of nodes in a safe medial axis graph. A safe medial axis graph can be a part of cell boundaries of a Voronoi diagram [4], and captures the topology features of the safe region R. The N_S separates the safe area R into several cells and becomes the border of them. Furthermore, the nodes in N_S are the local safest nodes. Consequently, we can use them as the backbone of our navigation paths.

N_S can be identified through local flooding. First, the border nodes of all the dangerous areas broadcast their dangerous area IDs (we denote the package as *dBroadcast* and dangerous area ID as *dID*). Every dBroadcast has a life time of max_hop. Second, every safe node records the numbers of hops to two closest dangerous areas. If the two numbers are the same, it can be identified as a medial node.

After constructing N_S, we link exits and user nodes to the medial nodes. (We denote a safe node with people around it as a user node.) If an exit is not a medial node, we should establish a path from N_S to the exit. Since N_S separates the whole safe area R into several cells, i.e., the medial nodes, are the farthest nodes in every cell. Lemma 3.1 in [8] guarantees that we can successfully build a path connecting an exit and N_S backbone without halting at an intermediate point of local minimum. We call the intersection of the path and N_S as *gateway*. Users arrive at a gateway and then go to the exits along the paths, as shown in Fig. 2(a).

A user node can count the number of people around it by their handsets. If it is not a medial node, it finds a path to N_S by the same way that connects an exit. The path connecting the user node intersects N_S backbone at a point called *source node*, which should record the hops from the user node and the number of people. After constructing the N_S backbone and connecting exits and user nodes, the remaining step is to find an optimal schedule to navigate users from source nodes to gateways which minimizes the total evacuation time.

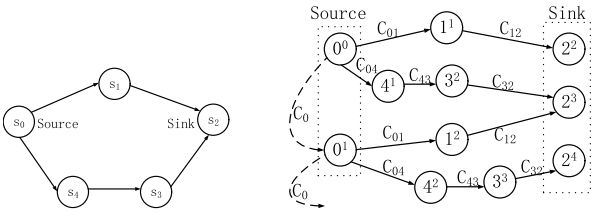
B. Formulating the Navigation Schedule Problem

A scheduling in N_S builds a path from a source node to a gateway for every user. Simple greedy and personal schemes (e.g., navigating users to the closest gateway, choosing the safest path for every user, or taking both safety and distance into account) can not guarantee the optimal scheduling. When the number of users is large and the capacities of safe paths are low, congestion may occur, and greatly increases the evacuation time resulting in more casualties. Therefore, we study the problem from a network flow perspective. However, the navigation scheduling problem can not be directly formulated as a traditional network flow problem. The reasons are two-sided:

- Time should be considered, e.g., the congestion only takes place at a node when the number of users around it within a unit time is larger than the safety capacity of the node.
- Waiting is inevitable when the number of users is larger than the maximum safety capacity of the network.

As shown in Fig. 3(a), N_S has 5 nodes. s_0 is a source node and s_2 is a sink node (a gateway node). An arc between two nodes means they are neighbor nodes. We simplify the time cost between two nodes to 1. If we want to navigate n users from s_0 to s_2 at time $t = 0$, the best choice is the path $s_0 \rightarrow s_1 \rightarrow s_2$, whose time cost is 2. However, if n is larger than the safety capacity of this path, the excess users can be

[1]Without loss of generality, as in [8], we assume that the area out of the sensor field is dangerous.



(a) Model I (b) Model II

Fig. 3. Network flow models

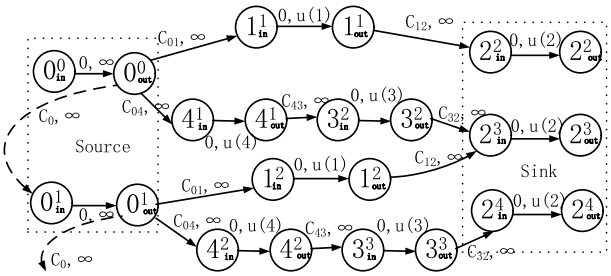


Fig. 4. Final Network Flow Model.

navigated to the other path, i.e., the path $s_0 \rightarrow s_4 \rightarrow s_3 \rightarrow s_2$. Furthermore, if n is larger than the total safety capacities of the both paths, i.e., the maximum flow, the excess users should wait at least one time unit, which has not been considered in any traditional network flow problem. In addition, let's assume that n_0 is the number of users around s_0 at time $t = 0$, and n_2 is the number at time $t = 2$. Even if the sum of n_0 and n_2 is larger than the safety capacity of the path $s_0 \rightarrow s_1 \rightarrow s_2$, all the users can be navigated through the path if both n_0 and n_2 are smaller than the safety capacity of the path. This is because in every unit of time, the number of users around s_0, s_1 or s_2 is less than the nodes' safety capacity. However, there is no concept of time in traditional network flow problem. Therefore, we propose a novel model with time parameter.

We create a graph $G(\mathcal{V}, \mathcal{E})$ and denote vertex $i^t \in \mathcal{V}, i \in N$ as the state of sensor node s_i at time t, and directed edge $edge(i, j, t) \in \mathcal{E}$ as the arc from vertex i^t to vertex j^{t+1}. A sensor node may have several vertices so as to represent the number of users in different time units. For example, if node s_0 has two users at $t = 0$, we can denote it as vertex 0^0 with $excess = 2$. Meanwhile,waiting can be described by this model, e.g., if vertex i^t has some users who should wait for one unit of time, we use $edge(i, i, t)$ to denote the number of users who reach i^{t+1} from i^t by waiting a unit of time. Thus, we can draw Fig. 3(a) as Fig. 3(b), and the navigation schedule problem can be formulated as a linear program:
Minimize:

$$\sum_{i,j,t} c_{ij} \times x_{i,j,t} + \sum_{i,t} c_i \times x_{i,t} \tag{1}$$

Subject to:

$$\sum_{i \notin S,t} x_{S,i,t} = N_u \tag{2}$$

$$\sum_{i \notin D,t} x_{i,D,t} = N_u \tag{3}$$

$$\sum_{j} x_{j,i,t-1} + x_{i,t-1} = \sum_{j} x_{i,j,t} + x_{i,t} \quad \forall i, t \tag{4}$$

$$x_{ijt} \geqslant 0 \quad \forall i, j, t \tag{5}$$

$$\sum_{i} x_{i,j,t} \leqslant u(j) \quad \forall j, t \tag{6}$$

Here formulae 1-6 are similar to a minimum cost flow problem. $x_{i,j,t}$ denotes the flow from vertex i^t to $j^{t+1}, i \neq j$, i.e., the number of users traveling from sensor node s_i to s_j at time t. $x_{i,t}$ is short for $x_{i,i,t}$ and denotes the number of users waiting around s_i at time t. c_{ij} is the time cost on traveling from s_i to s_j, and c_i is the time cost of waiting at s_i. Our goal is to minimize the sum of the traveling and waiting time costs. Constraint (2) shows that the number of users moving out of all source nodes should be N_u, which is also the number of users in the whole safe area. S is the set of all the source nodes. Constraint (3) indicates that all the users should reach destination nodes, i.e., gateway nodes in N_S. D is the set of destination nodes. Constraint (4) reflects that the number of users reaches vertex i^t at time $t - 1$, including the ones moved from other vertices and the ones waited around the same sensor node, is equal to the number of users moving out of i^t or waiting at i^t at time t. Constraint (5) guarantees any flow is not negative. Constraint (6) ensures the number of users at any node at any time is less than the safety capacity of the sensor node. Since the flow through a vertex is limited in this minimum cost flow problem, as shown in Constraint (6), we can further convert such problem to a traditional network flow problem which has only edge capacity constraints by the well known node-splitting method in network flow problems [3]. The final network flow model are shown in Fig. 4.

C. Designing the Distributed Algorithm

In this section, we present a distributed algorithm to find the optimal navigation scheduling for all the users. To understand the algorithm better, we first provide a distributed algorithm to solve a simpler problem which only gives a feasible navigation scheduling without minimizing the total evacuation time. Then we will show that with several iterations of this algorithm, the minimum cost flow can be achieved.

The minimum cost flow problem described in Section IV-B aims to achieve the optimal navigation scheduling which minimizes the total evacuation time. Let's simplify this goal by navigating all the users to the exits, so this problem becomes a maximum flow problem, i.e., maximize the flow from source nodes to the destination. There are several efficient algorithms for the maximum flow problem, e.g., Ford-Fulkerson augmenting path algorithm and the push-relabel algorithm [6]. Here we adopt the push-relabel structure which is a natural distributed algorithm.

The basic idea of push-relabel algorithm is to iteratively push excess flow of a higher vertex to neighboring vertices with lower heights, or relabel itself, i.e., increase its height, when a push can not be performed. The push and relabel operations repeat until no excess flow exists in the network. The details of push-relabel algorithm can be accessed from [3],

[7]. Here we mainly discuss the differences between original push-relabel algorithm and our algorithm.

First, we use node-splitting method [3] to convert the capacity limitation from vertices to arcs. Every vertex i^t is separated into two vertices, the input node i^t_{in} and the output node i^t_{out}. An arc (i^t_{in}, i^t_{out}) links the two vertices with cost $c = 0$ and capacity $u = u(i^t)$. In addition, the output vertex connects every neighboring input vertex j^{t+1}_{in} by an arc (i^t_{out}, j^{t+1}_{in}) with cost c_{ij} and capacity $u = \infty$.

Second, we describe waiting in a simple way. For an output vertex i^t_{out}, we add an arc (i^t_{out}, i^{t+1}_{in}) to connect the output vertex i^t_{out} and the input vertex i^{t+1}_{in} at the next time unit with cost $c = c_i$ and capacity $u = \infty$.

Algorithm 1 Navigation algorithm for node i

```
Collect node information of d(i) and u(i)
if ∃n(i^t) > 0 then
    Establish vertex i^t_in and i^t_out
    Set height h(i^t_in) = 2 × d(i) and h(i^t_out) = 2 × d(i) − 1
    Set excess e(i^t_in) = 0 and e(i^t_out) = n(i^t)
    while e(i^t_{in/out}) > 0 do
        Call Push-relabel(i^t_{in/out})
    end while
end if
Record the navigation schedule, i.e, the time and the number of users to
certain neighbor node
```

Algorithm 2 Push-relabel Algorithm for vertex v

```
if e(v) > 0 then
    while e(v) > 0 and ∃ arc(v, w) s.t. h(v) = h(w)+1 and the residual
    capacity of arc(v, w), cap(v, w) > 0 do
        Push amount of y = min{e(v), cap(v, w)} to w through
        arc(v, w) by sending a message to w
        e(w) = e(w) + y, e(v) = e(v) − y
        update cap(v, w)
    end while
    if e(v) > 0 then
        Update h(v) as 1 + min{h(w) : cap(v, w) > 0}
        Broadcast h(v) to neighbor nodes
    end if
end if
```

The details of our algorithms are shown in Algorithm 1 and Algorithm 2. In the beginning, through the flooding from gateway nodes, each node i in N_S knows its number of hops to the closest gateway node $d(i)$. In addition, node i can calculate its safety capacity $u(i)$ based on the number of hops to the closest danger node hop_d. When a source node realizes that some users will come at time t (i.e., the number of them $n(i^t) > 0$), it establishes two vertices (the input vertex i^t_{in} and the output vertex i^t_{out}), and initializes their heights as the distance to sinks in the final network flow model. Then the source node sets the excess of i^t_{out} as $n(i^t)$. For node i, if the excess of one of its vertices $e(i^t_{in/out}) > 0$, the node calls the push-relabel algorithm. As shown in Algorithm 2, if vertex v has a neighbor node w to push, i.e., $\exists arc(v, w)\, s.t. h(v) = h(w)+1$, the push operation runs by sending a package to w. If not, the relabel operation begins, increases the height of vertex i to $1 + min\{h(w) : cap(v, w) > 0\}$, and broadcasts this update to its neighbor nodes. The push-relabel algorithm runs until the excess $e(v) = 0$.

Algorithm Performance: Asynchronous distributed push-relabel algorithm runs in $O(|\mathcal{V}|^2)$ times and uses at most $O(|\mathcal{V}|^2|\mathcal{E}|)$ message exchanges [7]. In our algorithm, the network size $|\mathcal{V}|$ depends on the network size of N_S m and the number of users n_u. It's easy to prove that $|\mathcal{V}| \leq mn_u$. If we assume that the size of the sensor network is n, the density is λ, the side length of the whole emergency area is L, and the maximum number of dangerous areas is n_d, then $m \leq n_d \times 4L = 4n_d\sqrt{\frac{n}{\lambda}}\lambda$, and $|\mathcal{E}| \leq \pi\lambda + 1$. Therefore, $O(|\mathcal{V}|^2) = O(|mn_u|^2) = O(nn_u^2)$, and $O(|\mathcal{V}|^2|\mathcal{E}|) = O(nn_u^2)$. Therefore, our algorithm runs in $O(nn_u^2)$ times and uses at most $O(nn_u^2)$ message exchanges.

If the minimum total evacuation time is needed instead of an arbitrary feasible scheduling, we can use the cost scaling algorithm proposed by Goldberg et al. in [7], which uses $O(log(|\mathcal{V}|C))$ iterations of push-relabel processes to refine the cost of the solution, where C is the maximum cost of any edge. In our problem, our edge cost is only one. Therefore, finding the minimum movement cost schedule takes $O(log(n^{\frac{1}{2}}n_u)$ times more computational time and message exchanges than finding an arbitrary feasible movement schedule.

V. Numerical Results

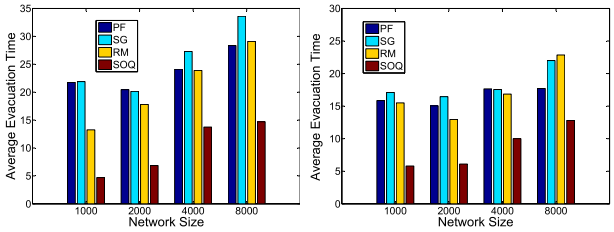
We evaluate the efficiency and scalability of SOS through large-scale simulation, and compare its performance with existing emergency navigation algorithms.

We randomly distribute sensor nodes in a square field, with an average connectivity degree of 28. In order to test the scalability, we change the size of this area but not the degree of the nodes. The number of nodes is from 1000 to 8000. We make 20 runs for every network size, and randomly create 1 to 5 groups of users in each run. The number of users in a group is created randomly between 1 and 50. In each run, we also randomly setup 1 to 5 exits and 3 to 6 dangerous areas. The size of each dangerous area is kept below 5% of the total field size. We set safe distance threshold $d_\Gamma = 1$ hop.

We compare the performance of SOS with the following three emergency navigation algorithms: PF — the potential field based approach [9]. We choose the function for calculating the potential value to be $1/dist^2$, which has been used all through the paper. SG — the skeleton graph based approach [5]. We choose the version based on adaptive skeleton graph, which has been shown superior to the uniform one in their paper. RM — the road map based approach [8].

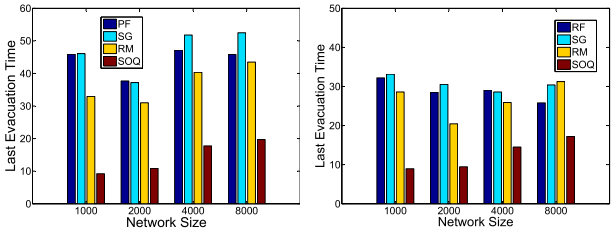
In our evaluation, we consider two capacity conditions: uniform capacity — the capacity of every sensor node is equal, and linear capacity — the capacity of a node in the N_S is linear with the number of hops from the node to the closest alarming node. Since the three compared approaches (PF, SG, and RM) do not specify the users' evacuation process, we let users move to the next sensor on their navigation paths, if doing so do not exceed the sensor's capacity. Otherwise, excess users need to wait at the current node.

We compare the four approaches using three metrics:



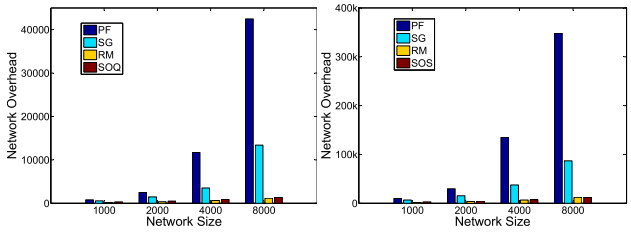
(a) Uniform capacity. (b) Linear capacity.

Fig. 5. Average evacuation time of the four evaluated approaches. SOS always achieves the shortest average evacuation time.



(a) Uniform capacity. (b) Linear capacity.

Fig. 6. Last evacuation time of the four evaluated approaches. SOS always achieves the shortest last evacuation time.



(a) Uniform capacity. (b) Linear capacity.

Fig. 7. Network overhead of the four evaluated approaches. SOS always achieves the smallest network overhead.

The average evacuation time reflects users' mean duration from emergency occurrence to safe evacuation. As shown in Fig. 5, the average evacuation time increases with the scale of the network. In uniform capacity networks (Fig. 5(a)), the average evacuation time of SOS is less than half of that of any other three approaches. In linear capacity networks (Fig. 5(b)), the average evacuation time of SOS is also the shortest one among all the four approaches. Comparing Fig. 5(a) and Fig. 5(b), we can see that the performance gaps between SOS and the other three decrease a little. This is because in linear capacity networks, the safest path is identified as the one with the largest capacity.

Fig. 6 shows the evaluation results of the last evacuation time for the four approaches. In uniform capacity networks (Fig. 6(a)), the last evacuation time of SOS is also less than half of that of any other three approaches. In linear capacity networks (Fig. 6(b)), the last evacuation time of SOS is also the shortest one among all the four approaches. Similarly, the gap between SOS and other approaches is smaller in linear capacity networks than that in uniform ones.

Finally, we evaluate the network overhead of the four approaches during the process of emergency evacuation. Here network overhead is simplified as the number of packets used in every navigation. Fig. 7 compares the network overhead of the four approaches in uniform capacity networks and linear capacity networks, respectively. PF produces the largest network overhead because it relies on flooding the entire network for potential value calculation and path establishment. Both SG and RM try to establish a subgraph to decrease the network overhead. We can see that the overhead of RM is much smaller than SG. In addition, although SOS adopts a more complex solution— push-relabel algorithm— the network overhead of SOS is still close to the that of RM. Evaluation results show that both RM and SOS are scalable, because their network overheads are not related to the size of the network.

VI. Conclusion

We have proposed SOS, a safe, ordered, and speedy emergency navigation algorithm in WSNs. To minimize users' evacuation time, we have converted the emergency evacuation problem to a traditional network flow problem and used push-relabel algorithm to solve it. Our results of large-scale simulations have shown that SOS is better than existing approaches in terms of average evacuation time, last evacuation time, and network overhead. In the future work, we will design an adaptation strategy for dynamic changes of danger and movements of users.

Acknowledgment

The work is partly supported by China NSF grants (60825205, 61073152, 61021062).

References

[1] [Online]. Available: http://news.xinhuanet.com/english2010/world/2011-02/10/c_13724927.htm
[2] [Online]. Available: http://en.wikipedia.org/wiki/List_of_terrorist_incidents
[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.
[4] J. Bruck, J. Gaoy, and A. Jiang, "Map: Medial axis based geometric routing in sensor networks," in *Proceedings of The Eleventh International Conference on Mobile Computing and Networking (MobiCom)*, Cologne, Germany, 2005.
[5] C. Buragohain, D. Agrawal, and S. Suri, "Distributed navigation algorithms for sensor networks," in *Proceedings of 25th Annual IEEE Conference on Computer Communications (INFOCOM)*, Barcelona, Spain, Apr. 2006.
[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. 2nd Ed. MIT Press and McGraw-Hill, 2001.
[7] A. Goldberg and R. Tarjan, "A new approach to the maximum-flow problem," *Journal of the ACM*, vol. 35, no. 4, pp. 921–940, 1988.
[8] M. Li, Y. Liu, and Z. Yang, "Sensor network navigation without locations," in *Proceedings of 28th Annual IEEE Conference on Computer Communications (INFOCOM)*, Rio de Janeiro, Brazil, Apr. 2009.
[9] Q. Li, M. D. Rosa, and D. Rus, "Distributed algorithms for guiding navigation across a sensor network," in *Proceedings of the Ninth International Conference on Mobile Computing and Networking (MobiCom)*, San Diego, CA, 2003.
[10] D. Niculescu and B. Nath, "Ad hoc positioning system (APS) using AOA," in *Proceedings of 22nd Annual IEEE Conference on Computer Communications (INFOCOM)*, San Francisco, CA, Apr. 2003.
[11] V. Park and M. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *Proceedings of 16th Annual IEEE Conference on Computer Communications (INFOCOM)*, Kobe, Japan, Apr. 1997.
[12] Y. Tseng, M. Pan, and Y. Tsai, "A distributed emergency navigation algorithm for wireless sensor networks," *IEEE Computers*, vol. 39, no. 7, pp. 55–62, July 2006.