Smartphone-Based Self Rescue System for Disaster Rescue

Xitong Zhou
zhou34@marshall.edu

SMARTPHONE-BASED SELF RESCUE SYSTEM FOR DISASTER RESCUE

A thesis submitted to
the Graduate College of
Marshall University
In partial fulfillment of
the requirements for the degree of
Master of Science
In
Computer Science
by
Xitong Zhou
Approved by
Dr. Cong Pu, Committee Chairperson
Dr. Wook-Sung Yoo
Dr. Paulus Wahjudi

Marshall University
December 2018

APPROVAL OF THESIS

We, the faculty supervising the work of Xitong Zhou, affirm that the thesis, *Smartphone-Based Self Rescue System for Disaster Rescue* meets the high academic standards for original scholarship and creative work established by the M.S. in Computer Science and the College of Information Technology and Engineering. This work also conforms to the editorial standards of our discipline and the Graduate College of Marshall University. With our signatures, we approve the manuscript for publication.

[illegible] ____________________ Committee Chairperson Nov 29. 2018 Date

[illegible] ____________________ Committee Member NOV 29, 2018 Date

[illegible] ____________________ Committee Member Nov. 29, '2018 Date

ACKNOWLEDGMENTS

First of all, I would like to thank the help from my advisor Dr. Cong Pu, Dr. Wook-Sung Yoo and Paulus Wahjudi for the master thesis.

Furthermore, I am very grateful to my parents and friends who supported me during the stressful master thesis process.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Recent ubiquitous earthquakes have been leading to mass destruction of electrical power and cellular infrastructures, and deprive the innocent lives across the world. Due to the wide-area earthquake disaster, unavailable power and communication infrastructure, limited man-power and resources, traditional rescue operations and equipment are inefficient and time-consuming, leading to the golden hours missed. With the increasing proliferation of powerful wireless devices, like smartphones, they can be assumed to be abundantly available among the disaster victims and can act as valuable resources to coordinate disaster rescue operations. In this paper, we propose a smartphone-based self-rescue system, also referred to as RescueMe, to assist the operations of disaster rescue and relief. The basic idea of RescueMe is that a set of smartphones carried by survivors trapped or buried under the collapsed infrastructure forms into a one-hop network and sends out distress signals in an energy-efficient manner to nearby rescue crews to assist rescue operations. We evaluate the proposed approach through extensive simulation experiments and compare its performance with the existing scheme TeamPhone. The simulation results show that the proposed approach can significantly reduce the schedule vacancy of broadcasting distress signals and improve the discovery probability with very little sacrifice of network lifetime, and indicate a potentially viable approach to expedite disaster rescue and relief operations.

I. INTRODUCTION

Unexpected natural disasters such as tornadoes, earthquakes, hurricanes, and tsunamis have been rising dramatically in recent years. In particular, earthquakes tremendously kill innocent lives and damage the environment around the globe, and the epicenter of an earthquake can occur anywhere and now no place would be safe from ubiquitous earthquakes. For example, as reported by Ansari (2016) a 5.6-magnitude earthquake struck Oklahoma and impacted six neighboring states in the U.S. on September 04, 2016. As reported by Karimi and Ansari (2016) the Ecuador earthquake (April 16, 2016) left a 272 death toll and more than 2,500 injured. An earthquake often happens in a flash but has the potential to massively destruct the infrastructures, buildings, and homes in a short period of time. As Nishiyama, Ito and Kato (2014) noted, after the disaster, it was impossible for disaster victims to utilize their communication devices, such as smartphone, tablet, or laptop, to notify their families and friends of their safety and confirm the safety of their loved ones since the communication infrastructures were physically damaged or lacked the energy necessary to operate. More importantly, since many people could be trapped beneath the rubble and brick, the victims may have a large chance to survive if they are located and rescued within "Golden 72 Hours." If a severely injured person does not receive care or medical treatment quickly, the probability of survival rapidly decreases. Thus, in order to minimize casualties and save innocent lives across the world, it is essential to plan and conduct expedited disaster rescue operations.

When an earthquake occurs, the rescue teams or planners of disaster rescue and relief mainly suffer from the following issues. First, since the current disaster situation of sudden earthquake may not be available, it is difficult to make a plan or decision on the priority of rescue operations in terms of the focused rescue areas, the distribution of rescue teams, or the allocation of equipment. Second, the impact areas are admittedly wide, ranging from a few miles to several U.S. states, but the number of rescue teams and man-power are very limited in reality. Third, due to the collapse of power and communication infrastructure, the impacted areas become a blackspot where Wi-Fi and 4G-LTE services are not available and the affected area is cut off from the outside. Last but not least, most rescue teams still heavily rely on the traditional operations and equipment, such as detection dogs, video cameras, or sound sensors. In summary, the traditional rescue operations and equipment are inefficient and time-consuming, leading to the golden hour missed.

On the other hand, smartphones have become an essential electronic device that people always carry for communication and social connection, or place them where they can be easily and immediately accessed. It has been predicted in Number of smartphone users in the U.S. 2010-2022 (n.d.) that the number of smartphone users in the United States is estimated to reach 148.68 million in 2019, and the number of smartphone users worldwide forecasts to exceed 2 billion by that time. Thus, with the increasing proliferation of smartphones, they can be assumed to be abundantly available among the disaster victims and act as valuable resources to coordinate rescue operations. For example, according to Bengtsson, Lu, Thorson, Garfield and Von Schreeb (2011), after the Haiti earthquake in 2010, there were approximately 2.8 million active mobile subscribers out of 10 million inhabitants contributing data for tracking the movement of population in the affected region.

According to these above reasons and conditions, this paper is to develop a novel disaster self-rescue system incorporating increasingly popular smartphones in the disaster area. Our major contribution is summarized in the following:

- We propose a smartphone-based self-rescue system, also referred to as RescueMe, to assist the operations of disaster rescue and relief. The basic idea of RescueMe is that a set of smartphones carried by survivors trapped or buried under the collapsed infrastructure forms into a one-hop network and sends out distress signals in an energy-efficient manner to nearby rescue crews to assist rescue operations.

- We develop a customized simulator framework and implement the proposed scheme for experiment study. For performance comparison, we revisit an existing approach TeamPhone proposed by Lu, Cao and Porta (2016), and modify it to work in the developed simulation framework.

We compare and analyze the performance of RescueMe and TeamPhone in terms of network lifetime and schedule vacancy of broadcasting distress signals through extensive simulation experiments. The simulation results show that the proposed approach can significantly reduce the schedule vacancy of broadcasting distress signals and improve the discovery probability with very little sacrifice of network lifetime, and indicate a potentially viable approach to expedite disaster rescue and relief operations.

The rest of the paper is organized as follows. The prior approaches are summarized and analyzed in Section II. The proposed smartphone-based self-rescue system is presented in Section III. Section IV presents extensive simulation results and analyses. Finally, we conclude the paper in Section V.

II. RELATED WORK

Using smartphones for constructing disaster recovery networks and assisting the operations of disaster rescue and relief has been extensively explored in the last decade.

Pal and Kant (2018) developed a mechanism to enable the devices to discover their neighbors autonomously and transmit data of disaster-affected areas by different networks to Wi-Fi access points using a smartphone-based Wi-Fi tethering technique. As reported by Raj, Kant and Das (2014) a novel architecture called energy aware disaster recovery network using Wi-Fi tethering is proposed to create the desired network infrastructure using wireless devices. The basic idea is to make use of Wi-Fi tethering technology ubiquitously available on wireless devices, like smartphones and tablets, to set up an ad hoc network for data collection in disaster scenarios. Ray, Sinha and Ray (2015) proposed a smartphone-based post-disaster management mechanism in the disaster affected areas using the concepts of Wi-Fi tethering, where smartphones in the affected areas may turn themselves into temporary Wi-Fi hotspots to provide Internet connectivity and important communication abilities to nearby Wi-Fi-enabled user devices. Nishiyama et al. (2014) reported that the concept of multihop device-to-device communication network systems integrating with different wireless technologies is proposed to deliver messages using only users' mobile devices, send out emergency messages from disconnected areas, and share information among people gathered in evacuation centers. Luo, Liu, Gao and Lu (2014) have noted that robot snakes with hyper-redundant body and unique gaits is proposed to offer a promising solution to search and rescue applications in the disaster. An unmanned aerial vehicles-aided disaster rescue system is proposed to locate possible victims in the research of Ho, Chen and Chen (2015), where unmanned aerial vehicles fly around the disaster area and sniff out wireless signals from any mobile devices to support the search team to narrow down the search area within meters.

The research of Hossain, Ray and Sinha (2016) presents a smartphone-assisted victim localization method in which smartphones belonging to trapped victims and other people in disaster affected areas can self-detect the occurrence of a disaster incident by monitoring the radio environment and can self-switch to a disaster mode to transmit emergency help messages with their location coordinates to other smartphones nearby. To locate other neighboring smartphones also operating in the disaster mode, each smartphone runs a rendezvous process. As Zamora, Suzuki and Kashihara (2017) noted, an application, also referred to as SOSCast, is proposed to propagate SOS messages from trapped survivors through a direct communication between smartphones. By collecting SOS messages that include significant information such as their name, state, and location, rescuers can estimate the locations of the survivors. Without relying on any infrastructure, the research of Al-Sadi and Awad (2017) presents a new algorithm that allows the smartphones of the rescuers and victims to seamlessly collaborate in order to estimate the locations of the victims by using both the received signal strength indicator of the Wi-Fi signals and the GPS information of the rescuers' smartphones. Hossain and Ray (2018) propose a smartphone and IoT devices-assisted emergency and recovery method in a post-disaster environment, where smartphones can utilize the IoT devices in the disaster affected areas to successfully relay the emergency messages to other smartphones.

In research of Lu, Cao, and La Porta (2016), by bridging the gap among different kinds of wireless networks, a system called TeamPhone is proposed to provide smartphones the capabilities of communications in disaster recovery. TeamPhone consists of two components: a messaging system and a self-rescue system. The messaging system integrates cellular networking, ad-hoc networking and opportunistic networking seamlessly, and enables communications among rescue workers. The self-rescue system energy-efficiently groups the smartphones of trapped survivors and sends out emergency messages so as to assist rescue operations. However, the self-rescue system does not consider that each smartphone of trapped survivors may carry different amounts of residual energy, and the smartphone with less residual energy may turn off quickly because of frequently broadcasting emergency messages. Thus, the schedule of sending out emergency messages should be dynamically adjusted accordingly when the network topology changes because of the death of certain smartphones.

In summary, various kinds of smartphone-based wireless communication technologies and hybrid networks have been widely investigated for disaster rescue and relief. However, to the best of author's knowledge, the proposed research focusing on dynamically adjusting the schedule of sending out distress signals according to the change of network topology is new.

III. THE PROPOSED SMARTPHONE-BASED SELF RESCUE SYSTEM

In this section, we first present an overview of the proposed smartphone-based self-rescue system, also referred to as RescueMe, and then discuss the RescueMe and its corresponding techniques in details.

A. OVERVIEW OF THE RESCUEME

With the proliferation of powerful smartphones, they can be assumed to be abundantly available among the trapped survivors in the affected region and act as valuable resources. Most smartphones are equipped with both Wi-Fi and Bluetooth transceivers. Wi-Fi technology allows a smartphone to connect to a wireless local area network (WLAN) based on the IEEE 802.11 standard. Wi-Fi running in the 2.4 GHz radio band covers a larger range than 5.0 GHz. Unlike Wi-Fi consuming high battery power, Bluetooth wireless technology (BWT) is primarily designed for a short range communication with low-power consumption. With the support of Wi-Fi or BWT, survivors even trapped under the rubble can periodically send out a distress signal using their smartphone. If each smartphone continuously stays awake and sends out a distress signal, this will increase discoverability and reachability of trapped survivors. However, continuously broadcasting a distress signal can quickly drain the battery, and rescue operations may even last for days after disasters occur. Thus, how to collaborate smartphones of survivors trapped under the collapsed infrastructure to send out distress signals in an energy-efficient manner is a challenge problem.

In this paper, we propose a smartphone-based self-rescue system, also referred to as RescueMe, to assist the operations of disaster rescue and relief in the disaster area. The basic idea of RescueMe is that a set of smartphones (later nodes) carried by survivors trapped or buried under the collapsed infrastructure forms into a one-hop network and sends out distress signals in an energy-efficient manner to nearby rescue crews to assist rescue operations. After detecting a seismic signal, the nodes can automatically enter into the self-rescue mode, or after disaster occurs, the trapped survivors can click a rescue app to enable the nodes to enter into the self-rescue mode. In the self-rescue mode, each node broadcasts a one-time Hello message, overhears bypassing Hello messages, and constructs a one-hop network. The one-hop network consists of cliques, in which the direct connection exists between every two nodes, and each node can communicate with every other node. In order to extend the battery lifetime through reducing energy consumption, the nodes in the one-hop network wake up alternatively in a coordinated way and send out distress signals to discover nearby rescue crew. In addition, each node may be equipped with different amounts of battery energy when the disaster happens, and the node with less amount of energy will turn off quickly. Thus, the wake-up schedule of broadcasting a distress signal should be dynamically adjusted in response to the change of one-hop network topology. In the following, we investigate three major issues to implement the RescueMe: (i) how to construct the one-hop network; (ii) how to determine the wake-up schedule of broadcasting distress signals; and (iii) how to dynamically adjust the wake-up schedule according to the change of network topology.

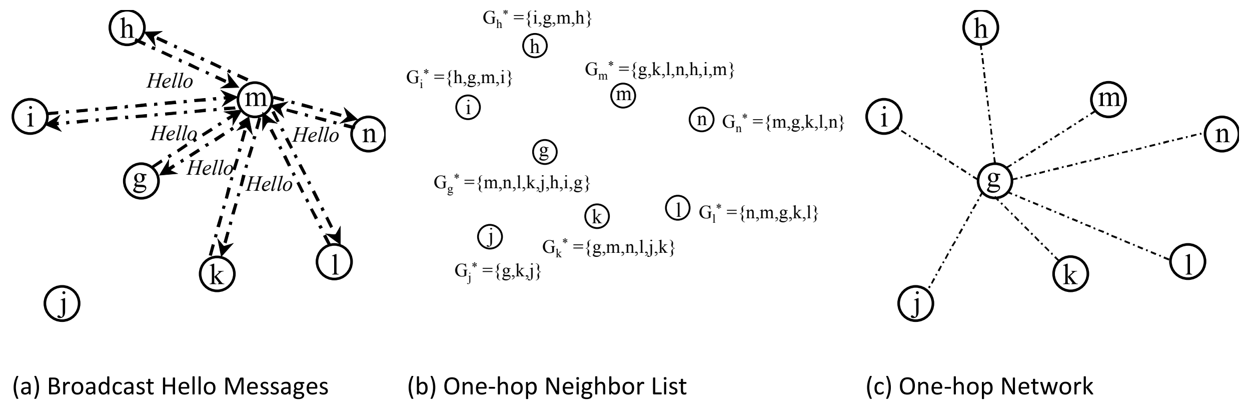
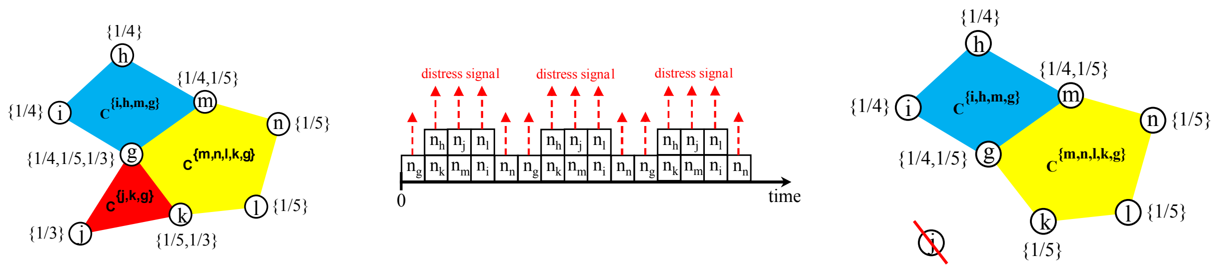


Fig. 1. A set of nodes broadcast Hello messages to build one-hop neighbor list G* and one-hop network.

B. RESCUEME: SMARTPHONE-BASED SELF RESCUE SYSTEM

First, according to the research of Pu, Lim, Chae and Jung (2017), each node broadcasts a one-time Hello message piggybacked with its node id, overhears Hello messages broadcast by other nodes, and then build the one-hop neighbor list G^*. For example, as shown in Fig. 1(a), n_m broadcasts a Hello message, and overhears the Hello messages broadcast by its adjacent nodes (e.g., n_h , n_i , n_g , n_k , n_l , and n_n). As a result, n_m can build its one-hop neighbor list, $G^*_m = \{h,i,g,k,l,n\}$, as shown in Fig. 1(b). By using the same technique, other nodes also can build their one-hop neighbor list. In this paper, each node also considers itself as a one-hop neighbor node and adds its id in the one-hop neighbor list. Thus, $G^*_m = \{h,i,g,k,l,n,m\}$. Each node then exchanges its one-hop neighbor list G^* with adjacent nodes, and identifies the center node who has the largest number of neighbor nodes and the G^* of all other nodes is a subset of the center node's one-hop neighbor list. Then the center node builds a one-hop network, where every other node can directly communicate with the center node, or vice versa. For example, as shown in Fig. 1(c), a one-hop network is built by the center node n_g.



(a) A Snapshot of Three Cliques

(b) Schedule of Sending Out Distress Signal

(c) Combine Cliques

Fig. 2. A snapshot of three cliques in one-hop network, schedule of sending out distress signal, and combine existing cliques.

Second, the center node examines the received G^* of all other nodes in the one-hop network, and groups the nodes that have the same subset of G^* into a clique, where a direct connection exits between every two nodes and each node can cover all other nodes in a clique. For example, in Fig. 1(b), n_g, n_i, n_h, and n_m contain the same subset of one-hop neighbor list, $G^*_{sub} = \{i,h,m,g\}$. Thus, the center node n_g groups n_i, n_h, and n_m along with itself into a clique, $C^{\{i,h,m,g\}}$, as shown in Fig. 2(a). In the one-hop network shown in Fig. 2(a), there are another two cliques, which are $C^{\{m,n,l,k,g\}}$ and $C^{\{j,k,g\}}$, respectively. In a clique, nodes are close to each other, thus, the area covered by one node can be a larger proportion of the area covered by all nodes in the clique. Here, Fig. 3 shows the change of coverage ratio between the coverage of one node and the total coverage of all nodes in a clique against the number of nodes in a clique.

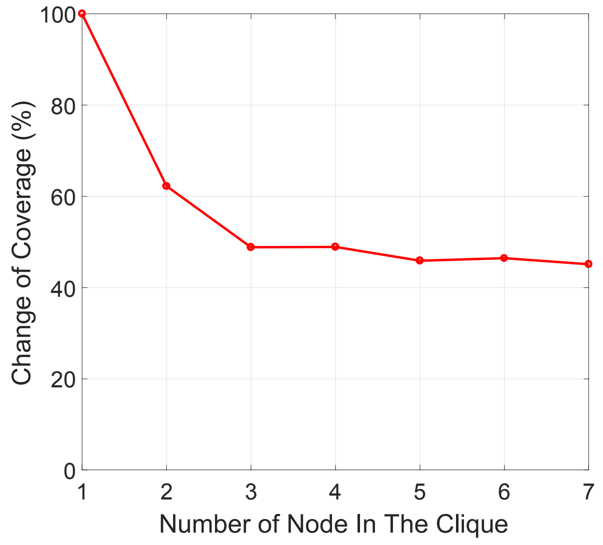


Fig. 3. The change of coverage ratio against the number of nodes in the clique.

Third, since the center node belongs to multiple cliques in the one-hop network, it will determine the schedule of broadcasting distress signals for all nodes across all the cliques. In order to reduce the energy consumption, the node that is located in the multiple cliques will choose the lowest fraction of the clique that it belongs to as its broadcasting frequency, denoted as ψ. For example, in Fig. 2(a), the center node n_g belongs to three cliques, $C^{\{i,h,m,g\}}$, $C^{\{m,n,l,k,g\}}$, and $C^{\{j,k,g\}}$, and the clique fraction of n_g is $\frac{1}{4}$ (since there are four nodes in $C^{\{i,h,m,g\}}$), $\frac{1}{5}$ and $\frac{1}{3}$ in the $C^{\{i,h,m,g\}}$,$C^{\{m,n,l,k,g\}}$, and $C^{\{j,k,g\}}$, respectively. Thus, the center node n_g chooses $\frac{1}{5}$ as its broadcasting frequency, $\psi_g = \frac{1}{5}$. In addition, the center node also calculates the sum of fractions of the cliques for each node, and determines the schedule of broadcasting distress signals accordingly. In this paper, the node with the larger sum of fractions of the cliques will broadcast distress signals earlier. For example, node n_m and n_h has the fractions of the cliques $\{ \frac{1}{4}, \frac{1}{5} \}$ and $\{ \frac{1}{4} \}$, respectively, the sum of fractions of the cliques is $\frac{9}{20}$ and $\frac{1}{4}$. Thus, n_m will broadcast a distress signal earlier than n_h. The center node determines the broadcasting schedule for all the nodes based on the following two criteria: (i) the broadcasting frequency for each node must be ψ; (ii) there is no more than one node broadcasting distress signals at any time in the clique. After determining the schedule, the center node broadcasts the Schedule packet, pkt^{sched}, piggybacked with the broadcast schedule to all other nodes in the one-hop network. When the node successfully receives the pkt^{sched} packet, it replies an Ack packet, pkt^{Ack}, back to the center node after a random backoff period [18]. Here, Fig. 2(b) shows the schedule of sending out distress signals of each node in the one-hop network of Fig. 2(a), where the schedule begins at time 0.

Fourth, since each node may carry a different amount of residual energy, and the node with less residual energy may turn off quickly because of frequently broadcasting a distress signal, thus, the schedule of broadcasting distress signals should be dynamically adjusted in response to the change of network topology. If the node is about to run out of the battery energy, it will notify the center node by sending a TurnOff packet, pkt^{off}. As a result, the center node removes the leaving node from existing cliques, and rebroadcasts the pkt^{off} packet to all other nodes in the one-hop network. After receiving the pkt^{off} packet, each node also removes the leaving node from its one-hop neighbor node list. Then, the center node examines the updated cliques, combines the cliques if one is the subset of another, determines a new broadcast schedule, and broadcasts the new schedule to all other nodes. For example, as shown in Fig. 2(c), node n_j leaves the network and the center node n_g combines $C^{\{j,k,g\}}$ and $C^{\{m,n,l,k,g\}}$ into $C^{\{m,n,l,k,g\}}$. If the center node has to leave the network, it will broadcast a Dismiss packet, pkt^{dis}, to all other nodes. After receiving the pkt^{dis} packet, all other nodes will restart RescueMe scheme from the beginning, and the same aforementioned operations will be applied.

IV. PERFORMANCE EVALUATION

We develop a customized simulation framework using Java to conduct our experiments. The number of nodes in the network is from 1 to 9, and the time interval of broadcasting distress signals is 5 seconds. In this paper, we measure the performance in terms of network lifetime and schedule vacancy by changing key simulation parameters, including the number of nodes, the number of broadcast distress signals in each time interval, and the number of cliques. For performance comparison, we compare the proposed scheme RescueMe with TeamPhone.

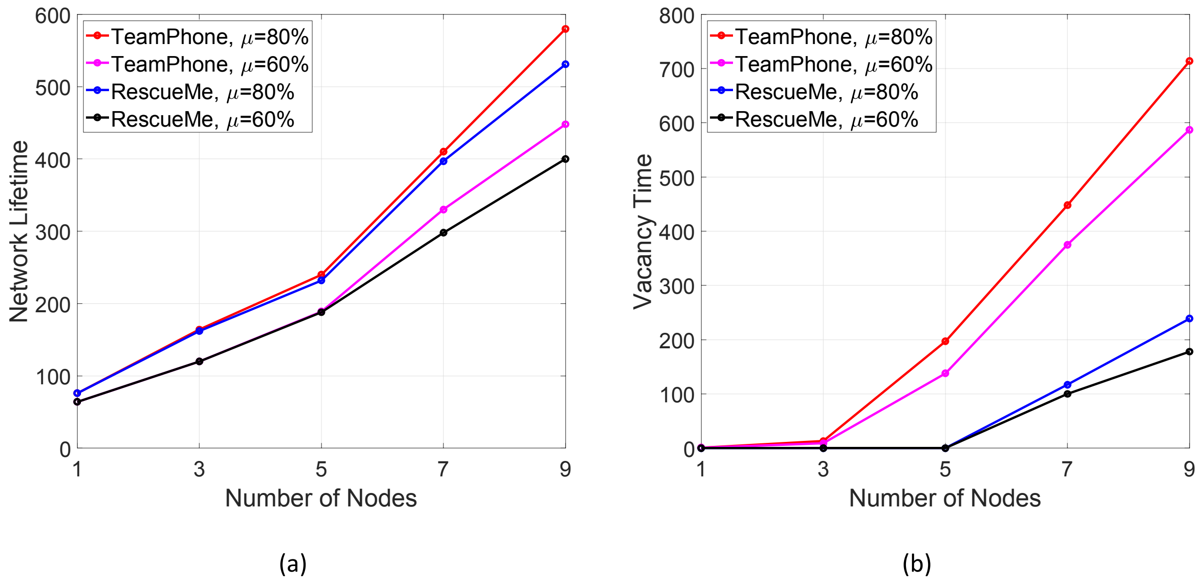


Fig. 4. The performance of network lifetime and schedule vacancy against the number of nodes.

First, we measure the network lifetime and schedule vacancy time by varying the number of nodes in Fig. 4. In Fig. 4(a), the overall network lifetime of RescueMe and TeamPhone increases as the number of nodes increases, which is because more nodes exist in the network, and the time interval of broadcasting distress signals for each node increases, resulting in a longer network lifetime. As the initial battery power of each node is increasing, the longer network lifetime will be observed, which is because the more power the node has, the longer time the node lasts, which results in a higher network lifetime. However, the RescueMe shows a slightly lower network lifetime than that of TeamPhone because TeamPhone uses the same schedule of broadcasting distress signals even if the node leaves the network. In Fig. 4(b), the overall schedule vacancy time of RescueMe and TeamPhone increases as the number of nodes increases, which is because the schedule of broadcasting distress signals of TeamPhone does not change when the nodes leave the network because of running out of battery. As a result, the schedule vacancy time increases significantly. The RescueMe shows a lower schedule vacancy time than that of TeamPhone. In the RescueMe, when the node is out of battery, the structure of the network will be changed and other nodes will fill in the empty interval of broadcasting distress signals, resulting in a lower schedule vacancy time.

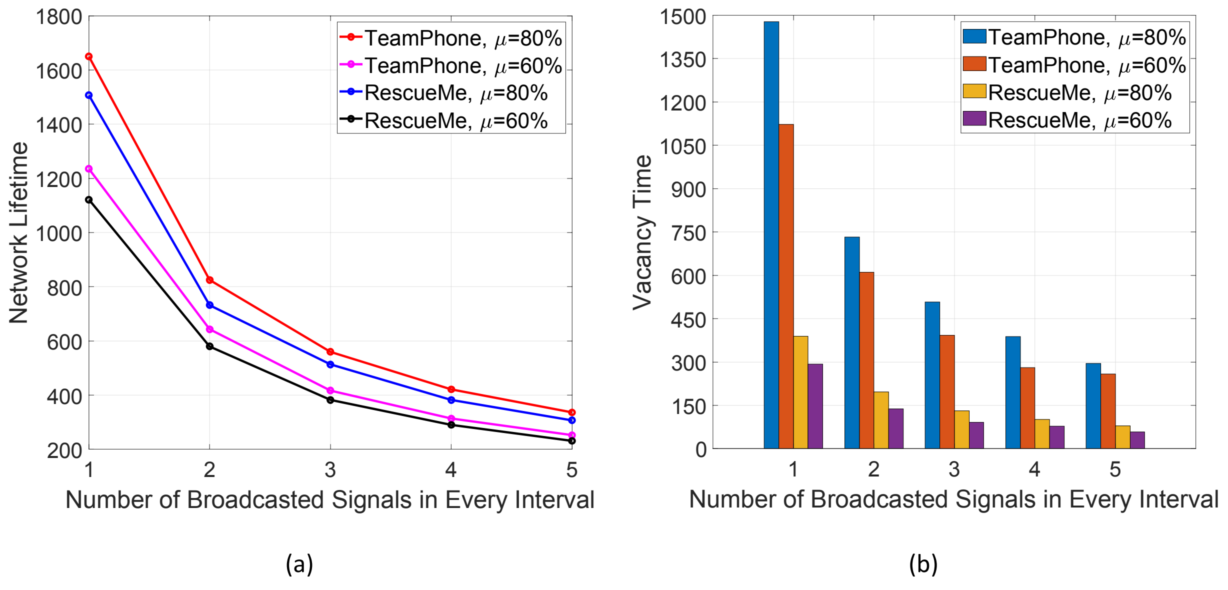


Fig. 5. The performance of network lifetime and schedule vacancy against the number of broadcast distress signals in each interval.

Second, we measure the network lifetime and schedule vacancy by varying the number of broadcast distress signals in each interval in Fig. 5. Overall, the network lifetime decreases as the number of broadcast distress signals increases in Fig. 5(a). With the number of broadcasted distress signals in each interval increases, the energy consumption in each interval increases. However, the RescueMe shows a slightly lower network lifetime than that of TeamPhone because the TeamPhone does not reschedule the interval of broadcasting distress signals whenever the node leaves the network. As shown in Fig. 5(b), the overall vacancy time of the RescueMe and TeamPhone decreases as the number of broadcast distress signals increases in every interval. The RescueMe shows a lower increment of schedule vacancy time than that of the TeamPhone, which is because if the node is out of power, the RescueMe reschedules the rest of the nodes in the network, which results in a lower vacancy time. Although the number of broadcast distress signals in every interval decreases and the network lifetime increases, the RescueMe still shows much smaller schedule vacancy time than that of TeamPhone.

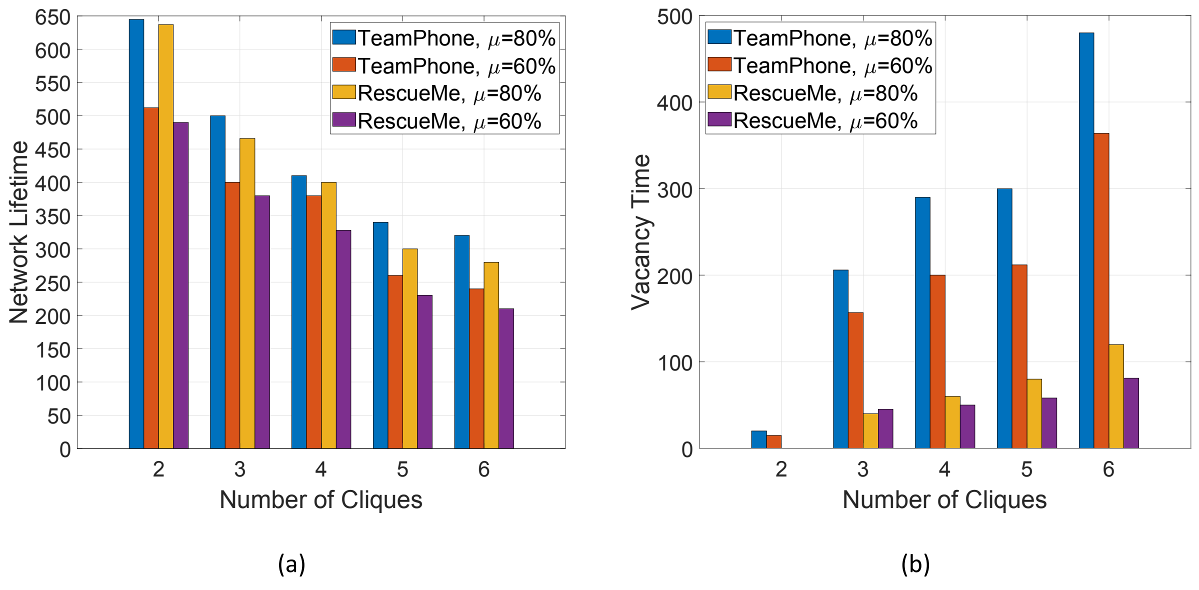


Fig. 6. The performance of network lifetime and schedule vacancy time against the number of cliques.

Third, Fig. 6 shows the network lifetime and schedule vacancy of the RescueMe and TeamPhone with varying number of cliques in the network. As shown in Fig. 6(a), the overall network lifetime decreases as the number of cliques increases, which is because when the number of cliques increases but the total number of nodes in the network is not changed, more nodes broadcast distress signals in every interval. As a result, the energy consumption of each interval increases, and the network lifetime decreases accordingly. The TeamPhone shows a higher network lifetime than that of RescueMe, which is because the TeamPhone does not reschedule the rest of nodes for broadcasting distress signals when certain nodes are out of power, which results in the higher schedule vacancy time. In Fig. 6(b), as the number of cliques increases, the overall schedule vacancy time increases. However, the RescueMe achieves a better performance than TeamPhone, which is because the RescueMe reschedules the network to fill in the empty time interval, which results in the lower vacancy time.

V. CONCLUSION

In this paper, we proposed a smartphone-based self rescue system to assist the operations of disaster rescue and relief. The basic idea of RescueMe is that a set of smartphones carried by survivors trapped or buried under the collapsed infrastructure forms into a one-hop network and sends out distress signals in an energy-efficient manner to nearby rescue crews to assist rescue operations. We evaluated the proposed approach through extensive simulation experiments and compared its performance with the existing scheme TeamPhone. The simulation results showed that the proposed approach can significantly reduce the schedule vacancy of broadcasting distress signals and improve the discovery probability with very little sacrifice of network lifetime, and indicate a potentially viable approach to expedite disaster rescue operations.

REFERENCES

Al-Sadi, A., & Awad, F. (2017, April). Smartphone-assisted location identification algorithm for search and rescue services. In *Information and Communication Systems (ICICS), 2017 8th International Conference* on (pp. 276-281). IEEE.

Ansari, A. (2016, September 04). Earthquake rattles Oklahoma, six other states. Retrieved from https://www.cnn.com/2016/09/03/us/oklahoma-earthquake/.

Bengtsson, L., Lu, X., Thorson, A., Garfield, R., & Von Schreeb, J. (2011). Improved response to disasters and outbreaks by tracking population movements with mobile phone network data: a post-earthquake geospatial study in Haiti. *PLoS medicine*, 8(8), e1001083.

Ho, Y. H., Chen, Y. R., & Chen, L. J. (2015, May). Krypto: assisting search and rescue operations using Wi-Fi signal with UAV. In Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use (pp. 3-8). ACM.

Hossain, A., Ray, S. K., & Sinha, R. (2016, December). A smartphone-assisted post-disaster victim localization method. In *High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference* on (pp. 1173-1179). IEEE.

Hossain, M. A., & Ray, S. K. (2018, April). A Smartphone-Assisted Device-to-Device Communication for Post-disaster Recovery. In *International Conference on Smart Grid Inspired Future Technologies* (pp. 11-20). Springer, Cham.

Karimi, F., & Ansari, A. (2016, April 18). Ecuador earthquake: Death toll jumps to 272. Retrieved from https://www.cnn.com/2016/04/17/americas/ecuador-deadly-earthquake.

Lu, Z., Cao, G., & La Porta, T. (2016, March). Networking smartphones for disaster recovery. In *Pervasive Computing and Communications (PerCom), 2016 IEEE International Conference* on (pp. 1-9). IEEE.

Luo, Y., Liu, J., Gao, Y., & Lu, Z. (2014, December). Smartphone-controlled robot snake for urban search and rescue. In *International Conference on Intelligent Robotics and Applications* (pp. 352-363). Springer, Cham.

Nishiyama, H., Ito, M., & Kato, N. (2014). Relay-by-smartphone: realizing multihop device-to-device communications. *IEEE Communications Magazine*, 52(4), 56-65.

Number of smartphone users in the U.S. 2010-2022. (n.d.). Retrieved from https://www.statista.com/statistics/201182/forecast-of-smartphone-users-in-the-us/

Pal, A., & Kant, K. (2018, January). E-Darwin2: A smartphone based disaster recovery network using WiFi tethering. In *Consumer Communications & Networking Conference (CCNC), 2018 15th IEEE Annual* (pp. 1-5). IEEE.

Pu, C., Gade, T., Lim, S., Min, M., & Wang, W. (2014, October). Lightweight forwarding protocols in *energy harvesting wireless sensor networks. In Military Communications Conference (MILCOM), 2014 IEEE* (pp. 1053-1059). IEEE.

Pu, C., Lim, S., Chae, J., & Jung, B. (2017). Active detection in mitigating routing misbehavior for MANETs. *Wireless Networks*, 1-15.

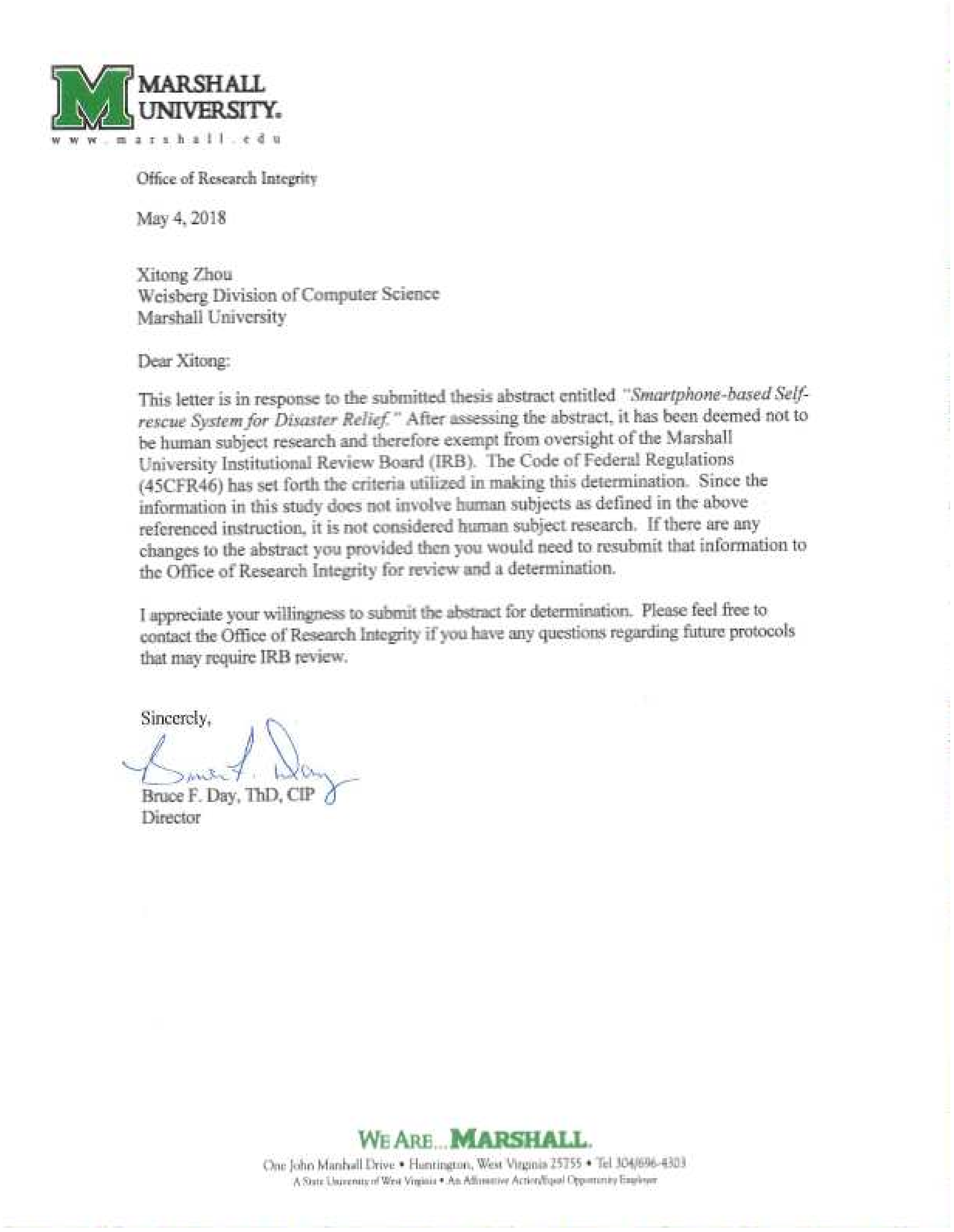
Pu, C., Lim, S., Jung, B., & Chae, J. (2018). EYES: Mitigating forwarding misbehavior in *energy harvesting motivated networks. Computer Communications*, 124, 17-30.

Raj, M., Kant, K., & Das, S. K. (2014, August). E-darwin: Energy aware disaster recovery network using wifi tethering. In *Computer Communication and Networks (ICCCN), 2014 23rd International Conference* on (pp. 1-8). IEEE.

Ray, S. K., Sinha, R., & Ray, S. K. (2015, June). A smartphone-based post-disaster management mechanism using WiFi tethering. In *Industrial Electronics and Applications (ICIEA), 2015 IEEE 10th Conference* on (pp. 966-971). IEEE.

Zamora, J. L. F., Suzuki, N., & Kashihara, S. (2017). SOS Message Distribution for Searching Disaster Victims. In *Smartphones from an Applied Research Perspective*. InTech.

APPENDIX A: APPROVAL LETTER



MARSHALL UNIVERSITY.

www.marshall.edu

Office of Research Integrity

May 4, 2018

Xitong Zhou
Weisberg Division of Computer Science
Marshall University

Dear Xitong:

This letter is in response to the submitted thesis abstract entitled *"Smartphone-based Self-rescue System for Disaster Relief."* After assessing the abstract, it has been deemed not to be human subject research and therefore exempt from oversight of the Marshall University Institutional Review Board (IRB). The Code of Federal Regulations (45CFR46) has set forth the criteria utilized in making this determination. Since the information in this study does not involve human subjects as defined in the above referenced instruction, it is not considered human subject research. If there are any changes to the abstract you provided then you would need to resubmit that information to the Office of Research Integrity for review and a determination.

I appreciate your willingness to submit the abstract for determination. Please feel free to contact the Office of Research Integrity if you have any questions regarding future protocols that may require IRB review.

Sincerely,

[illegible]

Bruce F. Day, ThD, CIP
Director

WE ARE... MARSHALL.

One John Marshall Drive • Huntington, West Virginia 25755 • Tel 304/696-4303
A State University of West Virginia • An Affirmative Action/Equal Opportunity Employer

APPENDIX B: SOURCE CODE

1) Test.java

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class test {
  public static void main(String[] args) {
    Random random = new Random();
    int[] power = new int[8];

    for(int i=0;i<8;i++) {
      power[i] = (int)Math.sqrt(5)*(int)random.nextGaussian()+60;
    }

    List<phone> clique1 = new ArrayList<phone>();
    clique1.add(new phone("g", power[0]));
    clique1.add(new phone("j", power[1]));
    List<phone> clique2 = new ArrayList<phone>();
    clique2.add(new phone("i", power[2]));
    clique2.add(new phone("g", power[0]));
    clique2.add(new phone("h", power[3]));
    List<phone> clique3 = new ArrayList<phone>();
    clique3.add(new phone("i", power[2]));
    clique3.add(new phone("j", power[1]));
    clique3.add(new phone("h", power[3]));
    clique3.add(new phone("m", power[4]));
    clique3.add(new phone("n",  power[5]));
    clique3.add(new phone("l",  power[6]));
    clique3.add(new phone("g",  power[0]));
    clique3.add(new phone("k",  power[7]));
    List<phone> clique4 = new ArrayList<phone>();
    clique4.add(new phone("n",  power[5]));
    clique4.add(new phone("l",  power[6]));
    List<phone> clique5 = new ArrayList<phone>();
    clique5.add(new phone("g", power[0]));
    clique5.add(new phone("h", power[3]));
    List<clique> cliques = new ArrayList<clique>();
    //cliques.add(new clique(clique1));
    //cliques.add(new clique(clique2));
    cliques.add(new clique(clique3));
    //cliques.add(new clique(clique4));

    network networks = new network(cliques);
    List<network> networks2 = new ArrayList<network>();
```

```
 networks2.add(networks);
 newSchduling firstSchduling = new newSchduling(networks2);
 List<Integer> timeList1 = firstSchduling.selfSaving();
 List<phone> clique6 = new ArrayList<phone>();
 clique6.add(new phone("g",  power[0]));
 clique6.add(new phone("j",  power[1]));
 List<phone> clique7 = new ArrayList<phone>();
 clique7.add(new phone("i",  power[2]));
 clique7.add(new phone("g",  power[0]));
 clique7.add(new phone("h", power[3]));
 List<phone> clique8 = new ArrayList<phone>();
 clique8.add(new phone("i", power[2]));
 clique8.add(new phone("j", power[1]));
 clique8.add(new phone("h", power[3]));
 clique8.add(new phone("m", power[4]));
 clique8.add(new phone("n",  power[5]));
 clique8.add(new phone("l",  power[6]));
 clique8.add(new phone("g",  power[0]));
 clique8.add(new phone("k",  power[7]));
 List<phone> clique9 = new ArrayList<phone>();
 clique9.add(new phone("n",  power[5]));
 clique9.add(new phone("l",  power[6]));
 List<phone> clique10 = new ArrayList<phone>();
 clique10.add(new phone("g", power[0]));
 clique10.add(new phone("h", power[3]));
 List<clique> cliqueTest = new ArrayList<clique>();
 //cliqueTest.add(new clique(clique6));
 //cliqueTest.add(new clique(clique7));
 cliqueTest.add(new clique(clique8));
//cliqueTest.add(new clique(clique9));
//cliqueTest.add(new clique(clique10));
schduling secondSchduling = new schduling(new network(cliqueTest));
List<Integer> timeList2 = secondSchduling.selfSaving();
if(timeList1.size()>0) {
   for(int i = 0;i<timeList1.size();i++) {
     System.out.print(timeList1.get(i)+"         ");
   }
   System.out.print(firstSchduling.getBlanktime());
   System.out.println();
 }

 if(timeList2.size()>0) {
   for(int i = 0;i<timeList2.size();i++) {
     System.out.print(timeList2.get(i)+"         ");
   }
   System.out.print(secondSchduling.getBlanktime());
```

```
      }
    }
  }
```

2) Scheduling.java

```
  import java.util.ArrayList;
  import java.util.List;

  public class schduling {
    private network netWork;
    private int time;
    private int blanktime;
    private List<Integer> timeList;

    public schduling(network netWork) {
      super();
      this.netWork = netWork;
      time = 0;
      blanktime = 0;
      timeList = new ArrayList<Integer>();
    }

    public network getNetWork() {
      return netWork;
    }

    public void setNetWork(network netWork) {
      this.netWork = netWork;
    }

    public List<Integer> selfSaving() {
      for(;;) {
        if(!SchduleFunction()) {
          timeList.add(time);
          System.out.println("total time is "+time);
          break;
        }
      }

      return timeList;
    }

    public boolean SchduleFunction() {
      boolean isSend = false;
      List<clique> networkClique = netWork.getClique();
      for( int i = 0; i < netWork.getBigNetworkNumber(); i++) {
```

```
  System.out.print(i+1+"    ");
  List<String> cliqueName = new ArrayList<String>();
  for(int j = 0;j<networkClique.size();j++) {
    int size = networkClique.get(j).getPhoneGroup().size();
    String name = networkClique.get(j).getPhone(size-1).getName();
    if(i>=networkClique.get(j).getPhoneGroup().size()) {
      if(i == netWork.getBigNetworkNumber()-1){
        if(!networkClique.get(j).reduceBattery(size-1)) {
          if(!cliqueName.contains(name)) {
            cliqueName.add(networkClique.get(j).getPhone(size-q);
            System.out.print(name+" send message, remind power"
                +networkClique.get(j).getPhone(size).getBetteryPercent()+"        ");

          }
          if(!isSend) {
            isSend = true;
          }
        } else {
          blanktime++;
        }
      } else {
        blanktime++;
      }
    } else {
      if(!networkClique.get(j).reduceBattery(i)) {
        if(!cliqueName.contains(networkClique.get(j).getPhoneName().get(i))){
          cliqueName.add(networkClique.get(j).getPhoneName().get(i));
          System.out.print(networkClique.get(j).getPhoneName().get(i)
          +" send message, remind power "
          +networkClique.get(j).getPhone(i).getBetteryPercent()+"        ");
        }
        if(!isSend) {
          isSend = true;
        }
      }
    }
  }
  System.out.println();
  time = time+1;
}
if(isSend) {
  System.out.println("--------------------------------------------------------------------");
  return true;
} else {
  return false;
}
```

```
    }

    public int getBlanktime() {
        return blanktime;
    }

    public void setBlanktime(int blanktime) {
        this.blanktime = blanktime;
    }
}
```

3) newSchduling.java

```
import java.util.ArrayList;
import java.util.List;

public class newSchduling {
    private List<network> compareNetwork;
    private int time;
    private int blanktime;
    private List<Integer> timeList;

    /*
     *
     * constructing function
     *
     */

    public newSchduling(List<network> netWork) {
        super();
        this.compareNetwork = new ArrayList<network>();
        compareNetwork = netWork;
        time = 0;
        blanktime = 0;
        timeList = new ArrayList<Integer>();
    }

    /*
     *
     * it is new way to schedule the node, which difference with old schedule is that when
     * one node runs out of power, it will reschedule.
     *
     */
```

```
public List<Integer> selfSaving() {
  for(;;) {
    if(!compareSchduleFunction()) {
      timeList.add(time);
      System.out.println("total time is "+time);
      break;
    }
  }
  return timeList;

}

/*
 *
 * schedule function
 *
 */

public boolean compareSchduleFunction() {
  boolean isbreak = false;
  for(int t = 0;t<compareNetwork.size();t++) {
    List<clique> networkClique = compareNetwork.get(t).getClique();
    int bigNetworkNumber = compareNetwork.get(t).getBigNetworkNumber();
    // this for loop will print the node which are calling rescue team by order
    for( int i = 0; i < bigNetwork; i++) {
      System.out.print(i+1+"    ");
      List<String> outputName = new ArrayList<String>();
      for(int j = 0;j < networkClique.size();j++) {
        int index = 0;
        int size = networkClique.get(j).getPhoneGroup().size();
        String name = null;

        /*
         *
         * when index is bigger than clique size and index is the last one for this
         * network, index will become the last one node of this clique
         *
         */

        if(i >= size && i == bigNetworkNumber - 1) {

          index = networkClique.get(j).getPhoneGroup().size()-1;
          name = networkClique.get(j).getPhone(index).getName();

        } else if (i < size){
          index = i;
  name = networkClique.get(j).getPhone(i).getName();
} else if(i >= size && i != bigNetworkNumber-1) {
  blanktime++;
}

/*
 *
 * if name is null, the index is bigger than clique size and index is not the last
 * one for this network, there are no node calling the rescue team in this clique
 *
 */

if(name!=null) {

/*
 *
 * outputName means that in this loop, this node never calling, if node have
 * called, this node will not consume power
 *
 */

  if(!outputName.contains(name)) {
    outputName.add(name);

    /*
     *
     * if this node is not out of power, it will print send message and remind
     * power
     *
     */

    if(!networkClique.get(j).reduceBattery(index)) {
      System.out.print(name + " send message, remind power "+
        networkClique.get(j).getPhone(index).getBetteryPercent()+"              ");
      List<Integer> groupIndex =
        networkClique.get(j).getPhone(index).getGroupLocation();

    /*
     *
     * in this for loop, this node in other clique will also consume the power
     *
     */

      for(int m = 0; m < groupIndex.size();m++) {
        List<String> phoneNs=
             networkClique.get(groupIndex.get(m)).getPhoneName();
```

```
                    for(int n = 0;n< phoneNs.size();n++){
                      if(phoneNs.get(n) == name && groupIndex.get(m)!=j) {

                        networkClique.get(groupIndex.get(m)).reduceBattery(n);
                      }
                    }
                  }
                } else {

                /*
                 *
                 * when node run out of power, the system will reschedule and enter into
                 * reschedule function
                 *
                 */

                  System.out.println(name+" run out of power");
                  timeList.add(time);
                  System.out.println("time = " + time);
                  System.out.println("rescheduling..............................");
                  reschdule(name);
                  isbreak =true;
                  break;
                }
              }
            }
          }
          time++;
          System.out.println();
          if (isbreak) {
            break;
          }
        }
        if(isbreak) {
          break;
        }
      }
      if(compareNetwork.size() == 0) {
        return false;
      } else {
        System.out.println(".......................................................................");
        return true;
      }
    }
    public int getBlanktime() {
        return blanktime;
    }

   public void setBlanktime(int blanktime) {
        this.blanktime = blanktime;
    }

    /*
     *
     * when a node runs out of power, the system will reschedule all the node.
     *
     */

    private void reschdule(String name) {
        List<network> newNetWork =  new ArrayList<network>();
        for(int i = 0;i<compareNetwork.size();i++) {
            newNetWork = compareNetwork.get(i).reSchdule(name);
        }
        compareNetwork = newNetWork;

    }
}
```

4) network.java

```
import java.util.ArrayList;
import java.util.List;


public class network {
    private List<clique> clique;
    private int bigNetworkNumber;

    public List<clique> getClique() {
        return clique;
    }

    public void setClique(List<clique> clique) {
        this.clique = clique;
    }

    public int getBigNetworkNumber() {
        return bigNetworkNumber;
    }
```

```
public void setBigNetworkNumber(int bigNetworkNumber) {
  this.bigNetworkNumber = bigNetworkNumber;
}

public network(List<clique> clique) {
  this.clique = new ArrayList<clique>();
  this.clique = clique;
  int bigNetworkNumber = 0;
  for(int i=0;i<clique.size();i++) {
    if(clique.get(i).getPhoneGroup().size()>bigNetworkNumber) {
      bigNetworkNumber = clique.get(i).getPhoneGroup().size();
    }
  }
  this.bigNetworkNumber = bigNetworkNumber;
  value();

  conectGroup();

  for (int i = 0; i < clique.size(); i++) {
    clique.get(i).sort(0, clique.get(i).getPhoneGroup().size()-1);
  }
}

/*
 *
 * this function will build connection between cliques if the node in one clique is in
 * other clique
 *
 */

private void conectGroup() {
  // TODO Auto-generated method stub
  for(int i=0;i<clique.size();i++) {
    List<phone> phoneGroup = new ArrayList<phone>();
    phoneGroup = clique.get(i).getPhoneGroup();
    List<Integer> connectGroup = new ArrayList<Integer>();
    for(int j=0;j<phoneGroup.size();j++) {
      List<Integer> groupLocation = new ArrayList<Integer>();
      groupLocation = phoneGroup.get(j).getGroupLocation();
      for(int t = 0; t<groupLocation.size();t++){
        if(!connectGroup.contains(groupLocation.get(t))&&groupLocation.get(t)!=i) {
          connectGroup.add(groupLocation.get(t));
        }
      }
    }
    clique.get(i).setConnectGroup(connectGroup);
```

```
   }
}

/*
 *
 * return value of the specific node
 *
 */

public float Value(String name) {
   float value = 0;
   List<Integer> groupLocation = new ArrayList<Integer>();
   for(int i=0;i < clique.size();i++) {
     if(clique.get(i).getPhoneName().contains(name)) {
       value+=(float)1/clique.get(i).getPhoneGroup().size();
        groupLocation.add(i);

     }
   }

   for(int i = 0;i<clique.size();i++) {
      for(int j = 0;j<clique.get(i).getPhoneName().size();j++) {
       if(name == clique.get(i).getPhoneName().get(j)) {
         clique.get(i).setGroupLocation(j, groupLocation);
       }
      }
   }

   return value;
}

/*
 *
 * reschedule function
 *
 */

public List<network> reSchdule(String deletName) {
   for(int i = 0; i<clique.size(); i++) {
     clique.get(i).setVisited(false);
  }
   List<network> newNetwork = new ArrayList<network>();
   List<Integer> changeIndex = new ArrayList<Integer>();

/*
 *
```

```
* in this for loop, the system will find the deleting node in the clique
*
*/

 for(int i = 0; i<clique.size(); i++)  {
   if(clique.get(i).getPhoneName().contains(deletName)) {
     boolean isRepeat = true;
     List<phone> cliquePhone = clique.get(i).getPhoneGroup();
     List<Integer> connetGroup = new ArrayList<Integer>();
     List<Integer> removeGroup = new ArrayList<Integer>();
     bigNetworkNumber = 0;
     for(int j = 0;j<cliquePhone.size();j++) {

     /*
      *
      * find it, remove it
      *
      */

       if(cliquePhone.get(j).getName() == deletName) {
         removeGroup = cliquePhone.get(j).getGroupLocation();
         clique.get(i).removep(j);

       } else {

     /*
      *
      * put connection into new arraylist
      *
      */

         for(int t = 0; t<cliquePhone.get(j).getGroupLocation().size();t++) {
           if(!connetGroup.contains(cliquePhone.get(j).getGroupLocation().get(t))) {
             connetGroup.add(cliquePhone.get(j).getGroupLocation().get(t));
           }
         }
       }
     if (bigNetworkNumber<clique.get(i).getPhoneGroup().size()) {
       bigNetworkNumber= clique.get(i).getPhoneGroup().size();

     }
   }

   if(connetGroup.size() == 1 && connetGroup.get(0) != i) {
     changeIndex.add(i);
   } else if(connetGroup.size() == 0){
        changeIndex.add(i);
      } else if(connetGroup.size() == 1 && connetGroup.get(0) == i){

      } else {
        for(int j = 0; j< connetGroup.size(); j++) {
        //this for loop will judge wheather this clique is belong to other clique
          for(int t = 0;t< clique.get(i).getPhoneGroup().size();t++) {
            List<Integer> groupLo =
              clique.get(i).getPhoneGroup().get(t).getGroupLocation();
            if((connetGroup.get(j) != i && !groupLo.contains(connetGroup.get(j)))) {
              isRepeat = false;
              break;
            }
          }
          if(isRepeat) {
            changeIndex.add(i);
            break;
          }
        }
      }

      /*
       *
       * remove the possible connection with other cliques
       *
       */

      for(int j = 0;j<removeGroup.size();j++) {
        if(connetGroup.contains(removeGroup.get(j))) {
          removeGroup.remove(j);
        }
      }

      /*
       *
       * remove the possible connection with other cliques
       *
       */

      for(int j = 0; j < removeGroup.size();j++) {
        clique.get(i).removeConnectGroup(removeGroup.get(j));
      }
    }
  }

/*
```

```
    *
    * remove the connection which is useless
    *
    */

   for(int i = 0;i< changeIndex.size();i++) {
     for(int j = 0;j<clique.size();j++) {
       if(clique.get(j).getConnectGroup().contains(changeIndex.get(i))) {
         clique.get(j).removeConnectGroup(changeIndex.get(i));
        }
     }
   }
   newNetwork = splitNetwork(changeIndex);
   return newNetwork;

 }

 /*
  *
  * this function will divide the network if one clique or some cliques have no connection
  * with others
  *
  */

 public List<network> splitNetwork(List<Integer> changeIndex) {
   List<network> newNetwork =new ArrayList<network>();
   for(int i = 0; i<clique.size();i++) {
     List<Integer> connetIndex = new ArrayList<Integer>();
     //add the useful clique into the new network
     if((!clique.get(i).isVisited())&& (!changeIndex.contains(i))) {
       connetIndex = buildconnect(i);
       List<clique> cliqueArray = new ArrayList<clique>();
       for(int j = 0;j<connetIndex.size();j++) {
         cliqueArray.add(clique.get(connetIndex.get(j)));
       }
       newNetwork.add(new network(cliqueArray));
     }
   }
   return newNetwork;
 }

 /*
  *
  * this function will return the connection with this clique and build new arraylist
  *
  */
```

```
private List<Integer> buildconnect(int j) {
  List<Integer> connectIndex = new ArrayList<Integer>();
    connectIndex.add(j);
    clique.get(j).setVisited(true);
    for(int i = 0;i< clique.get(j).getConnectGroup().size();i++) {
      if(!clique.get(clique.get(j).getConnectGroup().get(i)).isVisited()) {
        connectIndex.addAll(buildconnect(clique.get(j).getConnectGroup().get(i)));
      }
    }
    return connectIndex;
  }

  /*
   *
   * return the cliques which include the specific node
   *
   */

  public List<Integer> cliqueLocation(String name) {
    List<Integer> locationClique = new ArrayList<Integer>();
    for(int i = 0;i<clique.size();i++) {
      List<phone> phoneGroup = clique.get(i).getPhoneGroup();
      for(int j=0;j<phoneGroup.size();j++) {
        if(phoneGroup.get(j).getName()==name) {
          locationClique.add(i);
          break;
        }
      }
    }
    return locationClique;
  }

  /*
   *
   * calculate the value of every node according to the network
   *
   */

  public void value() {
    for(int i=0; i<clique.size();i++) {
      List<phone> changePhoneValue = clique.get(i).getPhoneGroup();
      float value = 0;
      for(int j = 0;j < changePhoneValue.size(); j++) {
        value = Value(changePhoneValue.get(j).getName());
        changePhoneValue.get(j).setValue(value);
```

```
            }
            clique.get(i).setPhoneGroup(changePhoneValue);
         }
      }
   }
```

5) clique.java

```
import java.util.ArrayList;
import java.util.List;


public class clique {
  private List<phone> phoneGroup;
  private List<Integer> connectGroup;
  private List<String> phoneName;
  private boolean visited;

  public List<Integer> getConnectGroup() {
    return connectGroup;
  }

  public void setConnectGroup(List<Integer> connectGroup) {
    this.connectGroup = connectGroup;
  }

  public List<String> getPhoneName() {
    return phoneName;
  }

  public void setPhoneName(List<String> phoneName) {
    this.phoneName = phoneName;
  }

  public List<phone> getPhoneGroup() {
    return phoneGroup;
  }

  public boolean reduceBattery(int index) {
    boolean isRunOut = false;
    int bettery = phoneGroup.get(index).getBetteryPercent();
    if(bettery >1) {
      phoneGroup.get(index).setBetteryPercent(bettery-1);
    } else {
      isRunOut = true;
  }
  return isRunOut;
}

public void setPhoneGroup(List<phone> phoneGroup) {
  this.phoneGroup = phoneGroup;
}
public void setPhonePercent(int x) {
  phoneGroup.get(x).setBetteryPercent(phoneGroup.get(x).getBetteryPercent()-1);
}

public clique(List<phone> phoneClique) {
  super();
  this.phoneGroup = new ArrayList<phone>();
  this.phoneName = new ArrayList<String>();
  for(int i=0;i<phoneClique.size();i++) {
    this.phoneGroup.add(phoneClique.get(i));
    phoneName.add(phoneGroup.get(i).getName());
  }
  visited = false;

}

public boolean isVisited() {
  return visited;
}

public void setVisited(boolean visited) {
  this.visited = visited;
}

public phone getPhone(int x) {
  return phoneGroup.get(x);
}

public void removep(int i) {
  boolean isAppear = false;

 /*
  *
  * in this for loop, the system will remove the connection which removing node have
  * and other node in the clique do not have
  *
  */

  for(int j = 0; j<phoneGroup.get(i).getGroupLocation().size();j++) {
      int temp = phoneGroup.get(i).getGroupLocation().get(j);
      isAppear = false;
      for(int t = 0;t< phoneGroup.size();t++) {
        List<Integer> getLo = phoneGroup.get(t).getGroupLocation();
        if(t!=i) {
          if(getLo.contains(temp)) {
            isAppear = true;
          }
        }
      }
      if(!isAppear) {
        for(int t = 0;t<connectGroup.size();t++) {
          if(connectGroup.get(t) == temp) {
            connectGroup.remove(t);
            break;
          }
        }
      }
    }
    phoneGroup.remove(i);
    phoneName.remove(i);
  }

  public String phoneName(int i) {
  // TODO Auto-generated method stub
    return phoneGroup.get(i).getName();
  }

  /*
   *
   * this function will sort the nodes in the clique by the value
   *
   */

  public void sort(int left, int right) {
    int leftIndex = left;
    int rightIndex = right;

    /*
     *
     * if only one element in this part, it is not necessary to divide
     *
     */

    if(leftIndex <= rightIndex) {
      phone key = phoneGroup.get(leftIndex);
```

```
/*
 *
 * in this loop, we will divide array into two parts
 * the left one is smaller than key number and the right one is bigger than key
 * number
 *
 */

 while(leftIndex != rightIndex) {

/*
 *
 * in this loop and judgment, they will put the first smaller than key number into the
 * empty left index
 *
 */

   while(phoneGroup.get(rightIndex).getValue() <= key.getValue() &&
     rightIndex > leftIndex) {
     rightIndex--;
   }
   if(phoneGroup.get(rightIndex).getValue() > key.getValue()) {
     phoneGroup.set(leftIndex, phoneGroup.get(rightIndex));
     phoneName.set(leftIndex, phoneName.get(rightIndex));
   }

   /*
    *
    * in this loop and judgment, they will put the first bigger than key number into
    * the empty right index
    *
    */

   while(phoneGroup.get(leftIndex).getValue() >= key.getValue() &&
     leftIndex<rightIndex) {
     leftIndex++;
   }
   if(phoneGroup.get(leftIndex).getValue() < key.getValue()) {
     phoneGroup.set(rightIndex, phoneGroup.get(leftIndex));
     phoneName.set(rightIndex, phoneName.get(leftIndex));
   }

}
phoneGroup.set(leftIndex, key);
phoneName.set(leftIndex, key.getName());
```

```
        /*
         *
         * use quick sort function sort left array and right array recursively
         *
         */

         sort(left, rightIndex-1);
         sort( leftIndex+1, right);
      }
    }

    public void setGroupLocation(int i, List<Integer> groupLocation) {
    // TODO Auto-generated method stub
      phoneGroup.get(i).setGroupLocation(groupLocation);
    }

    /*
     *
     * remove the connection of clique
     *
     */

    public void removeConnectGroup(Integer integer) {
      for(int i=0;i<connectGroup.size();i++) {
        if(connectGroup.get(i) == integer) {
          connectGroup.remove(i);
          break;
        }
      }

    }
  }
```

6) phone.java

```
  import java.util.ArrayList;
  import java.util.List;

  public class phone {
    private float value;
    private String name;
    private List<Integer> groupLocation;
    private double betteryPercent;

    public phone(String name, int betteryPercent) {
      super();
    this.name = name;
    this.betteryPercent = betteryPercent;
    groupLocation = new ArrayList<Integer>();
  }



  public double getBetteryPercent() {
    return betteryPercent;
  }

  public void setBetteryPercent(double betteryPercent) {
    this.betteryPercent = betteryPercent;
  }

  public List<Integer> getGroupLocation() {
    return groupLocation;
  }

  public void setGroupLocation(List<Integer> groupLocation) {
    this.groupLocation = groupLocation;
  }

  public void setValue(float value) {
    this.value = value;
  }

  public float getValue() {
    return value;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }
}
```