Dynamic Cost-effective Emergency Network Provision

Roop K. Sriramulu
Computer Engineering Department
San Jose State University
roopkumar.sriramulu@sjsu.edu

Younghee Park*
Computer Engineering Department
San Jose State University
younghee.park@sjsu.edu

Sang-Yoon Chang
Computer Science Department
University of Colorado at Spring
schang2@uccs.edu

Kaikai Liu
Computer Engineering Department
San Jose State University
kaikai.liu@sjsu.edu

ABSTRACT

The importance to our society of emergency network communications cannot be underestimated. The loss of communication and network systems in a state of emergency denies victims of disaster and city emergency response teams critical information about the crisis. It is essential to restore communication systems and service in order to ensure continuous and efficient emergency operations. This paper presents a cloud-based cost-effective emergency network management system to provide dynamic network services. We have developed a network application to enable users to access the Internet during an emergency. The application is ready to immediately restore lost connectivity through economical embedded systems and UAVs. We implement our prototypes based on Resin.io, a framework that utilizes Linux containers on IoT(Internet of Things) devices to deploy applications. We evaluate our systems on Raspberry Pi, a popular embedded system. We demonstrate the feasibility of our proposed system, showing that it is an economical infrastructure that it can successfully be used for an emergency network to replace a demolished network infrastructure.

KEYWORDS

Emergency Network Communications, Resin.io, Raspberry Pi, IEEE 802.11 WiFi, Drones

ACM Reference format:
Roop K. Sriramulu, Younghee Park, Sang-Yoon Chang, and Kaikai Liu. 2017. Dynamic Cost-effective Emergency Network Provision. In *Proceedings of I-TENDER'17:First CoNEXT Workshop on ICT Tools for Emergency Networks and DisastEr Relief , Republic of Korea, December 12,2017 (I-TENDER'17),* 7 pages.
DOI: 10.1145/3152896.3152904

*Dr.Younghee Park is a corresponding author. The work is supported by NSF CNS Award #1637371.

I-TENDER'17, Republic of Korea

DOI: 10.1145/3152896.3152904

1 INTRODUCTION

First responders to disasters need a complete picture of the communitys status in order to accurately assess the condition of the inhabitants and organize available resources to save lives, protect the environment, and prevent further damage in the community. In local emergency situations, this information is collected through calls to public safety answering point (9-1-1), city employee radio communications (fire, law, public works, transportation, building inspectors), and monitoring by the media and social media. However, in major disasters, these normal community services will be interrupted, including cell phone service, Internet connectivity, and utilities. In such circumstances, novel systems must be available to substitute for the lost connectivity, to allow residents to contact the public safety answering point, and to allow the Emergency Operations Center to collect and aggregate critical information across sectors to ensure that lifesaving operations are conducted expeditiously.

Satellites provide a reliable communication infrastructure, making them extremely important in crises. Communication providers like Inmarsat [5] and Delorme [25] offer global communication services through satellites. However, satellites are not only very expensive to maintain, but also have inherent limitations in terms of signal blockage. Currently the cellular network infrastructure is the most widely used method of voice and data communication. Cellular communications offer the advantages of mobility and flexibility, but cellular infrastructure is prone to be damaged during crisis situations like earthquakes, hurricanes, floods, or wildfires. This means that people may be injured or die if they are unable to contact emergency services in time. For instance, thousands of cellular towers went out of service for days during hurricane Katrina [2]. During such circumstances, cellular companies like AT&T have a disaster recovery team standing by [4]. Once the network goes down, a mobile cellular tower is driven to the emergency zone and a complementary network is accessed to provide people connectivity until the main communication backbone is restored. Use of smartphones using these backup connections during outages is hampered, however, because the towers are limited in the number of calls that can be routed. Hence, we need to have a distributed network infrastructure that will satisfy all the needs that arise during an emergency, providing an emergency network that is easy to manage and deploy.

This paper proposes a cloud-based emergency network management system by using embedded systems to prepare for emergencies. First, we build up an infrastructure to bring up network capable devices. We utilizes the Resin.io Cloud service to automatically and

easily deploy our network application regardless of the heterogeneous hardware specifications in embedded systems. Second, we design a management system to maintain and monitor our infrastructure for an emergency. The proposed system includes a device manager, a connectivity manager, and a location manager. The device manager deploys economical embedded systems that are prepared in advance of an emergency and installed meeting all requirements to function in a crisis. The connectivity manager keeps checking to verify whether the pre-deployed systems are able to connect users to the Internet. Lastly, the location manager tracks the devices based on real-time GPS information and monitors the current status of network infrastructure to share the information in real time. Our proposed system enables users to send rescue or safety messages through pre-configured hotspot devices when facing a natural disaster or other emergency that takes out communication services. We implemented our prototypes for the proposed systems and evaluated it by using the embedded systems, showing that we have designed a cost-effective network infrastructure by using economical embedded systems. The infrastructure is based on virtualization and the Resin OS for embedded systems and functions independently of different hardware specifications.

Our first contribution is that we proposed a framework to combine inexpensive embedded systems and UAVs into building up a network infrastructure for an emergency. Second, we developed a hotspot application and deployed it on any embedded systems by using the Resin OS and the containerization technique. Third, we evaluated our proposed framework with the real embedded systems and showed the feasibility of the proposed system through experiments.

The rest of the paper is organized as follows. Section 2 presents our proposed system and shows evaluation results in Section 3. We overview the previous work in Section 5 and discuss our future work with a conclusion in Section 6 and Section 7.

2 THE PROPOSED WORK

The tree architecture for the wireless Internet has resulted in a single point of failure in crises like the devastation that occurred in Red Hook during Hurricane Sandy in 2011 [3, 12]. Satellite communications are very expensive for normal users and experience limitations in terms of signal blockage. This paper presents a dynamic network service for emergency networks, deploying hotspot applications into low-priced embedded devices to provide Internet access in emergency zones.

2.1 System Architecture

This section presents our system architecture for monitoring and managing devices, connections, and locations, as shown in Figure 1. It consists of the device management, connectivity management, and location management needed to create emergency networks. The device management module determines the location for deploying embedded systems and installs hotpot software on the systems to create networks. The connectivity management module keeps polling network connectivity between two devices. The location management module records the exact location of current devices including embedded systems and unmanned aerial vehicle (UAVs).

2.1.1 Device Management. The device manager identifies where the deployment of embedded systems is needed. To locate each device, all the procedures must be able to be initiated automatically and updated easily from a remote, centralized point. We divide the emergency network into zones. We assume that each zone has pre-deployed embedded devices sufficient to serve as hotspots or relay nodes for data communications in emergencies.

Embedded systems are inherently heterogeneous with various types of devices, making it is a challenge to remotely install all necessary programs on all of the devices; however, the Rein.io Cloud service enables a nearly seamless remote and automatic management and updating of software on such heterogeneous device platforms. We designed a hotspot application to successfully build emergency networks using the platform-independent Resin.io software. In other words, to make ready-to-use embedded devices, a base OS image from the Resin.io service is installed on each device, after which a Docker container is created on top of the base image in order to install our network application. The device manager readies any device in two steps, as described below in detail. We target any embedded devices and unmanned aerial vehicles (UAVs).

(1) The first step is to install the Resin.io base image from the Resin.io Cloud Service: The Resin.io is a framework that makes it easy to deploy, maintain, and update the application or code on various types of devices [22]. It gives us a platform to employ Docker containerization technology on embedded devices. Using the Resin.io, users install a customized Yocto OS on their devices. This base OS with the Docker container enables us to utilize any software on any device. The Docker container is pre-installed and configured to function within the low memory constraints of the embedded devices. The Resin.io base OS image provides a platform for deploying any application in the form of Docker images [13, 22].

(2) The second step is to install Docker:To give a uniform running environment for an application, we use the Docker tool to package the application with all its dependencies. A Dockerfile [13] is written to first install and configure the runtime environment of the application. A docker image is built from the Dockerfile in the administrator local repository and is uploaded to a remote repository. The device manager downloads the docker image from the remote repository and runs it in his/her local device to get the application running inside a docker container. This container has its own dependencies but shares the kernel with the host operating system [13].

2.1.2 Connectivity Management. The connectivity manager instantaneously creates a backup network that can be used in emergencies. It mainly focuses on monitoring and controlling a network created by the pre-deployed devices. A graph G consists of vertices and edges. The vertices represent each device, and the edge represents the connection between two devices. A network can be expressed as $G = \{V, E\}$ where $|V|$ is the set of devices and E is the set of available connections. We define a connectivity in that two devices, v_i and e_j (i or j is a device.), are able to connect over a connection e_{ij} through a network utility like Ping. To ready pre-deployed devices for use in an emergency, we need to install our hopspot application on the devices in advance of the emergency.

When the device starts up after configuring the Resin OS, it connects to the Rein.io Cloud server [22] to fetch the hotspot application. The hotspot is a gateway application to allow users to access the Internet in an emergency situation. We set up this application

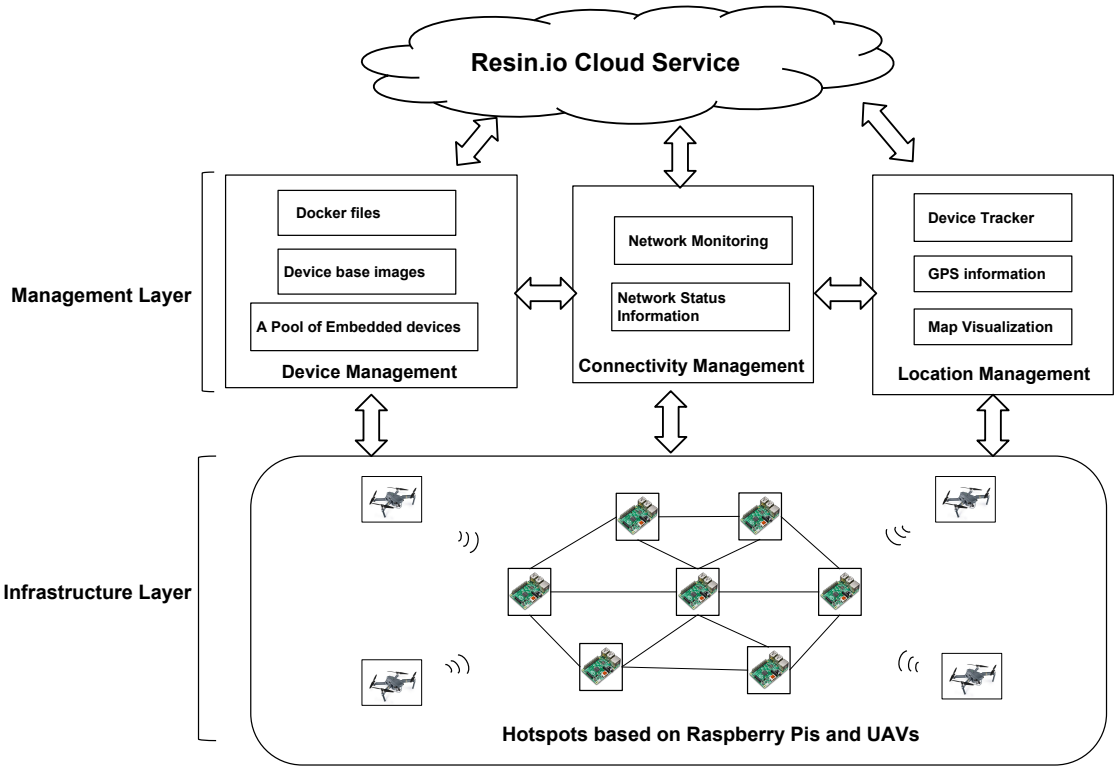

Figure 1: A system level view of the architecture

to be available to any devices through the Git repository [11] on the Resin.io Cloud server. Therefore, if the application contains a Dockerfile, the hotspot application can be automatically installed on the device by using the command specified in our implemented Dockerfile [13].

The connectivity manager first creates a hotspot for a communication on the device to make a connection between two devices. One device is connected to the Internet and the other is connected to another device to create a small network. A small network G can be created in a zone to let users access the Internet for data transfer in an emergency. The connectivity manager keeps communicating with the Cloud server to get information about device availability, GPS information, and network information. In an emergency, the connectivity manager keeps checking the connectivity for hop-by-hop communications among devices through Ping. If it detects a link failure from the back-up network topology, a drone is sent to make up the link failure so that users maintain access to the Internet. The drones are managed by a drone monitor that we developed in our previous work [19], a cloud-based UAV real-time monitoring and management system. While monitoring the connection, the link failure will be covered by the drones with the hotspot application on the drones. The drone enables users to communicate with other users or the Internet in an emergency.

2.1.3 Location Management. The location management module keeps updating through a device tracker that we previously developed [19]. The device tracker uses UAVs to send GPS information about the pre-deployed embedded systems to keep tracking their exact locations. Using Mapbox [1], an interactive map visualizes active device paths and current devices or UAV locations, allowing us to track exact locations in an emergency zone. With the help of the drones, it is easy to monitor any emergency zone. Through the GPS information provided by the UAVs, we can check the connectivity of any pre-deployed emergency devices. If some devices are not working, the drones can replace the out-of-order devices. To compensate for a link failure, we need to send a UAVto the midpoint between two working devices nearest to the broken devices. In other words, if V_i is out of order, we select the two nearest available working devices like V_{i-1} and V_{i+1}. Note that the location of V_i is $(X_k,\ Y_l, Z_m)$ where k and l and m are the GPS latitude and longitude information. We dispatch a UAV to the geographic midpoint location[2] between V_{i-1} and V_{i+1}. By doing so, we can still create a network so that users can access the Internet with the pre-deployed devices.

3 EVALUATION

3.1 Implementation

To provide Internet access to users in emergencies, we implement a platform-independent hotspot application for dynamic network provision based on the Resin.io Cloud service. The application is automatically installed into an embedded device through the Resin.io

[1] https://www.mapbox.com
[2] http://www.geomidpoint.com

framework. First, we need to install the base OS image and the application through Resin.io over the normal network interface. After installing the application in the device, the device serves as a hotspot to create a network. To allow users to access the Internet, at least one device needs two network interfaces. One is used to connect to the Internet and the other is used to provide Internet access to users through an emergency network.

We implement a hotspot application based on the Network Manager library to activate the connection [21]. By using the NetworkManager API [20], a Python script enables a Linux device to act as a hotspot. The script is run from the docker file, which packages all the dependencies needed to run this script in the embedded device. The Dockerfile should specify one available network interface when starting the script. The script obtains the connection name (i.e. SSID) and searches for available interfaces with Internet access. Finally, the script bridges the two connections by activating network interfaces to enable a device to act as a hotspot. The device will be a Wi-Fi hotspot since we currently handle only Wi-Fi.

Once these devices are deployed with the access point connection name and configuration, the connectivity manager takes care of the handover of user devices [26] through the Resin.io framework. In other words, Wi-Fi features (i.e. IEEE 802.11 series) connect a user device to the Internet through the first available hotspot device. If the user is mobile and the user device gets out of range of the hotspot, then the user can access the next available hotspot device through disassociation and broadcasting.

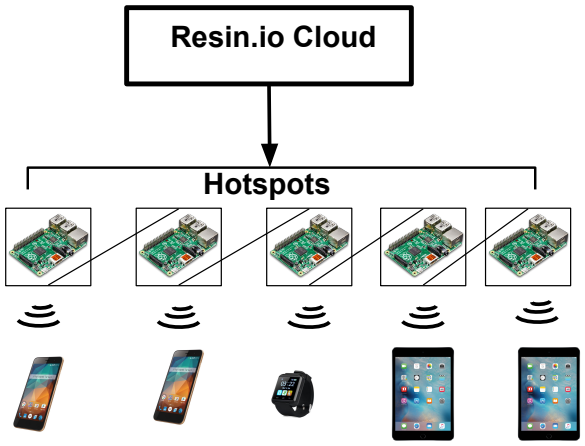

Figure 2: Test Environment

4 EXPERIMENTAL RESULTS

The test environment consists of the Resin.io Cloud, five Raspberry Pi-3 model B [1] devices, a laptop (Acer aspire e5-575G), a Netgear N600 router and an Internet connection, as shown in Figure 2. Instead of real UAVs, we used the Raspberry Pi for our basic experiments. We will test with real UAVs and mobile embedded boards in future work instead of using only embedded boards. The laptop has 8GB of RAM and an Intel Core i5-6200U CPU @ 2.30GHz processor. The router supports dual band Wi-Fi and 802.11n protocol and an Internet speed of up to 350 Mbps. The Internet connection downloads up to 75Mbps and uploads up to 35 Mbps. The Raspberry Pi system consists of a 1GB RAM and a 1.2 GHZ quad-core ARM Cortex A53 processor.

Our experiments evaluates the performance metrics of the proposed system in terms of CPU, memory, and disk usage. We also tested network performance by using download and upload speeds of the hotspot device that we designed. We implemented a Python script by using *flask* API [23] to provide a web application and *psutil* [18] to obtain the memory and CPU usage. Once the script was deployed in the device, the web server ran on port 5000 on the Raspberry Pi. In order for external devices to send GET requests to the devices, we exposed the Docker container 5000 port on port 80 to access the Resin.io server. To measure the performance of the entire setup, we tested internet connectivity using five user mobile phones. In order to note changes in performance, the devices were programmed to connect to the hotspot one at a time during minutes 6, 12, 22, 29 and 38.

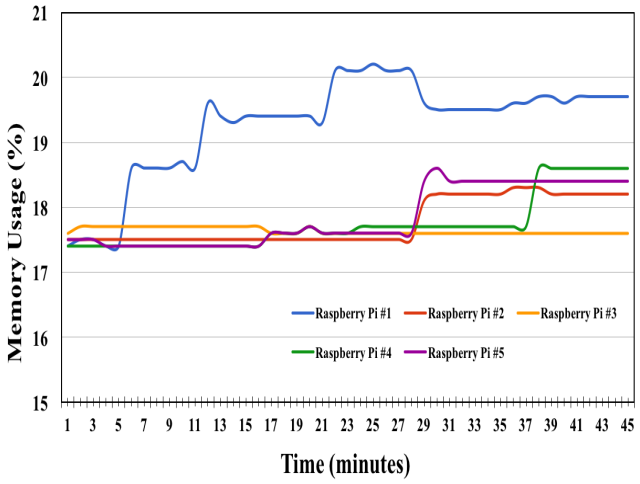

Figure 3: Memory usage of the hotspot devices

Figure 3 shows the memory usage of the hotspot devices. We connected one mobile phone to the hotspot device after five minutes and an additional phone was connected after several minutes: no one used the hotspot device during the first five minutes; at six minutes, one phone was connected; after forty five minutes, all five mobile devices were connected. Each time a user mobile phone got connected to the hotspot device, there was a considerable increase in memory usage, as shown in Figure 3. For example, the memory usage for the Raspberry Pi #1 spiked at 6 minutes, and so on. When three mobile phones were first connected to the hotspot, only the Raspberry Pi #1 was used and the rest of the devices were still idle. However, the memory usage of the Raspberry Pi #1 decreased after four user mobile phones were connected to the hotspot. The workload (i.e. the number of user mobile phones using the hotspot) was distributed into various hotspot devices. Thus, the workload of the Raspberry Pi #1 was divided among other Raspberry PI devices, thereby increasing the memory usage of those devices.

Figure 4 shows the CPU usage of hotspot devices according to the number of user mobile phones connected to the hotspot. Similar to

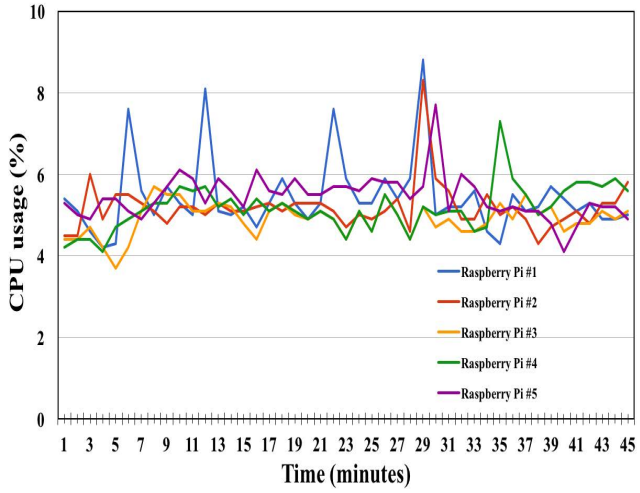

Figure 4: CPU usage of the hotspot devices

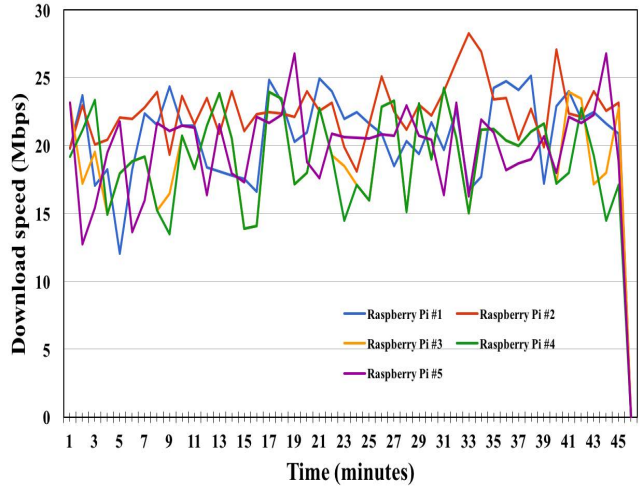

Figure 5: Download speeds of the hotspot devices

memory usage, the CPU usage showed noticeable spikes whenever user mobile devices got connected to the hotspot devices. CPU usage ranged from 4% to 8.8% depending on the number of users: the more users, the higher the CPU usage. However, the disk usage of the hotspot devices was not affected by the number of users, but rather remained constant. The majority of the disk is used by the Resin OS and a small portion was used by the deployed application. The disk usage of all five devices was 4.2 % of the total 16 Gigabytes.

Figure 5 and Figure 6 show the download and upload speeds of the hotspot devices. Since the hotspot devices access the network through a common router, their speeds are divided equally among them. Download speeds of a device ranged from 12 - 28 Mbps at any point in time and upload speeds varied from 5.2 - 6.2 Mbps. Due to the limitation of the number of available mobile phones, we couldn't evaluate our hotspot on a large scale. In future work, we

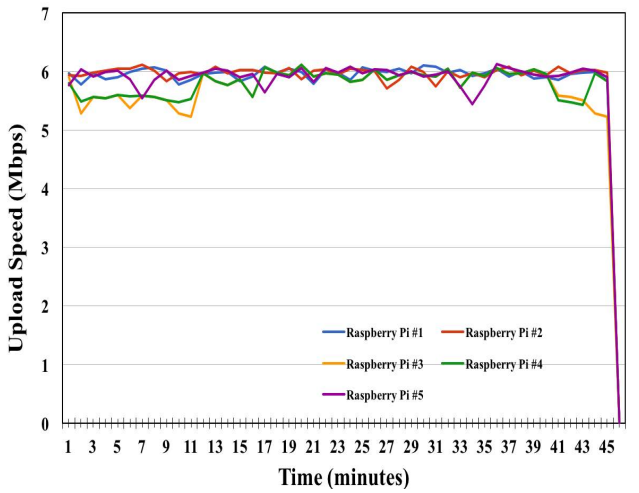

Figure 6: Upload speeds of the hotspot devices

will evaluate the proposed system with a large number of users and different devices.

5 RELATED WORK

WiFi technology has grown exponentially along with other technologies over the past decade. WiFi is a wireless networking technology based on 802.11 standards [27]. Wi-Fi operates on the 2.4GHz unlicensed band. It is a readily available technology in a wide variety of electronic devices. Smartphones and wearable devices like smart watches frequently use Wi-Fi to access the Internet. People have constant access to these smart devices. One of the most significant additions to 802.11 protocols is the ability for user devices to be mobile. For instance, users utilize WiFi service provided by businessed to access the Internet to make a VoIP call. The device's Wi-Fi radio will automatically search for access points to connect to provide a seamless handover [17]. This process is referred to as roaming. When the users device gets out of the range of one access point, the devices Wi-Fi radio will issue disassociation frames to this access point. It will then broadcast probe requests to scan for available access points and then re-associate [17, 26].

By forming a mesh network using WiFi devices, we are able to create resilient and self-configuring networks that solve many of the problems related to ad hoc communications [15]. WiFi mesh networks have been used to help UAVs in search and rescue operations [8, 16]. In a traditional communication network, clients rely on a base station or other external communication facilities that are not the best to use when a crisis hits the area. When the network infrastructure in a crisis hit zone is damaged, a wireless ad hoc network will have to be established in this zone. The nodes in this network need to be interconnected in order to provide effective emergency communications, with monitoring of a variety of factors including traffic load. As an additional functionality, by establishing a distributed Wi-Fi network using embedded devices, it is also possible to use sensors to collect essential information within the emergency zone. Such a distributed network has been used to collect surrounding information to carefully monitor a museum [24].

UAV technology has advanced dramatically along with its many possible applications in life for monitoring, marketing, and environmental tools. Moreover, UAVs have also been useful in emergencies [9, 10, 16]. Gao and et.al. utilized UAVs to create a relay network for a UAV-based search and rescue project [16]. They included functions like positioning, message pushing, and adjusting flight path. In addition to the aerial network, the authors also propose a self-organizing mobile ad hoc network on the ground. Paul Bupe and et.al proposed a method to deploy a large fleet of drones operating autonomously for temporary emergency communications [7]. UAVs were also used in collecting sensor data including images to identify victims and their location [6, 9, 10, 14].

6 DISCUSSION AND FUTURE WORK

The current Rein.IO supports various embedded devices, including a family of Raspberry Pi, a family of BeagleBone, several Samsung Artik devices, and Intel Edison, which are the most popular embedded devices for the Internet of Things (IoT). It also supports various types of system architectures like ARM and i386 and several Unix-based operating systems like Ubuntu, Fedora, and Debian. UAV devices need to be incorporated since UAVs are the most promising devices for our future emergency infrastructure; however, the development of the Resin.io base image for UAVs is outside of the scope of this paper. Instead, we were able to use Raspberry Pi to represent the UAVs that we intend to use in our emergency system.

Many research challenges still remain in order to obtain the most efficient handling of emergency communication networks and ensure nonstop user access to the Internet. Embedded boards are very inexpensive devices, but are powerful enough to create a network in emergencies. There are many remaining problems to solve: (1) How many devices are required to cover a specific emergency zone?, (2) Where is the best location of the device?, (3) What is the optimal range of UAVs to cover an emergency zone?, and (4) How fast can dynamic mobility connections and mobile emergency services be automatically validated?

In future work, we will address these remaining challenges. We will enable Wi-Fi based mesh networks for emergencies based on our infrastructure. To enable interaction and connectivity between emergency teams, victims, and communities, we enable multi-modal communication of an edge device to inter-operate with multiple existing network standards and segments. To achieve this, we will work on various types of network communications, such as Bluetooth and radio communication. In addition, UAVs are the most promising devices to be used for an emergency. We will design our own cost-effective UAVs.

7 CONCLUSION

In emergency and disaster scenarios, if the existing network infrastructure goes down, it is vital to have emergency network communications restored immediately. This paper presents a way to dynamically deploy a distributed Wi-Fi infrastructure using low cost embedded devices and providing network services on the go. To establish a distributed Wi-Fi infrastructure during disaster scenarios, we developed a hotspot application that can be used on any and all devices. We have used Resin.io as a platform for accessing embedded devices through the Internet and for pushing our application or updates. It offers a model for deploying network services quickly regardless of hardware specifications. By implementing this project, the time that it takes to deploy a temporary network infrastructure in a crisis-hit area can be greatly reduced. We implemented our prototype and evaluated it on a fleet of Raspberry pi devices. We have explained the effectiveness of our proposed system by performing experiments and illustrating the results.

REFERENCES

[1] Abishur. 2014. STICKY: Getting Started with the Raspberry Pi. (2014). https://www.raspberrypi.org/forums/viewtopic.php?f=91&t=4751

[2] WW Norton Anthony Townsend. 2013. Smart cities. *Places Journal* (2013).

[3] Dipankar Raychaudhuri Kiran Nagaraja Morley Mao Arun Venkataramani, James F Kurose and Suman Banerjee. 2014. Mobilityfirst: a mobility-centric and trustworthy internet architecture. *ACM SIGCOMM Computer Communication Review* (2014), 74–80.

[4] ATT. 2016. Network disaster recovery. (2016). http://www.corp.att.com/ndr/

[5] Carsten Willems Christof Paar Benedikt Driessen, Ralf Hund and Thorsten Holz. 2012. Donť trust satellite phones: A security analysis of two satphone standards. In *Security and Privacy (SP), 2012 IEEE Symposium*. IEEE, 128–142.

[6] Piero Boccardo, Filiberto Chiabrando, Furio Dutto, Fabio Giulio Tonolo, and Andrea Lingua. 2015. UAV deployment exercise for mapping purposes: Evaluation of emergency response applications. *Sensors* 15, 7 (2015), 15717–15737.

[7] Paul Bupe, Rami Haddad, and Fernando Rios-Gutierrez. 2015. Relief and emergency communication network based on an autonomous decentralized uav clustering network. In *SoutheastCon 2015*. IEEE, 1–8.

[8] Muhammad Hafeez Chaudhary and Bart Scheers. 2016. Dynamic channel and transmit-power adaptation of WiFi mesh-networks in search and rescue operations. In *Military Communications and Information Systems (ICMCIS), 2016 International Conference on*. IEEE, 1–8.

[9] Kyoungah Choi and Impyeong Lee. 2011. A UAV-based close-range rapid aerial monitoring system for emergency responses. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci* 38 (2011), 247–252.

[10] Kyoungah Choi, Impyeong Lee, Juseok Hong, Taewan Oh, and Sung Woong Shin. 2009. Developing a UAV-based rapid mapping system for emergency response. In *SPIE Defense, Security, and Sensing*. International Society for Optics and Photonics, 733209–733209.

[11] Software Freedom Conservancy. [n. d.]. Git. ([n. d.]). https://git-scm.com/

[12] Leandro Navarro Davide Vega, Llorenc Cerda-Alabern and Roc Meseguer. 2012. Topology patterns of a community network: Guifi. net. *Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 IEEE 8th International Conference* (2012), 612–619.

[13] Docker.com. 2017. Get Started, Part 1: Orientation and setup. (2017). https://docs.docker.com/get-started/

[14] Patrick Doherty and Piotr Rudol. 2007. A UAV search and rescue scenario with human body detection and geolocalization. In *Australian Conference on Artificial Intelligence*, Vol. 4830. Springer, 1–13.

[15] Glenn Fleishman. 2017. Wireless mesh networks: Everything you need to know. (2017). https://www.pcworld.com/article/3212444/wi-fi/mesh-network-explained.html

[16] Tianhan Gao, Fan Lang, and Nan Guo. 2016. An Emergency Communication System Based on UAV-assisted Self-Organizing Network. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2016 10th International Conference on*. IEEE, 90–95.

[17] Jim Geier. 2013. How Wi-Fi Roaming Really Works. (2013). http://www.wireless-nets.com/resources/tutorials/how_roaming_works.html

[18] Giampaolo Rodola' Jay Loden, Dave Daeschler. 2009. psutil. (2009). https://github.com/giampaolo/psutil

[19] Mihui Kim Mson Itkin and Younghee Park. 2017. Development of Cloud-Based UAV Monitoring and Management System. *Journal of Sensors* (2017).

[20] The Gnome Project. 2012. NetworkManager Reference Manual. (2012). https://developer.gnome.org/NetworkManager/1.8/

[21] Resin.io. 2016. Networking on resinOS 2.0. (2016). https://docs.resin.io/deployment/network/2.0.0/

[22] Resin.io. 2017. How does resin.io work? (2017). https://resin.io/how-it-works/

[23] Armin Ronacher. 2017. Flask API. (2017). http://flask.pocoo.org/docs/0.12/api/

[24] Marco Parvis Simone Corbellini. 2016. Wireless sensor network architecture for remote non-invasive museum monitoring. In *Systems Engineering (ISSE), 2016 IEEE International Symposium on*. IEEE.

[25] InfiniBand Solutions. 2011. Delormeś inreach. *Linux Journal* (2011), 209.

[26] Zhenyu Song, Longfei Shangguan, and Kyle Jamieson. 2017. Wi-Fi Goes to Town: Rapid Picocell Switching for Wireless Transit Networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 322–334.

[27] the Institute of Electrical and Inc. (IEEE) Electronics Engineers. 2017. IEEE 802.11TM WIRELESS LOCAL AREA NETWORKS. (2017). http://www.ieee802.org/11/#