Spring 2018

Constructing Dynamic Ad-hoc Emergency Networks using Software-Defined Wireless Mesh Networks

Roop Kumar Sriramulu
San Jose State University

CONSTRUCTING DYNAMIC AD-HOC EMERGENCY NETWORKS USING SOFTWARE-DEFINED WIRELESS MESH NETWORKS

A Thesis

Presented to

The Faculty of the Department of Computer Engineering

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Roop Kumar Sriramulu

May 2018

The Designated Thesis Committee Approves the Thesis Titled

CONSTRUCTING DYNAMIC AD-HOC EMERGENCY NETWORKS USING SOFTWARE-DEFINED WIRELESS MESH NETWORKS

by

Roop Kumar Sriramulu

APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING

SAN JOSÉ STATE UNIVERSITY

May 2018

Dr. Younghee Park	Department of Computer Engineering
Dr. Kaikai Liu	Department of Computer Engineering
Dr. Hyeran Jeon	Department of Computer Engineering

ABSTRACT

CONSTRUCTING DYNAMIC AD-HOC EMERGENCY NETWORKS USING SOFTWARE-DEFINED WIRELESS MESH NETWORKS

by Roop Kumar Sriramulu

Natural disasters and other emergency situations have the potential to destroy a whole network infrastructure needed for communication critical to emergency rescue, evacuation, and initial rehabilitation. Hence, the research community has begun to focus attention on rapid network reconstruction in such emergencies; however, research has tried to create or improve emergency response systems using traditional radio and satellite communications, which face high operation costs and frequent disruptions. This thesis proposes a centralized monitoring and control system to reconstruct ad-hoc networks in emergencies by using software-defined wireless mesh networks (SDWMN). The proposed framework utilizes wireless mesh networks and software-defined networking to provide real-time network monitoring services to restore Internet access in a targeted disaster zone. It dispatches mobile devices including unmanned aerial vehicles and self-driving cars to the most efficient location aggregation to recover impaired network connections by using a new GPS position finder (GPS-PF) algorithm. The algorithm is based on density-based spatial clustering that calculates the best position to deploy one of the mobile devices. The proposed system is evaluated using the common open research emulator to demonstrate its efficiency and high accessibility in emergency situations. The results obtained from the evaluation show that the performance of the emergency communication system is improved considerably with the incorporation of the framework.

ACKNOWLEDGMENTS

I would like to sincerely thank my advisor, Dr. Younghee Park, for giving me the opportunity to do this thesis under her guidance. I am incredibly grateful for her support, patience, and encouragement which kept me on track throughout my time as a graduate student. She helped me with organizing the thesis right from the initial draft by providing me with valuable feedback.

I would also like to thank Dr. Kaikai Liu and Dr. Hyeran Jeon for serving on the defense committee. I am also thankful to Abhilash Tuse for his help in performing the experiments. Thanks are in due to my friends, especially Goutham Nekkalapu, for hearing out my problems and giving me advice.

Above all, I would like to thank my parents for their enduring love and support. I dedicate this work to them.

TABLE OF CONTENTS

List of Figures

1 Introduction

Over the last few years, the increasing number of emergency and disaster scenarios worldwide has created the need for more reliable emergency network communications [1]. One of the familiar characteristics of a catastrophic scenario is the damage done to the power system and the communications infrastructure. This damage thwarts efforts of first responders and other concerned people to communicate with victims in the affected emergency zone. A survey on emergency communication systems during a disaster suggests the need for an “Always-On-Network” to ensure a reliable communication channel between the people in emergency zones and the outside world [2].

Existing solutions for providing communication during such disasters present certain disadvantages. Services like cell on wheels (CoW) [3], amateur radio systems [4], and satellite communications are examples of existing solutions that have not kept pace with current emergency communication requirements. According to Singh et al. [5], the disadvantages of using these technologies include traffic congestion, high operating costs, licensing, and complexity of usage. Hence, novel and updated approaches to emergency Internet service are needed to sidestep these complexities and provide a highly available, dynamically deployable, and centrally managed network [2].

Wireless technology has seen tremendous growth in the last two decades, playing a crucial role in search and rescue operations. The introduction of mesh networks has created a new class of self-configuring networks [6]. The wireless mesh network (WMN) can be of enormous help to first responders in rescue operations as nodes can join and leave the network at any time. Even though the research and development of WMNs have seen considerable improvement, a significant market has not yet developed for the use of WMNs in forming emergency networks.

As WMNs are decentralized, it is difficult to have consolidated control over the routing devices [6]. During disaster scenarios, however, it is critical to have a centralized view over the network to monitor the status of various devices. Software-defined networking (SDN) can add such centralization to mesh networks to accomplish the improved management and control required to effectively address emergency situations. The combination of SDN and WMN offers both a dynamic infrastructure and centralized management. This hybrid combination is referred to as software-defined wireless mesh networking (SDWMN) [7].

In a disaster scenario, there is a high probability of communication disruption due to battery drainage or physical damage to networking devices [5]. Since time is of the essence during search and rescue operations, the need to replace or fix damaged devices may result in human casualties. This means that ready-to-use backup devices are urgently needed to replicate the work of damaged devices. In 2011, the Federal Communications Commission (FCC) considered the use of unmanned aerial vehicles (UAVs) or drones to utilize the aerial advantage in temporarily restoring emergency communications [8]. Such UAVs and self-driving vehicles have enormous potential to carry backup devices to an area to mitigate localized failures. However costly, the use of unmanned vehicles like drones is an inevitable part of emergency systems. Nevertheless, it is not practical with existing technology to replace every failed node using unmanned vehicles. Thus, this thesis designs an efficient placement algorithm to provide the exact global positioning system (GPS) coordinates of the destination to deploy the vehicles and recover the network.

The scientific contributions of this thesis are as follows.

1. A new system architecture of emergency communications is proposed by using SDN and unmanned vehicles to manage WMNs.

2. The emergency communication system (ECS) is managed by a centralized control system with the help of SDN to identify failures in emergency networks and to dispatch ad-hoc devices for rapid reconstruction.

3. An efficient unmanned vehicle placement algorithm is proposed to effectively reduce the number of vehicles deployed to recover a failed network.

4. The proposed system is implemented in the common open research emulator (CORE) to evaluate network latency and packet processing times with different parameter settings [9].

The remainder of the thesis is organized as follows. Chapter 2 gives a brief background on the different technologies used in this work. Chapter 3 covers related work that has previously been done on emergency communications. Chapter 4 presents a detailed description of the proposed system design and implementation. Chapter 5 presents the evaluation of the system architecture. Chapter 6 presents the discussion and future work, followed by the conclusion in Chapter 7.

2 Background

This chapter provides an overview of various technology domains used in this work. Section 1 describes the concept of WMN, section 2 gives a brief introduction to SDN, and section 3 illustrates the concept of SDWMN.

2.1 Wireless mesh networks

WMNs are a type of mobile ad-hoc network (MANET) that can dynamically self-configure into arbitrary network topologies [6]. This feature enables a seamless connection of client devices to the network. Even though the concept of the WMN is not new, it has not been used in mainstream Wi-Fi or 802.11 until recently. WMNs fragment the long communication links into a sequence of smaller hops to enhance the signal through intermediary nodes [10]. These intermediate nodes not only increase signal strength but also help in routing the packets to the appropriate destination. Each intermediate node in the mesh topology develops its routing database by broadcasting some topology discovery packets. This topology discovery takes place due to the operation of distributed routing protocols on these nodes. Some examples of distributed routing protocols are better approach to mobile ad-hoc networking (B.A.T.M.A.N.) [11], optimized link state routing protocol (OLSR) [12], ad hoc on-demand distance vector (AODV) routing [13], and babel [14]. WMNs are thus completely decentralized as each node acts as its controller.

The infrastructure for WMN is entirely different from that of the traditional wireless local area network (WLAN). Even though the conventional WLAN is the currently preferred technology for providing wireless Internet connectivity to campus networks, office networks, and home networks, WLANs are constructed using a fixed network infrastructure. The user connects wirelessly to the WLAN access point within a single hop and thereby connects to the Internet backbone through a wired network. This single hop method increases the cost of constructing the wired infrastructure as the range of coverage increases. By contrast, a WMN does not rely on dedicated access points, and only one or a few of the routers in a WMN are connected to the backbone network. The rest of the nodes connect wirelessly to each other if they are in range and provide Internet access to users through one or multiple hops.

Constructing WMNs in bulk is an efficient way to provide cost-effective Internet access to a community of mobile users. The motivation for creating a community-based WMN, known as a wireless community network (WCN), is increasing in recent years due to a variety of factors such as the desire to provide free Internet access to the public, to support research purposes, or to provide better information access during disasters. The "guifi.net" community network in Spain is an example of a WCN containing more than 33,000 working nodes [15]. It is one of the largest open-access and free mesh networks connecting users to share information with each other.

2.2 Software-defined networking

The three main logical components of the Internet communications infrastructure are the management plane, the control plane, and the data plane. The data plane handles the processing of data packets to and from the user. The control plane takes care of the routing of data packets, whereas the management plane takes care of management and configuration services [16]. Since the routers in traditional networking have all three planes present inside them, manual configuration is required for every device. As a result, network administrators were in need of a physically centralized controller to manage, configure, and make routing decisions for all the network devices in a given network [17]. This single pane of management requires separation of the planes in network devices. Figure 1 illustrates the logical decoupling of the three planes in networking devices.

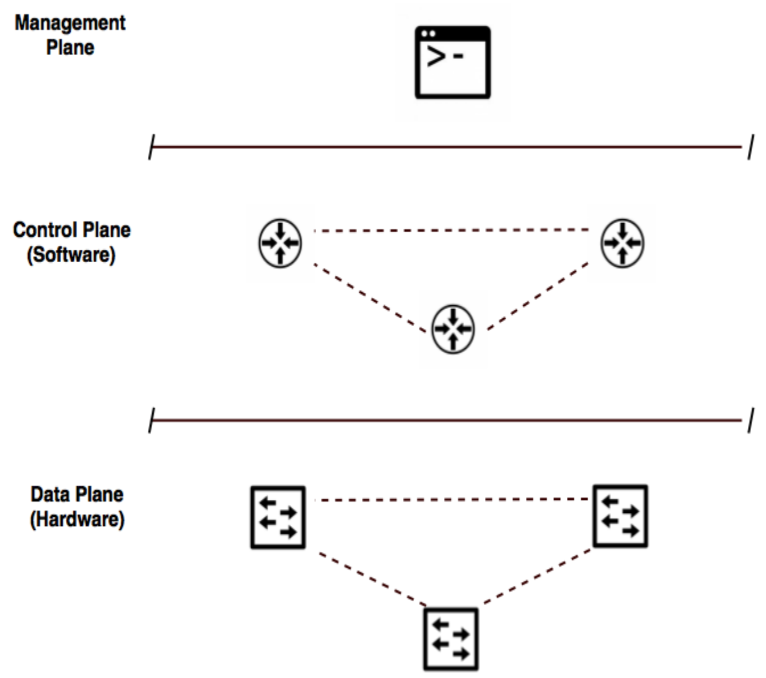


Figure 1: Decoupling of the control plane, data plane and management plane in computer networks.

A paradigm shift almost a decade ago resulted in the creation of SDN, with the aim of removing the restrictions of conventional networks. According to Marschke, Doyle and Moyer [16], "SDN is a conceptual framework in which networks are treated as abstractions and are controlled programmatically, with a minimal direct touch of individual network components." With the development of SDN, network administrators can build custom policies and program the network dynamically. The primary protocol used in the SDN controller to communicate with network devices is OpenFlow [16]. OpenFlow is a southbound protocol as it deals with the underlying network devices. It is an open-source protocol that allows inter-operation of devices from multiple vendors [17].

2.3 Software-defined wireless mesh networks

This section presents an overview of the combination of SDN and WMN technologies. The idea of introducing SDN into the conventional mesh network requires conformity of regular mesh routing and SDN based routing. Mesh-based routing gathers the topology details of the network while SDN-based routing offers services like routing, traffic engineering, and load balancing [18]. In SDN, the concepts of in-band and out-of-band differentiate the type of control channel communication with the switch [19]. The control channel is separate from the regular data network in an out-of-band network, while it interoperates with the data network in an in-band network. A switch could be included in the in-band network by installing a set of default flow-rules [20]. Figure 2 represents a simple topology of the in-band SDWMN.

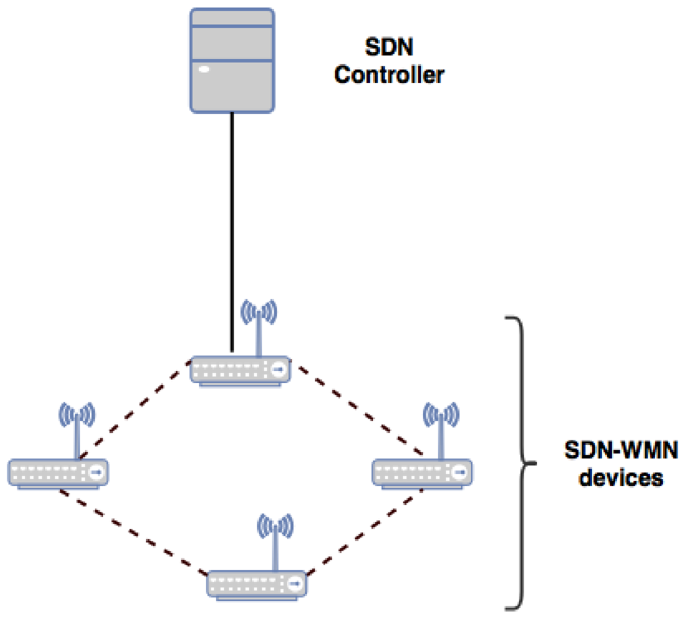


Figure 2: SDN controller connected with a WMN.

In the wired Internet, the reliability of the cable determines the reliability of the connection between terminals. Wired connections have a consistent cable capacity and negligible fluctuation due to several factors [19]. However, in wireless technology, the interface requires configuration and control for smooth operation of SDWMN. This configuration could be done by tweaking parameters like the frequency, service set identifiers (SSIDs) and transmission power. Hence, it is essential for SDWMN to provide access to control these parameters. As suggested by Rademacher et al. [19], the capabilities for querying and modifying wireless interface parameters are classified into three requirements.

1. The mesh nodes should be able to provide an interface for the controller to tap in and configure the interfaces.

2. The mesh nodes should be able to communicate over a secure channel with the controller over an SDN protocol.

3. The controller should be able to present the configurable parameters to the northbound applications to control the mesh nodes.

3 Related Work

Many approaches have been taken to provide communication services during an emergency, a majority of which involve the use of wireless technology. Recently, the concept of combining SDN with wireless ad hoc technology is gaining traction. This chapter discusses various techniques that have previously been developed to apply to ECS. The related work described below has inspired the proposed framework.

3.1 Wireless ad hoc networks assisting in ECS

In 2010, Bai et al. [21] proposed an integrated emergency communication system (IECS) combining a wireless sensor network (WSN), mobile ad hoc networking (MANET), satellite communication, and cellular communication. In IECS, a WLAN is established using a MANET, and a monitoring system is built using a WSN to gather relevant information. The IECS uses a dual gateway architecture to provide Internet access to users. The first gateway is established utilizing a satellite gateway to connect to the satellite network, and the second gateway is built using a cellular gateway to connect to the cellular network.

A few years later, Konak [22] proposed an ECS to increase client connectivity by dynamically locating a set of nodes corresponding to changes in network topology. The author utilizes a flocking-based algorithm to determine the placement location of the nodes based on the network metrics gathered. The author also defines a set of rules based on which the flocking algorithm determines new locations of problem nodes.

More recently, Xu, Ota, and Dong [23] instead designed algorithms to place ad hoc routers along with a routing strategy to dynamically connect users in a post-disaster scenario. This router placement algorithm resolves the content-centric networking (CCN) problem using the concept of graphic union covering, and the routing strategy algorithm applies the classical breadth-first search (BFS) concept in graph theory. The authors also conducted experiments simulating packet delivery using real-world datasets.

3.2 Unmanned aerial vehicles (UAVs) assisting in ECS

Gao, Guo, and Lang [24] proposed the use of UAVs in ECS. In this work, they presented a dual architecture for users to communicate with each other and the outside world. In their proposed design, UAVs form a self-organizing network in the sky to relay data traffic from client devices. Client devices on the ground together create a backup network in the absence of the UAVs. Grodi and Rawat [25] proposed a similar approach to ECS using UAVs. Their system describes the recovery of damaged communication networks through the deployment of UAVs throughout the cellular zone. The authors designed an algorithm to install the optimal number of UAVs for a given cellular tower based on its transmission range.

Mekikis, Antonopoulos, Kartsakli, Alonso and Verikoukis [26] studied the probability of successful network recovery in a post-disaster situation using aerial nodes. They performed a three-step recovery probability analysis: (1) between user devices and airborne nodes; (2) between aerial nodes; and (3) between aerial nodes and the emergency operations center (EOC). The authors also conducted experiments to confirm the probability of success using airborne nodes for network recovery.

Erdelj, Król and Natalizio [27] explored the roles of the WSN and UAVs in the context of emergency network communications. They provided a detailed list of application domains involving WSN and UAVs. The authors also provided visual illustrations of a fixed and of a mobile UAV station to be used in post-disaster areas. These stations were shown to efficiently carry multiple ready-to-go UAVs.

3.3 Software-defined networks assisting in ECS

Baldini, Sturman, Dalode, Kropp and Sacchi [28] proposed a disaster communication system called EULER, based on software-defined radio (SDR) supporting interoperability between different wireless terminals, including handheld terminals, vehicular terminals, nomadic terminals and base stations of a fixed infrastructure. The authors provided a detailed description of the integration of the system into backbone networks and demonstrated it with line-of-sight and beyond line-of-sight scenarios.

Xie, Wang, Wang and Lu [29] presented a disaster-resilient network based on SDN. To reduce controller overhead of the controller in an emergency, they proposed a two-level SDN controller, the first to proactively monitor local failures and the second to reactively monitor global failures. The authors implemented the system using the Nox SDN controller, OpenFlow southbound protocol, and the Mininet emulator.

Another novel approach to ECS using SDN was proposed by Manic et al. [30]. In their paper, the authors presented a system to provide dynamic bandwidth allocation and priority-based packet switching for users in an emergency zone. They also implemented services like the emergency notification system to test the framework.

4 System Design

This chapter introduces the SDWMN framework for the ECS and describes its components. The system provides highly available Internet access to users in an emergency zone. Figure 3 provides a general overview of the proposed ECS. It shows an example of the interface between a wireless mesh infrastructure and SDWMN zones for an emergency recovery system.

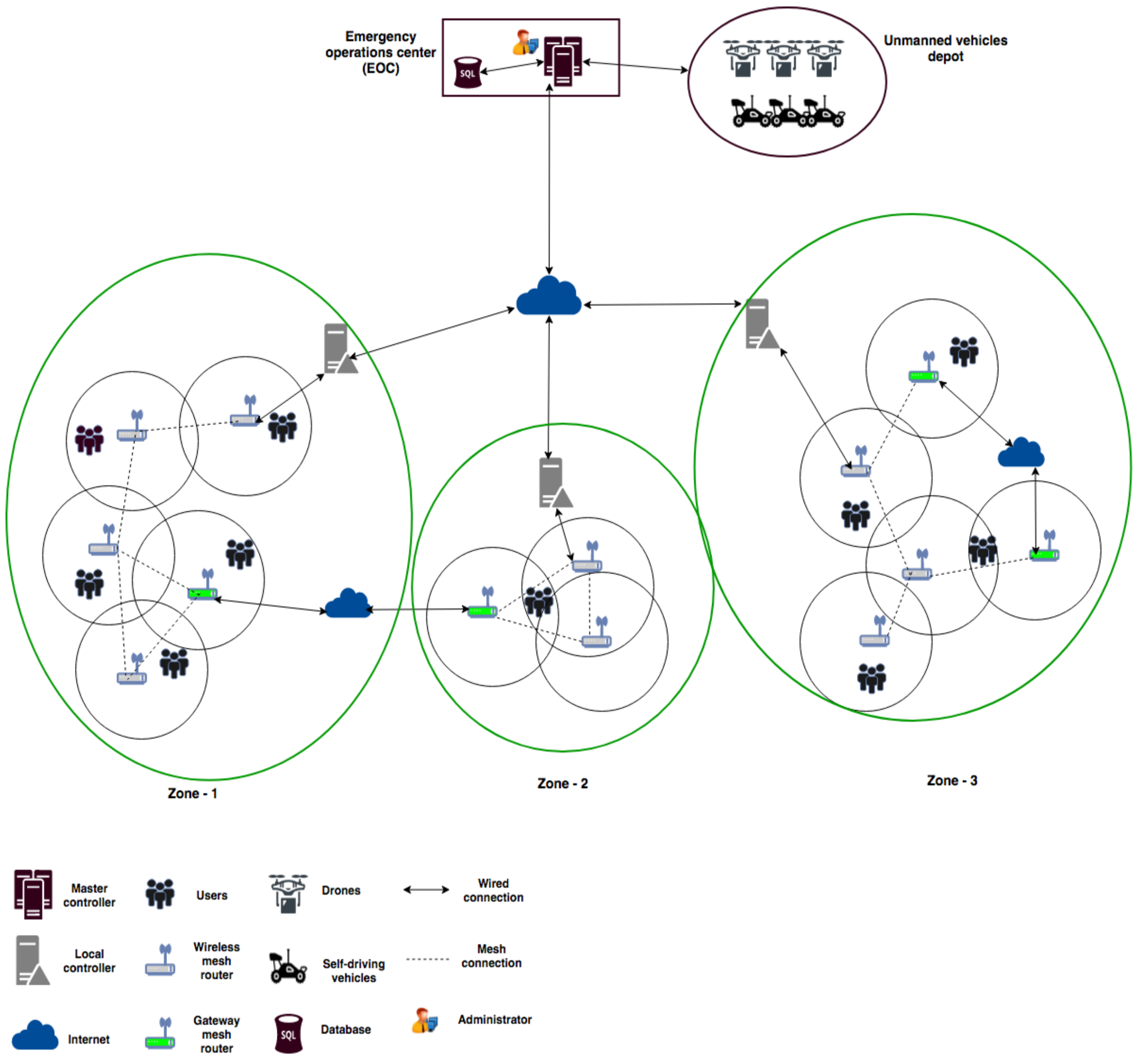


Figure 3: Overview of the proposed ECS, illustrating SDWMN zones monitored by an EOC.

A predeployment assumption is that the emergency zone will have a community-owned wireless mesh infrastructure and an EOC. The steps to establish the EOC are beyond the scope of this paper. Personnel in the EOC are assumed to have access to the monitoring system to oversee emergency operations. During a disaster, if the central management system recognizes a failure in the mesh network, the emergency personnel will authorize the deployment of an unmanned vehicle to an appropriate destination to restore the network. The system architecture consists of the following main components:

1. Wireless mesh infrastructure

   This is a pack of hybrid wireless routers in a region connected in a mesh topology providing Internet access to the public.

2. SDN controller

   For each local network, an SDN controller provides routing policies and updates a central database with the status of the local network in real time.

3. Central management system (CMS)

   A CMS has a centralized view over all SDN controllers and provides the mechanism for replacing failed mesh nodes.

4. Unmanned-vehicle depot

   This is a backup supply of small drones and self-driving cars that can be deployed to restore network connectivity.

5. User devices

   Users in an emergency region are assumed to have a heterogeneous set of devices, such as smartphones, laptop, and tablets.

The proposed framework is developed in a Linux environment, and the SDWMN is implemented using the open-source SDWMN tool called wmSDN developed by Detti, Pisa, Salsano and Blefari-Melazzi [31]. Figure 4 shows the logical hierarchical structure of the proposed system architecture.

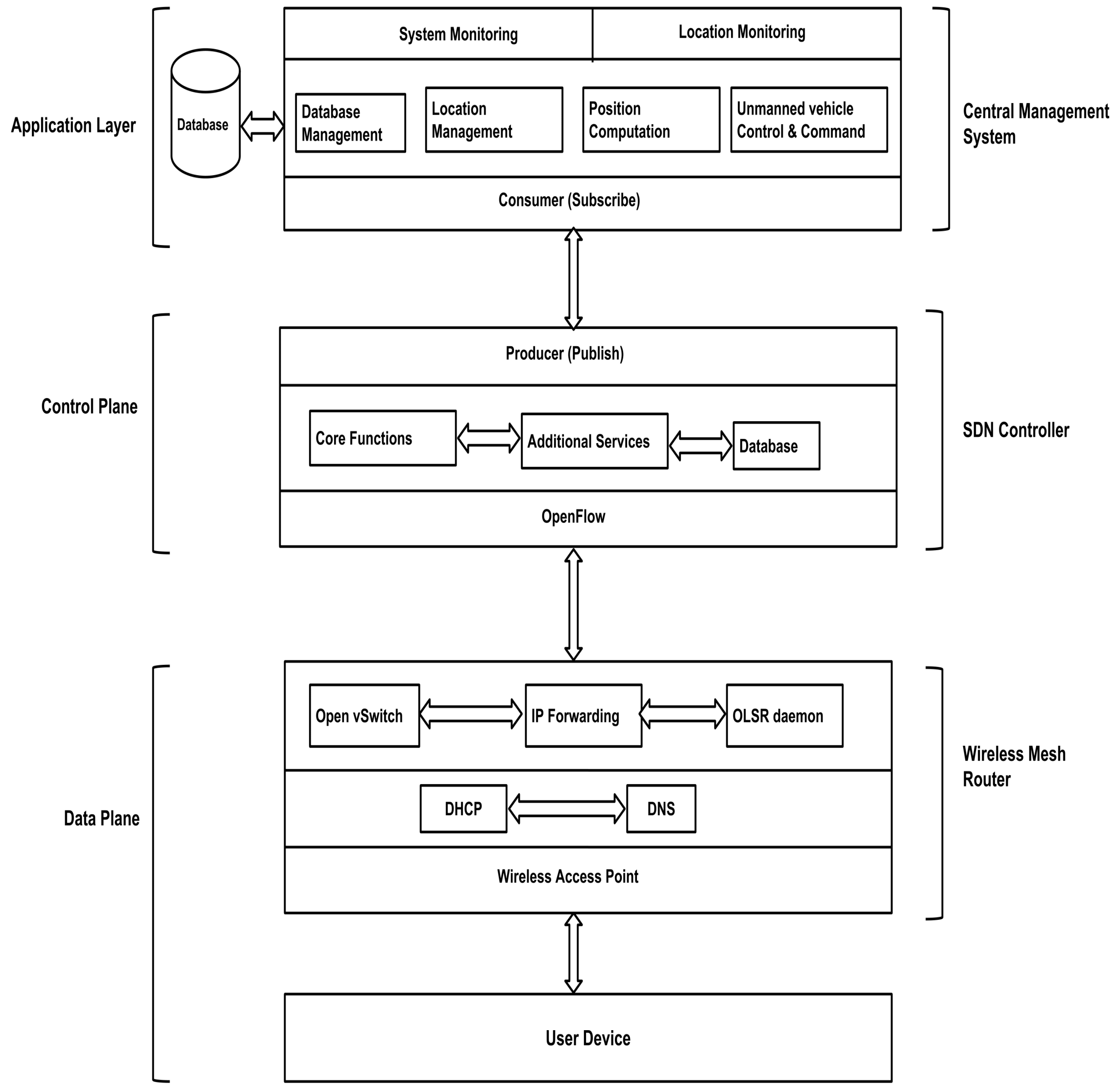


Figure 4: Components of the System Architecture.

4.1 Wireless mesh infrastructure

This section describes the workings of individual mesh routers that are assumed to be deployed as part of the community-owned Internet utility. Wireless mesh nodes fulfill the following functions.

1. They provide access points for users to connect to the Internet.
2. They dynamically assign IP addresses to connected users.
3. They forward and cache name resolution queries.
4. They establish basic IP forwarding between other mesh nodes and gather topology information.
5. They establish a control channel with the SDN controller to download the routing logic.

4.1.1 Wireless access point

The wireless access point is a transceiver for data traffic and is used to create a local area network (LAN) of user devices in a particular region. It provides a SSID for user devices to connect to by using a wireless interface [32]. Traffic is routed through the mesh network from user devices to the backbone network. The backbone network connects various LANs to each other or to the Internet. Hostapd is a userspace daemon that provides the functionality and management of an access point in the wireless mesh router [33]. The hostapd requires a Linux host with an 802.11 Wi-Fi card supporting the access point (AP) mode. Once the hostapd is configured and enabled, it presents users with the SSID to connect to the router. For example, let us assume that the SSID created by the hostapd is "emergency-hotspot." When user devices scan for available access points, the SSID "emergency-hotspot" will appear on the devices. Although the hostapd provides support for authentication of user devices, in an emergency scenario, authenticating user devices is not a priority.

4.1.2 DHCP and DNS services

Every device connecting to a network is assigned a unique numerical label called the Internet protocol (IP) address, which is necessary to communicate with other devices. Depending on the type of IP version, either a 32-bit or 128-bit number is used for the network interface identification. For human-readability, a hostname is assigned to each host device. The device translates the hostname into the corresponding IP address to communicate over the network. The user devices and the wireless mesh routers operate as clients and servers, respectively, and the latter provides service to the former through a dynamic host configuration protocol (DHCP) and the domain name system (DNS) [34, 35]. It is the DHCP server that dynamically assigns the IP addresses to the DHCP clients. The user devices connected to each mesh node form part of a single access network, and they are assigned IP addresses that identify the associated private subnet. The DNS server resolves the mapping of hostnames with the corresponding IP addresses and vice-versa. Dnsmasq is a userspace daemon in a Linux host and provides the DHCP and DNS services [36] to user devices. Dnsmasq is easy to use, lightweight, and hence it is suitable for small embedded devices with low processing power. DNS queries are resolved using rules in the configuration file, and they are cached in order to improve DNS lookup performance.

4.1.3 OLSR and OpenFlow

This subsection explains how the hybrid wireless mesh router works within the mesh infrastructure. The wireless mesh routers need to communicate with the SDN controller to provide highly available services to the users' traffic, such as load balancing. This hybrid requires the support of both the router's wireless mesh protocol and the SDN protocol. The wmSDN tool uses OLSR and OpenFlow as the corresponding protocols [31, 12, 37]. OLSR is an improved version of link state routing, and the main advantage of using OLSR is that all the nodes present in a mesh network are "aware" of every other node, gathering information by broadcasting topology discovery packets. OLSR is suitable for dense mobile ad-hoc networks [31].

The mesh nodes use the in-band network; in other words, both control and data traffic use the same SSID. Traffic consisting of the routing information is referred to as control traffic and is routed among the mesh nodes or between the mesh nodes and the SDN controller. The traffic between user devices and the backbone network is referred to as data traffic. OLSR is used to gather topology information from the control network, and OpenFlow uses this information to communicate with the SDN controller. OpenFlow is configured in the mesh node using Open vSwitch (OvS) [20]. OvS is a production quality, open-source virtual switch. Once implemented, OvS creates an OpenFlow table in the wireless mesh router, and OvS downloads the routing logic for the data traffic from the SDN controller. OvS applies the new routing logic from the controller to the corresponding flow. In SDN, a flow is a series of packets having the same source and destination hosts [37]. If the SDN controller fails, OLSR routes the data traffic using IP-based routing. A subset of mesh nodes connects to the backbone network or to the Internet using Ethernet or Wi-Fi. These nodes are responsible for routing traffic from the mesh network to and from the Internet. Hence, these nodes are gateway nodes. The local mesh infrastructure consists of multiple gateway nodes to provide fault-tolerant Internet service.

4.2 SDN controller

This section describes the SDN controller, which manages wireless mesh network traffic. In this system, the SDN controller is a POX controller [38], which is an open-source SDN framework developed in Python that is recognized for its ease of use in research and education. Figure 5 shows the modules present inside the SDN controller.

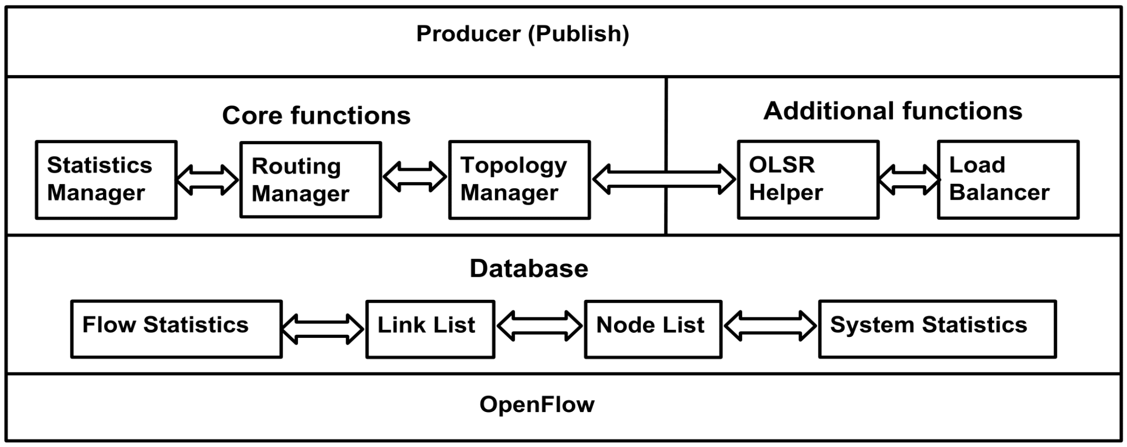


Figure 5: Components of the SDN Controller.

4.2.1 Database

The SDN controller makes routing decisions based on network statistics in its database, statistics that are essential to accurately characterize the network. The POX controller stores network statistics using inbuilt data structures, such as lists and dictionaries. The SDN controller collects network statistics in three main areas: flow, links, and nodes. The network state is primarily collected using the push-based and pull-based methods. In the push-based mode, the OvS sends or pushes flow statistics in the form of notifications to the SDN controller. A message is sent whenever a new event occurs, for example, when a new flow is added or deleted in the OvS. Flow statistics include packets, bytes, or duration of a corresponding flow in the switch. In the pull-based method, the SDN controller makes state requests in response to which the OvS pulls node and the link lists [17]. OLSR daemon collects the link and node information as part of its topology discovery. The database also stores the generated OpenFlow rules in the rule list.

4.2.2 Topology manager and OLSR helper

Topology discovery is one of the most fundamental roles of an SDN controller, providing real-time visibility of the entire network. In wired networks, an SDN controller predominantly uses the OpenFlow discovery protocol (OFDP), utilizing link layer discovery protocol (LLDP) packets to discover the topology [37]. OFDP and LLDP cannot be used in wireless mesh networks as they are multi-hop networks. Instead, a link-state routing protocol such as OLSR gathers the topology information, and the controller periodically queries the OLSR topology database in the mesh routers to gather the node list in its database. The OLSR helper module is used by the SDN topology manager to translate OLSR topology into OpenFlow-based topologies. The OLSR helper module is also used by the SDN routing manager to generate OpenFlow rules.

4.2.3 Routing manager

Whenever there is a new flow in the OvS from a user device, the OvS sends the first packet to the SDN controller to fetch the corresponding routing policy. The packet flow finds its way to the controller through the mesh network with the help of the default path set in the routing table of the router. The SDN controller decodes and analyzes the packet to check for matching rules in the rule list. The rules will provide the shortest path for the packet to flow through the mesh network. If there are rules present, then the rules are applied to the packet flow and are pushed to the corresponding OvS. If there are no rules, then new OpenFlow rules are created. The OLSR helper module is used for the creation of the OpenFlow rules from the OLSR topology. The routing manager module receives the source and destination addresses of the packet flow and uses the Dijkstra's algorithm [39] to calculate the shortest path between the two points in the OLSR topology. Dijkstra's algorithm is implemented using the Python-based NetworkX library developed by Hagberg, Swart and Chult [40]. The selected route is then passed to the OLSR helper module to generate OpenFlow rules. The helper module contains the match criteria and the actions to be taken for the subsequent packet flow. Once rules are created, they are stored in the rule list and then pushed to the OvS.

4.2.4 Load balancer

To improve mesh network utilization and provide a Quality-of-Service (QoS) guarantee to users, the SDN controller implements load balancing. Depending on the number of gateways present in a mesh network, there may be traffic congestion in gateway nodes. The SDN controller implements load balancing to select the gateway node to be used for a particular packet flow. This service avoids data layer traffic congestion. One of the simplest ways to distribute incoming requests from the mesh routers is to use the round-robin algorithm [41]. The Python-based NetworkX library is used to program load-balancing logic [40]. Every time a new packet-in message reaches the controller, this module selects the next in line gateway node and assigns it to the packet flow. Then, the packet flow is sent to the routing manager for data traffic path selection.

4.2.5 Producer (publish)

The SDN controller periodically updates the status of mesh nodes to the CMS for network monitoring. The producer module continually monitors the OLSR topology list and looks for updates. A list of failed nodes from the SDN controller is sent to the central database through the messaging infrastructure. The producer also collects system statistics from the top, a Linux-based process management system. It is used to gather CPU, memory, and disk usages of the controller [42]. A tunnel is established between the SDN controller and the CMS using the publish or subscribe messaging system based on message queuing telemetry transport (MQTT) [43]. This messaging system is programmed using the Python-based PubNub library [44]. The SDN controller acts as a producer of messages, and the CMS serves as a consumer (subscriber).

4.3 Unmanned-vehicle depot

The purpose of deploying the unmanned vehicles is to provide a backup mesh connection to recover a failed network. These unmanned vehicles are mobile systems, such as UAVs and self-driving cars. Aerial mobile systems are fast and efficient in retrieving communications. How unmanned vehicles operate is beyond the scope of this paper. A depot of standby unmanned vehicles is present for deployment. Based on the failed mesh zone, the CMS uploads the wireless mesh configuration files onto the router attached to the vehicle. The configuration of the wireless access point, the DHCP, the DNS, and the routing services are the same for all the wireless mesh routers. The only difference is in the IP address assigned to the routers. The CMS, described in the next section, calculates the corresponding IP address using the wireless mesh zone details from its database. It also estimates the destination where the vehicle is to be deployed A control application is required to set the destination coordinates for a given unmanned vehicle and could be integrated as part of the CMS to start or stop the vehicle. In case the unmanned vehicle is a UAV, a separate application is required to control the flight. Once the vehicle reaches the destination, it will scan for nearby mesh nodes and join the mesh network. An example scenario deploying an unmanned vehicle is described in Figure 6. The figure shows a failed mesh node in zone 3, at which point a notification is sent to the central database indicating that there is a network partition in its local network.

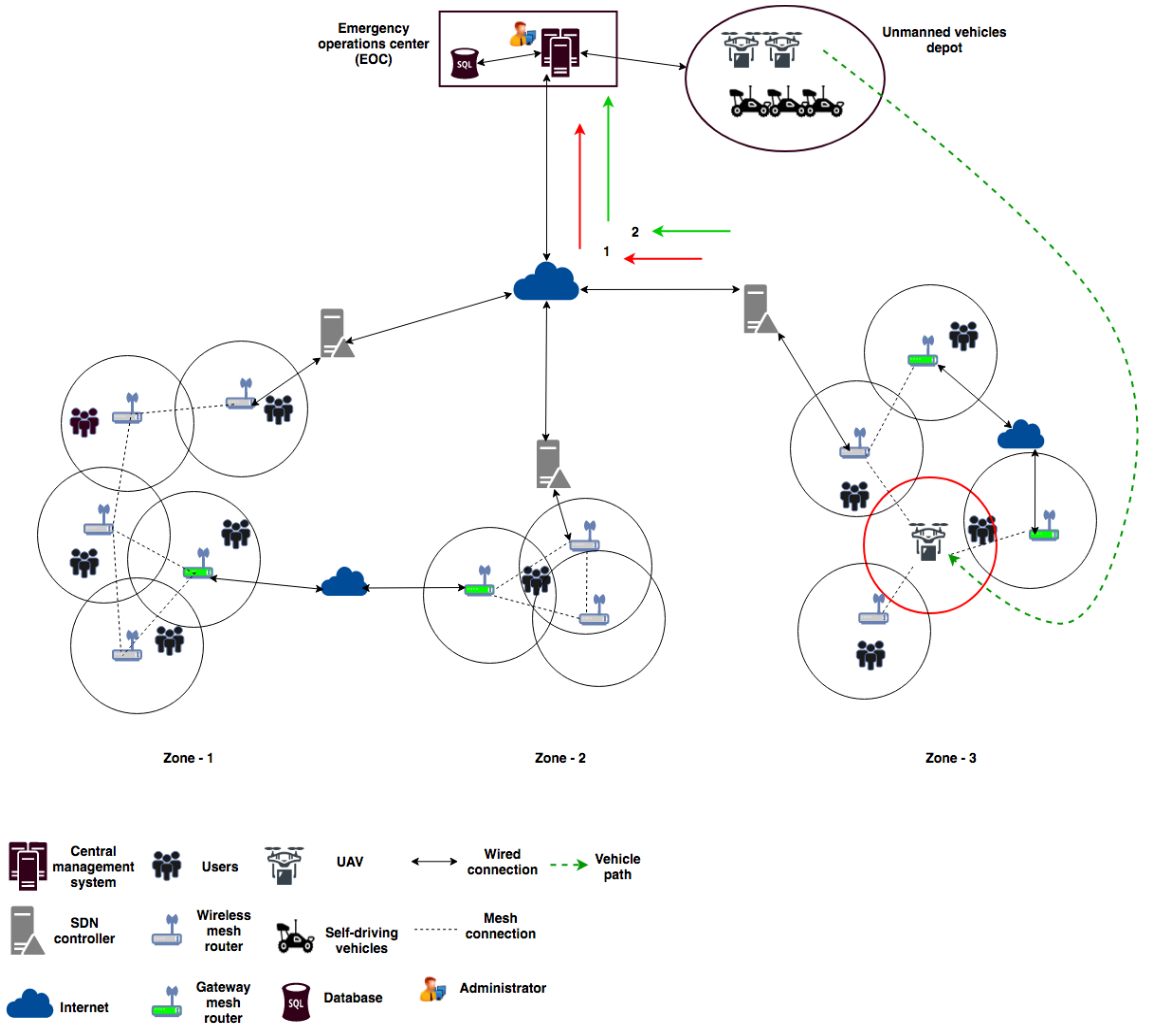


Figure 6: EOC deploying a vehicle in response to a notification from SDN controller.

The CMS estimates that the node nearest to the local controller needs to be replaced to recover the network. An unmanned vehicle is deployed to the location, and the router carried by the drone joins the mesh network.

4.4 Central management system (CMS)

This section describes the design of a central system to continuously monitor relevant data from the entire network and update operations personnel. This task is carried out by the CMS. The CMS is a northbound application that is connected virtually to all SDN controllers. The CMS is deployed in the cloud and is therefore readily accessible from anywhere on the Internet. It requires a central database to store incoming structured data from local networks to track the status of the devices. The administrator has access to two types of monitoring frameworks giving both network and location information. Figure 7 shows the components of the central management system.

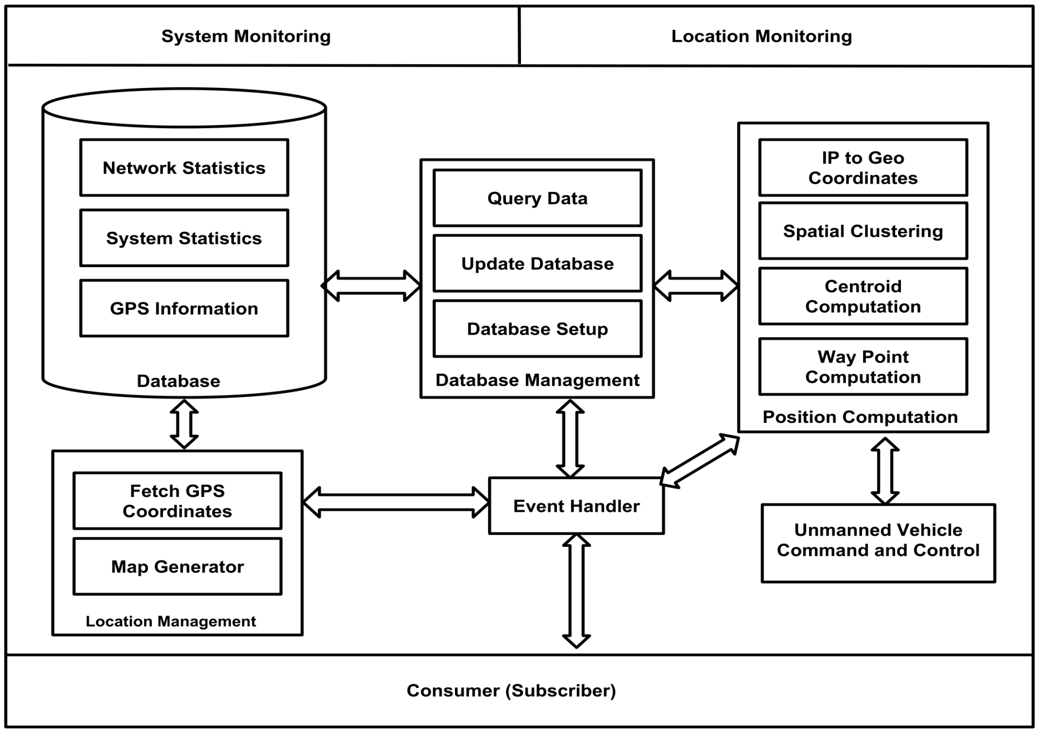
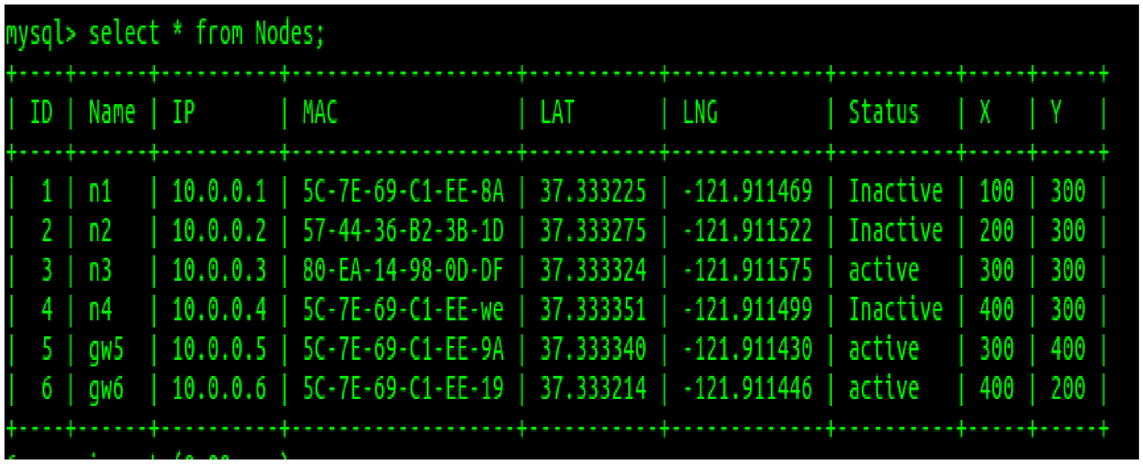


Figure 7: Components of the CMS.

4.4.1 Database and database management

Management of an efficient catalog of available resources is required to perform any productive work. The effectiveness of the work depends on the simplicity of the information storage. The database should be launched and accessible in minutes. Furthermore, the database should be secure, available, and reliable. These reasons beg for the usage of a centralized relational database which is suitable for storing structured data from the SDN controller. MySQL offers excellent performance for storing structured data and can be deployed using cloud services, such as AWS or Google Cloud Platform [45, 46]. The database stores details like network statistics, system statistics, and GPS information. Network and system statistics are updated continuously from the SDN controller. GPS information includes the latitude and longitude values of each device. Since the status of all routers from every local network must be stored, the database will allocate one table for each local network. Three submodules are designed using the Python-based MySQL-connector library to manage the database[47]. The database-setup submodule creates the database and the tables for corresponding local networks. Once the network is online, the update-database submodule populates the table with the data. System and location monitoring continually looks for changes in the database using the query-data submodule. Figure 8 shows a table in the MySQL database storing the statistics of a local network.

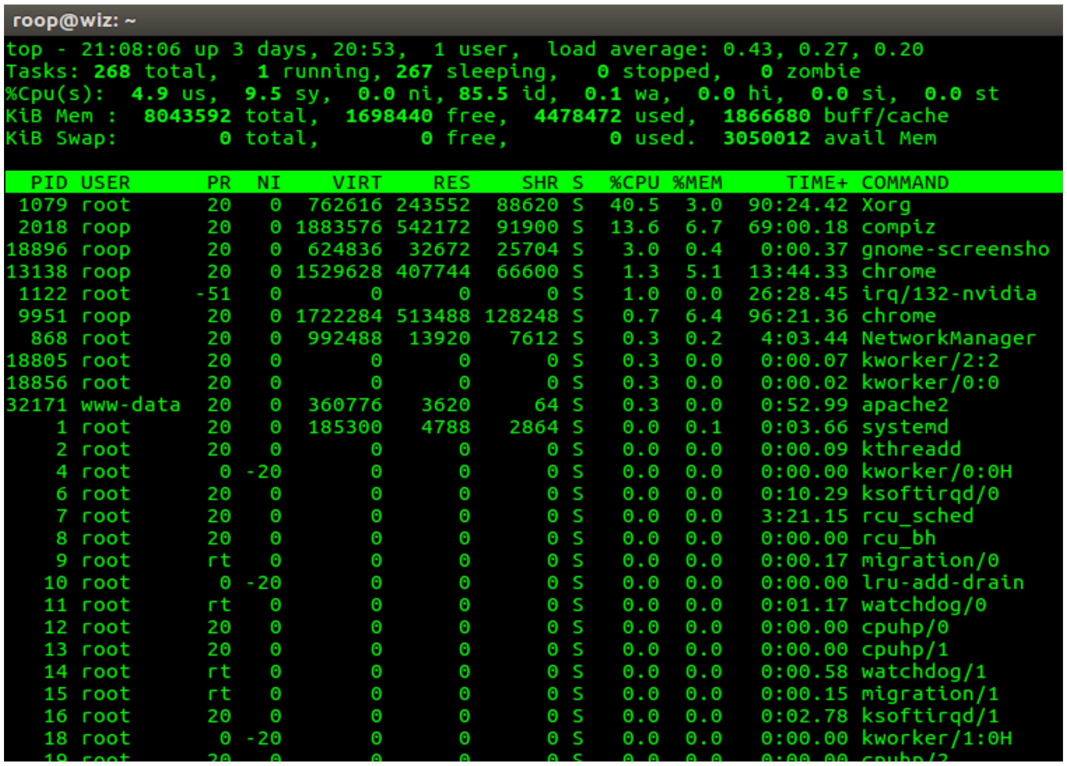


```
mysql> select * from Nodes;
+----+------+----------+-------------------+-----------+-------------+----------+-----+-----+
| ID | Name | IP       | MAC               | LAT       | LNG         | Status   | X   | Y   |
+----+------+----------+-------------------+-----------+-------------+----------+-----+-----+
|  1 | n1   | 10.0.0.1 | 5C-7E-69-C1-EE-8A | 37.333225 | -121.911469 | Inactive | 100 | 300 |
|  2 | n2   | 10.0.0.2 | 57-44-36-B2-3B-1D | 37.333275 | -121.911522 | Inactive | 200 | 300 |
|  3 | n3   | 10.0.0.3 | 80-EA-14-98-0D-DF | 37.333324 | -121.911575 | active   | 300 | 300 |
|  4 | n4   | 10.0.0.4 | 5C-7E-69-C1-EE-we | 37.333351 | -121.911499 | Inactive | 400 | 300 |
|  5 | gw5  | 10.0.0.5 | 5C-7E-69-C1-EE-9A | 37.333340 | -121.911430 | active   | 300 | 400 |
|  6 | gw6  | 10.0.0.6 | 5C-7E-69-C1-EE-19 | 37.333214 | -121.911446 | active   | 400 | 200 |
+----+------+----------+-------------------+-----------+-------------+----------+-----+-----+
```

Figure 8: Network statistics stored in a table using MySQL.

4.4.2 System monitoring

This module is used to display the status of the SDN controller of a given network. The update process is done over the publishing or subscribing messaging system. As described previously, the top is used to gather controller system statistics [42]. Network management protocols like the simple network management protocol (SNMP) and network configuration protocol (NETCONF) could be used to set up a monitoring system [48, 49]. SNMP and NETCONF run on the master-agent paradigm where the master is notified of updates on agents. When the setup is complete, these protocols provide device statuses, such as uptime, CPU, memory and disk usages. The rules defined within the configuration files of the network management protocols could be applied to all IP-enabled devices like switches, routers, and servers. Open source network management software like Nagios and OpenNMS provide APIs for developers to integrate into the system to take advantage of ready-to-use network monitoring functionality [50, 51]. Figure 9 shows the sample image of the top tool executing in the controller.



```
roop@wiz: ~
top - 21:08:06 up 3 days, 20:53,  1 user,  load average: 0.43, 0.27, 0.20
Tasks: 268 total,   1 running, 267 sleeping,   0 stopped,   0 zombie
%Cpu(s):  4.9 us,  9.5 sy,  0.0 ni, 85.5 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  8043592 total,  1698440 free,  4478472 used,  1866680 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  3050012 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1079 root      20   0  762616 243552  88620 S  40.5  3.0  90:24.42 Xorg
 2018 roop      20   0 1883576 542172  91900 S  13.6  6.7  69:00.18 compiz
18896 roop      20   0  624836  32672  25704 S   3.0  0.4   0:00.37 gnome-screensho
13138 roop      20   0 1529628 407744  66600 S   1.3  5.1  13:44.33 chrome
 1122 root     -51   0       0      0      0 S   1.0  0.0  26:28.45 irq/132-nvidia
 9951 roop      20   0 1722284 513488 128248 S   0.7  6.4  96:21.36 chrome
  868 root      20   0  992488  13920   7612 S   0.3  0.2   4:03.44 NetworkManager
18805 root      20   0       0      0      0 S   0.3  0.0   0:00.07 kworker/2:2
18856 root      20   0       0      0      0 S   0.3  0.0   0:00.02 kworker/0:0
32171 www-data  20   0  360776   3620     64 S   0.3  0.0   0:52.99 apache2
    1 root      20   0  185300   4788   2864 S   0.0  0.1   0:03.66 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.09 kthreadd
    4 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    6 root      20   0       0      0      0 S   0.0  0.0   0:10.29 ksoftirqd/0
    7 root      20   0       0      0      0 S   0.0  0.0   3:21.15 rcu_sched
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
    9 root      rt   0       0      0      0 S   0.0  0.0   0:00.17 migration/0
   10 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 lru-add-drain
   11 root      rt   0       0      0      0 S   0.0  0.0   0:01.17 watchdog/0
   12 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/0
   13 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/1
   14 root      rt   0       0      0      0 S   0.0  0.0   0:00.58 watchdog/1
   15 root      rt   0       0      0      0 S   0.0  0.0   0:00.15 migration/1
   16 root      20   0       0      0      0 S   0.0  0.0   0:02.78 ksoftirqd/1
   18 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/1:0H
   19 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/2
```

Figure 9: An image of the top tool in execution.

4.4.3 Location monitoring and management

This module is used to display the location of all wireless mesh routers in real-time. Gathering location data and monitoring the data using a standard web map is essential to coordinate response teams and allocate networking resources during a disaster. Location monitoring allows the administrator to view the location of the routers using a map visualization application. The map visualization module is designed using a combination of HTML, CSS, and javascript programming. The Google maps javascript API is used to initialize the map and plot the locations dynamically, while the incoming data is handled using the PubNub subscriber API [52, 44]. The Google maps geolocation API is used to resolve the marking of exact positions on the map using the concept of reverse geocoding [52]. The mesh nodes, the controllers, and the unmanned vehicles are plotted using markers. Mapbox [53], an open-source map visualization platform also provides APIs to build customized maps in real time. The real-time map is updated from three pipelines or topics. A topic is a partition in the publish/subscribe messaging system established using the PubNub API [44]. Wireless mesh routers use the first topic to update their locations during initial boot-up. The update-map submodule present in the CMS uses the second topic to update the map of the failed nodes from the database. The unmanned vehicles use the third topic when they are deployed to effectuate network recovery.

4.4.4 Consumer (subscriber) and event handler

An event subscriber or the consumer is the receiver end of a publish/subscribe messaging system that listens for events and passes the control to a callback function. An event handler is a module that helps the subscriber to sort the messages and call the corresponding functions. The event subscriber module is implemented using the PubNub python API [44]. As described in the previous subsection, the CMS subscribes to three topics. Depending on the channel in which the data arrives, the event handler passes the data to the corresponding module. For instance, let us assume that the mesh nodes send their location data as the topic "meshnodes." Then the event subscriber subscribes to this topic in the PubNub cloud server, which acts as a message broker. Once the event subscriber receives the location data, it is sent to the database management module to store in the database and to the location management module for plotting them on a real-time map.

4.4.5 Position computation

During an emergency scenario, it is highly likely that a subset of mesh nodes will fail due to a power outage in a particular zone or physical damage to the mesh node. Once the nodes are down, the SDN controllers update their status to the CMS. The failed mesh nodes result in loss of Internet access to a region, and a replacement node is needed in order to reestablish access. One solution would be to use unmanned vehicles to send working nodes to replace the failed nodes. This is not optimal for a WCN because the more failed mesh nodes, the more vehicles are needed. The position computation module in the proposed CMS is responsible for calculating where to deploy the unmanned vehicle. It is triggered in two circumstances. The first is when only one mesh node fails in a zone. As described in figure 6, the CMS notifies the control center to deploy a vehicle to the location of the failed node. The second is triggered when more than one mesh node fails in a region. Now control is passed to the core functionality of the module, that is, to the GPS-position finder (GPS-PF). GPS-PF uses spatial clustering as a preprocessing step.

Spatial clustering is the process of grouping spatial objects having similar characteristics [54]. In this case, the spatial objects are the wireless mesh nodes. In a wireless community network, nodes may be placed somewhat randomly. Hence, a clustering method is needed that can form a cluster from a group of randomly placed nodes. The four main categories of spatial clustering are as follows: partitioning, hierarchical, density-based, and grid-based. Density-based clustering is most suitable for forming clusters of arbitrary shapes within regions with a higher density of nodes separated by regions with a lower density of nodes [54]. The algorithm must be able to accept the range and the number of wireless mesh nodes in a region as input parameters from the user. This is precisely what the DBSCAN (density-based spatial clustering of applications with noise) algorithm can accomplish [55].

The DBSCAN algorithm is programmed using the Python-based API from the scikit-learn library [56]. It takes a series of parameters as inputs, such as ϵ, min, *algorithm* and *metric*. ϵ represents the maximum distance that the nodes can be from each other to be part of a cluster; in other words, ϵ defines the cluster neighborhood. ϵ is calculated using the formula c/R, where c is a constant that translates the range of the wireless mesh nodes into kilometers and R is the radius of the earth in kilometers. *min* represents the number of nodes or the total weight of a cluster. The ball-tree algorithm is given as an input into DBSCAN to analyze the neighborhood and discover other nodes [57]. The user also provides the metric to be used to calculate the distance. In this case, the metric is haversine distance [58]. The notation λ represents the maximum range of the wireless mesh nodes and F_{i} represents the list of failed nodes from the SDN controller. Algorithm 1 illustrates the procedure for computing position using the GPS-PF algorithm.

Algorithm 1 Algorithm for GPS-Position Finder (GPS-PF).

```
INPUT - F_i                                                      ▷ IP addresses of failed nodes

Listen($F_i$)
for $F_i$ do
    $G_i \Leftarrow Get\_Location\_From\_IP(F_i)$               ▷ GPS coordinates of failed nodes
    Form_Cluster($G_i$)
    for $G_i$ do
        $Cluster\_List \Leftarrow DBSCAN(\epsilon, min, algorithm, metric)$
    $C_i \Leftarrow Get\_Centroid(Cluster\_List)$                      ▷ Centroid of the cluster
    $A_i \Leftarrow Get\_Active\_Nodes()$                     ▷ GPS coordinates of active nodes
    for centroid in $C_i$ do
        $N_c \Leftarrow Find\_Minimum$(haversine($A_i$, centroid))   ▷ Nearest active node to the centroid
    Form_Pair($C_i, N_c$)
    for pair in ($C_i, N_c$) do
        $D_i \Leftarrow haversine(C_i, N_c)$              ▷ Distance between two GPS coordinates
        if $D_i \leq \lambda$ then
            send_UAV($D_i$)
        while $D_i > \lambda$ do
            $I_m \Leftarrow Find\_Intermediate\_Node(C_i, N_c, f)$          ▷ Intermediate GPS coordinate
            send_UAV($I_m$)
            $D_i \Leftarrow haversine(I_m, C_i)$
```

To effectively describe how the module works, let us assume a two-dimensional mesh grid in which each intersection represents a mesh node. In Figure 10, there are 25 mesh nodes present in a mesh zone.

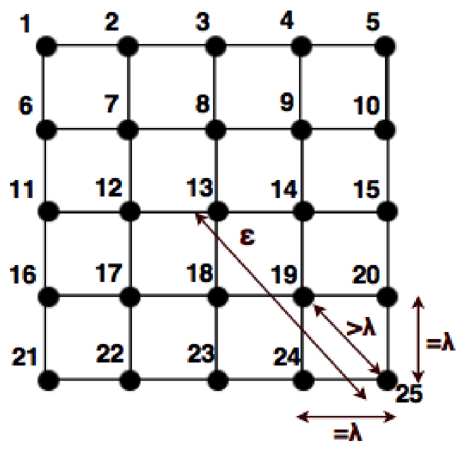


Figure 10: 2-D Mesh grid representing 25 nodes.

In this example, λ is the length of one side of the square and ϵ is the diagonal of the square times two. Figure 11 shows the nodes 1 and 25 are down for unknown reasons.

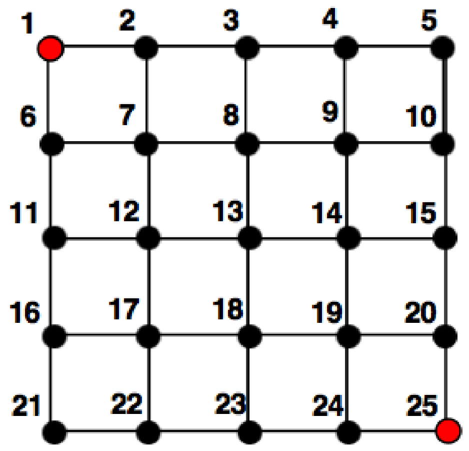


Figure 11: 2-D Mesh grid representing the failure of nodes 1 and 25.

Since the distance between nodes 1 and 25 is higher than ϵ, after applying the clustering algorithm, there will be two clusters formed. This case is the same as the one previously described where a working mesh node replaces each failed mesh node. It is important to note that a majority of mesh nodes rely on a single up-link node for connection. Hence, if a failure occurs in the up-link node, then all nodes that rely on the up-link node will lose access to the Internet. In this scenario, these secondary nodes will come back online if a backup node replaces the up-link node. Figure 12 illustrates a similar situation where nodes 19, 20, 23, 24 and 25 have failed.

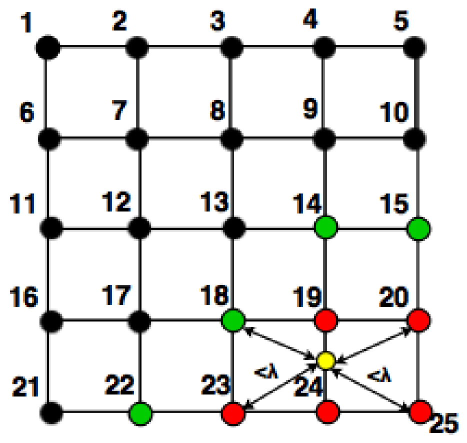


Figure 12: 2-D Mesh grid representing the placement of a replacement node.

Since the distance between each node is less than ϵ, all five nodes will become part of a single cluster. Now, the centroid of the cluster is computed, which is represented in the figure by the yellow dot. We can see that the centroid is within the maximum wireless range λ of the failed nodes and the nearest active node is node 18. When an unmanned vehicle is sent to the centroid position, then the data from the nodes in an outage is relayed to the nearest active node 18. In a situation where the nearest active node is not present within λ distance, intermediate points along the path between the centroid and the nearest active node are calculated with the threshold value as λ. The unmanned vehicle is deployed to all intermediate points to form an ad hoc network leading to the centroid, thus recovering the network.

5 Evaluation

The proposed ECS framework was evaluated using the CORE network emulator with sample network topologies [9]. The experiments were performed to test the following four features:

1. the feasibility of using unmanned vehicles for recovering communication;
2. the performance of the GPS-PF algorithm;
3. the performance of the SDWMN;
4. the performance of the proposed system in a multi-zone mesh network.

5.1 Feasibility of using unmanned vehicles for recovering communication

A practical experiment was conducted with a UAV to analyze the feasibility of unmanned vehicles to form a relay mesh network. Figure 13 shows the quadcopter that was deployed.

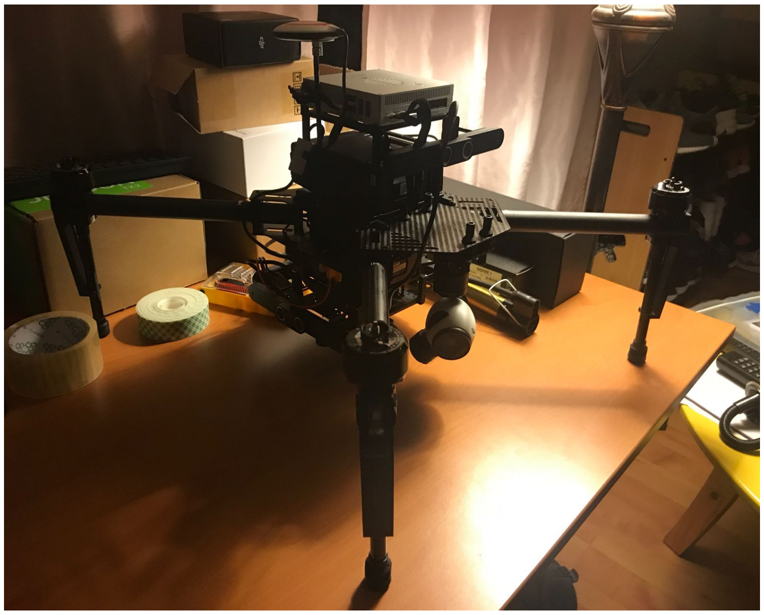


Figure 13: The quadcopter deployed.

In this experiment, the GPS modules were plugged into four Raspberry Pis using a USB to TTL cable [59] to get location information. The Raspberry Pis were pre-configured with the B.A.T.M.A.N. advanced mesh daemon [11], and so they acted as mesh routers. Three of the Raspberry Pis (M1, M2, M3) were kept in range to form a wireless mesh network. The fourth Raspberry Pi (D) was fitted on the quadcopter and was kept out of range. The four Raspberry Pis were configured to be part of the same IP subnet (192.168.1.0/24). In the mesh network, M1 was connected to a laptop to test Internet connectivity using ping, and M2 was connected to the access point created by a mobile phone. A ping with a time interval of one second was started from the client connected to M1 to the Google domain name server (8.8.8.8) and the round-trip time (RTT) was recorded. The RTT was the time taken by the packets to reach the destination and bounce back to the source. Initially, the RTT of the ping packets was around 25 milliseconds. Figure 14 shows the setup of the WMN in the real-time map.

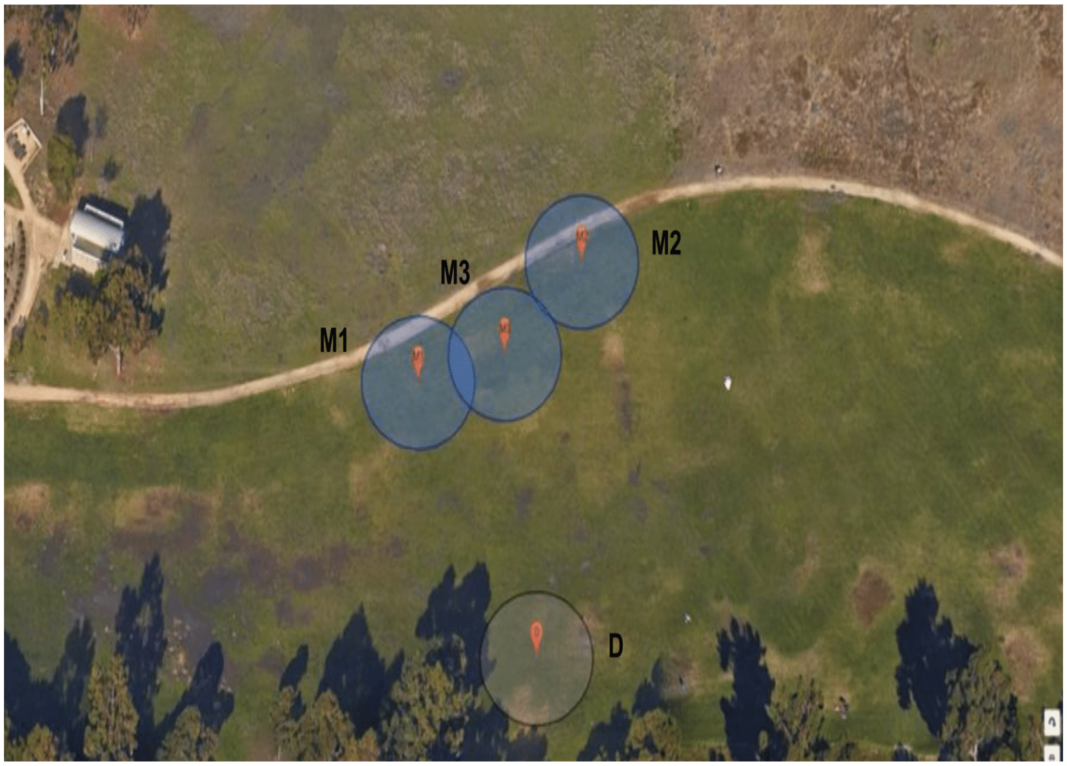


Figure 14: Real-time map visualization of the mesh nodes with the UAV.

At second 27, the middle node M3 was made to fail by turning both of the Wi-Fi interfaces down. Now, the ping packets from the client reached router M1 and got queued in the routers' buffer. If the ICMP packets reached their timeout value of one second, then they were dropped by M1. This sent a ‘‘network unreachable’’ message to the client. The real-time status monitoring noticed the failure of M3 and updated the map. As soon as the failed mesh node turned red in the map visualization, the UAV was sent to the location of M3. Once the UAV reached the destination, it was made to hover at a height of approximately 25 feet as the range of the onboard Wi-Fi adapter was measured to be strong within 30 feet. The UAV came with a pre-configured GPS which was used to get the current location. The ping revived when a temporary network was formed for the ping packets from M1 to M2 through the mesh node (D) on UAV. Figure 15 shows the deployment of the UAV in real-time.

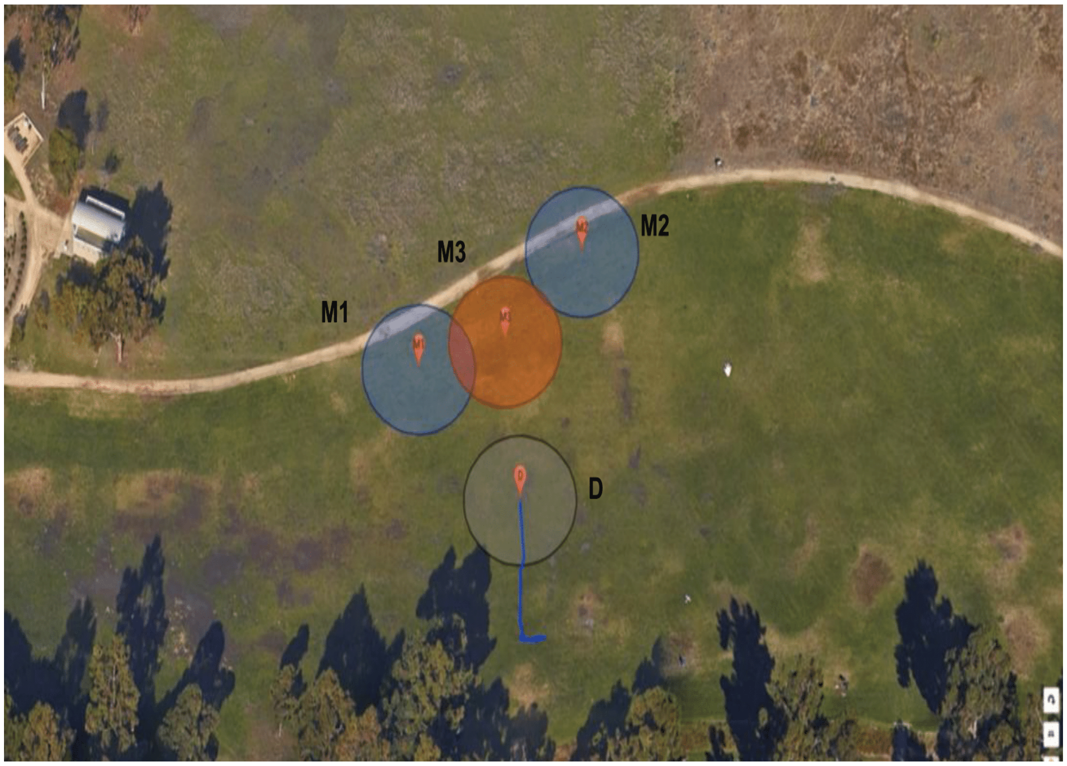


Figure 15: Real-time map visualization of the deployment of the UAV.

The round-trip time of the ping packets was measured and plotted against the ping sequence number. Figure 16 shows the results obtained for the network testing using ping.

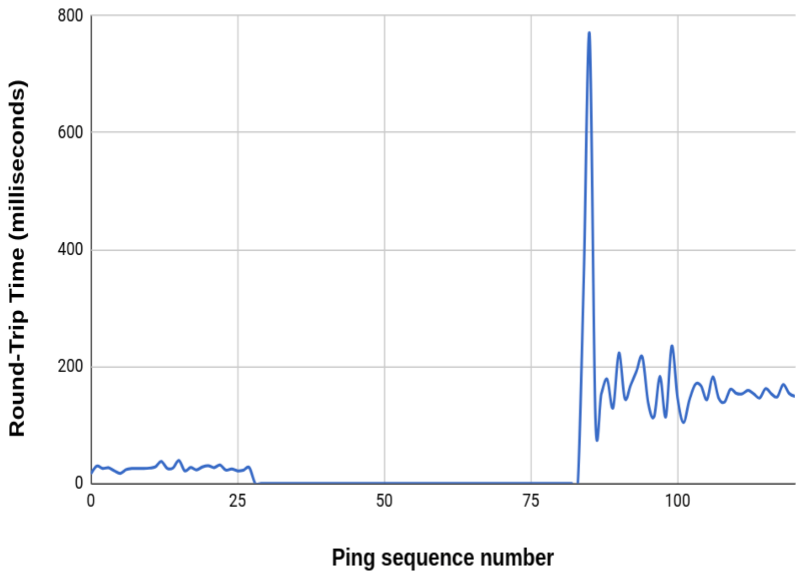
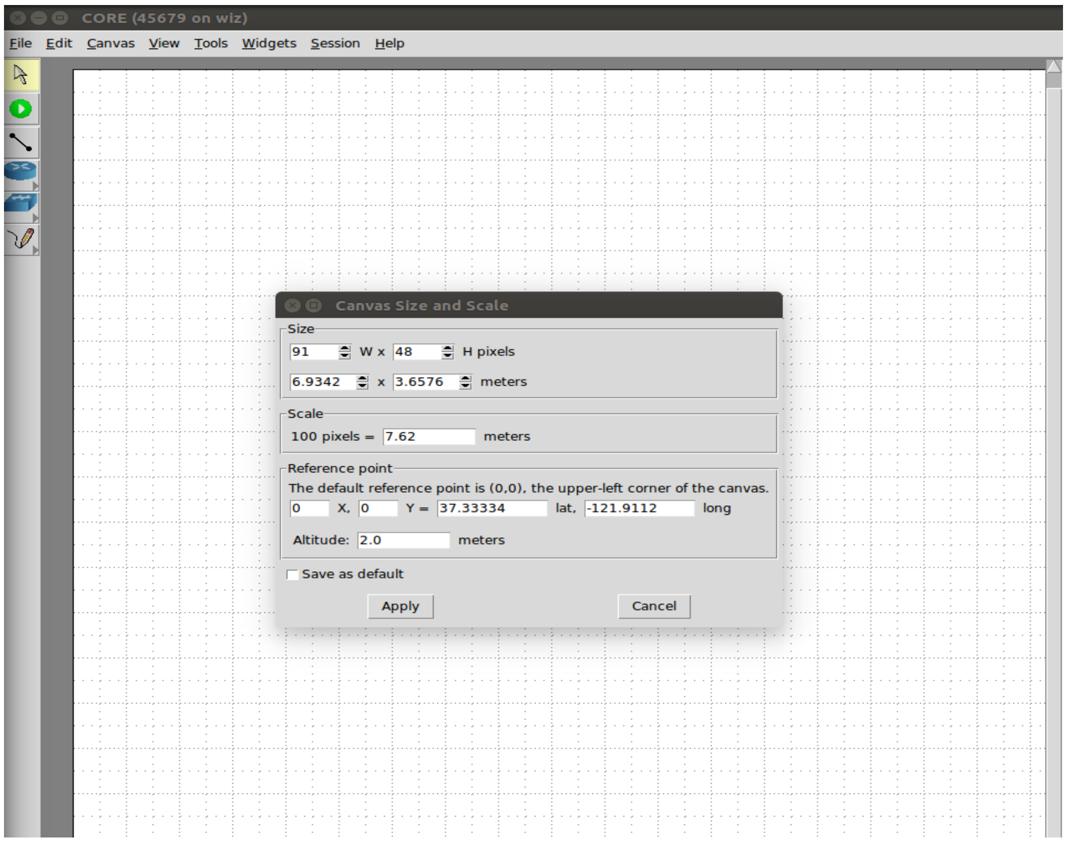


Figure 16: Round-Trip time for the ICMP packets during recovering the network.

In Figure 16, the ping packets from 28 to 83 were dropped right after the network disconnection between M1 and M2. At ping sequence 84, the network was revived via the node on the UAV, and there was a considerable delay in the RTT for sequence number 85. This delay was due to buffering of ping sequence 85 in router M1 right before the network recovery. Hence, it can be deduced from this experiment that it is feasible to use an unmanned vehicle for recovering the network.

5.2 Performance of the GPS-PF algorithm

The second experiment was done to evaluate the position computation module present in the CMS. The CORE network emulator provides users with the option of mapping the Cartesian coordinates (X, Y) with the geographic coordinates (latitude, longitude) [9]. The range of the wireless devices was calculated based on the pixel distance between the devices placed on the canvas. This functionality of mapping the two-dimensional to the three-dimensional system allowed users to simulate the test environment close to the practical test bed. The canvas size was adjusted to 1800 pixels in width and 950 pixels in height. The corresponding proportion in meters was approximately 137 meters in width and 72 meters in height. Hence the canvas area was about 9,900 meters. It is also worth noting that there were 703 mesh boxes arranged in a grid. Therefore, each mesh square had a side of 50 pixels or 3.81 meters in length. Figure 17 illustrates the two-dimensional to the three-dimensional mapping of the CORE tool.



Size
91 W x 48 H pixels
6.9342 x 3.6576 meters

Scale
100 pixels = 7.62 meters

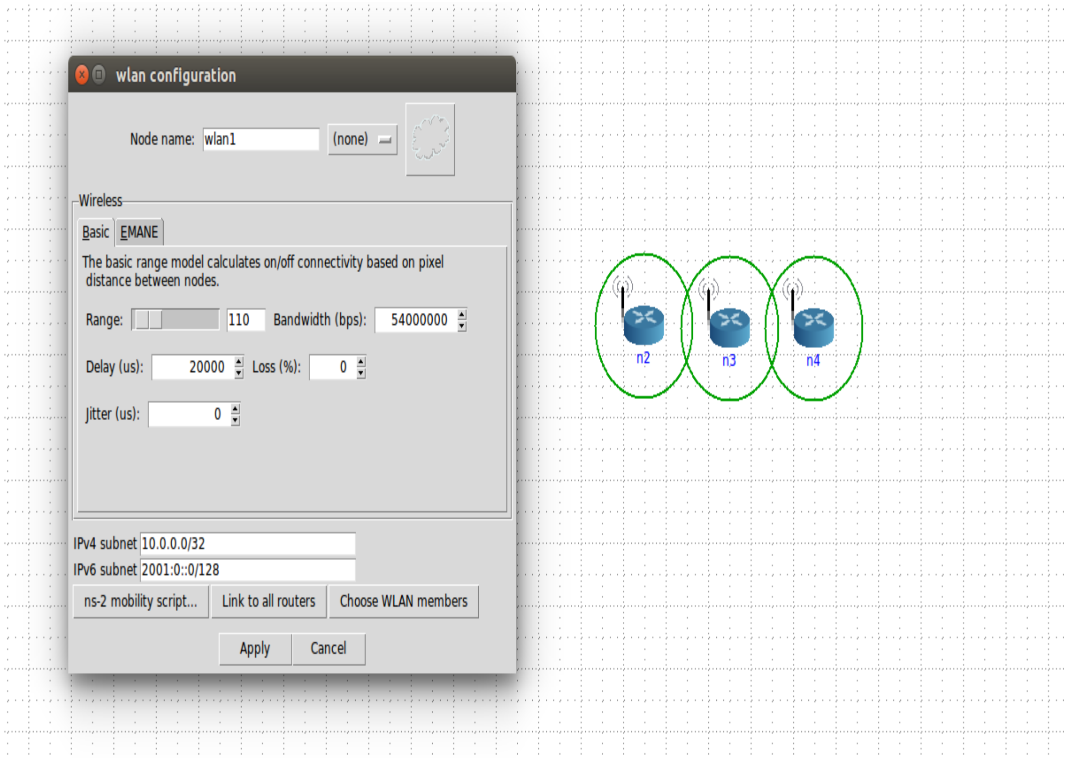
Reference point
The default reference point is (0,0), the upper-left corner of the canvas.
0 X, 0 Y = 37.33334 lat, -121.9112 long

Altitude: 2.0 meters

☐ Save as default

Figure 17: Customizing the canvas in the CORE network emulator.

The default reference point, (0,0), was set in the upper left corner of the canvas to a geographic coordinate of choice. This setting was done to match the geographic coordinates of all the mesh nodes present on the canvas to the Cartesian coordinates. The scale of the canvas was adjusted to 7.62 meters per 100 pixels which equaled 25 feet in length. The range of every wireless node needed to be within the range of at least one other node to form the mesh network. Therefore, the range of the individual mesh node was set to 110 pixels or 30 feet, which was a little more than the sides of two mesh boxes. Since there was a Wi-Fi wireless link, the bandwidth of the wireless link was adjusted to 54 Megabits per second. Finally, the nodes were linked to the “wlan1” to form part of the same local area network. Figure 18 shows the customization made to the canvas to adjust the range of the nodes.


wlan configuration

Node name: wlan1 (none)

Wireless

Basic EMANE

The basic range model calculates on/off connectivity based on pixel distance between nodes.

Range: 110 Bandwidth (bps): 54000000

Delay (us): 20000 Loss (%): 0

Jitter (us): 0

IPv4 subnet 10.0.0.0/32

IPv6 subnet 2001:0::0/128

Figure 18: Customizing the range of the wireless mesh nodes in the CORE network emulator.

The next step was to generate a random topology of one hundred nodes having wireless mesh functionality. In the node configuration, OLSR was enabled as the mesh protocol of choice. The random nodes were adjusted to be part of a single mesh network. The session was started to verify that all the mesh nodes were connected to each other, that is, a ping from node one needed to reach node one hundred. Since the testing was done to determine the effectiveness of the GPS-PF algorithm, the SDN controller was not needed. Therefore, the CMS was directly connected to the mesh network. Figure 19 shows the topology tested for the second experiment.

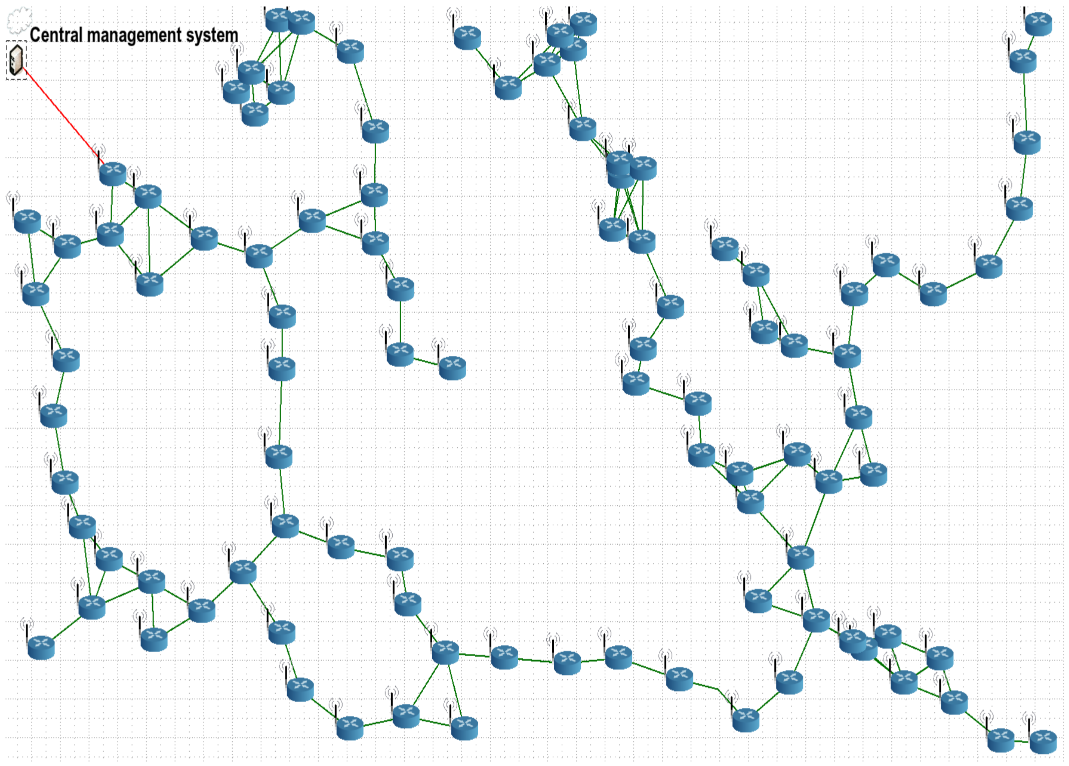


Figure 19: A random topology of 100 mesh nodes placed on the canvas.

Once the simulated location of the nodes was stored in the database, a number of these nodes were made to fail randomly in sets of 5 in order to evaluate the GPS-PF algorithm. The results of the algorithm provided the number of clusters formed, the number of unmanned vehicles required to recover the network, and the destination for the unmanned vehicles. Two graphs were drawn to check the effectiveness of the algorithm. Figure 20 is a graph plotted with the number of clusters formed compared to the number of failed nodes.

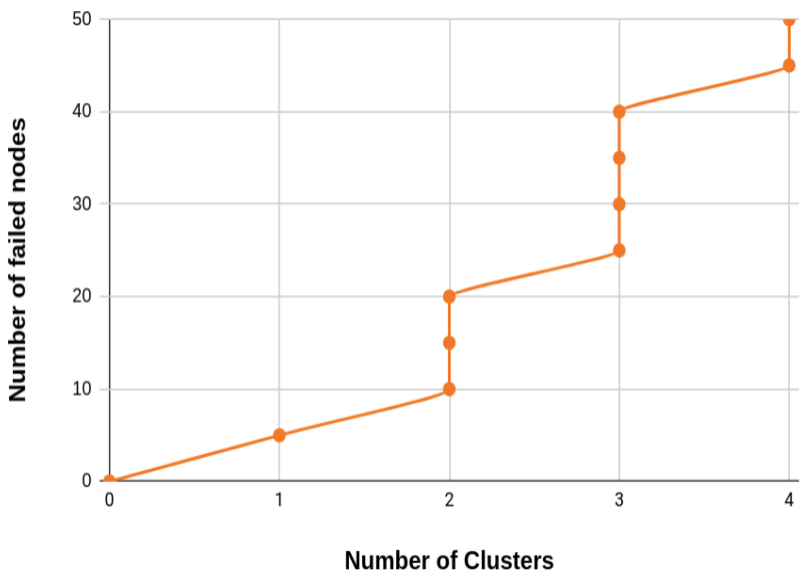


Figure 20: Number of clusters plotted against the number of failed nodes.

It is evident from the graph that the number of clusters formed among the failed nodes progressively increased with the number of failed nodes. It is also worth mentioning that the number of clusters depends on input parameters like the neighborhood distance, minimum number of nodes and the range of each wireless node. While the minimum number of nodes was set to one, the neighborhood distance, ϵ was set to 58.2564. As described in the previous chapter, ϵ was calculated by the formula c/R. The range of the wireless nodes, c, was set to 30 feet which was equivalent to 0.009144 kilometers, and the radius of the earth as 6371.0088 kilometers. Figure 21 is a graph plotted with the number of unmanned vehicles required to recover the network with and without the GPS-PF algorithm and the number of failed nodes.

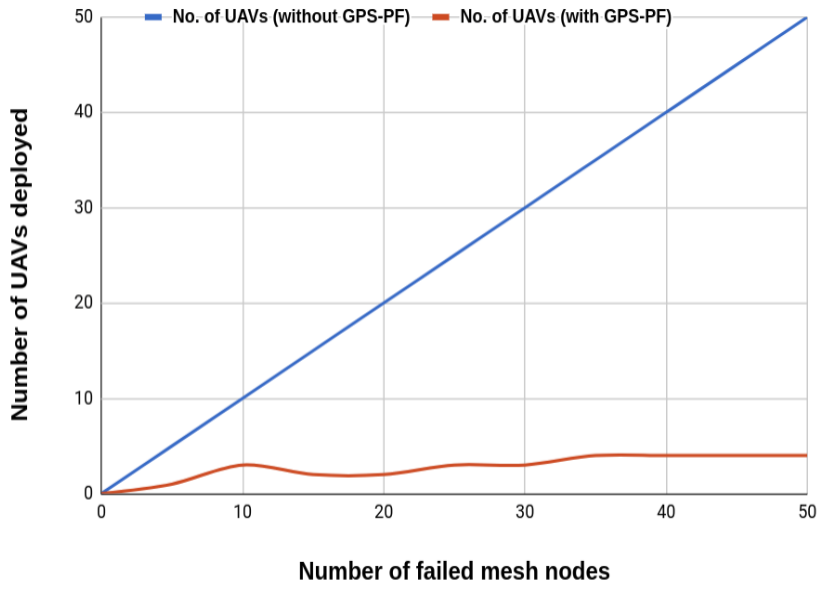


Figure 21: Number of unmanned vehicles to be used calculated with and without the GPS-PF.

It is deducted from the graph that the number of unmanned vehicles to be used is greatly reduced with the application of the GPS-PF algorithm.

5.3 Performance of the SDWMN

This experiment was done to analyze the performance of the SDN controller in a wireless mesh network using the wmSDN tool [31] and the CORE network emulator. For purposes of this analysis, an SDWMN consisting of six mesh nodes, one SDN controller, one server and five clients were considered. Two of the six mesh nodes were configured as gateway nodes, and they were connected to the clients through the hub. The green links in the topology represented wireless mesh links and the red links represented wired Ethernet connections. The server was connected to mesh node n1, and the SDN controller was connected to mesh node n3. Since CORE emulates each node by using lightweight virtual machines, the nodes provided access to the command line interface to run the individual configurations. Figure 22 shows the topology of the SDWMN.

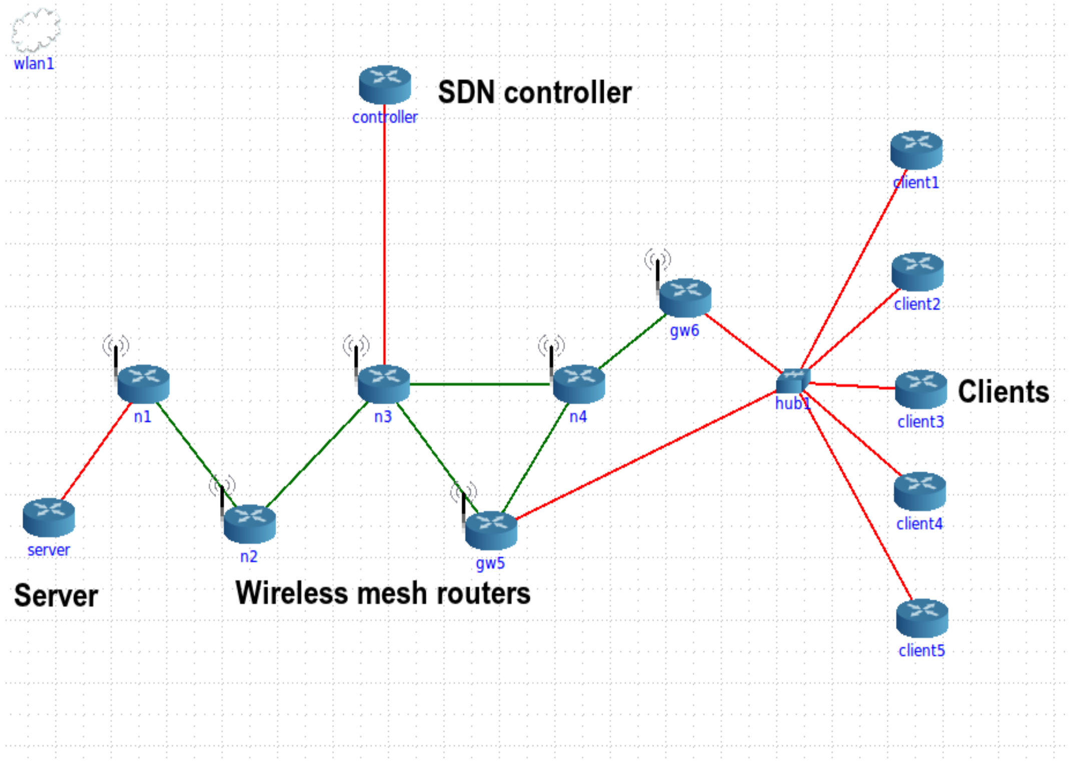


Figure 22: The SDWMN emulated by the CORE network emulator.

Figure 22 shows that there were different subnets for control network and the access network, that is, 10.0.0.0/24 for the control network and 192.168.0.0/24 for the access network. The idea was to analyze the throughput of the network using the Iperf tool with and without the SDN controller. At first, the SDN controller was not started, and the OLSR mesh protocol routed the data packets. The Iperf tool was started between the server and the clients. Here, the server was set up to generate traffic and clients were made to download the traffic. The following commands were used to start the data flow and throughput analysis on the server and the client respectively:

$$iperf - c < client - IP > -t600 - p < port - number >$$

$$iperf - s - i2 - fk - p < port - number >$$

During the first twenty seconds, only the first client was actively pulling data from the server, and the other clients were progressively activated every 20 seconds to do the same. The SDN controller was then started, and the process was repeated. The throughput performance of the first client was measured and plotted against the time in Figure 23.

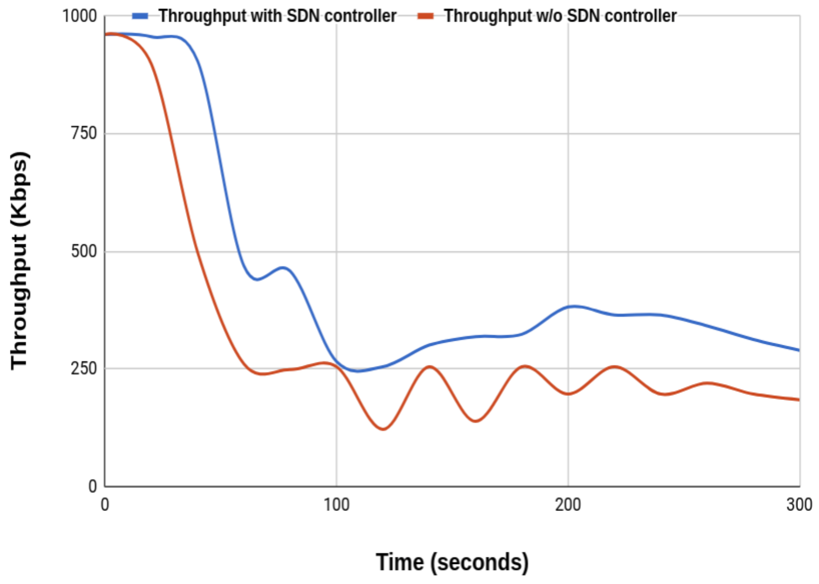


Figure 23: Throughput performance of the network with and without the SDN controller.

It is important to note that in the network without the SDN controller, OLSR chooses the gateway with the shortest path to the server. Therefore, gateway five was chosen as the default gateway for all clients. But once the SDN controller is started, the gateway is chosen based on the round-robin algorithm. As mentioned previously, only the first client was active during the initial twenty seconds, and so the throughput was as high as one megabit per second in both cases. In the subsequent twenty seconds, two clients were active, and for the network without the SDN controller, the throughput was reduced in half. For the network with the SDN controller, gateway six was chosen, and the performance of the network stayed the same. In the next cycle, the third client was assigned to gateway five by the controller, and so on. Hence, it can be deduced from Figure 23 that the mesh network managed by an SDN controller has better throughput than when managed only by the wireless mesh protocol.

5.4 Simulation of a multi-zone emergency mesh network

It is essential to confirm that the entire proposed framework works and so, it was integrated in a multi-zone testbed. To simulate an unmanned vehicle, the functionality of a wireless mesh router was added to the NS-3 mobility model [9]. The CORE tool provided the Python API to implement the NS-3 mobility model and the Wi-Fi model. The illustration of the drone was made with a tiny drone icon. The two mesh zones were part of a different WLAN, that is, they had different IP subnets. The gateways from the zones were connected through a hub to form a network. Since CORE allows users to connect to live networks, the CMS was run from the host system connected to the hub. Once the setup was complete, the session began and checked connectivity. A bash script was executed utilizing the CORE command line interface to progressively turn off the interfaces of one node from each zone. The SDN controller in each zone sent a notification to the CMS running on the host system through the hub. The CMS calculated the destination coordinates and sent to the bash script executing in the background. A UAV was selected from the depot and sent to the destination. This simulation of the UAV deployment can be seen in the Figure 24.

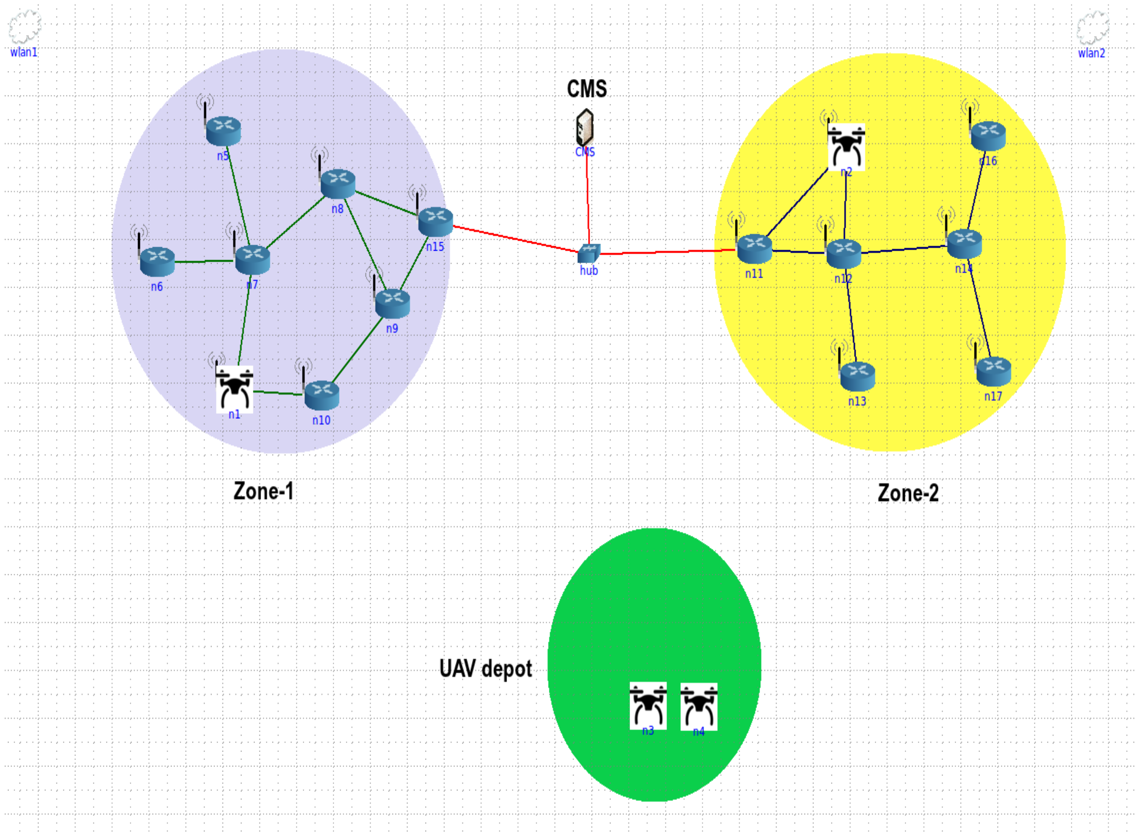


Figure 24: Simulation of UAV deployment to recover the network in a multi-zone mesh infrastructure.

From the above simulation, it can be deduced that the framework can be incorporated into a multi-zone mesh network.

6 Discussion and future work

Even though using unmanned vehicles is a costly approach, vehicles like drones will no doubt become integral to emergency recovery systems. At the same time, designing a new disaster network architecture using drones for recovery of an emergency network zone does present some unique challenges. Some of the questions unanswered in this thesis are:

1. How many mesh nodes are required to cover a specific emergency zone?
2. What other functionalities can be implemented by the mesh nodes?
3. How does one calculate the best location for placing the pre-deployment mesh nodes?
4. What is the maximum distance an unmanned vehicle can cover in an emergency zone?
5. What kind of security challenges are present in a disaster network?

In the future, these problems will be addressed to provide an upgraded version of the current emergency network recovery framework. Work will be carried out to integrate a multi-modal communication edge device that can scan for any available wireless network to join for Internet connectivity. In addition, work will be done to develop cost-effective self-driving cars and drones to be used as network recovery vehicles.

7 Conclusion

A modern emergency communication system is essential for maintaining communications in a disaster zone. This thesis presents a novel approach to such an ECS utilizing SDN, WMN, and unmanned vehicles. The presented architecture uses a modified wireless mesh router to support both wireless mesh and SDN protocols. The SDN controller is used to provide routing logic to the data traffic and load balancing logic to reduce congestion in gateway routers. In addition, a CMS is implemented to continuously monitor the local mesh networks and store the resulting statistics in a central database. The central management system also runs an efficient unmanned vehicle placement algorithm to effectively reduce the number of drones or self-driving cars deployed to recover the network. The SDN controllers use a publish/subscribe northbound interface to communicate with the central system. The proposed framework has been evaluated using the CORE network emulator showing the system to perform adequately.

References

[1] W. Cai, S. Borlace, M. Lengaigne, P. Van Rensch, M. Collins, G. Vecchi, A. Timmermann, A. Santoso, M. J. McPhaden, L. Wu, *et al.*, ‘‘Increasing frequency of extreme el niño events due to greenhouse warming,’’ *Nature climate change*, vol. 4, no. 2, pp. 111--116, 2014.

[2] V. Y. Kishorbhai and N. N. Vasantbhai, ‘‘Aon: A survey on emergency communication systems during a catastrophic disaster,’’ *Procedia Computer Science*, vol. 115, pp. 838--845, 2017.

[3] L. Rabieekenari, K. Sayrafian, and J. S. Baras, ‘‘Autonomous relocation strategies for cells on wheels in public safety networks,’’ in *Consumer Communications & Networking Conference (CCNC), 2017 14th IEEE Annual.* IEEE, 2017, pp. 41--44.

[4] P. Karn, H. Price, and R. Diersing, ‘‘Packet radio in the amateur service,’’ *IEEE Journal on Selected Areas in Communications*, vol. 3, no. 3, pp. 431--439, 1985.

[5] R. Singh, M. Thompson, S. A. Mathews, O. Agbogidi, K. Bhadane, and K. Namuduri, ‘‘Aerial base stations for enabling cellular communications during emergency situation,’’ in *2017 International Conference on Vision, Image and Signal Processing (ICVISP).* IEEE, 2017, pp. 103--108.

[6] R. Bruno, M. Conti, and E. Gregori, ‘‘Mesh networks: commodity multihop ad hoc networks,’’ *IEEE communications magazine*, vol. 43, no. 3, pp. 123--131, 2005.

[7] A. Abujoda, D. Dietrich, P. Papadimitriou, and A. Sathiaseelan, ‘‘Software-defined wireless mesh networks for internet access sharing,’’ *Computer Networks*, vol. 93, pp. 359--372, 2015.

[8] F. C. Commission *et al.*, ‘‘The role of deployable aerial communications architecture in emergency communications and recommended next steps,’’ *Washington, DC: FCC*, 2011.

[9] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, ‘‘Core: A real-time network emulator,’’ in *Military Communications Conference, 2008. MILCOM 2008. IEEE.* IEEE, 2008, pp. 1--7.

[10] P. Dely, A. Kassler, and N. Bayer, ‘‘Openflow for wireless mesh networks,’’ in *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on.* IEEE, 2011, pp. 1--6.

[11] A. Neumann, C. Aichele, M. Lindner, and S. Wunderlich, ‘‘Better approach to mobile ad-hoc networking (batman),’’ *IETF draft*, pp. 1--24, 2008.

[12] T. Clausen and P. Jacquet, ‘‘Optimized link state routing protocol (olsr),’’ Tech. Rep., 2003.

[13] I. D. Chakeres and E. M. Belding-Royer, ‘‘Aodv routing protocol implementation design,’’ in *Distributed Computing Systems Workshops, 2004. Proceedings. 24th International Conference on.* IEEE, 2004, pp. 698--703.

[14] J. Chroboczek, ‘‘The babel routing protocol,’’ as accessed on 02/01/2018. [Online]. Available: https://tools.ietf.org/html/rfc6126

[15] Guifi.net, ‘‘Guifi.net, commons telecommunications network open, free and neutral,’’ as accessed on 02/01/2018. [Online]. Available: https://guifi.net/en

[16] D. Marschke, J. Doyle, and P. Moyer, *Software Defined Networking (SDN): Anatomy of OpenFlow Volume I.* Lulu. com, 2015, vol. 1.

[17] M. Aslan and A. Matrawy, ‘‘Maintaining an up-to-date global network view in sdn,’’ *arXiv preprint arXiv:1612.04944*, 2016.

[18] S. Salsano, P. L. Ventre, L. Prete, G. Siracusano, M. Gerola, and E. Salvadori, ‘‘Oshi-open source hybrid ip/sdn networking (and its emulation on mininet and on distributed sdn testbeds),’’ in *Software Defined Networks (EWSDN), 2014 Third European Workshop on.* IEEE, 2014, pp. 13--18.

[19] M. Rademacher, K. Jonas, F. Siebertz, A. Rzyska, M. Schlebusch, and M. Kessel, ‘‘Software-defined wireless mesh networking: Current status and challenges,’’ *The Computer Journal*, vol. 60, no. 10, pp. 1520--1535, 2017.

[20] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, *et al.*, ‘‘The design and implementation of open vswitch.’’ in *NSDI*, 2015, pp. 117--130.

[21] Y. Bai, W. Du, Z. Ma, C. Shen, Y. Zhou, and B. Chen, ‘‘Emergency communication system by heterogeneous wireless networking,’’ in *Wireless Communications, Networking and Information Security (WCNIS), 2010 IEEE International Conference on.* IEEE, 2010, pp. 488--492.

[22] A. Konak, ‘‘Improving network connectivity in emergency ad hoc wireless networks.’’ in *ISCRAM*, 2014.

[23] J. Xu, K. Ota, and M. Dong, ‘‘Fast networking for disaster recovery,’’ *IEEE Transactions on Emerging Topics in Computing*, 2017.

[24] T. Gao, F. Lang, and N. Guo, ‘‘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, 2016, pp. 90--95.

[25] R. Grodi and D. B. Rawat, ‘‘Uav-assisted broadband network for emergency and public safety communications,’’ in *Signal and Information Processing (GlobalSIP), 2015 IEEE Global Conference on.* IEEE, 2015, pp. 10--14.

[26] P.-V. Mekikis, A. Antonopoulos, E. Kartsakli, L. Alonso, and C. Verikoukis, ‘‘Communication recovery with emergency aerial networks,’’ *IEEE Transactions on Consumer Electronics*, vol. 63, no. 3, pp. 291--299, 2017.

[27] M. Erdelj, M. Król, and E. Natalizio, ‘‘Wireless sensor networks and multi-uav systems for natural disaster management,’’ *Computer Networks*, vol. 124, pp. 72--86, 2017.

[28] G. Baldini, T. Sturman, A. Dalode, A. Kropp, and C. Sacchi, ‘‘An emergency communication system based on software-defined radio,’’ *EURASIP Journal on Wireless Communications and Networking*, vol. 2014, no. 1, p. 169, 2014.

[29] A. Xie, X. Wang, W. Wang, and S. Lu, ‘‘Designing a disaster-resilient network with software defined networking,’’ in *Quality of Service (IWQoS), 2014 IEEE 22nd International Symposium of.* IEEE, 2014, pp. 135--140.

[30] M. Manic, D. Wijayasekara, K. Amarasinghe, J. Hewlett, K. Handy, C. Becker, B. Patterson, and R. Peterson, ‘‘Next generation emergency communication systems via software defined networks,’’ in *Research and Educational Experiment Workshop (GREE), 2014 Third GENI.* IEEE, 2014, pp. 1--8.

[31] A. Detti, C. Pisa, S. Salsano, and N. Blefari-Melazzi, ‘‘Wireless mesh software defined networks (wmsdn),’’ in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on.* IEEE, 2013, pp. 89--95.

[32] K. G. Melkote and P. J. Iyer, ‘‘System and method for advertising the same service set identifier for different basic service sets,’’ May 24 2011, uS Patent 7,948,953.

[33] J. Malinen, ‘‘hostapd: Ieee 802.11 ap, ieee 802.1 x/wpa/wpa2/eap/radius authenticator,’’ *Hostapd: IEEE 802.11 AP, IEEE 802.1 X/WPA/WPA2/EAP/RADIUS Authenticator*, 2013.

[34] P. Mockapetris and K. J. Dunlap, *Development of the domain name system.* ACM, 1988, vol. 18, no. 4.

[35] R. Droms, ‘‘Dynamic host configuration protocol,’’ as accessed on 02/01/2018. [Online]. Available: https://tools.ietf.org/html/rfc2131

[36] S. Kelley, ‘‘Dnsmasq,’’ as accessed on 02/01/2018. [Online]. Available: http://www.thekelleys.org.uk/dnsmasq/docs/dnsmasq-man.html

[37] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, ‘‘Openflow: enabling innovation in campus networks,’’ *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69--74, 2008.

[38] S. Kaur, J. Singh, and N. S. Ghumman, ‘‘Network programmability using pox controller,’’ in *ICCCS International Conference on Communication, Computing & Systems, IEEE*, vol. 138, 2014.

[39] S. Skiena, ‘‘Dijkstra’s algorithm,’’ *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley*, pp. 225--227, 1990.

[40] A. Hagberg, P. Swart, and D. S Chult, ‘‘Exploring network structure, dynamics, and function using networkx,’’ Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[41] M. Shreedhar and G. Varghese, ‘‘Efficient fair queuing using deficit round-robin,’’ *IEEE/ACM Transactions on networking*, vol. 4, no. 3, pp. 375--385, 1996.

[42] M. Kerrisk, ‘‘top - display linux processes,’’ as accessed on 2017/10/18. [Online]. Available: http://man7.org/linux/man-pages/man1/top.1.html

[43] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, ‘‘Mqtt - a publish/subscribe protocol for wireless sensor networks,’’ in *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on.* IEEE, 2008, pp. 791--798.

[44] B. Krishnamachari and K. Wright, ‘‘The publish-process-subscribe paradigm for the internet of things,’’ 2017.

[45] MySQL, ‘‘Mysql documentation,’’ 2017, as accessed on 10/21/2017. [Online]. Available: https://dev.mysql.com/doc/

[46] Amazon, ‘‘Amazon web services,’’ as accessed on 02/01/2018. [Online]. Available: https://aws.amazon.com/

[47] A. Lukaszewski and A. Reynolds, *MySQL for Python.* Packt Publishing Ltd, 2010.

[48] W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2.* Addison-Wesley Longman Publishing Co., Inc., 1998.

[49] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, ‘‘Network configuration protocol (netconf),’’ as accessed on 02/01/2018. [Online]. Available: https://tools.ietf.org/html/rfc6241

[50] D. Josephsen, *Building a monitoring infrastructure with Nagios.* Prentice Hall PTR, 2007.

[51] OpenNMS, ‘‘What is opennms,’’ as accessed on 02/01/2018. [Online]. Available: https://wiki.opennms.org/wiki/What_is_OpenNMS

[52] Google, ‘‘Adding a google map with a marker to your website,’’ 2017, as accessed on 10/01/2017. [Online]. Available: https://developers.google.com/maps/documentation/javascript/adding-a-google-map

[53] L. MapBox, ‘‘Mapbox,’’ as accessed on 02/01/2018. [Online]. Available: https://www.mapbox.com/pricing

[54] J. Han, ‘‘Spatial clustering methods in data mining: A survey,’’ *Geographic data mining and knowledge discovery*, pp. 188--217, 2001.

[55] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, ‘‘A density-based algorithm for discovering clusters in large spatial databases with noise.’’ in *Kdd*, vol. 96, no. 34, 1996, pp. 226--231.

[56] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, ‘‘API design for machine learning software: experiences from the scikit-learn project,’’ in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108--122.

[57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, ‘‘Scikit-learn: Machine learning in python,’’ *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825--2830, 2011.

[58] N. R. Chopde and M. Nichat, ‘‘Landmark based shortest path detection by using a* and haversine formula,’’ *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 1, no. 2, pp. 298--302, 2013.

[59] R. Pi, ‘‘Raspberry pi 3 model b,’’ as accessed on 02/01/2018. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/