Using an energy-constrained relay network cluster to transmit disaster area information to command and control center

December 7th, 2016

Austin Bachman
Michael McKissick
Jeff Williams

Abstract

The need arises for communication from inside disaster areas with battery-constrained devices for one or more disasters. UAV and drone technology is currently being researched and tested for its tremendous potential in emergency/first response environments, since UAVS, drones, and vehicles can create multi-hop relays back to a base station. Emergency responders communicate to dispatch/base station command utilizing a variety of radio frequencies and repeaters used throughout the state to update initial and ongoing communication between command and adjoining forces. Since Reno and surrounding areas are mountainous, this produces dead zones within certain radio frequencies. There is line-of-sight transmission, which uses a local radio frequency that doesn't contact a repeater. Since it's not always possible for radios to hit a repeater or line-of-sight communication, utilizing UAVs with antennas, repeaters, OLSR, and Peer-to-Peer communication can fill these gaps. For this project, we will facilitate the transfer of disaster area information to a single control center through the use of an energy-constrained relay network cluster.

Goals

1. To simulate a relay network cluster.
2. To establish a connection from this cluster to a simulated command and control center.
3. To successfully transmit disaster area information as energy-efficiently as possible.
4. Simulate the proper data needed to provide an accurate update utilizing various technological tools, apps, and sensors.
5. Simulate a realistic environment with a finite number of UAVs.

Specifications

- Peer-to-Peer in relay and Client-Server to control center
- Network Protocols - OSI Model
- Programming language - C++
- OS - Run simulation on Linux

Roles

- Austin - UAV Relay Protocol Design and Implementation
- Michael - UAV Relay Simulation and Analysis
- Jeff - Program Documentation and Editing

Milestones

I. Generate drone relay network to a single disaster
II. Allow multiple relays to stem to multiple disasters
III. Simulate passage of time, battery drain, drone replacement, and error handling

Code Report

Our Simulator is split into four files: structs.cpp, simulator.h, simulator.cpp, and Main.cpp. In addition to this code report, each file contains internal documentation.

structs.cpp - This file contains two struct definitions, Drone and Disaster.

The Drone struct contains 5 integer data fields to hold the data for the drone identifier, battery life remaining, x-position, y-position, and disaster identifier that the Drone is linked to. It also contains two boolean fields. One to mark a drone at the Base Station, and one to mark a Drone as the closest Drone to a Disaster. Lastly there are two more fields that are both of type Drone pointer. These are used to create a relay of Drones using a linked-list type method.

The Disaster struct contains only three integer data fields to hold an x-position, y-position, and Disaster identifier.

simulator.h - This is the header file for simulator.cpp. It contains all of the function prototypes for the functions used in Main.cpp and implemented in simulator.cpp.

simulator.cpp - This file contains the function implementations for all of the functions used by our simulator. Each function will now be documented in detail.

generateFleet **-** Creates an array of Drones and initializes all Drones with default values for all data fields. Each Drone is given a unique ID, a random starting battery level between 50-100 (100 is a full battery, 0 is an empty battery), and default values for the rest of the fields in the Drone struct. rand() is used here from stdlib.h in order to start the Drones of with varying battery lives.

makeDisaster - Takes dynamic input from the user of the coordinates of a new disaster. Coordinates are saved into a Disaster struct.

generateRelay - Checks to see if a relay is possible to a disaster using available Drones. If it is not possible this function returns false. Otherwise the distance formula is used to calculate the distance to the disaster from the command and control center as well as from the closest Drone to the disaster. This information is used to place a Drone at the first step of the relay. This process repeats until a relay has been successfully formed.

highestPower - Searches through an array of Drones to find the one with the highest battery life remaining. This is useful for selecting a Drone from the command and control center to add to a relay.

Distance - Calculates the distance formula given two sets of x-y coordinates.

displayRelay - Prints relay information for each disaster including Drone identifiers, positions, and battery lives.

getDirVector - Scales x and y increase based on angle to next position.

findClosest - Checks every in flight Drone to find which is closest to a provided Disaster.

getUserChoice - This is the menu for the simulation. There are 6 options: 1 generate new disaster, 2 print relay network information, 3 simulate x minutes of time passing, 4 simulate sending a packet, 5 print map, and 6 quit.

simulateTime - Simulates x minutes passing based on user input. Time is simulated by decrementing the battery life of each in flight drone by 1 each minute. Charging Drones at the command and control center have their batteries incremented by 5 per minute.

updateBatteries - Called by simulateTime each simulated minute to loop through all of the Drones and either decrement their battery if they are in flight or increment their battery if they are charging at the command and control center.

replaceDrone - Calls highestPower to check if a drone is available to replace the low battery Drone. If a drone is available, all pointers are updated in order to swap into the relay the new Drone. Also the default values for the new Drone in the relay are also updated. Finally the swap is printed to the console.

sendPacket - Simulates sending a packed across a relay from a Disaster to the command and control center. The user inputs coordinates for where the packet needs to be sent from. If the packet can be sent a message is printed to the console indicating the path it would take. Drones are replaced as necessary as sending a packet drains the battery life of the Drones along the path. A console message is printed upon the command and control center receiving the packet.

printMap - Prints to the terminal a view of the entire xy-coordinate field including the command and control center, Disaster locations, and Drone relays.

Main.cpp - This is the main driver for the simulator. It begins by allocating an array of Drone structs as well as a vector to hold all of the Disaster structs. Next the simulator prompts the user for disaster coordinates which are used to generate a relay of Drones. Then the simulator enters a loop of prompting the user for a choice using getUserChoice and then carrying out the simulation of that choice by calling functions implemented in simulator.cpp.

Note: To compile our code we used a makefile. In Linux to compile and run the simulator put the four simulator files into a folder with the makefile and open a command prompt. Change directories to the directory that you put all of the files in and enter “make” into the command prompt. Finally enter “./Sim” to run the simulator. Follow on screen instructions.

Functionality

Our task for this project was to design an energy-constrained, relay network cluster to transmit disaster area information to command and control. We envisioned some of the issues first responders face when dealing with multiple incidents. As can be seen during major fire seasons, there can be hundreds of small to large wildland fires happening all around the state simultaneously. Bearing this in mind, we decided to use UAVs to relay disaster information back to a central base station in case a new disaster were to strike in a remote location.

When an incident or disaster is first reported, first responders will have to acquire a latitude and longitude, or a township and range. This is done by going out to the physical area with a GPS, and then calling the coordinates back to dispatch. Problems arise, however, when a first responder is unable to communicate with dispatch due to being out of range of a repeater due to difficult topology, or being in a very remote location. A first responder will have to go back to an area where they can communicate with dispatch, sometimes hours away, in order to relay information. This is where UAVs come in.

Since UAVs are a finite resource, and have a limited battery life, we had to figure out ways to best utilize them to set up a relay to communicate incident information to the dispatch control center. It makes sense to have UAVs centrally located at a base station, whereby they can be charged and set up to be rapidly deployed to new incidents as they occur. Once a new incident occurs, we form a chain of UAVs deployed straight to the incident to relay information back to the base station.

Simulating a latitude and longitude, for our purposes we decided to use a 2D x,y positive axis to relay disaster information. With the base station/dispatch center oriented at (0,0), we then simulate a new incident coming in from x and y coordinates, utilizing inputs from the user dynamically. We then calculate the distance to the incident from the base station using the distance formula:

$$\text{sqrt}((x2-x1)^2 + (y2-y1)^2)$$

Once the distance is calculated, we calculate an ideal number of UAVs to send to the incident by measuring the radial range each UAV has, leaving extra distance for error. For instance, if the distance to the incident was 80 miles, and each UAV has a connectivity radius of 20, we would require a minimum of 4 UAVs to relay information back to the base station.

Our problem with calculating distance and deploying UAVs wasn't as simple as a linear path. For one, maximizing the amount of space a UAV can touch another UAV with a signal poses several problems. If the UAVs are already maxed out on distance, and one goes down for a battery, a loss of communication between links will be inevitable. To account for this, we shortened the distance between UAVs, so we could simulate an additional UAV bumping in from the base station, while still allowing the already deployed UAVs the flexibility to maintain constant communication with the incident. Another problem that arose was figuring out a way to utilize a finite amount of drones in the case of multiple incidents. With this problem, we devised a solution that decreases the amount of UAVs used for multiple incidents.

Since incidents may be close together, and even if they're not, there may be a UAV closer to a new incident than the base station itself. When a new incident is initialized with coordinates, instead of sending out an entirely new UAV line from the base station, the distance is once again calculated between the base station and all other UAVs on the incident. If there is a UAV closer to the new incident than the base station, a smaller amount of UAVs are sent out to chain from the already existing UAV line. This drastically decreases the amount of UAVs used to respond to new incidents. This models a real world scenario in a place such as Elko, NV, where there can be 100's of fires and small spot fires happening within a one hundred mile radius simultaneously.

What is interesting about creating a dynamic tree that branches out to separate incidents, while using existing UAV lines to relay new information, is the amount of UAVs used for each incident has an inverse relationship with the number of incidents. The more incidents there are, the less likelihood a full UAV-line will have to be deployed. This is a good solution for real-world scenarios where resources are spread thin. This is a somewhat unique situation in the world of disaster response, since typically having more incidents will usually require a linear growth of responding units.

There are many different routing protocols being studied to determine the efficiency of UAV networks. Our model closely resembles Optimized Link State Routing (OLSR). In the paper, "Survey of Important Issues in UAV Communication Networks", Lav Gupta states, "OLSR is currently one of the most employed routing algorithms for ad hoc networks. Routes to all destinations are determined at startup and then maintained by an update process. Nodes exchange topology information with other nodes of the network regularly by broadcasting the link-state costs of its neighboring nodes to other nodes using a flooding strategy. Other nodes update their view of the network by choosing the next hop by applying shortest path algorithms to all destinations." From this perspective, we have taken certain attributes from the OLSR protocol, while simplified other versions. For instance, we route all destinations to a specific location before the node is deployed, by calculating the distance (path weight) beforehand, and then scanning all other nodes from the network to find a shortest path prior to the node being deployed. Since the nodes start from the base station and work their way out, and each node is calculated from the lowest distance to the higher, there will no be an extraneous links added,

and only the shortest path will be used. However, there is no need for us to constantly use a shortest path algorithm for the nodes for each new node added, since this was calculated prior to deploying each UAV.

We decided to use simplicity of our design in establishing back pointers, so each UAV node in the network doesn't constantly have to keep updating tables, but new nodes are able to be added dynamically to the network. This design approach could even be combined with other networks, if a current node is able to create a back-pointer and connect with a drone from, for instance, a vehicle network, a back-pointer is all that is needed. Since each node is able to piggyback off of an established network backbone, this design is easily scalable, and provides a further spanning network as the size and complexity of multiple incidents rises.

The picture below, taken from, "Communication Architectures and Protocols for Networking Unmanned Aerial Vehicles" shows a centralized UAV network on the top and UAV Ad Hoc Network on the bottom.

Fig. 1. Centralized UAV Network

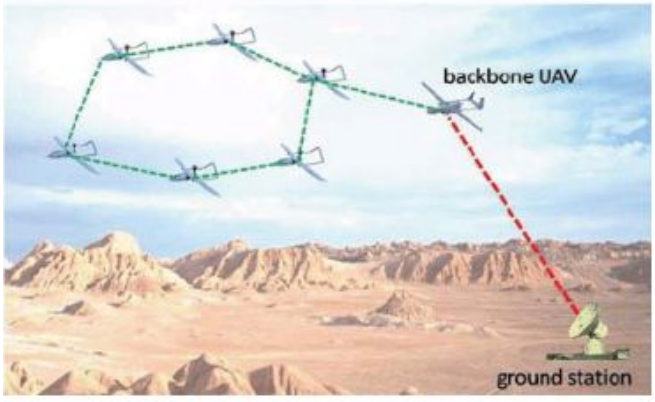


Fig. 2. UAV Ad Hoc Network

The centralized UAV network must relay data directly to the base station with other nodes using the base station as a relay to each other. This limits the range in comparison to the UAV ad hoc network, since the UAV Ad Hoc Network relies on backbones to communicate back the base station. This can be further developed into multi-group or multi-layer hops.

However, for our design, we encompassed the functionality of both. When the disaster is originally reported, if the incidents are far away from each other, there will be multiple UAV lines created, perhaps both originating from the base station. However, it will eventually always form a backbone of nodes with UAVs being deployed, but will not be limited to a single backbone as in figure 2. The network is able to be expanded, and always uses the shortest path back to the base station using a similar protocol as OLSR simulating shortest-path IP datagram packets. This will be further detailed with screen-shots in our results.

We have designed our simulator program with a menu as follows:

1. Generate new disaster
2. Print relay network information
3. Simulate x minutes of time passing
4. Simulate sending a packet from (x,y)
5. Print relay map

Upon starting the program, the user is prompted to enter the x and y coordinates of an initial disaster/incident. This simulates a first responder calling in an initial size-up response to dispatch using latitude/longitude or township/range coordinates. From there a UAV line of drones is deployed to the incident. After that, the user can enter in as many initial disaster/incidents as they want, so long as the base station remains with 20% available UAVs to swap out other UAVs due to power constraints. If a relay cannot be established with the remaining drones, reserving 20% for replacements, then no drones are placed and an error message is returned to the user. We added a simulate time function to simulate draining the batteries due to fly time, as well as a function to simulate sending a packet.

Our design shadows the OLSR model, such as using routes to all destinations maintained and known before use. We also use multi-point relays in a similar manner as OLSR, except we use backpointers back to the base station, so there is no need to constantly advertise link-state information. We simulate an IP datagram being sent through the network by simply finding the closest available node to the location, and then sending the datagram to the base station through the shortest path made available with back-pointers and proper calculations.

Deviations From Existing Research

A large amount of available research surrounding UAV networks treats them as ad hoc networks. The work “A Distributed Gateway Selection Algorithm for UAV Networks” denotes many parallel applications for their use of a mobile ad hoc network (MANET) as ours. There are several keys points in the paper, notably:

“According to the existing applications of UAV, there are four kinds of main communication requirements as follows:

1. Send back the sensor data.
2. Receiving the control commands
3. Cooperative trajectory planning
4. Dynamic task assignments.”

The paper highlights that the first two happen between the MANET and the external network, while the last two communications are typical with those existing in a usual MANET. Every UAV can connect to a command center through remote locations, and the number of remote locations should be controlled to avoid interference, and satisfy the restrictions of limited resources. The paper states, “Therefore, some superior nodes in the network should act as gateways, so that other nodes in the network can connect the command center through them rather than to establish a remote location.”

For our purposes, we used this line of thinking of “gateway” nodes to create our initial UAV line to an incident. Rather than have nodes constantly communicating with each other globally, we established connections of nodes in a parent-child relationship using back-pointers to the base station. This simplifies the overall design, while still maintaining the functionality needed for communication links responding from remote locations back to the base station. The initial UAV line creates a solid “backbone” where other nodes are able to join. While each individual node maintains a link between its child/parent nodes, the base station will have the locations of all the nodes to position nodes accordingly by extending off existing UAV-lines.

The article further states, “The selection of gateway in a UAV network is quite similar to the selection of a cluster head in an Ad Hoc network. However, the existing works regarding Ad-Hoc clustering have not considered the movement of nodes when selecting the cluster head… the cluster selection is based on each node’s local information, which is not global optimized.” However, for our project, we created a clear-link path back to the base station using only neighboring communication. From this point, I argue only 3 things are necessary to create an ideal UAV path to an incident, while maintaining a minimal amount of UAVs needed to maintain communication among new incidents.

1. Disaster coordinates
2. Radius of UAV communication link
3. Locations of all active UAVs on incidents

Since the initial incident/disaster coordinates are communicated back to the base station before the UAVs are deployed, an ideal path to the incident can be calculated previous to having the UAVs deployed. There shouldn't be a reason to send an excess amount of UAVs to an incident, while having them come up with an ideal formation dynamically after they've already been deployed when we have the initial coordinates that are already radioed in as per standard operating protocol of any first response incident. It is preferable to have the base station calculate where each UAV should go before being deployed, and have a record of where all UAVs are deployed over a grid to optimize where new UAVs will be sent in the case of any new incident. Since the UAVs have already created a solid line of direct communication back to the base station, rather than move that line in an awkward formation, it simplifies the design to take advantage of an established communication link, and only send out a fraction of UAVs to a new incident to minimize the amount of UAVs necessary while still spanning a large area.

Furthermore, since our network is energy-constrained, we will have to replace UAVs that must go down for a battery by bumping up the other UAVs in position, while pushing a freshly-charged UAV from the base station up to the first link. Not only does this keep the communication links in tact with a simple solution, it also helps anchor in an established "gateway" to the base station by having the closest UAV to the base station the most easily swapped out and replaced, since it has the shortest distance to travel. Since multiple incidents may be using the same UAV line, the UAVs closer to the base station where the initial UAV lines run from will be getting hit with the most traffic, but are also the most easy UAVs to swap out, since they have a minimal distance to initially fly.

In the paper, "Survey of Important Issues in UAV Communication Networks", the authors recognize the differences UAVs exhibit with other ad hoc networks, "In mobile and vehicular networks the nodes join and dissociate from the network frequently and, therefore, ad hoc networks have been found to be suitable in most situations… In UAV networks, the nodes could almost be static and hovering over the area of operation of scouting around at a rapid pace. Nodese could die out for many reasons and may be replaced by new ones." Indeed, these are the very issues our project was facing. However, why UAVs can provide such a useful service to emergency management is that, since they are flying, they aren't subject to the same factors as vehicles, namely: they are in the sky (no need to go over rugged or impossible terrain exhibited by vehicles), they can remain stationary (hovering), they can be replaced by new nodes by keeping track of battery life and pushing new nodes out from the base station. Therefore, we have devised a solution that is able to update nodes dynamically, while using stationary nodes to relay information once the new nodes locations have been updating. In other words, nodes are able to be added dynamically, while using the strengths of having stationary, hovering UAVs relay information back to the base station using a "backbone" network of existing nodes.

Results

Our results show that with an increase in disasters/incidents present, the number of UAVs needed to be deployed to a scene diminished. The purpose of our research was to create a network capable of spanning multiple incidents simultaneously, while still accounting for a finite amount of UAVs. In this end, we accomplished our goal. We are also able to deliver packets using a shortest path back to the base station, which models the OLSR protocol, using IP datagrams to send pertinent information to a command base station.

Where we could seek out improvements for our project is in bottlenecks and network congestion. While we implemented a network that would use the least amount of UAVs necessary to respond to incidents properly, if bandwidth became an issue this may have to be reconfigured. If UAVs are simply bouncing radio waves off of each other as repeaters, this could be solved with using multiple frequencies. It would be important to have a strong enough antenna to relay many crossing frequencies. As far as packets, congestion control would be somewhat mitigated, since each packet only has one route to the base station, and this alleviates broadcasting and multicasting. However, if there are many UAV lines dependent upon one centrally-located drone, there could be issues. This would have to be tested. For possible solutions, there could be a certain number of UAV lines sent out for a specific geographic area, so not too many UAV lines are reliant upon a single backbone framework.

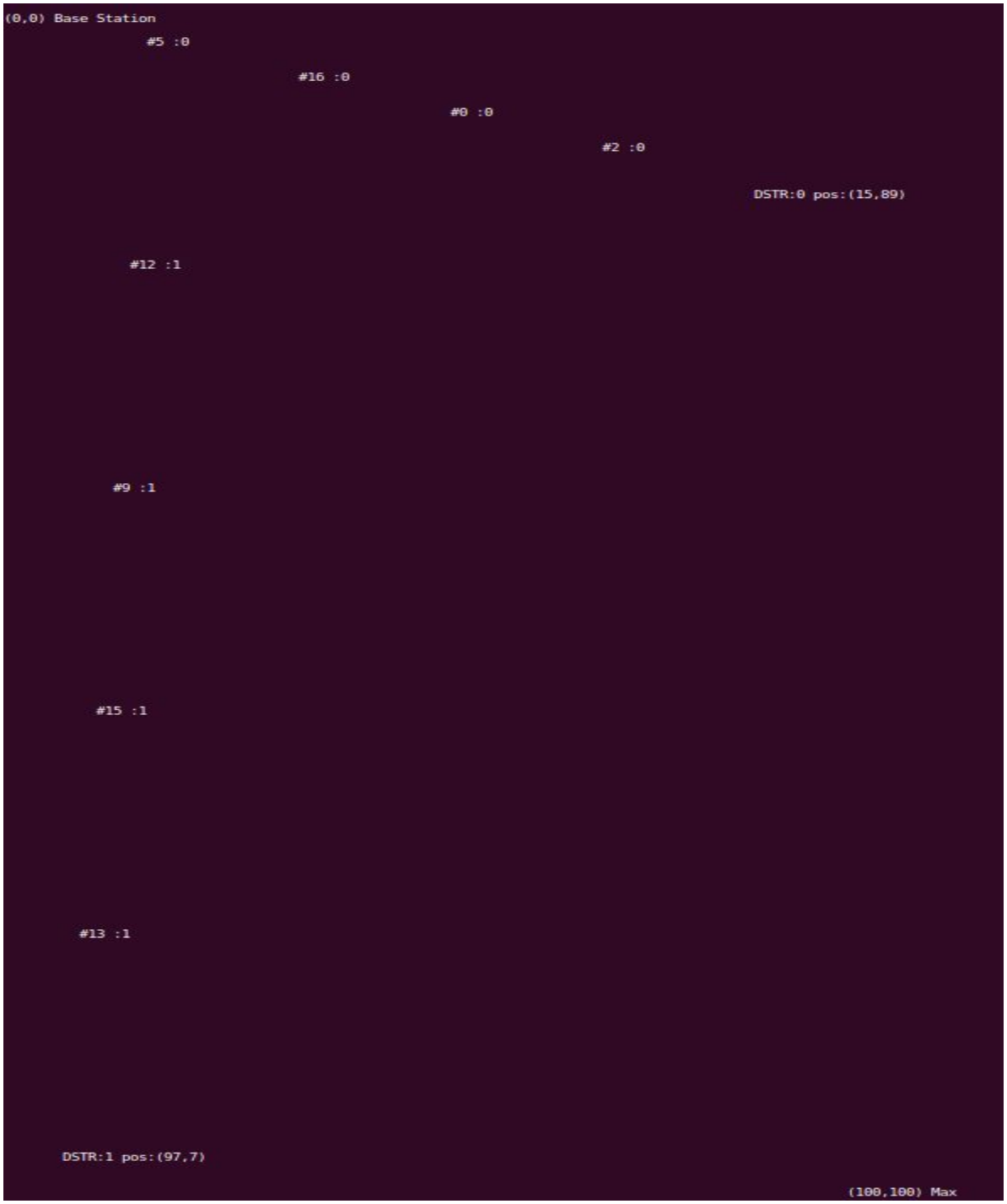
The following pages illustrate screen shots to better understand how our individual UAV lines react and organize themselves in the event of multiple incidents happening simultaneously.
The legend to identify the graph is as follows:

(0,0) Base Station coordinates

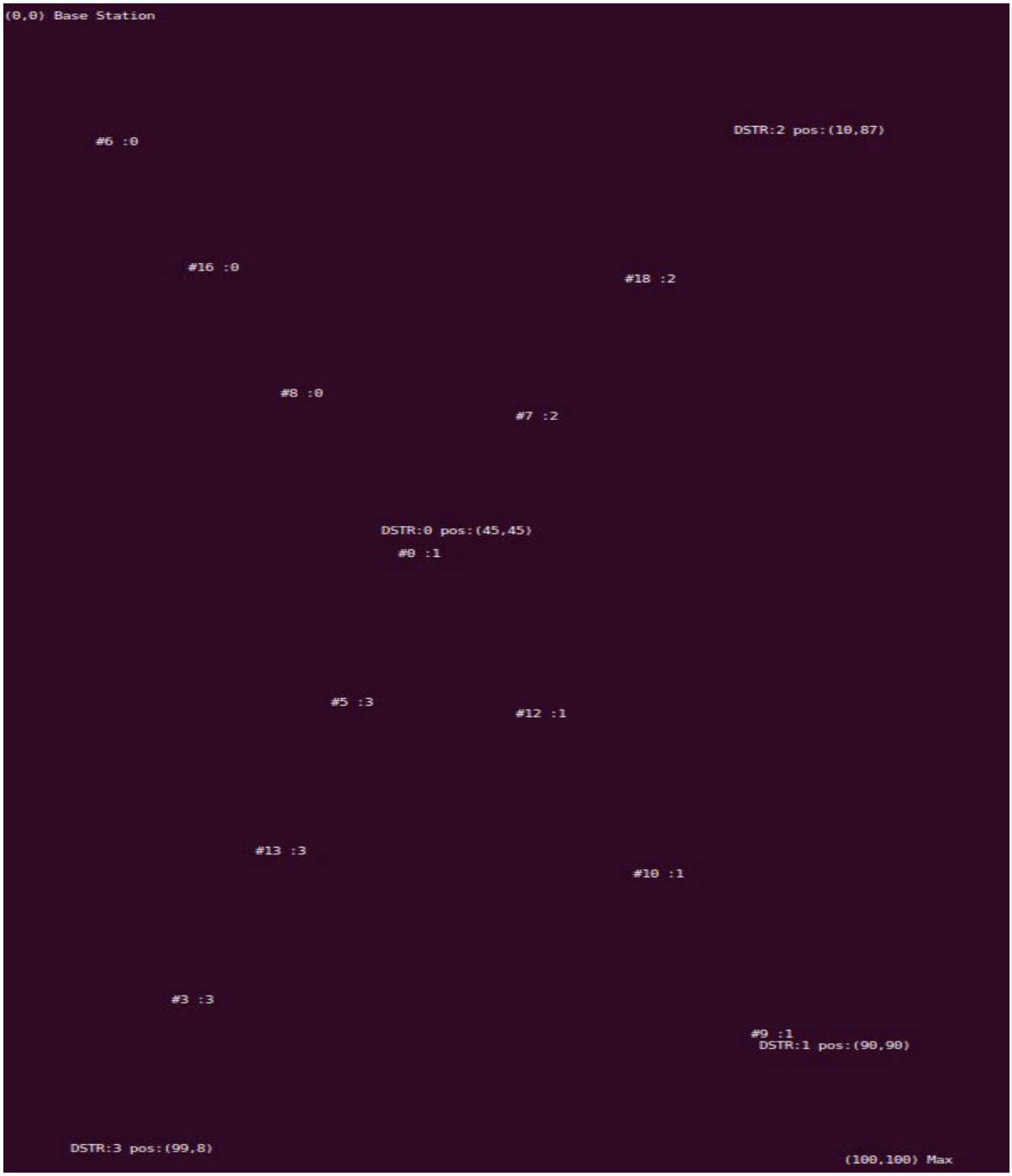
For each individual UAV:
#<droneID> : <disaster attached>

For each incident/disaster
DSTR: <disasterID> pos: (x,y)

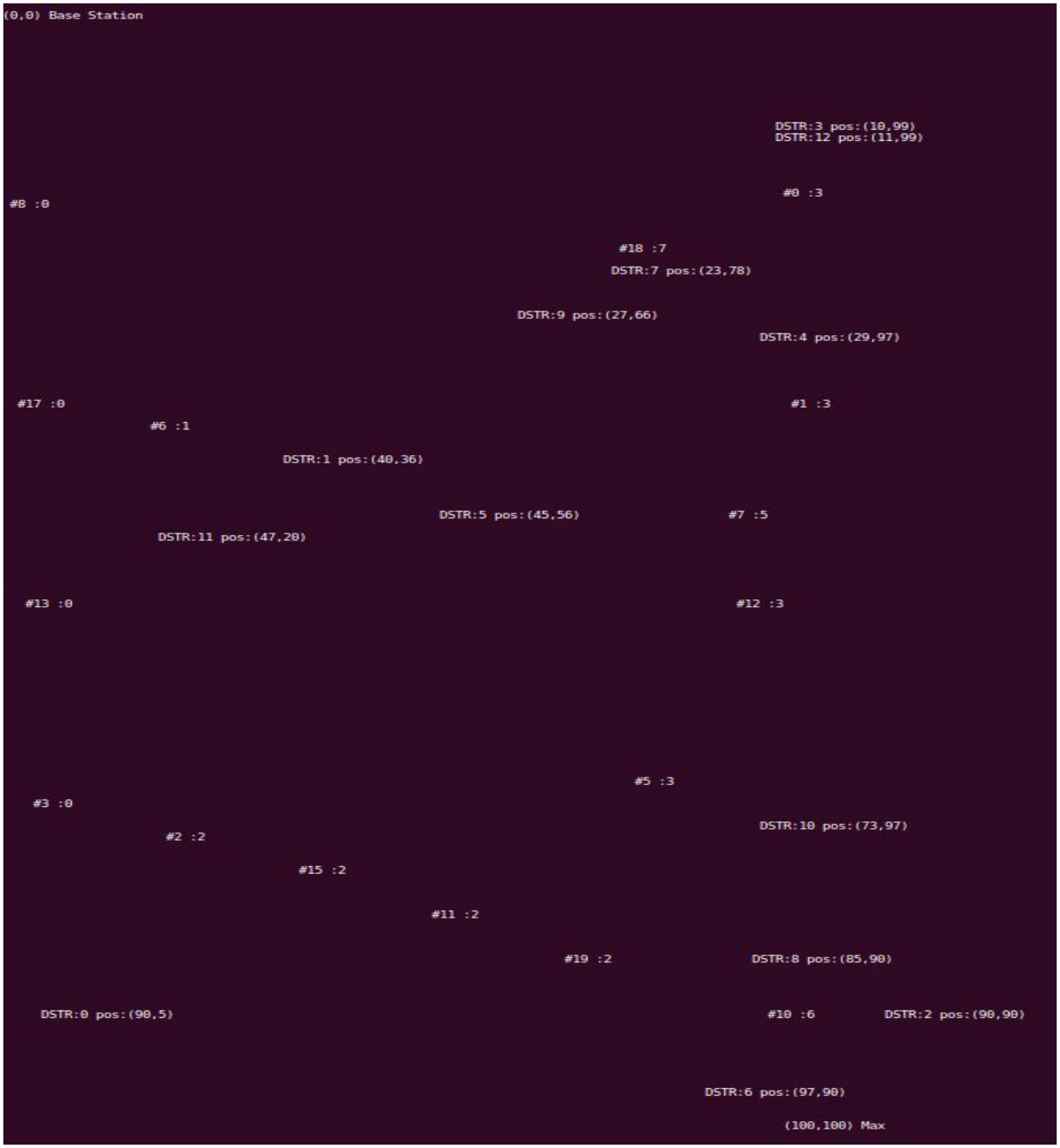
(100,100) Max position coordinates



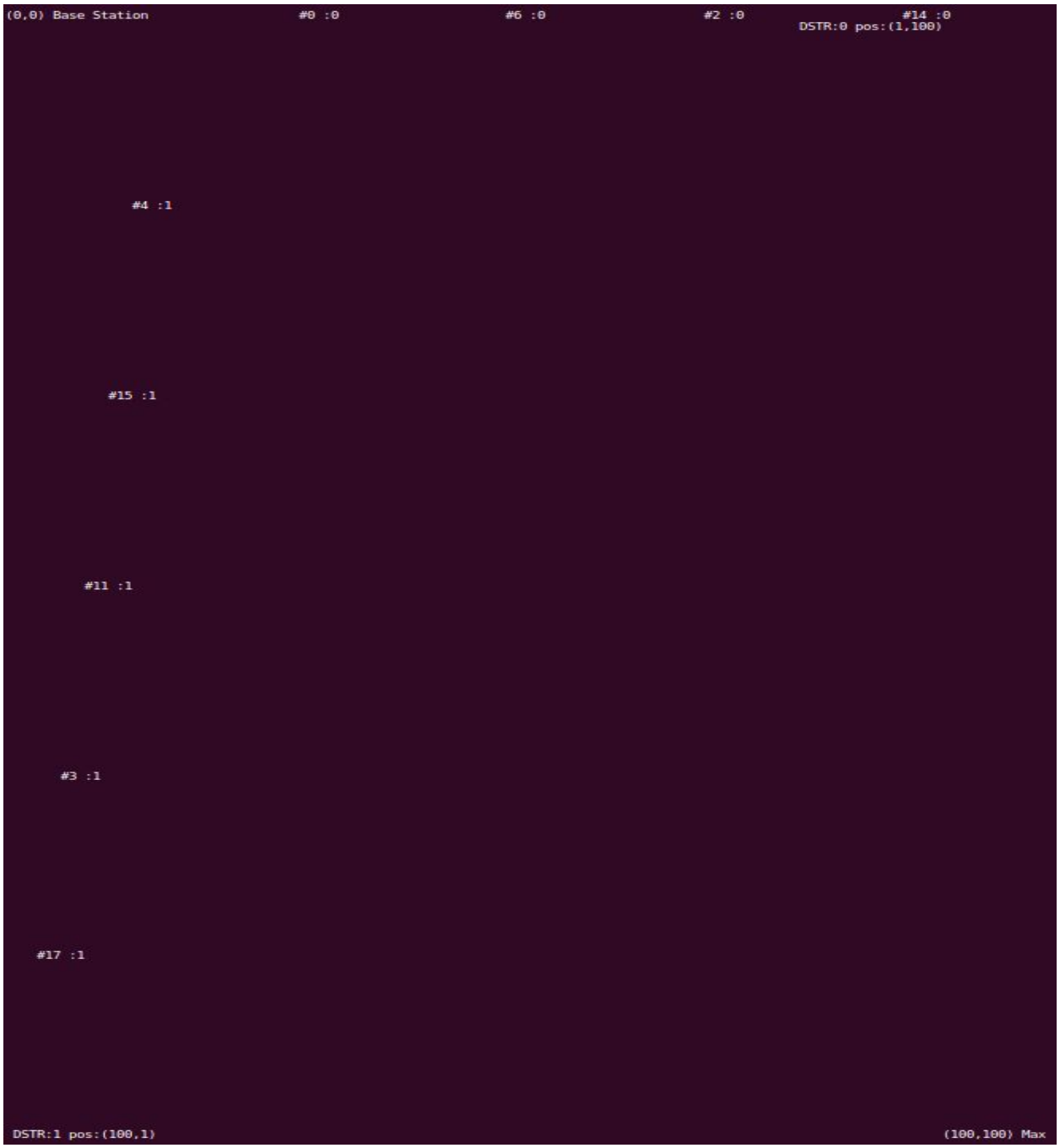
This screenshot highlights when there are two separate incidents in opposite locations, the UAV lines spread out according to the distance vectors accordingly, with UAV #5 becoming the backbone close to the base station for both UAV lines.



This screenshot shows 4 separate disasters happening simultaneously, and the each is in a corner of the grid, with one incident happening in the center of the grid. The UAVs for disaster 2 form off of disaster 0, using 2 drones to get to the disaster 2. This may be seen as moving backwards from the base station, but moving from disaster 0 takes the use of less UAVs than spanning from the base station. While many UAVs are overlapping networks, it depends on what our tightest constraints are: bandwidth or number of UAVs. We chose the approach to minimize the number of UAVs to an incident by always choosing the path that will require the least amount of UAVs to reach a disaster/incident.



Here, there are 12 disasters striking at once, yet the number of UAVs that must go out become less, since utilizing an existing UAV line becomes more frequent and available, the number of UAVs needing to be deployed has an inverse relationship with the number of disasters/incidents/fires going up. This is helpful to know that there should be a useful amount of UAVs always able to be deployed despite further disasters ensuing.



This screenshot shows two separate incidents/disasters happening at once, but with two completely separate UAV lines. This means that there is no current backbone, as both UAV lines are communicating back to the base station independently. This is a good scenario to have if there is a lot of traffic, and may be useful for future research if congestion-control becomes a future issue.

In this screenshot, we can see that there are 12 separate disasters happening at once, with many UAVs spread out in an even distribution around the center. This is because they are using two separate UAV backbones to return packets to the base station. Even though many of the incidents are close together towards the middle, each UAV is only operating off of a single backbone branch, so there is no overlap or graph functions to be continuously calculated.

As can be seen from the screenshots, there are many different spans and formations can configure themselves to, in response to dynamically changing disaster conditions. With the use of utilizing several backbone UAV lines, the amount of further UAVs sent out to new disasters is dramatically reduced. The more likelihood a UAV will be a backbone is much higher when the UAV is initially located near the base station, which is good because those UAVs are the quickest and easiest to replace in cases of loss of power due to battery.

Work Cited

J. Li, Y. Zhou, and L. Lamont, “Communication Architectures and Protocols for Networking Unmanned Aerial Vehicles,” Research Gate. [Online]. Available: Communication Architectures and Protocols for Networking Unmanned Aerial Vehicles.

F. Luo, C. Jiang, J. Du, J. Yuan, Y. Ren, S. Yu, and M. Guizani, “A Distributed Gateway Selection Algorithm for UAV Networks,” Emerging Topics In Computing, vol. 3, no. 1, pp. 22–31, Mar. 2015.

L. Gupta, R. Jain, and G. Vaszkun, “Survey of Important Issues in UAV Communication Networks,” IEEE Communications Surveys & Tutorials, vol. 18, no. 2, pp. 1123–1152, 2016.