Ad-hoc Wireless Network Design

Manushaqe Muco, Jessica Noss, Michelle Szucs

manjola@mit.edu, jmnoss@gmail.com, mszucs@mit.edu

Strauss/Simosa, TR 1,2 pm

05/12/2013

1. Introduction

Serving roving first responders requires a highly agile yet efficient network. This system design ensures quick message delivery by adapting to network changes. A dynamic determination of hop distance is coupled with an algorithm to distribute traffic between neighboring nodes in a manner that maximizes throughput. This design allows first responders to travel without disrupting their connection to the base station.

The system relies predominantly on decision-making at the individual node level to move messages toward the base station as quickly as possible. Localized decision-making reduces network traffic concerning node arrangement, freeing more network time for message transmission. Each node attempts to forward messages to neighboring nodes that are closer to the base station, distributing traffic in a manner that limits overall network congestion. The network leverages RSA public-key encryption to secure traffic.

The system implements an efficient routing protocol, which minimizes delay and reliably transmits images and location to the base station and between network nodes. Its design incorporates a security protocol, which ensures the system's robustness and protects the system from malicious attacks. Because of these implementations, the system achieves its purpose.

2. Design Implementation

2.1 Network Security

Before a node can be included in the network, it must be manually registered with the network, which uses RSA public-key encryption. During the registration process, a person in the base station enters the network's private key into each new radio device, registering the device as a node in the network at the same time. A public key is also attained at this time. The device does not need to be used immediately; it can be turned off until an emergency requires its use.

All information sent over the network is encrypted. Packets are structured into header and body. These two parts are encrypted in two separate blocks so that nodes can decrypt only the headers in the process of forwarding the message. Separate header encryption ensures that only a small part of a message must be decrypted to determine whether the sending node is a trusted network member and to continue forwarding that message. The exception to encryption is that the *secure_send(node, msg)* does not encrypt the node ID of the intended recipient so that nodes do not spend excess time decrypting messages clearly not meant for them.

Before a packet is encrypted and sent, the node adds its ID to the header. The receiving node decrypts the message header and checks that the sending node ID matches where the message came from. This process ensures that the network will not fall victim to replay attacks - should a malicious node attempt to repeat a message, the conflict in node IDs will become apparent. To accommodate this feature, the built-in functions are replaced with secure versions that include the encrypted sending node ID.

If a decrypted node ID does not match where the message came from, the node is deemed to be a threat. The receiving node will add the malicious node to a blacklist and ignore all subsequent packets from that node. The list can only be cleared through a manual process at the base station in the case that a compromised node is recovered.

2.2 Calculating hop distance to base station

Each node saves its distance from the base station, measured in hops, in *this.hops*. A few new encrypted commands are introduced to ensure the accuracy of the hop counts:

- *hopping(hop)*: broadcasts hop number to surrounding nodes.
- *hopscan()*: returns the list of nodes within radio range with the corresponding loss probability from the node performing the scan, the hop distance from each returning node to the base station, and the length of the node's image queue. The list is returned as a sequence of tuples (node, loss_prob, hop_distance, length_image_queue).
- *loop_notification()*: Broadcasts the node's list of its supposed parent nodes. It is used when an error occurs. When a node hears this message sent by its parent, it rebroadcasts the message and then sets its hop distance to null. This message is also encrypted to prevent wide-scale disruption by malicious nodes. Only hopping has higher priority than loop_notification.

Each node stores timestamps for the last time its hop distance was updated using each function. These two types of messages are handled immediately upon receipt. It is assumed that these messages are processed quickly enough to not require a separate queue.

When the network is initialized, the base station uses hopping to advertise a zero-hop distance. The hopping message propagates through each node in the network, and the process is repeated every 5 minutes. This period of time ensures that small errors inadequately addressed by hopscan do not accumulate to create large errors without creating an undue traffic burden of constantly updating the network configuration.

Upon receiving a hopping message, a node checks its hopping timestamp. If the timestamp is more than 5-ε minutes old (to allow for network delays and node relocation), the node sets its hop distance to the value in the message plus one. If the timestamp is less than or equal to 5-ε minutes old, the node only resets its hop distance and timestamp if this.hops will decrease. hopping will always overwrite null hop values.

For the most part, however, the system seeks to minimize time delays and network congestion by relying on the less-costly hopscan command. Every 30 seconds, each node issues a hopscan command to determine where to direct its traffic. This determination is made every 30 seconds since loss probabilities are relatively stable over 30-second intervals. hopscan will also run if two successive GPS readings differ by more than 30 meters (the lower bound of the radio range) because if a node has moved 30 meters, its hop distance and neighboring nodes may have changed. More information about how the *hopscan* command is used to forward message can be found in the **Routing Protocol** section.

When a node makes a hopscan call, its first task is to ensure that its hop number is correct. There are four possible cases:

A: The hop number is incorrect because it is set to null.

B: The hop number is correct because it is only within range of network nodes with *this.hops - 1, this.hops*, and *this.hops + n*, where n is a positive integer. (Note that the node doesn't care if other nodes have *this.hops + 2* - those nodes need to be updated, not the current node.)

C: The hop number is incorrect because there is at least one neighboring node with hop number of this.hops - 2.

D: The hop number is incorrect because there is no neighboring node with hop number *this.hops - n*, where n is a positive integer.

<u>Case A</u>

this.hops is updated to the smallest returned hop number plus 1. The list of parent nodes includes all nodes that return *this.hops - 1*, where *this.hops* is the newly-set value.

<u>Case B</u>

No changes are made to *this.hops*. The list of parent nodes is updated to include all nodes that returned *this.hops - 1*.

<u>Case C</u>

this.hops is updated to the smallest returned hop number plus 1. The list of parent nodes is updated to include all nodes that return *this.hops - 1*, where *this.hops* is the newly-set value. Child nodes of this will be updated on their next scans.

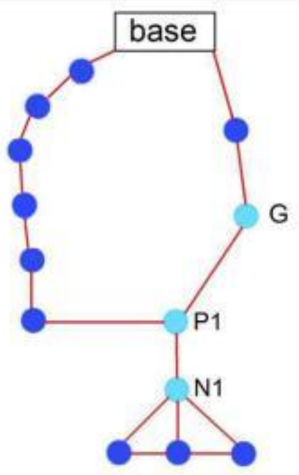
<u>Case D</u>

There are three main explanations for this case, which are illustrated in the Figure on the next page. Note that nodes archive their parent lists for the two previous periods. The cases are the following:

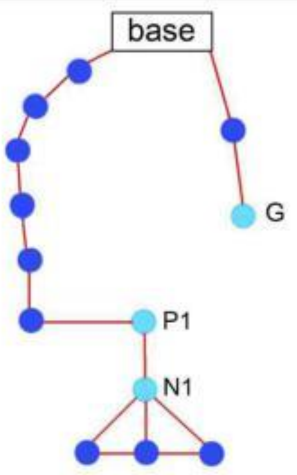
1. The last remaining parent node moved and has connected to another area of the graph.

2. The node broke connection with all its parent nodes.

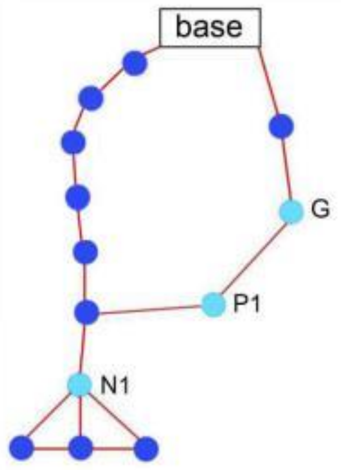
3. The connect between the last remaining parent node and the "grandparent node" (parent of the parent) was severed, and the parent-child relationship has inverted.



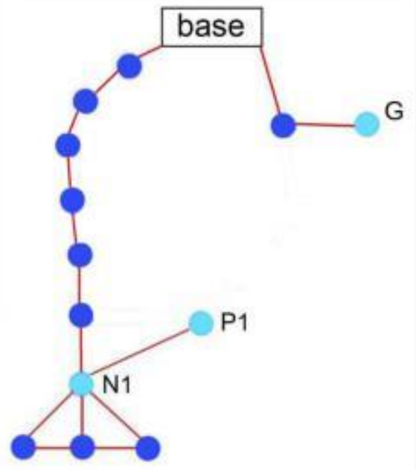
Scenario 0: Original scenario. P1 is the parent of node N1. G is the parent node of P1, "grandparent node" of N1.



Scenario 1: Some nodes get out of the range of parent P1. Node N1 is still connected to its parent. The parent node's path to the base has not changed, but N1 needs to find a new path to the base.



Scenario 2: Parent P1 has moved out of the range of node N1. Node N1 connects to a new parent, thus finding a new path to the base.



Scenario 3: The grandparent node G, has moved away, so that the parent's (P1) shortest path is now through the child (N1).

Figure: Some of the scenarios the ad hoc wireless network handles. Nodes are represented by blue circles. A node is connected to another node by a red line, if this latter node is in its range.

If some of the previous parent nodes are still in the new list returned by hopscan, the node will assume that **scenario 1** has happened, and set its hop number to one more than the lowest parent node. It will set its new parent node list to any nodes with equal hop distance as that parent node.

If no former parent nodes are returned in the latest hopscan, the node will assume **scenario 2**. It will assign its hop number to one more than the lowest returned value, setting nodes with that value as parents.

Scenario 3 will often initially be mistaken for scenario 1. Scenario 3 is the result of the parent node P1 running hopscan after disconnecting from the grandparent but before node N1 runs hopscan. P1 sees N1 as the neighboring node closest to the base station - in this case, the P1 node initially had hop distance of 3 and N1 had 4. When P1 runs hopscan, it assigns itself to a hop value of 5 (N1's plus one). Node N1 then runs hopscan, finding 1 node of hop distance 6 (which *should* be its new parent) and four nodes of hop distance 5 (P1 and N1's three children) around it. N1 assumes this is an instance of scenario 1, assigns P1 and the three children as its parents, and ends up sending all its packets to nodes that cannot get the messages closer to the base station.

This loop causes messages to be jockeyed back and forth between two nodes who list each other as their parents. To prevent this situation from occurring, receiving a packet from a node that is listed as a parent will trigger a *loop_notification()* message. This message will set to null the hop distance of any nodes involved in the parent loop as well as all of their children (since every node that sets its hop distance to null must also rebroadcast the message). Since loop_notification is the second-highest priority message in the system, processing is expected to be effectively instantaneous - since rebroadcast happens before altering the hop distance, the system assumes that all child nodes have received the message by the time the hop distance has been reset, allowing the node to immediately move forward with more calls. Next, the node begins calling hopscan() multiple times a second until parent nodes have been located and this.hops is no longer null. At this time, the node will send hopscan messages for one more second before resuming message to ensure that a steady state has been reached. Though this process will require a brief service interruption, the updates should cascade down the tree very quickly.

Should hopping calls interrupt loop_notification calls, loop_notification is cancelled and hopping is allowed to correctly configure the nodes.

When a new node becomes active in the system by turning on after registration with the base station, it should use hopscan to assign itself the correct hop number.

2.3 Node features

2.3.1 Blacklists

Each node will save two blacklists (one for images, one for locations) of nodes it should not send to - these lists will be populated if a predetermined number of sent messages to a particular node do not elicit an acknowledgement response within one second. This time frame is significantly less than 30 seconds so that an entire sending interval is not wasted sending messages to an unresponsive node. Nodes will be added to the blacklist based on a formula that takes into consideration the following factors:

- *avail_parents,* number of non-blacklisted parent nodes: Nodes will be more likely to be blacklisted when many other nodes are available to receive messages. This number ranges from 0 to P, where P is the total number of parent nodes.

- *fitness*, fitness of receiving node: If there are many other better or equivalent receiving nodes, the node will be blacklisted more readily. This is a number from 0 to 100, equal to score(node) or im_score(node), described in the **Routing Protocols** section.

The formula relates the maximum number of lost messages allowed (lost_max) to the parameters described above:

- lost_max(node P1) = c5* (c6*imq_len + locq_len) * (fitness/100) / (avail_parents/P)

- P is the total number of parent nodes, and c5 is a constant from 0 to 1. Initially, we will try c5 = 0.2. c5 represents the acceptable loss rate. c6 represents the average number of packets per image, which is a positive number on the order of 10.

- The formula is a product because if the fitness, number of available parents, or total queue length is doubled, the lost_max should be affected by a factor of 2.

- A greater absolute number of lost packets is permitted in nodes that are expected to have received more traffic (higher fitness).

Information about blacklisting due to full image queues is found in the **Queues** section. The two blacklists will usually be the same, with occasional variation occurring due to full image queues.

2.3.2 Headers

A message header includes the node ID which originally produced the message, the sending node, the node which the message is intended for, and the timestamp of when it left that node. An image message also includes the packet number. For instance, if an image was broken into 10 packets, the third packet would include “packet 3 of 10.” The size of the header will be no more than 20 bytes (4 bytes/piece of information), which is negligible compared to the size of a packet (~1500 bytes).

2.3.3 Queues

Each node has an image queue which stores up to 200 images and a separate location queue. The location queue contains location messages and their timestamps of entry into the queue. The capacity of the location queue is virtually unlimited because it uses a negligible amount of disk space.

When a node receives a message from a trusted node, it adds the message to the appropriate queue for forwarding. The header of the message is decrypted, but the body remains encrypted - this enables the node to quickly access important information about the message without taking the time to decrypt the entire message. If a message of the same type (image or location) originating from the same node is already present in the queue, the old message is replaced with the new one. In the case of images, the queue stores up to 200 complete images. If, for instance, packet 1 of 7 for Image A is received, the receiving node waits for up to 7 seconds (equal to the total number of packets) to receive the remaining packets. If it does not receive them, it discards the incomplete image and rejects any future packets of Image A. If it does receive all 7 packets, any older image from the same sending node which is still in the queue is deleted. Messages that do not replace another message from the same original node are always added to the end of the queue.

If the image queue is full, the node will broadcast a *full_queue_broadcast*(), which will be received by all nodes that list this node as their parent. Those nodes will immediately add the parent node to their blacklist for image messages. This case is the only time the image and location blacklists differ.

2.3.4 Parent node list and failure counters

As mentioned in other sections, the parent nodes for each 30-second period are saved in each node's memory. The parent nodes are saved as a list of tuples containing the node, loss rate, and number of failed messages attempted to be sent to the parent node: (*node, loss_rate, failed_messages*). If a parent node gets blacklisted, it is removed from the parent list and moved to both blacklists as a (*node, loss_rate*) tuple. If a node is blacklisted for images only, it remains in the parent list and is simply copied to the images blacklist. The list of parent nodes is retained until the next period's list has been fully developed after a hopscan() command, and is then discarded.

2.3.5 Processing the queue

When a node sends a message to another node, it re-encrypts the message header and uses the secured *send* command to send the message to a specific node as determined by the routing protocols. The message is not immediately removed from the queue; instead, a pending flag is set to *true* to signify that the message is being processed. Once an ACK notification is received, the message is removed from the queue. If an ACK notification is not received within a second, the pending flag is reset to *false* to trigger another *send* attempt. The failure counter for the unresponsive node -- which tracks the number of failed messages sent to that node -- is incremented. If more than 5 attempts are made or the intended recipient node becomes blacklisted, traffic is redistributed and the message will be sent to another node.

2.3.6 Losing connection

If a node has no nodes around it, the *hopscan* operation is called continuously until a connection is regained. The hop distance for the node is set to null so that the correct parent-child relationship will be established when the node comes in contact with the network again.

2.4 Routing Protocols

2.4.1 Prioritizing queues

The location queue is always prioritized over the image queue, so no images will be sent while there are location messages in the location queue.

2.4.2 Routing location information

When the node runs its hopscan every 30 seconds, it decides where to forward messages for the next 30 seconds.

Currently, only non-blacklisted parent nodes will receive messages; a more developed system may be able to send message to sibling nodes in times of high congestion.

Once the parent nodes are obtained, a scoring function assigns integer values to each parent node such that the scores sum to 100. (We assume that a node will not be within range of more than 100 parent nodes, but this number can be altered if necessary.) A higher score indicates the node is more fit. A node can be assigned a fitness score of 0. The scoring function depends on the following parameters:

- *loss_rate*, loss rate: Nodes with a lower loss rate are more fit, and will therefore receive more traffic. This is a number from 0 to 1.

- *bl_stat*, blacklist status: Parents that were blacklisted in the last time period are less fit, and therefore receive much less traffic. This negative weight assigns two purposes: to avoid wasting excessive traffic on a failing node, and to exploit the blacklist function's features to trigger a blacklist with a lower number of failed sends. *bl_stat* is 1 if the node was blacklisted in the last period, 0 otherwise.

- *bl_count*, blacklist status over time: If a node has been blacklisted multiple times in the past 10 minutes, it is much less fit. This is a number from 0 to 20, counting how many times the node was blacklisted in the past 10 minutes (20 periods).

The scoring function *score* is as follows:

- *raw_score*(node N) = c1*(1-*loss_rate*) + c2*(1-*bl_stat*) + c3*(20-*bl_count*)/20

- c1, c2, and c3 are constants between 0 and 1. The values will be optimized by experimentation. Initially, we will try c1 = 1.0, c2 = 0.3, and c3 = 0.7. This means the fitness score will be an equal balance of loss rate and blacklist history, with the blacklist history having an emphasis on the current blacklist status.

- *score*(node N) = 100 * *raw_score*(node N) / SUM(*raw_score*(node k) for all parent nodes k)

If a node has a fitness score of 0, it is *not* added to the blacklist for this time period - it is simply not sent any traffic. This decision ensures that nodes are given a chance to recover from temporary problems.

The fitness scores are used to distribute the load of messages over multiple nodes. The times at which the messages are sent are also distributed evenly. For instance, if one node will receive

50% of outgoing location traffic, it receives every other location message to distribute the load evenly over time.

2.4.3 Routing images

Images are broken into 1500-KB packets. Each packet for a given image is sent to the same node with "packet x of y" in the message header, where x <= y are positive integers. If the receiving node does not receive all y packets of the image within 10 seconds, the image is considered to have been lost. y is expected to be about 10, depending on the quality of the images.
Image packets are routed in a manner nearly identical to that of location messages, but the following parameter is also considered:

- *imq_len*, length of image queue: Nodes that have longer image queues receive a lower fitness score. Note that this parameter does not have a linear effect - the impact of a long queue is weighted more heavily than the impact of a short queue. This relationship will ensure that a node that clears its image queue does not become overwhelmed with an influx of messages in the next time period. This is a number from 0 to 200.

The scoring function *im_score* is as follows:

- *raw_im_score*(node N) = c1*(1-*loss_rate*) + c2*(1-*bl_stat*) + c3*(20-*bl_count*)/20
  + c4*(1 - log(201-imq_len)/log(201))

- The function uses the log of the image queue length because as imq_len approaches 200, it becomes more important empty it.

- c1, c2, c3, and c4 are constants between 0 and 1. Again, the values will be optimized by experimentation. Initially, we will try c1 = 1.0, c2 = 0.3, and c3 = 0.7, c4=1. This means the fitness score will be an equal balance of loss rate, blacklist history, and image queue length.

- *score*(node N) = 100 * *raw_score*(node N) / SUM(*raw_score*(node k) for all parent nodes k)

3. Analysis

The system will be evaluated in terms of robustness, reliability and security. Other performance factors will also be taken into consideration during the analysis.

3.1 Use of resources

Overall, the system is efficient in terms of resources. There are, however, some cases where excessive resources are needed to deliver a packet to its destination.

When the network configuration is static, hopscan and hopping messages are not necessary. If the nodes are moving around, non-continuous updating of network configuration can lead to some less-efficient paths being chosen using outdated information. Loops and other complex network changes can also require temporary service interruptions and network resources to be resolved.

Although the system may use more resources than necessary in few cases, it is still robust, reliable and secure.

3.2 Informing the base station of the location of first responders

There are situations in which the system may fail to inform the base station of the location of a first responder within 5 minutes. These situations will occur due to the loss of a parent node, which can happen for a number of reasons:
1. The parent node moves.
2. The node itself moves.
3. The parent becomes the node's child, which takes longer to resolve.

High network congestion could also slow the transmission of messages to the base station, especially if the origin node is very far away. The first responder could also have been disconnected from the network, which is an appropriate situation to elicit an alert.

These situations mostly affect the robustness and reliability of the system, while they do not pose evident threats to the network's security.

3.3 Attacks from malicious nodes

Malicious nodes can attack the system in many ways. The analysis will consider two attack scenarios.

In the first scenario, a malicious node in the system tries to prevent a first responder's message from reaching the base station. To do so, the node would have to get the private key to intercept or redirect messages. Otherwise, all it can do is flood the network with messages that interfere with the valid message. The security protocol of the system will prevent the malicious node from getting the private key. As a result, the first responder's message will reach the base station. However, the flood of messages in the network may delay the receipt of the message.

In the second scenario, a malicious node tries to mount a replay attack. To be more specific, the malicious node hears a legitimate message, and then broadcasts 1 million copies of the message pretending it received it from the source. In this case, it would be a problem if the legitimate nodes of the network picked up those messages and forwarded them to the base station. However, the implementation of the design already accounts for the replay attack case by encrypting the sender ID. If the unencrypted sender ID does not match where the message actually came from, the sending node will be declared malicious and added to a permanent blacklist so any future communications from it will be ignored.

The analysis of the attack scenarios shows that the security policy of the system accounts for various attacks from malicious nodes. The robustness and reliability of the system are not affected.

3.4 Traffic behaviors

3.4.1 Behavior under a no-traffic scenario

Under a no-traffic scenario, no information is to be forwarded in the network. The nodes simply call hopscan every 30 seconds to attempt to maintain accurate information about the network configuration for when traffic resumes.

Devices can also be turned off when not in use, though this practice is discouraged, as any device may be a necessary link to the base station for others.

The system is robust, reliable and secure under the no-traffic scenario.

3.4.2 Behavior under a light traffic scenario

The system will run smoothly under a light traffic scenario, as long as there is a sufficient density of first responders to allow each to have a path to the base station.

The system's robustness, reliability and safety are maintained under the light traffic scenario.

3.4.3 Behavior under a heavy traffic scenario

The heavy traffic scenario is more problematic for images, because their priority will keep getting lower. Some location messages might get delayed in queues, but they will still be sent pretty quickly. There might also be additional complications of nodes competing with each other to send messages, resulting in long waits for network resources. However, one of the assumption for the design was no inference or little chance of inference happening.

The security of the system is not affected by the heavy traffic. Under the system's assumptions, the effect on robustness and reliability are also minimal. Heavy traffic may, however, cause delays in sending and receiving messages.

3.5 Interference

The system will be analyzed for interference in the case where a node hears a message that was not intended for it. There are two scenarios to consider.

The first scenario considers the case where a route goes from node i to node j to node k to the base station. Although the probability is small, it might happen that when node i transmits a message, node k hears the message directly. The issue to consider is what node k will do with the message. However, because of the way the system is implemented, this will not be an issue for the system.

The system is designed so that there are no "paths" known by the node. The design does not use the broadcast functionality, and nodes are not configured to respond to send messages not intended for them. Node k will not receive the message unless the message was addressed to k. It must also be noted that nodes know their parents but not their children. As a result, node k does not keep track of the fact that node i is farther from the base station, except implicitly, because node i is not k's parent.

The robustness, reliability and security of the design are not affected. However, there's an implication to consider. Because the design is implemented as a solution that operates at the node level, the design does not always return the shortest path. That is, nodes do not know much about other nodes. The design is meant to be consistent and dependable, but is not designed for optimal speed.

The second scenario considers the case where a trusted node very close to the destination (or even the base station) hears a message that was not intended for it. With the current implementation, nothing unexpected happens in the system. Nodes near the base station operate like all other nodes. Nodes will only interact with messages that are intended for them. In this scenario, the robustness, reliability and security of the system are not affected.

3.6 Scalability

As the number of nodes increases, the most likely problem is congestion. Since it is impossible for a node to send a message without broadcasting it to all the nodes that are within radio range, every node will have to interact with more messages. In a smaller system, the interactions are negligible, because for most messages the node simply decrypts the header, finds that the message is not intended for it, and discards the message. However, as the size of the system increases, this additional traffic and decryption begins to use a non-negligible amount of time and memory.

As the system becomes more congested, queues may start to fill up faster than they can be emptied. If this happens, image messages will fail to be sent because the location messages always take priority. When image messages are sent, they may be incomplete (missing one or more packets) or they may be delayed to the extent that they are no longer relevant. However, we do not anticipate that any reasonable-sized system will be affected by these issues.

4. Conclusion

With encryption identifying trusted network nodes, first responders can rest assured that their communication is protected. Their devices will determine their hop distance from the base station in an efficient yet reliable manner. Prioritization of location messages ensures that GPS data will be updated as frequently as possible. Image distribution maximizes throughput while minimizing the amount of resources wasted on images that may be unsalvageable. An efficient distribution of traffic leverages network resources. This ad-hoc network will fulfill the needs of first responders by balancing efficiency with agility to adapt to a changing layout with high throughput needs.

There are further modifications that can be done to achieve better performance. The routing protocol can be modified to handle loops and complex network changes more efficiently in term of resources. Extra measures can be taken to ensure no packet loss when a node loses its connection to the parent node. The design can be improved to allow for optimal speed.
Before scaling the system, further changes need to be done to handle congestion.

5. Acknowledgements: Jorge Simosa

Word count: 5097