1 Introduction

When deployed to a disaster-stricken area, first responders need to be able to efficiently communicate their location and other information to their base camp. However, communication infrastructure is often unreliable or inaccessible during these crises. To solve this problem, we designed AdHocPro, an ad-hoc wireless network of portable devices that allows responders to send messages to their base camp in a reliable, efficient, and safe way. Our rigorous design accounts for the rapid changes in network topology that occur as the responders move around, as well as the possible presence of malicious agents in the field.

1.1 Design Overview

Our design is built on top of 802.11 MAC and uses TCP in order return ACKs when a node successfully receives a packet from another node.

Because wireless links suffer from interference if multiple nodes attempt to transmit at the same time, 802.11 MAC allows our nodes to sense the medium and transmit only if the medium is idle. ...MORE 802.11 IS GOOD STUFF HERE...(TODO)

Our link-state based routing protocol enables each first responder to send messages to the base anytime there is a sequence of connected nodes between the sender and the base station. The link-state advertisements sent by our routing protocol from each node at every advertisement interval of 30 seconds will contain the node's GPS information, reliably delivering the location information of each node to the base station no more than 5 minutes after the base previously learned the location of that node.

A cost metric, calculated by the integration step of our routing protocol, enables our design to measure the effectiveness of each possible path to the base station and choose paths which maximize the image throughput of the network. The design also draws upon elements of data center TCP (DCTCP) in order to relay information regarding queues at each node to minimize the effects of network congestion when there are many image messages to be delivered to the base station.

The final component of our design is an authentication system. Our design seeks to accept messages only from first responders, and should not forward messages from non-trusted nodes of include such nodes in its routes. To this end, the system uses a public/private key signature scheme, where each node has a private key which it uses to sign its messages. Other nodes may then decode the message using the sender's public key, but even if a non-trusted node hears and decodes a message using the public key, it will be unable to sign any messages without the private key of a known node. A sequence number attached to each message allows nodes to discard messages which they have previously heard, stymying replay attacks from malicious sources.

1.2 Tradeoffs and Design Decisions

The central design consideration of AdHocPro is efficiency. System components are designed to provide functionality which contributes to multiple system requirements while maintaining relative simplicity. This is emphasized in design decisions because of the number of the large number of different use cases, network topologies, and edge case scenarios which ad-hoc wireless networks experience. If these scenarios can all be accounted for without introducing additional components, our system will be more robust. A secondary design emphasis was placed on correctness. If given the choice between a less computationally expensive but possibly incorrect metric and a computation-heavy but likely correct metric, AdHocPro will use the latter. This is because every packet sent adds to network congestion, and mistakes are therefore very costly.

As a result of these decisions, link-state was chosen over distance vector as AdHocPro's routing protocol. Link-state does come with a higher initial setup cost than distance vector as a greater number of advertisements must flood the entire network. The overall bandwidth and time consumed by this operation is relatively high. However, link-state gives each node a complete view of the network. If a link a packet's original path fails, any node along the path may recompute a new path for a packet to take to the base. In distance vector, a node only knows of the best path to take to a destination. Therefore if a link fails, the packet will stall until the next advertisement window.

COST METRIC TRADEOFFS (TODO)

The authentification system was designed with an emphasis on security and simplicity. Each node signs its own messages with its private key, and the resulting ciphertext can be decrypted by any node that has the public key. Therefore, only trusted nodes will be able to compose and forward messages, but nodes may still use the BROADCAST function as all of its neighbors, including non-trusted nodes, are capable of deciphering its messages. Preventing non-trusted nodes from viewing messages is not a system requirement, and would also complicate our security protocol, so we decided not to support this.

2 Design

2.1 Routing protocol

An effective and efficient routing protocol is necessary for nodes to determine the best paths to send packets to the base without constantly congesting the network.

Our routing protocol is based on a link-state advertisement scheme in which neighboring nodes inform each other of incremental changes. Upon receiving an advertisement, a node will use the BROADCAST function to forward it to all its neighbors. As a result of this flooding process, each node's routing table will contain a complete map of the network. Then, each node will independently run a computation based on our cost metric to find the shortest routes to the base. As long as the nodes have a consistent view of the topology and the same metric, resulting routes at different nodes will correspond to a valid path.

2.1.1 Location information

Because the link-state routing protocol floods network updates across all nodes in the network, an advertisement from every single node will reach the base station. Thus, location updates can be built into the system's routing protocol itself. Location information for each node is added to its advertisements, so once the link-state protocol converges, the base station is guaranteed to know the location of every node reachable from the base.

2.1.2 Data structures

There are two data structures which our routing protocol uses: *link-state advertisements* (LSAs) and *network tables.*

Link-state advertisements have the following format:

$$[\text{nodeID}, \text{GPS}, \text{lsaseq}, (nbhr1, lossprob1), (nbhr2, lossprob2), (nbhr3, lossprob3), ...]$$

Where *nodeID* is the ID of the node, each *nhbr* is a current neighbor of the node, *lossprob* is that neighbor's corresponding loss probability, and *GPS* is the current location reading. Additionally, each LSA has a sequence number, *lsaseq* that starts at 0 when the node turns on and increments by 1 every time the node issues an LSA. When a node receives an LSA that originated at another node, n, it first checks the sequence number of the last LSA from n. If the current sequence number is greater than the saved value for n, then the node re-broadcasts the LSA to its neighbors, and updates the saved value. Otherwise, it discards the LSA, because it must have received a more recent LSA from that node.

Network tables: The network table must store LSAs issued from every node in the network. This table will enable a node to reconstruct the entire network and run a cost-finding algorithm in order to determine the best path a packet should take to reach the base. (TODO: FIGURES)

2.1.3 Updating the network topology

Initially, the network undergoes the following procedure that discovers the network topology:

1. Each node constructs its link-state advertisement by calling SCAN.
2. Nodes begin to flood the network with their LSAs, and build up their routing tables based on the advertisements they receive.
3. Once all the LSAs have been discarded according to the routing table protocol, each node will have a complete map of the network. This will take time proportional to the diameter of the network, because LSAs must be propagated throughout the entire network.

Every 30 seconds (based on each node's internal clock), all nodes in the network will call SCAN in order to determine whether or not the network topology and success probabilities of paths have changed.If there are changes, nodes that are affected will construct LSAs accordingly and send these advertisements into the network. Nodes that have not had major changes in status will not need to create new advertisements. Resultingly, in a scenario in which not all nodes have changed their position significantly over the course of 30 seconds, the network utilization of an update is smaller than the network utilization of the initial network setup. Less network congestion implies that this update procedure takes less time to complete than the setup.

There exist, however, some cases in which the state of the network changes dramatically in between network update intervals, and it is desirable to send LSAs as soon as we recognize a change. These situations are as follows:

1. A path between two nodes fails, even though both are still connected to the network via other paths
2. A path is added to the network without any new nodes joining the network
3. A node becomes disconnected from the network
4. A node is re-connected or added to the network
5. The loss probability of a path changes dramatically, significantly affecting the cost of routes which go through the path

In order to determine if one of these situations has occurred, a node will issue a SCAN whenever it encounters one of a few anomalous scenarios:

1. A node attempts to send a packet down a link, but experiences (TO BE DETERMINED NUMBER) consecutive timeouts without a successful send.

2. A node successfully sends packets down a link (TO BE DETERMINED NUMBER) times without experiencing a single timeout

3. A node hears a scan from a node that is not a neighbor in its view of the network topology

If one or more neighbor nodes from the last time the node scanned no longer appear in the SCAN results, either paths between nodes have been lost or at least one node has been disconnected from the network. If the SCAN returns no results, then the node knows that it has been disconnected from the network. If the node was not disconnected from the network but former neighbors are missing, the node will send an LSA which will propagate throughout the entire network. Similarly, if the scan returns the same neighbor nodes as before but one or more success probabilities have changed significantly, or if there are new neighbor nodes, the node will send an LSA.

This protocol will allow the network to update under any of the five situations in which the state of the network changes in between network update intervals. One final addition to the protocol pertains to disconnected nodes. When a node is disconnected from the network but wishes to join it, it will issue a SCAN every 5 seconds. This will fully connect these nodes to the network as soon as a connected node hears a scan.

2.2 Throughput for Sending Images

When the situation calls for the aid of first responders, it is important that images taken by these have the highest throughput possible when sending to the base. The algorithm described in this section is the algorithm for routing images. Since advertisements (containing the GPS information of nodes) and new public-key packets have higher priority than images, this algorithm will only start sending image packets, when the other priority queue is empty and the images queue is not empty. Note that one packet cannot hold a full image, so images will be tried to be decided in different packets such that as much information can be fit into one packet while still leaving some small portion for appending information to that packet.

2.3 Throughput Metric and the Routing Table

After the routing protocol has finished its update period, it will start the integration phase yield an updated graph for the current state of the network topology. The cost in each edge will be $1/p_i$, where $p_i = 1 - p_{loss}$ is the probability of success of sending to an adjacent node. The cost of an edge corresponds to the expected number of packets that a node has to send to yield one successful transmission. Once it has the graph it will construct a routing table containing the following information:

1. The best cost path by taking the link going to adjacent node j. These costs will be considered during times of congestion.

2. The Explicit Congestion Notification Bit (ECNB), which takes value 1 if going to node j takes us to a path that is considered congested or 0 otherwise.

The cost of path is defined by the following equation:

$$Total\ Expected\ Transmition\ Cost = \sum_{i=0}^{k} \frac{1}{p_i}$$

It's important to realize that we are seeking to minimize this metric.

To fill out the table we do the following:

1. Do Dijkstra's single shortest path from the current node.
2. Insert to the table the shortest path to the base on the entry where the node's own number is located.
3. Then run Dijkra's on every neighbor node to the current node. Notate the current values obtained be d_i.
4. Then, insert for node i the value d_i plus the cost of taking the link connecting the current node to the adjacent node. (Note that if current node number = i, then it just insert the minimum cost path from itself).

Consider the following table as an example:

Table 1: Routing Table

Node Number	Total Expectet Transmittion Cost [TETC]	Explicit Congention Notification Bit [ECNB]
0 [base]	19	0
1	29	1
2	31	0
3	17	1

2.4 Throughput algorithm

When sending images to the base, our goal is to try to send at maximum throughput with the current knowledge of the network. Therefore, in times of congestion the goal of our algorithm is to try to route around congestion, while still trying to attaining the maximum throughput with the current state of the network. Note that it will use the Explicit Congestion Notification Bit [ECNB] sent by adjacent nodes, to be aware of its own congestion. This algorithm will also mark image packets that are being trying to route around congestion, such that they actually take a different route than through the congested node.

Recall that we only send images when the priority queue for advertisments and public-keys is empty. The routing algorithm is as following:

1. First try sending a contention window of image packets to the best path if its ECNB bit is 0 and the packet is not marked that it is being re-routed.
2. If the current packet is being ??????????

2.5 Authentication

When designing a communication system for first responders, its important that first responders are able to trust the messages they are receiving from fellow first responders. Therefore it is important to establish a mechanism by which first responders can authenticate benign messages (i.e. messages not sent by a malicious adversary). In the protocol presented in this paper, we use the RSA public-key cryptosystem to generate private and public-keys that will be used to authenticate messages. From here on, we use the term node and first responders interchangeably.

2.6 Initialization

During initialization, each node is assigned a public/private key pair by the base. Each node constructs a table with the following information for every node, and distributes it across all the nodes:

1. The node's public-key.
2. The node's most recent message ID, initially 0.

2.6.1 The Table for Authentication

Recall that everyone will have a table with an entry for each ally node. The table will contain two important numbers for each node: the node's public-key and the node's latest message ID number. The latest message ID number is a counter-like number inserted to every message a node sends such that each node can idenfity duplicate messages and be protected from malicious nodes trying to plot replay attacks. Consider Table 1 for a depiction of this table:

Table 2: Authentication Table

First Responder ID	Public-Key	Latest Message ID
0 [base]	69	669
1	50	837
2	47	877
3	45	300

2.6.2 Signing Mechanism

Nodes should accept messages only from fellow first responders. To achieve this security, nodes sign their messages with their private-key. Consider the following function that signs messages:

$$\text{SIGN}(message, my\_private\_key, new\_message\_ID) = \sigma_A$$

The function returns a binary string σ_A, corresponding to the output of signing a message with the node's secret key. Since each private key assigned for each node is unique, each other node will need the corresponding public key to verify the signature. Since we also want to be protected from replay attacks, the message ID is incremented for each sent message.

2.7 Authentication Mechanism

Nodes can verify the validity of a message by using the following verification function:

$$\text{VERIFY}(signature, cooresponding\_public\_key) = b$$

The output of VERIFY will be the boolean true if the verification passes with that public-key or false if it fails. If the verification fails, then the message will be dropped and will not be forwarded. The receiving node uses the Authentication Table to decide which public key to use in order to authenticate a message. When forwarding messages, nodes re-sign the contents of the message with their own private key.

Once the authentication has passed, then the node will use the following function to get the message:

$$\text{GET\_MESSAGE}(message, corresponding\_public\_key) = m$$

This function returns the corresponding message when applying the inverse of the sign function using the appropriate public-key.

2.7.1 Dealing with replay attacks

Another problem arises when a malicious node overhears a properly signed message and attempts to flood the network with that message. This is a specific man-in-the-middle attack known as a replay attack. Our protocol prevents this by maintaining a counter, $message_ID$, at each node. This counter is increased before a message is sent. Therefore the format of a message looks like this:

$$message = (message\_information,\ message\_ID)$$

After VERIFY validates that the current message indeed was sent at one point by a trusted node, it will then check that the message ID number is not outdated. This is accomplished by the following procedure (given the node number and message ID):

```
$def\ check\_ID\_number(node\_number, current\_message\_ID)$
    $latest\_ID\_number\_from\_table = this.authentication\_table[node\_number].get\_Latest\_Message\_ID();$
    $if\ (current\_message.ID > latest\_ID\_number\_from\_table):$
        $this.authentication\_table[node\_number].update\_Latest\_Message\_ID(current\_message\_ID)$
        $return\ true$
    $else:$
        $return\ false$
```

2.7.2 Scalability and Updating the Authentication Table

It is possible that a new node will be added to the network after the deployment. Therefore, it is important that other nodes are able to add new these new nodes to their Authentication Table.

We address this by using the base as a trusted authority. The new node is required to report to the base to obtain its own public and private key, as well as all of the public keys of the nodes already in the network.

After this step, the base sends a special message (that it signs) and distributes the new public-key across the network. Then, the nodes forward the message, re-signing it each time, until every node has inserted the new public key of the new node into their table.

3 Analysis

3.1 Authentication

Under the assumption that the base and legitimate nodes are all trusted, the authentication system presented is extremely secure. This is because during the initialization phase, everyone obtained a secure key from the trusted authority (the base). Therefore, if no new first responders join the network, every node in the network so far will be able to verify and sign every message that it receives or sends.

However, we can not ignore the potential scalability problem of this initial design. Therefore, our design also accounts for adding new nodes to the network. As explained above, these new nodes undergo a similar procedure for obtaining keys from the trusted base. The rest of the nodes will not accept messages from this node until the base distributes the new node's public key. The old nodes will trust this new key, as the message containing it was signed by a trusted key.

Since the private keys are generated with RSA, the probability that a malicious node guesses a key and is able to find a colliding signature is negligible (given the intractability of factoring large numbers). The keys used for RSA are usually of length 2048 bits, so the probability that a key generated by a malicious node a collides with an already generated private key is 2^{-2048}. This case is extremely unlikely, but in a huge network where this could be a problem, the number of bits used for key generation could be increased.

The authentication mechanism presented is also secure against replay attacks. Since the message ID is included in the message signature, it is not enough for a malicious node to simply guess an incremented message ID, as it would have to resign the message accounting for the new key. This is unfeasable, given that the malicious node can not guess the nodes' private keys.

4 Conclusion

5 References