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 design 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

1.2 Tradeoffs and Design Decisions

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.

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

Link-state advertisements have the following format:

Routing tables:

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.

2.1.1 Updating the network topology

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 Cost algorithm

2.3 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, the base will use the

RSA public-key cryptosystem to generate private and public-keys that it will then distribute to the first responders. In addition, everyone will have a table with an entry for each fellow first responder containing two important numbers:

1. The first responder's public-key.
2. The first responder's most recent message ID.

From here on, we will use the term node and first responders interchangeably.

2.4 Initilization

At the initialization phase of the Ad-hoc network, all nodes are required that they first report to the base before departing. At this step of the protocol, each of the nodes will be assigned their own private-key. Since at this step of the protocol we know exactly which nodes are going to be at the initial newtork, then every node will be able to have an initial table with everyone's public-key. This will be quintessential for nodes' capability of verifying messages.

2.4.1 The Table for Authentication

Recall that everyone will have a table with an entry for each ally node. The table will containing 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 ruplicate messages and be protected from malicious nodes trying to plot replay attacks. Consider Table 1 for a depiction of this table:

Table 1: Athentication Table

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

2.4.2 Signing Mechanism

The system should accept messages only from fellow first responders. Therefore it will be very important that nodes sign their messages with their private-key. Consider the following function that signs messages:

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

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. Note also that fi we want to be protected from replay attacks, a new message ID number has to be inserted to the message before signing it.

2.5 Authentication Mechanism

Even though it is important to sign messages for fellow nodes, its equally important that nodes are also able to verify the messages they receive. Therefore, consider the following verification function that takes a specific public-key and verifies if the chosen public-key matches the private-key used to sign the message:

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

Notice that when we say, matches, we mean that verify is indeed the inverse function of sign when the correct public-key is applied relative to the correct private-key used to sign the message. The output of VERIFY will be either, the boolean true if the verification passes with that public-key, or false if it doesn't. If the verification fails, then the message will not be forwarded.

The way that a node decides which public-keys to apply is simple, since it knows it neighboring nodes, it indexes the Authentication table using each of the node's number and then obtains the corresponding public key. This will work since, when forwarding messages, nodes re-sign the contents of the message. 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 function of the sign function using the correct public-key.

2.5.1 Dealing with replay attacks

It is also a problem when malicious nodes get a valid signed message and then potentially try to flood the network with unnecessary messages. The way that our protocol deals with this problem is that, all messages are marked with a message ID number that is a counter that is matched to a specific node. This counter is increased everytime before a message is sent suring the signed procedure. Therefore the format of a message looks like this:

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

After VERIFY validates that the current message indeed was sent at one point by an allied node, it will then check that the message ID number is not outdated and thus invalid. The way to check this is by feeding the node number that made VERIFY accept along with the current ID message to the following procedure:

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

2.5.2 Scalability and updating the Authentication table

After initialization, it is also possible that a new node will be added to the network. Therefore, it is important that other nodes are able to add new these new nodes to their Authentication table.

The way that this issue is addressed is by using the base as a trusted authority. The new node is required to report to the base and from their obtain all of the public-keys of the node so far in the network and also obtain his own public and private-key.

After this the base will send a special message (that he will sign) and distribute the new public-key to the network. Then the nodes will forward the message and re-sign it and then everyone will insert the new public-key of the new node to the network. Note that the base can also sign the new pair of keys for the new node and then the node can distribute them himself with sign message if neccessary.

Jamming is a always a potential problem

3 Analysis

3.1 Authentication

Under the assumption that the base and nodes are not going to be overrun, the authentication system presented is extremely secure. The reason that this is the case is because, at 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. This is great for networks where we have a fixed number of nodes in the network.

However, we can not ignore the potential scalability problem of this initial design. Therefore, the design presented in this paper also addresses the issue of adding new nodes to the network. If new nodes to the network are added, then they have to report to the base. After this step, they are able to get secure keys from the trusted authority. Since these key are bran new to the network, the rest of the nodes will not accept signatures from this node until the base distributes the new public key. Therefore, the base will forward the new public key to neighbour nodes and if neccesary, the new node can also forward this message signed by the base. Therefore, now the whole network can be updated with the new public 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. This statement holds even when the node is able to see valid signatures because its extremely hard to obtain the private-key. This is true if one believes that factoring large numbers is an intractable problem with the present knowledge in computer science. Furthermore, even if a malicious node generates his own private-key, nodes are extremely unlikely to accept any messages that malicious node could sign. The only case where a malicious node could obtain a valid private-key, is if the key that he generated collides with a private-key that was already assigned. The probability of this even is extremely unlikely. The keys used for RSA are usually of length 2048 bits, therefore, the probability that one key generates by a malicious node a collides with an already generated private key is 2^{2048}. The likelihood of collisions starts being a problem only when the graph start getting exponentially large. This case is extremely unlikely and even if this is the case, the number of bits for keys could just be increased, making RSA again more secure.

The authentication mechanism presented is also secure against replay attacks. If a malicious node tries to forward a valid message, first responders will reject the message (or messages), because they will extract the message ID and compare it with the latest message ID they have received. If the message ID is outdated, then first responders will not forward during replay attacks.

If one wanted to relax the assumption that nodes or the base could be overrun and therefore some or all the keys in the network could be stolen one can make some improvments to this design. Note that these improvements may add to the complexity of the protocol.

If nodes could be hacked, one improvement that could be done is that if the figures out that it was hacked, it can report this to the base by going physically to the base and then the base can re-assign keys and distribute the change in keys.

However, if the base is compromised a different strategy has to take place. One possible stategy is that every 30 minutes or hour or some specific time interval, every node reports to the base and all the public and private-keys are re-generated. This way the node has would be forced to re-hack the base if we wants to be succesful again, and hacking the base multiple times should be difficult.

4 Conclusion

5 References