December 15 - Project 13

MAdHoc - Mobile Ad Hoc Network Messaging App

Software Design Document

Group Members

Cole Cummings

Cody Lougee

Ethan Niemeyer

Advisor

George Amariucai

Client

George Amariucai

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This design document describes the architecture and system design of a mobile ad hoc network using modified DSR protocol and encryption techniques. This document is for explaining the processes which we will take when implementing our Apple iOS app.

1.2 Scope

The goal of our project is to build a messaging application for mobile devices that supports text messaging communications in the absence of a standard wireless network. The main objective is to allow secure and encrypted communication between specific users because this seems to be missing from similar projects. The main benefits would be its usefulness in times when traditional networks fail, for instance, natural disasters may destroy infrastructure like cell towers but responders need to be able to communicate. Some other applications have also been used at times when the government restricted access to traditional internet. Firechat was used in Iraq and Hong Kong protests, but the communications were not encrypted.

1.3 Overview

This document is meant to provide the software development team with guidance to each layer of the application architecture. It is organized by software layers and broken into three layers. The system architecture describes program structure, sub system interfaces, and data flow throughout the application. The data design describes data structure, data organizations, data storage and relationships between data needed. The user interface design describes the visual design and functional design of the interface that the user will interact with. In this case, all section of the document are relative to an iOS mobile application.

1.4 Definitions and Acronyms

Node - Each device connected to a network that transmits data

Swift - Programming language used to write iOS apps.

iOS - An operating system used for mobile devices manufactured by Apple Inc.

Multipeer Connectivity Framework - The iOS library for peer to peer networking between mobile devices.

DSR - Dynamic Source Routing. This is a protocol for ad hoc networks that forms a route on demand when a transmitting node requests one. It uses source routing at each intermediate device.

UI - User interface and visual design.

UX - User experience and functional design.

Ad hoc Network - A network where every node on the network has the same status and is free to associate with any other node on the network.

Network Packet - Formatted unit of data carried/transferred over a network.

Routing Table - Each node keeps one of these tables of known routes to other nodes. This is updated as the node receives heartbeats.

Heartbeat - Each node sends out a periodic message, when received the nodes add the path to the origin and propagates the heartbeat. If a heartbeat from a node is not received within a timeframe, the target node is removed from the list of peers.

2. SYSTEM OVERVIEW

The mobile app should be able to communicate with other peers through wifi without any previous infrastructure. The pathways that the packets go through should be generated dynamically, allowing users to drop in and out of the network. Data being sent to a node outside of the sender's range should be forwarded by other nodes until reaching the target. When an app starts up it begins broadcasting a heartbeat and receiving a heartbeat from other active users so that it can populate a table of possible contacts without any previous information.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

There are three main components to the app, the UI, the networking logic, and the security logic. These three components work together to facilitate ad hoc communication between mobile devices using the app.

UI

This is the interface through which users use the app. Users will interact with this component to utilize the functionality of the app. Users can switch to different views, send messages, and use the other functions outlined in this document. When a user sends data to another device, the networking component is triggered to actually send the data. Data, like sent and received text messages, will be displayed on the UI.

Networking

This component will handle communication between devices. The responsibilities of this component is to build a routing table for packets, update the routing table when changes in the network are detected, and to send and receive data. This component works in the background on its own to discover other devices and receive data, and sends data when a user chooses to. The security component is used to encrypt and decrypt all communication except for when packets are sent out to discover other devices.

Security

The goal of this component is to make the communication between mobile devices using the app secure. Secure communication is accomplished through an encryption key strategy. Traditionally a public key/private key design is used, but since in an ad hoc network you have no way to transmit keys to another node securely, each node will use the path the message travels between itself and the receiver to generate a key, the receiver will be the only other node in the network with identical paths between itself and the sender, so it will be able to generate the key on its own.

3.2 Decomposition Description

3.2.1 Networking

- Route Discovery
  - Receiving Heartbeat
    - The route table is checked to see if this route has been received earlier from a faster path, if so nothing is done with it. If it is new, go to the next step
    - The Routing table is updated with the path that this came from.
    - The heartbeat has this device's id appended to the list and is sent on.
  - Sending Message
    - Select user from available users to send message
    - Message is broadcasted to all peers, only peers in the given path further propagate the message
    - When the message is received, send a Route Return along the same path
  - Route Return
    - When you receive a message, you send back a message received
    - The message received is broadcasted out, following the given path
    - Once the message received reaches the destination, the message sent is confirmed to have made it.

3.2.2 Security

- Encryption
  - Data sent needs to be encrypted
    - Outgoing data to a person will be encrypted based on their pair of keys
  - Data received needs to be decrypted
    - Incoming data to a person will be decrypted based on their pair of keys
- Keys
  - When enough unique paths to a target node are produced, both the nodes generate a key used to encrypt their data
  - Since you and the target node generate the same keys, there is no need to transfer them over the network
  - You use the same key to decrypt the data

3.3 Design Rationale

The architecture outlined in section 3.1 was chosen because the components represent the three major requirements of the app. Those are, a user interface to actually use the app, a networking component that facilitates an ad hoc network, and a security component that encrypts communication. The style of the user interface is meant to be simple since the main use of the app will be reading messages. Therefore, the UI should not be cluttered in order to make reading and navigation easier. For the networking component, iOS's Multi-peer functionality will be used. Multi-peer offers a platform on which to establish the ad hoc network. For the security component, our key generation strategy will be used since it is a way to securely communicate that does not require trusted nodes like a public/private key strategy.

4. DATA DESIGN

4.1 Data Description

The data in the system will be stored in a CoreData database on a mobile iOS device. All data processing and storage will be done directly on the mobile device. Each user connected in the application will be stored. The application will use the modified DSR protocol for the ad hoc network. We will store each node that any given user has come in contact with within the network. This works through the heartbeat system discussed earlier, heartbeats are periodically sent out and update all currently connected nodes on the identity of other nodes. Any given node has a username and unique identifier. A route request or message is a data object containing media with a target node identifier and sending node identifier.

4.2 Data Dictionary

Route - Represents a path between two nodes. Stores a start identifier, stop identifier, and graph of nodes.

Route Cache - A cache of possible paths between nodes in the network. This will only be relative to the users mobile device. Stores and an array of routes.

Message - Represents a chat message. Stores target identifier, sender identifier, message media, and route traveled.

Heartbeat - special non-chat message sent out periodically to update the Route Cache of other users in the network

User - Represents a user connected to the network. Stores unique identifier and username.

5. HUMAN INTERFACE DESIGN

5.1 Overview of User Interface

All user interaction will be done through the iOS app. The users will be able to define their name, and see those who are near them. The users will also have a friends list and have the ability to chat with their friends and those online. Data will be sent through a submission box on the phone and data received will be displayed in a text box on the phone.

6. REQUIREMENTS MATRIX

6.1 Functional Requirements

- Ad hoc network between mobile devices.
- Encrypted communication.
- A user interface that facilitates the following:
    - Text messaging
    - Picture messaging
- A user can create a friend list of users.
- A user can create a group of users to message.
- A user can send private direct messages.

6.2 Non Functional Requirements

- Routes established within 10 seconds of connecting.
- Heartbeat sent every 5 seconds.
- Routes reestablished correctly when a dropped node’s heartbeat is not received.
- Messages sent to specific users are encrypted if there are enough unique routes to generate keys.

- Each node can connect to up to 8 other nodes

7. Implementation Details

7.1 Basics

The application is written in Swift 2.1 and runs on Apple devices using the operating system iOS 7.0 and newer. Apple’s IDE Xcode was used to develop the application.

7.2 UI

The UI consists of several screens navigated via buttons. The buttons connect up to functions that cause a transition, called segues, to the next screen. A title screen is initially displayed. From there, the user is brought to a sign in screen where a username is chosen. Once a username is chosen, the user can decide to search for groups or friends. Choosing either sends the user to a similar screen that lists groups or users to chat with. Once a chat target is selected, a chat window is brought up. To send a message in the chat window, enter text into the textbox and press the send button.

7.3 Networking

Devices in the network are identified by a unique ID. This ID is generated from Apple’s vendor ID which generates a unique ID. Networking is composed of two layers working together, the Multipeer layer and the ad hoc layer. The first layer is the Multipeer layer. This is a code library in Swift that can create wireless connections between devices and send data between connected devices. The multipeer library only talks with devices it is directly connected to. In order to talk to devices across the network another network layer is needed. The ad hoc layer is on top of the Multipeer layer. The ad hoc layer figures out how to get data to its destination. Messages received contain information about who it is supposed to go to. The ad hoc layer looks up the route to get the data to its recipient and sends it to the next device in the path. Routes to other devices are generated using a heartbeat message. The heartbeat is a broadcast message that contains the route it has taken so far to get to the current device. The device adds its ID to the message and re-broadcasts the message. If the heartbeat has been seen before, discard it.

7.4 Security

After two peers have a number of unique paths between them, they generate automatically generate a unique key. Using this key both nodes will be able to encrypt, and decrypt the data they send to each other over the network. This is accomplished by creating a string using the median node in each of the generated paths. This string will be used as a key to encrypt and decrypt the messages outgoing to only each other.

8. Testing

8.1 Virtual

Using VMware we were able to simulate multiple OSX operating systems (virtual machines) on a pc. Each OSX operating system then simulates an iPhone running our application (1 device to 1 computer is the limit built into OSX). Using the settings in VMware we set up the network so that the middle virtual machine is connected as a man in the middle. In this way, the other two virtual machines have to talk through the middle virtual machine. Using this setup, an ad hoc network can be simulated.

8.2 Physical

Acquired iPads in order to test how the application would work on a physical device, and over wifi without using any infrastructure. In order to have only some of the devices be connected to others, we had to spread out farther than 100 feet.

8.3 Results

Using virtual machines we were able to test the application on 5 devices. Encryption and messaging works on this size of network as expected. With physical devices we were able to test on 3 iPads and text messaging works as expected. On physical devices, the app will sometimes crash if one user enters/exits a chat while another is sending them a message, we think this is a threading issue and are working on it.

9. Appendix I: Operation Manual

9.1 Basic Operation

Operation of the application is straightforward.

1. Have at least two devices within range of each other follow these steps. At least two devices are needed to use the application.
2. Open the MadHoc application
3. A main screen will appear. Press the get started button.
4. If a username was previously chosen, skip this step. Otherwise, a sign in screen is shown. Pick a username and sign in.
5. Press the button to either search for groups or users.
6. A screen with a list of groups or users (depending on what was chosen) will be displayed. Select one.
7. A chat screen will appear. Insert text into the textbox and press the send button to send messages.

10. Appendix II: Alternative Designs

10.1 Android Application to IOS

We were first started to design the application on android devices. We had a quick and simple UI made up before we found out that the android os does not support ad hoc, and that wifi-direct would not be enough for what we needed to do.

10.2 Change in UI

The team member who was initially designing the applications UI left after the first semester. Not having his specialty in UI design, and not having any notes from him meant that the design that we originally had needed to be remade into something that we better understood. These design changes, coupled with our lack of artistic talent, created the current UI we have.

10.3 Update to DSR

The DSR protocol had a few holes in its implementation. The design itself works fine, however it assumes you already know who it is you are sending a message to. Our solution to this problem was to send out a “heartbeat” every so often to let everyone in the network know we are their and still online. This solution ended up covering many of the same problems that DSR attempted to solve. Any overlapping solutions were not needed, so heartbeat changed up a lot of the DSR protocol that we used.

11. Appendix III: Other Considerations

We had a team member leave. He had spent last semester learning front end programming in Swift and his departure hit the team pretty hard.

No one on our team had any previous experience in programming Swift, so all of us had to learn how to use it. Over the life of the project Swift had 3 major updates to the syntax for their code. These updates broke their old boilerplate code and our application wouldn’t compile until internal settings were changed and a lot of code was rewritten. Relearning Swift was a challenge.

Apple requires iOS applications to be developed on OSX. This was a problem for our team since none of us had Apple computers or devices. As a result, we had to emulate OSX and simulate iPhones, which can be slow and resource consuming. One of our members, so irritated and fed up, ended up spending $1400 on a brand new Apple laptop.