CSCI 572: Computer Networks (Fall 2023)

Proposal: Ad-Hoc Messaging App

Michael Alvarez, Luke Beukelman, Ben Breisch

October 9th 2023

1 Introduction

If someone wants to send a message to a friend nearby, the message must travel much farther than the distance between devices. Messages sent through SMS or Apple's iMessage must travel from the originating device through a series of intermediate infrastructure nodes before reaching the destination device. This causes increased latency and unnecessary network overhead. In an increasingly connected (and congested) world - why bother wasting bandwidth on a message to a nearby device that can be sent directly from peer to peer? We aim to solve this issue by proposing and developing a peer-to-peer messaging app for Android using Wi-Fi Direct for multi-hop ad-hoc networking. An application of this nature has far-reaching applications from messaging and networking at an event r conference to emergency communication during a natural disaster. Although security is always an issue with wireless communication, that is out of the scope of this project. The table below shows some preliminary requirements for this proposed solution.

Category	Requirement
Networking	Connections between two devices shall use Wi-Fi Direct using the Android Wi-Fi P2P API
Networking	Devices shall provide routing of messages between source and destination nodes using an ad-hoc routing protocol
Networking	Devices shall provide acknowledgements when messages are received in order to ensure message delivery
User Experience	Users shall be able to send text and file messages
User Experience	Latency shall be minimized as much as possible when sending messages (ideally within 1 second)
User Experience	Devices shall save messages offline such that they can be viewed later
User Experience	Devices shall show a contact list of other nearby users that a user can start a conversation with
User Experience	Devices shall show a contact list of other nearby users that a user can start a conversation with

2 Related work

2.1 AirDrop

When it comes to ad hoc networking on mobile devices, AirDrop is probably to most widespread application. It supports sending files to nearby devices, without requiring devices to be connected to the same network. AirDrop operates using both wifi and bluetooth.

We aim to differ from this product in several regards:

1. We primarily aim to support messaging. Having file sharing as a back end is a possible implementation to build messaging on, but would need to be extended.
2. AirDrop uses iCloud services in order to authenticate users, and additionally only works on Apple devices. We aim to create a de-centralized version that could theoretically be deployed on any platform (although we are only developing the application for Android in the scope of this project).
3. AirDrop uses a combination of point-to-point WiFi connections and bluetooth. We aim to only use ad hoc wifi / wifi direct in our implementation. We realize that the technology choice restricts what devices we can connect with (Apple does not support WiFi direct).

2.2 Warpinator

Warpinator is a file sharing system devloped by the Linux Mint team in order to create an air-drop like file-sharing system for non-apple users. It operates over LAN, and supports Android, iOS, Windows, and Linux, and could likely be easily ported to other Unix like systems.

We aim to differ from this product in two regards:

1. As in AirDrop, we primarily aim to support messaging
2. Operate using ad hoc networking to support messaging. Warpinator requires a connection to an existing routed network, and can not operate its own ad hoc network.

2.3 Android Beam

Android beam is a discontinued feature in Android that allowed data to be transfered over NFC. Unlike the prior two works, Android Beam did not only support file transfer, and was an open interface that applications could use to communicate. Android Beam could provide a reasonable interface for a similar application as ours to communicate with, if it were not discontinued and limited to NFC.

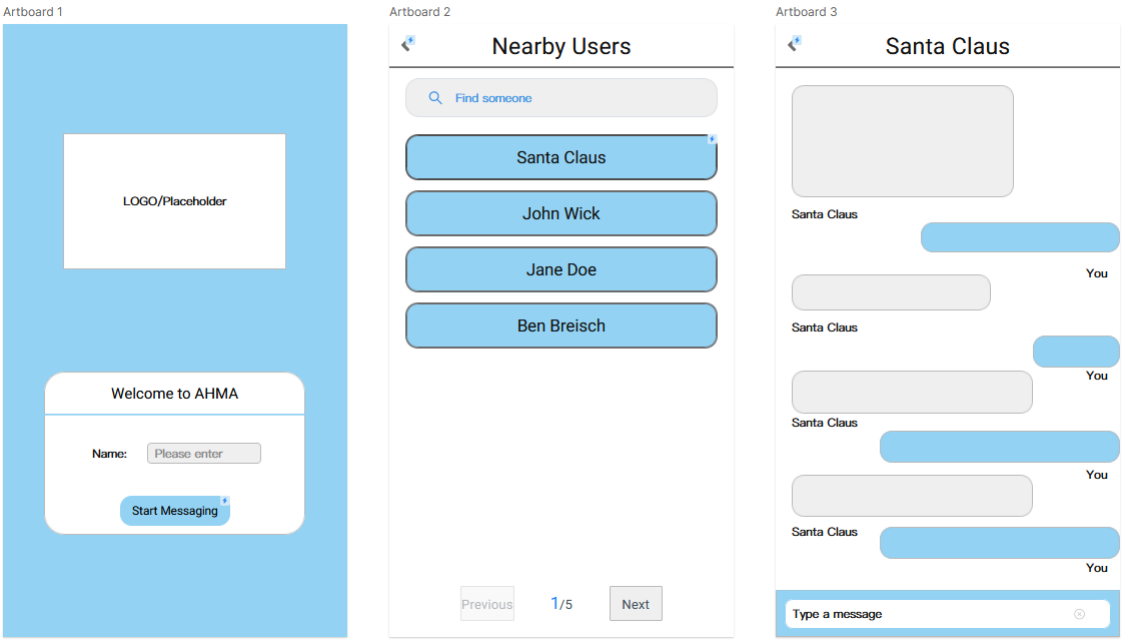
We aim to differ from this product in two regards:

1. We aim to use WiFi technologies, rather than NFC, due to the short range limitation of NFC.
2. We are creating an application, rather than an open interface to communicate over.

3 Approach

Mobile app development, especially for Android, has an extremely wide range of resources on the Internet. The Android platform is well supported by the community and we plan to leverage online reousrces for the development of this app. Android provides an existing Wi-Fi Direct library that we will leverage to speed up development. We will develop this app using Android's Studio Software Development Kit in Java. For our mobile app to function properly, we will need mock ups of different pages and screens. In the User Interface / User Experience (UI/UX) industry, these are called wireframes. We have created three wireframes, each showing the sign-in page, nearby users, and the messaging page of the application respectively. The wireframes were created in a free tool called WonderShare and are shown in the image below. These wireframes

will guide development of the user interface.



Behind the scenes, this application will use a complex set of functions and internal storage structures. Devices will periodically broadcast beacon frames with routing information, tagged with usernames for each device. This technique is essentially a fusion of AODV and DSDV routing. When the app is open on a device, it will listen for these beacon frames and update its routing table accordingly. If a device receives a frame destined for a different host, it will look up the host in its routing table and determine if it is reachable, forwarding the packet along if it is. When a frame is received at the destination host, it will detect errors with a checksum or CRC32 and reply with an ACK. Messages will be limited to 1024 characters in order to cut down on network latency and reduce packet loss. Conversations between users will be stored on the device.

4 Evaluation Plan

4.1 Scenarios

We plan to do all of our evaluation and experimentation on live devices. One of our team members has an android device and we are currently investigating how to emulate more android devices or acquire dummy devices for experimentation. To gain a full picture of our application's performance we plan to test it in the following scenarios:

- Single-hop communication
- Multi-hop communication
- New user event
- Dropping user event
- High pace usage
- File messages

- Low distance
- Maximum distance

The single-hop communication experiments will be performed with two devices which will have a single connection between them. The multi-hop communication experiments will use at least three devices and test the routing capabilities of the app. We will also test how quickly our solution can adapt to topology changes such as users opening the app or closing the app. That in particular is especially important since people are rarely just using a single app continuously. Typical users will flip quickly between several different apps while using their phone or device. Additionally, we will test the impact of high paced usage on network performance. Some users send messages back and forth as fast as their fingers can type and even do that with multiple different contacts. Most messaging apps allow for users to send each other multimedia messages or files such as photos and videos. So that capability will be tested as well. Finally, it is always important to study how the physical environment will impact performance. So we will run these experiments at a close distance range and then push it as high as we can go. All of these scenarios will be combined into an array of experiments that we can use to fully capture the performance of our app.

4.2 Metrics

For each experiment we will record the following metrics:

- Throughput
- Latency
- Packet loss rate

The throughput in this case is simply the data transmission rate throughout the whole network. So, all of the messages sent and received over time measured in data per second. The latency has multiple components that will each need to be analyzed individually as well as together. For example, there is latency from route acquisition, processing, potential queuing, transmission, propagation, re-transmission, etc. We will attempt to record as many of the individual components as possible. Finally, the packet loss rate is simply the rate at which a packet transmission fails and a re-transmission is required.

4.3 Criteria

There are two key results we are looking for by performing these experiments. The first is comparing our app to other existing solutions. We want to see if our approach of using WiFi-Direct performs better or worse than the typical Bluetooth approach. Furthermore, we want to analyze specific areas and metrics so as to conclude the pros and cons of the different technologies. The second result is if our solution meets the user's needs. Does it have the necessary throughput, latency, and reliability to provide a good experience for the casual user? That result will tell us whether or not our solution is usable and functional for the problem.

The criteria for our first result is fairly straightforward. We will simply compare our recorded metrics directs against the best existing solution we can find. This will allow us to compare and contrast the different approaches directly. In contrast, the criteria for our second result is much less straightforward. There is no benchmark for the minimum performance required on a messaging app. So we will evaluate the app against our own defined criteria of what is needed for a "good user experience".

These soft criteria are as follows:

- Able to handle constant activity, fast paced messaging between multiple pairs of users.
- Scales to a large number of users, at least 10 in a small area.
- Low latency

  - Less than one second for text messages
  - Less then ten seconds for files, pictures, or videos
- Reliable delivery, data gets delivered unless destination is unreachable.

These are the basic requirements for any messaging app to be useful. If any one of these are missing then the user experience is miserable. A messaging app needs to keep up with even the fastest typer. We can put a character limit to restrict total message size, but that wouldn't stop a user from sending many large messages in quick succession. Therefore, it is important that the app can handle such usage. One speedy user shouldn't be able to cause themselves or other users to experience delay. We will need to find out how fast a person can send out messages at maximum and then use that as a throughput threshold for how fast our app needs to be.

Scalability is the next concern. If users can't send or receive messages in a crowded environment then the app is not very useful. Settings such as a school, apartment complex, or office may have many participating users trying to use the app at once. More participating devices in the network obviously increases the total network bandwidth, but does that increase in bandwidth outpace the increase in required throughput? We don't have the equipment required to test a truly large scale deployment of our app, but we can still analyze the impact of more participating devices. Then by weighing the increase in both bandwidth and throughput, we can estimate how well the application will scale.

Since our app sends messages locally to a destination, rather than traveling through a complex network of infrastructure nodes, users will expect it to be very quick. Therefore, low latency is also a requirement. Nobody likes to wait for something as simple as a message to send. Especially when some conversations are time sensitive. So the text messages themselves should send very quickly, with a small wait required for larger files. Finally, messages need to be delivered. When a user sends a message they expect it to actually reach the destination. If that promise isn't upheld then that can cause confusion and frustration amongst users. If the destination is unreachable then the sending user needs to be alerted of this. This is a potential problem for our app as users could be hopping on and off the app at a rapid pace.

5 Milestones

The table below describes the milestones and associated due dates for each.

Milestone	Explanation
Install Development Tools	Oct 14, 2023
Initial Implementation	Oct 21, 2023
Begin Final Report Writing	Nov 1, 2023
Finish App Implementation	Nov 15,2023
Final Report Due	Nov 22, 2023

References