Energy Conservation in Smartphone Ad Hoc Networks through a Combination of Bluetooth and WiFi Direct

James Mariani
Computer Science and Engineering
Michigan State University
East Lansing, Michigan
Email: mariani4@msu.edu

Spencer Ottarson
Computer Science and Engineering
Michigan State University
East Lansing, Michigan
Email: ottarso5@msu.edu

Abstract—**In recent years we have seen the world of computing explode to the point that there are now more devices than there are people. While once it was common for every household to have one PC, it is now common for every person to have a mobile device. With the increase in use of smartphone devices, there has also been an increase in the need for mobile ad hoc networks, networks in which phones connect directly to each other without the need for an intermediate router. We propose the idea for a network including the use of both WiFi Direct and Bluetooth for device-to-device communication, and develop an application to create such a network with an emphasis on minimizing energy consumption while still maintaining an efficient network.**

I. Introduction

Due to a recent push towards mobile devices and mobile computing, many new fields and concepts have emerged in networking. One of the more promising and interesting fields regarding mobile devices is the idea of a proximity service (ProSe) application. A ProSe application is an app that tries to find instances of the same app running on devices physically near to itself. Once a ProSe application has found someone close to itself, the end result is the exchange of information such as a message or any other information important to the application [1].

This makes ProSe applications unique from traditional applications in that they are location centric and can only communicate with others near the device instead of a typical "friends list" type application [1]. There are two major use cases for ProSe applications, including public safety communications and discovery mode applications [2]. Public safety communication applications can be used in case of a network outage due to any number of things including inclement weather or your device being out of range. Discovery mode applications include things such as mobile social networking apps [2].

There are two main categories of ProSe applications: Over-the-top (OTT) and device-to-device (D2D). OTT applications rely on a server with periodic location updates from all users. This introduces a large amount of overhead that is not energy efficient. The topic of this paper will focus specifically on D2D ProSe applications. D2D solutions are developed so that two devices can communicate directly using no existing network. In this type of application you are restricted in who you can communicate with by the signal strength of the technology being used. The two major technologies used to implement D2D communications are Bluetooth and WiFi Direct.

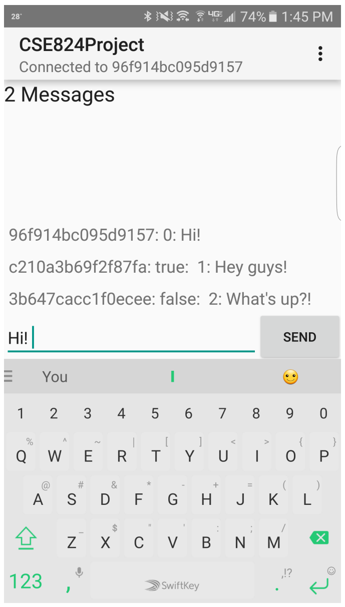
Within the realm of D2D applications, there are two different (but similar) types of applications. There are single-hop and multi-hop ad hoc networks. A single-hop ad hoc application creates a network in which it is only possible to communicate with a node within your device's current range. In a single-hop network each individual node does not act as a router and will not forward on information to its neighbors. This paper, however, is focused on the second type of D2D network, which is a multi-hop ad hoc network. A multi-hop ad hoc application creates a network in which each individual node will act as a router and forward on information and messages it receives to its neighbors. This greatly expands the reach of the network and communications.

The system we implemented and describe in this paper is an example of a multi-hop ad hoc network in which devices can communicate with each other even if they are not using the same technology (Bluetooth or WiFi Direct). They can also communicate if they are not connected directly to each other. Traditionally, D2D applications use one of Bluetooth or WiFi Direct to facilitate communication between devices. This runs into a few problems regarding devices being left out if they are not compliant with one of the technologies, as well as energy concerns if you are using WiFi Direct. We designed a chat application that can use both technologies and decide at each node whether to send messages via Bluetooth or WiFi Direct (assuming they are available) based on energy concerns of the device. The result was a success in that we were able to send text messages to devices running different technologies. Additionally, we were able to provide power savings on devices running low on battery by switching from WiFi Direct to Bluetooth whenever available.

Figures 1, 2, and 3, show an example of our final working chat application, in which a WiFi Direct device and the Bluetooth device are both connected to a group leader, and each device sends a message that is received by all nodes.

II. Related work

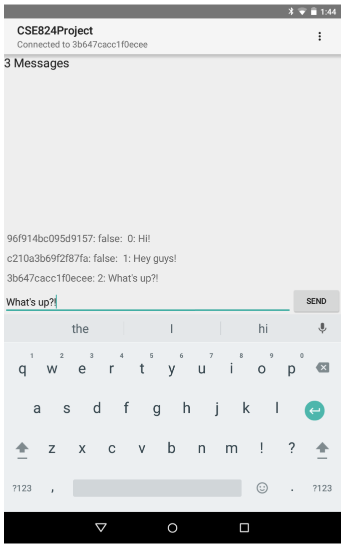
The proliferation of mobile devices (specifically phones and tablets) has led to an increase in demand for mo-


CSE824Project
Connected to 96f914bc095d9157
2 Messages

96f914bc095d9157: 0: Hi!
c210a3b69f2f87fa: true: 1: Hey guys!
3b647cacc1f0ecee: false: 2: What's up?!

Hi!

Fig. 1. Master.


CSE824Project
Connected to 3b647cacc1f0ecee
3 Messages

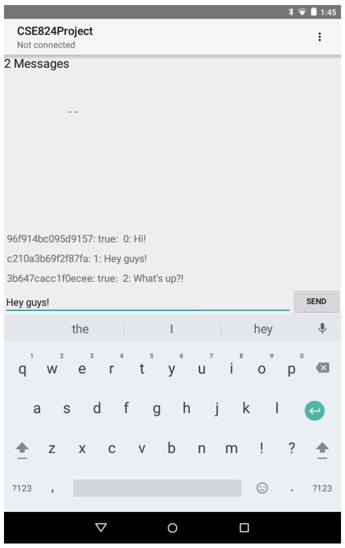
96f914bc095d9157: false: 0: Hi!
c210a3b69f2f87fa: false: 1: Hey guys!
3b647cacc1f0ecee: 2: What's up?!

What's up?!

Fig. 2. Bluetooth device.

bile/smartphone networks of connected devices. These networks are called smartphone ad hoc networks, and they are a form of mobile ad hoc networks (MANETs). MANETs are networks of connected mobile devices that communicate and share information device to device without any communication with the internet or an outside network.

Two of the major technologies used to implement smartphone ad hoc networks are Bluetooth and WiFi Direct. Bluetooth and WiFi Direct offer similar services, but differ in matters such as range, throughput, and energy consumption. Bluetooth and WiFi Direct work in similar ways on mobile devices. Bluetooth employs what it calls a master-slave paradigm in which one master node is connected to up to seven slave nodes. Slave nodes can only communicate with the master, and therefore a multi-hop system is needed for slaves to communicate with one another.

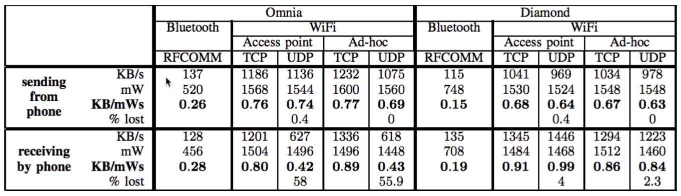

CSE824Project
Not connected
2 Messages

96f914bc095d9157: true: 0: Hi!
c210a3b69f2f87fa: 1: Hey guys!
3b647cacc1f0ecee: true: 2: What's up?!

Hey guys!

Fig. 3. WiFi Direct device.

WiFi Direct uses something almost exactly the same, except the term ”master” is replaced with ”group owner,” and ”slave” is replaced with ”client.” The only other major difference at a high level is that a WiFi Direct group consists of one group owner and up to four clients versus seven slaves with Bluetooth. In terms of performance, Bluetooth has a range of approximately 10 meters and a transfer rate of 24 Mbps. WiFi Direct has a range of around 60 meters, and a transfer rate 250 Mbps. It is clear that WiFi Direct is superior to Bluetooth in terms of throughput and range, but the advantage of Bluetooth comes from its energy efficiency. Sending a Bluetooth message uses about 520 mW, whereas sending a WiFi Direct message uses around 1560mW. There are similar energy savings on the receiving end; receiving a Bluetooth message takes 456 mW, while receiving a WiFi Direct message takes 1448 mW [3]. Figure 4 shows a table of these values.



		Omnia					Diamond				
		Bluetooth	WiFi				Bluetooth	WiFi			
			Access point		Ad-hoc			Access point		Ad-hoc	
		RFCOMM	TCP	UDP	TCP	UDP	RFCOMM	TCP	UDP	TCP	UDP
sending from phone	KB/s	137	1186	1136	1232	1075	115	1041	969	1034	978
	mW	520	1568	1544	1600	1560	748	1530	1524	1548	1548
	KB/mWs	**0.26**	**0.76**	**0.74**	**0.77**	**0.69**	**0.15**	**0.68**	**0.64**	**0.67**	**0.63**
	% lost			0.4		0			0.4		0
receiving by phone	KB/s	128	1201	627	1336	618	135	1345	1446	1294	1223
	mW	456	1504	1496	1496	1448	708	1484	1468	1512	1460
	KB/mWs	**0.28**	**0.80**	**0.42**	**0.89**	**0.43**	**0.19**	**0.91**	**0.99**	**0.86**	**0.84**
	% lost			58		55.9			4		2.3

Fig. 4. Energy consumption of Bluetooth and WiFi Direct devices in various states [3].

Because there are multiple competing technologies in smart phone ad hoc networks, there sometimes arises issues with devices communicating while running different technologies. For instance, I am running Bluetooth on my phone, and my colleague is running WiFi Direct on his phone. If we want to communicate directly, we run into a problem. An application called BWMesh was developed to attempt to addresses this problem [2].

The idea behind BWMesh is fairly simple: create a heterogeneous network in which users can communicate even if they are using different technologies. A heterogeneous network is any network that connects devices running different technologies. BWMesh was implemented in the application layer in the form of a simple android chat application that was deployed on multiple android compatible devices. The main idea is that I can send a message from my phone, and it will be send/forwarded to all devices running BWMesh, regardless of what technology they are using. It relies on at least one node running both Bluetooth and WiFi Direct to act as an intermediary node between devices that cannot usually communicate directly. For example, my phone using a Bluetooth connection is connected to a phone running both Bluetooth and WiFi Direct, and that phone is connected to another phone running only WiFi Direct. When a message is sent from my phone, it will be sent via Bluetooth to the intermediary node, who will then forward on via WiFi Direct to the third device [2].

Through use of these intermediary nodes, two devices that would otherwise be unable to communicate can now send messages to one another. BWMesh was developed as a proof of concept to the fact that different technologies could be combined to create heterogeneous network. Because it was only a proof of concept, there were some things it did not address. One of the major issues in all ad hoc networks is energy conservation, and BWMesh did not implement or consider energy needs during development [2].

III. Methodology

The idea of the BWMesh architecture proved an important concept; a mobile ad hoc network can be enhanced by the use of both Bluetooth and WiFi Direct. It proves that through the use of both, the range and device support of a network can be enhanced. Beyond that, however, there is still much that can be improved upon, specifically when looking into the decision of which method to use. As mentioned above, there are tradeoffs to using each of the technologies. WiFi Direct is faster and has a longer range, but Bluetooth consumes less energy, an important factor when working with mobile devices.

While we know that there are benefits and drawbacks to both technologies, we first had to decide what we are trying to prove by exploring the idea further. There is already work done on showing the power and throughput tradeoffs on Bluetooth and WiFi Direct, but we wanted to provide a more fully implemented, real world application to be able to prove our results. Since power consumption is such a major factor, that is where we put our focus. We decided to build our app around the idea that we can establish multiple connections, with the ideal option being WiFi Direct. In times which energy is of higher concern, specifically when the battery life on a smartphone is low, we can sever the WiFi Direct connection and opt for Bluetooth to save energy.

With that goal in mind, there are two main questions we must be able to answer: can we switch between the two connections on the fly, without data loss, and does this switch actually provide power savings? Most of existing work done on comparing WiFi and Bluetooth is done using both separately. This led to our idea of implementing everything in an application. To investigate and gain further insight into how these differences may be used for the benefit of users, we developed a new Android smartphone application. Android is free to develop for, and runs on a variety of devices. Nearly every new device has both Bluetooth and WiFi Direct capabilities, which makes it an ideal platform to test our idea. The application is written in Java using Android Studio, targeted for devices running API level 23. By creating a full application, we can gather real time data that we believe to be much more valuable than what we could do with a simulator or by simply using estimates of average power consumption.

Our new application is a multihop chat application similar to BWMesh, but slightly more extensive. The details of our application and its features will be discussed in the next section, here we will describe how we approached building the application.

To split the work between the two of us, we began by creating separate frameworks for each of the two technologies (Mariani - Bluetooth, Ottarson - WiFi Direct). After developing simple chat applications with each respective method, we combined the two into one.

Since our main focus is power saving, we needed some way to accurately measure the energy usage. To do this, we opted to use the free application Trepn Power Profiler, created by Qualcomm [4]. The app runs over top of any application and collects data on a variety of components, including cpu usage, network state, and most importantly, battery consumption.

IV. Design Rationale

From a high level, our solution to the energy concerns in the heterogeneous smartphone ad hoc network includes a few key differences between our product and BWMesh (a more detailed application flow is shown below). After the network of devices are connected appropriately, a message can be sent from any connected device. In BWMesh, they utilized a simple flooding algorithm which forwarded that message on to every neighbor. They had a source field in their message format that would indicate who sent the original message, and when the message got back to the source it would stop sending. Our solution, however, utilized a modified flooding algorithm that added more bounding parameters to greatly limit the amount of messages being sent across the network (while still reaching every node). We added a source and direct source (the specific node who sent you the message) field to our message format, and no message is ever sent back to the source or direct source. This alone greatly reduces the number of messages being sent across the network.

In BWMesh, the connections between two devices were static in that you would always send via the same technology to the same device. The main energy saving tactic that we employed is a system that will send via WiFi Direct (if available) to improve speed and latency, but will choose to send via Bluetooth (if available) if a particular node is in need of energy conservation. This solution balances network performance and battery concerns.

When first entering our app, you are greeted by an empty chat screen. There is a menu that will bring up options for what you might want to do, shown in Figure 5.

If you want to manually connect via Bluetooth, you can click on the connect Bluetooth menu option, which will present a list of available devices to connect with. Once connected, a toast will appear on the screen to confirm the pairing.

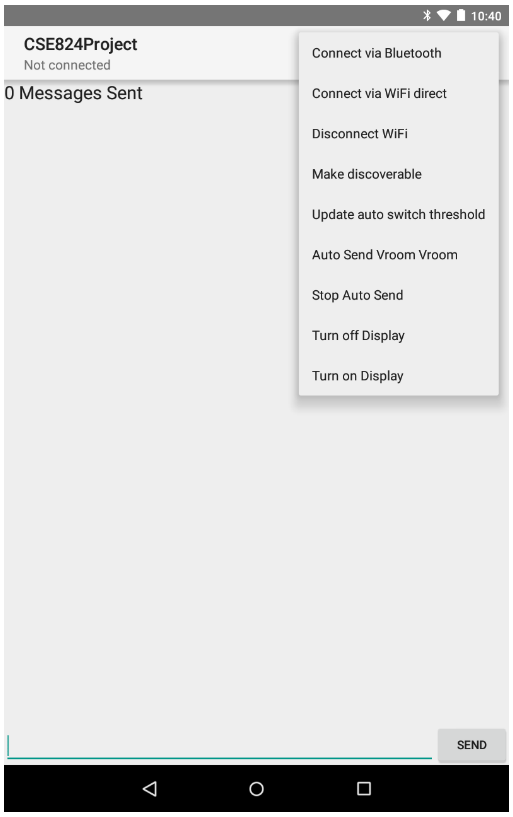
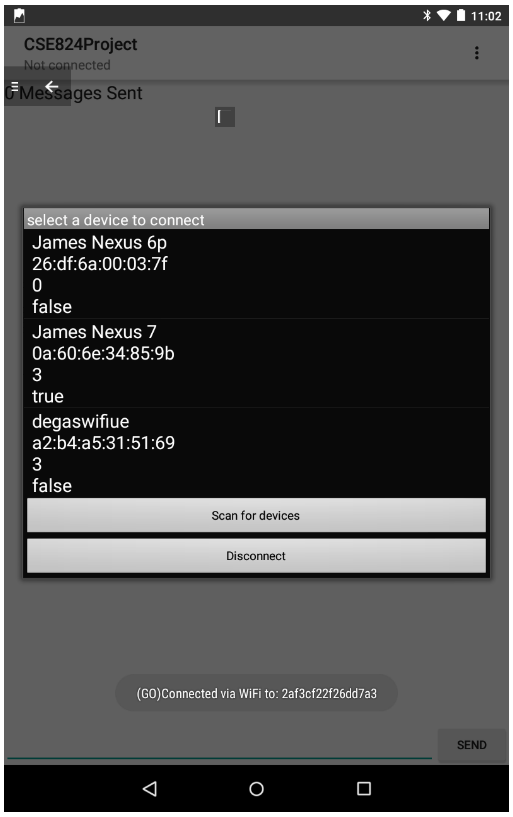


Fig. 5. Menu



```
James Nexus 6p
26:df:6a:00:03:7f
0
false
James Nexus 7
0a:60:6e:34:85:9b
3
true
degaswifiue
a2:b4:a5:31:51:69
3
false
```

(GO)Connected via WiFi to: 2af3cf22f26dd7a3

Fig. 6. WiFi Direct connected.

Similarly, if WiFi Direct is desired, you click on the menu option for WiFi Direct connections and this will present a list of devices that are available to connect with. Once a successful WiFi Direct connection is established, a toast confirming the connection will be shown as in Figure 6.

There is an additional menu option to make the device discoverable on Bluetooth. The app will request permission to make each device discoverable for one hour on initial launch, but if it is open for more than an hour you might want to enter discoverable mode again. There is also a menu option to disconnect WiFi manually in the case that you do not want to wait for a certain battery level to automatically shut off WiFi.

There is also an option to switch the battery threshold for the automatic switch from WiFi Direct to Bluetooth. The app is coded to switch from WiFi Direct to Bluetooth when a node reaches 30% battery, but if for any reason you want to change that value, the menu option to change the threshold will bring up a dialog box, shown in Figure 7, in which a new threshold can be specified.

For the purposes of testing the limits of our system as well as introducing additional battery strain, an option menu to automatically send 20 messages per second is given. In addition to the autosend there is also a menu option to stop a currently running autosend. The autosend feature is what was used while collecting data to test whether our system performed better for battery consumption than traditional ad hoc solutions. Figure 8 shows the result of using the autosend feature.

Finally, due to the strain on the device put forth from sending and displaying so many messages, it would sometimes create some lag on the display. To avoid this lag, there is a menu option to turn off the display, in which the messages getting sent and received are not shown on the screen, and instead only the number of messages is shown. There is also an option to turn the display back on.

The screenshot shown in Figure 9 demonstrates multiple features of our system. First, you will notice that the auto send feature is being utilized. Secondly, the option to turn off the display and turn it back on is being used due to the sizable gap in the sequence number of the messages on the screen. Lastly, the automatic switching from WiFi Direct to Bluetooth has occurred. You can tell whether something was received by Wifi Direct due to the presence of the value true immediately preceding the sequence number of the message. If a message was received via Bluetooth the value will be false. As seen in Figure 9, the sequence number on the screen jumps from 225 to 2435. This shows that the display was turned off and then back on, and the value before the sequence number switches from true to false, showing that the messages used to be received by WiFi Direct, but are now being received by Bluetooth.

The network graph in Figure 10 shows how a message would travel from a source node (A) to all of the other nodes. In this example, there are two sub pico-nets shown. Node B is a WiFi Direct group owner, connected via WiFi Direct to nodes A, C, and D. Node E is another WiFi Direct group owner and bluetooth master. Node F is connected to it via WiFi Direct, but node G is in power saving mode and has severed the WiFi connection in favor of Bluetooth. Node D is a bluetooth

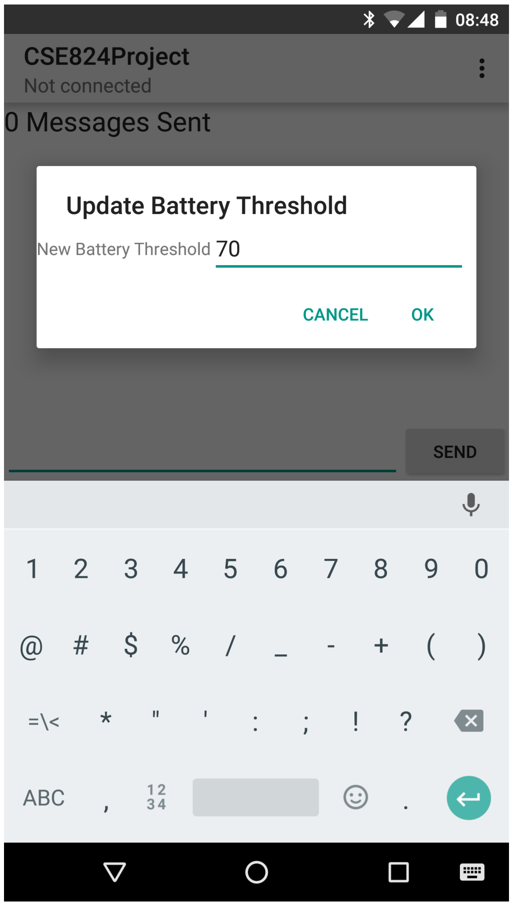
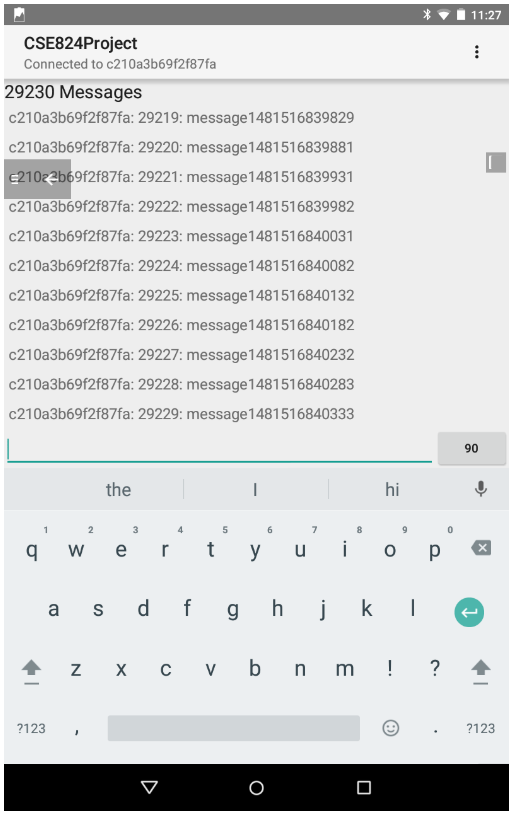


Fig. 7. Updating the threshold.


CSE824Project
Connected to c210a3b69f2f87fa
29230 Messages

```
c210a3b69f2f87fa: 29219: message1481516839829
c210a3b69f2f87fa: 29220: message1481516839881
c210a3b69f2f87fa: 29221: message1481516839931
c210a3b69f2f87fa: 29222: message1481516839982
c210a3b69f2f87fa: 29223: message1481516840031
c210a3b69f2f87fa: 29224: message1481516840082
c210a3b69f2f87fa: 29225: message1481516840132
c210a3b69f2f87fa: 29226: message1481516840182
c210a3b69f2f87fa: 29227: message1481516840232
c210a3b69f2f87fa: 29228: message1481516840283
c210a3b69f2f87fa: 29229: message1481516840333
```

Fig. 8. Messages sent via the autosend feature.

slave that has connected to the Bluetooth master of the other group, node E. This establishes a connection between the two pico-nets. When A sends a message, it sends it along its WiFi Direct connection to group owner B. Node B then forwards the message to all of it's children, except for node A, source of the message. Nodes C and D then forward the message along all of their connections. Since node B is the direct source of the message for these two, neither sends the message back to B. D does send it along its Bluetooth connection to E. Node E then forwards the message on to node F via WiFi Direct, and to node G via Bluetooth.

V. Technical Results and Analysis

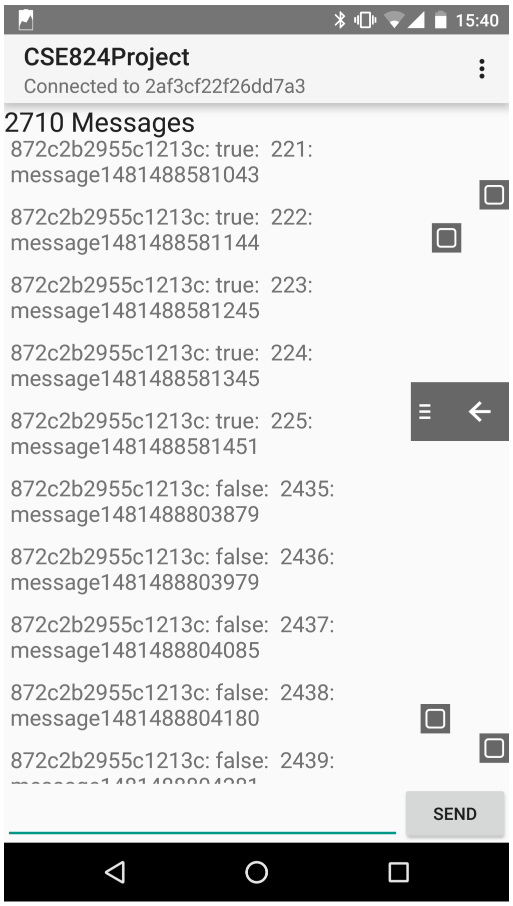
Once the application was designed, implemented, and tested for functionality, we set to work gathering data. First we tested whether the application could sever a WiFi Direct connection and continue sending the data via Bluetooth. There are two different possibilities for this, either the sender could be the one who disconnects, or the receiver. When the sender disconnects, our system reliably switches over to using the bluetooth connection immediately. In Figure 11 we can see that with our autosend feature, the receiver is receiving message over WiFi, then over Bluetooth without missing one.

However, our system ends up failing sometimes when the receiver is the one who does the disconnecting. Figure 12 shows a case in which some of the messages are dropped during the switch.

The reason for this is simple yet difficult to solve. Our autosender is sending at a rate of 20 messages/second. Unfortunately, if one of the devices disconnects, it takes time for that information to be discovered by the sender. Since the messages are being sent so fast, the sender ends up attempting to send a few messages over WiFi Direct until discovering that the connection no longer exists, and then switching to Bluetooth. There are ways of getting around this, but none of them are perfect. One possibility is sending an acknowledgement of every message received. The sender would thus then know which messages were missed out on, and be able to resend them. However, this would involve sending twice the number of messages. Since our main focus is on saving energy, this is not an acceptable solution.

Another option is that when the receiver wants to disconnect, it sends a disconnect message to the sender. It would then not stop the WiFi connection until after it begins receiving Bluetooth messages. The downsides to this are that it relies on a controlled setting. One thing we want our system to be able to handle is an uncontrolled setting where the WiFi may shut off through a system function, not called through our app. This problem would prove very difficult if not impossible to solve efficiently, and we did not address it with our application. We did not focus as much on this problem as it only occurs in limited situations, if a message is sent within a few milliseconds of the receiver disconnecting.

After determining the reliability of the system, we gathered information on how much power our application requires using the Trepn Power Profiler [4]. First we ran a control test in



CSE824Project
Connected to 2af3cf22f26dd7a3

2710 Messages

872c2b2955c1213c: true: 221: message1481488581043

872c2b2955c1213c: true: 222: message1481488581144

872c2b2955c1213c: true: 223: message1481488581245

872c2b2955c1213c: true: 224: message1481488581345

872c2b2955c1213c: true: 225: message1481488581451

872c2b2955c1213c: false: 2435: message1481488803879

872c2b2955c1213c: false: 2436: message1481488803979

872c2b2955c1213c: false: 2437: message1481488804085

872c2b2955c1213c: false: 2438: message1481488804180

872c2b2955c1213c: false: 2439:

Fig. 9. Switching while on autosend.

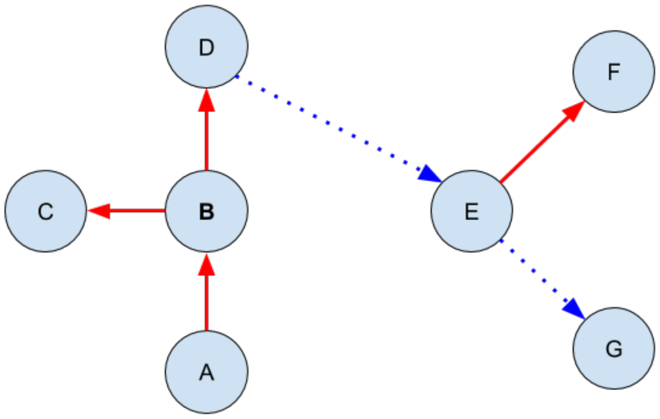
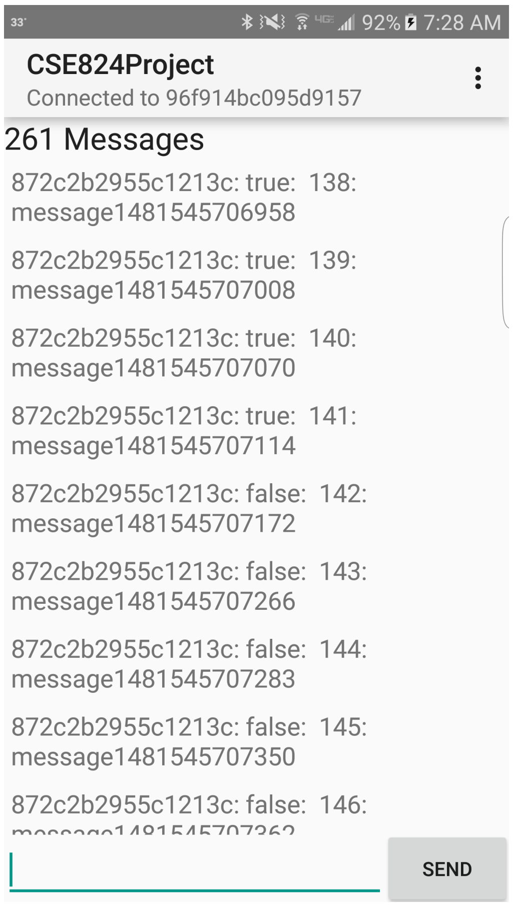


Fig. 10. Example setup

which we used the autosend to send messages exclusively over WiFi Direct, shown in Figure 13. The device used an average of 3,439 mW.

Then we ran a similar profile, except this time the application reached our WiFi Direct disconnect battery threshold. Figure 14 shows where the WiFi Direct disconnects and the messages begin sending over the Bluetooth connection. While sending via WiFi Direct, the device was consuming and average of 3,454 mW. Then while using Bluetooth, the device needed only an average of 2,860 mW. These power



CSE824Project
Connected to 96f914bc095d9157

261 Messages

872c2b2955c1213c: true: 138: message1481545706958

872c2b2955c1213c: true: 139: message1481545707008

872c2b2955c1213c: true: 140: message1481545707070

872c2b2955c1213c: true: 141: message1481545707114

872c2b2955c1213c: false: 142: message1481545707172

872c2b2955c1213c: false: 143: message1481545707266

872c2b2955c1213c: false: 144: message1481545707283

872c2b2955c1213c: false: 145: message1481545707350

872c2b2955c1213c: false: 146:

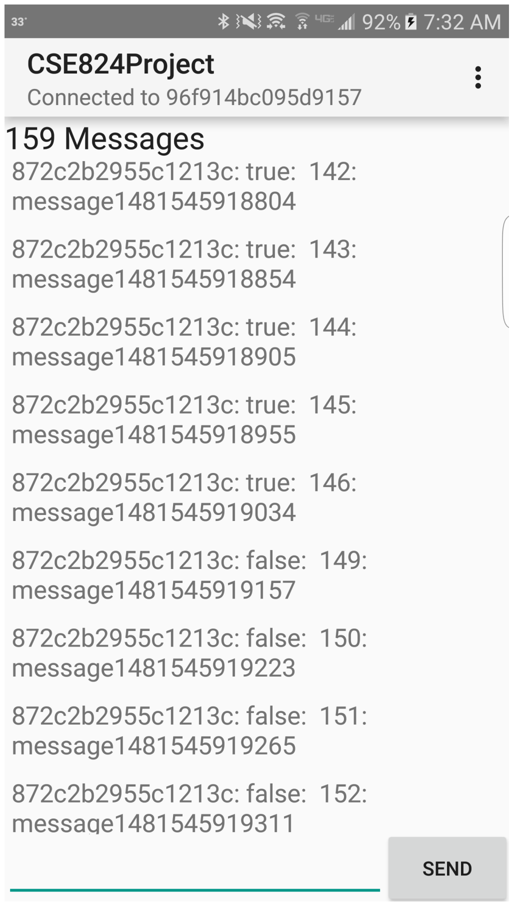
Fig. 11. Correctly disconnecting (messages with the label ”true” are messages received via WiFi Direct, messages with the label ”false” are messages received via Bluetooth).

consumption numbers are the power consumption of the entire smartphone, not only our application. We tooks steps to ensure that outside forces would have minimal impact on the energy usage. We used the same device model (Nexus 7), set the screen brightness to be the same on each, and closed all background apps.

Another important consideration that must be taken into account is when both channels are being used. If, for instance, a Group Owner is connected via WiFi Direct to one device, and Bluetooth to another, it will have to be actively using both to send a message. This brings up the obvious question of, how much extra power does that consume? Figure 15 shows the results when sending a message via WiFi Direct only, adding a second node via Bluetooth. When adding the Bluetooth connection, the power consumption spikes briefly, but then settles at a level only slightly above what it was with the WiFi Direct connection only. This shows that even in a large network, a device can be both a WiFi Direct Group Owner and Bluetooth master and send messages along both channels without a significant increase in power consumption.

VI. Summary and Lessons Learned

We have proven that we can develop a mobile ad hoc network using both Bluetooth and WiFi Direct, send multihop



872c2b2955c1213c: true: 142: message1481545918804

872c2b2955c1213c: true: 143: message1481545918854

872c2b2955c1213c: true: 144: message1481545918905

872c2b2955c1213c: true: 145: message1481545918955

872c2b2955c1213c: true: 146: message1481545919034

872c2b2955c1213c: false: 149: message1481545919157

872c2b2955c1213c: false: 150: message1481545919223

872c2b2955c1213c: false: 151: message1481545919265

872c2b2955c1213c: false: 152: message1481545919311

Fig. 12. Incorrectly disconnecting (messages with the label ”true” are messages received via WiFi Direct, messages with the label ”false” are messages received via Bluetooth).

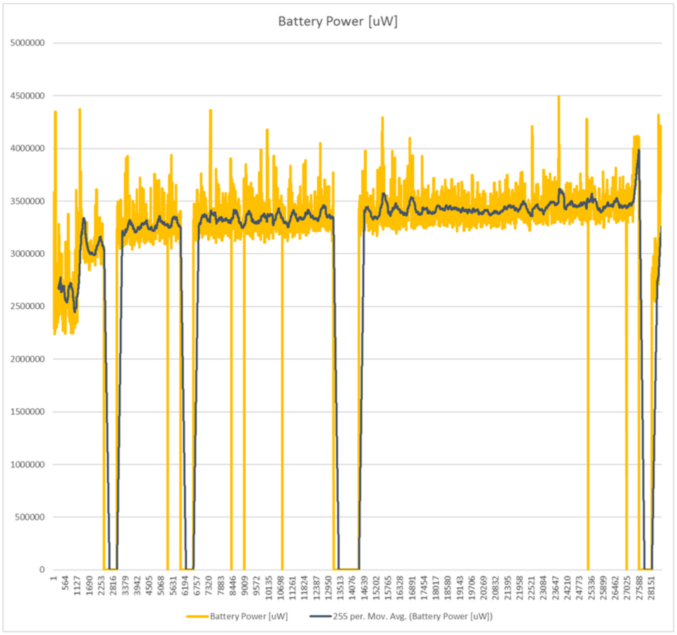


Fig. 13. Power usage sending via WiFi Direct only. We can see that the power consumption stays fairly steady throughout.

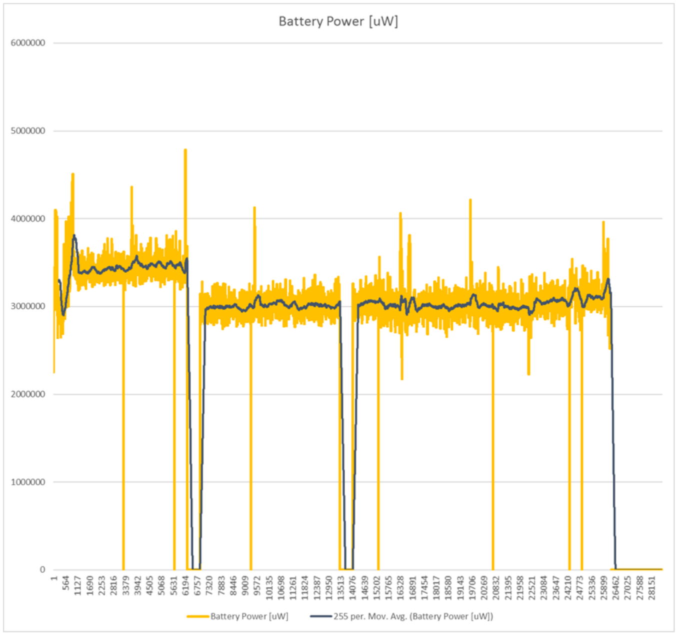


Fig. 14. Power usage sending via WiFi Direct at first, then switching to using Bluetooth. We can see in the graph the moment that this occurs, as the power consumption drops significantly.

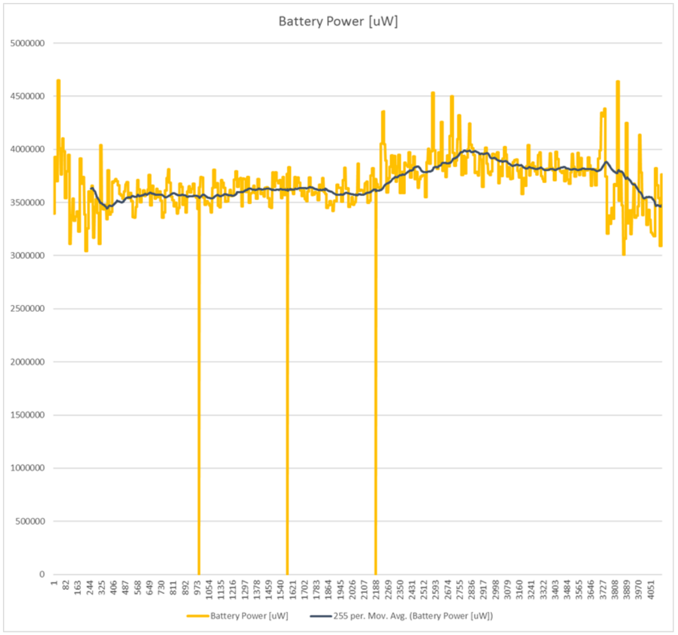


Fig. 15. Power usage sending via WiFi Direct at first, then connecting to another node via Bluetooth. As shown, when adding the second connection at t≈2188, the power consumption initally spikes a little bit, but then settles, and does not drastically increase from the WiFi Direct only level.

messages over both of them individually, as well as automatically switch from using WiFi Direct to Bluetooth on the fly. Mobile ad hoc networks are becoming increasingly common as the number of mobile devices grows rapidly, as well as the need for secure communications. Since saving energy is crucial when working with mobile devices, our system is a step towards efficiently managing a network while taking into concern that the battery of some devices may be low and energy is more critical to them. We have proven that we can connect multiple pico-nets of a small number of devices and correctly transmit messages throughout the entire network, and can do so with an efficient flooding algorithm that only forwards a message through the connections on which it has not yet traveled. With this work we have improved on existing architectures that have proposed similar ideas and extended the concept of what a mobile ad hoc network can do.

VII. Future Work

Even with the success of our system, there are still a few areas in which it is possible to improve. The major improvement that could possibly be implemented is automatic pairing/connecting of devices. When initially setting up the network of devices, each WiFi Direct group owner and Bluetooth master must be configured initially and then connect manually to all clients/slaves. We were able to simulate many different topologies with our system (multiple pico-nets connected, flat topologies, as well as the more traditional master-slave paradigm), but automatic connections will make the user experience much smoother and allow for more unique topologies to be created and studied.

Another area for future improvement presents itself in the auto send feature that we implemented. Currently, when sending messages at a high frequency (20 or more per second) we are seeing that a few messages get lost while switching from WiFi Direct to Bluetooth. At low sending rates this is not an issue, but there can be some optimizations implemented to help mitigate this problem at higher message sending rates.

Finally, there should be an evaluation of overall system optimization. Currently the app switches from WiFi Direct when at 30% battery (or the user specific threshold) for the simple purpose of extending battery life. There can be future study into the ideal threshold for the switch of technologies to maximize system performance and speed while still conserving energy.

Acknowledgment

We would like to thank Dr. Li Xiao and Michigan State University Department of Computer Science and Engineering.

References

[1] Chieh-Jan Mike Liang, Haozhun Jin, Yang Yang, Li Zhang, and Feng Zhao. Crossroads: A framework for developing proximity-based social interactions. In *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 168–180. Springer, 2013.

[2] YF Wang, J Tang, Q Jin, and JH Ma. Bwmesh: a multi-hop connectivity framework on android for proximity service. In *Proceeding of The 12th IEEE International Conference on Ubiquitous Intelligence and Computing, UIC*, 2015.

[3] Roy Friedman, Alex Kogan, and Yevgeny Krivolapov. On power and throughput tradeoffs of wifi and bluetooth in smartphones. *IEEE Transactions on Mobile Computing*, 12(7):1363–1376, 2013.

[4] Qualcomm. Trepn power profiler, 2016.