mumble: Framework for Seamless Message Transfer on Smartphones

Bhojan Anand, Tan Guo Wei
School of Computing, National University of Singapore
banand@comp.nus.edu.sg, guowei.tan@u.nus.edu

ABSTRACT

This work explores the possibility of transferring data between mobile devices that are nearby each other without the need of pairing, authentication, superuser access and Internet connectivity. Such technology can be used for emergency broadcast, traffic congestion avoidance, IoT, smart city, social and dating systems and games. Except Bluetooth Low Energy technology, there is no other protocol or technology available that allows pairing-free data transfer without Internet connectivity. Bluetooth Low Energy is capable of doing so but at a relatively short range. Using Wi-Fi Direct's Service Broadcast and Discovery, a simple yet novel method is developed which allows for seamless long range (Wi-Fi range) data transfer without Internet connectivity.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: Framework

General Terms

Design, Message Transfer, Mobile Communication, Proximity

Keywords

Authentication-free, Automatic, Message Transfer, Android, Proximity-Based, Messaging, Wi-Fi Direct, Pairing-free, Short Message

1. INTRODUCTION

The idea of content sharing between mobile devices which are nearby is not new and has been gaining popularity in social, dating, Internet of things (IoT), smart city and game apps. Currently, content sharing between nearby devices is done using one of the following methods:

1. Direct peer-to-peer connection between nearby devices through Wi-Fi ad-hoc mode, Wi-Fi Direct, Bluetooth and Bluetooth Low Energy.
2. Location information obtained via GPS, Cell-towers or the Internet, is sent to a server where location matching is done [3][2].

Mobicom-SmartObjects'15, September 7, 2015, Paris, France.
© 2015 ACM. ISBN 978-1-4503-3535-5/15/09 ...$15.00.
DOI: http://dx.doi.org/10.1145/2797044.2797048.

Technology	Authentication	Internet Connectivity	Superuser Permissions	Range
Bluetooth Classic	Yes	No	No	Medium
BLE	No	No	No	Short
Wi-Fi Ad-Hoc Mode	No	No	Yes	Far
Wi-Fi Direct	Yes	No	No	Far

Table 1: Summary of Available Technologies

Although the first method allows accurate detection of nearby devices, it suffers from a major inconvenience; users must physically authenticate the connection through a button press. This makes it impossible to be used for automatic content sharing with other users, as participating users have to be actively using the app to establish a connection. The second method solves the problem of authentication requirement as it connects to trusted servers in the Internet. Although the second method solves the issue of authentication, location information obtained suffers from either accuracy and/or power consumption problems - accurate positioning requires GPS which is high in power consumption and low power positioning via cell-towers is inaccurate. Table 1 summarises the current available technologies for direct messaging.

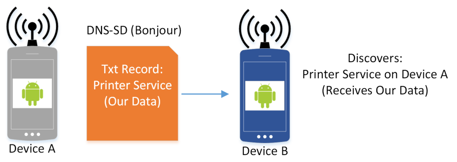
Naturally, the next step in content sharing between nearby devices would be to develop a method which allows accurate nearby device discovery, with low power consumption and without the need for a server or Internet connectivity for location matching. Most importantly, this new method of content sharing should remove both the standard pairing and authentication (any form of key press) process, allowing seamless message transfer between nearby devices.

In this paper, we present our framework for seamless message transfer between nearby smartphones using Wi-Fi Direct's Service Broadcast and Discovery (SBD). This method is verified to be feasible on Android devices, transferring data to devices within Wi-Fi range. Our approach has been implemented as an app on Android devices running Android 4.1 and above. We have proved that the power consumption of our approach falls within the acceptable range on mobile devices through tests and simulations. We also show that the power consumption can be further reduced through broadcast period reduction and sleep intervals manipulation and the feasibility of such implementation.

2. RELATED WORK

The Streetpass feature of Nintendo 3DS[1] is what our solution aims to emulate for Andriod platform - automatic data transfer between nearby devices.

[1] http://www.nintendo.com/3ds/features/streetpass


DNS-SD (Bonjour)
Txt Record: Printer Service (Our Data)
Discovers: Printer Service on Device A (Receives Our Data)
Device A
Device B
Figure 1: How data is transferred using DNS-SD.

In CoCam[9], users attending the same event are able to share photos and videos taken through Wi-Fi using the tethering capabilities of Android devices. This solution of transferring data is not feasible for our use cases as it requires a centralised server to group and coordinate the transfers.

For location based services, there are several works that uses both GPS and cell towers to pin point the location of devices. Similarly, there are also works that uses GPS along with Wi-Fi access points for indoor positioning of the devices such as [2][4][7]. Although these works are able to pin point the devices both indoors and outdoors, they require Internet connectivity.

There are also several proximity based content sharing works using peer-to-peer protocols such as [5][6][10] [8]. These works allow content and data transfer between devices based on the location of the devices. However, they require either Internet connectivity, superuser permissions or user authentication per peer-to-peer connection.

Silent broadcasting[11] is a connectionless messaging framework on Android using Wi-Fi Direct's service broadcast and discovery[2]. The requirement of superuser access makes this method of data transfer not practical in real-world usage.

Our solution differs from the above. We are able to achieve seamless data transfer between nearby devices without the need for Internet connectivity. In addtion, our method is power and communication efficient.

3. MUMBLE - FRAMEWORK DESIGN

3.1 Seamless Message Transfer

Android's Wi-Fi Direct's Service Broadcast and Discovery (SBD) functionality is used for detecting services available on other devices, prior to establishing a peer-to-peer Wi-Fi Direct connection. There are 2 types of SBDs available on Android's Wi-Fi Direct; uPnP and DNS-SD (Bonjour). In particular, Bonjour uses a component called Txt Record[3] to indicate the services available on a device. Mumble framework places data in Bonjour's Txt Record to transfer data without any device-to-device connection or authentication from the user (as shown in Figure 1). Mumble has the following design goals - 1) Seamless data transfer, 2) No superuser access requirement, 3) Accessible to every app installed on the device, and 4) Device owners must be able to see and moderate apps using the framework and turn the framework on/off as they prefer.

As shown in Figure 2, the functionalities of the framework are encompassed within a separate app, installed on the Android device. Any external app can then make use of the framework by using the API (Section 3.6), carrying out app-to-app communication to send and retrieve data. The following is a flow of how the app works:
1) App passes data to framework app through the API.
2) Device owner can choose to turn on/off the framework through the app GUI.
3) If the framework is turned on, the app will maintain a background service which automatically sends and receives data to and from nearby devices.
4) App can retrieve data from the app through the API.

[2]http://developer.android.com/training/connect-devices-wirelessly/nsd-wifi-direct.html
[3]http://www.ietf.org/rfc/rfc6763.txt

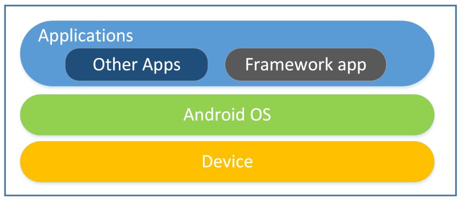
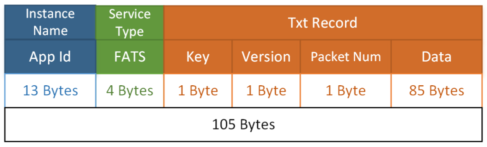


Figure 2: Architecture diagram of the framework.



Instance Name	Service Type	Txt Record			
App Id	FATS	Key	Version	Packet Num	Data
13 Bytes	4 Bytes	1 Byte	1 Byte	1 Byte	85 Bytes
105 Bytes					

Figure 3: Packet design used by the framework.

3.2 Packet Design

With reference to the SBD variables and their limitations (eg. the Service type and Instance name variables are restricted to only characters, numbers and symbols), the packet is designed in the format as shown in Figure 3. Each app being broadcasted is mapped as a single service. If there are multiple apps, multiple services are broadcasted simultaneously. The following is a detailed discussion of each variable in the packet:

App Identifier. This is used to identify the app being broadcasted by the device. The unique identifiers for apps on Android are the root package names for each app. A shortened package name is obtained via the following steps: 1) Run 64-bit MurmurHash3 on the app package name to obtain a 8 byte identifier. 2) Convert the 64-bit hash into a base-36 string representation. 3) Resultant app identifier is 13 characters (bytes) long consisting of a-z and 0-9. The 64-bit (8 bytes) hash was converted into a 13 bytes identifier, using up an additional 5 bytes. This is done so due to the implicit restriction on special characters and symbols stated above.

Service Type. This is an identifier used to identify services belonging the framework. The string "fats" is used as the service type.

Key. A character used for mapping the data in the Txt Record. This is required as the Txt Record required by the API is a Java *HashMap* class.

Version. This is used for identifying duplicate packets received by a device.

Packet Number. This is used only when the data is too large (Section 3.3), and is split into fragments. It is used to recombine the packet on the receiving device.

Data. Data the app wants to send.

3.3 Limitations

There are several limitations in using Wi-Fi Direct's SBD for data transmission.

Data Size Limit. Android's Wi-Fi Direct's SBD limits 105 bytes per service broadcast (leaving only 85 bytes for the framework), inclusive of service type, instance name, Txt record and its mapping key. *Service Limit*. The number of services that can be broadcasted simultaneously is also implicitly limited by Android. This limit was found to be 13 simultaneous services. It is not mentioned in any Android's developer reference. If this limit is exceeded, services being broadcasted is chosen based on the latest 13 services added.

Broadcast Duration Limit. Android's Wi-Fi Direct's SBD will automatically terminate itself after 2 minutes of broadcasting.

1-Way Communication. Transferring data using Wi-Fi Direct's SBD is a 1-way communication. As such, the device will constantly broadcast the same set of data from an app. Additionally, Wi-Fi Direct's SBD on Android will only notify the user that a service has been discovered once per broadcast session. This

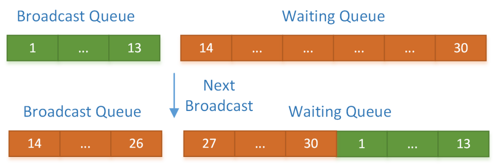


Figure 4: How the broadcast queue works.

means that even if the data being broadcasted by other devices have changed, the user's device will not see the changes until the broadcast on the user's device is restarted.

Max Data Rate Achievable. The max data rate achievable is not really calculable as it depends heavily on the time it takes for an Android device to start, respond and stop the Wi-Fi Direct's discovery request. This time value changes across devices and custom Android implementations by different manufacturers.

3.4 Overcoming Limitations

Due to the limitations mentioned in the previous section, the basic framework is limited to send 85 bytes of data per app and up to a total of 13 apps. The following section discusses the additional features implemented in the framework that help reduced these limitations.

Increase Byte Limit. The 85 bytes of data limit per app can be increased to 1105 bytes (13 services x 85 bytes) by using several services to broadcast data from an app. Apps with data more than 85 bytes are fragmented into multiple packets, with each packet being broadcasted as a service, and then reassembled at the receiver side.

Scheduled Services. The 13 apps limit is resolved by using a scheduler implemented in the framework. This scheduler consists of 2 parts; a broadcast queue and a quick swap feature.

Broadcast Queue. The broadcast queue (Figure 4) is implemented using two queues; a broadcasting queue and a waiting queue. After a broadcasting period, the app will automatically restart the broadcast with a new set of data from the waiting queue, while broadcasted apps will return to the end of queue. This ensures that all apps will be broadcasted in turn.

Quick Swap Feature. The addition of a broadcast queue introduced a new problem; exchange of data between the same app on different devices will not be possible unless both devices are broadcasting the same app at the same time. This problem results in a reduced efficiency of transferring data as successful data exchange between an app on two devices can only happen in one of the following three possible scenarios: 1) Both devices not broadcasting the app (not successful), 2) 1 of the device is broadcasting the app (not successful), and 3) Both the devices are broadcasting the app (successful).

In order to improve the efficiency, a quick swap feature is implemented along with the broadcast queue. The quick swap feature is implemented using a third queue; the Interrupt queue as shown in Figure 5.

The quick swap feature will quickly swap in packets of apps that are currently in the Waiting queue into the Interrupt queue, and subsequently the Broadcast queue, if a nearby device is found to be broadcasting that app.

By adding the Interrupt queue, data exchange can now occur in two of the three possible cases: 1) Both devices not broadcasting the app (not successful), 2) One of the device is broadcasting the app (successful), and 3) Both the devices are broadcasting the app (successful).

This increases the chances of devices exchanging data from the same app as data exchange can occur as long as one of the device is broadcasting the app. This increases the maximum number of app data exchanged between two devices from 13 to 26 in the best case

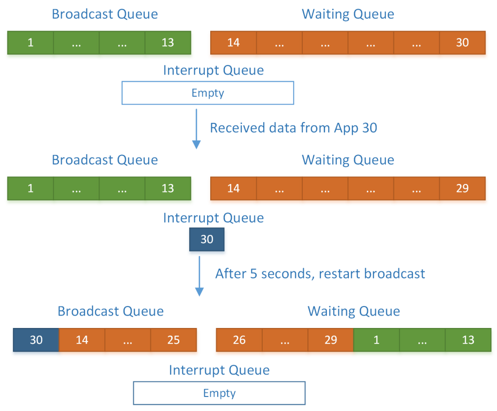


Figure 5: How the interrupt queue works.

scenario. However, in the worst case scenario, devices will still not be able to exchange data from the same app if neither of the devices are broadcasting that app.

3.5 Security

As with any communication framework that shares and exchanges data, a key factor in the usability of the framework is the security of the data being transferred. Since the framework is designed to share data between devices automatically, users are therefore not encouraged to share any sensitive data. Additionally, apps using the framework are encouraged to encrypt the data before sending it to the framework for added security. Malicious apps might attempt to retrieve data from the framework. This is countered by requiring a security key and a reply address to retrieve data from the framework. A malicious app can pretend to be the framework app, using the actual package name of the framework app. A third party certification is used to counter this issue. A DDoS attack can be attempted using a single or multiple device constantly broadcasting discovery requests with multiple services. However, the targeted device will only receive each malicious request once per discovery request (targeted device's own discovery request). A man-in-middle attack can be easily overcome by advising apps to encrypt the data being exchanged. Due to space constraints, details about the security features are moved to project homepage [1].

3.6 API

An API has been developed for communicating with the framework app. The following section discusses the set of functionalities that has been made available.

Register. Registers the external app. Reply address and data to be transmitted are recorded.

Unregister. Deletes registered apps and all related data. Once unregister is called, the framework will no longer broadcast data from the external app, or respond to any nearby devices that the external app is available on the device.

Update. After registering data from an external app to the framework, any changes to the data can only be made through Update. A passcode is required to update the data. The Update functionality will only work if the external app is registered in the framework.

Retrieve. An external app can retrieve data received by the framework by using the Retrieve functionality. The external app must be listening on the reply address provided during Register before calling the Retrieve functionality as the data will be sent directly to the reply address provided. All data received by the framework since the last retrieval will be sent. A passcode is also required to retrieve the data.

Passerbys. Apart from retrieving data specific to an external app, the API also allows users to obtain the following information about devices that the framework has exchanged data with (Passerbys):

1) MAC address of Passerbys, 2) Device name of Passerbys, 3) Time the Passerby was last encountered, and 4) Number of times the Passerby was encountered.

4. IMPLEMENTATION

The framework app that encompasses the framework design and the API discussed in the previous section has been implemented. It requires devices running Android 4.1 (Jelly Bean) and above with Wi-Fi capabilities. The app currently uses Wi-Fi-Direct's SBD as the method of seamless data transfer. Developers who want to use the framework must have both the framework app and their app on both sending and receiving devices. Communication between the developer's app and the framework app must be done using the API provided. A demo of the framework is available at project page [1].

5. EVALUATION METHODOLOGY

Our goal is to make the framework pratically useable by the app developers. Hence, we focused on evaluating the feasibility and communication efficiency of the framework under different power saving requirements and crowd densities.

5.1 Power Measurement

To measure the amount of power consumed, the following tests are carried out: 1) Theoretical estimation of the power consumption of the Wi-Fi chip with reference to the amount of usage according to the design of the framework, and 2) Real power consumption measurement on an Android device to verify the usability of the framework in a real-world scenario.

The framework requires broadcasting in constant intervels to find the neighbours. As this will result in high in power consumption, we have employed duty-cycling techniques and studied their effects on overall power and communication efficiency. We have studied the effects of manipulating broadcast period, static and dynamic sleep intervals. The broadcast period reduction sets the broadcast period between 10 and 120 seconds as compared to the default 120 seconds. This is paired with a sleep interval to achieve a reduction in overall broadcast duration. A sleep interval is a period of time whereby the framework stops Wi-Fi Direct's SBD. The static interval is set between 10 and 120 seconds. The dynamic interval is set to 10 seconds initially and is automatically adjusted later depending on the approximated network density. Automatic dynamic interval adjustment method is given below with an illustration in Figure 6.

Automatic dynamic interval adjustment:

1) Initially, sleep interval is set to 10 seconds.

2) After X amount of broadcast-sleep cycles, double the amount of sleep interval time (increases duration between sleep interval increment.

3) Repeat broadcast-sleep cycles and doubling process, keeping the maximum sleep interval to 120 seconds (ie:10, 20, 40, 80, 120).

4) If any device is discovered during any of the broadcast-sleep cycles, set the sleep interval back to 10 seconds.

The dynamic sleep interval is designed this way due to the assumption that there is a high chance of increase in network density after a device has been discovered. The results of the two tests with and without the power saving methods are then compared to verify the power efficiency of the framework.

5.2 Simulation

A reasonabally realistic user study is hard and requires much larger time to conduct. Large number of users for several weeks with live GPS and Indoor location tracking facility are required for a decent study. Instead, a simulation is carried out to verify the viability of the design and the power consumption of the framework.

The simulation is designed to mimic the exchange of data and power consumption of the framework in the real-world use case.

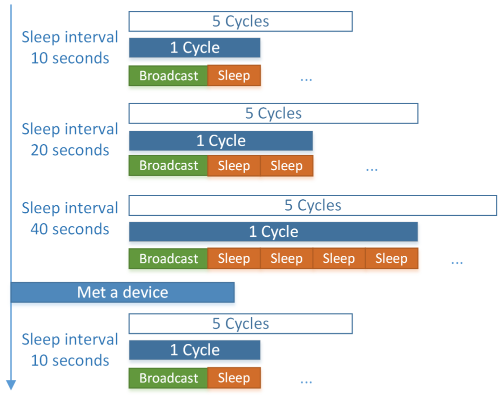


Figure 6: How Dynamic Sleep Interval works

The goal is to measure the difference in contact/communication efficiency of the framework with different power saving methods implemented. The following describes how the simulation is designed to track a single user device with multiple other devices passing by,

1) The user's device will be running the framework, with and without power saving implemented, continuously from the start till the end of simulation. 2) Devices passing by will be running the framework with the following randomly generated variables: i) State of the framework (broadcasting / sleep) when device meets the user, ii) Time remaining in the current state, and iii) Amount of time the device will stay within the range of the user; minimum of 1 second and maximum of 60 seconds.

3) Interval between devices passing by are randomly generated according to the following network types: *Dense* - maximum of 5 minutes interval, *Medium* - maximum of 30 minutes interval, and *Sparse* - maximum of 6 hours interval.

4) For each network type, 2 test cases of 1000 devices passing by are randomly generated.

As the design of the framework makes it impossible to exchange data between devices that are not broadcasting concurrently, all passing by devices are configured to broadcast for at least 1 second(after exiting sleep state) during the time it is within range of the user. This improves the accuracy of the simulation as devices that do not broadcast when they are within the range should not be measured, since these devices do not contribute to the measure of devices missed due to the power saving methods implemented.

The simulation is repeated for all networks types with every combination of the power saving methods mentioned in the previous section. In each simulation round, the following variables are measured - *Total broadcast duration, Devices met, Devices missed, and Efficiency measure*. As the power consumption of the Wi-Fi can vary across different devices, the total broadcast duration is used as a common variable for power measurement in the simulation. The efficiency of each simulation is determined by the number of devices met and total broadcast duration in comparison with the control simulation. This allows a clear comparison between the efficiency of the default framework and the framework with power saving methods implemented.

6. EVALUATION RESULTS

The parameters used in the evaluation are given in Table 2. The results show the feasability of power efficient communication with our framework.

6.1 Power Measurement

The theoretical estimation of power consumption of the framework takes into account only the Wi-Fi power consumed as the there are no intensive operations in the framework app. As such,

Parameter	Description
Duration of tests	1 hour
Control (Default)	Constantly broadcasting
Static 10	10s broadcast, 10s sleep per cycle
Static 60	10s broadcast, 60s sleep per cycle
Static 120	10s broadcast 120s sleep per cycle
Dynamic Sleep 1	5 cycles, 10s broadcast
Dynamic Sleep 2	5 cycles, 10s broadcast, met device at approximately 30 minute mark

Table 2: Experiment parameters

Mode	Active State (sec)	Standby State (sec)
Control	3600	0
Static Sleep 10	1800	1800
Static Sleep 60	540	3060
Static Sleep 120	360	3240
Dynamic Sleep 1	450	3150
Dynamic Sleep 2	520	3080

Table 3: Estimated duration of active and standby states

the power consumed by the CPU while using the framework app is considered to be negligible.

A more generic approach is taken for the theoretical calculation. Table 3 shows the number of seconds the Wi-Fi chip is in active and standby state across the different power saving methods over 1 hour. *Control mode* represents the default framework without any power saving features. As an example, these values are then used to calculate the power consumption on a Galaxy Nexus whose Wi-Fi draws 120mA & 4mA in Wi-Fi active and Wi-Fi on (standby) modes respectively at 3.7 Volts. The following formula is used: $Amperage \times Time(hour) = Power\ Consumption\ (mAh)$.

The results are shown in Table 4. The broadcast period reduction and sleep intervals did play a part in reducing the power consumption of the Wi-Fi chip. The power consumption differences were mainly in line with the proportion of broadcast duration to sleep interval ratio. The results are also verified with real power measurement with Xiaomi RedMi 1 Smartphone. Keeping the default (control mode) framework turned on for 1 hour it consumed about 3% of battery and for static sleep mode it consumed 2%. Note, Wi-Fi is completely turned OFF during sleep the interval. The power savings are slightly less than simulation due to the penalties involved in turning the Wi-Fi to ON and OFF states.

6.2 Simulation

The simulation was run with the following combination of power saving settings:
Control: Broadcasting the entire duration of simulation.
Broadcast period (seconds): 10, 20, 30, 60, 90, 120.
Static sleep interval (second): 10, 20, 40, 80, 120.
Dynamic sleep interval cycles: 3 (very fast), 5 (fast), 10 (slow).

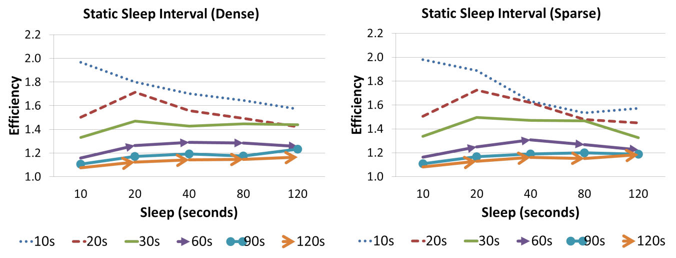
The simulation results of every single combination can be found in the project homepage [1]. The Efficiency Measure(EM) is calculated using the following formula:

$$\frac{Devices\ Met}{Total\ Number\ of\ Devices} \div \frac{Broadcast\ Duration}{Control\ Broadcast\ Duration} = EM$$

The control mode simulation will give an efficiency measure of 1; an efficiency measure of more than 1 means that it is better than the control. Figure 7 shows the efficiency measure of the framework with static sleep intervals across the dense and sparse

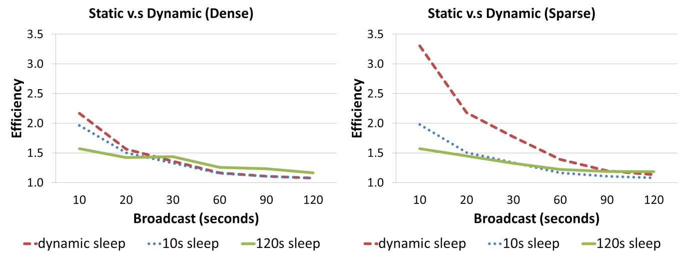
Mode -	Power Consumption	Comparison with Control
Control	120 mAh	1.0
Static Sleep 10	62 mAh	0.52
Static Sleep 60	21.4 mAh	0.18
Static Sleep 120	15.6 mAh	0.13
Dynamic Sleep 1	18.5 mAh	0.15
Dynamic Sleep 2	20.8 mAh	0.17

Table 4: Approximated power consumption on a Galaxy Nexus



(a) Dense (b) Sparse

Figure 7: Efficiency of static sleep interval in different networks



(a) Dense (b) Sparse

Figure 9: Comparison of efficiency of dynamic and static sleep intervals different networks

networks. As observed from Figure 7, static sleep intervals with shorter broadcast periods and sleep intervals achieved better efficiency across sparse and dense networks. Efficiency is best when broadcast period to sleep interval ratio is 1:1. Most optimal value is achieved when both broadcast period and sleep interval are set to 10 seconds.

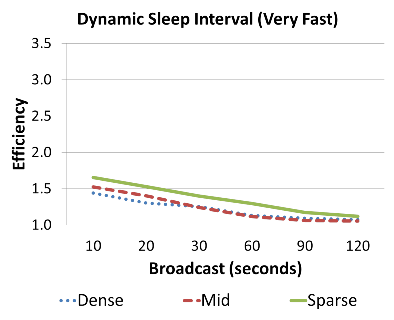
Figure 8 shows the efficiency measure of the framework with very fast, fast and slow dynamic sleep interval cycles across the dense, medium and sparse networks. As observed from Figure 8, fast dynamic sleep interval is slightly more efficient than slow dynamic sleep interval. Fast and slow dynamic sleep interval has better efficiency in sparse network as compared to dense network. Very fast dynamic sleep interval suffers a significant drop in efficiency.

Figure 9 shows a comparison between the minimum and maximum static sleep intervals, and the fast dynamic sleep interval over both dense and sparse networks. Dynamic sleep interval is more efficient across both dense and sparse networks. Dynamic sleep interval is significantly more efficient in the sparse network as compared to the static sleep intervals.

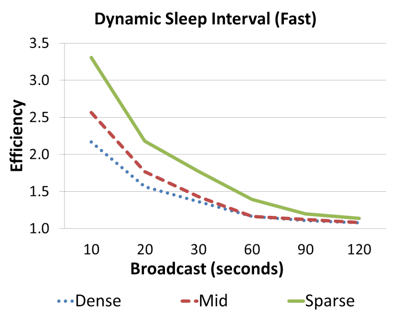
As such, from the observations, the dynamic sleep interval power saving method proved to be a much better choice in real-world usage due to its ability to adapt the power consumption according to the current density of the network.

6.3 Discussions

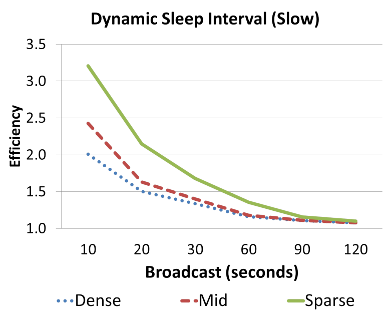
Power Saving Methods and Variables. For the best efficiency in terms of power consumption to devices-met ratio, it is recommended that the dynamic sleep interval power saving method should be used in combination with shorter broadcast period. The following values are recommended: 5 cycles dynamic sleep interval and 10 seconds broadcast period. However, even though the above recommended settings give the best efficiency, it might not be the most optimal setting for real-world use case. This is because of significant drop in the percentage of devices met. Figure 10 highlights the problem with the dynamic sleep interval. Referring to Figure 10, it can be seen that, hit rate of the slow cycle dynamic sleep interval has slightly higher number of devices met. Significant drop in hit rate for both dynamic sleep intervals in sparser networks. Broadcast period of 10 seconds has the worst performance.



(a)

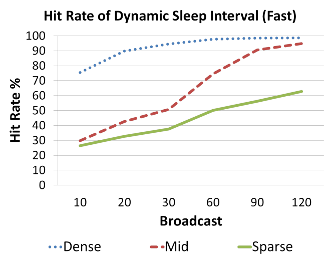


(b)

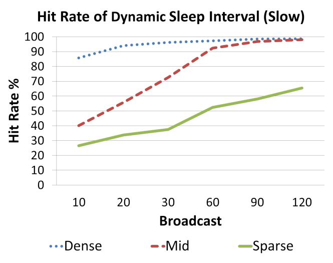


(c)

Figure 8: Efficiency of dynamic sleep interval with very fast(a), fast(b) and slow(c) cycles



(a) Fast



(b) Slow

Figure 10: Comparison of hit rate of different dynamic sleep intervals

Broadcast (s)	Sleep (s)	Duration (s)	Hits	Misses	HitRate %	Efficiency
10	10	79254	901	99	90.09	1.966
10	120	13215	121	879	12.01	1.572

Table 5: Partial results of static sleep interval simulation

This problem is further highlighted using the static sleep interval simulation results as shown in Table 5. Dynamic sleep interval perform well with shorter sleep intervals. when When sleep interval increases, there is a significant drop of hit rate to 12%. There is only 12% chance for broadcasting speed to return to maximum. As it is unclear how this drastic drop in hit rate will affect the real-world use case, a large scale user study is required, which is deferred to future work.

Power Consumption and Usability. The low power consumption of the framework with and without power saving implemented shows that it is feasible for real-world use. When power saving mechanisms are enabled it reduces power consumption by approximately 30% to 60% depending on network density, consumes approximately 25% battery power over 10 hours in dense network, and averages around 15% battery power consumption over 10 hours, depending on usage.

However, a major problem that plagues the power saving methods is Android's implementation of Wi-Fi Direct. The power consumption does not reduce even with the power saving methods implemented, due to some issues with Android's implementation of Wi-Fi Direct. This proves to be a problem as constantly turning on and off the Wi-Fi of a user's device is not something an app should do; it will affect the user if they are using Wi-Fi for Internet. As such, the viability of power reduction is not possible with the current Android API.

Due to the limitations of Android API, the real-world usability of the app is slightly affected. Users of the app must set aside slightly more battery power for it as compared to a power saving version of the app.

Overall - The Framework is a Viable Solution. From the above discussions, it is clear that the framework system can be pratically used for seamless peer-to-peer communication by applications in user space without getting superuser permissions. The framework consumes less than 3% (per hour) of the battery energy even without power saving measures which makes it suitable for real-world use cases. Overall, the framework system is viable, works very well, simple and ready to use.

7. CONCLUSION

This paper discussed the technical factors involved in ensuring the feasibility of a framework to transfer data between nearby smartphones without the need of pairing, user authentication and Internet. The framework is evaluated to show its communication and power efficiency under various conditions and real-world use cases. A demo is available at project homepage [1]. The framework will be open sourced.

8. REFERENCES

[1] Anand, B. and Wei, T. G. *Seamless Message Transfer on Smartphones - Demo*. National University of Singapore. `http://www.comp.nus.edu.sg/~bhojan/fats/index.html`, Nov 2014.

[2] Bisio, I. et al. Smartphone-based automatic place recognition with wi-fi signals for location-aware services. *Communications (ICC), 2012 IEEE International Conference on*, pages 4943–4948, June 2012.

[3] Gressmann, B. et al. Towards ubiquitous indoor location based services and indoor navigation. *WPNC, 2010 7th Workshop on*, pages 107–112, March 2010.

[4] Kao, K.-F. et al. An indoor location-based service using access points as signal strength data collectors. *IPIN, 2010 International Conference*, pages 1–6, Sept 2010.

[5] Konstantinidis, A. et al. Smartp2p: A multi-objective framework for finding social content in p2p smartphone networks. *MDM, 2012 IEEE 13th International Conference on*, pages 324–327, July 2012.

[6] Pyattaev, A. et al. Proximity-based data offloading via network assisted device-to-device communications. *VTC Spring, 2013 IEEE 77th*, pages 1–5, June 2013.

[7] Sadhukhan, P. et al. A scalable location-based services infrastructure combining gps and bluetooth based positioning for providing services in ubiquitous environment. *IMSAA, 2010 IEEE 4th International Conference on*, pages 1–6, Dec 2010.

[8] Salmon, J. and Yang, R. A proximity-based framework for mobile services. *MS, 2014 IEEE International Conference on*, pages 124–131, June 2014.

[9] Toledano, E. et al. Cocam: A collaborative content sharing framework based on opportunistic p2p networking. *CCNC, 2013 IEEE*, pages 158–163, Jan 2013.

[10] Trifunovic, S. et al. Wifi-opp: Ad-hoc-less opportunistic networking. *Proceedings of the 6th ACM Workshop on Challenged Networks*, CHANTS '11, pages 37–42, New York, NY, USA, 2011. ACM.

[11] Yun, M. et al. Silent broadcast: Experience of connectionless messaging using wi-fi p2p. *ICIDT, 2012 8th International Conference on*, volume 2, pages 239–242, June 2012.