IJCSC 0973-7391 Volume 7 • Number 2 March 2016 - Sept 2016 pp. 20-34 available online at www.csjournalss.com

Mobile Torrent: Peer-To-Peer File Sharing In Android Devices

Rutul Shah[1], Zunnun Narmawala[2]
[123]Institute of Technology, Nirma University,Computer Science and Engineering Department, Ahmedabad, India.
[1]14mcen20@nirmauni.ac.in, [2]zunnun80@gmail.com

Abstract: Peer-to-peer file sharing is very popular on the Internet. But, to use it, one has to be connected to the Internet and it incurs significant data cost. We are developing an android application for peer-to-peer file sharing named "Mobile Torrent" using which files can be shared between smartphones within a campus without using the Internet. Each application user can share files and can register a request to download a file shared by any other application user even if the source is not directly connected. Initially, we have implemented the same for single hop; i.e. when two smartphones come in the Wi-Fi communication range of each other, they auto-discover each other using Wi-Fi Direct and automatically exchange required files.
Keywords: Opportunistic network, Delay Tolerant Network (DTN), Peer-to-peer file sharing, Android

1. Introduction

File sharing provides the mechanism to share different kinds of files between devices. Peer-to-peer file sharing is a different approach for file sharing. This kind of file sharing became in existence with the cooperation of the users who have the file and those who want the file. Bit-torrent is the well-known example which works without any central server but it requires the Internet to connect to the available peers. Further, users incur significant data cost to share and download files.

Today, smartphones are becoming very popular. These devices have evolved from very simple mobile phones to devices using which users are able to browse the Internet, access and read emails, and watch video streams directly from the network. Further, these phones can connect with each other using Wi-Fi or Bluetooth. So, an entirely new network paradigm has emerged which is called as an opportunistic network. In this type of network, when two smartphones come in the communication range of each other, they can discover each other automatically and transfer packets, i.e. communication between devices takes place whenever there is an opportunity. As Wi-Fi supports larger communication range and better data rates as compared to Bluetooth, it is preferred for such communication.

We are developing an android application for peer-to-peer file sharing named "Mobile Torrent" using which files can be shared between smartphones within a campus opportunistically without using the Internet. Each application user can share files and can register a request to download a file shared by any other application user even if the source is not directly connected. Initially, we have implemented the same for single hop; i.e. when two smartphones come in the Wi-Fi communication range of each other, they auto-discover each other using Wi-Fi-Direct and automatically exchange required files. Ideally, instead of using Wi-Fi Direct, an ad hoc network of smartphones should be established which is truly peer-to-peer. But, unfortunately, android does not support ad hoc networking. So, we are using Wi-Fi Direct instead.

The paper is organized as follows. The Related work is discussed in Section 2. In Section 3, we discuss implementation details of the application in android. Section 4 concludes the paper and Section 5 presents the future work.

2. Related Work

In the literature, researchers have proposed peer-to-peer file sharing for the opportunistic network. But, all of them are simulation based. There is no work in our knowledge which actually implements peer-to-peer file sharing for the opportunistic network. As, in the opportunistic network, there may not be any end-to-end path between the source of a file and an interested user's smartphone, there may be a significant delay in delivering a file. So, an opportunistic network is a type of Delay Tolerant Network (DTN).

DTN needs not be solely concerned with deep space communications but can also be useful in terrestrial networks [1]. In some environment, networks may be subjected to a high probability of disruption. One example is in military applications, where adopting DTN allows the retrieval of critical information in mobile battlefield scenarios using only intermittently connected network communications. Another more peaceful application is the adoption of DTN to overcome a major natural disaster. In such a situation terrestrial infrastructures may have been swept away and delay tolerant protocols must be used to coordinate rescue teams. DTN networks are characterized by i) intermittent connectivity and ii) long delivery delay [2].

To overcome problems associated with all these factors, DTN uses store-carry-forward mechanism as shown in Fig. 1; i.e. a forwarding node not only stores received packet but also carries the packet till suitable forwarding opportunity arises. Replication techniques can be adopted to increase the delivery ratio, copying the carried data and passing it to other nodes following a different physical path.

In DTN, messages are stored in storage mediums which can hold them for long periods of time, called persistent storage. It is different from Internet protocols as in the Internet, routers need to store the incoming packet in very short-term storage for few millisecond in a queue for the next hop. In DTN, routers require long term storage.

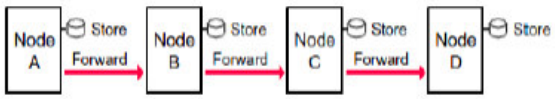

Figure 1. Example of the store-carry-forward technique [3]

In an opportunistic network, mobility and low node density require new strategies to implement a file sharing application which allows the exchange of multimedia contents between mobile devices. M2MShare [4] uses Bluetooth to create a peer-to-peer overlay network and uses node mobility to deliver data content to other local disconnected networks. The main idea of this protocol is to use a store-delegate-forward model. This is similar to the store-carry-forward strategy widely used in opportunistic networks, but M2MShare adds the delegate part to the model. The result is an asynchronous communication model where a peer delegates an unaccomplished task to other peers in the overlay network.

Delegations are used to extend search and diffusion area for a shared file while M2MShare dynamically establishes forward routes along the destination path by exploiting social relations existing between peer users. This is done by limiting delegations only to frequently encountered peers, which visit the same geographical location at the same time frequently enough. The history of previous encounters is then used as the heuristic evaluation of whether a peer is a good candidate for delegations. This allows assigning tasks only to nodes that we should meet again in the future so that they can return the result of the task back to us. M2MShare uses an asynchronous communication strategy in which a client peer, in search for a file, can delegate to another peer (a servant) the task of searching for the file and returning it to the requester. Delegation system is at the root of M2MShare. This permits widely extending the area of where to look for the searched file, in a network composed of spread out and poorly connected nodes. The performance of M2MShare is evaluated in [5] through simulation.

ORION [6] is another protocol based on an application layer overlay network for peer-to-peer file sharing in an opportunistic network. In an overlay network, overlay routes are set up on demand by using the searching algorithm and redundant transfer path on per file basis. ORION protocol combines the network layer and application layer for routing all type of messages like queries, response, and file transmission. Benefits of using ORION protocol are reduced control overhead, low overhead file transfer and increased probability of successful file transfer. ORION protocol is simulated in the NS-2 simulator and it enables more reliable file transfer with low overhead [7].

Network coding can be used as a tool to improve the delivery probability of a file to an interested user. In network coding, a forwarding node linearly combines two or more incoming packets into one or more outgoing packets. In peer-to-peer file sharing, a file is divided into chunks of equal size. Ideally, nodes should forward chunks such that all chunks have the same number of copies in the network. But, it is not possible to keep track of the number of copies of a chunk in the network. In network coding, each combined chunk contributes the same to the eventual delivery of the file because network coding does not require delivery of the specific chunk but it requires enough number of independent chunks. A review of network coding based peer-to-peer file sharing in DTN is available in [8]. We intend to implement network coding in our android application in future.

From the related work, we conclude that no actual implementation of peer-to-peer file sharing in opportunistic network is available. So, we propose one such implementation in this paper. As smartphones are very common and android is the most popular smartphone operating system, we have implemented the application in android.

3. Implementation

File sharing provides a facility to the users to share files they have with other users. Peer-to-peer file sharing provides the advantage that a node can act as a server as well as a client. It can be the source of a file and at the same time, it can download a file from other nodes as a client. In this application, we have used WiFi-Direct. WiFi-Direct does not require any wireless access point to connect with each other. This is relevant for applications that share data among the users such as file sharing, photo sharing or multi-layer gaming. In the

following subsections, we discuss the implementation of the application in detail. The overall design of the application is depicted in Fig. 2.

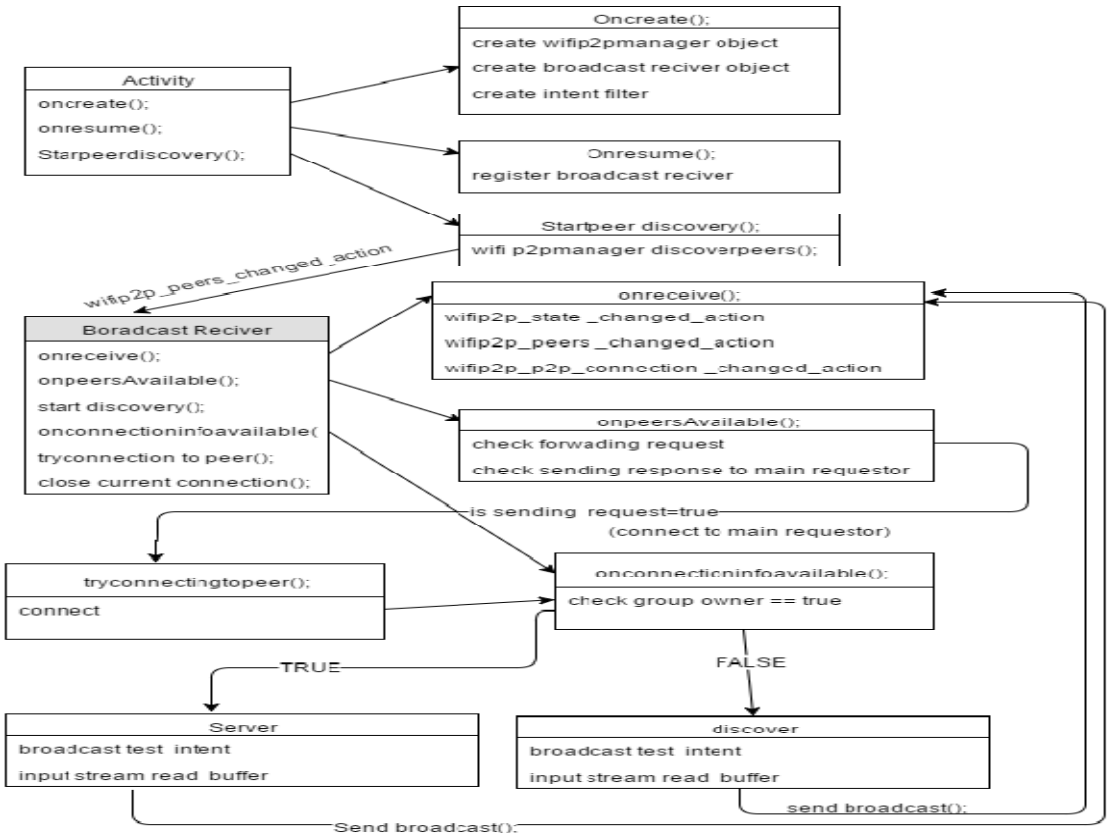

Figure 2. Design of the application

3.1 Android

Android provides WiFi-Direct in the versions above 4.0. This version provides multitasking environment, deep interactivity and powerful new ways for communication and sharing. It also supports WiFi-Direct for connectivity without any Internet or tethering. It lets us connect directly to the nearby peers. Android has different components to provide different functionality. Basically, android application is written based on Java programming language.

Components of android application: There are different components of the android application, each of them serve a different purpose. The life cycle of these components defines how they are created and how they get destroyed. Some of the important components are described here [9]:

- Activities: Activities interact with a user. So, an activity presents the user with the screen to interact with. Activities in the system are managed using a stack. Whenever an activity is started, it is placed on the top of the stack and it starts running. Any previous activity cannot be resumed until the top running activity is finished. Activity is implemented as the subclass of *activity*. Activity has different methods. The full lifetime of the activity is from *onCreate()* to *onDestroy()*, visible lifetime is from *onStart()* to *onStop()*, and foreground lifetime is from *onPause()* to *onResume()*.
- Services: Services run in the background to perform the operations. Even if an application is not running, its service continues to run in the background. Services perform inter-process communication. To create a service, we need to implement the subclass of the *service* class and s*tartService()* and *stopService()* are the methods to invoke services.
- Content providers: Content provider manages the access to the data. It provides the interface which connects the data in one process and code running in another process. There is no need to create a content provider for the application if there is no need to share data.
- Broadcast receivers: To receive system-wide broadcast announcements as well as for providing the response towards it, we use the broadcast receiver. To use this, we need to implement the subclass of B*roadcast Receiver*class.

- Intents and intent filter: To request any action from an application, we need to pass intent as an object. There are two types of intents: implicit and explicit. The intent object carries the information of which components to start within an application.
- Manifest: This should be present at the root of the directory. It defines permissions required for the application. It also defines the API level required for the application. Our Mobile Torrent application works on the API level equal to or greater than 14.
- Resources: Application interface requires images as well as layouts. These are defined in resources. The layout files are defined in XML. Layout files represent user interface of an application. It consists of different tabs, buttons, and text fields etc.
- Generated Java files: There are two auto-generated Java files present in the system. First is *R.java* and second is *Buildconfig.java*. *R.java* file contains the unique identifiers for the resources for the easy access of the elements. While *BuildConfig.java* has a debug constant that is automatically set according to build type.

3.2 Initializing the Application

For using WiFi APIs, the application must be able to access the hardware and the smartphone should have the support of WiFi-Direct. We can check this at broadcast receiver when it receives the WiFi P2P STATE CHANGED ACTION intent.

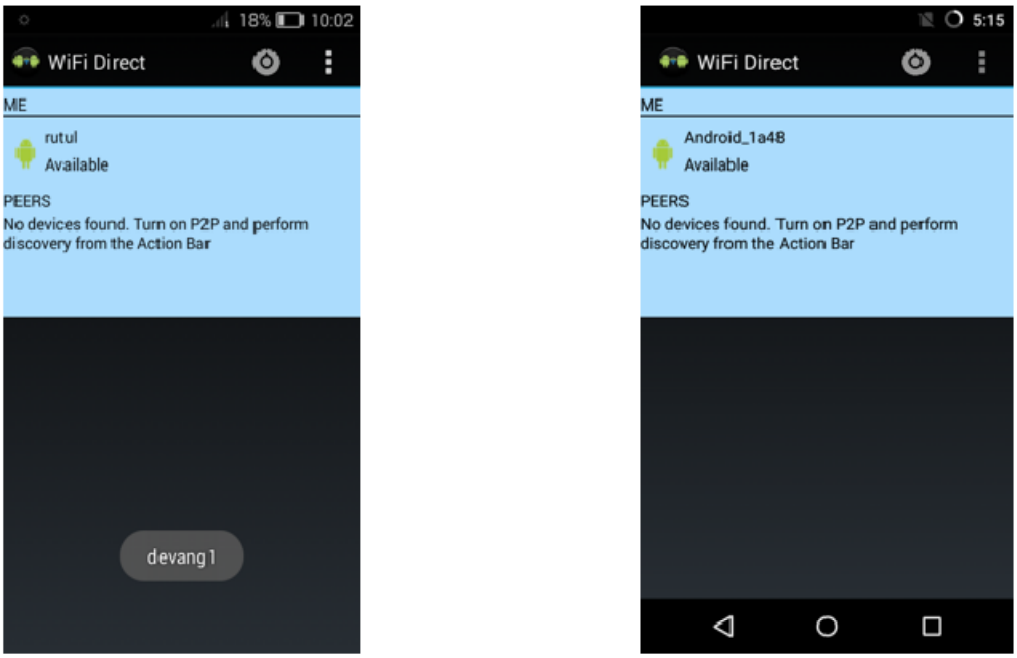

Figure 3. Starting the application

Following are the steps to use WiFi-Direct. In the *onCreate()* method, take the instance of *WiFiP2pManager* and call *initialize()* to register the application with the WiFi P2P framework. This method returns a *WiFIP2pManager.Channel*, which helps the application to connect to the WiFi P2P framework. Create an intent filter and add same intents for which the broadcast receiver checks. In the *onResume()* method, register the broadcast receiver and unregister it in the *onPause()* method of the current activity. Now use Wi-Fi P2P features by calling the methods in *WiFIP2pManager*. Fig. 3 shows screen shots of the application when it is started.

3.3 Auto Discovering Peers

For discovering peers which are available within the communication range, call *discoverPeers()*. This call is asynchronous and success or failure is communicated to the application via *onSuccess()* and *onFailure()*, if *WifiP2pManager.*

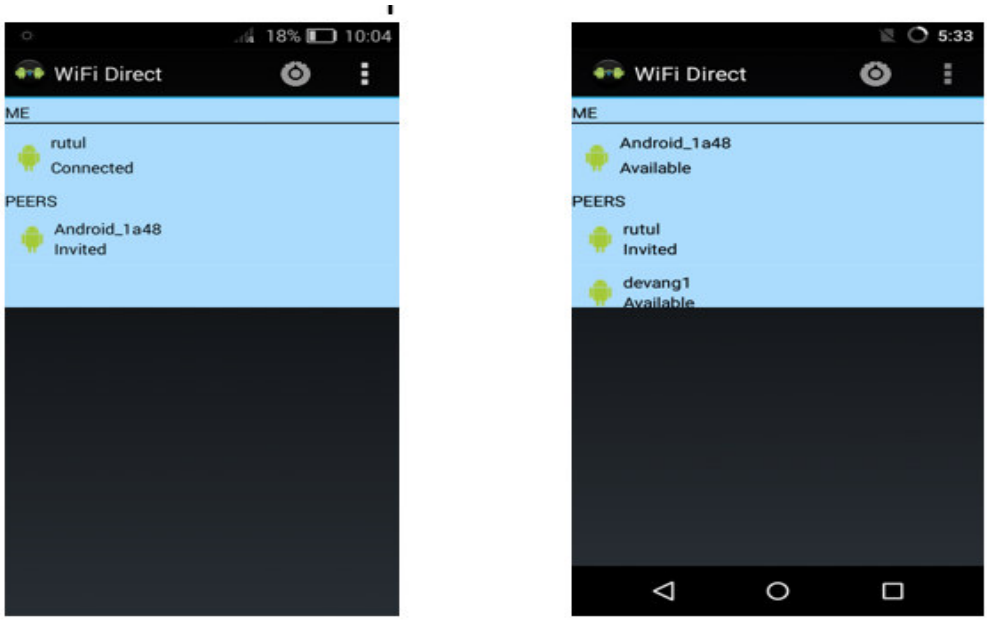

Figure 4. Discovered peers

For discovering peers which are available within the communication range, call *discoverPeers()*. This call is asynchronous and success or failure is communicated to the application via *onSuccess()* and *onFailure()*, if *WifiP2pManager.ActionListener* is already created. The *onSuccess()* method would notify only about the success of peer discovery and would not provide any information about the discovered peers. Fig. 4 shows discovered peers.

3.4 Auto Connecting Peers

The *discoverPeers()* method obtains the list of possible peers in the WiFi range of a device. Then, the application connects to the first device among the list of devices. For the same, the application also needs a *WifiP2pConfig* object that contains the information about which device wants to connect. Fig. 5 shows connected peers.

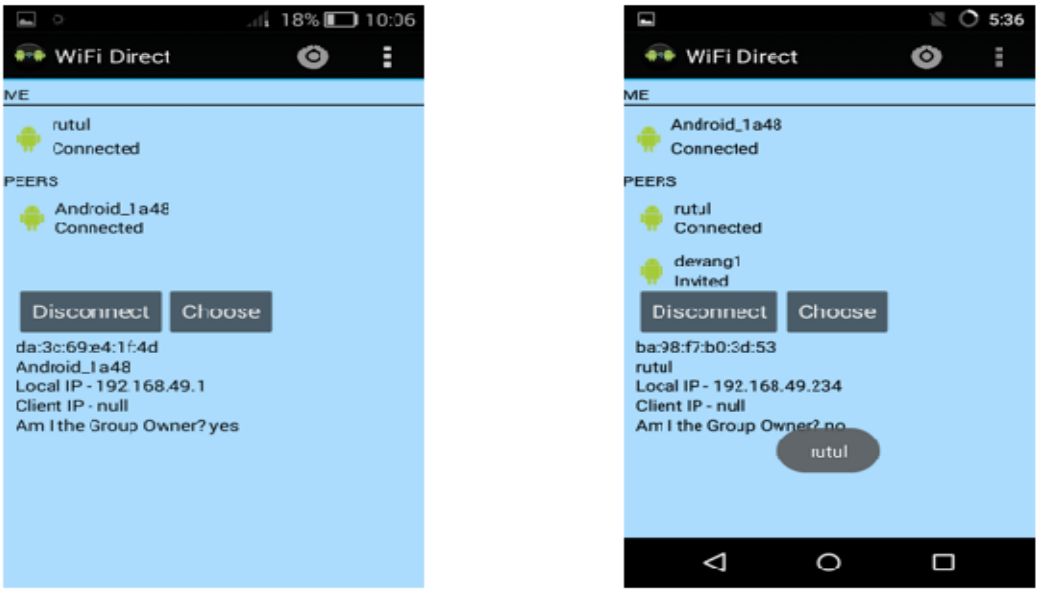

Figure 5. Connected peers

Once two devices are connected, they exchange shared files with each other and then disconnect themselves. Please note that in the entire process, there is no user intervention.

4. Conclusion

Peer-to-peer file sharing between smartphones is explored in the literature only through simulation. We have developed an actual android application for the same. For opportunistic networking, it is required that phones should be able to discover other phones when they come in the communication range. A phone should also be able to connect to a neighbour phone and transfer required content. All these should be done automatically without any user intervention. We are successful in achieving the same using WiFi-Direct.

5. Future Work

Currently, our application transfers a single file to an immediate neighbour successfully. The next step is to implement network coding based multicopy forwarding algorithm while dividing a file into chunks.

References

[1]. K. Fall, "A delay-tolerant network architecture for challenged internets," in Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 27–34, ACM, 2003.

[2]. Bujari and C. E. Palazzi, "Opportunistic communication for the internet of everything," in Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th, pp. 502–507, IEEE, 2014.

[3]. C. E. Palazzi and A. Bujari, "A delay/disruption tolerant solution for mobile-to-mobile file sharing," in Wireless Days (WD), 2010 IFIP, pp. 1–5, IEEE, 2010.

[4]. C. E. Palazzi, A. Bujari, and E. Cervi, "P2p file sharing on mobile phones: Design and implementation of a prototype," in Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on, pp. 136–140, IEEE, 2009.

[5]. A. Bujari, C. E. Palazzi, and D. Bonaldo, "Performance evaluation of a file sharing dtn protocol with realistic mobility," in Wireless Days (WD), 2011 IFIP, pp. 1–6, IEEE, 2011.

[6]. A. Duran and C.-C. Shen, "Mobile ad hoc p2p file sharing," in Wireless Communications and Networking Conference, 2004. WCNC. 2004 IEEE, vol. 1, pp. 114–119, IEEE, 2004.

[7]. A. Klemm, C. Lindemann, and O. P. Waldhorst, "A special-purpose peer-to-peer file sharing system for mobile ad hoc networks," in Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th, vol. 4, pp. 2758–2763, IEEE, 2003.

[8]. V. Parikh and Z. Narmawala, "A Survey on Peer-to-Peer File Sharing using Network Coding in Delay Tolerant Networks" in International Journal of Computer Science and Communication (IJCSC), vol. 4, pp. 74-79, 2014.

[9]. "Wi-Fi direct: peer to peer file sharing, http://developer.android.com/training/connect-devices-wirelessly/wifidirect. html."