

Title Page

# Android App for File Transfer

(using a bloom filter)

PROJECT REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF TECHNOLOGY.

IN

**INFORMATION AND TECHNOLOGY**

UNDER THE SUPERVISION OF

Mr.Shailendra Shukla

BY

Rishabh Bhaskar

111472

to



Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

## CERTIFICATE

This is to certify that project report entitled “....**Android App for File Transfer (using a bloom filter)**....”, submitted by .....**Rishabh Bhaskar**..... in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

**Date:**

**Supervisor's Name**

**Designation**

## Acknowledgement

Every project big or small is successful largely due to the effort of a number of wonderful people who have always given their valuable advice or lent a helping hand. I sincerely appreciate the inspiration; support and guidance of all those people who have been instrumental in making this project a success.

I, RISHABH BHASKAR, the student of JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY (IT), am extremely grateful to “JUIT” for the confidence bestowed in me and entrusting my project entitled “ANDROID APP FOR FILE TRANSFER USING A BLOOM FILTER” at this juncture I feel deeply honored in expressing my sincere thanks to our project supervisor Mr. Hemraj Saini for making the resources available at right time and providing valuable insights leading to the successful completion of my project.

I also extend my gratitude to my Project Guide Mr. SHAILENDRA SHUKLA, who assisted me in compiling the project.

I would also like to thank all the faculty members of JUIT for their critical advice and guidance without which this project would not have been possible. Last but not the least I place a deep sense of gratitude to my family members and my friends who have been constant source of inspiration during the preparation of this project work.

Date:

RISHABH BHASKAR

# Table of contents

Sr. No.	Content	Page No.
<b>1</b>	<b>Abstract</b>	1
<b>2</b>	<b>Introduction</b>	2-3
<b>3</b>	<b>Preliminaries</b>	4-7
<b>4</b>	<b>Implementation</b>	8-29
<b>5</b>	<b>Results</b>	30-33
<b>6</b>	<b>Future Work</b>	34
<b>7</b>	<b>References</b>	35

## List of Tables and Figures

### Tables

1. WiFi P2P Methods (Pg : 9)
2. WiFi P2P Listeners (Pg : 10)
3. WiFi P2P Intents (Pg : 11)

## Figures

1. Bloom Filter Algorithm(Pg:6)
2. Architecture of the App(Pg:23)
3. App default page (Pg:30)
4. Browsing of Directories Screenshot(Pg:30)
5. File Sent Screenshot(Pg:31)
6. Insertion in Bloom Filter(Pg:32)
7. Check Element in Bloom Filter(Pg:32)
8. Check Size and Clear bloom Filter(Pg:33)

# Abstract

WiFi Direct File Transfer is an open source application that will enable sharing of data between Android devices running Android 4.0 or higher using a WiFi direct connection without the use of a separate WiFi access point. This will enable data transfer between devices without relying on any existing network infrastructure. This application is intended to provide a much higher speed alternative to Bluetooth file transfer.

There will be a client and a server i.e. the two devices that are connected through wifi direct. Once the connection is established, you can transfer data through sockets. Send data from the client to the server. When the client socket successfully connects to the server socket, you can send data from the client to the server with byte streams. When a connection happens, the server device can receive the data from the client. Carry out any actions with this data, such as saving it to a file or presenting it to the user.

Another prospect of this project is implementing a bloom filter in the app so as to expedite the selection of files to be exchanged. The files from the devices will be hashed into a bloom filter and then, a comparison will be made between the files present in the two devices and the missing files in the receiving device will be transferred to it.

# Introduction

## **WiFi Direct**

**Wi-Fi Direct**, initially called Wi-Fi P2P, is a Wi-Fi standard enabling devices to easily connect with each other without requiring a wireless access point. It is usable for everything from internet browsing to file transfer, and to communicate with more than one device simultaneously at typical Wi-Fi speeds. One advantage of Wi-Fi Direct is the ability to connect devices even if they are from different manufacturers. Only one of the Wi-Fi devices needs to be compliant with Wi-Fi Direct to establish a peer-to-peer connection that transfers data directly between them with greatly reduced setup.

Wi-Fi Direct negotiates the link with a Wi-Fi Protected Setup system that assigns each device a limited wireless access point. The "pairing" of Wi-Fi Direct devices can be set up to require the proximity of a near field communication, a Bluetooth signal, or a button press on one or all the devices. Wi-Fi Direct may not only replace the need for routers, but may also replace the need of Bluetooth for applications that do not rely on low energy.

## **Working of Wifi-Direct**

### **Technical description**

Wi-Fi Direct essentially embeds a software access point ("Soft AP"), into any device that must support Direct. The soft AP provides a version of Wi-Fi Protected Setup with its push-button or PIN-based setup.

When a device enters the range of the Wi-Fi Direct host, it can connect to it, and then gather setup information using a Protected Setup-style transfer. Connection and setup is so simplified that some suggest it may replace Bluetooth in some situations.

Soft APs can be as simple or as complex as the role requires. A digital picture frame might provide only the most basic services needed to allow digital cameras to connect and upload images. A smart phone that allows data tethering might run a more complex soft AP that adds the ability to bridge to the Internet. The standard also includes WPA2 security and features to control

access within corporate networks. Wi-Fi Direct-certified devices can connect one-to-one or one-to-many and not all connected products need to be Wi-Fi Direct-certified. One Wi-Fi Direct enabled device can connect to legacy Wi-Fi certified devices.

The Wi-Fi Direct certification program is developed and administered by the Wi-Fi Alliance, the industry group that owns the "Wi-Fi" trademark. The specification is available for purchase from the Wi-Fi Alliance. (Ref: Wikipedia)

## Precedents

1. Superbeam | WiFi Direct Share
2. Wifi Shoot
3. HitcherNet

The above apps , no doubt, are fast in file sharing and have good user interface but none of them uses a bloom filter to keep record of the files to be exchanged. I, therefore, will try to implement bloom filter in my app to make it state of the art.



# Preliminaries

## Android

**Android** is a mobile operating system(OS) based on the Linux kernel and currently developed by Google. With a user interface based on direct manipulation, Android is designed primarily for touchscreen mobile devices such as smartphones and tablet computers, with specialized user interfaces for televisions (Android TV), cars (Android Auto), and wrist watches (Android Wear). The OS uses touch inputs that loosely correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching to manipulate on-screen objects, and a virtual keyboard. Despite being primarily designed for touchscreen input, it also has been used in game consoles, digital cameras, regular PCs (e.g. the HP Slate 21) and other electronics.

Android is the most widely used mobile OS and, as of 2013, the highest selling OS overall. Android devices sell more than Windows, iOS, and Mac OS X devices combined, with sales in 2012, 2013 and 2014 close to the installed base of all PCs. As of July 2013 the Google Play store has had over 1 million Android apps published, and over 50 billion apps downloaded. A developer survey conducted in April–May 2013 found that 71% of mobile developers develop for Android. At Google I/O2014, the company revealed that there were over 1 billion active monthly Android users, up from 538 million in June 2013.

Android's source code is released by Google under open source licenses, although most Android devices ultimately ship with a combination of open source and proprietary software. Initially developed by Android, Inc., which Google backed financially and later bought in 2005, Android was unveiled in 2007 along with the founding of the Open Handset Alliance—a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices. Android's open nature has encouraged a large community of developers and enthusiasts to use the open-source code as a foundation for community-driven projects, which add new features for advanced users or bring Android to devices which were officially released running other operating systems.(Ref:wikipedia)

# *Bloom Filter*

## *Hashing*

A hash function is any function that can be used to map digital data of arbitrary size to digital data of fixed size, with slight differences in input data producing very big differences in output data. The values returned by a hash function are called hash values, hash codes, hash sums, or simply hashes. One practical use is a data structure called a hash table, widely used in computer software for rapid data lookup. Hash functions accelerate table or database lookup by detecting duplicated records in a large file. An example is finding similar stretches in DNA sequences. They are also useful in cryptography. A cryptographic hash function allows one to easily verify that some input data matches a stored hash value, but makes it hard to reconstruct the data from the hash alone. This principle is used by the PGP algorithm for data validation and by many password checking systems.

Hash functions are related to (and often confused with) checksums, check digits, fingerprints, randomization functions, error-correcting codes, and ciphers. Although these concepts overlap to some extent, each has its own uses and requirements and is designed and optimized differently. The Hash Keeper database maintained by the American National Drug Intelligence Center, for instance, is more aptly described as a catalog of file fingerprints than of hash values.

### Properties

Good hash functions, in the original sense of the term, are usually required to satisfy certain properties listed below. The exact requirements are dependent on the application.

1. Determinism
2. Uniformity
3. Defined range
4. Continuity
5. Non-Invertible

## Bloom filter

A **Bloom filter** is a space-efficient probabilistic data structure, conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set. False positive matches are possible, but false negatives are not, thus a Bloom filter has a 100% recall rate. In other words, a query returns either "possibly in set" or "definitely not in set". Elements can be added to the set, but not removed (though this can be addressed with a "counting" filter). The more elements that are added to the set, the larger the probability of false positives.

Bloom proposed the technique for applications where the amount of source data would require an impracticably large hash area in memory if "conventional" error-free hashing techniques were applied. He gave the example of a hyphenation algorithm for a dictionary of 500,000 words, out of which 90% follow simple hyphenation rules, but the remaining 10% require expensive disk accesses to retrieve specific hyphenation patterns. With sufficient core memory, an error-free hash could be used to eliminate all unnecessary disk accesses; on the other hand, with limited core memory, Bloom's technique uses a smaller hash area but still eliminates most unnecessary accesses. For example, a hash area only 15% of the size needed by an ideal error-free hash still eliminates 85% of the disk accesses (Bloom (1970)).

More generally, fewer than 10 bits per element are required for a 1% false positive probability, independent of the size or number of elements in the set

### Algorithm description

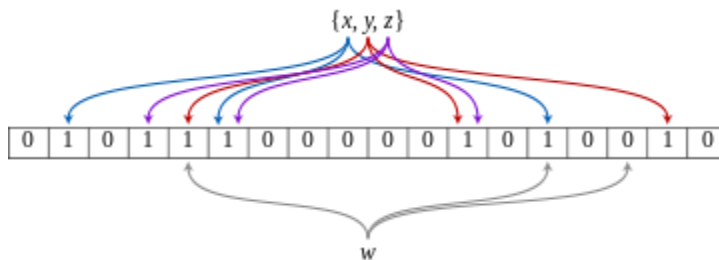


Figure 1

REF: Wikipedia

An example of a Bloom filter, representing the set  $\{x, y, z\}$ . The colored arrows show the positions in the bit array that each set element is mapped to. The element  $w$  is not in the set  $\{x, y, z\}$ , because it hashes to one bit-array position containing 0. For this figure,  $m = 18$  and  $k = 3$ .

An *empty Bloom filter* is a bit array of  $m$  bits, all set to 0. There must also be  $k$  different hash functions defined, each of which maps or hashes some set element to one of the  $m$  array positions with a uniform random distribution.

To *add* an element, feed it to each of the  $k$  hash functions to get  $k$  array positions. Set the bits at all these positions to 1.

To *query* for an element (test whether it is in the set), feed it to each of the  $k$  hash functions to get  $k$  array positions. If any of the bits at these positions are 0, the element is definitely not in the set – if it were, then all the bits would have been set to 1 when it was inserted. If all are 1, then either the element is in the set, or the bits have by chance been set to 1 during the insertion of other elements, resulting in a false positive. In a simple Bloom filter, there is no way to distinguish between the two cases, but more advanced techniques can address this problem.

# Implementation

## Setting Up the Wifi –Direct Network

Wi-Fi peer-to-peer (P2P) allows Android 4.0 (API level 14) or later devices with the appropriate hardware to connect directly to each other via Wi-Fi without an intermediate access point (Android's Wi-Fi P2P framework complies with the Wi-Fi Alliance's Wi-Fi Direct certification program). Using these APIs, you can discover and connect to other devices when each device supports Wi-Fi P2P, then communicate over a speedy connection across distances much longer than a Bluetooth connection. This is useful for applications that share data among users, such as a multiplayer game or a photo sharing application.

The Wi-Fi P2P APIs consist of the following main parts:

- Methods that allow you to discover, request, and connect to peers are defined in the `WifiP2pManager` class.
- Listeners that allow you to be notified of the success or failure of `WifiP2pManager` method calls. When calling `WifiP2pManager` methods, each method can receive a specific listener passed in as a parameter.
- Intents that notify you of specific events detected by the Wi-Fi P2P framework, such as a dropped connection or a newly discovered peer.

You often use these three main components of the APIs together. For example, you can provide a `WifiP2pManager.ActionListener` to a call to `discoverPeers()`, so that you can be notified with the `ActionListener.onSuccess()` and `ActionListener.onFailure()` methods.

`AWIFI_P2P_PEERS_CHANGED_ACTION` intent is also broadcast if the `discoverPeers()` method discovers that the peers list has changed.

### API Overview

---

The `WifiP2pManager` class provides methods to allow you to interact with the Wi-Fi hardware on your device to do things like discover and connect to peers. The following actions are available:

Method	Description
initialize()	Registers the application with the Wi-Fi framework. This must be called before calling any other Wi-Fi P2P method.
connect()	Starts a peer-to-peer connection with a device with the specified configuration.
cancelConnect()	Cancels any ongoing peer-to-peer group negotiation.
requestConnectInfo()	Requests a device's connection information.
createGroup()	Creates a peer-to-peer group with the current device as the group owner.
removeGroup()	Removes the current peer-to-peer group.
requestGroupInfo()	Requests peer-to-peer group information.
discoverPeers()	Initiates peer discovery
requestPeers()	Requests the current list of discovered peers.

**Table 1.** Wi-Fi P2P Methods

WifiP2pManager methods let you pass in a listener, so that the Wi-Fi P2P framework can notify your activity of the status of a call. The available listener interfaces and the corresponding WifiP2pManager method calls that use the listeners are described in the following table:

Listener interface	Associated actions
WifiP2pManager.ActionListener	connect(), cancelConnect(), createGroup(), removeGroup(), and discoverPeers()
WifiP2pManager.ChannelListener	initialize()
WifiP2pManager.ConnectionInfoListener	requestConnectInfo()
WifiP2pManager.GroupInfoListener	requestGroupInfo()
WifiP2pManager.PeerListListener	requestPeers()

**Table 2.** Wi-Fi P2P Listeners

The Wi-Fi P2P APIs define intents that are broadcast when certain Wi-Fi P2P events happen, such as when a new peer is discovered or when a device's Wi-Fi state changes. You can register to receive these intents in your application by creating a broadcast receiver that handles these intents:

Intent	Description
WIFI_P2P_CONNECTION_CHANGED_ACTION	Broadcast when the state of the device's Wi-Fi connection changes.
WIFI_P2P_PEERS_CHANGED_ACTION	Broadcast when you call discoverPeers(). You usually want to

	call requestPeers() to get an updated list of peers if you handle this intent in your application.
WIFI_P2P_STATE_CHANGED_ACTION	Broadcast when Wi-Fi P2P is enabled or disabled on the device.
WIFI_P2P_THIS_DEVICE_CHANGED_ACTION	Broadcast when a device's details have changed, such as the device's name.

**Table 3. Wi-Fi P2P Intents**

### Creating a Broadcast Receiver for Wi-Fi P2P Intents

---

A broadcast receiver allows you to receive intents broadcast by the Android system, so that your application can respond to events that you are interested in. The basic steps for creating a broadcast receiver to handle Wi-Fi P2P intents are as follows:

1. Create a class that extends the BroadcastReceiver class. For the class' constructor, you most likely want to have parameters for the WifiP2pManager, WifiP2pManager.Channel, and the activity that this broadcast receiver will be registered in. This allows the broadcast receiver to send updates to the activity as well as have access to the Wi-Fi hardware and a communication channel if needed.
2. In the broadcast receiver, check for the intents that you are interested in onReceive(). Carry out any necessary actions depending on the intent that is received. For example, if the broadcast



receiver receives a `WIFI_P2P_PEERS_CHANGED_ACTION` intent, you can call the `requestPeers()` method to get a list of the currently discovered peers.

The following code shows you how to create a typical broadcast receiver. The broadcast receiver takes a `WifiP2pManager` object and an activity as arguments and uses these two classes to appropriately carry out the needed actions when the broadcast receiver receives an intent:

```
/**
 * A BroadcastReceiver that notifies of important Wi-Fi p2p events.
 */
public class WifiDirectBroadcastReceiver extends BroadcastReceiver {

    private WifiP2pManager mManager;
    private Channel mChannel;
    private MyWifiActivity mActivity;

    public WifiDirectBroadcastReceiver(WifiP2pManager manager, Channel channel,
        MyWifiActivity activity) {
        super();
        this.mManager = manager;
        this.mChannel = channel;
        this.mActivity = activity;
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
            // Check to see if Wi-Fi is enabled and notify appropriate activity
        } else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {
```

```

        // Call WifiP2pManager.requestPeers() to get a list of current peers
    } else if
(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {
        // Respond to new connection or disconnections
    } else if
(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)) {
        // Respond to this device's wifi state changing
    }
}
}
}

```

## Creating a Wi-Fi P2P Application

---

Creating a Wi-Fi P2P application involves creating and registering a broadcast receiver for your application, discovering peers, connecting to a peer, and transferring data to a peer. The following sections describe how to do this.

### Initial setup

Before using the Wi-Fi P2P APIs, you must ensure that your application can access the hardware and that the device supports the Wi-Fi P2P protocol. If Wi-Fi P2P is supported, you can obtain an instance of `WifiP2pManager`, create and register your broadcast receiver, and begin using the Wi-Fi P2P APIs.

1. Request permission to use the Wi-Fi hardware on the device and also declare your application to have the correct minimum SDK version in the Android manifest:

```

<uses-sdk android:minSdkVersion="14" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />

```

```
<uses-permission android:name="android.permission.INTERNET" />
```

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

2. Check to see if Wi-Fi P2P is on and supported. A good place to check this is in your broadcast receiver when it receives the `WIFI_P2P_STATE_CHANGED_ACTION` intent. Notify your activity of the Wi-Fi P2P state and react accordingly:

```
@Override
```

```
public void onReceive(Context context, Intent intent) {
```

```
...
```

```
String action = intent.getAction();
```

```
if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
```

```
    int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
```

```
    if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
```

```
        // Wifi P2P is enabled
```

```
    } else {
```

```
        // Wi-Fi P2P is not enabled
```

```
    }
```

```
}
```

```
...
```

```
}
```

3. In your activity's `onCreate()` method, obtain an instance of `WifiP2pManager` and register your application with the Wi-Fi P2P framework by calling `initialize()`. This method returns a `WifiP2pManager.Channel`, which is used to connect your application to the Wi-Fi P2P framework. You should also create an instance of your broadcast receiver with the `WifiP2pManager` and `WifiP2pManager.Channel` objects along with a reference to your activity. This allows your broadcast receiver to notify your activity of interesting events and update it accordingly. It also lets you manipulate the device's Wi-Fi state if necessary:

```
WifiP2pManager mManager;
```

```
Channel mChannel;
```

```

BroadcastReceiver mReceiver;
...
@Override
protected void onCreate(Bundle savedInstanceState){
    ...
    mManager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
    mChannel = mManager.initialize(this, getMainLooper(), null);
    mReceiver = new WifiDirectBroadcastReceiver(mManager, mChannel, this);
    ...
}

```

4. Create an intent filter and add the same intents that your broadcast receiver checks for:

```

IntentFilter mIntentFilter;
...
@Override
protected void onCreate(Bundle savedInstanceState){
    ...
    mIntentFilter = new IntentFilter();
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
    mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);
    ...
}

```

5. Register the broadcast receiver in the onResume() method of your activity and unregister it in the onPause() method of your activity:

```

/* register the broadcast receiver with the intent values to be matched */
@Override
protected void onResume() {
    super.onResume();
    registerReceiver(mReceiver, mIntentFilter);
}
/* unregister the broadcast receiver */
@Override
protected void onPause() {
    super.onPause();
    unregisterReceiver(mReceiver);
}

```

When you have obtained a `WifiP2pManager.Channel` and set up a broadcast receiver, your application can make Wi-Fi P2P method calls and receive Wi-Fi P2P intents.

You can now implement your application and use the Wi-Fi P2P features by calling the methods in `WifiP2pManager`. The next sections describe how to do common actions such as discovering and connecting to peers.

### Discovering peers

To discover peers that are available to connect to, call `discoverPeers()` to detect available peers that are in range. The call to this function is asynchronous and a success or failure is communicated to your application with `onSuccess()` and `onFailure()` if you created a `WifiP2pManager.ActionListener`. The `onSuccess()` method only notifies you that the discovery process succeeded and does not provide any information about the actual peers that it discovered, if any:

```

mManager.discoverPeers(channel, new WifiP2pManager.ActionListener() {
    @Override
    public void onSuccess() {
        ...
    }
}

```

```

    }

    @Override
    public void onFailure(int reasonCode) {
        ...
    }
});

```

If the discovery process succeeds and detects peers, the system broadcasts the `WIFI_P2P_PEERS_CHANGED_ACTION` intent, which you can listen for in a broadcast receiver to obtain a list of peers. When your application receives the intent, you can call `requestPeers()` to request a list of peers. The following code shows how to request a list of peers:

```

    if (mManager != null) {
        mManager.requestPeers(mChannel, myPeerListListener);
    }
}

```

The `requestPeers()` method is also asynchronous and can notify your activity when a list of peers is available with `onPeersAvailable()`, which is defined in the `WifiP2pManager.PeerListListener` interface. The `onPeersAvailable()` method provides you with an `WifiP2pDeviceList`, which you can iterate through to find the peer that you want to connect to.

### Connecting to peers

When you have figured out the device that you want to connect to after obtaining a list of possible peers, call the `connect()` method to connect to the device. This method call requires a `WifiP2pConfig` object that contains the information of the device to connect to. You can be notified of a connection success or failure through the `WifiP2pManager.ActionListener`. The following code shows you how to create a connection to a desired device:

```

//obtain a peer from the WifiP2pDeviceList
WifiP2pDevice device;
WifiP2pConfig config = new WifiP2pConfig();

```

```

config.deviceAddress = device.deviceAddress;
mManager.connect(mChannel, config, new ActionListener() {

    @Override
    public void onSuccess() {
        //success logic
    }

    @Override
    public void onFailure(int reason) {
        //failure logic
    }
});

```

## Transferring data

Once a connection is established, you can transfer data between the devices with sockets. The basic steps of transferring data are as follows:

1. Create a `ServerSocket`. This socket waits for a connection from a client on a specified port and blocks until it happens, so do this in a background thread.
2. Create a client `Socket`. The client uses the IP address and port of the server socket to connect to the server device.
3. Send data from the client to the server. When the client socket successfully connects to the server socket, you can send data from the client to the server with byte streams.
4. The server socket waits for a client connection (with the `accept()` method). This call blocks until a client connects, so call this in another thread. When a connection happens, the server device can receive the data from the client. Carry out any actions with this data, such as saving it to a file or presenting it to the user.

The following example, modified from the Wi-Fi P2P Demo sample, shows you how to create this client-server socket communication and transfer JPEG images from a client to a server with a service. For a complete working example, compile and run the Wi-Fi P2P Demo sample.

```
public static class FileServerAsyncTask extends AsyncTask {

    private Context context;
    private TextView statusText;

    public FileServerAsyncTask(Context context, View statusText) {
        this.context = context;
        this.statusText = (TextView) statusText;
    }

    @Override
    protected String doInBackground(Void... params) {
        try {

            /**
             * Create a server socket and wait for client connections. This
             * call blocks until a connection is accepted from a client
             */
            ServerSocket serverSocket = new ServerSocket(8888);
            Socket client = serverSocket.accept();

            /**
             * If this code is reached, a client has connected and transferred data
             * Save the input stream from the client as a JPEG file
             */
            final File f = new File(Environment.getExternalStorageDirectory() + "/"
```



```

        + context.getPackageName() + "/wifip2pshared-" + System.currentTimeMillis()
        + ".jpg");

File dirs = new File(f.getParent());
if (!dirs.exists())
    dirs.mkdirs();
f.createNewFile();
InputStream inputstream = client.getInputStream();
copyFile(inputstream, new FileOutputStream(f));
serverSocket.close();
return f.getAbsolutePath();
} catch (IOException e) {
    Log.e(WiFiDirectActivity.TAG, e.getMessage());
    return null;
}
}

/**
 * Start activity that can handle the JPEG image
 */
@Override
protected void onPostExecute(String result) {
    if (result != null) {
        textStatus.setText("File copied - " + result);
        Intent intent = new Intent();
        intent.setAction(android.content.Intent.ACTION_VIEW);
        intent.setDataAndType(Uri.parse("file://" + result), "image/*");
        context.startActivity(intent);
    }
}
}

```

```
}
```

On the client, connect to the server socket with a client socket and transfer data. This example transfers a JPEG file on the client device's file system.

```
Context context = this.getApplicationContext();
String host;
int port;
int len;
Socket socket = new Socket();
byte buf[] = new byte[1024];
...
try {
    /**
     * Create a client socket with the host,
     * port, and timeout information.
     */
    socket.bind(null);
    socket.connect((new InetSocketAddress(host, port)), 500);

    /**
     * Create a byte stream from a JPEG file and pipe it to the output stream
     * of the socket. This data will be retrieved by the server device.
     */
    OutputStream outputStream = socket.getOutputStream();
    ContentResolver cr = context.getContentResolver();
    InputStream inputStream = null;
    inputStream = cr.openInputStream(Uri.parse("path/to/picture.jpg"));
    while ((len = inputStream.read(buf)) != -1) {
```

```

        outputStream.write(buf, 0, len);
    }
    outputStream.close();
    inputStream.close();
} catch (FileNotFoundException e) {
    //catch logic
} catch (IOException e) {
    //catch logic
}

/**
 * Clean up any open sockets when done
 * transferring or if an exception occurred.
 */
finally {
    if (socket != null) {
        if (socket.isConnected()) {
            try {
                socket.close();
            } catch (IOException e) {
                //catch logic
            }
        }
    }
}
}

```

e WIFI\_P2P\_PEERS\_CHANGED\_ACTION intent, you can request a list of the discovered peers with requestPeers(). The following code shows how to set this up:

```

PeerListListener myPeerListListener;
...
if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {

```

```

// request available peers from the wifi p2p manager. This is an
// asynchronous call and the calling activity is notified with a
// callback on PeerListListener.onPeersAvailable()

```

## Architecture of the App

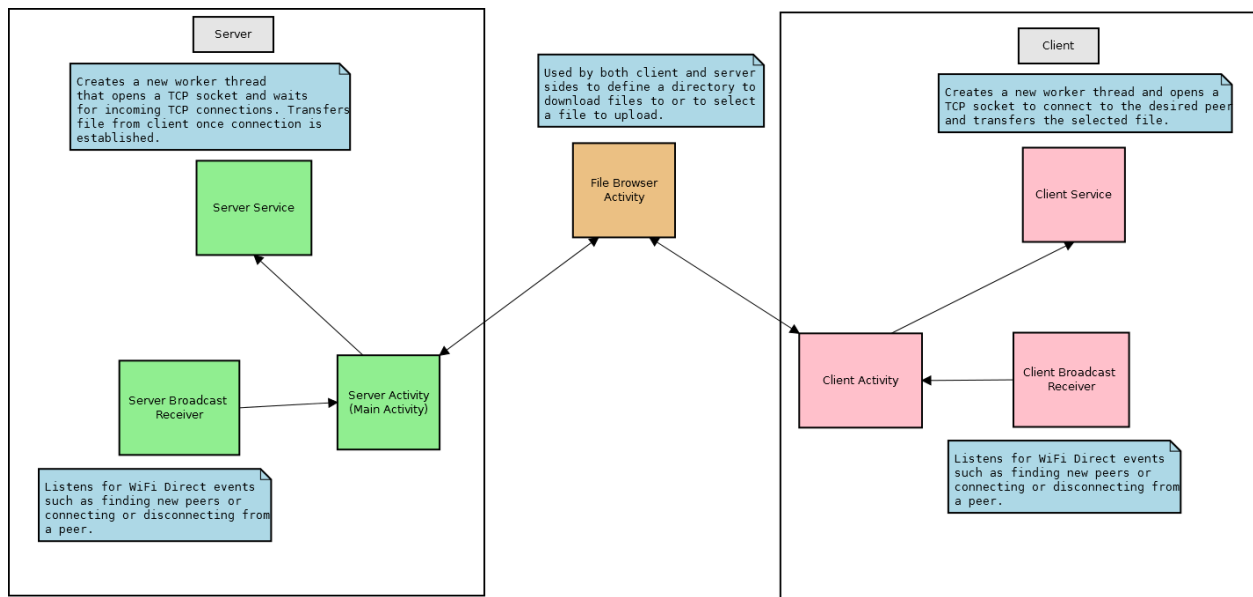


Figure:2

## Bloom Filter Code

```

import java.util.*;
import java.security.*;
import java.math.*;
import java.nio.*;

```

```

/* Class BloomFilter */
class BloomFilter
{
    private byte[] set;
    private int keySize, setSize, size;
    private MessageDigest md;

    /* Constructor */
    public BloomFilter(int capacity, int k)
    {
        setSize = capacity;
        set = new byte[setSize];
        keySize = k;
        size = 0;
        try
        {
            md = MessageDigest.getInstance("MD5");
        }
        catch (NoSuchAlgorithmException e)
        {
            throw new IllegalArgumentException("Error : MD5 Hash not found");
        }
    }

    /* Function to clear bloom set */
    public void makeEmpty()

```

```

{
    set = new byte[setSize];
    size = 0;
    try
    {
        md = MessageDigest.getInstance("MD5");
    }
    catch (NoSuchAlgorithmException e)
    {
        throw new IllegalArgumentException("Error : MD5 Hash not found");
    }
}

/* Function to check is empty */
public boolean isEmpty()
{
    return size == 0;
}

/* Function to get size of objects added */
public int getSize()
{
    return size;
}

/* Function to get hash - MD5 */
private int getHash(int i)
{

```

```

md.reset();

byte[] bytes = ByteBuffer.allocate(4).putInt(i).array();

md.update(bytes, 0, bytes.length);

return Math.abs(new BigInteger(1, md.digest()).intValue()) % (set.length - 1);
}

/* Function to add an object */

public void add(Object obj)
{
    int[] tmpset = getSetArray(obj);

    for (int i : tmpset)
        set[i] = 1;

    size++;
}

/* Function to check is an object is present */

public boolean contains(Object obj)
{
    int[] tmpset = getSetArray(obj);

    for (int i : tmpset)
        if (set[i] != 1)
            return false;

    return true;
}

/* Function to get set array for an object */

private int[] getSetArray(Object obj)
{

```

```

int[] tmpset = new int[keySize];
tmpset[0] = getHash(obj.hashCode());
for (int i = 1; i < keySize; i++)
    tmpset[i] = (getHash(tmpset[i - 1]));
return tmpset;
}
}

/* Class BloomFilterTest */
public class BloomFilterTest
{
public static void main(String[] args)
{
Scanner scan = new Scanner(System.in);
System.out.println("Bloom Filter Test\n");

System.out.println("Enter set capacity and key size");
BloomFilter bf = new BloomFilter(scan.nextInt() , scan.nextInt());

char ch;

/* Perform bloom filter operations */
do
{
System.out.println("\nBloomFilter Operations\n");
System.out.println("1. insert ");

```



```
System.out.println("2. contains");
System.out.println("3. check empty");
System.out.println("4. clear");
System.out.println("5. size");

int choice = scan.nextInt();

switch (choice)
{
case 1 :
    System.out.println("Enter integer element to insert");
    bf.add( new Integer(scan.nextInt()) );
    break;
case 2 :
    System.out.println("Enter integer element to search");
    System.out.println("Search result : "+ bf.contains( new Integer(scan.nextInt()) ));
    break;
case 3 :
    System.out.println("Empty status = "+ bf.isEmpty());
    break;
case 4 :
    System.out.println("\nBloom set Cleared");
    bf.makeEmpty();
    break;
case 5 :
    System.out.println("\nSize = "+ bf.getSize() );
```

```
        break;
    default :
        System.out.println("Wrong Entry \n ");
        break;
    }

    System.out.println("\nDo you want to continue (Type y or n) \n");
    ch = scan.next().charAt(0);
    } while (ch == 'Y' || ch == 'y');
}
}
```

# Results

## WiFi Direct App

### Screenshots

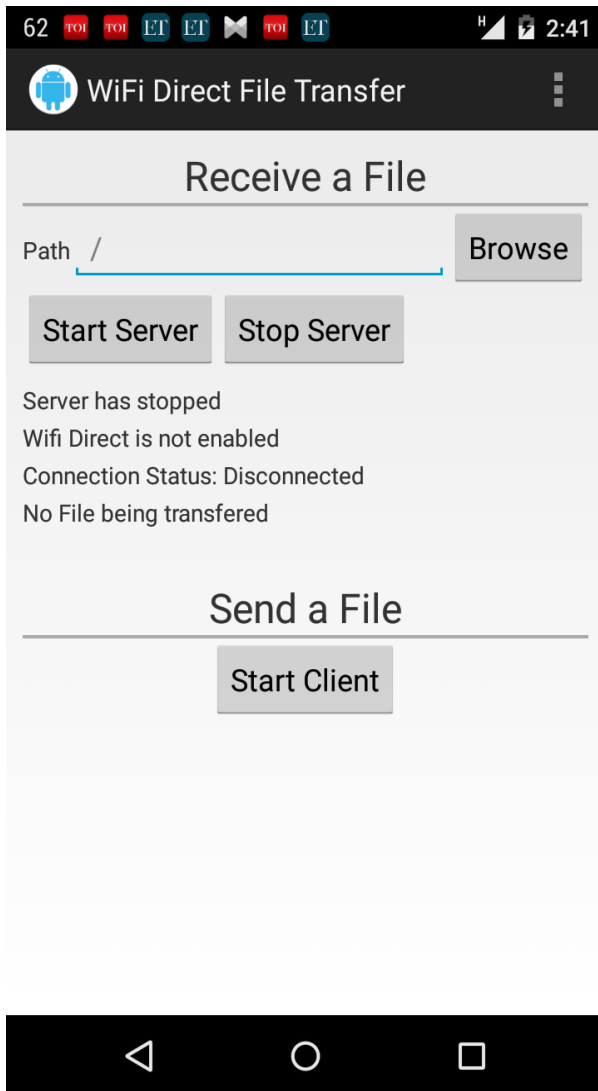


Figure 3 (Default Screen )

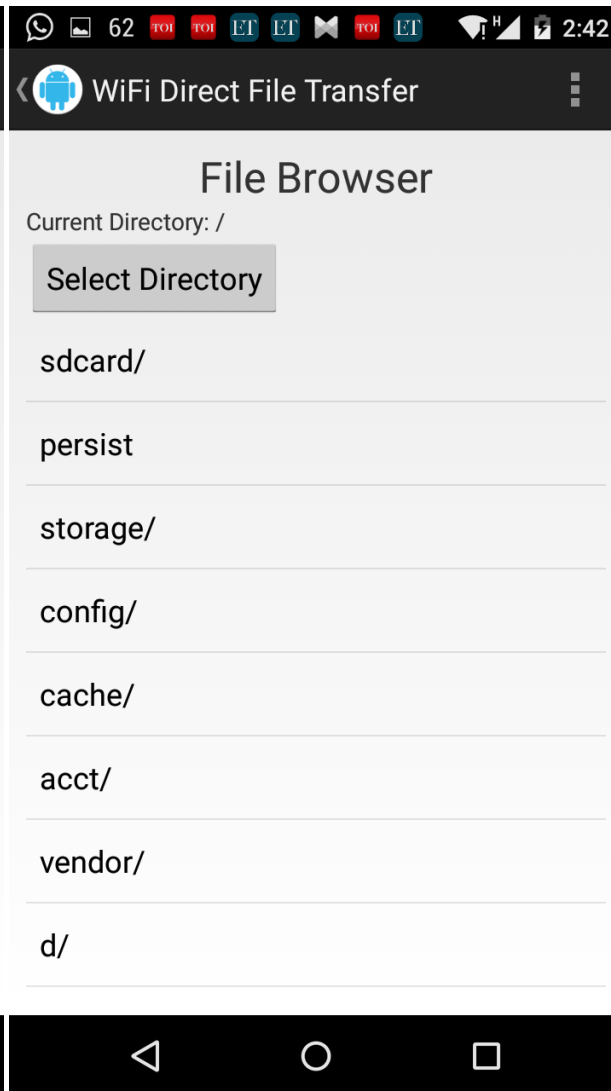


Figure 4 (Selecting Dir. for server device)

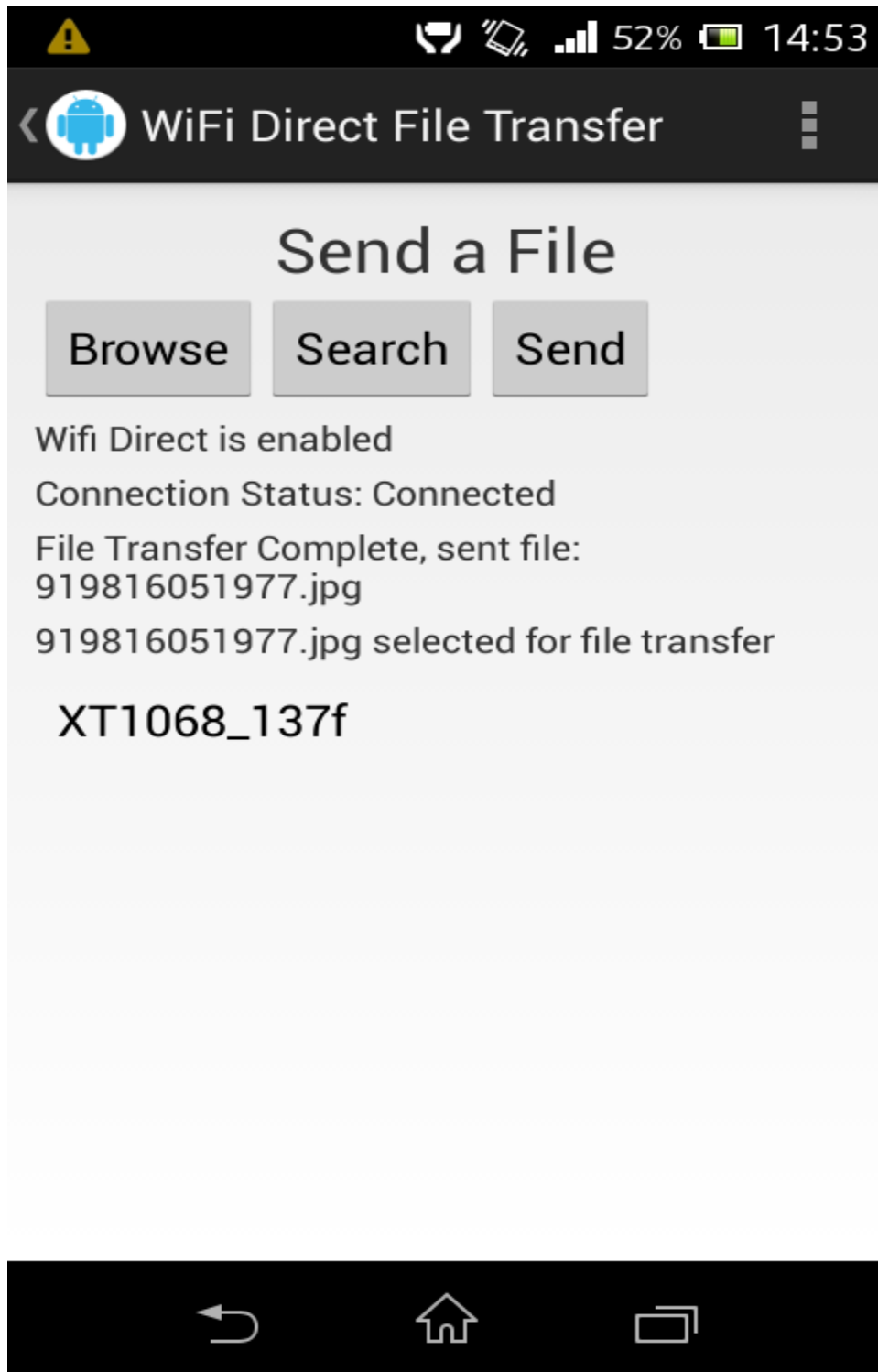
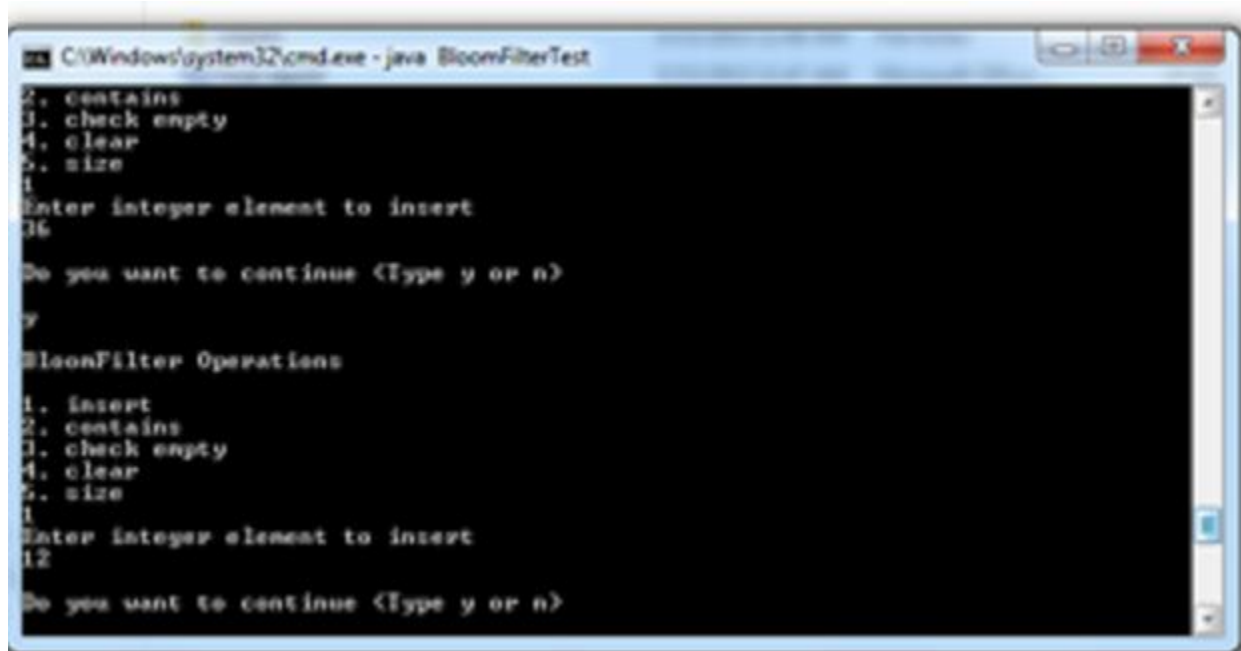


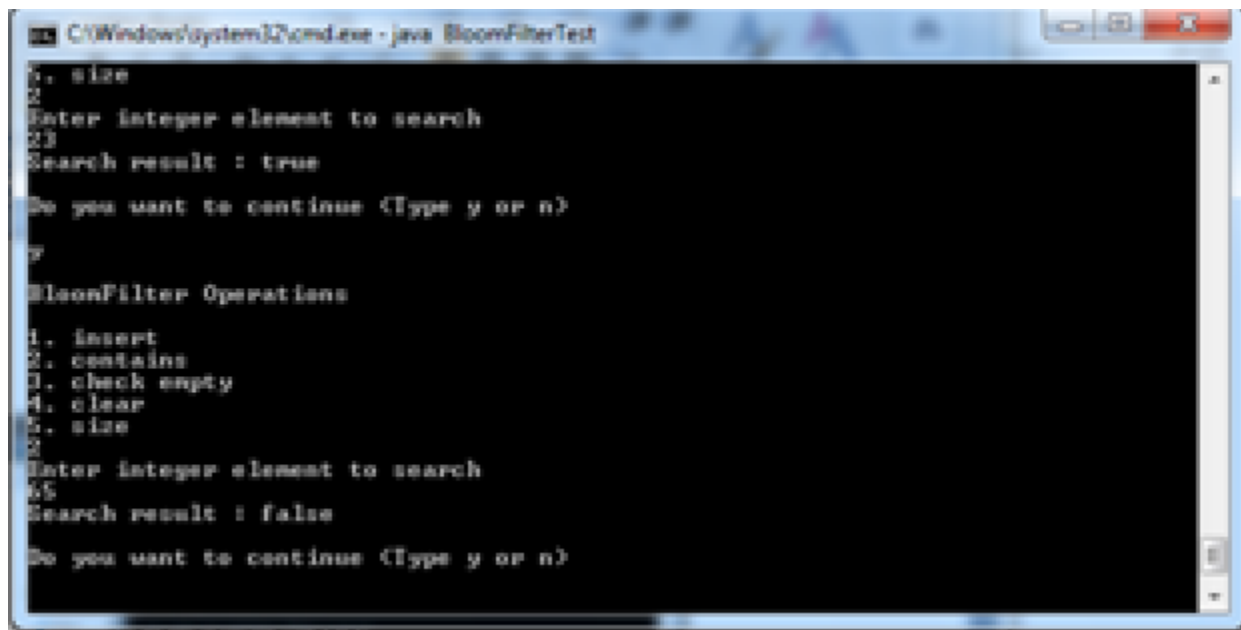
Figure 5 (File Sent)

## Bloom Filter Working Screenshots



```
C:\Windows\system32\cmd.exe - java BloomFilterTest
2. contains
3. check empty
4. clear
5. size
1
Enter integer element to insert
36
Do you want to continue (Type y or n)
y
BloomFilter Operations
1. insert
2. contains
3. check empty
4. clear
5. size
1
Enter integer element to insert
12
Do you want to continue (Type y or n)
```

Figure: 6 ( Insertion of elements in the bloom filter)



```
C:\Windows\system32\cmd.exe - java BloomFilterTest
5. size
2
Enter integer element to search
23
Search result : true
Do you want to continue (Type y or n)
y
BloomFilter Operations
1. insert
2. contains
3. check empty
4. clear
5. size
2
Enter integer element to search
65
Search result : false
Do you want to continue (Type y or n)
```

Figure: 7 (Checking whether an element belongs to the set or not)

```
C:\Windows\system32\cmd.exe - java BloomFilterTest
1. check empty
4. clear
5. size
5
Size = 5
Do you want to continue (Type y or n)
y
BloomFilter Operations
1. insert
2. contains
3. check empty
4. clear
5. size
4
Bloom set Cleared
Do you want to continue (Type y or n)
```

Figure: 8 (Checking the size of and clearing the bloom filter)

# Future Work

In future, I will be incorporating the bloom filter into my app so as to increase the efficiency and speed of the transaction between the devices.

I will also try to send a file to more than one receivers concurrently.

I will also work on the user interface to make it more convenient and user friendly.

# References

1. Ilya Mironov, "Hash functions: Theory, attacks, and applications", Microsoft Research, Silicon Valley Campus, November 14, 2005
2. B. J. MCKENZIE, R. HARRIES AND T. BELL, "Selecting a Hashing Algorithm", Department of Computer Science, University of Canterbury, 1989
3. B. Bloom. "Space/Time Tradeoffs in Hash Coding with Allowable Errors.", 1970
4. IEEE 802.11-2007 Standard, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 2007.
5. Jin-Shyan Lee; Yu-Wei Su; Chung-Chou Shen, A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi, Industrial Electronics Society, 2007. IECON 2007
6. Wi-Fi Alliance, Wi-Fi Protected Setup Specification v1.0h, Dec. 2006.
7. Wi-Fi Alliance, P2P Technical Group, Wi-Fi Peer-to-Peer (P2P) Technical Specification v1.0, December 2009