相比于通过 Wiif Direct 进行文件传输，通过 Wifi 热点进行设备配对更加方便，逻辑也更为直接，传输一个1G左右的压缩包用了5分钟左右的时间，平均传输速率有 3.5 M/S 左右。此外，相对于上个版本，新版本除了提供传输进度外，还提供了传输速率、预估完成时间、文件传输前后的MD5码等数据

实现的效果如下所示：

开启Ap热点接收文件

[](https://www.writebug.com/office/imagePreview?imgurl=https:/www.writebug.com/myres/static/uploads/2022/7/21/0fa60356e2f6c39743d14044a151a96b.writebug)

连接Wiif热点发送文件

[](https://www.writebug.com/office/imagePreview?imgurl=https:/www.writebug.com/myres/static/uploads/2022/7/21/737bf8ebaeb2d5f44a8836ea0ee9c5dc.writebug)

文件传输完成后校验文件完整性

[](https://www.writebug.com/office/imagePreview?imgurl=https:/www.writebug.com/myres/static/uploads/2022/7/21/22dafdece451add81b8c48a3ef8f6a9f.writebug)

开发步骤分为以下几点：

1. 在 AndroidManifest 中声明相关权限（网络和文件读写权限）
2. 文件接收端开启Ap热点，作为服务器端建立Socket，在指定端口等待客户端的连接
3. 文件发送端连接到Wifi热点，作为客户端主动连接到服务器端
4. 文件发送端将待发送的文件信息模型（包括文件路径，文件大小和文件MD5码等信息）通过Socket发送给文件接收端
5. 文件发送端发起实际的文件传输请求，接收端和发送端根据已接收到或已发送的的文件字节数，计算文件传输进度、文件传输速率和预估完成时间等数据
6. 文件传输结束后，对比文件信息模型携带来的MD5码值与本地文件重新计算生成的MD5码是否相等，以此校验文件完整性

### **一、声明权限**

本应用并不会消耗移动数据，但由于要使用到 Wifi 以及 Java Socket，所以需要申请网络相关的权限。此外，由于是要实现文件互传，所以也需要申请SD卡读写权限。

<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" />

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />

### **二、文件接收端**

文件接收端作为服务器存在，需要主动开启Ap热点供文件发送端连接，由于通过反射来开启热点的方法在高版本系统上无法实现，所以需要用户主动去开启热点，并设置固定的热点名称

此处需要先定义一个文件信息模型 FileTransfer ，FileTransfer 包含三个字段，MD5码值用于校验文件的完整性，fileLength 是为了用于计算文件的传输进度和传输速率

public class FileTransfer implements Serializable {

//文件路径 private String filePath;

//文件大小 private long fileLength;

//MD5码 private String md5;

···

}

Ap热点开启成功后，就可以启动一个服务在后台等待文件发送端来主动连接了，这里使用 IntentService 在后台监听客户端的Socket连接请求，并通过输入输出流来传输文件。此处的代码比较简单，就只是在指定端口一直堵塞监听客户端的连接请求，获取待传输的文件信息模型 FileTransfer ，之后就进行实际的数据传输

文件传输速率是每一秒计算一次，根据这段时间内接收的字节数与消耗的时间做除法，从而得到传输速率，再通过将剩余的未传输字节数与传输速率做除法，从而得到预估的剩余传输时间

@Override

protected void onHandleIntent(Intent intent) {

if (intent != null && ACTION\_START\_RECEIVE.equals(intent.getAction())) {

clean();

File file = null;

Exception exception = null;

try {

serverSocket = new ServerSocket();

serverSocket.setReuseAddress(true);

serverSocket.bind(new InetSocketAddress(Constants.PORT));

Socket client = serverSocket.accept();

Log.e(TAG, "客户端IP地址 : " + client.getInetAddress().getHostAddress());

inputStream = client.getInputStream();

objectInputStream = new ObjectInputStream(inputStream);

fileTransfer = (FileTransfer) objectInputStream.readObject();

Log.e(TAG, "待接收的文件: " + fileTransfer);

if (fileTransfer == null) {

exception = new Exception("从文件发送端发来的文件模型为null");

return;

} else if (TextUtils.isEmpty(fileTransfer.getMd5())) {

exception = new Exception("从文件发送端发来的文件模型不包含MD5码");

return;

}

String name = new File(fileTransfer.getFilePath()).getName();

//将文件存储至指定位置 file = new File(Environment.getExternalStorageDirectory() + "/" + name);

fileOutputStream = new FileOutputStream(file);

startCallback();

byte[] buf = new byte[512];

int len;

while ((len = inputStream.read(buf)) != -1) {

fileOutputStream.write(buf, 0, len);

total += len;

}

Log.e(TAG, "文件接收成功");

stopCallback();

if (progressChangListener != null) {

//因为上面在计算文件传输进度时因为小数点问题可能不会显示到100%，所以此处手动将之设为100% progressChangListener.onProgressChanged(fileTransfer, 0, 100, 0, 0, 0, 0);

//开始计算传输到本地的文件的MD5码 progressChangListener.onStartComputeMD5();

}

} catch (Exception e) {

Log.e(TAG, "文件接收 Exception: " + e.getMessage());

exception = e;

} finally {

FileTransfer transfer = new FileTransfer();

if (file != null && file.exists()) {

transfer.setFilePath(file.getPath());

transfer.setFileSize(file.length());

transfer.setMd5(Md5Util.getMd5(file));

Log.e(TAG, "计算出的文件的MD5码是：" + transfer.getMd5());

}

if (exception != null) {

if (progressChangListener != null) {

progressChangListener.onTransferFailed(transfer, exception);

}

} else {

if (progressChangListener != null && fileTransfer != null) {

if (fileTransfer.getMd5().equals(transfer.getMd5())) {

progressChangListener.onTransferSucceed(transfer);

} else {

//如果本地计算出的MD5码和文件发送端传来的值不一致，则认为传输失败 progressChangListener.onTransferFailed(transfer, new Exception("MD5码不一致"));

}

}

}

clean();

//再次启动服务，等待客户端下次连接 startActionTransfer(this);

}

}

}

因为客户端可能会多次发起连接请求，所以当此处文件传输完成后（不管成功或失败），都需要重新 startService ，让服务再次堵塞等待客户端的连接请求

为了让界面能够实时获取到文件的传输状态，所以此处除了需要启动Service外，界面还需要绑定Service，所以需要用到一个更新文件传输状态的接口

public interface OnProgressChangListener {

/

\* 当传输进度发生变化时回调

\*

\* @param fileTransfer 文件发送方传来的文件模型

\* @param progress 文件传输进度

\* @param speed 文件传输速率

\* @param remainingTime 预估的剩余完成时间

\*/

void onProgressChanged(FileTransfer fileTransfer, int progress, double speed, long remainingTime);

//当传输结束时 void onTransferFinished(FileTransfer fileTransfer);

}

在界面层刷新UI

private FileReceiverService.OnReceiveProgressChangListener progressChangListener = new FileReceiverService.OnReceiveProgressChangListener() {

private FileTransfer originFileTransfer;

@Override

public void onProgressChanged(final FileTransfer fileTransfer, final long totalTime, final int progress, final double instantSpeed, final long instantRemainingTime, final double averageSpeed, final long averageRemainingTime) {

this.originFileTransfer = fileTransfer;

runOnUiThread(new Runnable() {

@Override

public void run() {

if (isCreated()) {

progressDialog.setTitle("正在接收的文件： " + originFileTransfer.getFileName());

if (progress != 100) {

progressDialog.setMessage("原始文件的MD5码是：" + originFileTransfer.getMd5()

+ "\n\n" + "总的传输时间：" + totalTime + " 秒"

+ "\n\n" + "瞬时-传输速率：" + (int) instantSpeed + " Kb/s"

+ "\n" + "瞬时-预估的剩余完成时间：" + instantRemainingTime + " 秒"

+ "\n\n" + "平均-传输速率：" + (int) averageSpeed + " Kb/s"

+ "\n" + "平均-预估的剩余完成时间：" + averageRemainingTime + " 秒"

);

}

progressDialog.setProgress(progress);

progressDialog.setCancelable(true);

progressDialog.show();

}

}

});

}

@Override

public void onStartComputeMD5() {

runOnUiThread(new Runnable() {

@Override

public void run() {

if (isCreated()) {

progressDialog.setTitle("传输结束，正在计算本地文件的MD5码以校验文件完整性");

progressDialog.setMessage("原始文件的MD5码是：" + originFileTransfer.getMd5());

progressDialog.setCancelable(false);

progressDialog.show();

}

}

});

}

@Override

public void onTransferSucceed(final FileTransfer fileTransfer) {

runOnUiThread(new Runnable() {

@Override

public void run() {

if (isCreated()) {

progressDialog.setTitle("传输成功");

progressDialog.setMessage("原始文件的MD5码是：" + originFileTransfer.getMd5()

+ "\n" + "本地文件的MD5码是：" + fileTransfer.getMd5()

+ "\n" + "文件位置：" + fileTransfer.getFilePath());

progressDialog.setCancelable(true);

progressDialog.show();

Glide.with(FileReceiverActivity.this).load(fileTransfer.getFilePath()).into(iv\_image);

}

}

});

}

@Override

public void onTransferFailed(final FileTransfer fileTransfer, final Exception e) {

runOnUiThread(new Runnable() {

@Override

public void run() {

if (isCreated()) {

progressDialog.setTitle("传输失败");

progressDialog.setMessage("原始文件的MD5码是：" + originFileTransfer.getMd5()

+ "\n" + "本地文件的MD5码是：" + fileTransfer.getMd5()

+ "\n" + "文件位置：" + fileTransfer.getFilePath()

+ "\n" + "异常信息：" + e.getMessage());

progressDialog.setCancelable(true);

progressDialog.show();

}

}

});

}

};

### **三、文件发送端**

文件发送端作为客户端存在，需要主动连接文件接收端开启的Wifi热点

/

\* 连接指定Wifi

\*

\* @param context 上下文

\* @param ssid SSID

\* @param password 密码

\* @return 是否连接成功

\*/

public static boolean connectWifi(Context context, String ssid, String password) {

String connectedSsid = getConnectedSSID(context);

if (!TextUtils.isEmpty(connectedSsid) && connectedSsid.equals(ssid)) {

return true;

}

openWifi(context);

WifiConfiguration wifiConfiguration = isWifiExist(context, ssid);

if (wifiConfiguration == null) {

wifiConfiguration = createWifiConfiguration(ssid, password);

}

WifiManager wifiManager = (WifiManager) context.getApplicationContext().getSystemService(Context.WIFI\_SERVICE);

if (wifiManager == null) {

return false;

}

int networkId = wifiManager.addNetwork(wifiConfiguration);

return wifiManager.enableNetwork(networkId, true);

}

/

\* 开启Wifi

\*

\* @param context 上下文

\* @return 是否成功

\*/

public static boolean openWifi(Context context) {

WifiManager wifiManager = (WifiManager) context.getApplicationContext().getSystemService(Context.WIFI\_SERVICE);

return wifiManager != null && (wifiManager.isWifiEnabled() || wifiManager.setWifiEnabled(true));

}

/

\* 获取当前连接的Wifi的SSID

\*

\* @param context 上下文

\* @return SSID

\*/

public static String getConnectedSSID(Context context) {

WifiManager wifiManager = (WifiManager) context.getApplicationContext().getSystemService(Context.WIFI\_SERVICE);

WifiInfo wifiInfo = wifiManager == null ? null : wifiManager.getConnectionInfo();

return wifiInfo != null ? wifiInfo.getSSID().replaceAll("\"", "") : "";

}

连接到指定Wifi后，在选择了要发送的文件后，就启动一个后台线程去主动请求连接服务器端，然后就是进行实际的文件传输操作了

demo 提供的例子是只用来传输图片，但理论上是可以传输任意格式的文件的

private void navToChose() {

Matisse.from(this)

.choose(MimeType.ofImage())

.countable(true)

.showSingleMediaType(true)

.maxSelectable(1)

.capture(false)

.captureStrategy(new CaptureStrategy(true, BuildConfig.APPLICATION\_ID + ".fileprovider"))

.restrictOrientation(ActivityInfo.SCREEN\_ORIENTATION\_UNSPECIFIED)

.thumbnailScale(0.70f)

.imageEngine(new Glide4Engine())

.forResult(CODE\_CHOOSE\_FILE);

}

获取选取的文件的实际路径，并启动 FileSenderService 去进行文件传输操作

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

super.onActivityResult(requestCode, resultCode, data);

if (requestCode == CODE\_CHOOSE\_FILE && resultCode == RESULT\_OK) {

List<String> strings = Matisse.obtainPathResult(data);

if (strings != null && !strings.isEmpty()) {

String path = strings.get(0);

File file = new File(path);

if (file.exists()) {

FileTransfer fileTransfer = new FileTransfer(file);

Log.e(TAG, "待发送的文件：" + fileTransfer);

FileSenderService.startActionTransfer(this, fileTransfer, WifiLManager.getHotspotIpAddress(this));

}

}

}

}

将服务器端的IP地址作为参数传给 FileSenderService，在正式发送文件前，先发送包含文件信息的 FileTransfer ，并在发送文件的过程中实时更新文件传输状态

@Override

protected void onHandleIntent(Intent intent) {

if (intent != null && ACTION\_START\_SEND.equals(intent.getAction())) {

clean();

fileTransfer = (FileTransfer) intent.getSerializableExtra(EXTRA\_PARAM\_FILE\_TRANSFER);

String ipAddress = intent.getStringExtra(EXTRA\_PARAM\_IP\_ADDRESS);

Log.e(TAG, "IP地址：" + ipAddress);

if (fileTransfer == null || TextUtils.isEmpty(ipAddress)) {

return;

}

if (TextUtils.isEmpty(fileTransfer.getMd5())) {

Logger.e(TAG, "MD5码为空，开始计算文件的MD5码");

if (progressChangListener != null) {

progressChangListener.onStartComputeMD5();

}

fileTransfer.setMd5(Md5Util.getMd5(new File(fileTransfer.getFilePath())));

Log.e(TAG, "计算结束，文件的MD5码值是：" + fileTransfer.getMd5());

} else {

Logger.e(TAG, "MD5码不为空，无需再次计算，MD5码为：" + fileTransfer.getMd5());

}

int index = 0;

while (ipAddress.equals("0.0.0.0") && index < 5) {

Log.e(TAG, "ip: " + ipAddress);

ipAddress = WifiLManager.getHotspotIpAddress(this);

index++;

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

if (ipAddress.equals("0.0.0.0")) {

return;

}

try {

socket = new Socket();

socket.bind(null);

socket.connect((new InetSocketAddress(ipAddress, Constants.PORT)), 20000);

outputStream = socket.getOutputStream();

objectOutputStream = new ObjectOutputStream(outputStream);

objectOutputStream.writeObject(fileTransfer);

inputStream = new FileInputStream(new File(fileTransfer.getFilePath()));

startCallback();

byte[] buf = new byte[512];

int len;

while ((len = inputStream.read(buf)) != -1) {

outputStream.write(buf, 0, len);

total += len;

}

Log.e(TAG, "文件发送成功");

stopCallback();

if (progressChangListener != null) {

//因为上面在计算文件传输进度时因为小数点问题可能不会显示到100%，所以此处手动将之设为100% progressChangListener.onProgressChanged(fileTransfer, 0, 100, 0, 0, 0, 0);

progressChangListener.onTransferSucceed(fileTransfer);

}

} catch (Exception e) {

Log.e(TAG, "文件发送异常 Exception: " + e.getMessage());

if (progressChangListener != null) {

progressChangListener.onTransferFailed(fileTransfer, e);

}

} finally {

clean();

}

}

}

### **四、校验文件完整性**

文件的完整性主要是通过对比文件前后的MD5码值来校验了，文件发送端在发送文件前，先计算得到文件的MD5码，将值赋给 FileTransfer 模型传给文件接收端，文件接收端在传输结束后，再次计算本地的文件MD5码值，通过对比前后值是否相等，就可以判断文件是否传输完整 MD5码值通过如下方法计算得到

/

\* 作者：chenZY

\* 时间：2018/4/3 15:20

\* https://github.com/leavesC \*/public class Md5Util {

public static String getMd5(File file) {

InputStream inputStream = null;

byte[] buffer = new byte[2048];

int numRead;

MessageDigest md5;

try {

inputStream = new FileInputStream(file);

md5 = MessageDigest.getInstance("MD5");

while ((numRead = inputStream.read(buffer)) > 0) {

md5.update(buffer, 0, numRead);

}

inputStream.close();

inputStream = null;

return md5ToString(md5.digest());

} catch (Exception e) {

return null;

} finally {

if (inputStream != null) {

try {

inputStream.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

private static String md5ToString(byte[] md5Bytes) {

StringBuilder hexValue = new StringBuilder();

for (byte b : md5Bytes) {

int val = ((int) b) & 0xff;

if (val < 16) {

hexValue.append("0");

}

hexValue.append(Integer.toHexString(val));

}

return hexValue.toString();

}

}