International Journal of Engineering and Technical Research (IJETR)
ISSN: 2321-0869 (O) 2454-4698 (P), Volume-5, Issue-4, August 2016

Comparative study of data transfers using Wi-Fi modules

Gaurav Khadse, Ninad Adhav

***Abstract*— The need of wireless transfer of data from a microcontroller to an Android device or a desktop PC can be fulfilled by the use of a Bluetooth or Wi-Fi module. A token or a few bytes of data can be transferred using any Bluetooth module. However, a problem arises when the size of data increases to a few megabytes. A low cost Bluetooth module does not buttress the high data rate and switching to an efficient and faster Bluetooth module is an expensive alternative. A cost effective solution to this is using an easily available Wi-Fi module which is comparatively cheaper.**

This paper describes some important steps for setting up a Wi-Fi module, sending large amount of data using the Wi-Fi module, and comparing the speeds of the same module with different microcontrollers.

***Index Terms*— Arduino Mega, Arduino Uno, ESP8266, Teensy 3.2, Data Integrity, Baud Rate.**

I. INTRODUCTION

Data transfer from a microcontroller to an Android device or a desktop PC is easy when a newbie in the field of electronics has to send a token or a byte of data. The available wireless communication modules like Bluetooth and Wi-Fi make it possible to transfer data in minimal time. A newbie can setup a Bluetooth module in a few minutes by writing a simple sketch, and can send data over a Bluetooth module. Setting up a Wi-Fi module is cumbersome as compared to a Bluetooth module and it requires profound knowledge of embedded systems and networking.

When the amount of data to be transferred on an Android or desktop PC from an external SD card increases, the transfer speed becomes a challenge when it comes to a wireless transfer. Bluetooth speed deteriorates when the module is used with a high baud rate. The possible solutions can be using a high performance Bluetooth module or using Wi-Fi module. The high performance Bluetooth module has a large buffer size and a capability to handle high data rates. However, it is expensive as compared to a Wi-Fi module which is sufficient to satisfy the data transfer purpose given the overhead of setting it up.

To send a large amount of data from the sd card may take several minutes. To lessen the transfer time is also one of the challenges. A few tests have been performed and the data transfer is made fast and easy using one of the cheapest components available in the market.

Gaurav Khadse, Electronics and Telecommunication, University of Pune, Pune, India, 9860076725.

Ninad Adhav, Networking and Telecommunications, University of Texas at Dallas, Richardson, TX, 9970113169.

II. LITERATURE SURVEY

The transfer rate would depend upon a lot of factors. Out of all these, the two main factors would be:

1. How fast the sd card is being read and
2. How fast and efficiently the data is being handled by the Bluetooth or Wi-Fi module.

The second factor has been studied thoroughly and numerous experiments were performed on different Bluetooth and Wi-Fi modules to have a practically achievable transfer rate. This was reinforced by 100% data integrity.

The Bluetooth modules that had been taken into consideration were classic Bluetooth modules like HC05, RN41, RN42, and BT33. HC05, RN41, and RN42 don't have enough buffer size to handle large amount of data continuously. Hence the flow control was implemented using the RTS and CTS pins provided by these Bluetooth modules. But, after implementing the flow control, the transfer rate started to deteriorate drastically. These Bluetooth modules took approximately 1 Minute to send 1Mb of data. The BT33 seemed more efficient than the rest of the Bluetooth modules and therefore, the BT33 module, after having a proper implementation of flow control, was able to achieve higher transfer rate than the HC05, RN41, and RN42. But BT33 was not cost-effective. Also, only one instance of the Bluetooth module could be connected at a time with an Android device or with a Desktop PC. This could be overcome using a Wi-Fi module.

Therefore, Wi-Fi module was chosen and research began with finding a suitable Wi-Fi module which is reliable as well as cost-effective. After going through several Wi-Fi modules, the Espressif ESP8266-12e and Huzzah CC3000 were shortlisted.

After a lot of extensive comparison based on the following performance, compatibility, supporting forums, and cost of the modules, we selected the ESP8266-12e as the module for our tests.

ESP8266-2e has a programmable microcontroller but we preferred to test it with an external controller as we had other time-constraining tasks to perform using an external controller.

The following table outlines the basic structure of CC3000 and ESP 8266.

Wi-Fi chip/module	CC3000	ESP8266
Wi-Fi Standards	802.11 b/g	802.11 b/g/n
Packets	TCP and UDP	TCP and UDP
Modes	Client and Server	Client and Server
Concurrent Sockets	4	5
Access Point Modes	No	P2P, Soft-AP
Size	26.22 x 40.45 x 2.95mm	24 x 16mm
Interface	SPI	TTL Serial
Encryption	Up to WPA2-PSK	Up to WPA2-PSK
Sleep Current	-	<10 uA
Transmit Current	350mA	215 mA (typ.)
Receive Current	-	~60 mA
Digital Pins	0	9
Analog Pins	0	1
Other Pins	0	0 (E variant adds more)
Programmable Microcontroller	No	Yes
Cost (US Dollars)	$34.95	$3.37 – 6.95

Table 1. Comparison of Wi-Fi modules

III. Proposed Architecture

The esp8266-12e was tested with Arduino UNO and Arduino Mega in server mode. It was also tested with Teensy 3.2 in server as well as client mode. The mode is said to be a server mode when the esp8266-12e acts as a server and an external mobile or a desktop PC acts as a client. The mode is said to be a client mode when the esp8266-12e acts as a client. In this mode, the python server has to be created on a desktop PC.

General Configurations:

Wi-Fi Module Name: ESP8266-12e

Firmware used:

esp_iot_sdk_1.5.4 (New Firmware)

esp_iot_sdk_0.9.4 (Old Firmware)

Software Used:

Arduino IDE , Teensyduino, Python IDLE.

Hardware used:

Arduino Uno, Arduino Mega, Teensy 3.2

Connection:

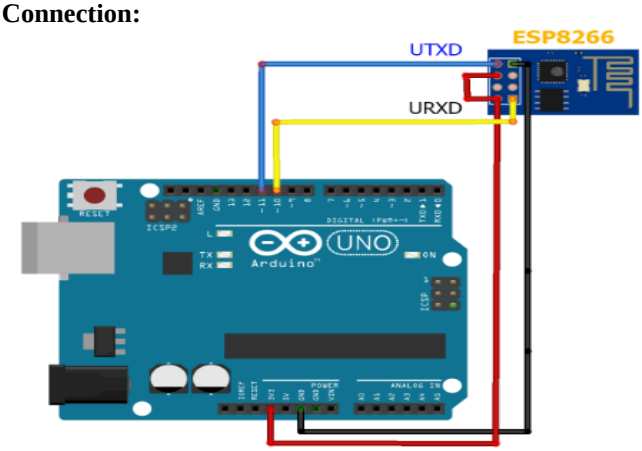

Fig 1. Connecting ESP8266 with Arduino UNO

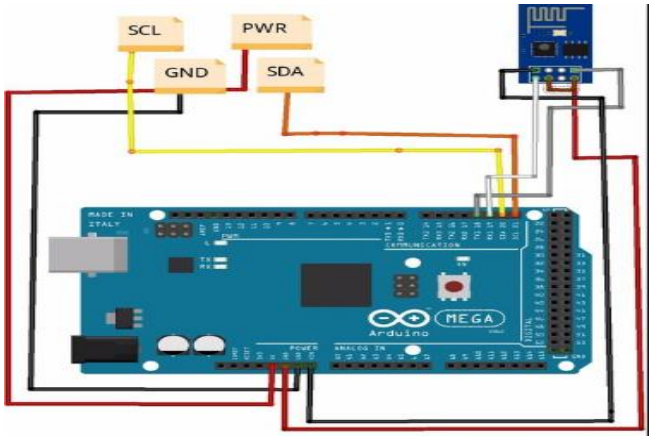

Fig 2. Connecting ESP8266 with Arduino Mega

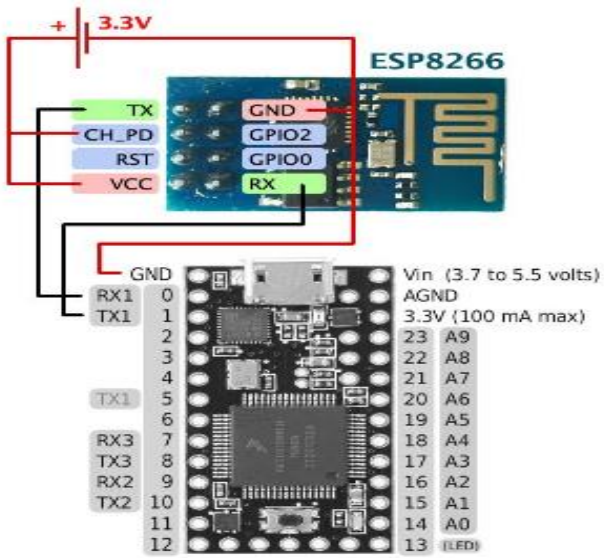

Fig 3. Connecting ESP8266 with Teensy 3.2

Common Steps for Data Transfer and Data Logging :

- Power up the device and upload the code.
- When checking data transfer on **Android Phone** –
  - Start Admin Hands App → Hosts
  - Create a new connection by selecting "+" sign at the bottom left.
  - Fill in the Fields with the IP Address and Port Number of the Wi-Fi module and select Telnet as a medium of transfer.
- When a code is complied, connect the phone's Wi-Fi to the Esp8266 SSID.
- Wait for 20 Seconds from the time of compilation (Delay provided manually in program) and then select the new connection which has just been created by you on the App. If everything is correct you shall see a Blank Black screen on the App.
- If connection is not established properly you will see the following message on the same black screen. "Connecting to "IP Address"...."
- After the successful connection and after 10 seconds you will see the data transfer.
- When checking data transfer on **desktop PC/laptop :**
  - Open Putty
  - Select Logging (Under Session on the left) → All Session Output →Browse the folder

where you want the Logged File to be saved.

- Click Session → Fill in IP Address and Port Number of the module to be connected to → Connection Type – Telnet
- Under the Saved Sessions bar give a name and click on Save to save all the above settings so as to use them in the future again instead of doing all the above steps every time.
- Once code is complied, start the Putty Session and after 20 Seconds you should see the Data Transfer. And after entire file is transferred, close putty and browse to the file where data is logged to get the file. File will not be created until Putty is closed.

Following AT command set was used in the tests:

AT - Test module response

AT+GMR - Module Information

AT+IPR=2000000 - Change the baud rate of the module – (Old Firmware)

AT+UART_DEF=2000000,8,1,0,1 - Change the baud rate of module – (New Firmware)

AT+CWMODE=2 - Change mode of the module 1-3 (1-client,2-Server,3- Server+client)

AT+CIPMUX=1 - Accept multiple connections (0-Single , 1- Multiple)

AT+CIPSERVER=1,80 - Set module as the server

AT+CIPAP="192.168.4.1" - Set the IP Address of the module in Server mode

AT+CIPSTA="192.168.5.1" - Set IP address of module in client mode

AT+CWSAP="DRILL","password",3,2 - Set the SSID and PW of the module

AT+CIPSTAMAC? - Get current MAC address of the module in station mode

AT+CIPAPMAC? - Get current MAC Address of the module in SoftAP mode.

AT+CIPMODE=1 – Put module in Transparent Transmission Mode.

AT+CIPSTART=1,"TCP","ip address","port" (when AT+CIPMUX=1) //doesn't work on new f/w (This command is for the client mode)

AT+CIPSTART="TCP","ip address","port" (when AT+CIPMUX=0)

AT+CIFSR - Get Ip address of module as the client

AT+CIPSEND=0,2048 - Send data packets

IV. EXPERIMENTS

1. ESP8266 with Arduino Uno (Server Mode)

The ESP8266 was first tested with Arduino UNO. The hardware-serial was used as it has dedicated Tx, Rx pins which results in faster communication than the software-serial.

The Following sketch was uploaded in Arduino UNO:

```
#include <SPI.h>
#include <SD.h>
#define TIMEOUT 5000 // mS
#define LED 13
char buf[512];
const int chipSelect = 10;
File myFile;
char invar = 0;
char invar1 = 0;
//-----------------------------------------------
void setup()
{
pinMode(LED,OUTPUT);
Serial.begin(4000000);
/*--------------SD CARD INIT----------------------*/
//Serial.print("Initializing SD card...");
pinMode(SS, OUTPUT);
if (!SD.begin(chipSelect)) {
   return;
  }
SendCommand("AT", "Ready");
SendCommand("AT+CWMODE=2","OK");
Serial.println("AT+CWSAP=\"TFM_DRILL\",\"password\",
3,2");
//SendCommand("AT+CIFSR", "OK");
SendCommand("AT+CIPMUX=1","OK");
SendCommand("AT+CIPSERVER=1,80","OK");
/*----------------FIND IMPORT STRING--------------------*/
String IncomingString="";
boolean StringReady = false;
delay(25000);
StringReady= true;
if (StringReady){
 /*----------------READ FILE IF FOUND IMPORT
STRING---------------------*/
  myFile = SD.open("DATA50.TXT");
 if (myFile)
  {
    while (myFile.available())
    {
     Serial.println("AT+CIPSEND=0,512");
     while(1){
       if(Serial.find(">")){
         invar = 1;
        }
       if(invar == 1) break;
       }
     myFile.read(buf,512);
     Serial.write(buf,512);
     while(1){
       if(Serial.find("OK")){
         invar1 = 1;
        }
       if(invar1 == 1)
       break;
       }
     invar = 0;
     invar1 = 0;
    }
    myFile.close();
  }
 else
  {
    Serial.println("error opening test.txt");
  }
 }
}
/*-------------------------LOOP---------------------*/
void loop(){
}
```

```
/*-------------------FUNCTIONS-------------------------*/
boolean SendCommand(String cmd, String ack){
 Serial.println(cmd); // Send "AT+" command to module
 if (!echoFind(ack)) // timed out waiting for ack string
   return true; // ack blank or ack found
}

boolean echoFind(String keyword){
byte current_char = 0;
byte keyword_length = keyword.length();
long deadline = millis() + TIMEOUT;
while(millis() < deadline){
 if (Serial.available()){
   char ch = Serial.read();
   //Serial.write(ch);
   if (ch == keyword[current_char])
     if (++current_char == keyword_length){
      //Serial.println();
      return true;
   }
  }
 }
return false; // Timed out
}
/*--------------------END OF FUNCTIONS------------------*/
```

For this setup, 1Mb file took 35 Seconds to transfer. The limitation of this setup was the memory size. Due to this we could only send buffered data of maximum 512 characters. This caused added cycles of the loop in the program which increased the processing time which ultimately resulted in fairly slow transfer speed.

2. ESP8266 with Arduino Mega (Server Mode):

To solve the above limitation in the case of Arduino Uno, we switched to Arduino Mega which has more memory than the Uno. The Mega could handle large buffer size and so could increase the transfer speed.
We made some changes in the code which was used in the Arduino UNO. The buffer size which was 512 bytes in the Arduino UNO was made 2048 bytes in the Arduino Mega. The Mega, could handle large amount of data in a single go. Also, the Serial baud rate was increased to 5000000 from 4000000.
For this setup, 1Mb file took 15 Seconds to transfer. Transfer time is significantly improved as now we are sending data in block sizes of 2048 which is the maximum sending size for the ESP8266.

3. ESP8266 with Teensy 3.2 (Server Mode)

Finally we tested the ESP8266 with the Teesny 3.2 which is way faster than the Arduino Uno and Mega. Also, it has more memory than the other two.
In this setup, 1Mb data took 5 Seconds to transfer. This was the fastest time achieved amongst the 3 methods. We are sending data at the maximum baud rate of 5 Million and maximum buffer size of 2048.

4. ESP8266 with Teensy (Client Mode- Python Server)

The connection for this setup is same as the server mode. The only difference here is that we configure the ESP module in client mode and send data through the Transparent Transmission mode of the module. For the server we use a script which is written in Python.

Python Server Sketch:

```
import socket               # Import socket module
import time
s = socket.socket()         # Create a socket object
host = '192.168.0.119' # Get local machine name
port = 80                   # Reserve a port for your service.
s.bind((host, port))        # Bind to the port
f = open('Got_File.txt','wb')
s.listen(5)                 # Now wait for client connection.
while True:
   c, addr = s.accept()     # Establish connection with client.
   print ('Got connection from', addr)
   start_time = time.time()
   print ("Receiving Data from Client...")
   l = c.recv(1024)
   while (l):
      print ("Receiving...")
      f.write(l)
      #print (l)
      l = c.recv(1024)
   f.close()
   #s.shutdown(socket.SHUT_WR)
   print ("Done Receiving")
   print("--- %s seconds ---" % (time.time() - start_time))
   c.send("Thank you for connecting")
   c.close()
/********** End of Code**********************/
```

For this setup, 1Mb file took 3 minutes to transfer. Work is still underway on this setup. We need to reduce the transfer times as the current results are not ideal for us. So we are still sticking with the results we got using the “ESP with teensy – Server Mode”.

V. RESULTS

Following are the final and best results that we have achieved from all scenarios and setups.

Arduino Uno and ESP:
Server Mode:

Baud Rate: **4 Million**	Buffer Size: **512**
File Size: **1Mb**	Time: **35 Seconds**
File Size: **8Mb**	Time: **2 Minutes 30 Seconds**
Data Integrity: **100%**	

Arduino Mega and ESP:
Server Mode:

Baud Rate: **5 Million**	Buffer Size: **2048**
File Size: **1Mb**	Time: **15 Seconds**
File Size: **8Mb**	Time: **2 Minutes**
Data Integrity: **100%**	

Teensy and ESP:
Server Mode:

Baud Rate: **5 Million**	Buffer Size: **2048**
File Size: **1Mb**	Time: **4 Seconds**
File Size: **8Mb**	Time: **35 Seconds**
Data Integrity: **100%**	

Teensy and ESP:
Client Mode: Python Server: Transparent Transmission Mode

Baud Rate: **2 Million**	Buffer Size**: No Buffer**
File Size: **1Mb**	Time: **3 Minutes**
File Size: **8Mb**	Time: Not Tested
Data Integrity: **100%**	

VI. CONCLUSION

Performing several experiments on ESP8266 and various Microcontrollers, we came to a conclusion that the large amount of data can be transferred using the ESP8266 Wi-Fi module along with the teensy 3.2 in a small amount of time. Thus, the need of the wireless transfer of data from an external sd card to the Android device or the desktop PC can be fulfilled by the use of Wi-Fi module as a communication medium.

REFERENCES

[1] Yoonsun, "Wearable Devices," Joono OpenStory, 2014, pp. 30-96.
[2] Harold Davis, "Absolute Beginner's Guide to Wi-Fi Wireless Networking," Que Publishing, 2004, pp. 51-203
[3] Jim Geier, "Designing and Deploying 802.11n Wireless Networks (Networking Technology)," Kindle Edition, pp. 87-150.
[4] Wesley J Chun, "Core Python Programming (2nd Edition)," Kindle Edition, pp. 1-196.
[5] Lubanovic Bill, "Introducing Python," Shroff Publishers, pp. 107-180.
[6] Jin-Shyan Lee, Yu-Wei Su, and Chung-Chou Shen, "A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi," *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*, 5-8 Nov. 2007, pp. 46-5
[7] Kok-Kiong Yap, Te-Yuan Huang, Masayoshi Kobayashi, Yiannis Yiakoumis, Nick McKeown, Sachin Katti, and Guru Parulkar, "Making use of all the networks around us: a case study in android," *CellNet '12 Proceedings of the 2012 ACM SIGCOMM workshop on Cellular networks: operations, challenges, and future design*, 2012, pp. 19-24
[8] Notes on the inexpensive ESP8266 WiFi module. Available: http://www.labradoc.com/i/follower/p/notes-esp8266
[9] The easy way to build Internet of Things. Available: http://iot-playground.com/
[10] Connecting the ESP8266 to an Arduino. Available: http://www.teomaragakis.com/hardware/electronics/how-to-connect-an-esp8266-to-an-arduino-uno/
[11] Data Integrity, definition and introduction. Available: https://docs.oracle.com/cd/A57673_01/DOC/server/doc/SCN73/ch7.htm

Gaurav Khadse, received his B.E. in Electronics and Telecommunications from University of Pune. Specialized in Embedded Systems –wearable devices and computing. Played a key role in developing TFM-drill, a wearable device for tracking a football player, a device developed under thefootballmind.com. Working since July 2014 as an Embedded Developer at Connasys.com. pvt. ltd, Pune, India.

Ninad Adhav, received his B.E. in Electronics and Telecommunications from University of Pune. Worked in the field of Networking and Wearable computing at thefootballmind.com. Specialized in Networking and Telecommunications and is currently pursuing Master of Science at University of Texas at Dallas.