Python Radio 25: UDP networking

Simon Quellen Field

Simon Quellen Field

Follow

7 min read

·

Sep 16, 2024

Listen

Share

More

One Datagram at a time

Press enter or click to view image in full size

Photo by the author

On a recent hike, a friend of mine surprised me. She is not someone you would expect to be interested in either radio or Python programming, but she had casually asked me what I had been up to lately, and I described one of the Morse code projects in this series.

She thought having Morse code radios to call her son to dinner would be fun, as he frequently ignored her texts. His room is in a separate building about a hundred feet from the kitchen.

Any of the little transceiver modules I have written about in this series would work fine. Still, I took this as an opportunity to explore an aspect of networking I had not used much before. Connectionless data packets using UDP over Wi-Fi.

Most Wi-Fi communication involves setting up a connection between two computers. When you read your email or visit a web page, the two computers set up a protocol using TCP/IP. This allows them to resend data packets if they get lost or damaged, and involves two-directional communication where the receiver responds to let the sender know the data was received properly.

With UDP, packets are just sent. It is simple and quick, and if a packet is lost or corrupted along the way, the assumption is that this is tolerable.

In the Morse code radio I will be describing here, several things combine to make such unreliable communication work just fine. The first one is that there is a person at each end who can ask the other to repeat anything that didn’t come through properly. This is quite common in CW radio sent by hand with Morse code keys.

Another thing in our favour here is that switches bounce.

When a person presses a switch or a button, there is a very short period when the contacts touch and then bounce back open, often several times. This is usually a problem, and code is written to wait until the bounces settle down before testing whether the switch is actually open or closed.

In our application here, we ignore that. Every time we notice the switch has made contact, we send a data packet telling the other radio to beep. We might send a dozen or more packets in a few milliseconds. At the other end, any tiny interruption in the sound only happens in the first millisecond or two and is scarcely noticeable.

When the switch is opened, there is also some bouncing. We thus have redundancy. We send several packets for ON, and several for OFF, and if a few are missed, it is no big deal.

There is a chance that we will send an ON signal and the OFF signals will be missed. The receiver would keep making noise. But our code makes noise for the sender as she presses the key, so if her receiver is stuck, she can just tap the key to stop it. The son would hear a quick beep at his end, but that would not be interpreted as a Morse message.

So now I needed to build two transceivers. I have a number of nice (but expensive) Morse code keys:

Press enter or click to view image in full size

Photo by the author

For this project, however, I wanted something as cheap as I am. I found some six-dollar keys on Amazon.com made from strips of metal on a plastic base that fit the bill nicely:

Press enter or click to view image in full size

Photo by the author

The little Wemos D1 Mini is the perfect computer for this project. I get them ten at a time from Aliexpress.com for less than two dollars each. They can send reliably over Wi-Fi for over 350 feet, which is more than enough range for the dinner bell.

Lastly, we need a cheap little speaker. I have a number of these little 2-inch speakers around, and they are perfect, and probably cost me fifty cents each a few years back. AliExpress has some like it for eighty-eight cents each. Since we will be flashing the LED on the D1 Mini as well as making beeps, even the speaker is optional if you are cash-strapped.

There are only four places where we need to solder something. The wires from the key connect to pins D8 and D7 on the D1 Mini. The speaker wires connect to D3 and D7. Rather than try to solder two wires to D7, I connected the key to the speaker terminal that was connected to D7, since the solder pad there was bigger.

The result looks like this:

Press enter or click to view image in full size

Photo by the author

All that’s left is to write the code. Here it is:

From name import name, receiver, address

From network import WLAN, STA\_IF, AP\_IF, AUTH\_OPEN

Try:

Import usocket as socket

Except:

Import socket

From machine import Pin, PWM

Import network

Ip = 0

Dgram\_socket = 0

SEND\_ON = “on”

SEND\_OFF = “off”

UDP\_PORT = 5001

LED = Pin(2, Pin.OUT)

LED.value(1) # Start with the LED off

D3 = Pin(0, Pin.OUT)

D7 = Pin(13, Pin.OUT)

D8 = Pin(15, Pin.IN, Pin.PULL\_UP)

ANY\_CHANGE = const(3)

KEY = D8

SPEAKER = D3

Pwm = PWM(SPEAKER, freq=800, duty=1023)

D7.value(1)

Def key\_changed(pin):

Global LED, pwm, ip, dgram\_socket

If pin.value():

LED.value(0)

Pwm.duty(512)

If ip and dgram\_socket:

Dgram\_socket.sendto(SEND\_ON, (ip, UDP\_PORT))

Else:

LED.value(1)

Pwm.duty(1023)

If ip and dgram\_socket:

Dgram\_socket.sendto(SEND\_OFF, (ip, UDP\_PORT))

KEY.irq(trigger=ANY\_CHANGE, handler=key\_changed)

Def say\_OK():

From time import sleep

For x in range(6):

Pwm.freq(500 + x \* 100)

Pwm.duty(512)

Sleep(.1)

Pwm.duty(1023)

Sleep(.1)

Pwm.freq(800)

Def connect():

From time import sleep

Global name, ip, address

Ssid = “”

Station = WLAN(STA\_IF)

Station.active(True)

Try:

Network\_list = station.scan()

Except Exception as e:

Station.disconnect()

Station.active(False)

Station = WLAN(STA\_IF)

Station.active(True)

Network\_list = station.scan()

Print(“Available networks:”)

For net in network\_list:

Ssid\_name = net[0].decode(“utf-8”)

Rssi = net[3]

If ssid\_name.find(receiver, 0, len(receiver)) == 0:

Print(“ -> “, ssid\_name, rssi)

Ssid = ssid\_name

Else:

Print(“ “, ssid\_name, rssi)

Ap = WLAN(AP\_IF)

Ap.ifconfig((address, “255.255.255.0”, “10.90.20.1”, “1.1.1.1”))

Ap.active(True)

Ap.config(essid=name, authmode=AUTH\_OPEN)

While station.isconnected() == False:

Print(“Connecting to: “, ssid)

Station.connect(ssid, “”)

For x in range(50):

Sleep(0.1)

Print()

Ip = station.ifconfig()[0]

Print(ip)

Ap.config(essid=name, authmode=AUTH\_OPEN)

Print(“Connected to”, ssid)

Say\_OK()

Def main():

Global dgram\_socket

Connect()

Dgram\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

Dgram\_socket.bind((‘’, UDP\_PORT))

While True:

Packet = dgram\_socket.recv(128)

If packet == b’’:

Connect()

Dgram\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

Dgram\_socket.bind((‘’, UDP\_PORT))

Else:

Msg = packet.decode(“utf-8”)

If msg == “on”:

LED.value(0)

Pwm.duty(512)

Else:

LED.value(1)

Pwm.duty(0)

Main()

Let’s walk through it.

The little blue LED on the D1 Mini is connected to pin 2.

We set up D3 and D7 as outputs, and D8 as an input for the key. We set D7 to 1 so that when the key is pressed, D8 will also read as 1.

The key\_changed() function is called whenever D8 changes state. If D8 reads as a 1, we turn on the LED (by sending a 0 to pin 2) and we beep by setting the duty cycle on the PWM to 512 (to generate a square wave).

Lastly, the function sends a packet with either “on” or “off” as the data.

It takes each radio a second or three to boot up and find the other radio’s Wi-Fi SSID. When it is ready, it emits six quick beeps each 100 Hertz higher than the last. This lets the user know that communication can now take place, as both ends are ready.

The connect() function scans the Wi-Fi SSIDs it can see, looking for the SSID of its partner. It connects to that as a “station”. It then sets up its own SSID and sets up as an access point. Each radio connects to the other radio’s access point using its “station” connection. The IP address of the other radio is stored in the variable “ip” to be used when sending packets in the key\_changed() function.

The main() function calls connect() and then creates a socket for the UDP datagrams. It listens to port 5001 (UDP\_PORT) for packets sent to that port by the other radio. When it sees a packet, it turns on or off the LED, and makes the speaker beep or be quiet.

This project also works as a remote doorbell, or as a notifier when the postman opens and closes the mailbox. It can alert you when someone steps on a doormat, opens a refrigerator, or when the bird feeder is empty.

You can also combine this with the Morse text-to-code program to send messages to either radio from your laptop, phone, or desktop computer.

In addition to the main.py program shown above, there is a name.py module for each radio. The one for Iris looks like this:

Name = “Iris”

Receiver = “Ryan”

Address = “10.10.10.10”

The one for her son Ryan looks like this:

Name = “Ryan”

Receiver = “Iris”

Address = “10.10.10.10”

Having the names in the name.py file allows the main.py file to be the same for both radios. This was very convenient during the programming and debugging phases, as I could load main.py onto each machine after making changes, and not have to have a different main.py for each radio.

Radio Hackers

Python

Udp