Wi-Fi Direct on Raspberry Pi — Beginner-Friendly Setup Guide with File Sharing Example

Are you working on an IoT project, want to stream data wirelessly, or simply connect two devices without needing a Wi-Fi router? **Wi-Fi Direct** allows your Raspberry Pi to communicate directly with another device (like a phone or laptop) without needing an internet connection or access point.

In this guide, we'll walk step-by-step through setting up Wi-Fi Direct on a Raspberry Pi, configuring a DHCP server, and finally sharing files (like videos or PDFs) using Python.

🔧Step 1: Set Up a DHCP Server

What is DHCP and Why is It Needed?

When devices connect to a network, they need an IP address to communicate. A **DHCP server** is responsible for assigning these IPs. In our Wi-Fi Direct setup, your Raspberry Pi acts as a mini-router for the other device, so it must assign IP addresses — that's why we set up our own DHCP server.

1.1 Assign a Static IP to Wi-Fi Direct Interface

We must assign a **static IP** to the Raspberry Pi's Wi-Fi Direct virtual interface (usually `p2p-wlan0-0`) so that it's always reachable at a known address.

```
sudo nano /etc/dhcpcd.conf
```

Add these lines:

```
interface p2p-wlan0-0  # This is your Wi-Fi Direct interface name
static ip_address=192.168.0.34  # This is the static IP you want the Pi to use
static routers=192.168.0.1      # This is a fake router IP (we're not routing, but it's required syntactically)
static domain_name_servers=192.168.0.34 8.8.8.8  # Local DNS and Google DNS fallback
```

Why these?

- `interface` binds the config to a specific interface.
- `static ip_address` ensures your Pi always has this address when acting as the Wi-Fi Direct host.
- `routers` and `domain_name_servers` are placeholders that prevent resolution failures even if no internet is used.

1.2 Install and Configure dnsmasq (DHCP Server)

We use `dnsmasq`, a lightweight tool that can handle DHCP and DNS.

```
sudo apt install dnsmasq
sudo nano /etc/dnsmasq.conf
```

Paste this configuration:

```
interface=p2p-wlan0-0  # The interface to serve IPs on
bind-dynamic            # Binds when the interface is up
domain-needed           # Don't forward short names (like 'localhost') to DNS
bogus-priv              # Blocks private IP lookups from going to upstream DNS

# Range of IPs to assign to peers
# Make sure it doesn’t overlap with the static IP above

dhcp-range=192.168.0.100,192.168.0.200,255.255.255.0,24h
```

1.3 Restart Services

After setting everything, restart the dnsmasq service and reboot to apply changes:

```
sudo service dnsmasq restart
sudo reboot
```

📂Step 2: Configure Wi-Fi Direct Settings

Now, let’s configure the Wi-Fi Direct behavior of the Raspberry Pi so it knows how to behave when a peer device tries to connect.

2.1 Create a Custom Wi-Fi Configuration File

We create a new `wpa_supplicant` config to avoid conflicting with regular Wi-Fi.

```
sudo nano /etc/wpa_supplicant/wpa_supplicant-wlan0.conf
```

Paste the following:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev  # Allow network
interfaces to communicate
update_config=1  # Allow wpa_cli to change this file dynamically
country=DE  # Set your 2-letter country code here

# Device identity shown to peers
device_name=DIRECT-PI

# High value prefers the Pi to act as Group Owner (i.e., access point)
p2p_go_intent=15

# Allow fast Wi-Fi (802.11n)
p2p_go_ht40=1

# Define what type of device this is
# 6-0050F204-1 = network access point
device_type=6-0050F204-1

# Required for Pi to act as P2P device
driver_param=p2p_device=6
```

Explanation:

- `p2p_go_intent`: 0 (least desire to be host) to 15 (most desire).
- `device_type`: Identifies Pi as a network device to others.
- `ctrl_interface` and `update_config` let you manage this via command line.

2.2 Set File Permissions and Enable Service

```
sudo chmod 600 /etc/wpa_supplicant/wpa_supplicant-wlan0.conf
sudo systemctl disable wpa_supplicant.service
sudo systemctl enable wpa_supplicant@wlan0.service
sudo rfkill unblock wlan  # Enable wireless hardware if blocked
```

2.3 Configure systemd-networkd

This makes sure the virtual interface gets IP settings when it appears.

```
sudo nano /etc/systemd/network/12-p2p-wlan0.network
```

Add:

```
[Match]
Name=p2p-wlan0-*

[Network]
Address=192.168.0.34/24
DHCPServer=yes
```

Then reboot:

```
sudo reboot
```

🖥️Step 3: Automate Connection with Shell Script

Instead of typing each command manually, we'll create a script to automate Wi-Fi Direct connection every time.

3.1 Create the Script

```
sudo nano wifi-direct.sh
```

Paste this:

```
sudo su  # Switch to root

# Set connection method (push button)
wpa_cli -i p2p-dev-wlan0 set config_methods virtual_push_button

# Create a P2P group
wpa_cli p2p_group_add
sleep 3

# Re-set GO intent
wpa_cli set p2p_go_intent 15

# Assign IP and bring interface up
ifconfig p2p-wlan0-0 192.168.0.34 netmask 255.255.255.0
ifconfig p2p-wlan0-0 up

# Start listening for connection
wpa_cli p2p_listen
```

```
# Restart DHCP server
device systemctl restart dnsmasq

# Loop to wait for connection
while true; do
  wpa_cli wps_pbc
  sleep 5
done
```

3.2 Run the Script

```
chmod +x wifi-direct.sh
./wifi-direct.sh
```

This will continuously listen for a peer trying to connect and assign it an IP.

3.3 Check dnsmasq status

```
sudo service dnsmasq status
```

This helps confirm your DHCP server is working correctly.

📤Step 4: Send Files Over Wi-Fi Direct

After establishing a Wi-Fi Direct connection, you can send files just like over any local network. Let's create a basic TCP file server.

Python File Sharing Script

```
import socket
import threading
import os

SERVER_IP = '192.168.0.34'  # The IP of the Pi
SERVER_PORT = 12345
VIDEO_DIRECTORY = '/path/to/video/files'  # Folder with your video files

def handle_client(client_socket):
    try:
        for filename in os.listdir(VIDEO_DIRECTORY):
            if filename.endswith('.mp4'):
                filepath = os.path.join(VIDEO_DIRECTORY, filename)
                with open(filepath, 'rb') as video_file:
                    while chunk := video_file.read(4096):
                        client_socket.sendall(chunk)
    except Exception as e:
        print(f"Error: {e}")
    finally:
        client_socket.close()

def main():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((SERVER_IP, SERVER_PORT))
    server_socket.listen(5)
    print(f"Server listening on {SERVER_IP}:{SERVER_PORT}")
    try:
        while True:
            client_socket, addr = server_socket.accept()
            print(f"Connection from {addr}")
            threading.Thread(target=handle_client,
args=(client_socket,)).start()
    except KeyboardInterrupt:
        print("Shutting down server.")
        server_socket.close()

if __name__ == "__main__":
    main()
```

This creates a basic TCP server that sends `.mp4` files to any connected client.

📌Bonus: Send PDF Over Wi-Fi Direct

Here's how to generate a PDF and send it:

```
from fpdf import FPDF
import socket

pdf = FPDF()
pdf.add_page()
pdf.set_font("Arial", size=12)
pdf.cell(200, 10, txt="Wi-Fi Direct Report", ln=True, align='C')
pdf.output("/home/pi/report.pdf")

with socket.socket() as s:
    s.connect(("192.168.0.101", 12345))  # Use the peer's IP
    with open("/home/pi/report.pdf", "rb") as f:
        s.sendall(f.read())
```

📁Notes

- `#` are comments for explanation — do not copy/paste them.
- Double-check your IP addresses and file paths.
- You must be running **Raspberry Pi OS with systemd**.

💡Why Use This?

This setup is perfect for:

- Offline IoT communication
- Direct peer-to-peer sensor logging
- Low-latency media sharing

📰Share Your Experience

Have you tried Wi-Fi Direct with Raspberry Pi? Planning a cool use-case?

Leave a comment or message me — and don't forget to share this guide with others building edge-connected systems!

#raspberrypi #iot #wifidirect #linux #python #networking #edgecomputing