Hamming Code Error Detection

start from the execution point and slowly navigate to others

Execution Point

At the end of the script, the program starts execution with the following line:

```python
sniff(iface='Wi-Fi', prn=packet_callback, filter="ip", store=0)
```

Step 1: `sniff()` Function

What happens here?

1. `iface='Wi-Fi'`:
   - Specifies the network interface on which to capture packets. In this case, it targets the interface named "Wi-Fi". This must be the correct name of the interface on your system.
   - If it doesn't match the actual interface name, the program won't capture packets.
2. `prn=packet_callback`:
   - Tells `sniff()` to call the `packet_callback` function each time a packet is captured.
3. `filter="ip"`:
   - Uses a Berkeley Packet Filter (BPF) to restrict captured packets to those containing the IP protocol only (e.g., no ARP or Ethernet-only packets).
4. `store=0`:
   - Ensures packets are not stored in memory, reducing resource consumption during long packet captures.

What happens next?

- The `sniff()` function starts monitoring packets on the "Wi-Fi" interface.
- When a packet matching the `"ip"` filter is captured, it is passed to the `packet_callback` function for processing.

Step 2: `packet_callback(packet)` Function

When a packet is captured, `packet_callback()` is executed with the captured packet as an argument.

Step 2.1: Check if the Packet Contains an IP Layer

```python
if IP in packet:
```

- Verifies whether the captured packet contains an IP layer.
- If `IP` is present, the program continues processing; otherwise, the function exits for that packet.

Step 2.2: Extract Information from the IP Layer

```python
ip_layer = packet[IP]
src_ip = ip_layer.src
dst_ip = ip_layer.dst
protocol = ip_layer.proto
```

- `ip_layer = packet[IP]` : Extracts the IP layer of the packet for further processing.
- `src_ip` : The source IP address (e.g., the sender).
- `dst_ip` : The destination IP address (e.g., the receiver).
- `protocol` : The protocol number field in the IP header, indicating which transport layer protocol (TCP, UDP, ICMP, etc.) is used.

Step 2.3: Determine the Protocol

```python
protocol_name = ""
if protocol == 1:
    protocol_name = "ICMP"
elif protocol == 6:
    protocol_name = "TCP"
elif protocol == 17:
    protocol_name = "UDP"
else:
    protocol_name = "Unknown Protocol"
```

- **ICMP** (`1`): Internet Control Message Protocol, used for diagnostic or control messages.
- **TCP** (`6`): Transmission Control Protocol, a reliable connection-oriented protocol (e.g., for HTTP, HTTPS).
- **UDP** (`17`): User Datagram Protocol, a faster but less reliable protocol (e.g., for DNS, video streaming).
- If the protocol doesn't match the above, it's labeled as "Unknown Protocol."

Step 2.4: Print Packet Details

```python
print(f"Protocol: {protocol_name}")
print(f"Source IP: {src_ip}")
print(f"Destination IP: {dst_ip}")
print("-" * 50)
```

- Displays the protocol type, source IP address, and destination IP address.
- The line `"-" * 50` creates a visual separator for readability.

Execution Flow Recap

1. `sniff()` starts capturing packets on the "Wi-Fi" interface and calls `packet_callback()` for each captured packet.
2. `packet_callback()`:
    - Checks for an IP layer.
    - Extracts the protocol, source IP, and destination IP.
    - Identifies the protocol name.
    - Prints packet details.

This loop continues until you stop the program (e.g., by pressing `Ctrl+C`).

Sample Output

If a TCP packet is captured with the following details:

- **Source IP**: `192.168.1.2`
- **Destination IP**: `192.168.1.1`

The program outputs:

```markdown

Protocol: TCP
Source IP: 192.168.1.2
Destination IP: 192.168.1.1
--------------------------------------------------
```