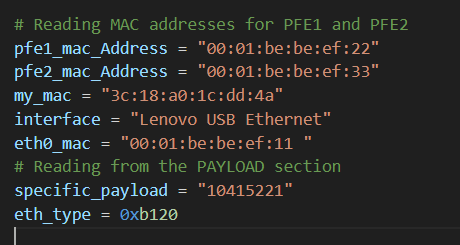
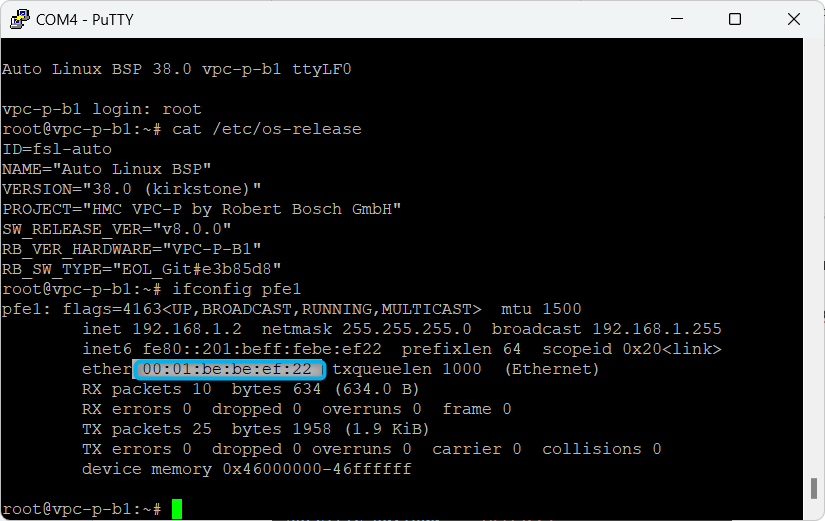
Ethernet Network Test Documentation

# Test Overview

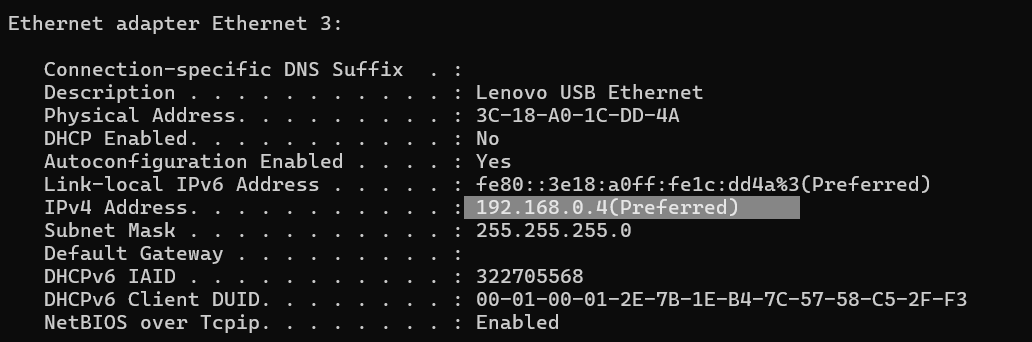
* To run the Ethernet network test manually, use the following script:
* test\_scripts\tester\VPC-P\_Testing\_Framework\Ethernet\_Network\_Test\manual\_test.py
* When running the test, provide the following details: PFE1 MAC address, PFE2 MAC address, setup MAC address, interface name (for the setup), eth0 MAC (for B2/C0) for B1, use the default), payload, and eth\_type. For payload, use "10415221" for inverted and "efbeadde" for non-inverted; for eth\_type, use 0xb120 for inverted and 0xb121 for non-inverted.



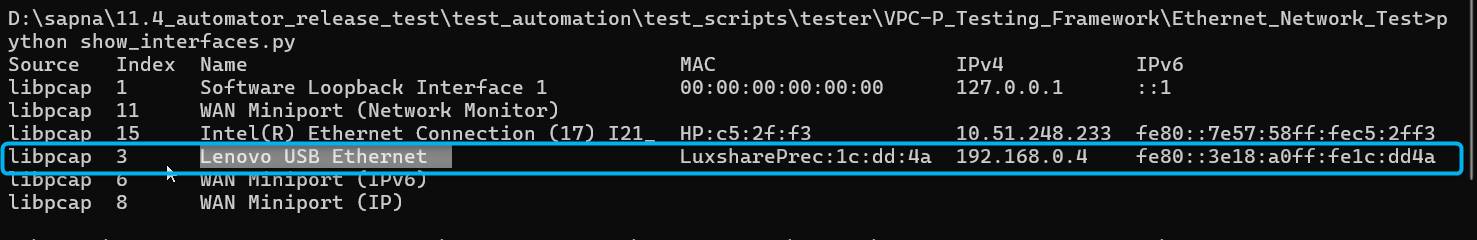
* PFE1, PFE2, and eth0 MAC addresses can be obtained using ifconfig



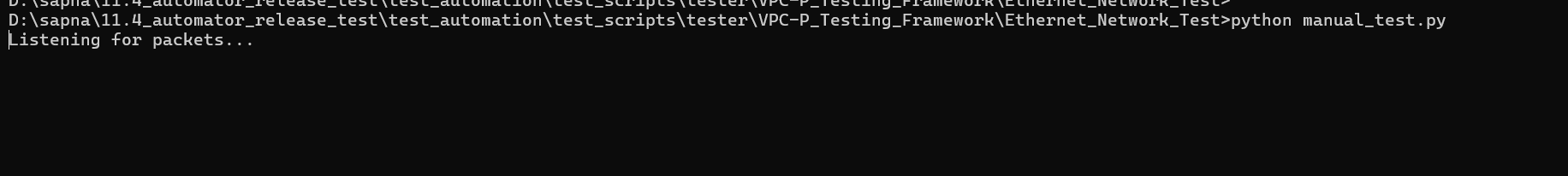
* For the setup's MAC address, run ipconfig /all and check the remote setup's IPv4 address.



* For the interface name, run show\_interface.py (located in the same folder as manual\_test.py).



* First run manual\_test.py to start listening for packets.

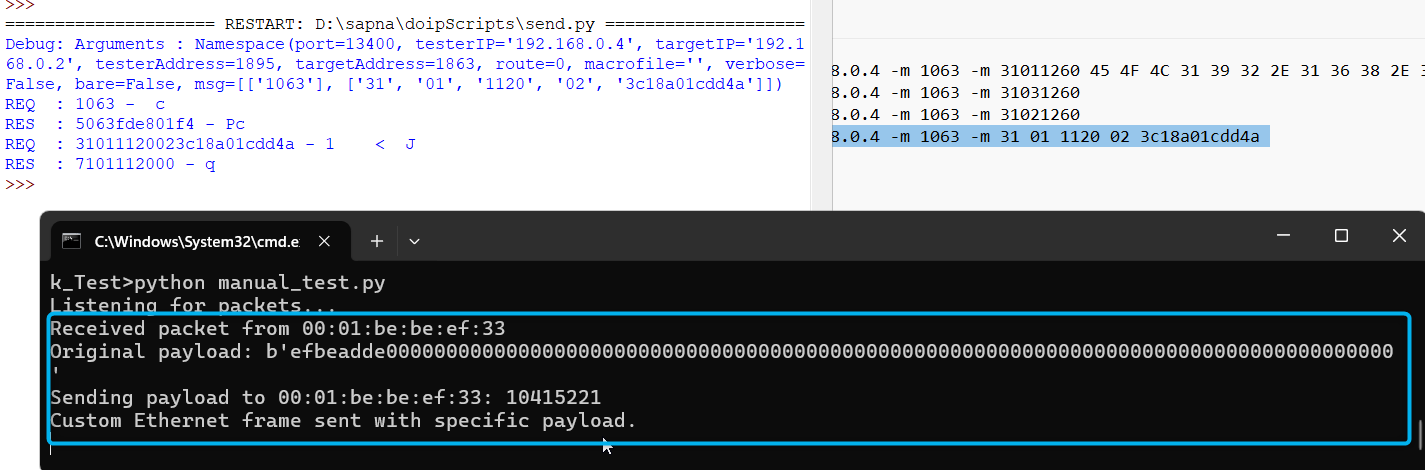


* Then run the XTS commands for inverted and non-inverted Ethernet:

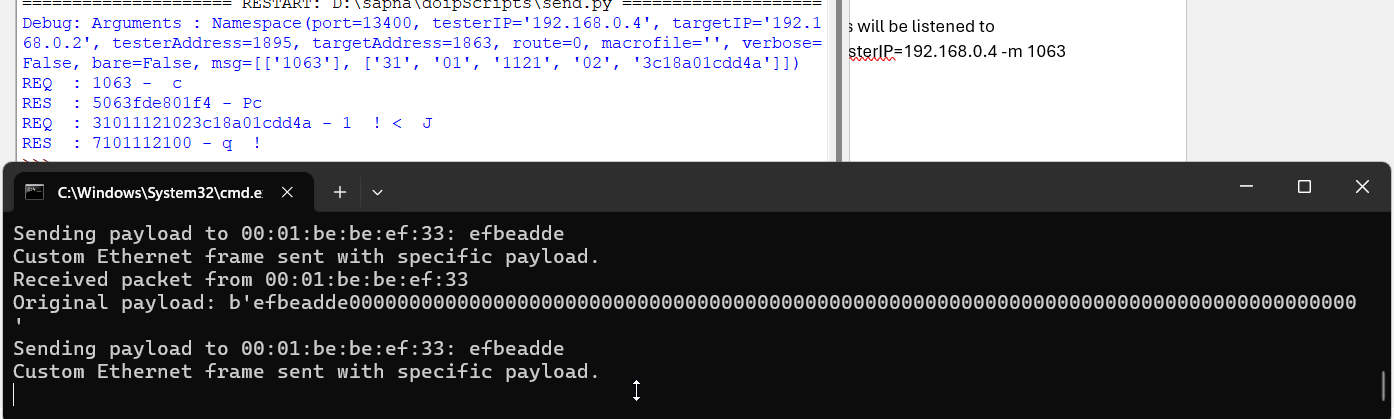
- Inverted Ethernet: specific\_payload = "10415221" and eth\_type = 0xb120.

- Non-inverted Ethernet: specific\_payload = "efbeadde" and eth\_type = 0xb121.

* For test case CMD2: Use the command '31 01 1120 02 <Dest MAC>' (ensure <Dest MAC> is the remote setup MAC address without colons, e.g., 3c18a01cdd4a).
* Example XTS command for inverted Ethernet:
* --targetIP=192.168.0.2 --testerIP=192.168.0.4 -m 1063 -m 31 01 1120 02 3c18a01cdd4a
* (eth\_type = 0xb120 and payload = "10415221")



* For non-inverted Ethernet, the XTS command is:
* --targetIP=192.168.0.2 --testerIP=192.168.0.4 -m 1063 -m 31 01 1121 02 3c18a01cdd4a
* (eth\_type = 0xb121 and payload = "efbeadde")



* Ensure that manual\_test.py is already running and listening for packets before issuing the XTS commands.
* Verify that eth\_type and the DID match; otherwise, no packets will be captured.

# Flow of manual\_test.py / eth\_frames.py

* **Overview:** The script monitors a network interface for Ethernet frames from specific MAC addresses. When a matching packet is detected, a custom Ethernet frame is sent back. This communication occurs at the data link (MAC) layer and is useful for testing, debugging, or implementing proprietary protocols outside of standard IP/TCP/UDP layers.
* **Python packages used:**
* **Scapy (from scapy.all):**

- Ether: Creates the Ethernet header.

- Raw: Attaches a raw payload to the frame.

- sendp: Sends packets at layer 2.

- AsyncSniffer: Enables asynchronous packet sniffing.

• **binascii:** Converts binary data to hexadecimal for debugging.

• **threading:** Launches threads to send responses without blocking packet capture.

• **time:** Keeps the main thread running (using sleep) until manually interrupted.

# Communication at MAC Level

• Layer 2 Communication: The script builds and sends raw Ethernet frames directly using MAC addresses, bypassing higher-level protocols.

• Custom EtherType: A specific EtherType (0xb121/0xb120) distinguishes these custom frames from regular network traffic.

# Script Flow

1. **Initialization of Variables:** The script sets MAC addresses for two devices (referred to as PFE1 and PFE2), the local MAC address (my\_mac), an additional MAC (eth0\_mac), the network interface name, and a custom payload (given in hexadecimal). It also defines a custom EtherType to tag the frames.
2. **Defining the send\_custom\_frame Function:** 
   1. Converts the hexadecimal payload string into bytes.
   2. Constructs an Ethernet frame with:
      1. Destination MAC: The MAC address of the sender of the detected packet.
      2. Source MAC: The local machine’s MAC address (my\_mac).
      3. EtherType: The custom type.
      4. Payload: The raw data (converted from the hex string).
   3. Sends the frame on the specified interface using sendp.
3. **Defining the handle\_packet Callback Function:** This function is triggered whenever the sniffer picks up a packet.
   * 1. checks if the packet has an Ethernet layer, whether the source MAC matches one of the specified addresses (PFE1 or PFE2), and if the EtherType equals 0xb121/0xb120.
     2. If these conditions are met, it prints the source MAC and payload (after converting it to hex for clarity).
     3. It then spawns a new thread to execute send\_custom\_frame, ensuring that sending the response does not block further packet processing.
4. **Setting Up the AsyncSniffer:**
   1. The sniffer is configured to listen on the specified network interface.
   2. A BPF (Berkeley Packet Filter) is applied so that it only captures packets from the specified MAC addresses (including an extra one eth0\_mac).
   3. The handle\_packet function is set as the callback to process each captured packet.
   4. The store=False parameter indicates that packets are not saved in memory, keeping the process lightweight.
5. **Main Execution Loop:**
   1. The sniffer is started asynchronously so it runs in a separate thread.
   2. The main thread enters an infinite loop (using time.sleep(1)) to keep the program running.
   3. A try/except block handles a KeyboardInterrupt (such as pressing Ctrl+C) to gracefully stop the sniffer and exit.

# Protocol and Communication Flow

* The communication is based on a custom protocol built on top of Ethernet. Instead of relying on standard protocols (like IP), the script creates and processes raw Ethernet frames. The specific EtherType distinguishes these frames from others.
* **Flow Summary:**
* Listening: AsyncSniffer monitors the interface for Ethernet frames from PFE1, PFE2, or the additional MAC that match the custom EtherType.
* Packet Detection: When a matching frame is detected, handle\_packet is called.
* Payload Processing: The raw payload is extracted and printed.
* Response: A new thread is spawned to send a custom response frame using the predefined payload.
* Looping: The script continues to listen and respond until manually stopped.

# Why Is It Needed?

* Testing and Debugging: It enables testing device communication at the MAC layer without higher-level protocols, useful for hardware testing or developing proprietary protocols.
* Low-Level Communication: Direct manipulation of Ethernet frames allows simulation of network behaviors, diagnostics, or specialized network interactions not possible with standard IP-based communications.
* In summary, the script uses Scapy to create, send, and sniff custom Ethernet frames with a defined payload and EtherType. It listens for frames from specific devices, processes them, and responds in real time using threading—all operating at the MAC layer.