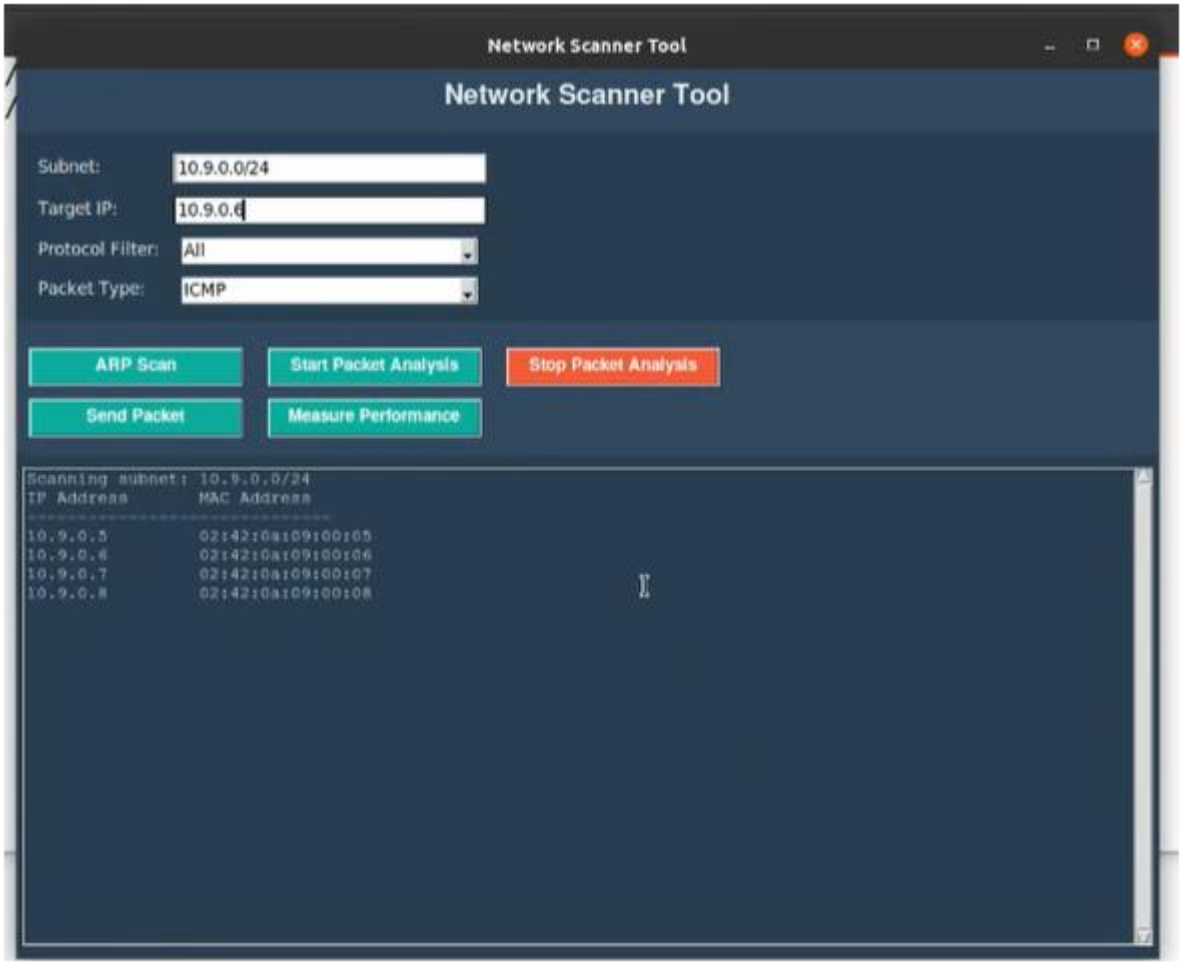
1. Network Discovery (ARP Scan)

```
def arp_scan_gui():
    subnet = entry_subnet.get()
    if not subnet:
        messagebox.showerror("Input Error", "Please enter a subnet.")
        return

    output_text.delete(1.0, tk.END)
    output_text.insert(tk.END, f"Scanning subnet: {subnet}\n")

    def perform_arp_scan():
        arp = ARP(pdst=subnet)
        ether = Ether(dst="ff:ff:ff:ff:ff:ff")
        packet = ether / arp
        answered, _ = srp(packet, iface="br-979cf59e77a5", timeout=2, verbose=0)

        output_text.insert(tk.END, "{:<15}  {}\n".format("IP Address", "MAC Address"))
        output_text.insert(tk.END, "-" * 30 + "\n")

        for sent, received in answered:
            ip = received.psrc
            mac = received.hwsrc
            output_text.insert(tk.END, "{:<15}  {}\n".format(ip, mac))
            log_to_csv("ARP Scan", f"{ip}, {mac}")

    Thread(target=perform_arp_scan).start()
```

- **Performs an ARP scan on a user-specified subnet to discover devices on the network.**
- **Displays the IP and MAC addresses of detected devices in the GUI and logs the results into the CSV file.**

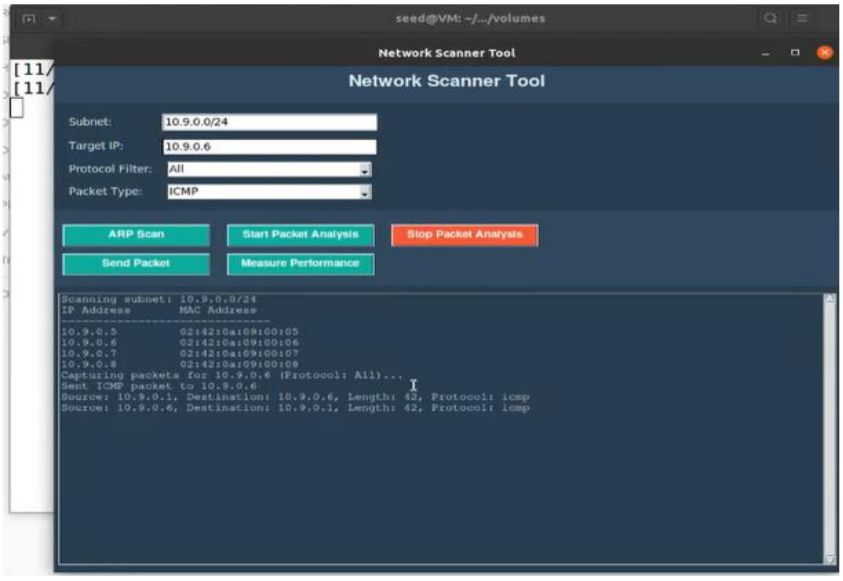


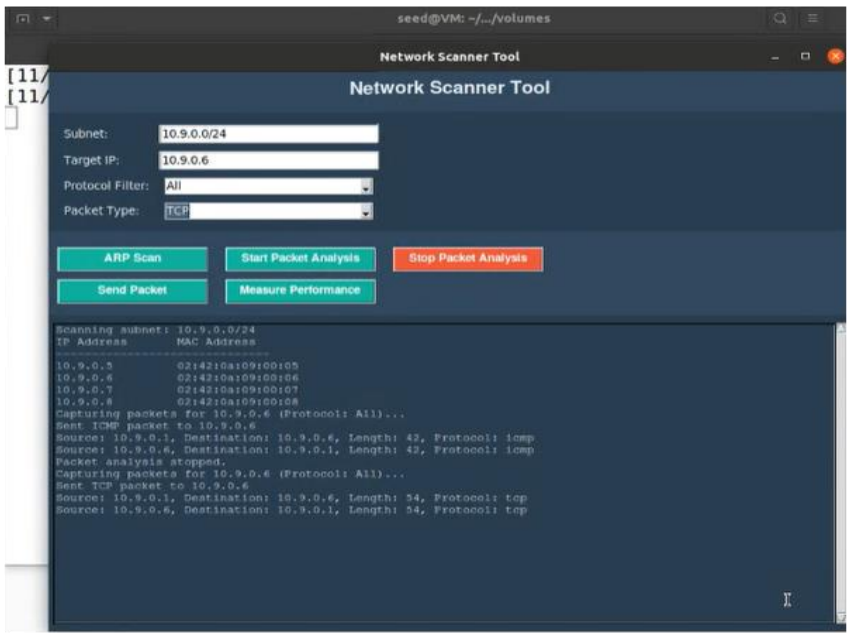
2. Packet Analysis

```
def packet_analysis_gui():
    global packet_analysis_active
    target_ip = entry_target_ip.get()
    protocol_filter = protocol_choice.get()

    if not target_ip:
        messagebox.showerror("Input Error", "Please enter a target IP address.")
        return

    output_text.insert(tk.END, f"Capturing packets for {target_ip} (Protocol: {protocol_filter})...\n")

    def analyze_packet(packet):
        if IP in packet:
            src_ip = packet[IP].src
            dst_ip = packet[IP].dst
            length = len(packet)
            proto = packet.sprintf("%IP.proto%")

            if protocol_filter == "All" or proto == protocol_filter:
                output_text.insert(tk.END, f"Source: {src_ip}, Destination: {dst_ip}, Length: {length}, Protocol: {proto}\n")
                log_to_csv("Packet Analysis", f"Source: {src_ip}, Destination: {dst_ip}, Length: {length}, Protocol: {proto}")

    def sniff_packets():
        while packet_analysis_active:
            sniff(iface="br-979cf59e77a5", filter=f"host {target_ip}", prn=analyze_packet, timeout=2)
```

```
    if not packet_analysis_active:
        packet_analysis_active = True
        Thread(target=sniff_packets).start()

def stop_packet_analysis():
    global packet_analysis_active
    packet_analysis_active = False
    output_text.insert(tk.END, "Packet analysis stopped.\n")
    log_to_csv("Packet Analysis", "Packet analysis stopped")
```

- **Starts sniffing packets for a target IP address, filtered by a user-selected protocol (e.g., All, TCP, UDP).**
- **Analyzes packets in real-time and logs details like source IP, destination IP, protocol, and packet length.**
- **Stops the ongoing packet analysis process by setting a global control variable to False.**
- **Logs the action and provides a status update in the GUI.**





3. Custom Packet Creation

```
def send_custom_packet_gui():
    target_ip = entry_target_ip.get()
    packet_type = packet_type_choice.get()

    if not target_ip:
        messagebox.showerror("Input Error", "Please enter a target IP address.")
        return

    if packet_type == "ICMP":
        packet = IP(dst=target_ip) / ICMP()
    elif packet_type == "TCP":
        packet = IP(dst=target_ip) / TCP(dport=80)
    else:
        packet = IP(dst=target_ip) / UDP(dport=53)

    send(packet, verbose=0)
```

```
    output_text.insert(tk.END, f"Sent {packet_type} packet to {target_ip}\n")
    log_to_csv("Custom Packet", f"Sent {packet_type} packet to {target_ip}")
```

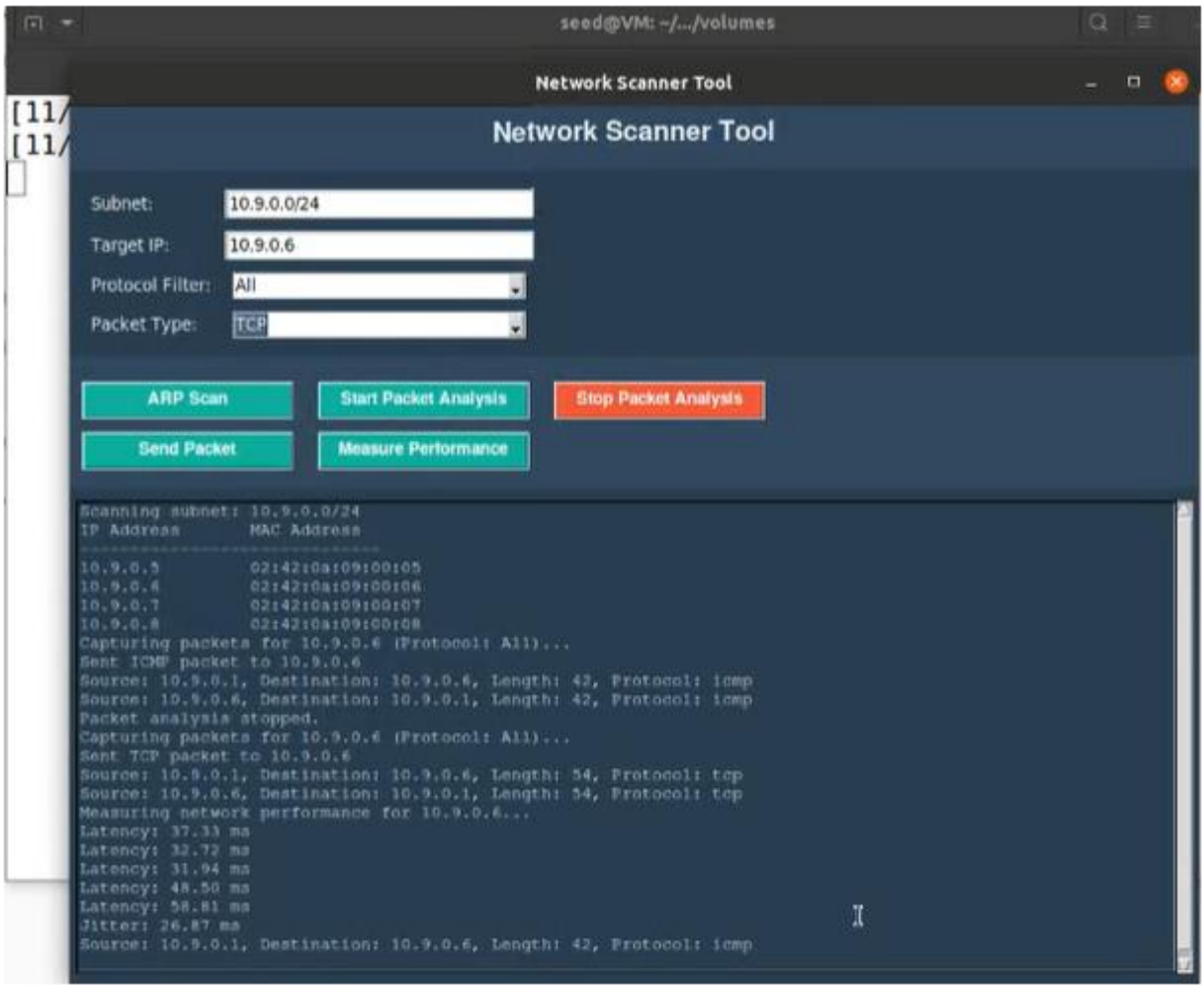
- **Sends a custom packet (ICMP, TCP, or UDP) to a specified target IP address based on the user's choice.**
- **Provides feedback in the GUI about the packet sent and logs the action into the CSV file.**

4. Network Performance Measurement

```
------------------------------------------------------------
def calculate_network_performance_gui():
    target_ip = entry_target_ip.get()
    if not target_ip:
        messagebox.showerror("Input Error", "Please enter a target IP address.")
        return

    latencies = []
    output_text.insert(tk.END, f"Measuring network performance for {target_ip}...\n")

    for _ in range(5):
        start_time = time.time()
        response = sr1(IP(dst=target_ip) / ICMP(), timeout=2, verbose=0)
        if response:
            latency = (time.time() - start_time) * 1000
            latencies.append(latency)
            output_text.insert(tk.END, f"Latency: {latency:.2f} ms\n")
        else:
            output_text.insert(tk.END, "Request timed out\n")

    if len(latencies) > 1:
        jitter = max(latencies) - min(latencies)
```

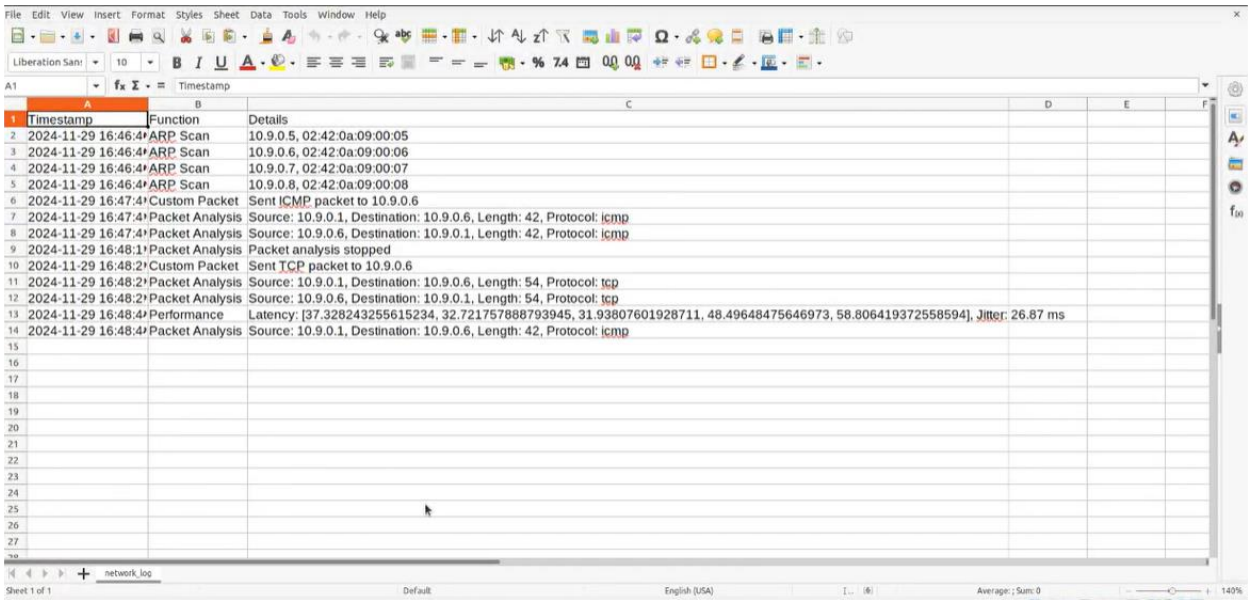
```
else:
    jitter = 0.0

output_text.insert(tk.END, f"Jitter: {jitter:.2f} ms\n")
log_to_csv("Performance", f"Latency: {latencies}, Jitter: {jitter:.2f} ms")
```

- **Measures network latency and jitter for a target IP by sending ICMP echo requests and timing the responses.**
- **Displays latency and jitter results in the GUI and logs them into the CSV file for performance analysis.**



```
#Initialize the CSV file for logging
log_file = "network_log.csv"
with open(log_file, "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(["Timestamp", "Function", "Details"])


# Function to log data to CSV file
def log_to_csv(function, details):
    with open(log_file, "a", newline="") as f:
        writer = csv.writer(f)
        writer.writerow([time.strftime("%Y-%m-%d %H:%M:%S"), function, details])


# Global variable to control packet analysis
packet_analysis_active = False
```

- **This function logs the specified function name and its details (e.g., event information) into a CSV file with a timestamp.**
- **It ensures that all significant actions are documented for later analysis.**



Timestamp	Function	Details
2024-11-29 16:46:4	ARP Scan	10.9.0.5, 02:42:0a:09:00:05
2024-11-29 16:46:4	ARP Scan	10.9.0.6, 02:42:0a:09:00:06
2024-11-29 16:46:4	ARP Scan	10.9.0.7, 02:42:0a:09:00:07
2024-11-29 16:46:4	ARP Scan	10.9.0.8, 02:42:0a:09:00:08
2024-11-29 16:47:4	Custom Packet	Sent ICMP packet to 10.9.0.6
2024-11-29 16:47:4	Packet Analysis	Source: 10.9.0.1, Destination: 10.9.0.6, Length: 42, Protocol: icmp
2024-11-29 16:47:4	Packet Analysis	Source: 10.9.0.6, Destination: 10.9.0.1, Length: 42, Protocol: icmp
2024-11-29 16:48:1	Packet Analysis	Packet analysis stopped
2024-11-29 16:48:2	Custom Packet	Sent TCP packet to 10.9.0.6
2024-11-29 16:48:2	Packet Analysis	Source: 10.9.0.1, Destination: 10.9.0.6, Length: 54, Protocol: tcp
2024-11-29 16:48:2	Packet Analysis	Source: 10.9.0.6, Destination: 10.9.0.1, Length: 54, Protocol: tcp
2024-11-29 16:48:4	Performance	Latency: [37.328243255615234, 32.721757888793945, 31.93807601928711, 48.49648475646973, 58.806419372558594], Jitter: 26.87 ms
2024-11-29 16:48:4	Packet Analysis	Source: 10.9.0.1, Destination: 10.9.0.6, Length: 42, Protocol: icmp