ACN LAB - 02
Study of Wireshark Tool - Socket Programming for TCP Packets Inside UDP with Simulating Packet Drop Analysis Using Wireshark

Chaitanya Talware (MIS No: 712422005)
Yogesh Toshniwal (MIS No: 712422021)

November 9, 2024

1 Introduction

This assignment demonstrates a 3-node system using Python's socket programming. It simulates communication through TCP and UDP protocols. The system tracks packet loss and uses Wireshark to analyze network traffic. Node 1 sends data over TCP to Node 2, which forwards it over UDP to Node 3. Node 3 receives and tracks packet loss, ensuring the system models real-world network conditions effectively.

1.1 Node 1: TCP Server

Node 1 functions as a TCP server that listens for client connections. It receives data from a client and forwards it to Node 2 via TCP. This node generates the data packets that are sent to the next node for further processing.

- Listens for UDP packets from Node 2.
- Tracks received packets and detects losses based on sequence numbers.
- Prints received and lost packet statistics.

1.2 Node 2: TCP-to-UDP Forwarder

Node 2 receives data from Node 1 over TCP, then forwards it to Node 3 using the UDP protocol. It simulates packet loss with a 10% drop rate to mimic network instability, making this node an intermediary in the communication.

- Receives data from Node 1 via TCP.

- Forwards data to Node 3 via UDP.
- Simulates packet loss.

1.3 Node 3: UDP Receiver with Packet Loss Tracking

Node 3 is responsible for receiving UDP packets from Node 2. It tracks the number of packets received and identifies lost packets by comparing the expected sequence number with the actual received packets. It prints out the total number of received and lost packets.

- Listens for incoming TCP connections.
- Receives and forwards data to Node 2.

1.4 Wireshark Analysis

Wireshark is used to capture and visualize the network traffic between the nodes. It provides insights into the transmitted packets, helping confirm the packet loss and verify the system’s behavior under network conditions. Wireshark allows us to:

- Capture live network traffic in real-time.
- Analyze specific protocol packets such as TCP and UDP.
- Inspect packet contents and verify the integrity of data transmission.
- Track packet loss and retransmissions across the network.

2 Source Code

2.1 Node 1 (Server)

```

import socket
import time

def main():
    # Set up TCP server
    server_socket = socket.socket(socket.AF_INET,
        socket.SOCK_STREAM)
    host = ’127.0.0.1’  # Localhost
    port = 12345
    server_socket.bind((host, port))
    server_socket.listen(5)
    print(f"Node 1 (Server) listening on
        {host}:{port}...")

    while True:
        # Accept connection from Node 2
        client_socket, addr = server_socket.accept()
        print(f"Connected with Node 2 at {addr}")

        # Ask for sequence upper limit from user
        upper_limit = int(input("Enter upper limit for the sequence (e.g., 100): "))

        for i in range(1, upper_limit + 1):
            message = str(i)  # Prepare each number as a message
            client_socket.send(message.encode(’utf-8’))  # Send message to Node 2
            print(f"Sent {message} to Node 2")
            time.sleep(0.5)  # Delay to simulate interval between packets

        print("Sequence transmission complete.")
        client_socket.send(b’q’)  # Signal end of sequence
        client_socket.close()
        break  # Exit loop after one client session

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

2.2 Node 2 (TCP-to-UDP Forwarder)

```
# Node 2 (TCP-to-UDP Forwarder)
import socket
import random
import time

def main():
    # Set up TCP client to Node 1
    tcp_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host = ’127.0.0.1’  # IP of Node 1
    port = 12345
    tcp_client.connect((host, port))
    print("Connected to Node 1 over TCP.")

    # Set up UDP socket to send to Node 3
    udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    udp_ip = ’127.0.0.1’  # IP of Node 3
    udp_port = 6000
```

```python

    while True:
        # Receive data from Node 1 over TCP
        data = tcp_client.recv(1024).decode('utf-8')
        if data == 'q':  # Check for end of sequence
            break
        print(f"Received {data} from Node 1")

        # Simulate packet loss (10% drop chance)
        if random.random() < 0.1:
            print(f"Packet {data} lost (simulated).")
            continue

        # Forward data to Node 3 over UDP
        udp_socket.sendto(data.encode('utf-8'), (udp_ip, udp_port))
        print(f"Forwarded {data} to Node 3 via UDP")

        time.sleep(0.1)  # Small delay to mimic network conditions

    tcp_client.close()
    udp_socket.close()

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

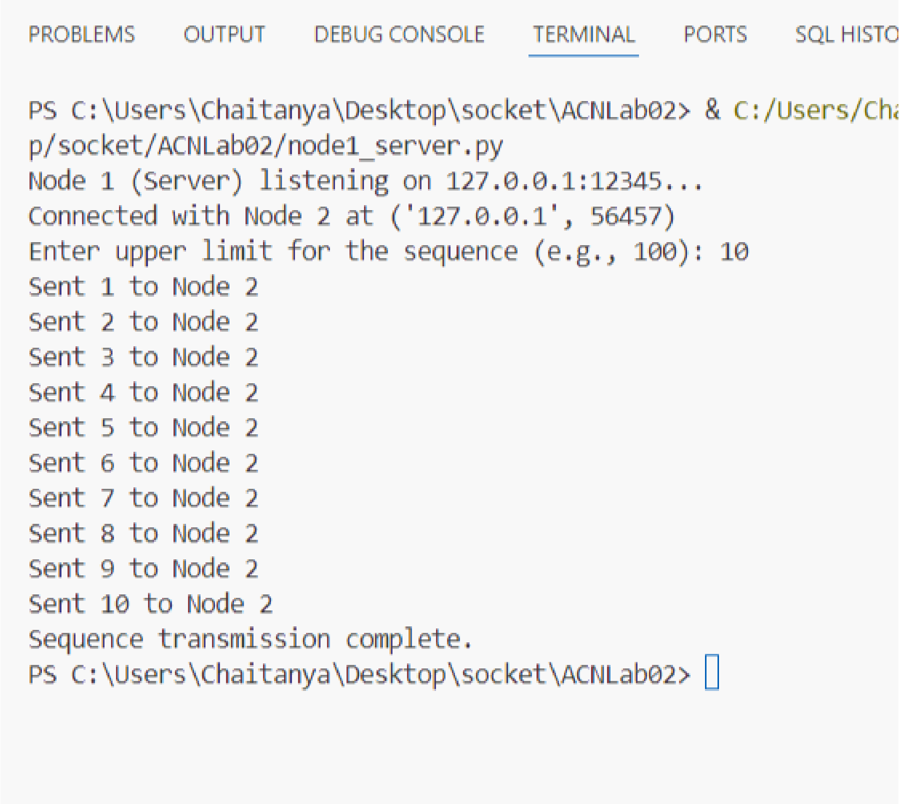
2.3 Node 3 (UDP Receiver with Packet Loss Tracking)

```python

# Node 3 (UDP Receiver with Packet Loss Tracking)
import socket

def main():
    # Set up UDP server for receiving data from Node 2
    udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    udp_ip = '127.0.0.1'
    udp_port = 6000
    udp_socket.bind((udp_ip, udp_port))

    print(f"Node 3 listening for UDP packets on {udp_ip}:{udp_port}...")

    received_count = 0  # Track received packets
    expected_number = 1  # Start with the first expected number

```

```
        while True:
            try:
                # Set a timeout for receiving packets to detect loss
                udp_socket.settimeout(5.0)
                data, addr = udp_socket.recvfrom(1024)
                number = int(data.decode(’utf-8’))
                print(f"Received {number} from Node 2")

                # Increment received count and update expected number
                received_count += 1

                # Detect any missed packets between expected and received
                if number > expected_number:
                    lost_packets = number - expected_number
                    print(f"Lost {lost_packets} packets.")
                else:
                    lost_packets = 0

                expected_number = number + 1  # Update to the next expected number

            except socket.timeout:
                # If timeout occurs, assume the remaining packets are lost
                print(f"No more packets received. Assuming remaining packets lost.")
                break

        print(f"Total received packets: {received_count}")
        print(f"Total lost packets: {expected_number - 1 - received_count}")

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

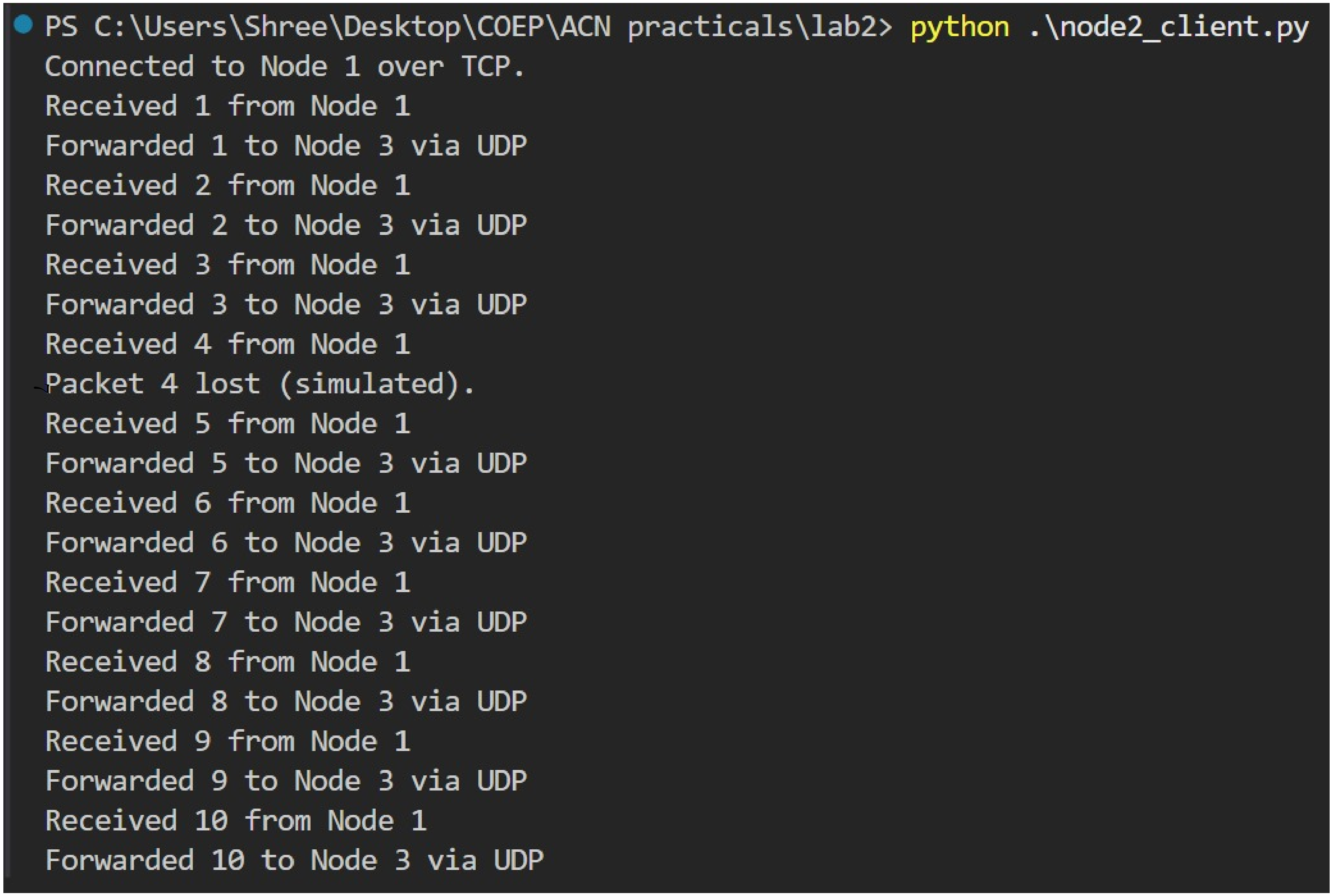
3 Output

3.1 Images of the Program Output



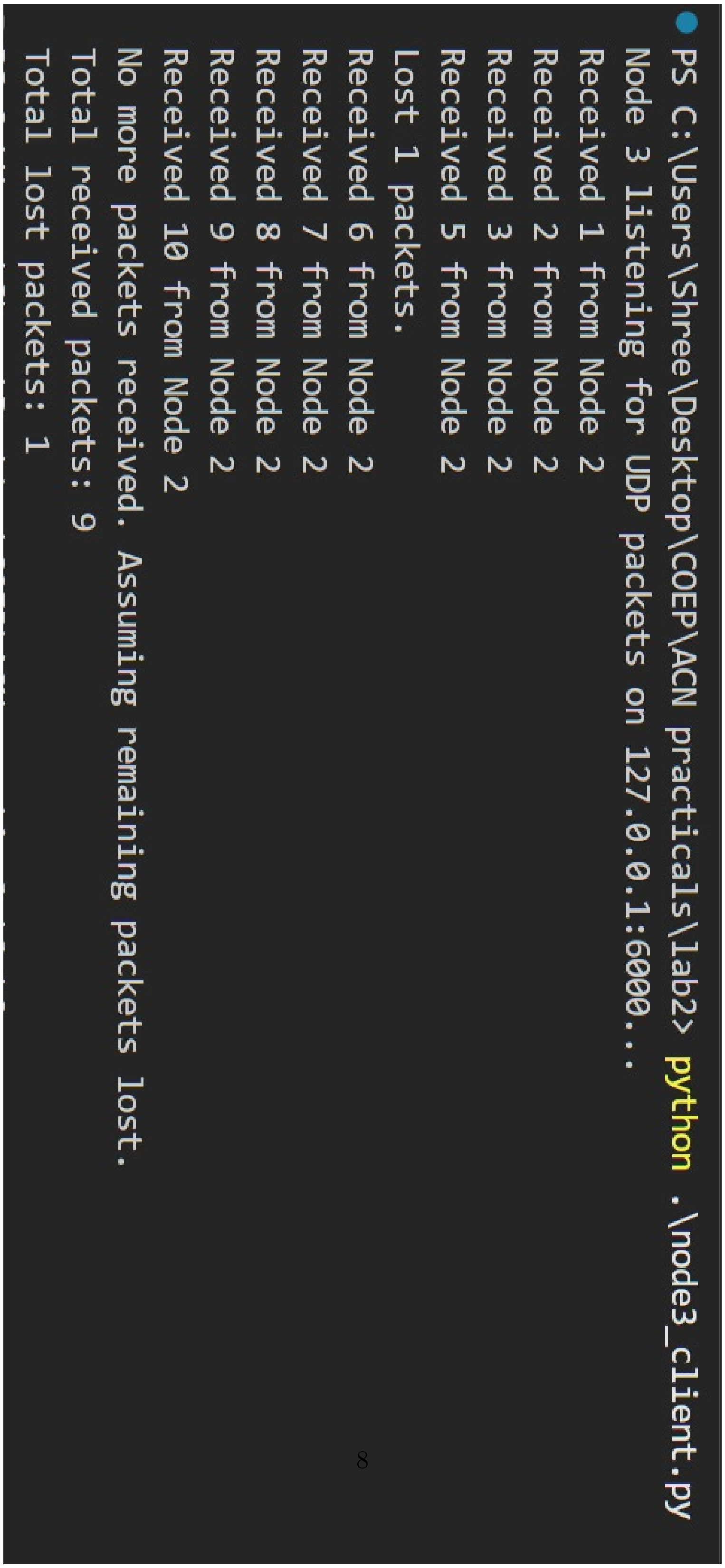
```
PS C:\Users\Chaitanya\Desktop\socket\ACNLab02> & C:/Users/Cha
p/socket/ACNLab02/node1_server.py
Node 1 (Server) listening on 127.0.0.1:12345...
Connected with Node 2 at ('127.0.0.1', 56457)
Enter upper limit for the sequence (e.g., 100): 10
Sent 1 to Node 2
Sent 2 to Node 2
Sent 3 to Node 2
Sent 4 to Node 2
Sent 5 to Node 2
Sent 6 to Node 2
Sent 7 to Node 2
Sent 8 to Node 2
Sent 9 to Node 2
Sent 10 to Node 2
Sequence transmission complete.
PS C:\Users\Chaitanya\Desktop\socket\ACNLab02>
```

Figure 1: Node 1 output



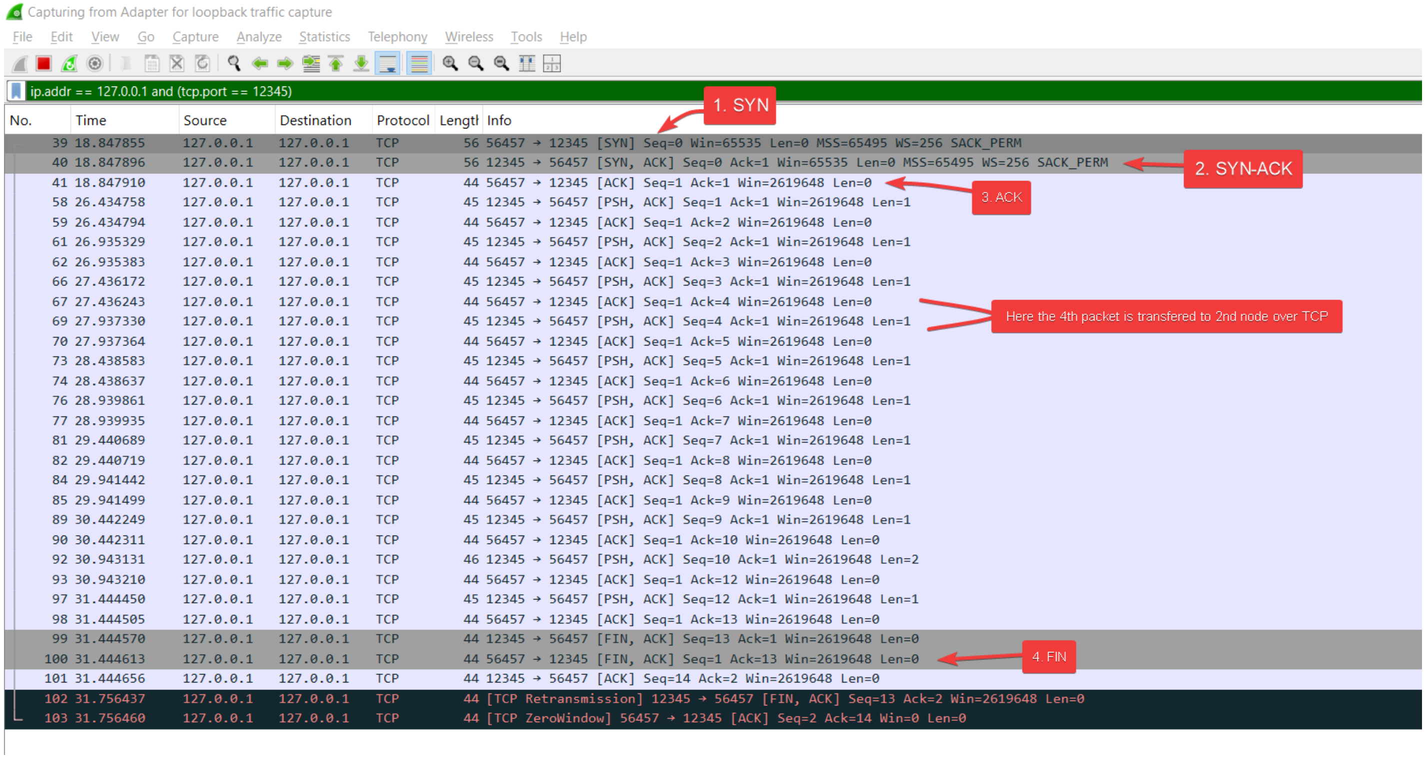
```
PS C:\Users\Shree\Desktop\COEP\ACN practicals\lab2> python .\node2_client.py
Connected to Node 1 over TCP.
Received 1 from Node 1
Forwarded 1 to Node 3 via UDP
Received 2 from Node 1
Forwarded 2 to Node 3 via UDP
Received 3 from Node 1
Forwarded 3 to Node 3 via UDP
Received 4 from Node 1
Packet 4 lost (simulated).
Received 5 from Node 1
Forwarded 5 to Node 3 via UDP
Received 6 from Node 1
Forwarded 6 to Node 3 via UDP
Received 7 from Node 1
Forwarded 7 to Node 3 via UDP
Received 8 from Node 1
Forwarded 8 to Node 3 via UDP
Received 9 from Node 1
Forwarded 9 to Node 3 via UDP
Received 10 from Node 1
Forwarded 10 to Node 3 via UDP
```

Figure 2: Node 2 output



```
PS C:\Users\Shree\Desktop\COEP\ACN practicals\lab2> python .\node3_client.py
Node 3 listening for UDP packets on 127.0.0.1:6000...
Received 1 from Node 2
Received 2 from Node 2
Received 3 from Node 2
Received 5 from Node 2
Lost 1 packets.
Received 6 from Node 2
Received 7 from Node 2
Received 8 from Node 2
Received 9 from Node 2
Received 10 from Node 2
No more packets received. Assuming remaining packets lost.
Total received packets: 9
Total lost packets: 1
```

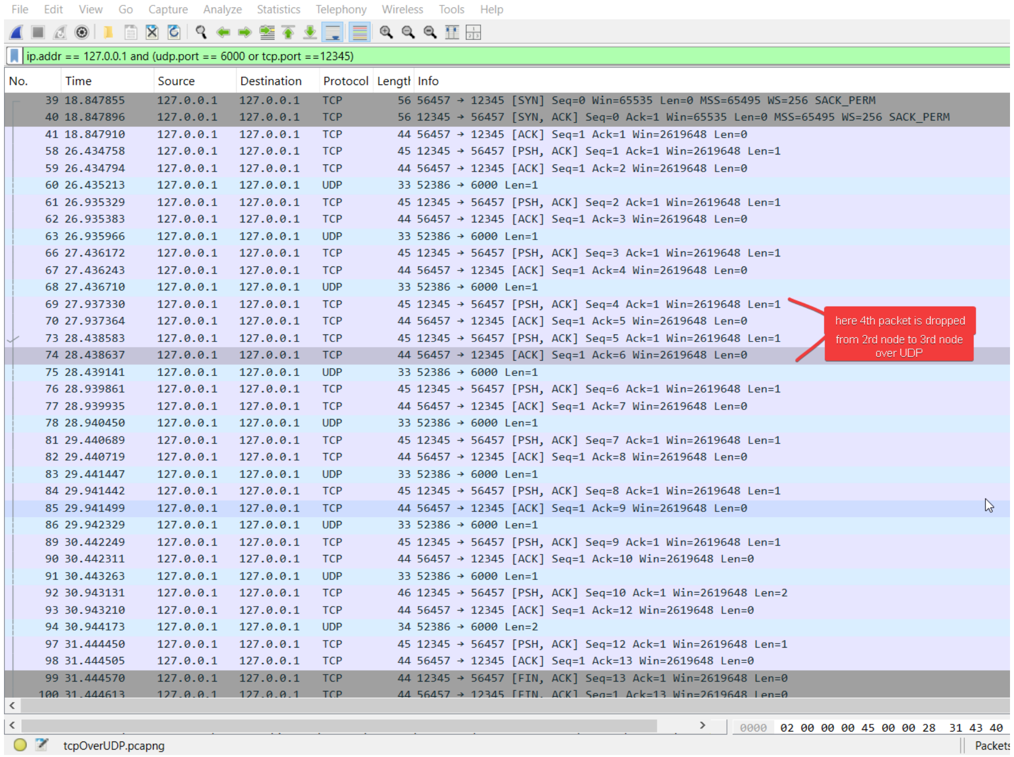
Figure 3: Node 3 output



Capturing from Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
ip.addr == 127.0.0.1 and (tcp.port == 12345)
1. SYN
2. SYN-ACK
3. ACK
Here the 4th packet is transfered to 2nd node over TCP
4. FIN

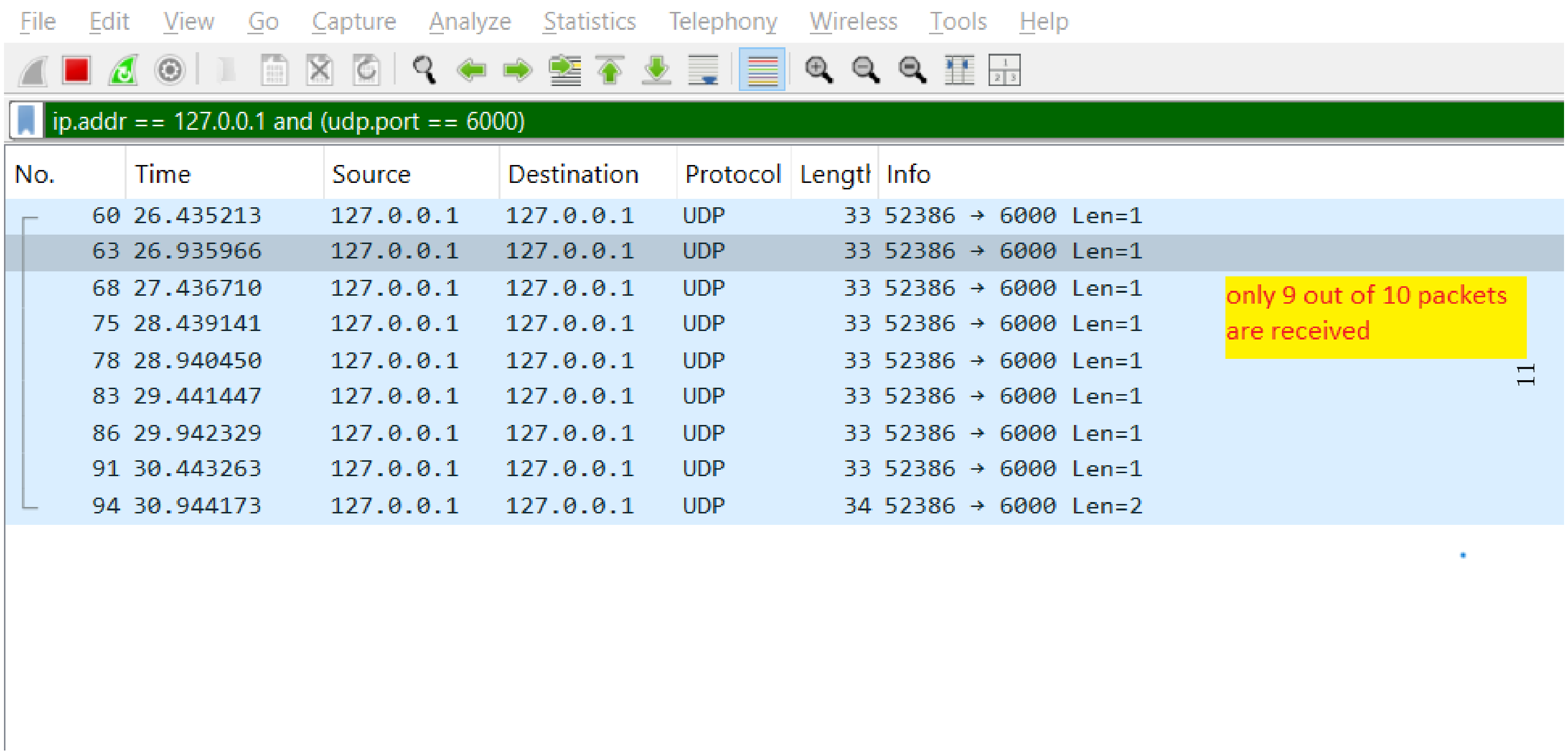
```
No.   Time        Source      Destination  Protocol Lengtl Info
  39 18.847855   127.0.0.1   127.0.0.1    TCP        56 56457 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  40 18.847896   127.0.0.1   127.0.0.1    TCP        56 12345 → 56457 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
  41 18.847910   127.0.0.1   127.0.0.1    TCP        44 56457 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
  58 26.434758   127.0.0.1   127.0.0.1    TCP        45 12345 → 56457 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=1
  59 26.434794   127.0.0.1   127.0.0.1    TCP        44 56457 → 12345 [ACK] Seq=1 Ack=2 Win=2619648 Len=0
  61 26.935329   127.0.0.1   127.0.0.1    TCP        45 12345 → 56457 [PSH, ACK] Seq=2 Ack=1 Win=2619648 Len=1
  62 26.935383   127.0.0.1   127.0.0.1    TCP        44 56457 → 12345 [ACK] Seq=1 Ack=3 Win=2619648 Len=0
  66 27.436172   127.0.0.1   127.0.0.1    TCP        45 12345 → 56457 [PSH, ACK] Seq=3 Ack=1 Win=2619648 Len=1
  67 27.436243   127.0.0.1   127.0.0.1    TCP        44 56457 → 12345 [ACK] Seq=1 Ack=4 Win=2619648 Len=0
  69 27.937330   127.0.0.1   127.0.0.1    TCP        45 12345 → 56457 [PSH, ACK] Seq=4 Ack=1 Win=2619648 Len=1
  70 27.937364   127.0.0.1   127.0.0.1    TCP        44 56457 → 12345 [ACK] Seq=1 Ack=5 Win=2619648 Len=0
  73 28.438583   127.0.0.1   127.0.0.1    TCP        45 12345 → 56457 [PSH, ACK] Seq=5 Ack=1 Win=2619648 Len=1
  74 28.438637   127.0.0.1   127.0.0.1    TCP        44 56457 → 12345 [ACK] Seq=1 Ack=6 Win=2619648 Len=0
  76 28.939861   127.0.0.1   127.0.0.1    TCP        45 12345 → 56457 [PSH, ACK] Seq=6 Ack=1 Win=2619648 Len=1
  77 28.939935   127.0.0.1   127.0.0.1    TCP        44 56457 → 12345 [ACK] Seq=1 Ack=7 Win=2619648 Len=0
  81 29.440689   127.0.0.1   127.0.0.1    TCP        45 12345 → 56457 [PSH, ACK] Seq=7 Ack=1 Win=2619648 Len=1
  82 29.440719   127.0.0.1   127.0.0.1    TCP        44 56457 → 12345 [ACK] Seq=1 Ack=8 Win=2619648 Len=0
  84 29.941442   127.0.0.1   127.0.0.1    TCP        45 12345 → 56457 [PSH, ACK] Seq=8 Ack=1 Win=2619648 Len=1
  85 29.941499   127.0.0.1   127.0.0.1    TCP        44 56457 → 12345 [ACK] Seq=1 Ack=9 Win=2619648 Len=0
  89 30.442249   127.0.0.1   127.0.0.1    TCP        45 12345 → 56457 [PSH, ACK] Seq=9 Ack=1 Win=2619648 Len=1
  90 30.442311   127.0.0.1   127.0.0.1    TCP        44 56457 → 12345 [ACK] Seq=1 Ack=10 Win=2619648 Len=0
  92 30.943131   127.0.0.1   127.0.0.1    TCP        46 12345 → 56457 [PSH, ACK] Seq=10 Ack=1 Win=2619648 Len=2
  93 30.943210   127.0.0.1   127.0.0.1    TCP        44 56457 → 12345 [ACK] Seq=1 Ack=12 Win=2619648 Len=0
  97 31.444450   127.0.0.1   127.0.0.1    TCP        45 12345 → 56457 [PSH, ACK] Seq=12 Ack=1 Win=2619648 Len=1
  98 31.444505   127.0.0.1   127.0.0.1    TCP        44 56457 → 12345 [ACK] Seq=1 Ack=13 Win=2619648 Len=0
  99 31.444570   127.0.0.1   127.0.0.1    TCP        44 12345 → 56457 [FIN, ACK] Seq=13 Ack=1 Win=2619648 Len=0
 100 31.444613   127.0.0.1   127.0.0.1    TCP        44 56457 → 12345 [FIN, ACK] Seq=1 Ack=13 Win=2619648 Len=0
 101 31.444656   127.0.0.1   127.0.0.1    TCP        44 12345 → 56457 [ACK] Seq=14 Ack=2 Win=2619648 Len=0
 102 31.756437   127.0.0.1   127.0.0.1    TCP        44 [TCP Retransmission] 12345 → 56457 [FIN, ACK] Seq=13 Ack=2 Win=2619648 Len=0
 103 31.756460   127.0.0.1   127.0.0.1    TCP        44 [TCP ZeroWindow] 56457 → 12345 [ACK] Seq=2 Ack=14 Win=0 Len=0
```

Figure 4: Wireshark Communication from Node 1 to Node 2



```
ip.addr == 127.0.0.1 and (udp.port == 6000 or tcp.port ==12345)

No.  Time       Source     Destination Protocol Lengtl Info
 39 18.847855  127.0.0.1  127.0.0.1   TCP      56 56457 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 40 18.847896  127.0.0.1  127.0.0.1   TCP      56 12345 → 56457 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
 41 18.847910  127.0.0.1  127.0.0.1   TCP      44 56457 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
 58 26.434758  127.0.0.1  127.0.0.1   TCP      45 12345 → 56457 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=1
 59 26.434794  127.0.0.1  127.0.0.1   TCP      44 56457 → 12345 [ACK] Seq=1 Ack=2 Win=2619648 Len=0
 60 26.435213  127.0.0.1  127.0.0.1   UDP      33 52386 → 6000 Len=1
 61 26.935329  127.0.0.1  127.0.0.1   TCP      45 12345 → 56457 [PSH, ACK] Seq=2 Ack=1 Win=2619648 Len=1
 62 26.935383  127.0.0.1  127.0.0.1   TCP      44 56457 → 12345 [ACK] Seq=1 Ack=3 Win=2619648 Len=0
 63 26.935966  127.0.0.1  127.0.0.1   UDP      33 52386 → 6000 Len=1
 66 27.436172  127.0.0.1  127.0.0.1   TCP      45 12345 → 56457 [PSH, ACK] Seq=3 Ack=1 Win=2619648 Len=1
 67 27.436243  127.0.0.1  127.0.0.1   TCP      44 56457 → 12345 [ACK] Seq=1 Ack=4 Win=2619648 Len=0
 68 27.436710  127.0.0.1  127.0.0.1   UDP      33 52386 → 6000 Len=1
 69 27.937330  127.0.0.1  127.0.0.1   TCP      45 12345 → 56457 [PSH, ACK] Seq=4 Ack=1 Win=2619648 Len=1
 70 27.937364  127.0.0.1  127.0.0.1   TCP      44 56457 → 12345 [ACK] Seq=1 Ack=5 Win=2619648 Len=0
 73 28.438583  127.0.0.1  127.0.0.1   TCP      45 12345 → 56457 [PSH, ACK] Seq=5 Ack=1 Win=2619648 Len=1
 74 28.438637  127.0.0.1  127.0.0.1   TCP      44 56457 → 12345 [ACK] Seq=1 Ack=6 Win=2619648 Len=0
 75 28.439141  127.0.0.1  127.0.0.1   UDP      33 52386 → 6000 Len=1
 76 28.939861  127.0.0.1  127.0.0.1   TCP      45 12345 → 56457 [PSH, ACK] Seq=6 Ack=1 Win=2619648 Len=1
 77 28.939935  127.0.0.1  127.0.0.1   TCP      44 56457 → 12345 [ACK] Seq=1 Ack=7 Win=2619648 Len=0
 78 28.940450  127.0.0.1  127.0.0.1   UDP      33 52386 → 6000 Len=1
 81 29.440689  127.0.0.1  127.0.0.1   TCP      45 12345 → 56457 [PSH, ACK] Seq=7 Ack=1 Win=2619648 Len=1
 82 29.440719  127.0.0.1  127.0.0.1   TCP      44 56457 → 12345 [ACK] Seq=1 Ack=8 Win=2619648 Len=0
 83 29.441447  127.0.0.1  127.0.0.1   UDP      33 52386 → 6000 Len=1
 84 29.941442  127.0.0.1  127.0.0.1   TCP      45 12345 → 56457 [PSH, ACK] Seq=8 Ack=1 Win=2619648 Len=1
 85 29.941499  127.0.0.1  127.0.0.1   TCP      44 56457 → 12345 [ACK] Seq=1 Ack=9 Win=2619648 Len=0
 86 29.942329  127.0.0.1  127.0.0.1   UDP      33 52386 → 6000 Len=1
 89 30.442249  127.0.0.1  127.0.0.1   TCP      45 12345 → 56457 [PSH, ACK] Seq=9 Ack=1 Win=2619648 Len=1
 90 30.442311  127.0.0.1  127.0.0.1   TCP      44 56457 → 12345 [ACK] Seq=1 Ack=10 Win=2619648 Len=0
 91 30.443263  127.0.0.1  127.0.0.1   UDP      33 52386 → 6000 Len=1
 92 30.943131  127.0.0.1  127.0.0.1   TCP      46 12345 → 56457 [PSH, ACK] Seq=10 Ack=1 Win=2619648 Len=2
 93 30.943210  127.0.0.1  127.0.0.1   TCP      44 56457 → 12345 [ACK] Seq=1 Ack=12 Win=2619648 Len=0
 94 30.944173  127.0.0.1  127.0.0.1   UDP      34 52386 → 6000 Len=2
 97 31.444450  127.0.0.1  127.0.0.1   TCP      45 12345 → 56457 [PSH, ACK] Seq=12 Ack=1 Win=2619648 Len=1
 98 31.444505  127.0.0.1  127.0.0.1   TCP      44 56457 → 12345 [ACK] Seq=1 Ack=13 Win=2619648 Len=0
 99 31.444570  127.0.0.1  127.0.0.1   TCP      44 12345 → 56457 [FIN, ACK] Seq=13 Ack=1 Win=2619648 Len=0
100 31.444613  127.0.0.1  127.0.0.1   TCP      44 56457 → 12345 [FIN, ACK] Seq=1 Ack=13 Win=2619648 Len=0

0000  02 00 00 00 45 00 00 28  31 43 40

tcpOverUDP.pcapng                                    Packets
```

Figure 5: Wireshark Combined Communication View



```
ip.addr == 127.0.0.1 and (udp.port == 6000)
```

```
No.    Time         Source      Destination  Protocol  Lengtl  Info
   60  26.435213    127.0.0.1   127.0.0.1    UDP           33  52386 → 6000 Len=1
   63  26.935966    127.0.0.1   127.0.0.1    UDP           33  52386 → 6000 Len=1
   68  27.436710    127.0.0.1   127.0.0.1    UDP           33  52386 → 6000 Len=1
   75  28.439141    127.0.0.1   127.0.0.1    UDP           33  52386 → 6000 Len=1
   78  28.940450    127.0.0.1   127.0.0.1    UDP           33  52386 → 6000 Len=1
   83  29.441447    127.0.0.1   127.0.0.1    UDP           33  52386 → 6000 Len=1
   86  29.942329    127.0.0.1   127.0.0.1    UDP           33  52386 → 6000 Len=1
   91  30.443263    127.0.0.1   127.0.0.1    UDP           33  52386 → 6000 Len=1
   94  30.944173    127.0.0.1   127.0.0.1    UDP           34  52386 → 6000 Len=2
```

only 9 out of 10 packets are received

Figure 6: Wireshark Communication from Node 2 to Node 3