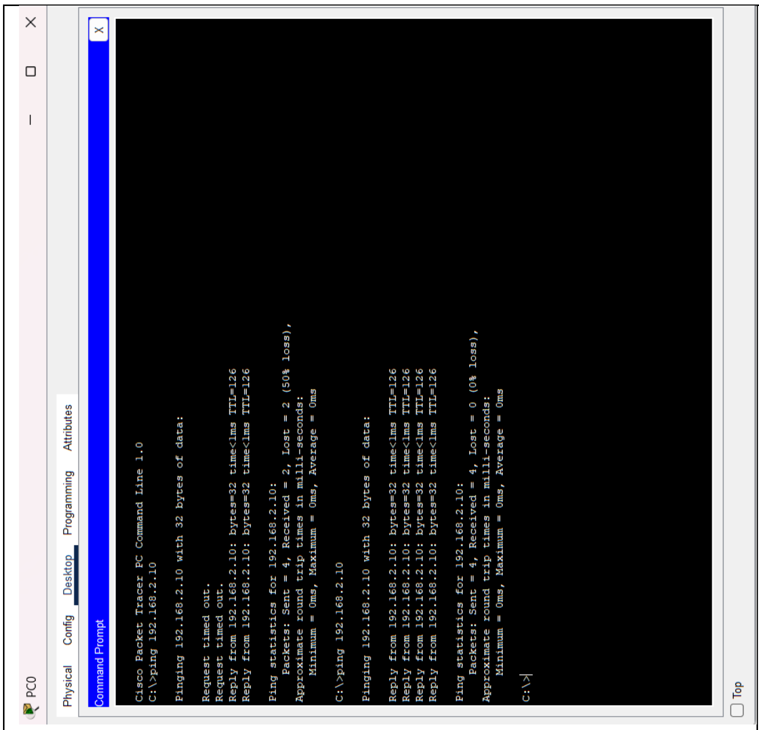
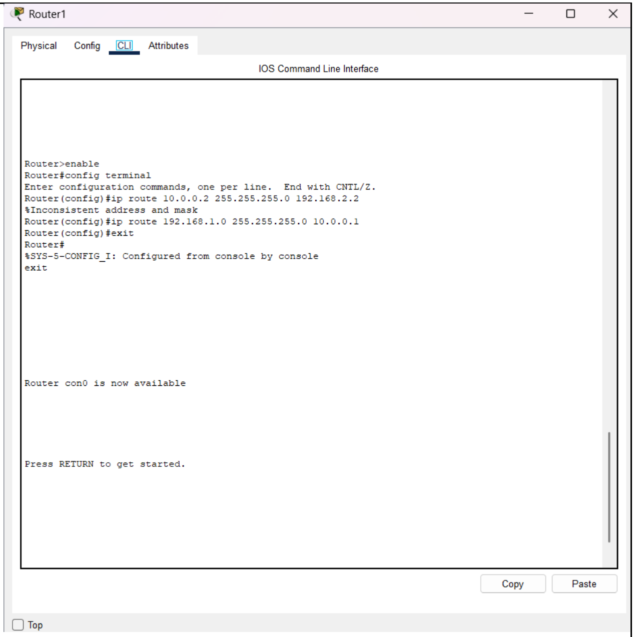


```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 192.168.2.10

Pinging 192.168.2.10 with 32 bytes of data:

Request timed out.
Request timed out.
Reply from 192.168.2.10: bytes=32 time<1ms TTL=126
Reply from 192.168.2.10: bytes=32 time<1ms TTL=126

Ping statistics for 192.168.2.10:
    Packets: Sent = 4, Received = 2, Lost = 2 (50% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\>ping 192.168.2.10

Pinging 192.168.2.10 with 32 bytes of data:

Reply from 192.168.2.10: bytes=32 time<1ms TTL=126
Reply from 192.168.2.10: bytes=32 time<1ms TTL=126
Reply from 192.168.2.10: bytes=32 time<1ms TTL=126
Reply from 192.168.2.10: bytes=32 time<1ms TTL=126

Ping statistics for 192.168.2.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\>
```

Figure 1: asgn8



```
Router>enable
Router#config terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip route 10.0.0.2 255.255.255.0 192.168.2.2
%Inconsistent address and mask
Router(config)#ip route 192.168.1.0 255.255.255.0 10.0.0.1
Router(config)#exit
Router#
%SYS-5-CONFIG_I: Configured from console by console
exit







Router con0 is now available






Press RETURN to get started.
```

Figure 2: asgn 8

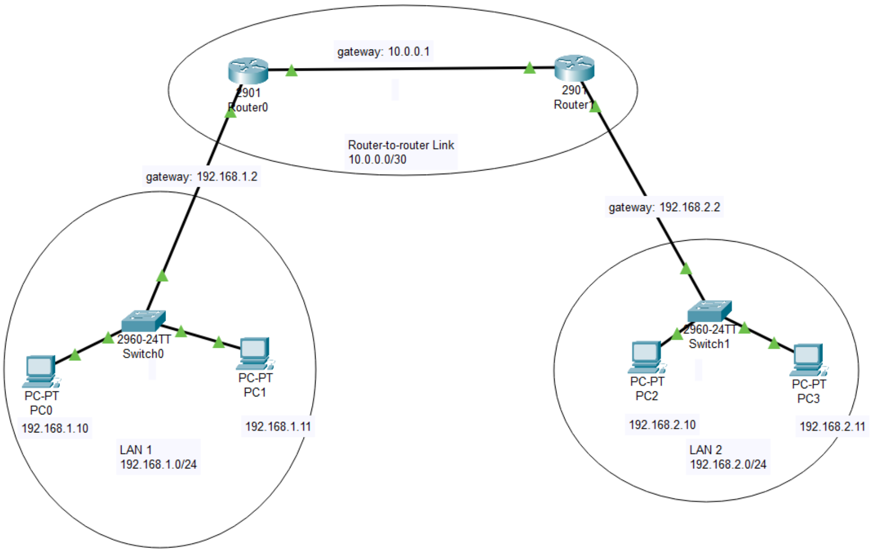


Figure 3: asgn 8

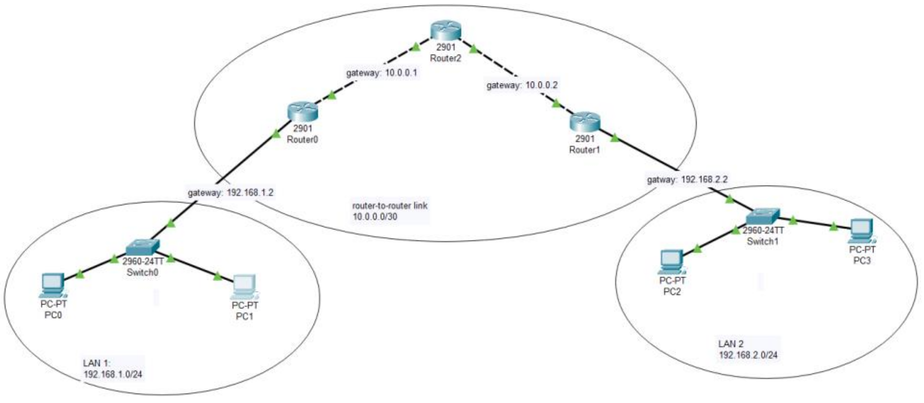
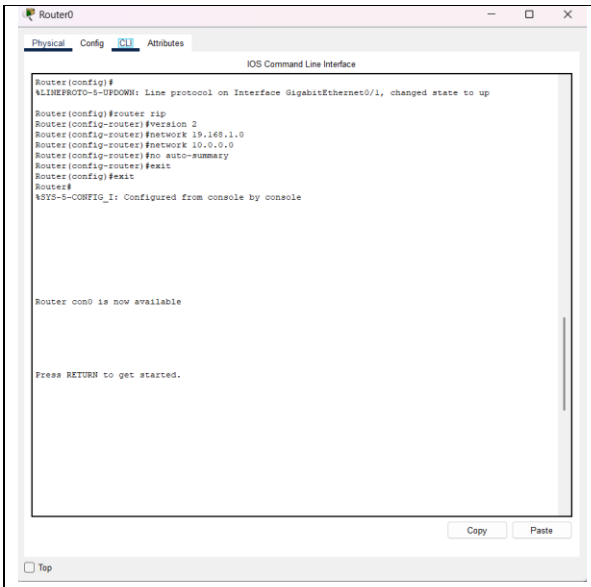
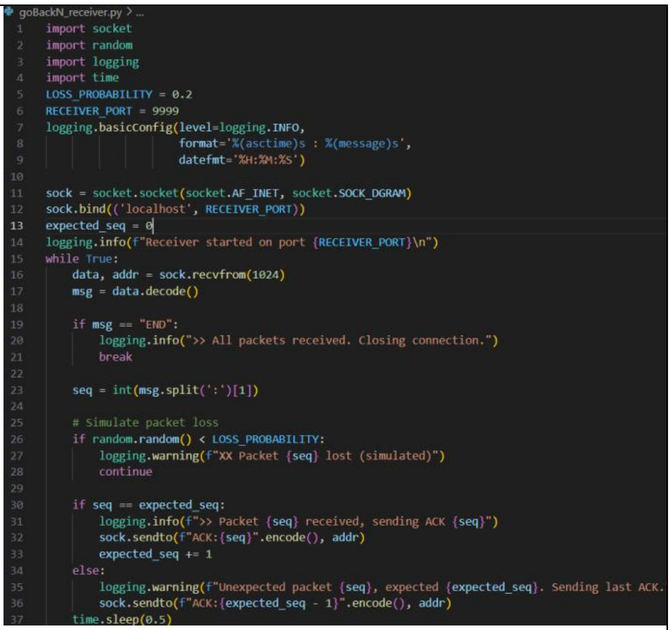


Figure 4: asgn 9



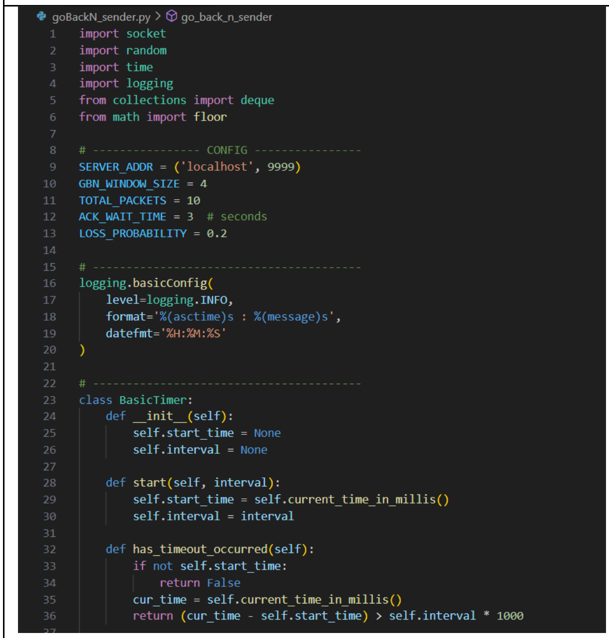
```
Router(config)#
%LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/1, changed state to up

Router(config)#router rip
Router(config-router)#version 2
Router(config-router)#network 19.168.1.0
Router(config-router)#network 10.0.0.0
Router(config-router)#no auto-summary
Router(config-router)#exit
Router(config)#exit
Router#
%SYS-5-CONFIG_I: Configured from console by console




Router con0 is now available



Press RETURN to get started.
```

Figure 5: asgn 9



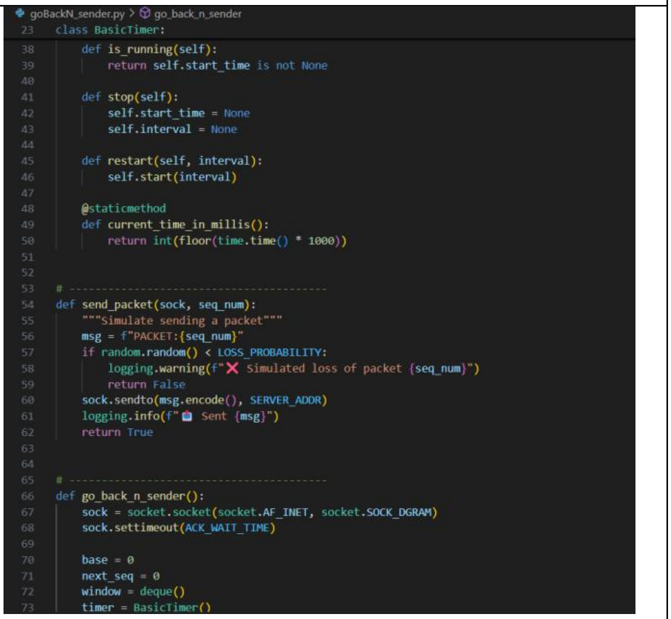
```python
import socket
import random
import logging
import time
LOSS_PROBABILITY = 0.2
RECEIVER_PORT = 9999
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s : %(message)s',
                    datefmt='%H:%M:%S')

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(('localhost', RECEIVER_PORT))
expected_seq = 0
logging.info(f"Receiver started on port {RECEIVER_PORT}\n")
while True:
    data, addr = sock.recvfrom(1024)
    msg = data.decode()

    if msg == "END":
        logging.info(">> All packets received. Closing connection.")
        break

    seq = int(msg.split(':')[1])

    # Simulate packet loss
    if random.random() < LOSS_PROBABILITY:
        logging.warning(f"XX Packet {seq} lost (simulated)")
        continue

    if seq == expected_seq:
        logging.info(f">> Packet {seq} received, sending ACK {seq}")
        sock.sendto(f"ACK:{seq}".encode(), addr)
        expected_seq += 1
    else:
        logging.warning(f"Unexpected packet {seq}, expected {expected_seq}. Sending last ACK.
        sock.sendto(f"ACK:{expected_seq - 1}".encode(), addr)
    time.sleep(0.5)
```

Figure 6: asgn 7 gbn rec



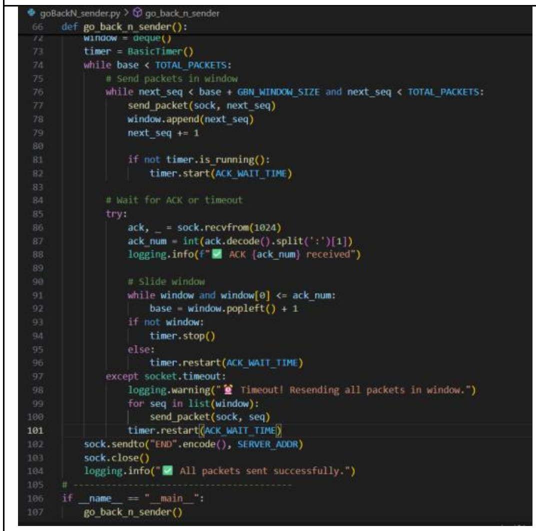
```python
import socket
import random
import time
import logging
from collections import deque
from math import floor

# --------------- CONFIG ---------------
SERVER_ADDR = ('localhost', 9999)
GBN_WINDOW_SIZE = 4
TOTAL_PACKETS = 10
ACK_WAIT_TIME = 3  # seconds
LOSS_PROBABILITY = 0.2

# --------------------------------------
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s : %(message)s',
    datefmt='%H:%M:%S'
)

# --------------------------------------
class BasicTimer:
    def __init__(self):
        self.start_time = None
        self.interval = None

    def start(self, interval):
        self.start_time = self.current_time_in_millis()
        self.interval = interval

    def has_timeout_occurred(self):
        if not self.start_time:
            return False
        cur_time = self.current_time_in_millis()
        return (cur_time - self.start_time) > self.interval * 1000
```

Figure 7: asgn 7 gbn1 sen



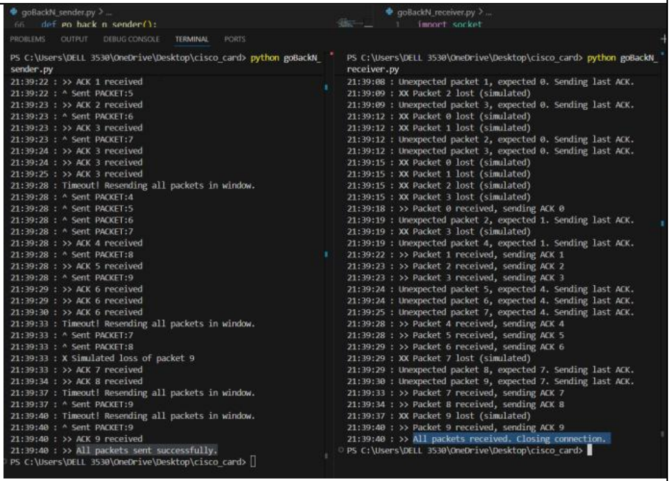
```python
class BasicTimer:
    def is_running(self):
        return self.start_time is not None

    def stop(self):
        self.start_time = None
        self.interval = None

    def restart(self, interval):
        self.start(interval)

    @staticmethod
    def current_time_in_millis():
        return int(floor(time.time() * 1000))


# --------------------------------------
def send_packet(sock, seq_num):
    """Simulate sending a packet"""
    msg = f"PACKET:{seq_num}"
    if random.random() < LOSS_PROBABILITY:
        logging.warning(f"❌ Simulated loss of packet {seq_num}")
        return False
    sock.sendto(msg.encode(), SERVER_ADDR)
    logging.info(f"📤 Sent {msg}")
    return True


# --------------------------------------
def go_back_n_sender():
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.settimeout(ACK_WAIT_TIME)

    base = 0
    next_seq = 0
    window = deque()
    timer = BasicTimer()
```

Figure 8: asgn7 gbn2 sen



```python
def go_back_n_sender():
    window = deque()
    timer = BasicTimer()
    while base < TOTAL_PACKETS:
        # Send packets in window
        while next_seq < base + GBN_WINDOW_SIZE and next_seq < TOTAL_PACKETS:
            send_packet(sock, next_seq)
            window.append(next_seq)
            next_seq += 1

            if not timer.is_running():
                timer.start(ACK_WAIT_TIME)

        # Wait for ACK or timeout
        try:
            ack, _ = sock.recvfrom(1024)
            ack_num = int(ack.decode().split(':')[1])
            logging.info(f"✅ ACK {ack_num} received")

            # Slide window
            while window and window[0] <= ack_num:
                base = window.popleft() + 1
            if not window:
                timer.stop()
            else:
                timer.restart(ACK_WAIT_TIME)
        except socket.timeout:
            logging.warning("⏰ Timeout! Resending all packets in window.")
            for seq in list(window):
                send_packet(sock, seq)
            timer.restart(ACK_WAIT_TIME)
    sock.sendto("END".encode(), SERVER_ADDR)
    sock.close()
    logging.info("✅ All packets sent successfully.")
# --------------------------------------
if __name__ == "__main__":
    go_back_n_sender()
```

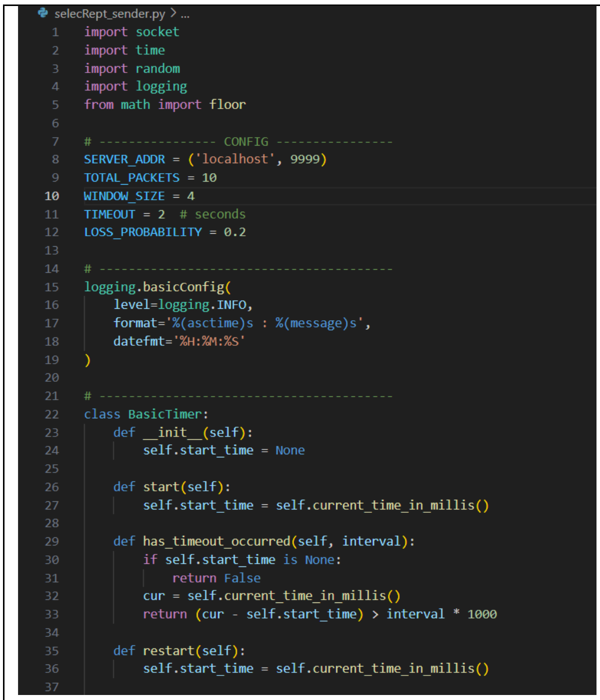
Figure 9: asgn 7 gbn3 sen



```
PS C:\Users\DELL 3530\OneDrive\Desktop\cisco_card> python goBackN_
sender.py
21:39:22 : >> ACK 1 received
21:39:22 : ^ Sent PACKET:5
21:39:23 : >> ACK 2 received
21:39:23 : ^ Sent PACKET:6
21:39:23 : >> ACK 3 received
21:39:23 : ^ Sent PACKET:7
21:39:24 : >> ACK 3 received
21:39:24 : >> ACK 3 received
21:39:25 : >> ACK 3 received
21:39:28 : Timeout! Resending all packets in window.
21:39:28 : ^ Sent PACKET:4
21:39:28 : ^ Sent PACKET:5
21:39:28 : ^ Sent PACKET:6
21:39:28 : ^ Sent PACKET:7
21:39:28 : >> ACK 4 received
21:39:28 : ^ Sent PACKET:8
21:39:28 : >> ACK 5 received
21:39:28 : ^ Sent PACKET:9
21:39:29 : >> ACK 6 received
21:39:29 : >> ACK 6 received
21:39:30 : >> ACK 6 received
21:39:33 : Timeout! Resending all packets in window.
21:39:33 : ^ Sent PACKET:7
21:39:33 : ^ Sent PACKET:8
21:39:33 : X Simulated loss of packet 9
21:39:33 : >> ACK 7 received
21:39:34 : >> ACK 8 received
21:39:37 : Timeout! Resending all packets in window.
21:39:37 : ^ Sent PACKET:9
21:39:40 : Timeout! Resending all packets in window.
21:39:40 : ^ Sent PACKET:9
21:39:40 : >> ACK 9 received
21:39:40 : >> All packets sent successfully.
PS C:\Users\DELL 3530\OneDrive\Desktop\cisco_card> 
```

```
PS C:\Users\DELL 3530\OneDrive\Desktop\cisco_card> python goBackN_
receiver.py
21:39:08 : Unexpected packet 1, expected 0. Sending last ACK.
21:39:09 : XX Packet 2 lost (simulated)
21:39:09 : Unexpected packet 3, expected 0. Sending last ACK.
21:39:12 : XX Packet 0 lost (simulated)
21:39:12 : XX Packet 1 lost (simulated)
21:39:12 : Unexpected packet 2, expected 0. Sending last ACK.
21:39:12 : Unexpected packet 3, expected 0. Sending last ACK.
21:39:15 : XX Packet 0 lost (simulated)
21:39:15 : XX Packet 1 lost (simulated)
21:39:15 : XX Packet 2 lost (simulated)
21:39:15 : XX Packet 3 lost (simulated)
21:39:18 : >> Packet 0 received, sending ACK 0
21:39:19 : Unexpected packet 2, expected 1. Sending last ACK.
21:39:19 : XX Packet 3 lost (simulated)
21:39:19 : Unexpected packet 4, expected 1. Sending last ACK.
21:39:22 : >> Packet 1 received, sending ACK 1
21:39:23 : >> Packet 2 received, sending ACK 2
21:39:23 : >> Packet 3 received, sending ACK 3
21:39:24 : Unexpected packet 5, expected 4. Sending last ACK.
21:39:24 : Unexpected packet 6, expected 4. Sending last ACK.
21:39:25 : Unexpected packet 7, expected 4. Sending last ACK.
21:39:28 : >> Packet 4 received, sending ACK 4
21:39:28 : >> Packet 5 received, sending ACK 5
21:39:29 : >> Packet 6 received, sending ACK 6
21:39:29 : XX Packet 7 lost (simulated)
21:39:29 : Unexpected packet 8, expected 7. Sending last ACK.
21:39:30 : Unexpected packet 9, expected 7. Sending last ACK.
21:39:33 : >> Packet 7 received, sending ACK 7
21:39:34 : >> Packet 8 received, sending ACK 8
21:39:37 : XX Packet 9 lost (simulated)
21:39:40 : >> Packet 9 received, sending ACK 9
21:39:40 : >> All packets received. Closing connection.
PS C:\Users\DELL 3530\OneDrive\Desktop\cisco_card> 
```

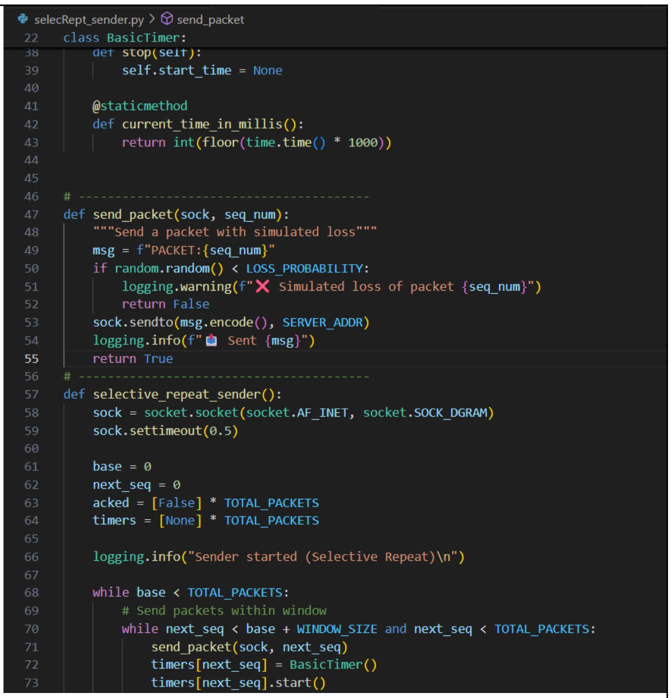
Figure 10: asgn gbn op



selecRept_sender.py > ...

```python
import socket
import time
import random
import logging
from math import floor

# --------------- CONFIG ---------------
SERVER_ADDR = ('localhost', 9999)
TOTAL_PACKETS = 10
WINDOW_SIZE = 4
TIMEOUT = 2  # seconds
LOSS_PROBABILITY = 0.2

# ----------------------------------------
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s : %(message)s',
    datefmt='%H:%M:%S'
)

# ----------------------------------------
class BasicTimer:
    def __init__(self):
        self.start_time = None

    def start(self):
        self.start_time = self.current_time_in_millis()

    def has_timeout_occurred(self, interval):
        if self.start_time is None:
            return False
        cur = self.current_time_in_millis()
        return (cur - self.start_time) > interval * 1000

    def restart(self):
        self.start_time = self.current_time_in_millis()

```

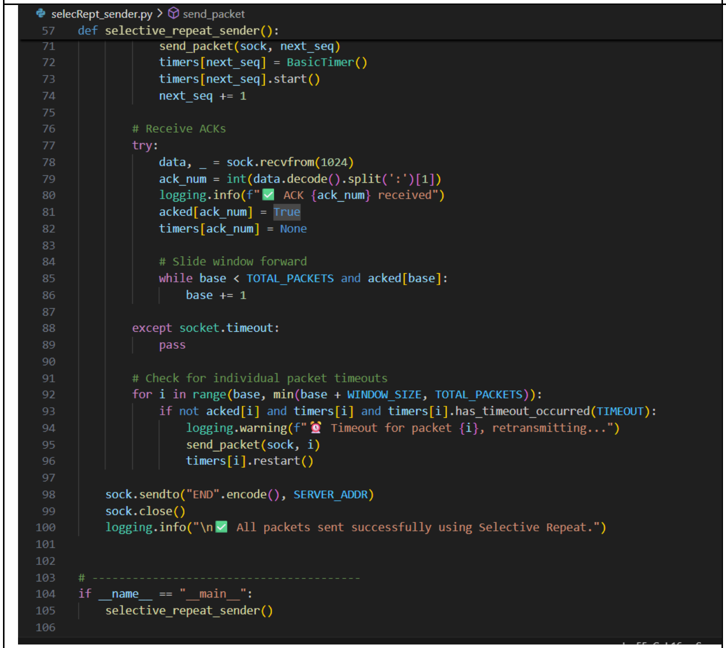
Figure 11: asgn 7 SR1 send



selecRept_sender.py > send_packet

```python
class BasicTimer:
    def stop(self):
        self.start_time = None

    @staticmethod
    def current_time_in_millis():
        return int(floor(time.time() * 1000))


# ----------------------------------------
def send_packet(sock, seq_num):
    """Send a packet with simulated loss"""
    msg = f"PACKET:{seq_num}"
    if random.random() < LOSS_PROBABILITY:
        logging.warning(f"❌ Simulated loss of packet {seq_num}")
        return False
    sock.sendto(msg.encode(), SERVER_ADDR)
    logging.info(f"📤 Sent {msg}")
    return True
# ----------------------------------------
def selective_repeat_sender():
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.settimeout(0.5)

    base = 0
    next_seq = 0
    acked = [False] * TOTAL_PACKETS
    timers = [None] * TOTAL_PACKETS

    logging.info("Sender started (Selective Repeat)\n")

    while base < TOTAL_PACKETS:
        # Send packets within window
        while next_seq < base + WINDOW_SIZE and next_seq < TOTAL_PACKETS:
            send_packet(sock, next_seq)
            timers[next_seq] = BasicTimer()
            timers[next_seq].start()
```

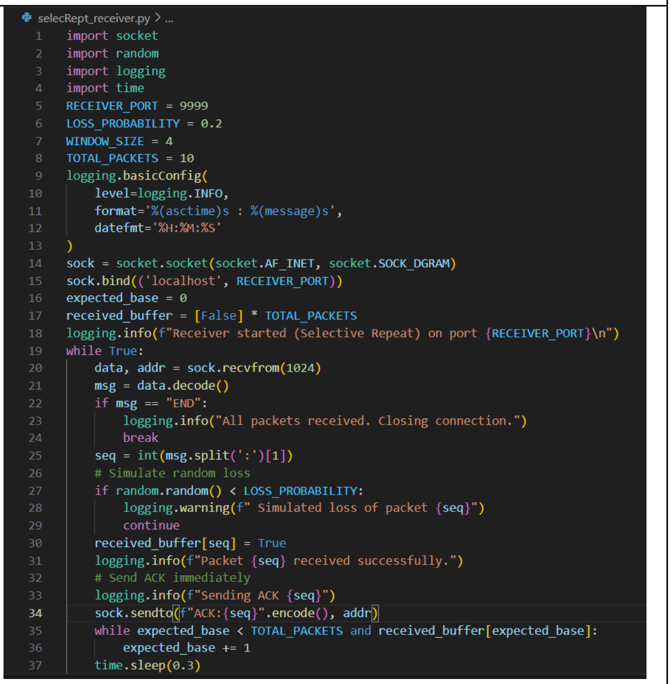
Figure 12: asgn 7 sr2 send



selecRept_sender.py > send_packet

```python
def selective_repeat_sender():
            send_packet(sock, next_seq)
            timers[next_seq] = BasicTimer()
            timers[next_seq].start()
            next_seq += 1

        # Receive ACKs
        try:
            data, _ = sock.recvfrom(1024)
            ack_num = int(data.decode().split(':')[1])
            logging.info(f"✅ ACK {ack_num} received")
            acked[ack_num] = True
            timers[ack_num] = None

            # Slide window forward
            while base < TOTAL_PACKETS and acked[base]:
                base += 1

        except socket.timeout:
            pass

        # Check for individual packet timeouts
        for i in range(base, min(base + WINDOW_SIZE, TOTAL_PACKETS)):
            if not acked[i] and timers[i] and timers[i].has_timeout_occurred(TIMEOUT):
                logging.warning(f"⏰ Timeout for packet {i}, retransmitting...")
                send_packet(sock, i)
                timers[i].restart()

    sock.sendto("END".encode(), SERVER_ADDR)
    sock.close()
    logging.info("\n✅ All packets sent successfully using Selective Repeat.")


# ----------------------------------------
if __name__ == "__main__":
    selective_repeat_sender()

```

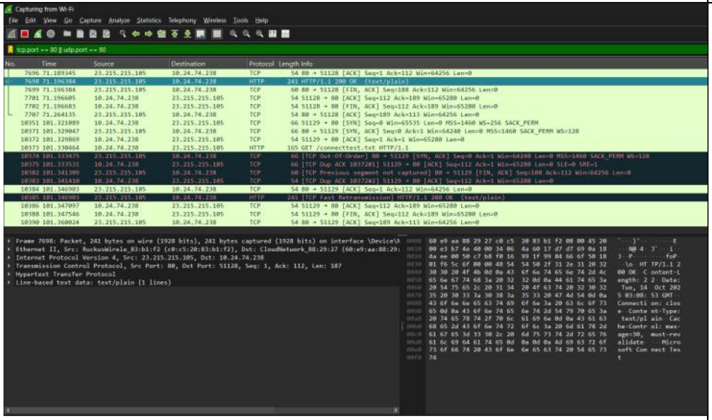
Figure 13: asgn 7 sr3 send



selecRept_receiver.py > ...

```python
import socket
import random
import logging
import time
RECEIVER_PORT = 9999
LOSS_PROBABILITY = 0.2
WINDOW_SIZE = 4
TOTAL_PACKETS = 10
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s : %(message)s',
    datefmt='%H:%M:%S'
)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(('localhost', RECEIVER_PORT))
expected_base = 0
received_buffer = [False] * TOTAL_PACKETS
logging.info(f"Receiver started (Selective Repeat) on port {RECEIVER_PORT}\n")
while True:
    data, addr = sock.recvfrom(1024)
    msg = data.decode()
    if msg == "END":
        logging.info("All packets received. Closing connection.")
        break
    seq = int(msg.split(':')[1])
    # Simulate random loss
    if random.random() < LOSS_PROBABILITY:
        logging.warning(f" Simulated loss of packet {seq}")
        continue
    received_buffer[seq] = True
    logging.info(f"Packet {seq} received successfully.")
    # Send ACK immediately
    logging.info(f"Sending ACK {seq}")
    sock.sendto(f"ACK:{seq}".encode(), addr)
    while expected_base < TOTAL_PACKETS and received_buffer[expected_base]:
        expected_base += 1
    time.sleep(0.3)
```

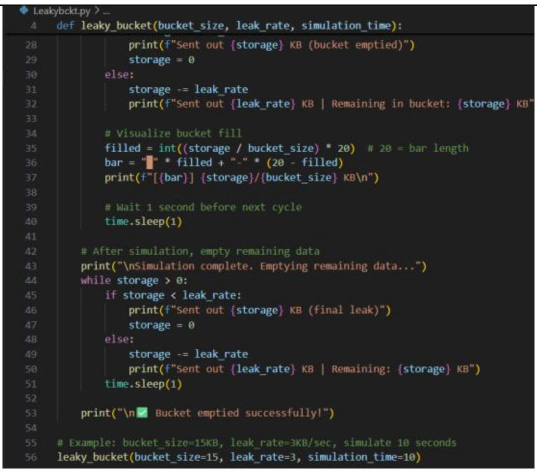
Figure 14: asgn 7 sr rec



Capturing from Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp.port == 80 || udp.port == 80
No.   Time       Source          Destination     Protocol Length Info
 7696 71.189345  23.215.215.105  10.24.74.238    TCP      54 80 → 51128 [ACK] Seq=1 Ack=112 Win=64256 Len=0
 7698 71.196384  23.215.215.105  10.24.74.238    HTTP     241 HTTP/1.1 200 OK  (text/plain)
 7699 71.196384  23.215.215.105  10.24.74.238    TCP      60 80 → 51128 [FIN, ACK] Seq=188 Ack=112 Win=64256 Len=0
 7701 71.196605  10.24.74.238    23.215.215.105  TCP      54 51128 → 80 [ACK] Seq=112 Ack=189 Win=65280 Len=0
 7702 71.196683  10.24.74.238    23.215.215.105  TCP      54 51128 → 80 [FIN, ACK] Seq=112 Ack=189 Win=65280 Len=0
 7707 71.264135  23.215.215.105  10.24.74.238    TCP      54 80 → 51128 [ACK] Seq=189 Ack=113 Win=64256 Len=0
10351 101.321089 10.24.74.238    23.215.215.105  TCP      66 51129 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
10371 101.329047 23.215.215.105  10.24.74.238    TCP      66 80 → 51129 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
10372 101.329869 10.24.74.238    23.215.215.105  TCP      54 51129 → 80 [ACK] Seq=1 Ack=1 Win=65280 Len=0
10373 101.330464 10.24.74.238    23.215.215.105  HTTP     165 GET /connecttest.txt HTTP/1.1
10374 101.333475 23.215.215.105  10.24.74.238    TCP      66 [TCP Out-Of-Order] 80 → 51129 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
10375 101.333531 10.24.74.238    23.215.215.105  TCP      66 [TCP Dup ACK 10372#1] 51129 → 80 [ACK] Seq=112 Ack=1 Win=65280 Len=0 SLE=0 SRE=1
10382 101.341309 23.215.215.105  10.24.74.238    TCP      60 [TCP Previous segment not captured] 80 → 51129 [FIN, ACK] Seq=188 Ack=112 Win=64256 Len=0
10383 101.341410 10.24.74.238    23.215.215.105  TCP      54 [TCP Dup ACK 10372#2] 51129 → 80 [ACK] Seq=112 Ack=1 Win=65280 Len=0
10384 101.346903 23.215.215.105  10.24.74.238    TCP      54 80 → 51129 [ACK] Seq=1 Ack=112 Win=64256 Len=0
10385 101.346903 23.215.215.105  10.24.74.238    HTTP     241 [TCP Fast Retransmission] HTTP/1.1 200 OK  (text/plain)
10386 101.347097 10.24.74.238    23.215.215.105  TCP      54 51129 → 80 [ACK] Seq=112 Ack=189 Win=65280 Len=0
10388 101.347546 10.24.74.238    23.215.215.105  TCP      54 51129 → 80 [FIN, ACK] Seq=112 Ack=189 Win=65280 Len=0
10390 101.360024 23.215.215.105  10.24.74.238    TCP      54 80 → 51129 [ACK] Seq=189 Ack=113 Win=64256 Len=0

› Frame 7698: Packet, 241 bytes on wire (1928 bits), 241 bytes captured (1928 bits) on interface \Device\N
› Ethernet II, Src: RuckusWirele_83:b1:f2 (c0:c5:20:83:b1:f2), Dst: CloudNetwork_88:29:27 (60:e9:aa:88:29:
› Internet Protocol Version 4, Src: 23.215.215.105, Dst: 10.24.74.238
› Transmission Control Protocol, Src Port: 80, Dst Port: 51128, Seq: 1, Ack: 112, Len: 187
› Hypertext Transfer Protocol
› Line-based text data: text/plain (1 lines)

0000  60 e9 aa 88 29 27 c0 c5  20 83 b1 f2 08 00 45 20   `···)'·· ······E
0010  00 e3 b7 4e 40 00 34 06  4a 60 17 d7 d7 69 0a 18   ···N@·4· J`···i··
0020  4a ee 00 50 c7 b8 f0 16  99 1f 99 84 66 6f 50 18   J··P···· ····foP·
0030  01 f6 5c 6f 00 00 48 54  54 50 2f 31 2e 31 20 32   ··\o··HT TP/1.1 2
0040  30 30 20 4f 4b 0d 0a 43  6f 6e 74 65 6e 74 2d 4c   00 OK··C ontent-L
0050  65 6e 67 74 68 3a 20 32  32 0d 0a 44 61 74 65 3a   ength: 2 2··Date:
0060  20 54 75 65 2c 20 31 34  20 4f 63 74 20 32 30 32    Tue, 14  Oct 202
0070  35 20 30 33 3a 30 38 3a  35 33 20 47 4d 54 0d 0a   5 03:08: 53 GMT··
0080  43 6f 6e 6e 65 63 74 69  6f 6e 3a 20 63 6c 6f 73   Connecti on: clos
0090  65 0d 0a 43 6f 6e 74 65  6e 74 2d 54 79 70 65 3a   e··Conte nt-Type:
00a0  20 74 65 78 74 2f 70 6c  61 69 6e 0d 0a 43 61 63    text/pl ain··Cac
00b0  68 65 2d 43 6f 6e 74 72  6f 6c 3a 20 6d 61 78 2d   he-Contr ol: max-
00c0  61 67 65 3d 33 30 2c 20  6d 75 73 74 2d 72 65 76   age=30,  must-rev
00d0  61 6c 69 64 61 74 65 0d  0a 0d 0a 4d 69 63 72 6f   alidate· ···Micro
00e0  73 6f 66 74 20 43 6f 6e  6e 65 63 74 20 54 65 73   soft Con nect Tes
00f0  74                                                 t
```

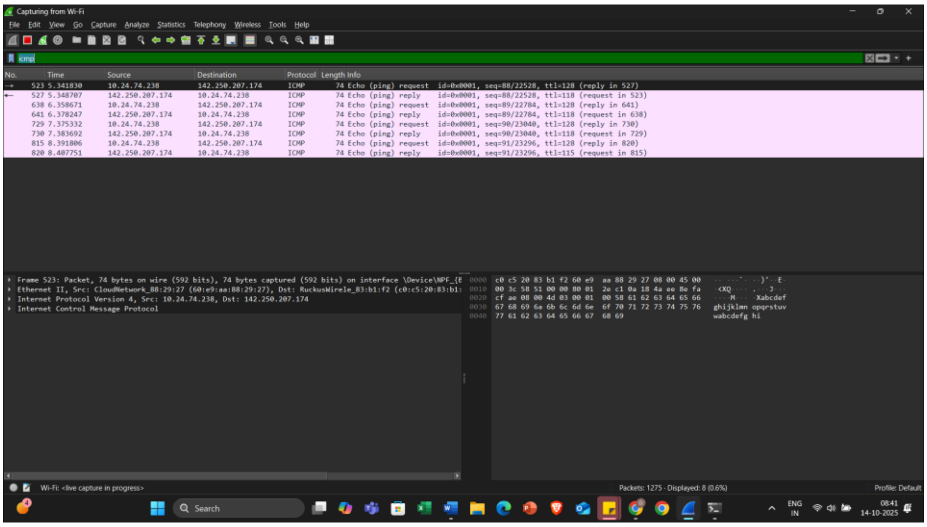
Figure 15: asgn 10



Leakybckt.py > ...

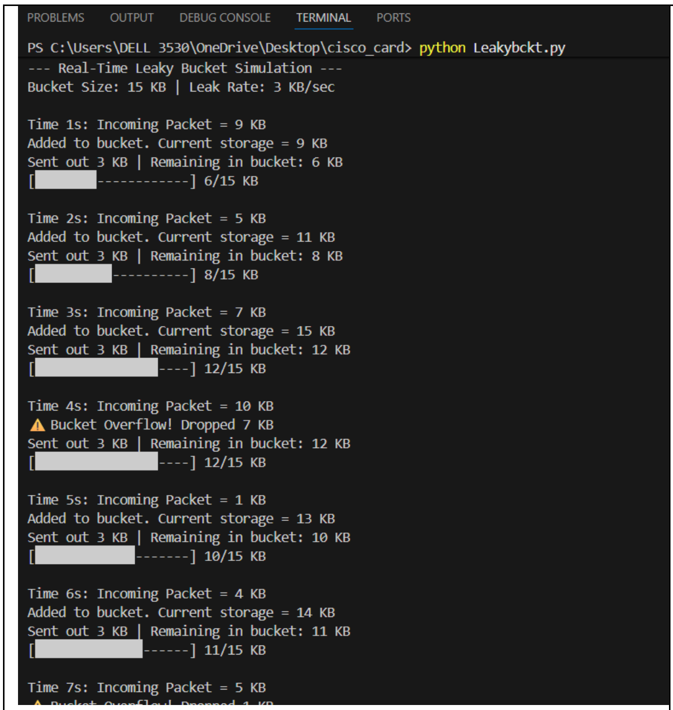
```python
def leaky_bucket(bucket_size, leak_rate, simulation_time):
                print(f"Sent out {storage} KB (bucket emptied)")
                storage = 0
            else:
                storage -= leak_rate
                print(f"Sent out {leak_rate} KB | Remaining in bucket: {storage} KB"

            # Visualize bucket fill
            filled = int((storage / bucket_size) * 20)  # 20 = bar length
            bar = "█" * filled + "-" * (20 - filled)
            print(f"[{bar}] {storage}/{bucket_size} KB\n")

            # Wait 1 second before next cycle
            time.sleep(1)

        # After simulation, empty remaining data
        print("\nSimulation complete. Emptying remaining data...")
        while storage > 0:
            if storage < leak_rate:
                print(f"Sent out {storage} KB (final leak)")
                storage = 0
            else:
                storage -= leak_rate
                print(f"Sent out {leak_rate} KB | Remaining: {storage} KB")
            time.sleep(1)

        print("\n✅ Bucket emptied successfully!")

# Example: bucket_size=15KB, leak_rate=3KB/sec, simulate 10 seconds
leaky_bucket(bucket_size=15, leak_rate=3, simulation_time=10)
```

Figure 16: asgn 11 ip2


Capturing from Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

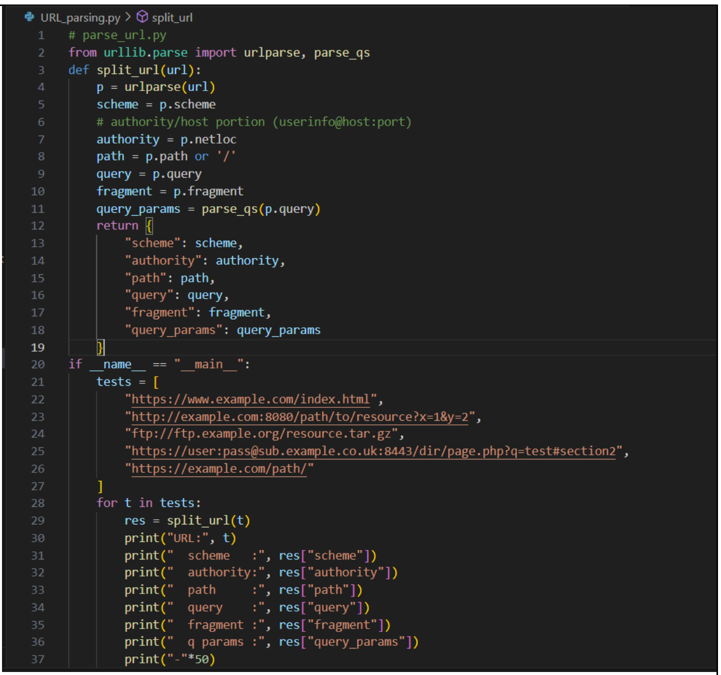
```
icmp
No.      Time        Source           Destination      Protocol Length Info
→   523 5.341830   10.24.74.238     142.250.207.174   ICMP      74 Echo (ping) request  id=0x0001, seq=88/22528, ttl=128 (reply in 527)
←   527 5.348707   142.250.207.174  10.24.74.238      ICMP      74 Echo (ping) reply    id=0x0001, seq=88/22528, ttl=118 (request in 523)
    638 6.358671   10.24.74.238     142.250.207.174   ICMP      74 Echo (ping) request  id=0x0001, seq=89/22784, ttl=128 (reply in 641)
    641 6.378247   142.250.207.174  10.24.74.238      ICMP      74 Echo (ping) reply    id=0x0001, seq=89/22784, ttl=118 (request in 638)
    729 7.375332   10.24.74.238     142.250.207.174   ICMP      74 Echo (ping) request  id=0x0001, seq=90/23040, ttl=128 (reply in 730)
    730 7.383692   142.250.207.174  10.24.74.238      ICMP      74 Echo (ping) reply    id=0x0001, seq=90/23040, ttl=118 (request in 729)
    815 8.391806   10.24.74.238     142.250.207.174   ICMP      74 Echo (ping) request  id=0x0001, seq=91/23296, ttl=128 (reply in 820)
    820 8.407751   142.250.207.174  10.24.74.238      ICMP      74 Echo (ping) reply    id=0x0001, seq=91/23296, ttl=115 (request in 815)

> Frame 523: Packet, 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{E
> Ethernet II, Src: CloudNetwork_88:29:27 (60:e9:aa:88:29:27), Dst: RuckusWirele_83:b1:f2 (c0:c5:20:83:b1:
> Internet Protocol Version 4, Src: 10.24.74.238, Dst: 142.250.207.174
> Internet Control Message Protocol

0000  c0 c5 20 83 b1 f2 60 e9  aa 88 29 27 08 00 45 00   ·· ···`· ··)'··E·
0010  00 3c 58 51 00 00 80 01  2e c1 0a 18 4a ee 8e fa   ·<XQ···· .···J···
0020  cf ae 08 00 4d 03 00 01  00 58 61 62 63 64 65 66   ····M··· ·Xabcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                     wabcdefg hi

Wi-Fi: <live capture in progress>        Packets: 1275 · Displayed: 8 (0.6%)        Profile: Default
```

Figure 17: asgn 10



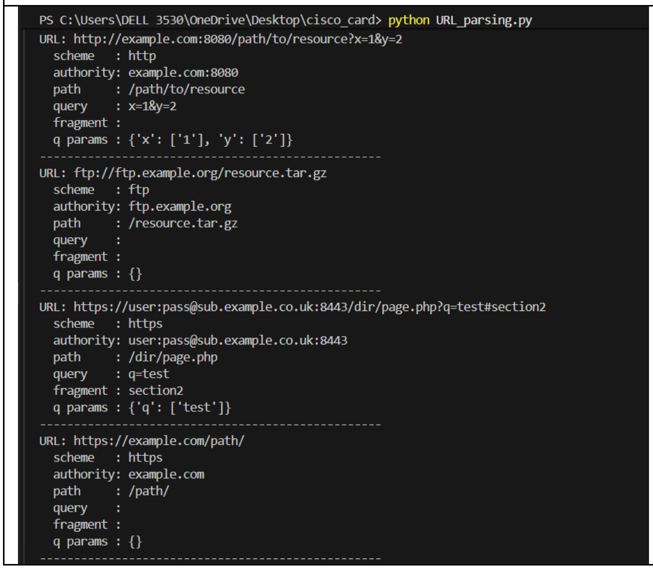
```
PS C:\Users\DELL 3530\OneDrive\Desktop\cisco_card> python Leakybckt.py
--- Real-Time Leaky Bucket Simulation ---
Bucket Size: 15 KB | Leak Rate: 3 KB/sec

Time 1s: Incoming Packet = 9 KB
Added to bucket. Current storage = 9 KB
Sent out 3 KB | Remaining in bucket: 6 KB
[████████------------] 6/15 KB

Time 2s: Incoming Packet = 5 KB
Added to bucket. Current storage = 11 KB
Sent out 3 KB | Remaining in bucket: 8 KB
[██████████----------] 8/15 KB

Time 3s: Incoming Packet = 7 KB
Added to bucket. Current storage = 15 KB
Sent out 3 KB | Remaining in bucket: 12 KB
[████████████████----] 12/15 KB

Time 4s: Incoming Packet = 10 KB
⚠ Bucket Overflow! Dropped 7 KB
Sent out 3 KB | Remaining in bucket: 12 KB
[████████████████----] 12/15 KB

Time 5s: Incoming Packet = 1 KB
Added to bucket. Current storage = 13 KB
Sent out 3 KB | Remaining in bucket: 10 KB
[█████████████-------] 10/15 KB

Time 6s: Incoming Packet = 4 KB
Added to bucket. Current storage = 14 KB
Sent out 3 KB | Remaining in bucket: 11 KB
[██████████████------] 11/15 KB

Time 7s: Incoming Packet = 5 KB
⚠ Bucket Overflow! Dropped 1 KB
```

Figure 18: asgn 11

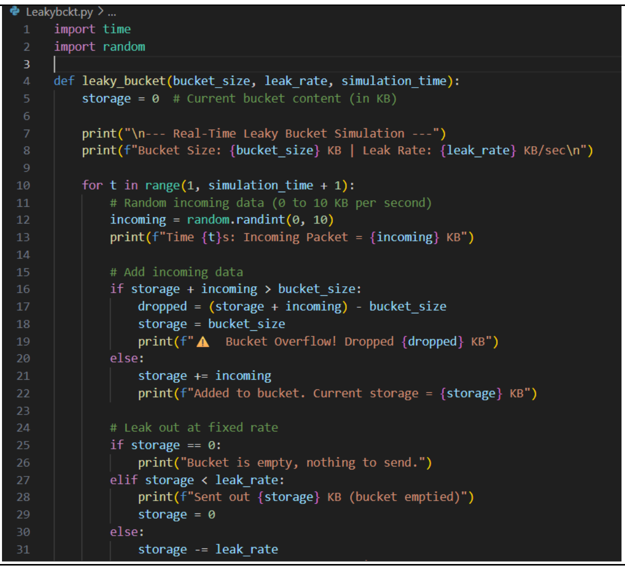


```python
# parse_url.py
from urllib.parse import urlparse, parse_qs
def split_url(url):
    p = urlparse(url)
    scheme = p.scheme
    # authority/host portion (userinfo@host:port)
    authority = p.netloc
    path = p.path or '/'
    query = p.query
    fragment = p.fragment
    query_params = parse_qs(p.query)
    return {
        "scheme": scheme,
        "authority": authority,
        "path": path,
        "query": query,
        "fragment": fragment,
        "query_params": query_params
    }
if __name__ == "__main__":
    tests = [
        "https://www.example.com/index.html",
        "http://example.com:8080/path/to/resource?x=1&y=2",
        "ftp://ftp.example.org/resource.tar.gz",
        "https://user:pass@sub.example.co.uk:8443/dir/page.php?q=test#section2",
        "https://example.com/path/"
    ]
    for t in tests:
        res = split_url(t)
        print("URL:", t)
        print("  scheme    :", res["scheme"])
        print("  authority:", res["authority"])
        print("  path      :", res["path"])
        print("  query     :", res["query"])
        print("  fragment :", res["fragment"])
        print("  q params :", res["query_params"])
        print("-"*50)
```

Figure 19: asgn 12-I



```
PS C:\Users\DELL 3530\OneDrive\Desktop\cisco_card> python URL_parsing.py
URL: http://example.com:8080/path/to/resource?x=1&y=2
  scheme    : http
  authority: example.com:8080
  path      : /path/to/resource
  query     : x=1&y=2
  fragment :
  q params : {'x': ['1'], 'y': ['2']}
--------------------------------------------------
URL: ftp://ftp.example.org/resource.tar.gz
  scheme    : ftp
  authority: ftp.example.org
  path      : /resource.tar.gz
  query     :
  fragment :
  q params : {}
--------------------------------------------------
URL: https://user:pass@sub.example.co.uk:8443/dir/page.php?q=test#section2
  scheme    : https
  authority: user:pass@sub.example.co.uk:8443
  path      : /dir/page.php
  query     : q=test
  fragment : section2
  q params : {'q': ['test']}
--------------------------------------------------
URL: https://example.com/path/
  scheme    : https
  authority: example.com
  path      : /path/
  query     :
  fragment :
  q params : {}
--------------------------------------------------
```



```python
import time
import random

def leaky_bucket(bucket_size, leak_rate, simulation_time):
    storage = 0  # Current bucket content (in KB)

    print("\n--- Real-Time Leaky Bucket Simulation ---")
    print(f"Bucket Size: {bucket_size} KB | Leak Rate: {leak_rate} KB/sec\n")

    for t in range(1, simulation_time + 1):
        # Random incoming data (0 to 10 KB per second)
        incoming = random.randint(0, 10)
        print(f"Time {t}s: Incoming Packet = {incoming} KB")

        # Add incoming data
        if storage + incoming > bucket_size:
            dropped = (storage + incoming) - bucket_size
            storage = bucket_size
            print(f"⚠  Bucket Overflow! Dropped {dropped} KB")
        else:
            storage += incoming
            print(f"Added to bucket. Current storage = {storage} KB")

        # Leak out at fixed rate
        if storage == 0:
            print("Bucket is empty, nothing to send.")
        elif storage < leak_rate:
            print(f"Sent out {storage} KB (bucket emptied)")
            storage = 0
        else:
            storage -= leak_rate
```

Figure 20: asgn 12	*Figure 21: asgn 11 ip1*