# Bansilal Ramnath Agarwal Charitable Trust’s

Vishwakarma Institute of Technology, Pune-37

*(Autonomous Institute of Savitribai Phule Pune University)*



**Department of Computer Engineering**

|  |  |
| --- | --- |
| **Division** | **CS TY B** |
| **Batch** | **3** |
| **GR no.** | **12320165** |
| **Roll no.** | **83** |
| **Name** | **Komal Mahadev Potdar** |

**Assignment No. 4**

**Title:** Write a program to simulate Go back N and Selective Repeat Modes of Sliding Window Protocol in peer-to-peer mode and demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer-to-peer mode. Further extend it to real implementation of Flow Control over TCP protocol.

**Go Back N :** Implementing the Go-Back-N protocol involves setting up both the sender (client) and receiver (server) sides with specific logic to handle the sliding window mechanism. The Go-Back-N protocol allows the sender to send several frames before needing an acknowledgment for the first one, but if a frame is lost or an error occurs, all subsequent frames are retransmitted starting from the lost frame.

**Server**

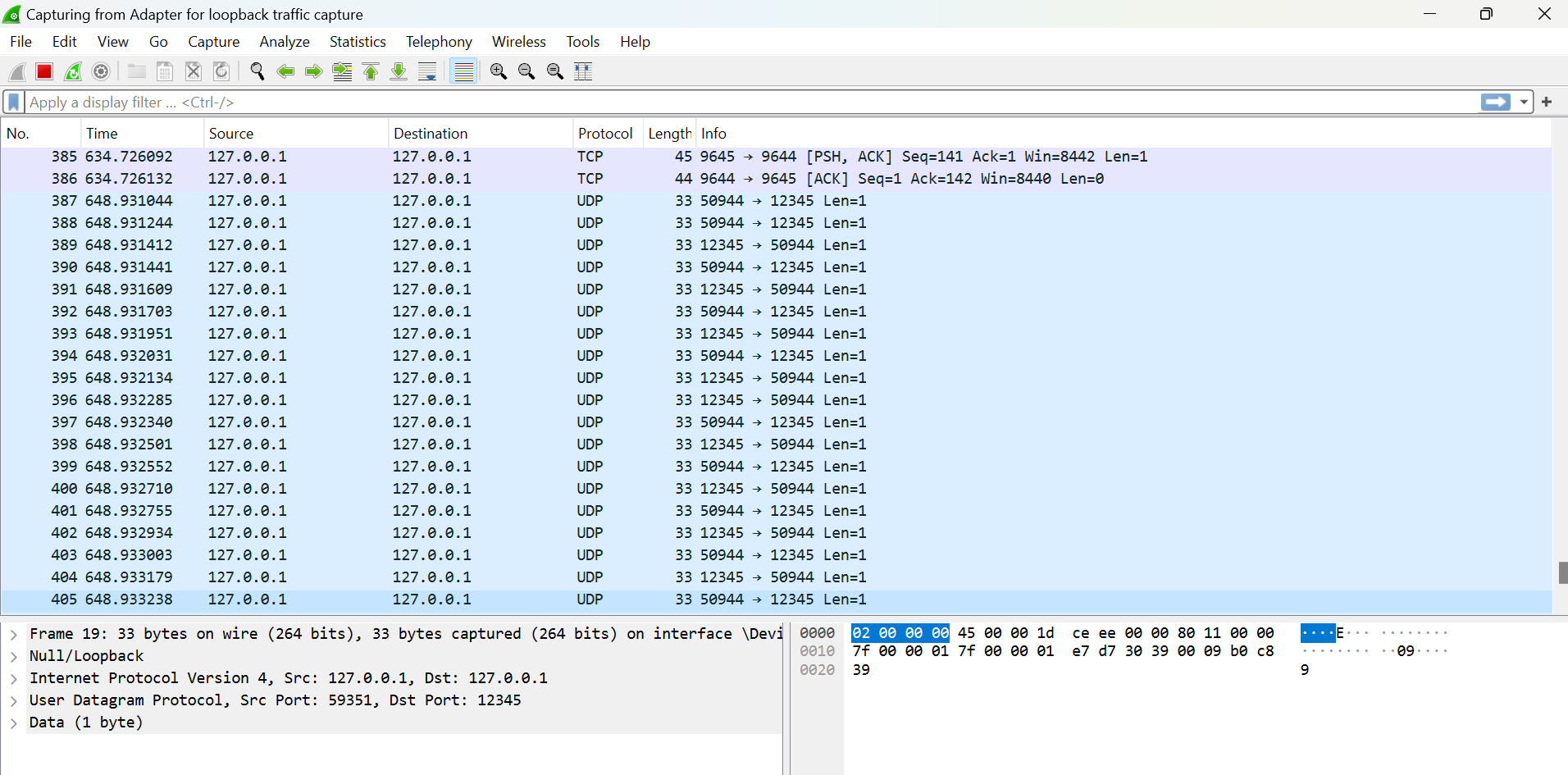
|  |
| --- |
| import socket  def server():      host = '127.0.0.1'      port = 12345      buffer\_size = 1024        expected\_frame = 0  # Initialize expected\_frame here        with socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) as sock:          sock.bind((host, port))          print("Server started... Listening for frames.")            while True:              data, addr = sock.recvfrom(buffer\_size)              frame = int(data.decode())              print(f"Received Frame: {frame}")                if frame == expected\_frame:                  print(f"Sending ACK {frame}")                  sock.sendto(str(frame).encode(), addr)                  expected\_frame += 1              else:                  print(f"Frame {frame} out of order. Expected {expected\_frame}")                  # Resend the last ACK for the expected frame                  sock.sendto(str(expected\_frame - 1).encode(), addr)  if \_\_name\_\_ == "\_\_main\_\_":      server() |

**Client**

|  |
| --- |
| import socket  import threading  import time  # Client Configuration  host = '127.0.0.1'  port = 12345  buffer\_size = 1024  window\_size = 4  # Window size for Go-Back-N  timeout = 2  # Timeout in seconds  def send\_frame(sock, frame, addr):      print(f"Sending Frame: {frame}")      sock.sendto(str(frame).encode(), addr)  def client():      sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)      sock.settimeout(timeout)      addr = (host, port)        base = 0      next\_seq\_num = 0      total\_frames = 10      ack\_received = [False] \* total\_frames        while base < total\_frames:          while next\_seq\_num < base + window\_size and next\_seq\_num < total\_frames:              send\_frame(sock, next\_seq\_num, addr)              next\_seq\_num += 1            try:              ack, \_ = sock.recvfrom(buffer\_size)              ack = int(ack.decode())              print(f"Received ACK for Frame: {ack}")              ack\_received[ack] = True                while base < total\_frames and ack\_received[base]:                  base += 1          except socket.timeout:              print("Timeout occurred. Resending all frames from base:", base)              next\_seq\_num = base      print("All frames sent. Closing client.")      sock.close()  if \_\_name\_\_ == "\_\_main\_\_":      client() |





****

**Selective Repeat Mode:** Selective Repeat (SR) is another sliding window protocol used in reliable data transfer over networks. Like Go-Back-N, it is designed to handle packet loss, errors, and out-of-order delivery, but it does so in a more efficient manner by allowing the receiver to accept out-of-order frames and the sender to retransmit only the specific frames that were lost or received with errors.

**Client**

|  |
| --- |
| import socket  import threading  import time  # Client Configuration  host = '127.0.0.1'  port = 12345  window\_size = 4  timeout = 2  # seconds  # Shared data  frames = list(range(10))  acks = [False] \* len(frames)  lock = threading.Lock()  # Function to send frames  def send\_frame(sock, frame):      sock.sendto(str(frame).encode(), (host, port))      print(f"Sending Frame: {frame}")  def receive\_ack(sock):      global acks      while True:          try:              ack, \_ = sock.recvfrom(1024)              ack = int(ack.decode())              with lock:                  if ack < len(frames):                      acks[ack] = True                      print(f"Received ACK for Frame: {ack}")          except:              break  # Client Function  def client():      global acks      sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)      sock.settimeout(timeout)      # Start a thread to receive ACKs      threading.Thread(target=receive\_ack, args=(sock,)).start()      base = 0      while base < len(frames):          with lock:              while base < len(frames) and acks[base]:                  base += 1  # Move the window              for i in range(base, min(base + window\_size, len(frames))):                  if not acks[i]:                      send\_frame(sock, i)          time.sleep(1)  # Wait for ACKs      print("All frames sent. Closing client.")      sock.close()  if \_\_name\_\_ == "\_\_main\_\_":      client() |

**Server**

|  |
| --- |
| import socket  import threading  import signal  import sys  # Server Configuration  host = '127.0.0.1'  port = 12345  buffer\_size = 1024  expected\_frame = 0  received\_frames = {}  shutdown\_flag = False  # Function to handle incoming frames  def handle\_frames(sock):      global expected\_frame, received\_frames, shutdown\_flag      while not shutdown\_flag:          try:              frame, addr = sock.recvfrom(buffer\_size)              frame = int(frame.decode())              if frame not in received\_frames:                  print(f"Received Frame: {frame}")                  received\_frames[frame] = True                  sock.sendto(str(frame).encode(), addr)  # Send ACK                  # Slide the window if the expected frame is received                  while expected\_frame in received\_frames:                      expected\_frame += 1              else:                  print(f"Duplicate Frame: {frame}, already received.")                  sock.sendto(str(frame).encode(), addr)  # Resend ACK for duplicate frame          except socket.timeout:              continue  # Continue to check for shutdown\_flag  def signal\_handler(sig, frame):      global shutdown\_flag      print("\nTermination signal received. Closing server...")      shutdown\_flag = True  def server():      global shutdown\_flag      sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)      sock.bind((host, port))      sock.settimeout(1)  # Set timeout to 1 second to allow checking for shutdown\_flag      print("Server started... Listening for frames.")      # Set up signal handling      signal.signal(signal.SIGINT, signal\_handler)      handle\_frames(sock)      print("Shutting down server.")      sock.close()  if \_\_name\_\_ == "\_\_main\_\_":      server() |

