CN-Programming-Assignment-3

Sahil Gupta (2022430) Avi Sharma (2022119)

7th November 2024

1 Approach

The code represents the bidirectional Go Back N protocol in which there are two entities which have nearly the same code base , but the change in them is the different interchanged socket they bind to and the different interchanged remote address they send the packets to. This used UDP sockets for communication and includes features such as sequence numbering , acknowledgments , timeouts and retransmission mechanisms.

2 Protocol Configuration

The implementation uses several key configuration parameters:

- Maximum sequence number (maxSEQ): 7 (sequence numbers 0-7)
- Total packets to send: 10,000
- Timeout value: 2 seconds
- Packet drop probability: 10%

The following data structures are implemented:
1) **Packet Class**

```
@dataclass
class Packet:
    data: str
    timeval: float
```

This stores actual data string and packet creation
2) **Frame Class**

```
@dataclass
class Frame:
    seq: int    # Sequence number
    ack: int    # Acknowledgment number
    info: Packet
```

This encapsulates a packet with sequence number for ordering , acknowledgment number for flow control , actual packet data and json serialization/deserialization for network transmission.

Protocol Implementation
The following state variables are used for data flow:

- `nextFrameToSend`: Next sequence number to be used
- `ackExpected`: Oldest unacknowledged frame

- `frameExpected`: Next frame expected by receiver
- `nBuffered`: Number of buffered unacknowledged frames
- `buffer`: Array storing unacknowledged packets
- `timer`: Array tracking timeout for each sequence number

3 Core Mechanisms

3.1 Flow Control

The implementation uses a sliding window mechanism with the following characteristics:

- Window size: maxSEQ frames
- Buffer management: Sender can transmit up to maxSEQ unacknowledged frames
- Flow control: Network layer is ready only when buffer space is available

3.2 Error Control Implementation

The error control mechanism is implemented as follows:

```
def checktimeout(self):
    if self.packetsSend >= TotalPacketsToSend:
        self.stopEvent.set()
        return False
    for i in range(maxSEQ + 1):
        if self.timer[i] != 0:
            if time.time() - self.timer[i] > self.timeoutval:
                return True
    return False
```

4 Threading Model

The implementation utilizes three main threads:

4.1 Network Layer Thread

```
def networkLayer(self):
    if self.packetsSend >= TotalPacketsToSend:
        exit(0)
    cnt = 0
    T1 = 0.1 ; T2 = 0.2
    while not self.stopEvent.is_set():
        if self.netWorkLayerReady:
            time.sleep(random.uniform(T1 , T2))
            packet = Packet(
                data = f"Packet_Entity_2_{cnt}",
                timeval = time.time()
            )
            self.queue1.append((EventType.networkLayerReady, packet))
            cnt += 1
        time.sleep(0.01)
```

4.2 Receiver Thread

```
def reciever(self):
    self.sock.settimeout(0.5)
    while not self.stopEvent.is_set():
        try:
            data, unused = self.sock.recvfrom(1024)
            frame = Frame.fromJson(data.decode())
            self.queue1.append((EventType.frameArrival, frame))
            time.sleep(random.uniform(0.1, 0.2))
        except socket.timeout:
            continue
```

5 Error Recovery

5.1 Retransmission Process

When a timeout occurs, the protocol implements the following retransmission logic:

```
if self.checktimeout():
    print(f'----------Timeout, resending window')
    self.retransmissions += 1
    nextFrame = self.ackExpected
    for i in range(self.nBuffered):
        self.helpersender(nextFrame, self.buffer)
        nextFrame = (nextFrame + 1) % (maxSEQ + 1)
```

6 Performance Metrics

The implementation tracks several performance indicators:

```
def CalcStat(self):
    print("Packets sent : ", self.packetsSend)
    print("Packets received : ", self.packetsReceived)
    print("Retransmissions : ", self.retransmissions)
    total_delay = 0
    valid_delays = 0
    for seq in self.receiveTimes:
        if seq in self.sendTimes:
            total_delay += self.receiveTimes[seq] - self.sendTimes[seq]
            valid_delays += 1
    avg_delay = total_delay / TotalPacketsToSend
    print(f"Average delivery delay: {avg_delay:.4f} seconds")
    total_attempts = sum(self.sendAttempts.values())
    avg_attempts = total_attempts / TotalPacketsToSend
    print(f"Average transmission attempts per frame: {avg_attempts:.2f}")
```

7 Network Configuration

The implementation uses the following network configuration:

- Protocol: UDP sockets
- Socket timeout: 0.5 seconds
- Simulated network delays: 0.1-0.2 seconds
- Configurable ports for bidirectional communication

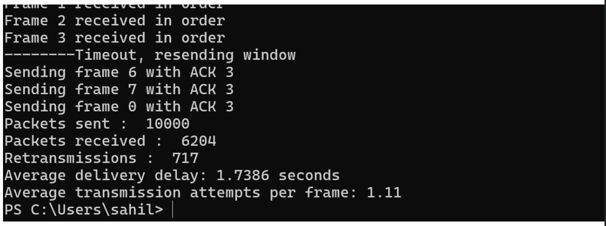
8 Statistical Analysis

The protocol collects and analyzes the following metrics:

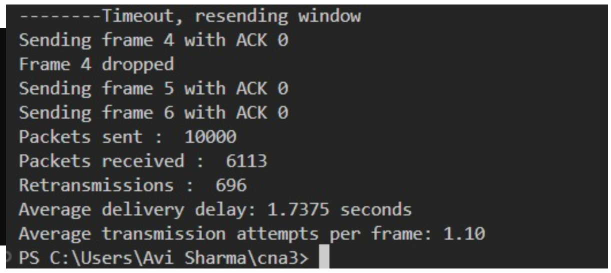
- Transmission success rate
- Number of retransmissions
- Average delivery delay
- Per-frame transmission attempts

9 Results

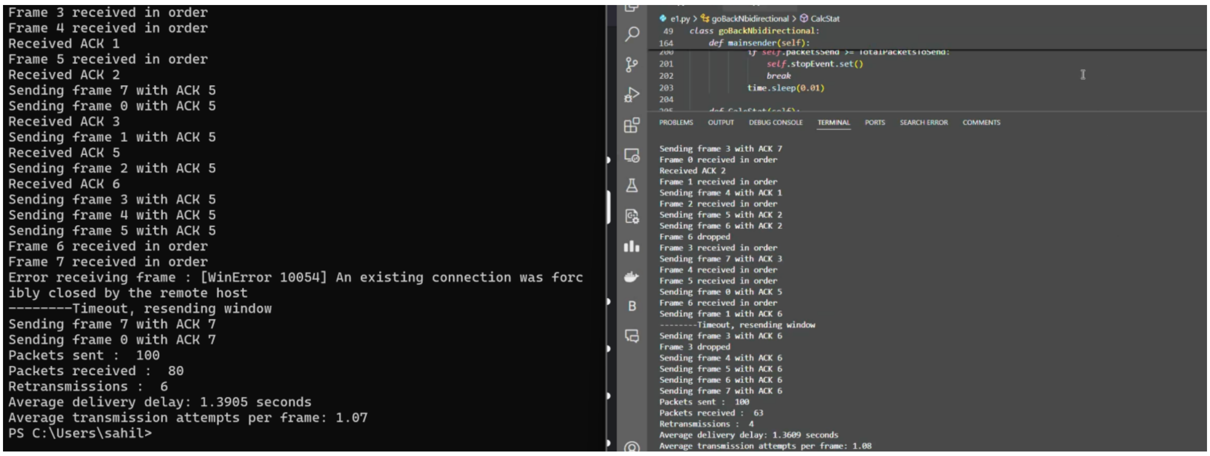
The values are as follows:
T1 = 0.1 , T2 = 0.2 , T3 = 0.1 , T4 = 0.2 , p = 0.1



Entity 1 on 1st machine for 10k packets



Entity 2 on 2nd machine for 10k packets



Results on different machines for 100 packets as a demo

Figure 1: Changed Value of (Drop probabilty) p = 0.9

Figure 2: Changed Value of T3 and T4 to be 0.1 and 0.2

Figure 3: Changed Value of T3 and T4 to be 0.9 and 1.0

10 References

Tanenbaum Page 236 - Code for Go Back N protocol