Simulated Go-Back-N ARQ Protocol

Overview

This document explains the implementation of a simulated **Go-Back-N ARQ protocol** using Python with UDP sockets to exchange packets between two machines (`machine1.py` and `machine2.py`). The protocol simulates:

- Sending and receiving packets between sender and receiver.
- Functionalities such as retransmissions, window-based flow control, and loss handling.

—

1. Constants and Parameters

General Parameters

```
T1 = 1; T2 = 3; T3 = 1; T4 = 3; dropProb = 0.1
N = 8; windowSize = 7; timeout = 1.0
```

- `T1, T2`: Bounds for time interval between generating packets.
- `T3, T4`: Bounds for time interval between sending packets and acknowledgments.
- `dropProb`: Probability of dropping a frame or acknowledgment (to simulate loss).
- `N`: Maximum sequence number (frames cycle modulo `N`).
- `windowSize`: Sender's window size in the Go-Back-N protocol.
- `timeout`: Timeout duration for retransmission.

Global Variables

```
base, nextSeqNum, expectedSeqNum = 0, 0, 0
sendBuffer, packetStats = {}, {}
queue1, queue2 = queue.Queue(), queue.Queue()
sentpackets, receivedPackets, droppedPackets = 0, 0, 0
ackSend, ackRecv = 0, 0
```

- `base`: Oldest unacknowledged packet's sequence number.
- `nextSeqNum`: Next packet's sequence number to send.
- `expectedSeqNum`: Receiver's expected sequence number (in-order delivery).
- `sendBuffer`: Stores packets waiting for acknowledgment.
- `packetStats`: Logs packet send time, acknowledgment time, and retransmission count.
- `queue1, queue2`: Buffers for outgoing and incoming packets at the network layer.
- `sentpackets, receivedPackets, droppedPackets`: Counters for metrics.

—

2. Functions

Network Layer Functions

`networkgenPackets()`

Simulates generating packets in the network layer.

- Generates 100 packets (e.g., `Packet_0`, `Packet_1`, etc.).
- Each packet is pushed into `queue1` (outgoing buffer) after a random time (`T1` to `T2` seconds).

`networkrecvpackets()`

Simulates receiving packets at the receiver's network layer.

- Continuously fetches packets from `queue2` (incoming buffer).
- Increments `receivedPackets`.

—

Transport Layer Functions

`framesender(sock, remoteAddr)`

Implements the **sender's data transmission logic**.

1. Sends packets within the window size.
2. Logs details in `sendBuffer` and `packetStats`.
3. Simulates packet loss using `dropProb`.
4. Sends frames via `sock.sendto()` if not dropped.

`framereciever(sock, remoteAddr)`

Implements the **receiver's data reception and acknowledgment**.

1. Receives and decodes incoming frames.
2. Updates `base` if acknowledgment is received.
3. Sends ACKs for valid frames.
4. Simulates loss for acknowledgments.

`timeouthandler(sock, remoteAddr)`

Handles **timeout and retransmissions**.

- Detects timeout for unacknowledged packets.
- Retransmits packets starting from `base`.
- Updates retransmission count in `packetStats`.

—

Statistics Calculation

`statscalc()`

Computes protocol efficiency metrics:

- **Average Delay**: Total delay for all acknowledged packets divided by the number of acknowledged packets.
- **Average Retransmissions**: Total retransmissions divided by the number of acknowledged packets.

—

3. Thread Management

The program runs several functions concurrently using threads:

```
threads = [
    threading.Thread(target=networkgenPackets),
    threading.Thread(target=networkrecvpackets),
    threading.Thread(target=framesender, args=(sock,
       remoteAddr)),
    threading.Thread(target=framereciever, args=(sock,
       remoteAddr)),
    threading.Thread(target=timeouthandler, args=(sock,
       remoteAddr))
]
```

Each thread simulates independent processes for:

- Generating packets.
- Receiving packets.
- Sending frames.
- Receiving frames/ACKs.
- Handling retransmissions.

—

4. Sockets

Each machine binds UDP sockets (`sock.bind()`) to different ports (`5009` and `5010`) and exchanges frames using `sendto()` and `recvfrom()`.

—

5. Key Aspects

- **Window-based ARQ**: Implements Go-Back-N with a fixed window size.
- **Loss Simulation**: Randomly drops packets and ACKs to simulate unreliable networks.
- **Metrics**: Measures efficiency using delay and retransmission stats.
- **Concurrency**: Threads simulate real-world independent processes.