Theory Question
How would you simulate the Network entity as a process that produces packets at random time instants? And similarly simulate a network entity that consumes incoming packets that have been added to a queue of incoming packets? Are threads an option?

Proposed Solution

Threads are indeed a good option for simulating network entities that produce and consume packets at random times

1. **Producer (Network Entity)**:
   A network entity that generates packets at random intervals can be implemented as a producer. The producer thread generates packets after sleeping for random durations, then places them into a shared queue. This mimics real-world packet generation in a network.
2. **Consumer (DataLink Entity)**:
   A data link entity that consumes packets from the queue processes them (e.g., sends them over a network). This entity continuously checks the queue for incoming packets and processes them as they arrive. The consumer can operate concurrently with the producer using a separate thread.
3. **Concurrency with Threads**:
   Threads allow the producer and consumer to run simultaneously without blocking each other. The producer can generate packets and add them to the queue, while the consumer can independently pick up and process them. This models asynchronous real-world network behavior.
4. **Queue for Synchronization**:
   The `queue.Queue` is thread-safe, meaning it handles the sharing of packets between the producer and consumer without race conditions. This allows safe, concurrent packet generation and processing.
5. **Why Not Processes?**
   While you could use processes (via `multiprocessing`), threads are generally more efficient for I/O-bound tasks like packet transmission and reception. Threads share memory space, making it easier to communicate (via the queue), and are lighter on resources compared to processes.

In summary, threads work well here for simulating network entities because they provide efficient concurrency, simple communication via queues, and the ability to model asynchronous packet processing and generation.

Code Explanation

1. NetworkEntity Class (Producer)

This class simulates a network entity that **produces packets** at random time intervals and adds them to a shared queue.

- **`__init__(self, packet_queue, T1, T2, max_packets=10000)`**:
  Initializes the `NetworkEntity` instance, where:
  - `packet_queue` is the queue where packets will be placed.
  - `T1` and `T2` define the range for the random time delay before producing the next packet.
  - `max_packets` specifies the total number of packets to be generated.
- **`run(self)`**:
  The `run()` method is executed when the thread starts. It keeps generating packets until it reaches `max_packets`:
  - It sleeps for a random time between `T1` and `T2` using `random.uniform(T1, T2)` to simulate random packet generation intervals.
  - Each generated packet is added to the `packet_queue` using `self.packet_queue.put(packet)`.
- **`generate_packet(self)`**:
  Generates a packet. In this example, it's just a string `"Packet"`, but in a real application, it could represent more complex data.

2. DataLinkEntity Class (Consumer)

This class simulates a **data link entity** that **consumes packets** from the queue, processes them, and handles acknowledgments (ACKs).

- **`__init__(self, socket, address, packet_queue, entity_name, start_time, ...)`**:
  Initializes the `DataLinkEntity`, where:
  - `socket` and `address` represent the communication channel and destination.
  - `packet_queue` is the shared queue from which the entity consumes packets.
  - `entity_name` is a name used for identifying this entity in the logs.
  - `start_time` is the start time for calculating elapsed times.
  - `window_size`, `P` (packet drop probability), `T3`, and `T4` are various network parameters, such as window size for sliding window protocol and random delays for simulating network conditions.

- **`run(self)`**:
  The `run()` method is executed when the thread starts:
  - It continuously checks the queue to consume packets.
  - If a packet is available in the queue, it calls `send_packet()` to simulate sending it.
  - It also listens for incoming ACKs using `receive_packet()`.
- **`send_packet(self, packet)`**:
  This method simulates sending a packet. It:
  - Checks if the current sequence number is within the allowed window size (`self.send_base + self.window_size`).
  - Sends the packet over the socket if the packet is not dropped (with a random probability `P`).
  - Records the send time and increments the sequence number (`next_seq_num`).
- **`drop_packet(self)`**:
  This method simulates packet loss. It uses `random.random()` to determine if a packet should be dropped based on the probability `P`.
- **`receive_packet(self)`**:
  This method simulates receiving packets (or ACKs) from the network. It:
  - Receives a frame from the socket.
  - If the received frame is an acknowledgment (`ACK`), it updates the `send_base` to the acknowledged sequence number.
  - If it's a data packet, it checks if it matches the expected sequence number and then sends an acknowledgment back.
- **`send_ack(self, ack_num)`**:
  Sends an acknowledgment back to the sender indicating that the packet was received.
- **`create_frame(self, packet, seq_num)`**:
  Creates a frame from the sequence number and packet data. The frame consists of a sequence number, acknowledgment number, and the packet itself.

3. Main Code Execution

At the bottom of the code, we initiate the network simulation:

- **Sockets**: Two UDP sockets (`socket1` and `socket2`) are created and bound to ports `12345` and `12346` respectively. These sockets simulate communication between two network entities.
- **Queues**: Two queues (`packet_queue1` and `packet_queue2`) are created. These queues are used for communication between the producer (which generates packets) and the consumer (which processes packets).
- **Network Entities**: Two `NetworkEntity` instances are created (for packet generation), each with its own queue and random time interval (`T1`, `T2`) for packet generation.

- **DataLinkEntities**: Two `DataLinkEntity` instances are created to simulate the receiving and processing of packets from each other:
    - `dle1` simulates the behavior of a server on `socket1` (sending packets to `address2`).
    - `dle2` simulates the behavior of a server on `socket2` (sending packets to `address1`).
- **Thread Start**: Each of the network entities (`network_entity1`, `network_entity2`, `dle1`, `dle2`) is started by calling `start()`, which runs each one in its own thread. This allows them to execute concurrently.
- **Join Threads**: Finally, `join()` is called on each thread. This ensures the main program waits for all threads to complete before it exits.

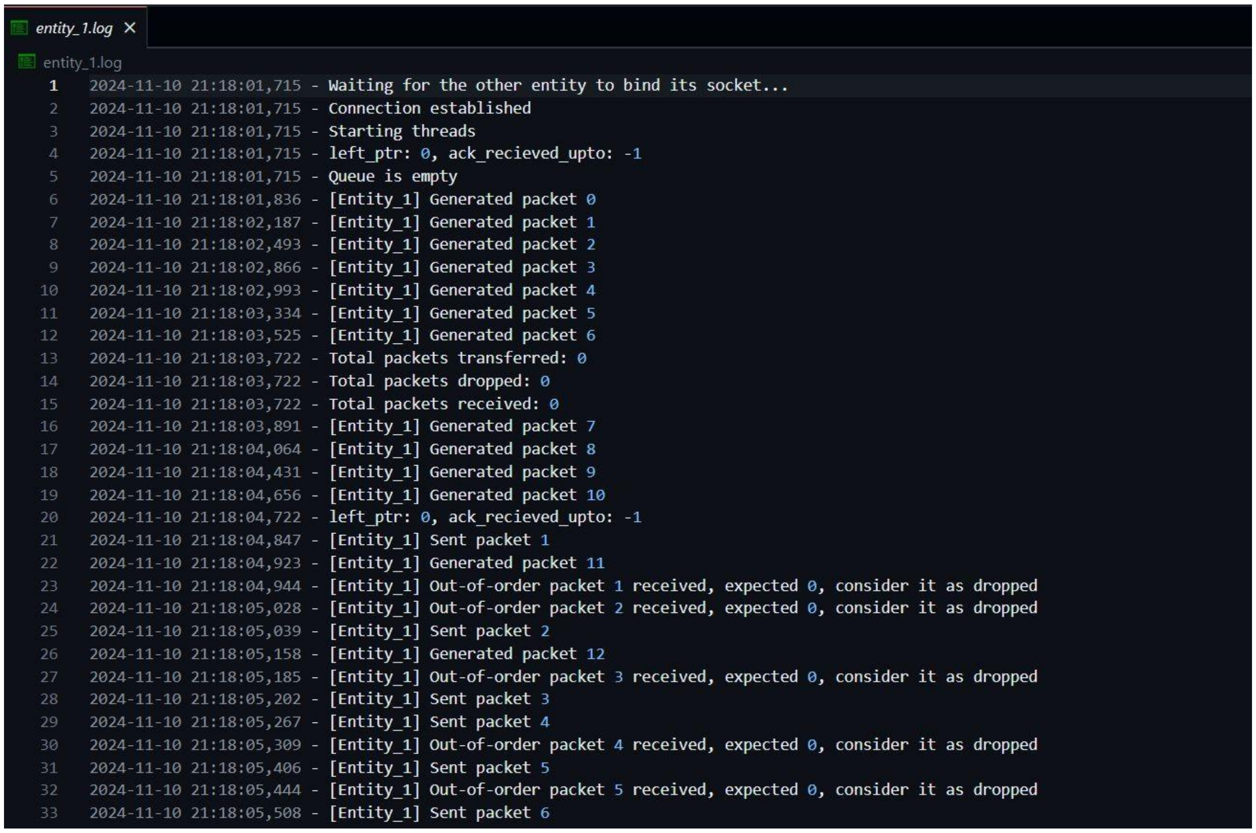
Summary of Simulation Flow

1. **Packet Generation**:
   The producer threads (`NetworkEntity`) generate packets at random intervals and place them into the queue.
2. **Packet Consumption**:
   The consumer threads (`DataLinkEntity`) continuously check the queue for new packets and process them (e.g., by simulating sending them over the network and handling acknowledgments).
3. **Concurrency**:
   Both packet generation and packet consumption happen concurrently in different threads, allowing for an asynchronous simulation of network traffic.
4. **Network Behavior**:
   The simulation includes randomness (for packet delays, drops, and transmission times) and uses the sliding window protocol-like mechanism, where each `DataLinkEntity` manages a sequence of packets to send and expects corresponding ACKs.

Conclusion

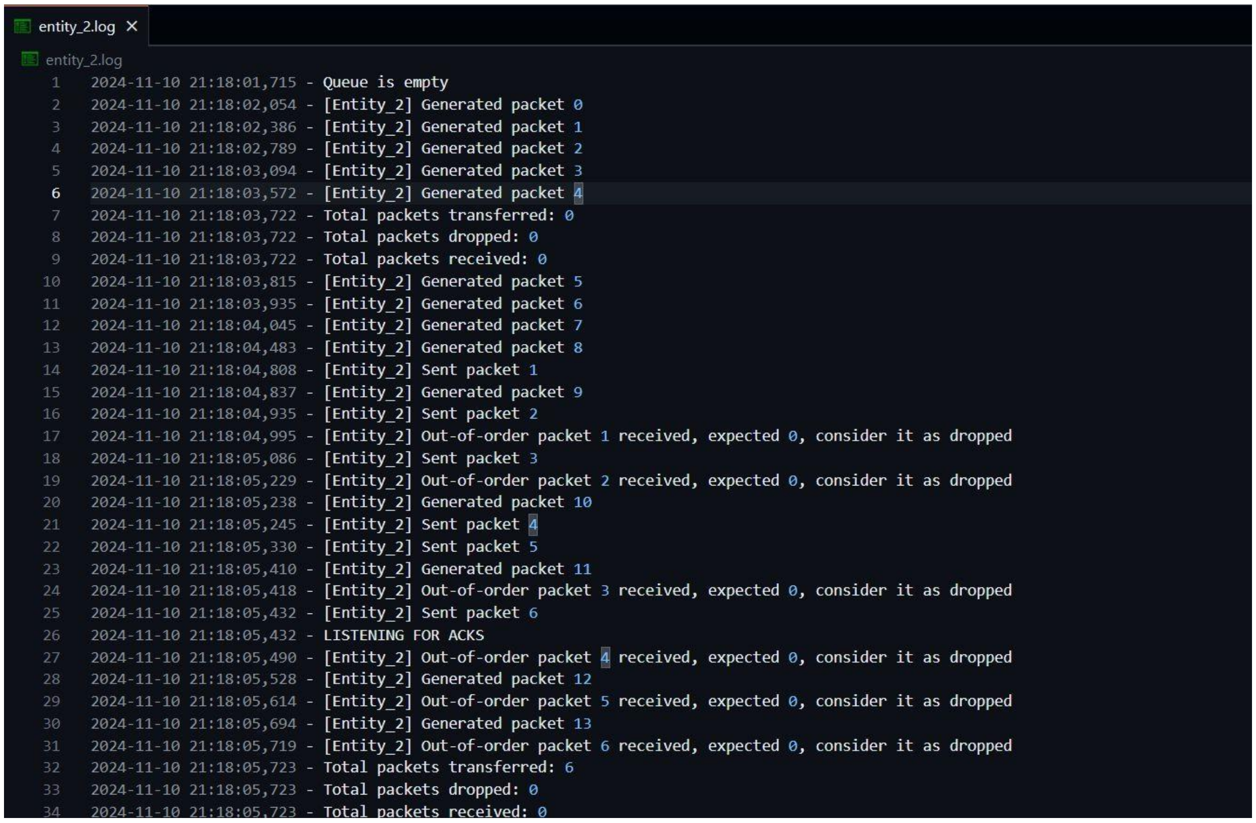
This code models a simple network communication scenario where one entity generates packets, places them in a queue, and the other entity consumes them, sends them over a simulated network, and acknowledges receipt. By using threads and queues, it simulates the concurrent, asynchronous nature of packet-based communication in networking.

Outputs



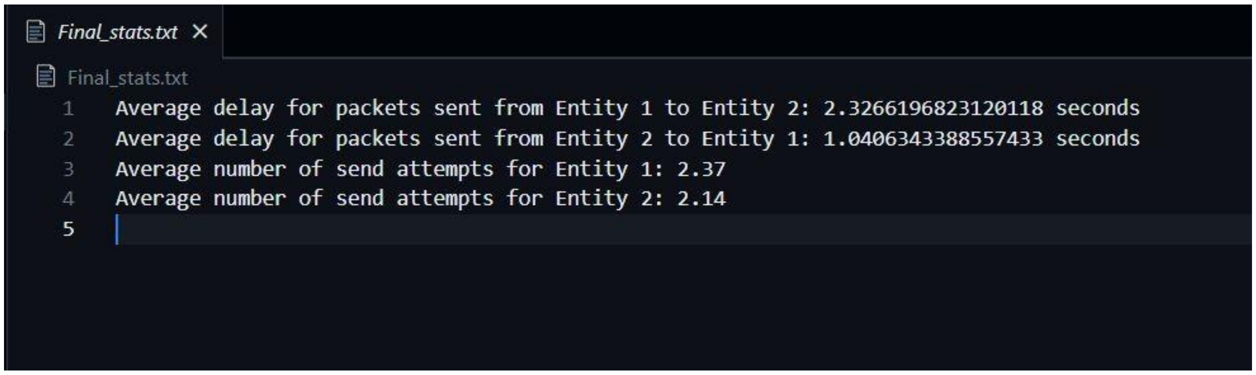
```
2024-11-10 21:18:01,715 - Waiting for the other entity to bind its socket...
2024-11-10 21:18:01,715 - Connection established
2024-11-10 21:18:01,715 - Starting threads
2024-11-10 21:18:01,715 - left_ptr: 0, ack_recieved_upto: -1
2024-11-10 21:18:01,715 - Queue is empty
2024-11-10 21:18:01,836 - [Entity_1] Generated packet 0
2024-11-10 21:18:02,187 - [Entity_1] Generated packet 1
2024-11-10 21:18:02,493 - [Entity_1] Generated packet 2
2024-11-10 21:18:02,866 - [Entity_1] Generated packet 3
2024-11-10 21:18:02,993 - [Entity_1] Generated packet 4
2024-11-10 21:18:03,334 - [Entity_1] Generated packet 5
2024-11-10 21:18:03,525 - [Entity_1] Generated packet 6
2024-11-10 21:18:03,722 - Total packets transferred: 0
2024-11-10 21:18:03,722 - Total packets dropped: 0
2024-11-10 21:18:03,722 - Total packets received: 0
2024-11-10 21:18:03,891 - [Entity_1] Generated packet 7
2024-11-10 21:18:04,064 - [Entity_1] Generated packet 8
2024-11-10 21:18:04,431 - [Entity_1] Generated packet 9
2024-11-10 21:18:04,656 - [Entity_1] Generated packet 10
2024-11-10 21:18:04,722 - left_ptr: 0, ack_recieved_upto: -1
2024-11-10 21:18:04,847 - [Entity_1] Sent packet 1
2024-11-10 21:18:04,923 - [Entity_1] Generated packet 11
2024-11-10 21:18:04,944 - [Entity_1] Out-of-order packet 1 received, expected 0, consider it as dropped
2024-11-10 21:18:05,028 - [Entity_1] Out-of-order packet 2 received, expected 0, consider it as dropped
2024-11-10 21:18:05,039 - [Entity_1] Sent packet 2
2024-11-10 21:18:05,158 - [Entity_1] Generated packet 12
2024-11-10 21:18:05,185 - [Entity_1] Out-of-order packet 3 received, expected 0, consider it as dropped
2024-11-10 21:18:05,202 - [Entity_1] Sent packet 3
2024-11-10 21:18:05,267 - [Entity_1] Sent packet 4
2024-11-10 21:18:05,309 - [Entity_1] Out-of-order packet 4 received, expected 0, consider it as dropped
2024-11-10 21:18:05,406 - [Entity_1] Sent packet 5
2024-11-10 21:18:05,444 - [Entity_1] Out-of-order packet 5 received, expected 0, consider it as dropped
2024-11-10 21:18:05,508 - [Entity_1] Sent packet 6
```

Entity1 logs



```
2024-11-10 21:18:01,715 - Queue is empty
2024-11-10 21:18:02,054 - [Entity_2] Generated packet 0
2024-11-10 21:18:02,386 - [Entity_2] Generated packet 1
2024-11-10 21:18:02,789 - [Entity_2] Generated packet 2
2024-11-10 21:18:03,094 - [Entity_2] Generated packet 3
2024-11-10 21:18:03,572 - [Entity_2] Generated packet 4
2024-11-10 21:18:03,722 - Total packets transferred: 0
2024-11-10 21:18:03,722 - Total packets dropped: 0
2024-11-10 21:18:03,722 - Total packets received: 0
2024-11-10 21:18:03,815 - [Entity_2] Generated packet 5
2024-11-10 21:18:03,935 - [Entity_2] Generated packet 6
2024-11-10 21:18:04,045 - [Entity_2] Generated packet 7
2024-11-10 21:18:04,483 - [Entity_2] Generated packet 8
2024-11-10 21:18:04,808 - [Entity_2] Sent packet 1
2024-11-10 21:18:04,837 - [Entity_2] Generated packet 9
2024-11-10 21:18:04,935 - [Entity_2] Sent packet 2
2024-11-10 21:18:04,995 - [Entity_2] Out-of-order packet 1 received, expected 0, consider it as dropped
2024-11-10 21:18:05,086 - [Entity_2] Sent packet 3
2024-11-10 21:18:05,229 - [Entity_2] Out-of-order packet 2 received, expected 0, consider it as dropped
2024-11-10 21:18:05,238 - [Entity_2] Generated packet 10
2024-11-10 21:18:05,245 - [Entity_2] Sent packet 4
2024-11-10 21:18:05,330 - [Entity_2] Sent packet 5
2024-11-10 21:18:05,410 - [Entity_2] Generated packet 11
2024-11-10 21:18:05,418 - [Entity_2] Out-of-order packet 3 received, expected 0, consider it as dropped
2024-11-10 21:18:05,432 - [Entity_2] Sent packet 6
2024-11-10 21:18:05,432 - LISTENING FOR ACKS
2024-11-10 21:18:05,490 - [Entity_2] Out-of-order packet 4 received, expected 0, consider it as dropped
2024-11-10 21:18:05,528 - [Entity_2] Generated packet 12
2024-11-10 21:18:05,614 - [Entity_2] Out-of-order packet 5 received, expected 0, consider it as dropped
2024-11-10 21:18:05,694 - [Entity_2] Generated packet 13
2024-11-10 21:18:05,719 - [Entity_2] Out-of-order packet 6 received, expected 0, consider it as dropped
2024-11-10 21:18:05,723 - Total packets transferred: 6
2024-11-10 21:18:05,723 - Total packets dropped: 0
2024-11-10 21:18:05,723 - Total packets received: 0
```

Entity 2 logs



```
Average delay for packets sent from Entity 1 to Entity 2: 2.3266196823120118 seconds
Average delay for packets sent from Entity 2 to Entity 1: 1.0406343388557433 seconds
Average number of send attempts for Entity 1: 2.37
Average number of send attempts for Entity 2: 2.14
```

Final Statistics