```
import threading
import random
import time

class Frame:
    def __init__(self, seq_num):
        self.seq_num = seq_num

class Sender:
    WINDOW_SIZE = 4
    MAX_FRAMES = 10
    TIMEOUT = 2  # seconds

    def __init__(self):
        self.frames = [Frame(i) for i in range(self.MAX_FRAMES)]
        self.base = 0
        self.next_seq_num = 0
        self.ack_received = [False] * self.MAX_FRAMES
        self.lock = threading.Lock()
        self.timer = None
        self.stop_sending = False

    def send_frame(self, frame):
        print(f"Sending frame: {frame.seq_num}")
        if random.random() < 0.9:  # 90% chance to "send" successfully
            print(f"Frame {frame.seq_num} sent.")
        else:
            print(f"Frame {frame.seq_num} lost.")

    def receive_ack(self, ack_num):
        with self.lock:
            if ack_num < self.MAX_FRAMES and not self.ack_received[ack_num]:
                self.ack_received[ack_num] = True
                print(f"ACK received for frame {ack_num}")
                while self.base < self.MAX_FRAMES and
self.ack_received[self.base]:
                    self.base += 1
                if self.base < self.next_seq_num and not self.stop_sending:
                    self.start_timer()

    def start_timer(self):
        if self.timer is not None:
            self.timer.cancel()
        self.timer = threading.Timer(self.TIMEOUT, self.timeout)
        self.timer.start()
        print(f"Timer started at {self.base}")
```

```
    def timeout(self):
        with self.lock:
            print(f"Timeout: Resending frames {self.base} to
{self.next_seq_num - 1}")
            for i in range(self.base, self.next_seq_num):
                self.send_frame(self.frames[i])
            self.start_timer()

    def run(self):
        def sender_thread():
            while self.base < self.MAX_FRAMES and not self.stop_sending:
                with self.lock:
                    while self.next_seq_num < self.base + self.WINDOW_SIZE and
self.next_seq_num < self.MAX_FRAMES:
                        self.send_frame(self.frames[self.next_seq_num])
                        self.next_seq_num += 1
                        if self.base == self.next_seq_num - 1:
                            self.start_timer()
                time.sleep(0.1)  # Adjusted sleep time for responsiveness

        threading.Thread(target=sender_thread).start()

    def stop(self):
        self.stop_sending = True
        if self.timer is not None:
            self.timer.cancel()

class Receiver:
    def __init__(self, sender):
        self.sender = sender
        self.expected_seq_num = 0
        self.stop_receiving = False

    def receive_frame(self, frame):
        print(f"Received frame: {frame.seq_num}")
        if frame.seq_num == self.expected_seq_num:
            if random.random() < 0.9:  # 90% chance to "acknowledge"
successfully
                print(f"Acknowledging frame {frame.seq_num}")
                self.sender.receive_ack(frame.seq_num)
                self.expected_seq_num += 1
            else:
                print(f"ACK for frame {frame.seq_num} lost.")
        else:
            print(f"Unexpected frame {frame.seq_num}. Expected
{self.expected_seq_num}. Ignoring.")

    def run(self):
```

```
        def receiver_thread():
            while not self.stop_receiving and self.sender.base <
Sender.MAX_FRAMES:
                for i in range(self.sender.base, self.sender.next_seq_num):
                    if not self.sender.ack_received[i]:
                        self.receive_frame(self.sender.frames[i])
                time.sleep(0.5)

        threading.Thread(target=receiver_thread).start()

    def stop(self):
        self.stop_receiving = True

if __name__ == "__main__":
    sender = Sender()
    receiver = Receiver(sender)

    # Start the sender and receiver
    sender.run()
    receiver.run()

    # Let the simulation run for a certain period
    try:
        while sender.base < Sender.MAX_FRAMES:
            time.sleep(1)
    except KeyboardInterrupt:
        pass
    finally:
        sender.stop()
        receiver.stop()
        print("Simulation ended.")
```

```
Sending frame: 0
Frame 0 sent.
Timer started at 0
Sending frame: 1
Frame 1 sent.
Sending frame: 2
Frame 2 sent.
Sending frame: 3
Frame 3 sent.
Received frame: 0
Acknowledging frame 0
ACK received for frame 0
Timer started at 1
Received frame: 1
ACK for frame 1 lost.
Received frame: 2
Unexpected frame 2. Expected 1. Ignoring.
Received frame: 3
Unexpected frame 3. Expected 1. Ignoring.
Sending frame: 4
Frame 4 lost.
Received frame: 1
Acknowledging frame 1
ACK received for frame 1
Timer started at 2
...
Received frame: 9
Acknowledging frame 9
ACK received for frame 9
Simulation ended.
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...