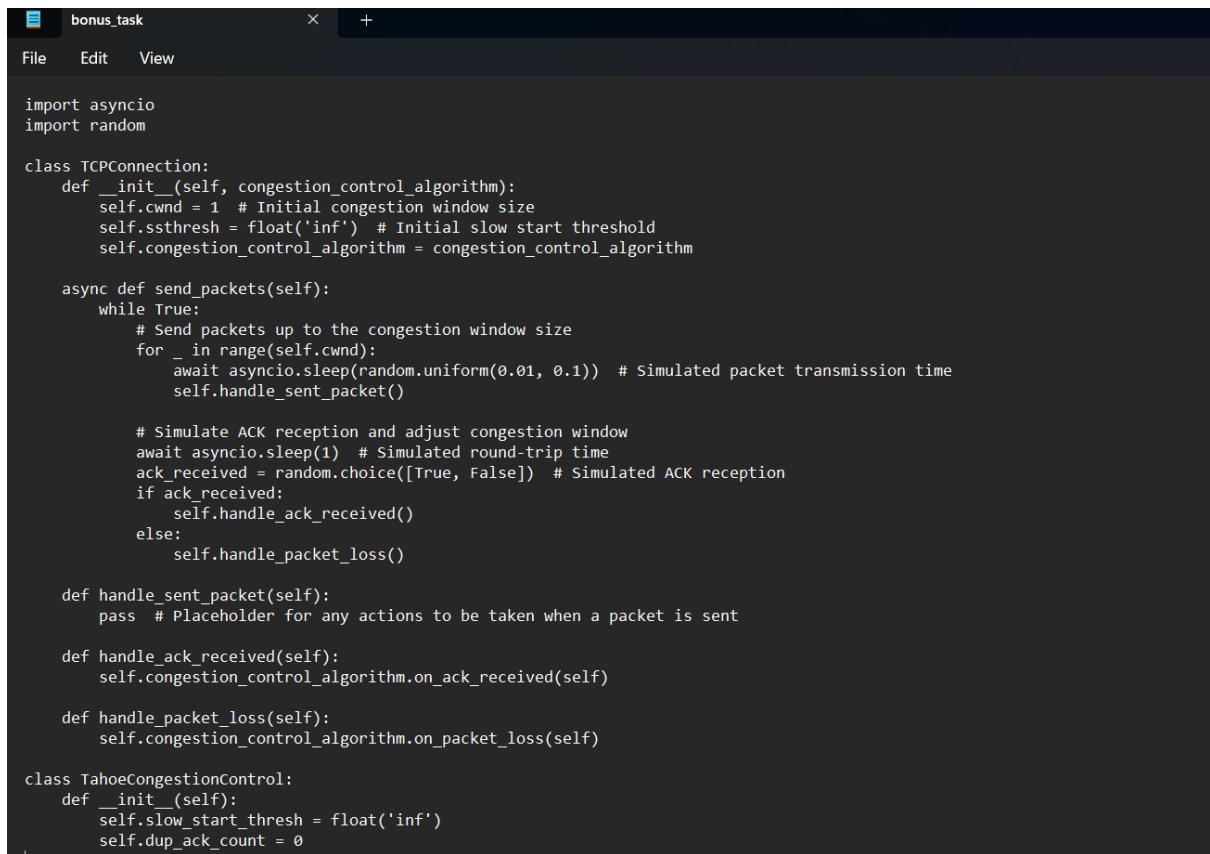


Group 16

CS420(G) Computer Networks Python programming Assignment part 1 of 2 Bonus task from Milestone 3

By Sushma Gajulapalli, Sathvika Kumbham, Aishwarya Khatpe, Meghana Kandala

***Write an algorithm in Python to demonstrate TCP congestion control.



The screenshot shows a code editor window titled "bonus_task". The menu bar includes "File", "Edit", and "View". The code itself is a Python script for a TCP connection. It defines a class `TCPConnection` with methods for sending packets, handling ACK reception, and handling packet loss. It also defines a subclass `TahoeCongestionControl` with its own initialization logic. The code uses `asyncio` and `random` modules.

```
import asyncio
import random

class TCPConnection:
    def __init__(self, congestion_control_algorithm):
        self.cwnd = 1 # Initial congestion window size
        self.ssthresh = float('inf') # Initial slow start threshold
        self.congestion_control_algorithm = congestion_control_algorithm

    async def send_packets(self):
        while True:
            # Send packets up to the congestion window size
            for _ in range(self.cwnd):
                await asyncio.sleep(random.uniform(0.01, 0.1)) # Simulated packet transmission time
                self.handle_sent_packet()

            # Simulate ACK reception and adjust congestion window
            await asyncio.sleep(1) # Simulated round-trip time
            ack_received = random.choice([True, False]) # Simulated ACK reception
            if ack_received:
                self.handle_ack_received()
            else:
                self.handle_packet_loss()

        def handle_sent_packet(self):
            pass # Placeholder for any actions to be taken when a packet is sent

        def handle_ack_received(self):
            self.congestion_control_algorithm.on_ack_received(self)

        def handle_packet_loss(self):
            self.congestion_control_algorithm.on_packet_loss(self)

class TahoeCongestionControl:
    def __init__(self):
        self.slow_start_thresh = float('inf')
        self.dup_ack_count = 0
```

```

class TahoeCongestionControl:
    def __init__(self):
        self.slow_start_thresh = float('inf')
        self.dup_ack_count = 0

    def on_ack_received(self, connection):
        if self.dup_ack_count == 3:
            self.handle_triple_dup_ack(connection)
        else:
            self.handle_regular_ack(connection)

    def on_packet_loss(self, connection):
        self.handle_packet_loss(connection)

    def handle_regular_ack(self, connection):
        connection.cwnd += 1
        if connection.cwnd >= self.slow_start_thresh:
            connection.cwnd += 1 # Congestion avoidance phase

    def handle_triple_dup_ack(self, connection):
        connection.ssthresh = max(connection.cwnd / 2, 2)
        connection.cwnd = 1
        self.dup_ack_count = 0

    def handle_packet_loss(self, connection):
        self.dup_ack_count += 1

async def main():
    congestion_control_algorithm = TahoeCongestionControl()
    connections = [TCPConnection(congestion_control_algorithm) for _ in range(5)] # Create multiple TCP connections

    # Start sending packets for each connection
    await asyncio.gather(*[connection.send_packets() for connection in connections])

asyncio.run(main())

```

Save the code in the filename as bonus_task.py

This code sets up a basic simulation of multiple TCP connections with simulates ACK reception and packet loss.

It defines a TCPConnection class managing congestion window and packet transmission, and a TahoeCongestionControl class handling ACK reception and packet loss events.

The main function creates multiple TCP connections with the Tahoe congestion control algorithm and starts sending packets asynchronously, simulating packet transmission, ACK reception, and packet loss based on random events and congestion control logic

The congestion control algorithm adjusts the congestion window size and slow start threshold based on the received ACKs and packet loss events.