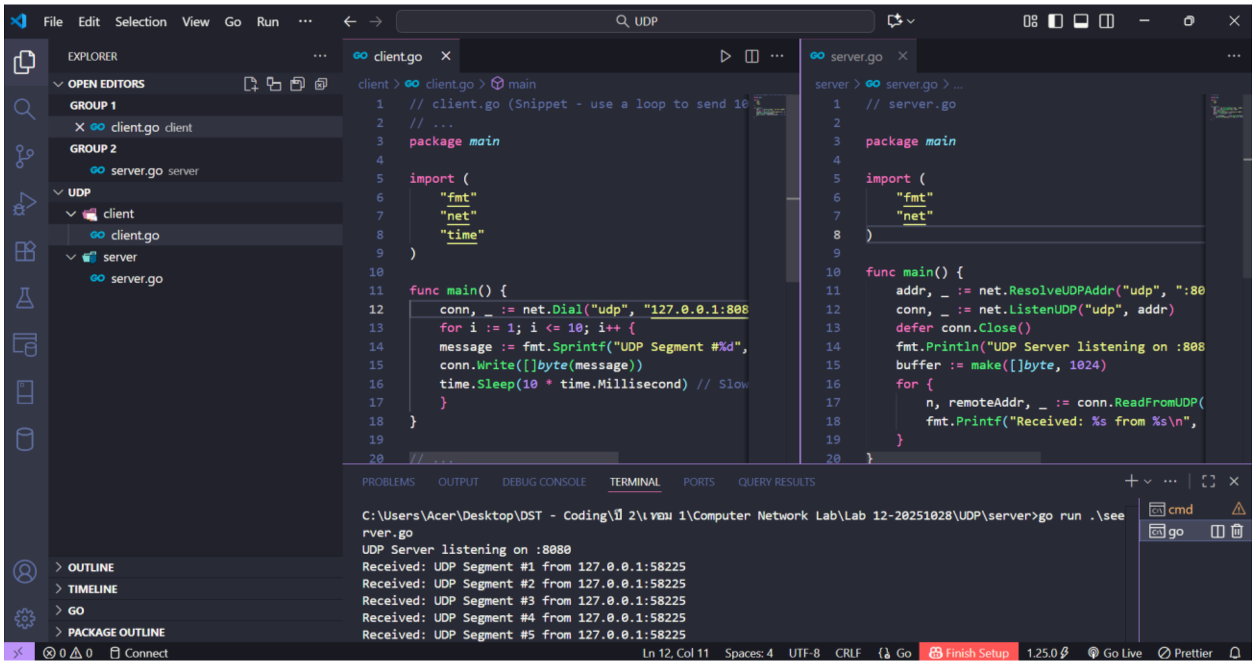
นายศักดิ์สิทธิ์ จิตตะโสภี 6787077 Section 2

1. Setup and Basic UDP Communication

Part 1: Setup and Basic UDP Communication

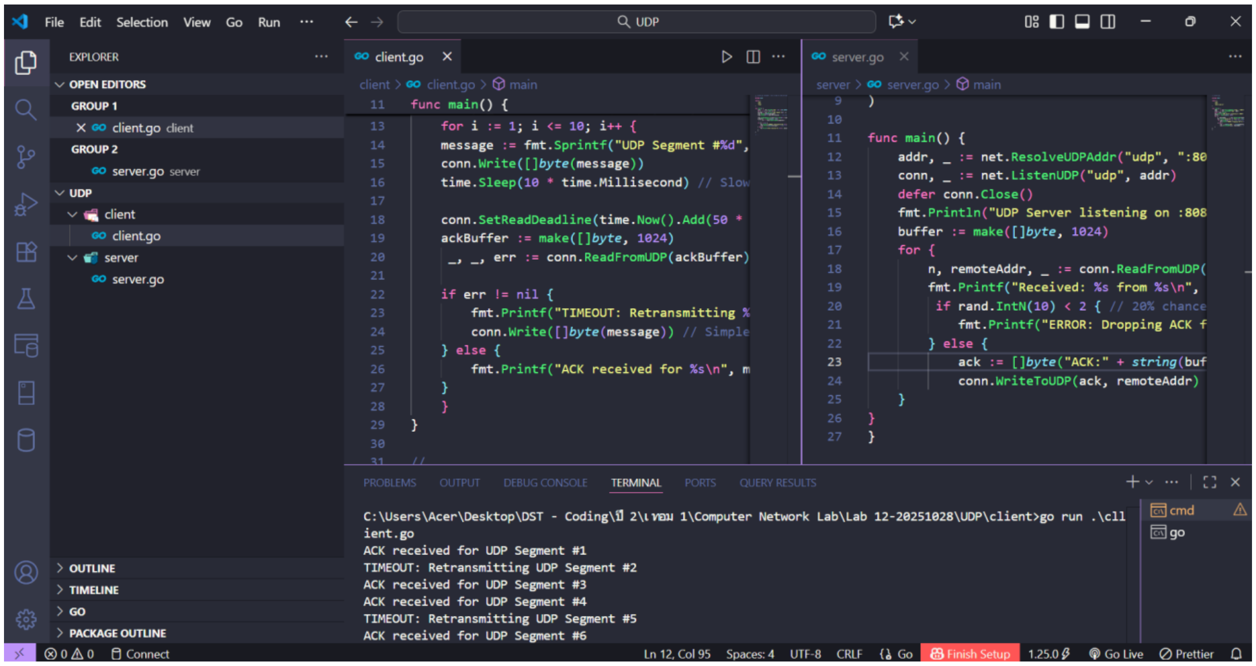


```
// client.go (Snippet - use a loop to send 10
// ...
package main

import (
    "fmt"
    "net"
    "time"
)

func main() {
    conn, _ := net.Dial("udp", "127.0.0.1:808
    for i := 1; i <= 10; i++ {
    message := fmt.Sprintf("UDP Segment #%d",
    conn.Write([]byte(message))
    time.Sleep(10 * time.Millisecond) // Slow
    }
}

// ...
```

```
// server.go

package main

import (
    "fmt"
    "net"
)

func main() {
    addr, _ := net.ResolveUDPAddr("udp", ":80
    conn, _ := net.ListenUDP("udp", addr)
    defer conn.Close()
    fmt.Println("UDP Server listening on :808
    buffer := make([]byte, 1024)
    for {
        n, remoteAddr, _ := conn.ReadFromUDP(
        fmt.Printf("Received: %s from %s\n",
    }
}
```

```
C:\Users\Acer\Desktop\DST - Coding\ปี 2\เทอม 1\Computer Network Lab\Lab 12-20251028\UDP\server>go run .\see
rver.go
UDP Server listening on :8080
Received: UDP Segment #1 from 127.0.0.1:58225
Received: UDP Segment #2 from 127.0.0.1:58225
Received: UDP Segment #3 from 127.0.0.1:58225
Received: UDP Segment #4 from 127.0.0.1:58225
Received: UDP Segment #5 from 127.0.0.1:58225
```

Part 2: Implementing Basic Error Control

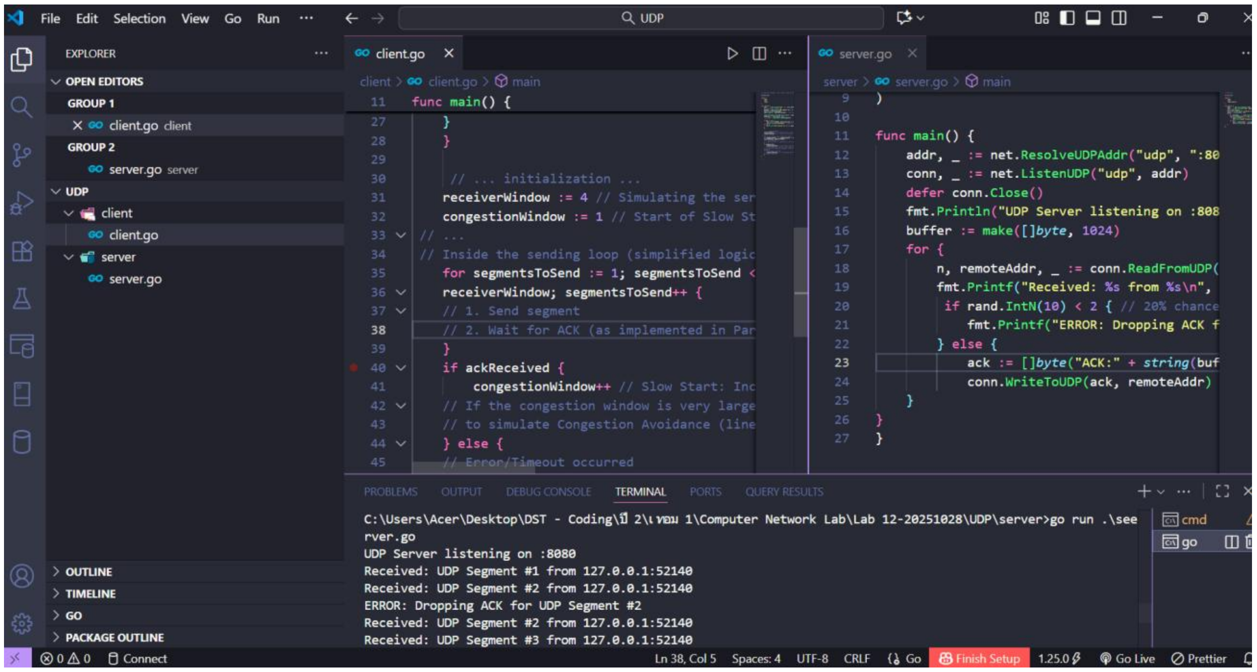


```
func main() {
    for i := 1; i <= 10; i++ {
    message := fmt.Sprintf("UDP Segment #%d",
    conn.Write([]byte(message))
    time.Sleep(10 * time.Millisecond) // Slow

    conn.SetReadDeadline(time.Now().Add(50 *
    ackBuffer := make([]byte, 1024)
     _, _, err := conn.ReadFromUDP(ackBuffer)

    if err != nil {
        fmt.Printf("TIMEOUT: Retransmitting %
        conn.Write([]byte(message)) // Simple
    } else {
        fmt.Printf("ACK received for %s\n", m
    }
    }
}

//
```

```
)

func main() {
    addr, _ := net.ResolveUDPAddr("udp", ":80
    conn, _ := net.ListenUDP("udp", addr)
    defer conn.Close()
    fmt.Println("UDP Server listening on :808
    buffer := make([]byte, 1024)
    for {
        n, remoteAddr, _ := conn.ReadFromUDP(
        fmt.Printf("Received: %s from %s\n",
         if rand.IntN(10) < 2 { // 20% chance
            fmt.Printf("ERROR: Dropping ACK f
        } else {
            ack := []byte("ACK:" + string(buf
            conn.WriteToUDP(ack, remoteAddr)
        }
    }
}
```

```
C:\Users\Acer\Desktop\DST - Coding\ปี 2\เทอม 1\Computer Network Lab\Lab 12-20251028\UDP\client>go run .\cl
ient.go
ACK received for UDP Segment #1
TIMEOUT: Retransmitting UDP Segment #2
ACK received for UDP Segment #3
ACK received for UDP Segment #4
TIMEOUT: Retransmitting UDP Segment #5
ACK received for UDP Segment #6
```

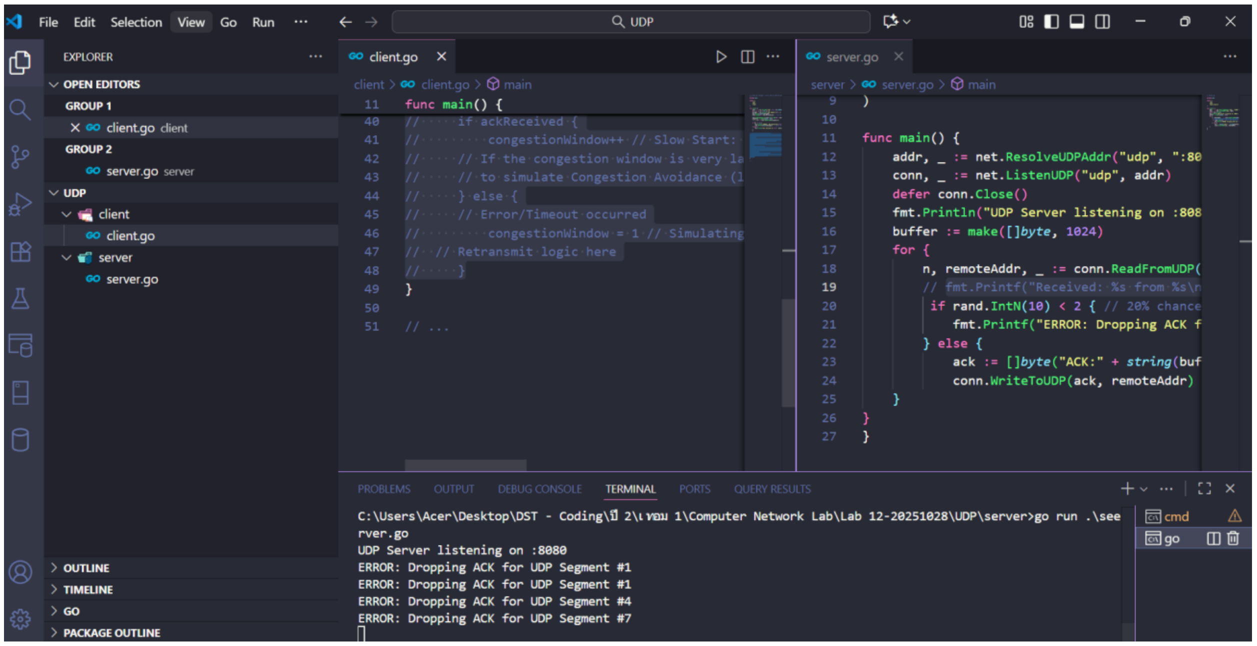
นายศักดิ์สิทธิ์ จิตตะโสภี 6787077 Section 2



```go
func main() {
        }
    }

     // ... initialization ...
    receiverWindow := 4 // Simulating the ser
    congestionWindow := 1 // Start of Slow St
// ...
// Inside the sending loop (simplified logic
    for segmentsToSend := 1; segmentsToSend <
    receiverWindow; segmentsToSend++ {
    // 1. Send segment
    // 2. Wait for ACK (as implemented in Par
    }
    if ackReceived {
        congestionWindow++ // Slow Start: Inc
    // If the congestion window is very large
    // to simulate Congestion Avoidance (line
    } else {
    // Error/Timeout occurred
```

```go
)

func main() {
    addr, _ := net.ResolveUDPAddr("udp", ":80
    conn, _ := net.ListenUDP("udp", addr)
    defer conn.Close()
    fmt.Println("UDP Server listening on :808
    buffer := make([]byte, 1024)
    for {
        n, remoteAddr, _ := conn.ReadFromUDP(
        fmt.Printf("Received: %s from %s\n",
         if rand.IntN(10) < 2 { // 20% chance
            fmt.Printf("ERROR: Dropping ACK f
        } else {
            ack := []byte("ACK:" + string(buf
            conn.WriteToUDP(ack, remoteAddr)
        }
    }
}
```

```
C:\Users\Acer\Desktop\DST - Coding\ปี 2\เทอม 1\Computer Network Lab\Lab 12-20251028\UDP\server>go run .\see
rver.go
UDP Server listening on :8080
Received: UDP Segment #1 from 127.0.0.1:52140
Received: UDP Segment #2 from 127.0.0.1:52140
ERROR: Dropping ACK for UDP Segment #2
Received: UDP Segment #2 from 127.0.0.1:52140
Received: UDP Segment #3 from 127.0.0.1:52140
```



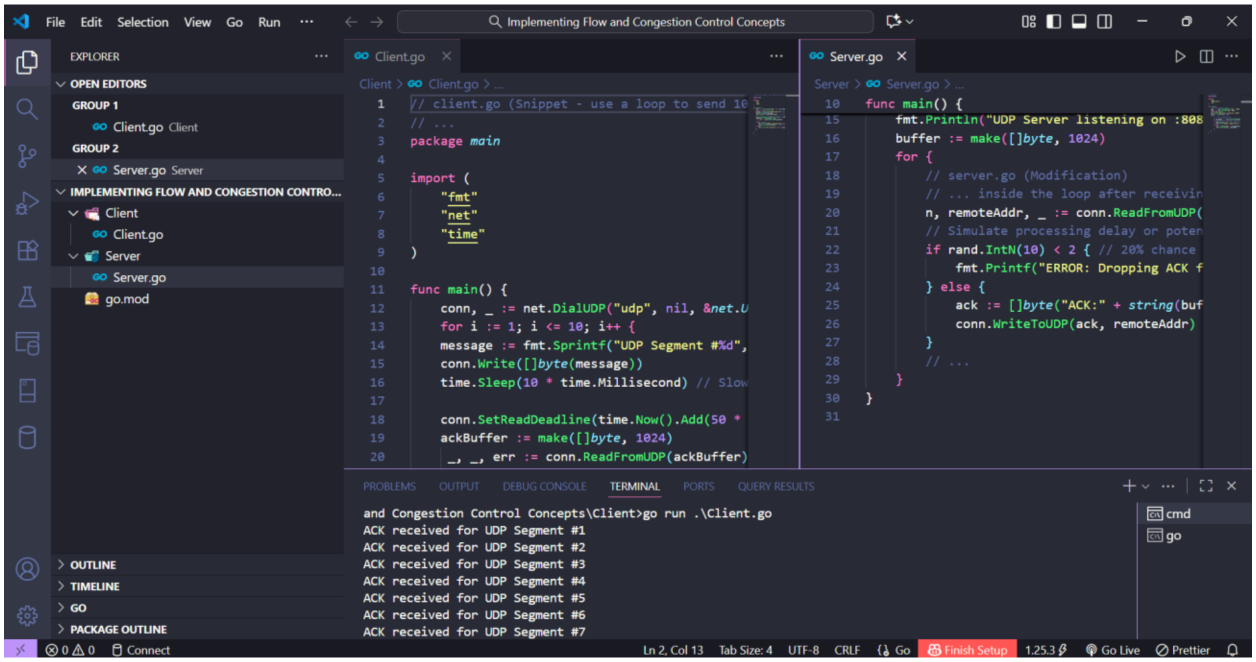
```go
func main() {
//     if ackReceived {
//         congestionWindow++ // Slow Start:
//     // If the congestion window is very la
//     // to simulate Congestion Avoidance (l
//     } else {
//     // Error/Timeout occurred
//         congestionWindow = 1 // Simulating
//   // Retransmit logic here
//     }
}

// ...
```

```go
)

func main() {
    addr, _ := net.ResolveUDPAddr("udp", ":80
    conn, _ := net.ListenUDP("udp", addr)
    defer conn.Close()
    fmt.Println("UDP Server listening on :808
    buffer := make([]byte, 1024)
    for {
        n, remoteAddr, _ := conn.ReadFromUDP(
        // fmt.Printf("Received: %s from %s\n
         if rand.IntN(10) < 2 { // 20% chance
            fmt.Printf("ERROR: Dropping ACK f
        } else {
            ack := []byte("ACK:" + string(buf
            conn.WriteToUDP(ack, remoteAddr)
        }
    }
}
```

```
C:\Users\Acer\Desktop\DST - Coding\ปี 2\เทอม 1\Computer Network Lab\Lab 12-20251028\UDP\server>go run .\see
rver.go
UDP Server listening on :8080
ERROR: Dropping ACK for UDP Segment #1
ERROR: Dropping ACK for UDP Segment #1
ERROR: Dropping ACK for UDP Segment #4
ERROR: Dropping ACK for UDP Segment #7
```

Part 3: Implementing Flow and Congestion Control Concepts

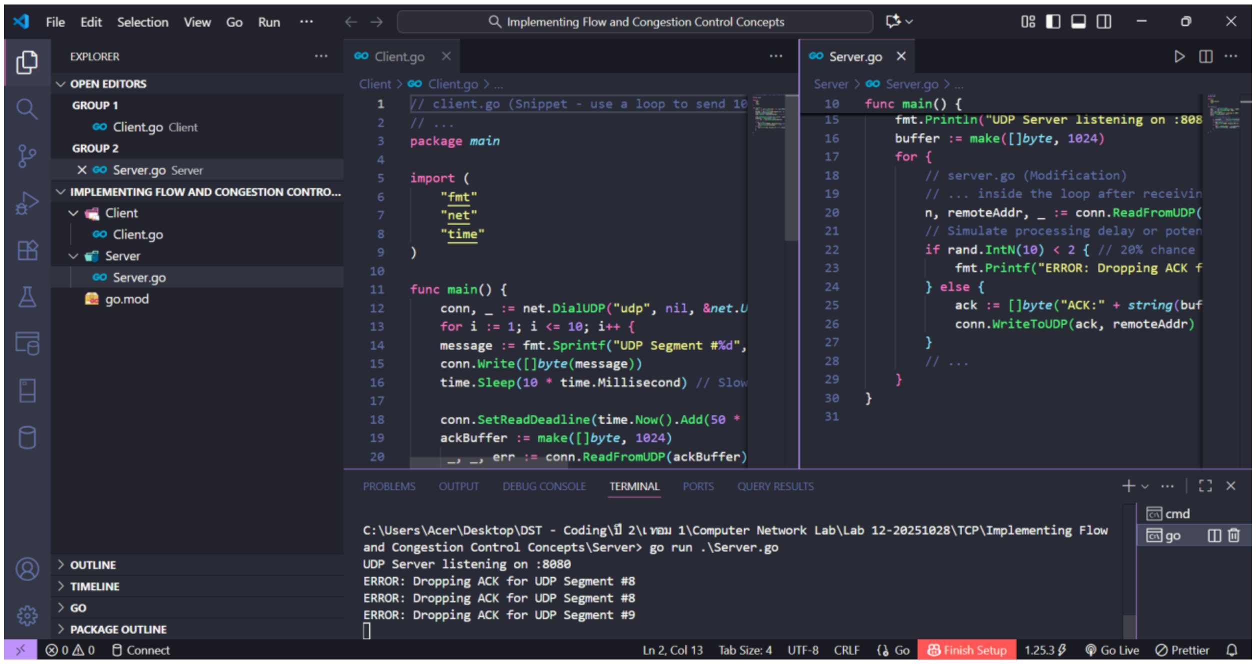
นายศักดิ์สิทธิ์ จิตตะโสภี 6787077 Section 2



```
// client.go (Snippet - use a loop to send 10
// ...
package main

import (
    "fmt"
    "net"
    "time"
)

func main() {
    conn, _ := net.DialUDP("udp", nil, &net.U
    for i := 1; i <= 10; i++ {
    message := fmt.Sprintf("UDP Segment #%d",
    conn.Write([]byte(message))
    time.Sleep(10 * time.Millisecond) // Slow

    conn.SetReadDeadline(time.Now().Add(50 *
    ackBuffer := make([]byte, 1024)
     _, _, err := conn.ReadFromUDP(ackBuffer)
```

```
func main() {
    fmt.Println("UDP Server listening on :808
    buffer := make([]byte, 1024)
    for {
        // server.go (Modification)
        // ... inside the loop after receivin
        n, remoteAddr, _ := conn.ReadFromUDP(
        // Simulate processing delay or poten
        if rand.IntN(10) < 2 { // 20% chance
            fmt.Printf("ERROR: Dropping ACK f
        } else {
            ack := []byte("ACK:" + string(buf
            conn.WriteToUDP(ack, remoteAddr)
        }
        // ...
    }
}
```

```
and Congestion Control Concepts\Client>go run .\Client.go
ACK received for UDP Segment #1
ACK received for UDP Segment #2
ACK received for UDP Segment #3
ACK received for UDP Segment #4
ACK received for UDP Segment #5
ACK received for UDP Segment #6
ACK received for UDP Segment #7
```



```
// client.go (Snippet - use a loop to send 10
// ...
package main

import (
    "fmt"
    "net"
    "time"
)

func main() {
    conn, _ := net.DialUDP("udp", nil, &net.U
    for i := 1; i <= 10; i++ {
    message := fmt.Sprintf("UDP Segment #%d",
    conn.Write([]byte(message))
    time.Sleep(10 * time.Millisecond) // Slow

    conn.SetReadDeadline(time.Now().Add(50 *
    ackBuffer := make([]byte, 1024)
     _, _, err := conn.ReadFromUDP(ackBuffer)
```

```
func main() {
    fmt.Println("UDP Server listening on :808
    buffer := make([]byte, 1024)
    for {
        // server.go (Modification)
        // ... inside the loop after receivin
        n, remoteAddr, _ := conn.ReadFromUDP(
        // Simulate processing delay or poten
        if rand.IntN(10) < 2 { // 20% chance
            fmt.Printf("ERROR: Dropping ACK f
        } else {
            ack := []byte("ACK:" + string(buf
            conn.WriteToUDP(ack, remoteAddr)
        }
        // ...
    }
}
```

```
C:\Users\Acer\Desktop\DST - Coding\ปี 2\เทอม 1\Computer Network Lab\Lab 12-20251028\TCP\Implementing Flow
and Congestion Control Concepts\Server> go run .\Server.go
UDP Server listening on :8080
ERROR: Dropping ACK for UDP Segment #8
ERROR: Dropping ACK for UDP Segment #8
ERROR: Dropping ACK for UDP Segment #9
```

Part 4: Analysis and Comparison

1. Protocol Comparison:

- **In your implementation, what specific line of code or logic provided Error Control?**

ส่วนของ **client** ที่ตรวจ **timeout** และส่งซ้ำ (**retransmit)** กับ **server** ที่ส่ง **ACK** กลับหลังรับข้อมูล

How did your simplified Flow Control prevent the client from overwhelming the receiver's buffer?

Flow Control ใช้ตัวแปร **rwnd** ที่ **server** ส่ง กลับมาเพื่อบอกว่ารับได้อีกเท่าไร
client จะจำกัดจำนวนแพ็กเก็ตที่ส่งค้าง (**inflight)** ไม่ให้เกิน **rwnd**
ทำให้ **server** ไม่ถูกส่งข้อมูลเข้ามามากเกินกว่าที่ **buffer** จะรับ

- **What was the core difference in behavior (speed vs. reliability) between the native UDP implementation (Part 1) and your custom TCP-like implementation (Parts 2 & 3)?**

UDP เดิมส่งข้อมูลได้เร็วกว่าเพราะไม่รอการตอบกลับ แต่ไม่รับประกันความถูกต้อง
ส่วนระบบที่ปรับให้คล้าย **TCP** จะช้ากว่าเพราะต้องรอ **ACK** และอาจต้องส่งซ้ำ
แต่ข้อดีคือข้อมูลถูกส่งถึงครบถ้วนและเรียงลำดับถูกต้อง

2. **TSAPs (Ports): Briefly explain where the concept of the Transport Service Access Point (TSAP) or Port is still essential in your Go programs, even though you used a simpler Dial function. (Hint: The address 127.0.0.1:8080 contains the port.)**

พอร์ต (**TSAP)** ยังคงจำเป็นในโปรแกรม **Go** เพราะใช้ระบุบริการที่ต้องการเชื่อมต่อ
เช่น **127.0.0.1:8080** หมายถึงโปรแกรม **server** ที่เปิดอยู่บนพอร์ต **8080**
ฝั่ง **client** และ **server** ต้องใช้พอร์ตเดียวกัน เพื่อให้ข้อมูลส่งถึงกันได้อย่างถูกต้อง