

1. Setup and Basic UDP Communication

Part 1: Setup and Basic UDP Communication

The screenshot shows the Visual Studio Code interface with two Go files: `client.go` and `server.go`. The `client.go` file contains a `main` function that sends 10 UDP segments to the server at `127.0.0.1:8080`. The `server.go` file contains a `main` function that listens on `:8080` and prints the received segments. The terminal output shows the server listening and receiving 5 segments.

```

// client.go
1 // client.go (Snippet - use a loop to send 10 segments)
2 // ...
3 package main
4
5 import (
6     "fmt"
7     "net"
8     "time"
9 )
10
11 func main() {
12     conn, _ := net.Dial("udp", "127.0.0.1:8080")
13     for i := 1; i <= 10; i++ {
14         message := fmt.Sprintf("UDP Segment #%d", i)
15         conn.Write([]byte(message))
16         time.Sleep(10 * time.Millisecond) // Slow down
17     }
18 }
19
20 // ...

// server.go
1 // server.go
2 // ...
3 package main
4
5 import (
6     "fmt"
7     "net"
8 )
9
10 func main() {
11     addr, _ := net.ResolveUDPAddr("udp", ":8080")
12     conn, _ := net.ListenUDP("udp", addr)
13     defer conn.Close()
14     fmt.Println("UDP Server listening on :8080")
15     buffer := make([]byte, 1024)
16     for {
17         n, remoteAddr, _ := conn.ReadFromUDP(buffer)
18         fmt.Printf("Received: %s from %s\n", string(buffer[:n]), remoteAddr)
19     }
20 }

```

Terminal Output:

```

C:\Users\Acer\Desktop\DST - Coding\12\Network 1\Computer Network Lab\Lab 12-20251028\UDP\server>go run .\server.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

The screenshot shows the Visual Studio Code interface with the same `client.go` and `server.go` files, but with error control implemented. The `client.go` file now includes a timeout and retransmission logic. The `server.go` file includes a 20% chance of dropping an ACK. The terminal output shows the client receiving ACKs and retransmitting segments due to timeouts.

```

// client.go
11 func main() {
12     for i := 1; i <= 10; i++ {
13         message := fmt.Sprintf("UDP Segment #%d", i)
14         conn.Write([]byte(message))
15         time.Sleep(10 * time.Millisecond) // Slow down
16
17         conn.SetReadDeadline(time.Now().Add(50 * time.Millisecond))
18         ackBuffer := make([]byte, 1024)
19         _, err := conn.ReadFromUDP(ackBuffer)
20
21         if err != nil {
22             fmt.Printf("TIMEOUT: Retransmitting %d\n", i)
23             conn.Write([]byte(message)) // Simple retransmission
24         } else {
25             fmt.Printf("ACK received for %s\n", message)
26         }
27     }
28 }
29
30 // ...

// server.go
9 )
10
11 func main() {
12     addr, _ := net.ResolveUDPAddr("udp", ":8080")
13     conn, _ := net.ListenUDP("udp", addr)
14     defer conn.Close()
15     fmt.Println("UDP Server listening on :8080")
16     buffer := make([]byte, 1024)
17     for {
18         n, remoteAddr, _ := conn.ReadFromUDP(buffer)
19         fmt.Printf("Received: %s from %s\n", string(buffer[:n]), remoteAddr)
20         if rand.IntN(10) < 2 { // 20% chance
21             fmt.Printf("ERROR: Dropping ACK\n")
22         } else {
23             ack := []byte("ACK:" + string(buffer[:n]))
24             conn.WriteToUDP(ack, remoteAddr)
25         }
26     }
27 }

```

Terminal Output:

```

C:\Users\Acer\Desktop\DST - Coding\12\Network 1\Computer Network Lab\Lab 12-20251028\UDP\client>go run .\client.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

```

The image displays two screenshots of a Go IDE (Visual Studio Code) showing the implementation of a UDP server and client with congestion control logic.

Top Screenshot:

- client.go:**

```

11 func main() {
27 }
28 }
29 }
30 // ... initialization ...
31 receiverWindow := 4 // Simulating the ser
32 congestionWindow := 1 // Start of Slow St
33 // ...
34 // Inside the sending loop (simplified logic
35 for segmentsToSend := 1; segmentsToSend <
36 receiverWindow; segmentsToSend++ {
37 // 1. Send segment
38 // 2. Wait for ACK (as implemented in Par
39 }
40 if ackReceived {
41 congestionWindow++ // Slow Start: Inc
42 // If the congestion window is very large
43 // to simulate Congestion Avoidance (line
44 } else {
45 // Error/Timeout occurred

```
- server.go:**

```

9 )
10 }
11 func main() {
12 addr, _ := net.ResolveUDPAddr("udp", ":80
13 conn, _ := net.ListenUDP("udp", addr)
14 defer conn.Close()
15 fmt.Println("UDP Server listening on :808
16 buffer := make([]byte, 1024)
17 for {
18 n, remoteAddr, _ := conn.ReadFromUDP(
19 fmt.Printf("Received: %s from %s\n",
20 if rand.IntN(10) < 2 { // 20% chance
21 fmt.Printf("ERROR: Dropping ACK f
22 } else {
23 ack := []byte("ACK:" + string(buf
24 conn.WriteToUDP(ack, remoteAddr)
25 }
26 }
27 }

```
- Terminal Output:**

```

C:\Users\Acer\Desktop\DST - Coding\2\1\1\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

```

Bottom Screenshot:

- client.go:**

```

11 func main() {
40 // ... if ackReceived {
41 // ... congestionWindow++ // Slow Start:
42 // ... If the congestion window is very la
43 // ... to simulate Congestion Avoidance (1
44 // ... } else {
45 // ... // Error/Timeout occurred
46 // ... congestionWindow = 1 // Simulating
47 // ... Retransmit logic here
48 // ... }
49 }
50 }
51 // ...

```
- server.go:**

```

9 )
10 }
11 func main() {
12 addr, _ := net.ResolveUDPAddr("udp", ":80
13 conn, _ := net.ListenUDP("udp", addr)
14 defer conn.Close()
15 fmt.Println("UDP Server listening on :808
16 buffer := make([]byte, 1024)
17 for {
18 n, remoteAddr, _ := conn.ReadFromUDP(
19 // fmt.Printf("Received: %s from %s\n
20 if rand.IntN(10) < 2 { // 20% chance
21 fmt.Printf("ERROR: Dropping ACK f
22 } else {
23 ack := []byte("ACK:" + string(buf
24 conn.WriteToUDP(ack, remoteAddr)
25 }
26 }
27 }

```
- Terminal Output:**

```

C:\Users\Acer\Desktop\DST - Coding\2\1\1\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

The screenshot shows the VS Code interface with two Go files open: `Client.go` and `Server.go`. The `Client.go` file contains a `main` function that sends 10 UDP segments. The `Server.go` file contains a `main` function that listens on port 8080 and processes incoming segments. The terminal output shows the following messages:

```
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
```

The screenshot shows the VS Code interface with the same Go files. The terminal output shows the following messages:

```
C:\Users\Acer\Desktop\DST - Coding\ปี 2\Lab 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** ไม่ถูกส่งข้อมูลเข้ามามากเกินไปกว่าที่ **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** ฝ่าย **client** และ **server** ต้องใช้พอร์ตเดียวกัน เพื่อให้ข้อมูลส่งถึงกันได้อย่างถูกต้อง