

## 1. Setup and Basic UDP Communication

### Part 1: Setup and Basic UDP Communication

```

client > client.go > main
1 // client.go (Snippet - use a loop to send 10 UDP 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
17     }
18 }
19
20 // ...

server > server.go > main
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.String())
19     }
20 }

C:\Users\Acer\Desktop\NST - Coding\Lab 2\Lab 1\Computer Network Lab\Lab 12-20251028\UDP>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
Received: UDP Segment #6 from 127.0.0.1:58225
Received: UDP Segment #7 from 127.0.0.1:58225
Received: UDP Segment #8 from 127.0.0.1:58225
Received: UDP Segment #9 from 127.0.0.1:58225
Received: UDP Segment #10 from 127.0.0.1:58225

```

### Part 2: Implementing Basic Error Control

```

client > client.go > main
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
16
17         conn.SetReadDeadline(time.Now().Add(50 * time.Second))
18         ackBuffer := make([]byte, 1024)
19         _, err := conn.ReadFromUDP(ackBuffer)
20
21         if err != nil {
22             fmt.Println("TIMEOUT: Retransmitting %d", i)
23             conn.Write([]byte(message)) // Simple
24         } else {
25             fmt.Println("ACK received for %d", i)
26         }
27     }
28 }
29
30 // ...

server > server.go > main
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
18     for {
19         n, remoteAddr, _ := conn.ReadFromUDP(buffer)
20         if rand.Intn(10) < 2 { // 20% chance
21             fmt.Println("ERROR: Dropping ACK for %d", n)
22         } else {
23             ack := []byte("ACK:" + string(buffer[:n]))
24             conn.WriteToUDP(ack, remoteAddr)
25         }
26     }
27 }

C:\Users\Acer\Desktop\NST - Coding\Lab 2\Lab 1\Computer Network Lab\Lab 12-20251028\UDP>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

```

## นายศักดิ์ลักษณ์ จิตตะโน๊ก 6787077 Section 2

The image displays two side-by-side screenshots of a code editor interface, likely Visual Studio Code, showing Go code for a UDP client and server. Both screenshots show the same two files: client.go and server.go.

**Top Screenshot (client.go logic commented out):**

```
client > client.go > main
11 func main() {
27     }
28     }
29     // ... initialization ...
30     receiverWindow := 4 // Simulating the ser
31     congestionWindow := 1 // Start of Slow St
32     // ...
33     // Inside the sending loop (simplified logic
34     for segmentsToSend := 1; segmentsToSend <
35         receiverWindow; segmentsToSend++ {
36             // 1. Send segment
37             // 2. Wait for ACK (as implemented in Par
38             if ackReceived {
39                 congestionWindow++ // Slow Start: Inc
40                 // If the congestion window is very large
41                 // to simulate Congestion Avoidance (line
42                 } else {
43                     // Error/Timeout occurred
44                 }
45             }
46             // ...
47         }
48     }
49 }
```

```
server > server.go > main
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         if rand.IntN(10) < 2 { // 20% chance
20             fmt.Printf("Received: %s from %s\n",
21             string(buffer[:n]), remoteAddr.String())
22         } else {
23             ack := []byte("ACK:" + string(buffer[:n]))
24             conn.WriteToUDP(ack, remoteAddr)
25         }
26     }
27 }
```

**Bottom Screenshot (client.go logic un-commented):**

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

```
server > server.go > main
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",
20         if rand.IntN(10) < 2 { // 20% chance
21             fmt.Println("ERROR: Dropping ACK for UDP Segment #1")
22         } else {
23             ack := []byte("ACK:" + string(buffer[:n]))
24             conn.WriteToUDP(ack, remoteAddr)
25         }
26     }
27 }
```

**Terminal Output (Top Screenshot):**

```
C:\Users\Acer\Desktop\NST - Coding\第 2\例題 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
```

**Terminal Output (Bottom Screenshot):**

```
C:\Users\Acer\Desktop\NST - Coding\第 2\例題 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

```

Client > Client.go ...
1 // client.go (Snippet - use a loop to send 10 UDP segments)
2 // ...
3 package main
4
5 import (
6     "fmt"
7     "net"
8     "time"
9 )
10
11 func main() {
12     conn, _ := net.DialUDP("udp", nil, &net.UDPAddr{Port: 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         conn.SetReadDeadline(time.Now().Add(50 * time.Second))
19         ackBuffer := make([]byte, 1024)
20         _, err := conn.ReadFromUDP(ackBuffer)

```

```

Server > Server.go ...
10 func main() {
11     fmt.Println("UDP Server listening on :8080")
12     buffer := make([]byte, 1024)
13     for {
14         // server.go (Modification)
15         // ... inside the loop after receiving
16         n, remoteAddr, _ := conn.ReadFromUDP(
17             buffer, nil)
18         // Simulate processing delay or potential error
19         if rand.Intn(10) < 2 { // 20% chance
20             fmt.Printf("ERROR: Dropping ACK for UDP Segment %d\n", n)
21         } else {
22             ack := []byte("ACK:" + string(buffer))
23             conn.WriteToUDP(ack, remoteAddr)
24         }
25     }
26 }
27
28
29
30
31

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

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

Ln 2, Col 13 Tab Size: 4 UTF-8 CRLF ⌂ Go ⌂ Finish Setup 1.25.3 ⌂ Go Live ⌂ Prettier

```

Client > Client.go ...
1 // client.go (Snippet - use a loop to send 10 UDP segments)
2 // ...
3 package main
4
5 import (
6     "fmt"
7     "net"
8     "time"
9 )
10
11 func main() {
12     conn, _ := net.DialUDP("udp", nil, &net.UDPAddr{Port: 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         conn.SetReadDeadline(time.Now().Add(50 * time.Second))
19         ackBuffer := make([]byte, 1024)
20         _, err := conn.ReadFromUDP(ackBuffer)

```

```

Server > Server.go ...
10 func main() {
11     fmt.Println("UDP Server listening on :8080")
12     buffer := make([]byte, 1024)
13     for {
14         // server.go (Modification)
15         // ... inside the loop after receiving
16         n, remoteAddr, _ := conn.ReadFromUDP(
17             buffer, nil)
18         // Simulate processing delay or potential error
19         if rand.Intn(10) < 2 { // 20% chance
20             fmt.Printf("ERROR: Dropping ACK for UDP Segment %d\n", n)
21         } else {
22             ack := []byte("ACK:" + string(buffer))
23             conn.WriteToUDP(ack, remoteAddr)
24         }
25     }
26 }
27
28
29
30
31

```

C:\Users\Acer\Desktop\NST - Coding\II 2\week 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 #8

Ln 2, Col 13 Tab Size: 4 UTF-8 CRLF ⌂ Go ⌂ Finish Setup 1.25.3 ⌂ Go Live ⌂ Prettier

## Part 4: Analysis and Comparison

### 1. Protocol Comparison:

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

ส่วนของ client ที่ติดต่อ 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 เดิมส่งข้อมูลได้เร็วกว่า เพราะไม่รอการตอบกลับ แต่ไม่รับประกันความถูกต้อง ส่วนระบบที่ปรับให้คล้าย TCP จะช้ากว่า เพราะต้องรอ 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 เพราะใช้ระบบรับบริการที่ต้องการเชื่อมต่อ เช่น 127.0.0.1:8080 หมายถึงโปรแกรม server ที่เปิดอยู่บนพอร์ต 8080 ผู้client และ server ต้องใช้พอร์ตเดียวกัน เพื่อให้ข้อมูลส่งถึงกันได้อย่างถูกต้อง