TCP, UDP : transport layer protocol

network layer : host 跟 host 的溝通

transport layer : process 跟 process 的溝通 (架在 network layer

- TCP : congestion control (擁塞), flow control, connection setup
- UDP : no-frill extension of best-offer
- service not available : delay guarantees, bandwidth guarantees

QoS : 1. delay guarantees 2. bandwidth guarantees

3. delay jitter (抖動) 4. loss rate

Multiplexing / demultiplexing

Multiplexing : handle data from multiple socket (at sender)

demultiplexing : use header info to deliver received segment to correct socket (at receiver)

How demultiplexing works :

① source IP address ② destination IP address ← this two are 32 bits

③ source port number ④ destination port number ← this two are 16 bits

UDP 只看 destination IP and Port ; TCP 四個都要看

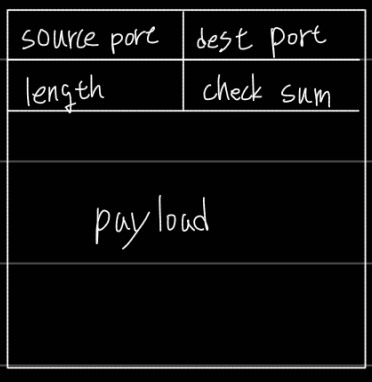
UDP：

base-effort：不做任何保證

用在 Streaming，DNS，SNMP (simple network management protocol)

速度超快

header：



length：in bytes of UDP segment (include header)

check sum：把整個封包切成 16 bits 後加起來(端迴進位)

之後做 1's，到3接收端再切一次，(包括checksum)加起來不是全1就代表err

Automatic Repeat Request (ARQ)

Base on Re-transmission

Reliable data transfer (rdt)

1. No data corruption
2. No data lost
3. In order delivery (照順序傳)

use finite state machine

(↓這是教授的原話)(洗勒公ㄙㄤ)

rdt-send(data)：把上面要傳送的資料，傳送成一個Packet

↳ Encapsulation：把上層傳來的 data 加 Transport layer 的 header

rdt-rcv(packet)：接收

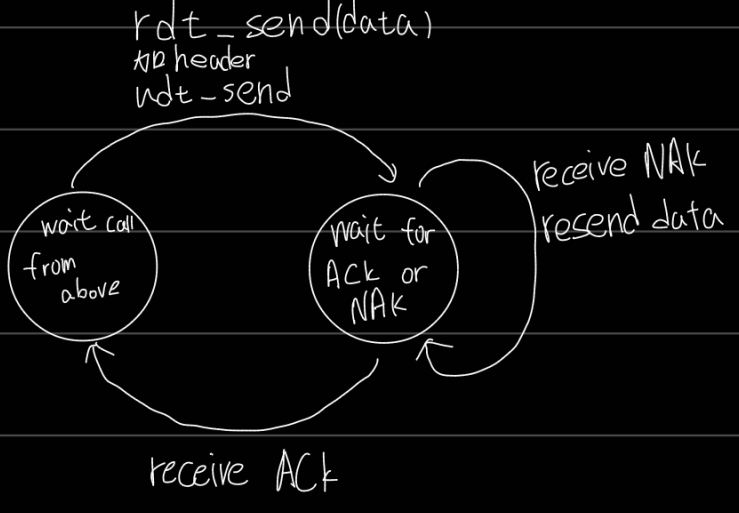
→ Extract：把 header 拆下來順便 Checksum (if any)

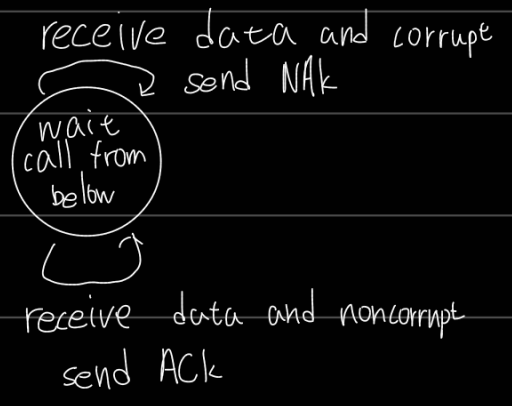
→ Delivery：把 data 傳上去

rdt 2.0：channel with bit errors → use stop-and-wait protocal 等 feedback 回來才傳下一個

checksum can detect errors

Question：how to recover them → Retransmission (ARQ)





rdt2.0 缺點：ACK 會壞，如果 ACK 變成 NAk 會重傳一次 (Duplicate)

→ 在封包裡面塞一個 seq# (number)，接收端發現後 discard

---

rdt 2.1 ⇒ 不知道他在供三小

---

rdt 2.2 ⇒ NAK-free protocal ⇒ only ACK, no NAK

Sender 的 seq# 是 0→1→0→1 在輪的，Receiver 回傳的 ACK 會包含收到的 seq#

如果收到的有問題就回傳上一次的，Sender 收到 ACK 再透過 seq# 判斷

rdt
- 1.0 ⇒ no pkg corruption, no loss
- 2.0 ⇒ pkg corruption, no loss
- 2.1 ⇒ pkg, ACK, NAK corr…, no loss
- 2.2 ⇒ pkg, ACK corr…, no loss

---

rdt 3.0 ⇒ pkg, ACK will corrupt and loss ⇒ stop and wait

加一個 Timer, Time out 就重傳

Receiver 發現序列號 duplicate 就丟棄

Premature timeout ⇒ 網路擁塞導致 timeout, 會有兩個 duplicate

→ 使用 checksum, ACK, Re-Transmission, Seq#, Timer

e.g. 1Gbps network, delay 15ms, 8000 bits data, calculate

$$\frac{L}{R} = \frac{8\times10^{3}\ \text{bits}}{10^{9}\ \text{bits/sec}} = 8\times10^{-6}\ \text{sec}$$

$$U = \frac{L/R}{RTT + L/R} = \frac{0.008}{15\times2 + 0.008} = 0.00027$$

Pipelined Protocal

- go-Back-N ⇒ 設 k 個 timeout, k 之後 N 個全部重傳
- selective repeat ⇒

go-Back-N

1、一開始送N個 (按照順序的才會動作)

2、每收到一個ACK，N後移一位＋傳最新一個

3、receiver如果收到的包有跳號，傳回上一個 seq#，並且丟掉最新的

4、sender ignore duplicate ACK，如果有timeout 把N內的全部重傳

Selective repeat

1、每個封包都要設Timer

2、receiver收到跳號的就先塞到buffer並傳回ACK，直到跳號的收到再一起傳到上層

3、At sender, which Timeout will be resend

4、Sender一樣有N (sliding window) 的結構

e.g. 2掉包，3、4、5 ok，當2的ACK回來後 N會一次跳到6789 (會記錄ACK的意思)

---

go-Back-N ⇒ 簡單，沒效率，不佔資源

selective repeat ⇒ complex，效率好，Timer多到爆

TCP

Point-to-point, reliable, pipelined, full duplex, connection-oriented

flow controled

URG: urgent data (generally not use)

ACK: ACK # valid

PSH: push data now (generally not use)

RST, SYN, FIN (connection estab)
- setup
- teardown
- commands

source port #	dest port #
sequence number	
acknowledgement number	
head len / Not used / U A P R S F	receive window
checksum	Urg data pointer
options (variable length)	
application data (variable length)	

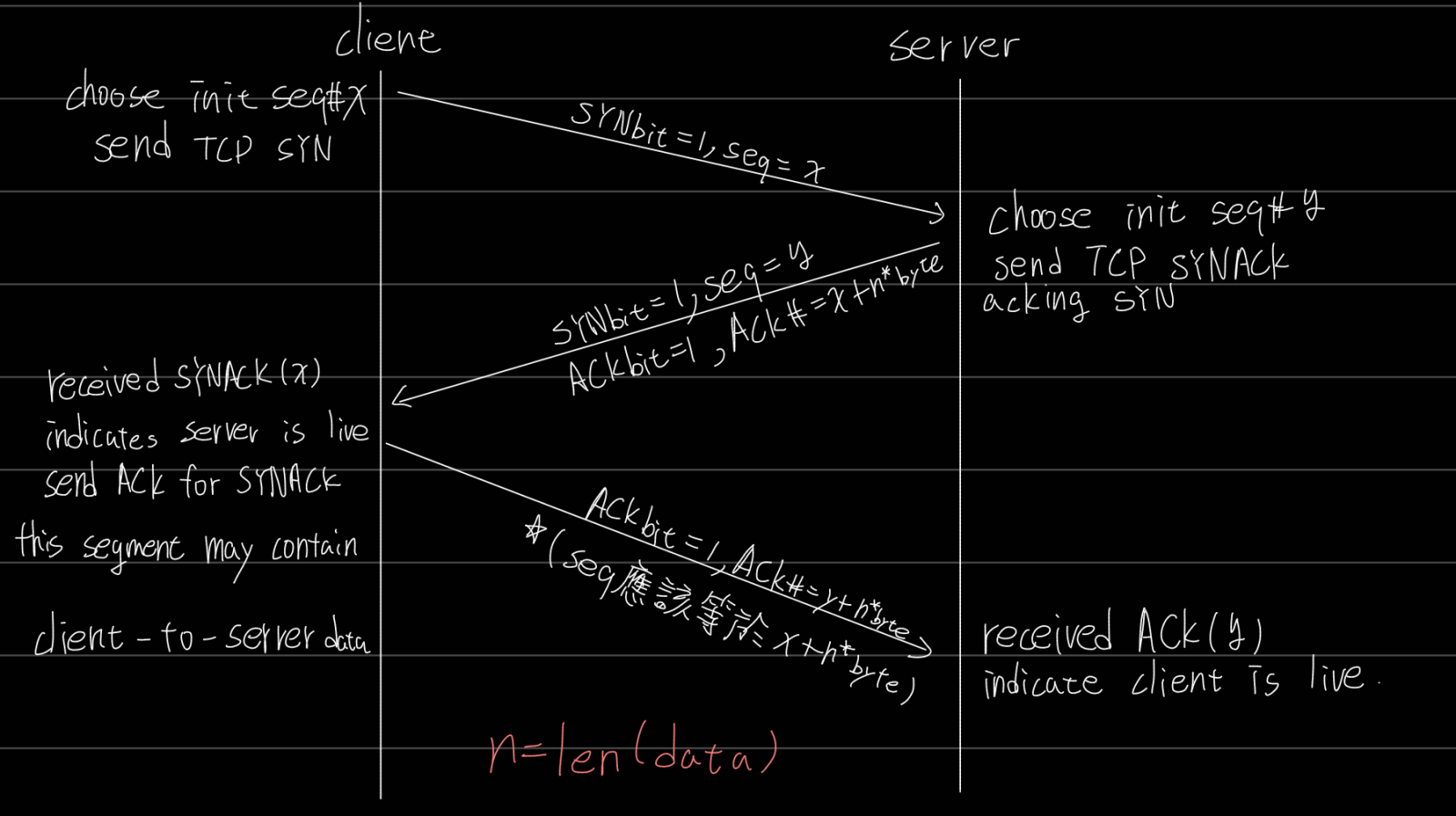
Counting by bytes of data (not segment)

bytes rcvr willing to accept

ACK: 預期下一個接收的byte count (maybe > 1)

Seq: 傳了第幾個byte, 開始於 random initial value (避免攻擊)

TCP 3-Way Handshaking



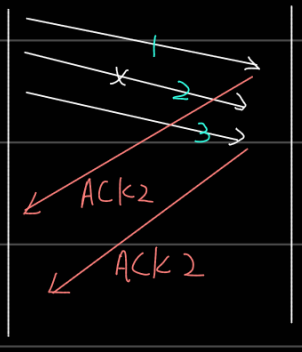
TCP RTT (Round-Trip Time)

RTT too short : premature time out

too long : slow reaction

Time > RTT + 4 dev , will retransmission

TCP rdt
- Pipelined Segments
- Cumulative Acks
- Single timer



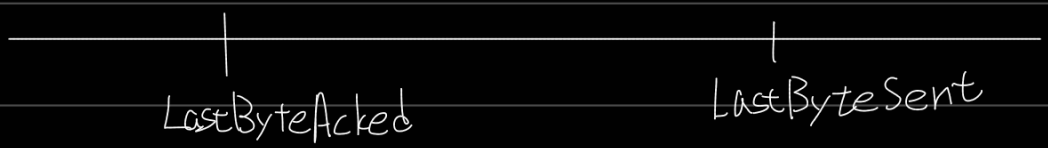
TCP retransmission triggered by : 1. timeout 2. duplicate ACKs (3)

TCP flow control : Avoid overflow

receiver

rwnd = RcvBuffer − (LastByteReceive − LastByteRead)

(receiver window)



LBS - LBA < rwnd

Congestion：懒得抄了，去看彭老师的