

# FCH1

## 基本定義

### Internet

**host:** 終端架構(像是手機電腦等等透過應用介面上網) 包括client跟server

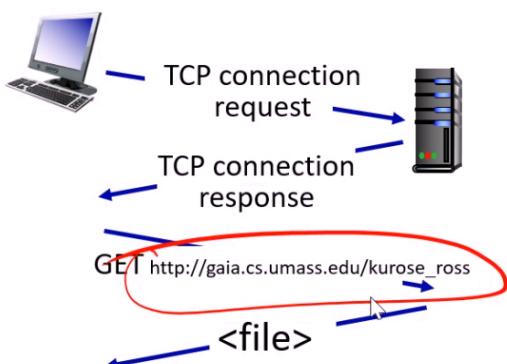
**Packet switch:** 透過路由器 交換器傳送資料



路由器 交換器符號

**protocol(通訊協定):** 規定了數據在計算機網絡中的傳輸和通信方式。

**HTTP連線:** 先建立TCP通道, 再傳送資料



**路由器 (Router):**

**Forwarding:** 通過檢查封包的目的地地址, 然後根據路由表等信息, 將封包送往正確的目的地

**Routing:** 使用路由算法 (routing algorithms) 來確定每個封包的最佳路徑

1. **網絡層設備:** 路由器工作在網絡的第三層(網絡層), 負責在不同的網絡之間進行數據包的路由和轉發。它連接不同的網絡, 例如連接區域網(LAN)和互聯網。
2. **跨網絡通信:** 主要用於將數據包從一個網絡傳遞到另一個網絡, 通過查找目標IP地址來確定路由。因此, 它在不同的網絡之間實現了連接。
3. **NAT和防火牆:** 路由器通常支持網絡地址轉換(NAT)功能, 允許多個區域網設備共享一個公共IP地址。它們也通常包括防火牆功能, 用於網絡安全。
4. **廣域網(WAN)連接:** 通常用於連接到廣域網服務提供商(ISP)的設備, 使家庭或企業能夠訪問互聯網。

**交換機 (Switch):**

1. **數據鏈路層設備:** 交換機工作在數據鏈路層, 負責在同一個區域網(LAN)內部的設

備之間交換和轉發數據幘。

2. **區域網內部通信**: 主要用於區域網內的設備之間的通信，通過學習和查找MAC地址來確定數據幘的傳遞。
3. **高性能區域網**: 交換機通常具有高性能和低延遲，使得區域網內的數據傳輸更加高效和可靠。
4. **不涉及網絡層路由**: 交換機通常不涉及跨越不同網絡的路由功能，它們主要處理同一個網絡內部的數據流量。

IETF 代表 "Internet Engineering Task Force"，它是一個國際性的、自願參與的組織，專注於互聯網協議的開發和標準化。他們通過協作、討論和共同製定 RFC。RFC 是一種用於定義互聯網技術和協議的標準文檔

## Sends packets of data(傳輸資料時間計算)

封包大小為L(bit)

傳輸速率為R(單位bps)

$$\text{packet transmission delay} = \frac{\text{time needed to transmit } L\text{-bit packet into link}}{R \text{ (bits/sec)}} = \frac{L \text{ (bits)}}{R \text{ (bits/sec)}}$$

計算傳送時間(packet transmission delay)則為L/R

**Store and forward**: 資料先被(router)完整地接收、儲存再轉送到下一個目的地

**queueing delay**: 在路由器中儲存等待傳送的時間

**loss**: 在路由器中儲存已經到達上限而繼續傳送資料則會丟失

## Packet switching

不會建立連接(connection setup/teardown)

不會預留頻寬(bandwidth reservation)

透過路由表routing path selection

會有queue的等待延遲問題，將封包切成數個，可支援較多人用  
人多的時候會卡due to store and forward，所以會掉大量的封包

## circuit switching

會建立連接(connection setup/teardown)

會預留頻寬(bandwidth reservation)

會為通信的兩端分配和保留專用的傳輸資源，

這種方式確保了通信資源的專用性，不存在資源的共享

不會有queueing delay

特點包括：

專用資源：每個呼叫會分配到特定的電路資源，保證了通信的性能。

無資源共享：電路交換中的資源不共享，每個電路僅供一個呼叫使用。

電路空閒時段：如果電路未被呼叫使用，它將保持空閒，不會被其他呼叫共享。

## FDM&TDM

FDM為用頻寬分割(同時處理多個但頻寬較小),

TDM為用時間分割(可以用較大速率/頻寬傳遞)

都可以拿來分割Channel

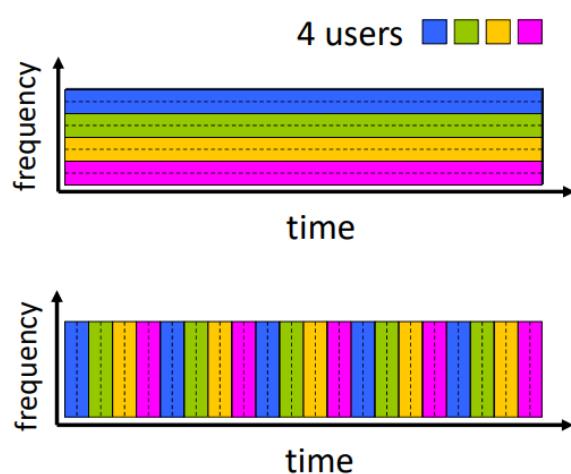
## Circuit switching: FDM and TDM

### Frequency Division Multiplexing (FDM)

- optical, electromagnetic frequencies divided into (narrow) frequency bands
- each call allocated its own band, can transmit at max rate of that narrow band

### Time Division Multiplexing (TDM)

- time divided into slots
- each call allocated periodic slot(s), can transmit at maximum rate of (wider) frequency band, but only during its time slot(s)



Introduction: 1-30

計算機網路通常是用Packet switching

計算circuit switching時為total link/個人分配的 link(故為1Gb/100Mb)

計算有n位使用機率則用二項式

$$\binom{40}{n} p^n (1-p)^{40-n}.$$

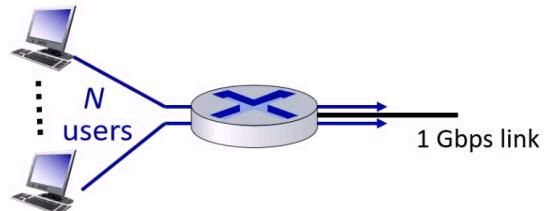
假設總人數40 有n個人在使用

## Packet switching versus circuit switching

*packet switching allows more users to use network!*

Example:

- 1 Gb/s link
- each user:
  - 100 Mb/s when “active”
  - active 10% of time
- *circuit-switching: 10 users*
- *packet switching: with 35 users, probability > 10 active at same time is less than .0004 \**
- *packet switching: with 35 users, probability > 10 active at same time is less than .0004 \**



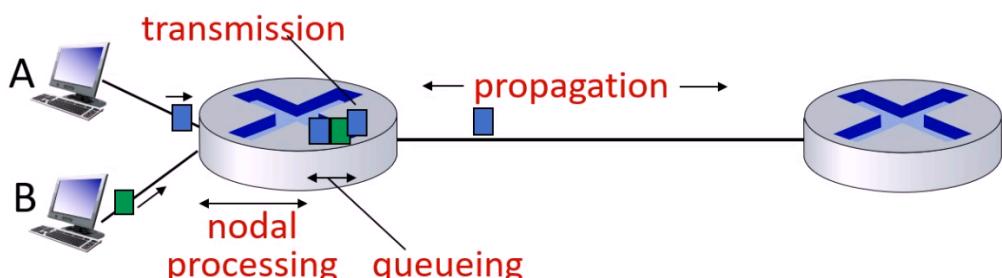
Q: how did we get value 0.0004?

Q: what happens if > 35 users ?

Q: how did we get  
Q: what happens

$$\sum_{k=1}^{35} C^{35}_k \cdot 0.1^k \cdot 0.9^{35-k}$$

## Packet delay: four sources



$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

nodal processing 確認資料正確性的時間(看硬體狀況決定)

queueing delay 資料等待傳輸的時間(無法預測)

transmission delay 傳送時間(資料大小L/傳輸速率R)

propagation 傳輸時間(跟介質有關)(距離 D/ 傳輸速率 S)

高速(HIGH SPEED)網路和寬頻(broadband)網路是一樣的

高速網路並不是每秒公尺跑比較多跑比較快。

他是指R比較大(跑速是固定的)。 $(\sim 2 \times 10^8 \text{ m/sec})$

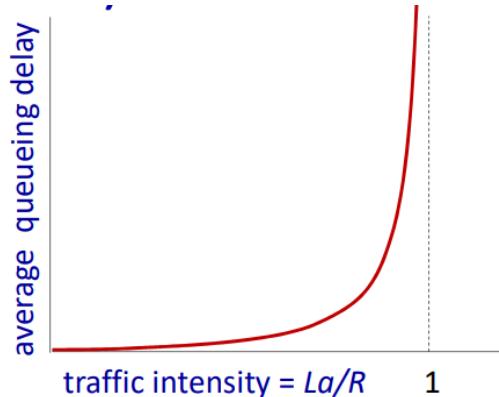
## Packet queueing delay (revisited)

被傳輸速率(bps) 封包大小(bits) 平均每秒來幾個封包所影響

越靠近1則delay會越高

超過1則為無限大

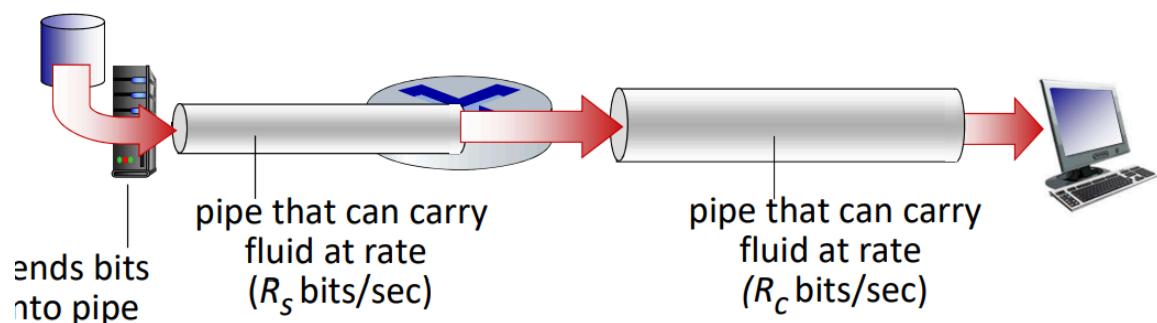
- $R$ : link bandwidth (bps)
- $L$ : packet length (bits)
- $a$ : average packet arrival rate
- $La/R \sim 0$ : avg. queueing delay small
- $La/R \rightarrow 1$ : avg. queueing delay large
- $La/R > 1$ : more “work” arriving is more than can be serviced - average delay infinite!



## Throughput(吞吐量)

loading: 從發送方向接收方發送bit的速率 (bit/sec)

throughput: 接收方能接受的有效bit的速率 (bit/sec)



average: rate over longer period of time。

簡單來說，RS和RC比較小的那個是決定這平均值的關鍵。

也可以當瞬間的傳輸速率。

下層為上層服務(socket)

同層之間會有一個協定來做協調。

## Internet protocol five stack(網路通訊協定5層重要)TCP/IP model

application(應用): 支援網路應用

- IMAP, SMTP, HTTP

運輸: 過程-過程之間的資料傳輸

- TCP, UDP

網路: 資料封包從來源到目的地的路由<3

- IP, routing protocols

鏈路(link): 相鄰網路元素之間的資料傳輸

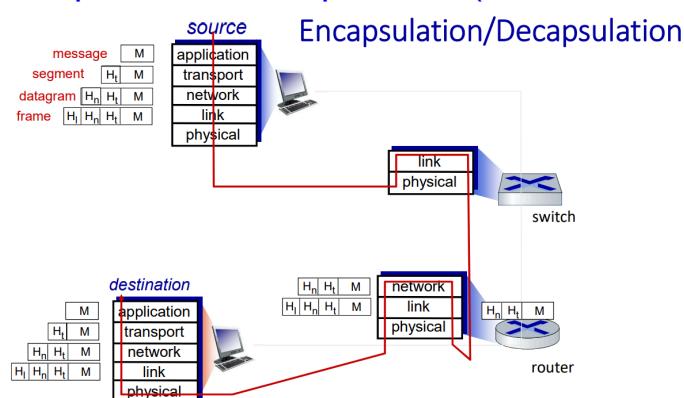
- Ethernet, 802.11 (WiFi), PPP

physical(物理層): bits “on the wire”(ps:不知道怎麼翻)

像是傳輸線或是光纖等實際物理傳輸

ISO/OSI: 應用層分割多出兩層presentation(表示)session(會議)(但現在主要用TCP/IP)

### Encapsulation/Decapsulation(封裝/解封裝)



source會先進行封裝->switch跟router解封看完資料會在封裝傳送->destination最後在解封

### Client-server

server ip固定且不關機, client ip會變動且間歇性地連接

Client的傳輸永遠對server, client跟client之間通常不會直接傳輸

p2p可同時扮演兩個 相對client-server比較不會有效能問題

### TCP

1. 發送和接收進程之間的可靠傳輸。
2. 流量控制: 發送方不會淹沒接收方。
- 3.擁塞控制: 網路過載時限制發送者。
- 4.面向連線: 客戶端和伺服器進程之間需要設定。
- 5.不提供: 定時、最小吞吐量保證、安全性。

### UDP

1. 發送和接收進程之間的資料傳輸不可靠。

2.不提供:可靠性、流量控制、擁塞控制、定時、吞吐量保證、安全性或連接設定。

## 傳輸層安全性 (TLS)

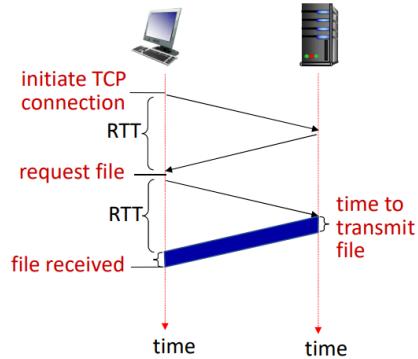
在應用層加密 不涵蓋在TCP(EX:https)

1.提供加密的 TCP 連接

2.資料的完整性

3.端點認證

## Non-persistent HTTP(1.0)& persistent HTTP(1.1)



Non-persistent without parallel  $\Rightarrow (1+N\text{個object}) * 2\text{RTT} = (1+N)*2 \text{ RTT}$

Non-persistent with parallel  $\Rightarrow 2(\text{建連線 要求網頁}) + 2\text{RTT}(\text{建連線 要求N個物件}) = 4 \text{ RTT}$

persistent without pipelining function  $\Rightarrow 2(\text{建連線 要求網頁}) + N(\text{要求N個物件}) = 2+N \text{ RTT}$   
(因為效果比Non-persistent with parallel差故不用)

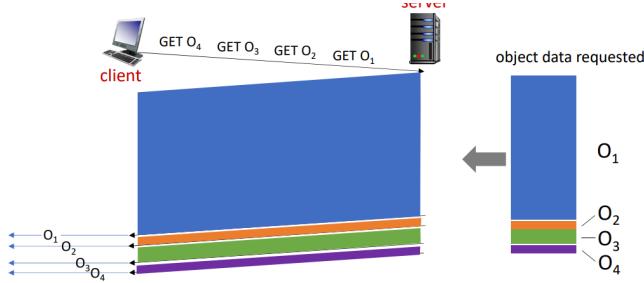
persistent with pipelining function  $\Rightarrow 2(\text{建連線 要求網頁}) + 1 \text{ RTT}(\text{要求N個物件}) = 3\text{RTT}$

HTTP1.0(Non-persistent):傳完資料會斷掉連線

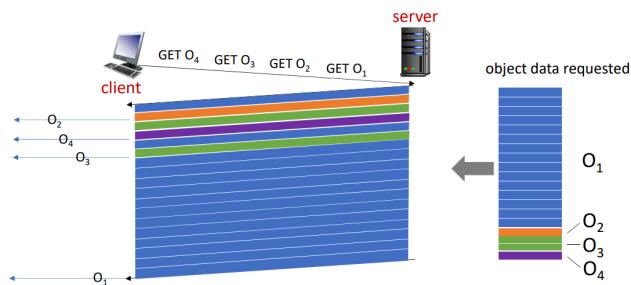
HTTP1.1(persistent):傳完資料不會斷掉連線，直到傳完。

HTTP/2:透過pipeline的方式來傳

但因為1會按照順序來傳,但如果第一個檔案太大的話,後面小檔案要排很久。



HTTP/2: objects divided into frames, frame transmission interleaved



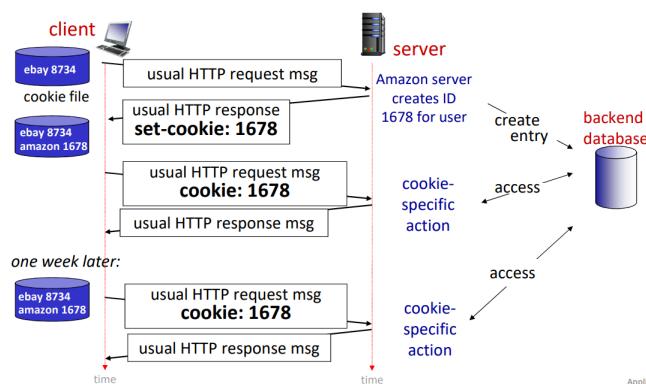
*O<sub>2</sub>, O<sub>3</sub>, O<sub>4</sub> delivered quickly, O<sub>1</sub> slightly delayed*

所以會將它分成好幾個封包輪流傳送

HTTP/3:因為傳送失敗還是會塞車所以,新增了UDP並整合錯誤偵測跟一些tcp功能

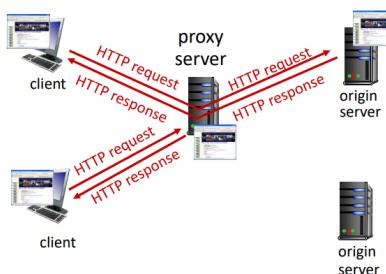
(congestion control)和security

cookies



第一次拜訪網站時,server會問你要不要cookie,如果你接受了就會給你一個ssn,他會去記錄說你訪問過哪個網站...。有點像資料庫的支援多個資料視角。

## Web caches (proxy servers)

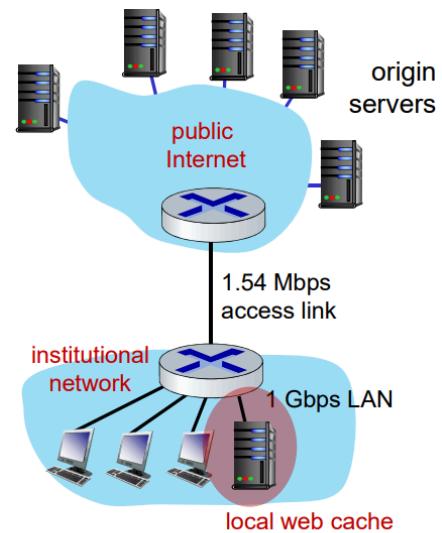


client訪問一個網站時會先去看proxy server有沒有紀錄,有的話就不用去sever端訪問,沒有則需要。這可以加快速度。

# Caching example: install a web cache

## Calculating access link utilization, end-end delay with cache:

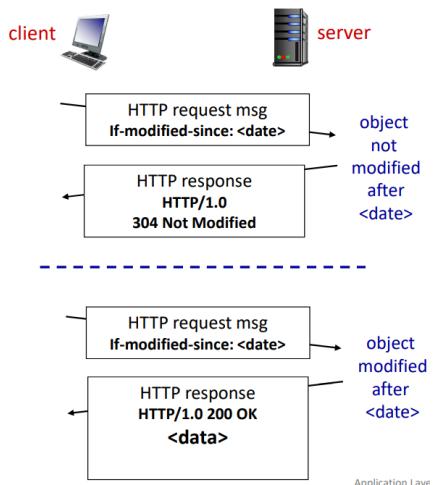
- suppose cache hit rate is 0.4: 40% requests satisfied at cache, 60% requests satisfied at origin
- access link: 60% of requests use access link
- data rate to browsers over access link  
 $= 0.6 * 1.50 \text{ Mbps} = .9 \text{ Mbps}$
- utilization =  $0.9 / 1.54 = .58$
- average end-end delay  
 $= 0.6 * (\text{delay from origin servers}) + 0.4 * (\text{delay when satisfied at cache})$   
 $= 0.6 (\sim 2.0 \text{ secs}) + 0.4 (\sim \text{msecs}) = \sim 1.2 \text{ secs}$



*lower average end-end delay than with 154 Mbps link (and cheaper too!)*

Applicati

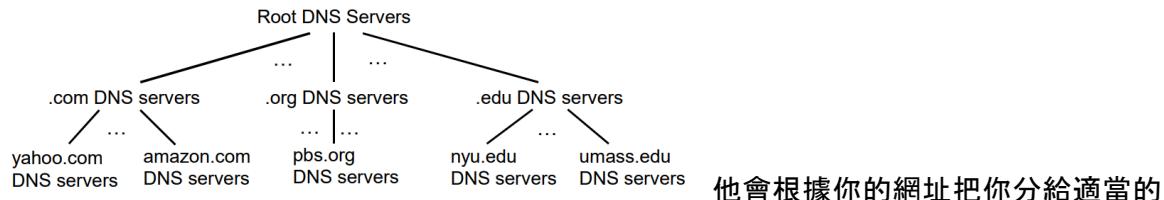
## 範例



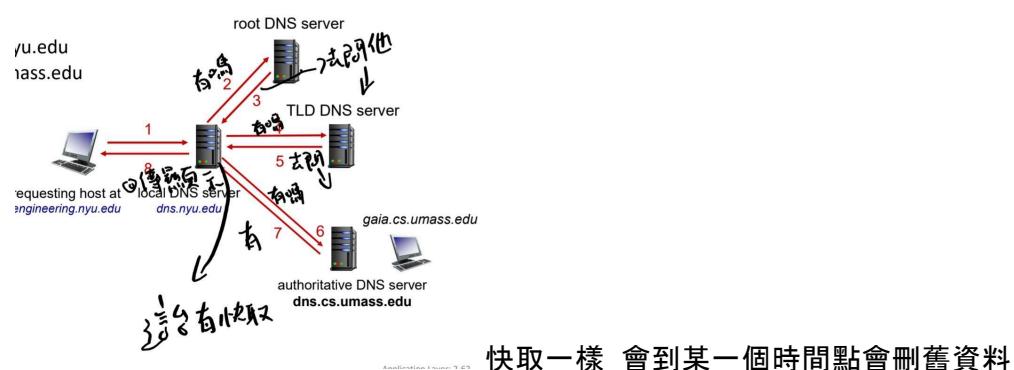
cached會問有沒有修改過資料。沒有的話就不用重傳了。有的話就重傳。

## DNS: Domain Name System(UDP)

主要就是將讀取的網址名稱轉換成IP

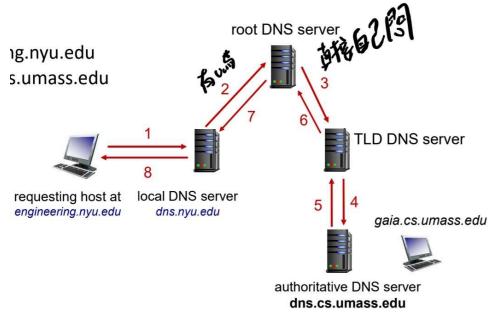


## DNS name resolution: iterated query(主要用這個)



## DNS name resolution: recursive query

## Iteration: recursive query



## DNS records

**DNS:** distributed database storing resource records (**RR**)

**RR format:** (name, value, type, ttl)

### type=A

- name is hostname
- value is IP address

### type=NS

- name is domain (e.g., foo.com)
- value is hostname of authoritative name server for this domain

### type=CNAME

- name is alias name for some “canonical” (the real) name
- www.ibm.com is really severeast.backup2.ibm.com
- value is canonical name

### type=MX

- value is name of mailserver associated with name

## P2P

### BitTorrent: requesting, sending file chunks

- 1.使用者稱為peer沒有分sever跟client
- 2.每筆檔案會被分割成256kb chunks
- 3.要求檔案會先要求最少peer有的chunk
- 4.給檔案時若有多人要求則先選擇給過自己檔案的peer(互利)  
但也會每30秒隨機選一個peer發送

## CH3

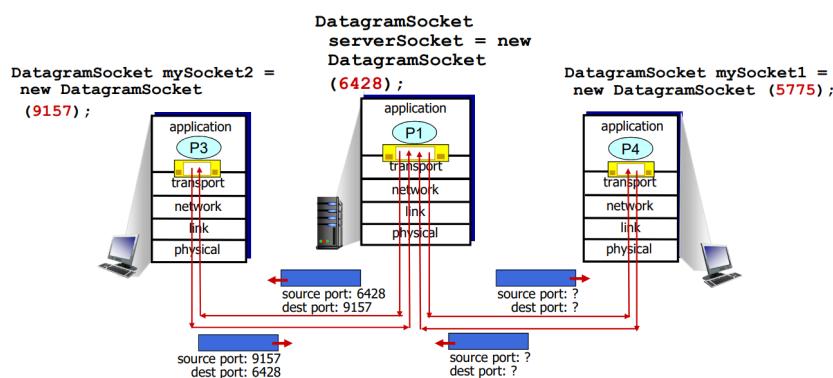
### multiplexing

發送端將很多不同的socket透過傳輸層(例如tcp)整合在一起送出, 利用header的方式

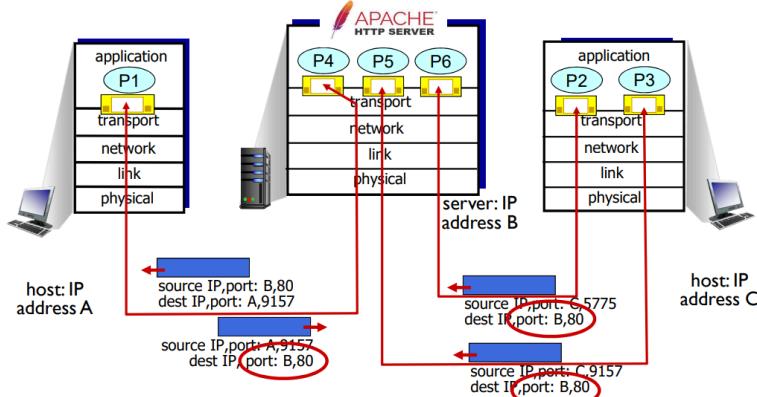
### demultiplexing

接收端利用ip adress & port number將很多不同的socket做分類給不同process

UDP: demultiplexing using destination port number (only)



TCP: demultiplexing using 4-tuple: source and destination IP addresses, and port numbers



## UDP checksum

detect errors

example: add two 16-bit integers

wraparound	$  \begin{array}{r}  1\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0 \\  1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \\  \hline  1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1  \end{array}  $
sum	$1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0$
checksum	$0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 1$

UDP:8byte

TCP:20byte

## rdt1.0: 在可靠通道上的可靠傳輸

步驟

- |                   |                 |
|-------------------|-----------------|
| 1.在沒有任何錯誤的情況      | 在沒有任何錯誤的情況      |
| 2.sender等待指令傳資料   | receiver等待指令收資料 |
| 3.收到指令後增加header發送 | 收到指令後移除header   |
| 4.保持原狀態           | 保持原狀態           |

## rdt2.0: 具有位元錯誤的通道

### 1.stop and wait

sender傳完封包會等待receiver回

acknowledgements (ACKs): 接收端確認訊息正確回復發送端OK

negative acknowledgements (NAKs, 只有rdt2.1有): 接收端發現錯誤訊息回復發送端ERROR

### 2.handling duplicates(重傳):

sender adds sequence number to each pkt

步驟:

- |                      |                 |
|----------------------|-----------------|
| 1.sender等待指令傳資料      | receiver等待指令收資料 |
| 2.收到指令後增加header發送    | 收到指令後移除header   |
| 3.等待對方向回覆ACK或NAK(重來) | 回覆ACK或NAK(重新等待) |
| 4.回復原狀態              | 保持原狀態           |

## rdt2.1 如果 ACK/NAK被損壞

必須檢查接收到的 ACK/NAK 是否被損壞。

需要兩倍的狀態數量

狀態必須“記住”“預期”封包應該具有序列號 0 或 1

處理:

1.sender會重新傳送當前的封包

2.sender會為每個封包添加sequence number(用一個bit欄位即可)

2.receive會丟棄重複的封包。

## rdt2.2 使用僅限 ACK 來實現與 rdt2.1 相同的功能, 不使用 NAK:

只回復最新的正確data的seq #不會發送NAK

如果發送端接收到重複的 ACK, 則採取與 NAK 相同的操作: 重新傳送當前的封包

TCP使用這種方法

## rdt3.0 :在面對具有錯誤和封包遺失

超時重傳 (Timeout and Retransmission)

sender在發送封包後等待一段「合理」的時間,

如果在時間內沒有收到ACK, 就認為封包丟失了或者確認遭遇了延遲, 然後進行重新傳送

### U sender(發送端的利用率)

U sender越高, 表示發送端在傳送數據方面的效率越高,

花費大多時間傳輸資料, 花費少的時間在等待資料

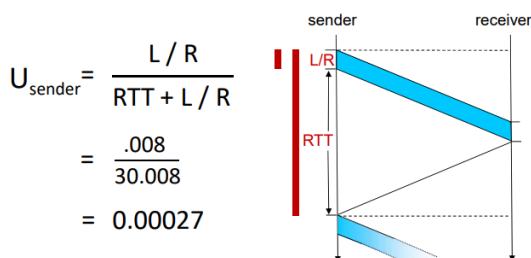
- $U_{\text{sender}}$ : utilization – fraction of time sender busy sending

- example: 1 Gbps link, 15 ms prop. delay, 8000 bit packet

- time to transmit packet into channel:

$$D_{\text{trans}} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microsecs}$$

## rdt3.0: stop-and-wait operation



- if RTT=30 msec, 1KB pkt every 30 msec: 33kB/sec thruput over 1 Gbps link

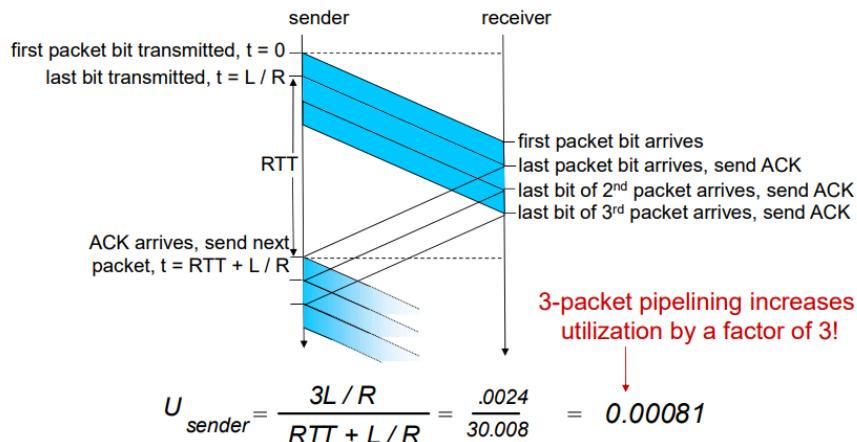
- rdt 3.0 protocol performance stinks!

- Protocol limits performance of underlying infrastructure (channel)

## Pipelining: increased utilization

可以同時間發送多個封包，不用等待封包的回覆

接收端可以在接收到封包後立即發送確認，不需要等待上一個確認的封包的處理

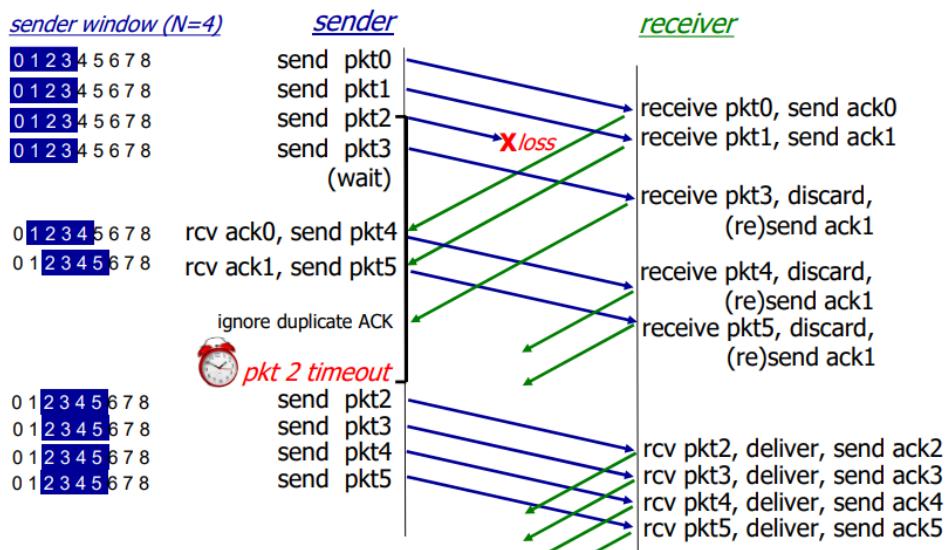


## GBN(Go-Back-N: sender)

1. 每個封包都有各自的Sequence Numbers(序列號)
2. 發送端會有一個固定大小(發送上限)的window(裡面封包全部都發送等待確認)
3. 只有最早的封包有計時器，要重發直接從緩存(window裡)取用
4. 如果收到了正確順序的確認(ACK)則window向後滑動
5. 傳送失敗window則會回推到失敗的資料

接收端總是發送 **ACK**，確認已經接收的封包，並指示已經接收且按照順序的最高序列號  
下圖中的2傳送失敗因此window回退到seq #2重新傳送

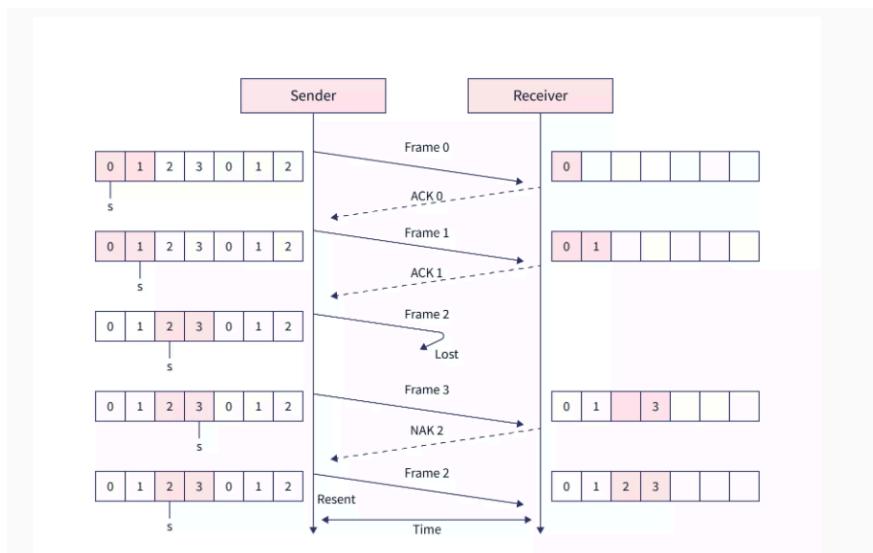
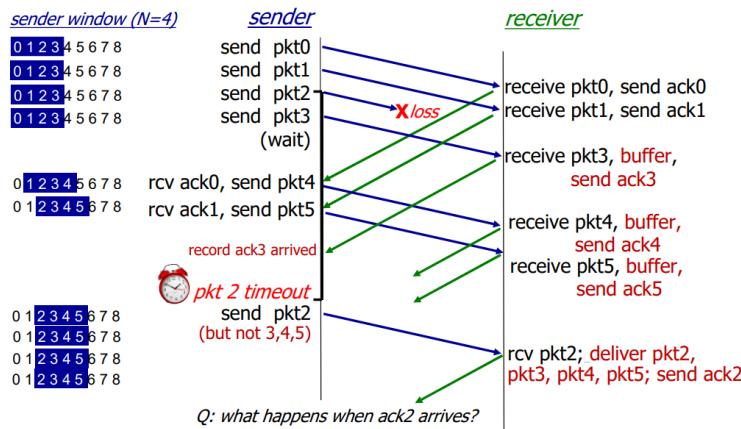
## Go-Back-N in action



## Selective Repeat

1. sender只重送接收端未正確收到的封包
2. 會將排序不正確的封包暫存
3. 傳送端、接收端雙方的 window位置各自不同
4. 每個封包都綁有 timer, 當各個封包的 ACK 逾時未收到, 則重送該封包
5. window大小必須等於有限序號大小的一半

## Selective Repeat in action

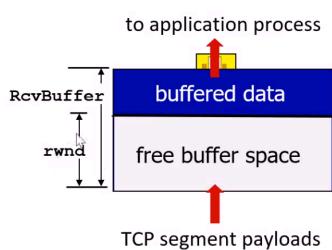


## flow control

透過tcp中的receive window

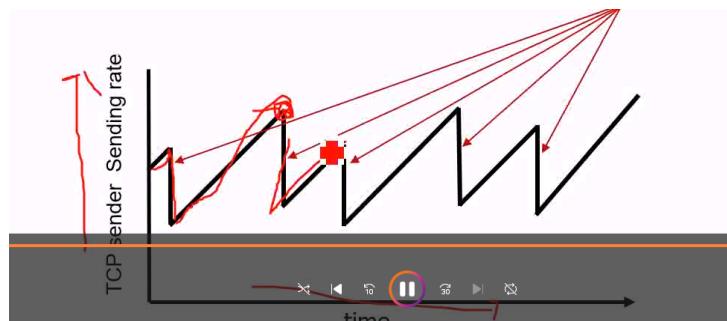
receiver控制sender, 避免sender傳輸速度過快, 導致receiver的緩衝區溢出

receivera滿時, 可以動態配置buffer大小增加封包儲存的空間, 並降低sender傳輸量



## TCP congestion control: AIMD

每一個RTT會去看線路有沒有塞車  
如果沒有塞車的話會嘗試增加一個封包傳輸量  
直到下次塞車後砍半



**考古題** 哈囉你好嗎，衷心感謝，珍重再見，期待再相逢

是誰住在深海的大鳳梨裡，海綿寶寶

1. 比較電路和分組交換策略在以下方面的主要特徵/差異

**(a) connection setup/teardown**

circuit: 會建立連接(connection setup/teardown)

packet: 不會建立連接(connection setup/teardown)

**(b) bandwidth reservation**

circuit: 會預留頻寬(bandwidth reservation)

packet: 不會預留頻寬(bandwidth reservation)

**(c) routing path selection**

circuit: 建立好連線，傳資料就經過相同路由器

packet: 透過路由表選擇封包傳送路徑

**(d) queueing delay**

circuit: 建立連線後，不用等待時間直接到目的地

packet: 人多的時候會卡因為store and forward, 需要排隊

**(e) channel efficiency**

circuit: 建立好連線後，及4~沒有傳送訊息會一直保持連線

packet: 會一直按照順序傳送直到沒有資料

2. What are the five layers in the Internet protocol stack? Describe briefly the main responsibilities of each layer?

application(應用層): (支援網路應用)supporting network applications

- IMAP, SMTP, HTTP

transport(傳輸層): (process到process之間的資料傳送)process-process data transfer

- TCP, UDP

network(網路層): (負責處理在不同網絡之間的數據傳輸 路由選擇)routing of datagrams from source to destination

- IP, routing protocols

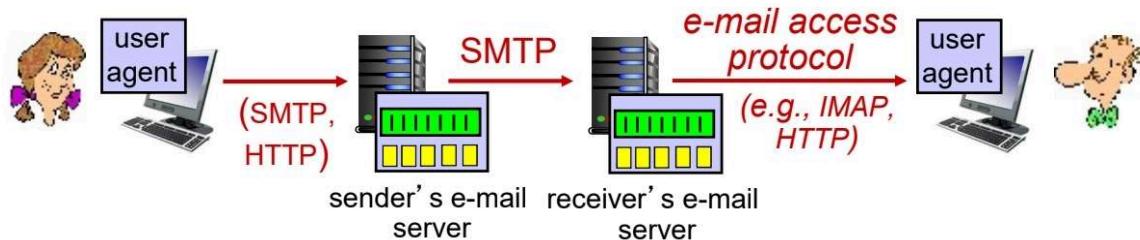
Link(連接層): (負責在相鄰的兩個節點，提供可靠的數據傳輸) data transfer between neighboring network elements

- Ethernet, 802.11 (WiFi), PPP

Physical(實體層): (數據傳輸的兩個設備之間提供物理連接) bits “on the wire”

3.

**Assume that Alice, with a Web-based e-mail account (such as Hotmail or gmail), sends a message to Bob, who accesses his mail from his mail server using POP3. Please list the series of application-layer protocols that are used to move the message from Alice's host computer to Bob's host computer.**



4. 主機 *A* 使用資料包交換策略向主機 *B* 發送  $L = 2000$  位的檔案。

*A* 和 *B* 之間有三個鏈路(和兩個分組交換機)，並且鏈路未擁塞(即沒有排隊延遲)。 $S_u$  表示每條鏈路上的傳播速度  $s$  為  $2 * 108 \text{ m/s}$ ，分組交換機處理延遲  $d$  在每台交換機上產生的  $d_{proc}$  為  $1 \text{ msec}$ 。鏈路傳輸速率(從第一個鏈路開始)分別為  $R_1 = 1 \text{ Mbps}$ 、 $R_2 = 2 \text{ Mbps}$  和  $R_3 = 500 \text{ Kbps}$ 。第一條鏈路的長度  $d_1$  為  $600$  公里，第二條鏈路  $d_2$  的長度為  $200$  公里，最後一條鏈路  $d_3$  的長度為  $400$  公里。發送此檔的總端到端延遲  $D$  總計是多少？(請先寫一個方程式，然後計算出答案  $d_i$ ,  $R_i$ , ( $i = 1, 2, 3$ ),  $d_{proc}$ ,  $s$  和  $L$ ，然後計算最終的答案。)

$$\begin{aligned}
 d_{total} &= d_{proc} + d_{queue} + d_{trans} + d_{prop} \\
 &= 1m * 2 + 0 + [(2000/500k) + (2000/2M) + (2000/1M)] + [(600k + 200k + 400k)/2*10^8] \\
 &= 15ms
 \end{aligned}$$

5. Assume you click a Web browser to obtain a Web page. However the Web caching proxy server (located inside your host) does not contain the IP address for the associated URL. So a DNS lookup is carried out to obtain the IP address. Suppose that 4 DNS servers are visited before your host receives the IP address from DNS; these successive visits incur an RTT of  $RTT_1, \dots, RTT_4$ . Let  $RTT_o$  denote the RTT between the local host and the server containing the Web page. Further suppose that the Web page consists of a very simple HTML text which references to another 5 very small objects. Assuming zero transmission time of the Web browser until you finally receive all objects with

- (a)  $2RTT_0 + 2RTT_0 + RTT_1 + RTT_2 + RTT_3 + RTT_4$

(b)  $2RTT0 + 5*2RTT0 + RTT1 + RTT2 + RTT3 + RTT4$

(c)  $2RTT0 + RTT0 + RTT1 + RTT2 + RTT3 + RTT4$

備註: