Computer Networking Notes Ch. 2

How to create a network app?

- 軟體主要是開發在終端上面，也就是 application layer 相互運作。
- 在開發網路軟體時，不需要考慮到網路設備。

Architecture

Client - Server architecture

- Server
  - 向 client 提供資料。
  - 永遠都在開機狀態。
  - 具有固定的 IP Address。
- Client
  - 向伺服器做聯繫並取得資料。
  - 不會與其他 client 端做聯繫。

P2P architecture

- 去中心化，client 也有可能是 server，也有可能兩者都是。
- 一個 peers 會像其他的 peers 請求資訊，因此會從其他的 peers 取得資料。
- peers 的 IP 是動態的。
- 最常使用的是 P2P File Sharing。

TCP / UDP

Application Requriement

- 應用程式需要的四大：
  - data intergrity
  - throughput
  - timing
  - security
- 依照需求，有些應用程式要求網速、時間、不能有任何 data loss...
- 主要分成兩種不同的 transport protocals services：TCP、UDP

TCP

- 主要有五大特點：
  - reliable transport：TCP 會去確認丟出去的封包是不是能夠正確的被 Receiver 所接收，故可以確保沒有 Data Loss，適合不允許有 Data Loss 的應用程式。
  - flow control
  - congestion control
  - connection-oriented
- 不提供這些服務：

UDP

- 與 TCP 相反：
  - unreliable transport：UDP 不會去確認丟出去的封包是不是能夠被 receiver 所接收，不管對方是死是活就是砸。
- 不提供這些服務：

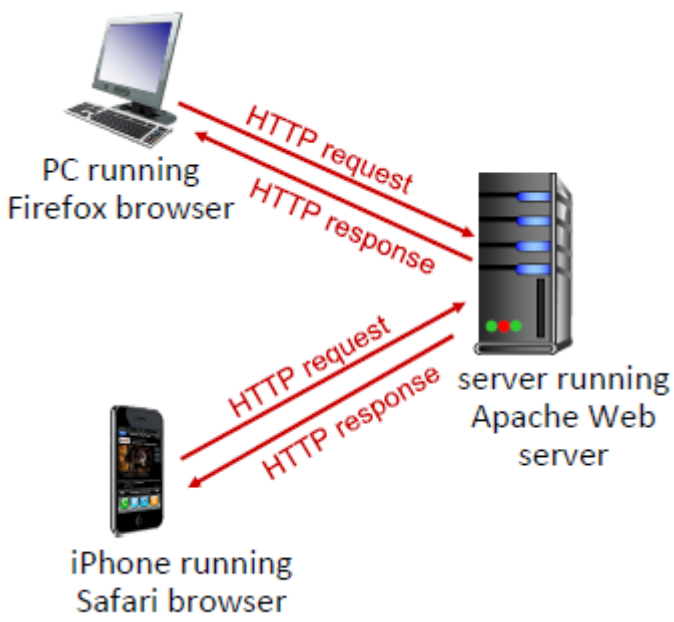
Web and HTTP

HTTP/1

- 全名為 hypertext transfer protocol，使用 Client-Server Model。

Client-Server Model

- 主要步驟：
  - client 會向 server 提出請求（Request），然後呈現 server 提供的網路物件。
  - server 會向發出請求的 client 端傳送物件（Response）。
- 過程中使用 TCP 的 Port 80。
- 是一個 stateless，不會記錄前一個使用者的請求。



Two type of connection

- Non-persistent HTTP (HTTP 1.0)
  - 流程：開始連接 → *下載一個物件* → 關閉連接。
  - 通常如果要下載多個物件，就需要有多個連接。
  - 這樣的設計最簡單，但沒有效率。
- Persistent HTTP (HTTP 1.1 - beyond)
  - 流程：開始連接 → *下載多個物件* → 關閉連接。

Non-presistent HTTP

The example of Non-persistent HTTP

如果有一個連結 http://uriah-website.net/home.index，含有文字與 10 個圖片：

- Client 初始化 TCP 連接，利用連結來連接 Server。
- Server 接受 Client 的連接，回傳 Accept。
- Client 利用 Socket 傳送請求訊息，請求 `home.index` 物件。
- Server 利用 Socket 回傳物件。
- Client 得到文字物件，Server 關閉連接。

用以上的方法來得到 10 個圖片，每次要求一個物件，故需要連接 10 次。

> Here should have a image to describe the description above.

The response time of Non-persistent HTTP

- RTT：小封包從 Client 往 Server 端的發送，並且回傳結果所需的時間。
- 使用 RTT 來估算 HTTP 的時間：
  - 對於 Non-persistent HTTP 來說，回傳 Accept 需要一個 RTT，回傳物件也需要一個 RTT，所以總共需要兩個 RTT。
  - 因此我們可以估算對於 Non-persistent HTTP 的反應時間為：$2\mathrm{RTT} + \text{file transmission time}$。
- 若我們考慮傳輸距離，則這個 RTT 可能會很大或很小，對於一些需要 timing 的應用程式來說，可能會太久，故效率不好。

> Here should have a image to describe the description above.

Presistent HTTP

> 待補

HTTP Message

訊息分成兩種：請求（request）與回應（response）

HTTP request Message

訊息使用 ASCII 編碼，主要分成三個部分：

- request line：具有請求方法、請求連結與協定。
  - 請求方法例如 `GET`、`POST`、`PUT` 等等。
- header line
- body：訊息本身的本體（可空可不空）。

HTTP response Message

訊息使用 ASCII 編碼，主要分成三個部分：

- request line：具有協定、請求狀態。
  - 請求狀態具有一串數字，與這串數字的定義，例如 `200 OK` 或者 `404 Not Found`。
- header line
- data：訊息本身的本體（可空可不空）。

Cookie

- Client - Server architecture 是無狀態（stateless）的。
- 商業的需求下，越來越希望在 HTTP 的架構下，具有狀態上的紀錄，因此衍伸了 cookie 的方法。
- 建立方法：
  - 在對網站進行請求時，建立並送予一個 ID。
  - 從後端資料庫找出這個 ID 的紀錄、或者記錄這個 ID 的行為。
- Cookie 的應用：
  - 認證（Authorization）
  - 購物車（Shopping Cart）
  - 建議（Recommendation）
  - 使用者對談狀態（User Session State）

Web Caches

- 為了讓 Client 的要求不一定要經過很遠的 Origin Server 來做存取，而是使用代理伺服器（Proxy Server）來存取。
- Proxy Server 會將 Origin Server 的資源存起來，讓 Client 來存取 Proxy Server 的資源。
- 若 Client 要求了 Proxy Server 沒有，但 Origin Server 有的資源，

  Proxy Server 會向 Origin Server 要求資源並存一份資源，再回傳給使用者同時做紀錄。

Caching example

Problem Description

- Caching 可以節省多少時間呢？
- 假設我們有：
  - 一個 $1.54\ \text{Mbps}$ 的 access link rate。
  - 當前的 RTT 為 $2\ \text{sec}$。
  - 當前的 Web object size 為 100k bits。
  - 平均從 browser 對 origin server 的 request rate 為 $15/\text{sec}$
- 我們可以計算出對於 browser 的平均 data rate 為 $1.50\ \text{Mbps}$
- 我們可以計算出 access link utilization 為 $1.50\ \text{Mbps}/1.54\ \text{Mbps} = 0.97$
- 端對端的延遲為 $\text{Internet delay} + \text{access link delay} + \text{LAN delay} = 2\ \text{sec} + \text{minutes} + \mu\text{secs}$
- 若 minutes 過慢則會導致 Queueing Delay（utilization ≈ 0.97，非常大）。

Solution

- 提升 access link rate：提升至 $154\ \text{Mbps}$，有效但成本很高。
- 安裝 Web Cache：
  - 若我們假設有 60% 的資料會從 origin server 要求，40% 的資料可以從 cache 要求。
  - 那麼瀏覽器的 data rate 可以被壓為 $0.6 \times 1.50\ \text{Mbps} = 0.9\ \text{Mbps}$

    - 可以算出 utilization 為 $0.9/1.54 \approx 0.58$
    - 平均端對端的時間為 $0.6 \times 2.01 + 0.4 \times (\approx \text{msecs}) \approx 1.2 \text{ secs}$
    - 這個方案比提升寬頻還要來得好，甚至更便宜。

Conditional GET

- 若網頁沒有特別更新，則 Server 將會回傳 `304 Modified` 來告知使用者的網頁是最新的，否則會回傳 `200 OK` 來更新網頁。
- 使用 Header 的 `If-modified-since: <date>` 來向伺服器確定是否進行變更。

HTTP/2

- HTTP/1.1 具有以下的缺點：
    - HTTP/1.1 使用先來先到（FCFS）原則，也就是伺服器使用 GET Methods 回應的物件是先來先處理。
    - 使用 FCFS，小物件可能會因為大物件導致 Head-of-line blocking。
        - 必須要等大物件下載完，小物件才能下載。
        - 開始存在輸出的競態關係。
- HTTP/2 改善了 HTTP/1.1 的缺點：
    - 將物件分割成小物件（frame），frame 的傳輸是交錯的，

      讓物件在傳輸時不需要純粹等待大物件的傳輸完成，而是做出類似同步傳輸的概念。

HTTP/2 to HTTP/3

- HTTP/2 存在單一 TCP 的問題，TCP 沒有辦法分辨出 Packet Loss 與 Congestion。
- 在 TCP 中存在壅塞控制的問題，在掉包的情況下會出現重複的確認，在檔案數量一多的情況下會導致更長時間的 Delay。
- 解決方法：每一個 Object 都是獨立的 UDP 傳輸，而不是單一的 TCP。

Email, SMTP and IMAP

Component

傳輸信件需要三個組件：

- SMTP：Simple Mail Transfer Protocol，傳送信件的協定。
- User Agent：信件閱讀器（例如：Outlook）。
- Email Server：信件伺服器，具有以下幾個子組件：
    - Mailbox：存放給使用者的信件
    - Message Queue：存放即將傳送的信件
    - SMTP Protocal 可以讓信件伺服器發送信件（Client）以及接收寄來的信件（Server）。

RFC 5321

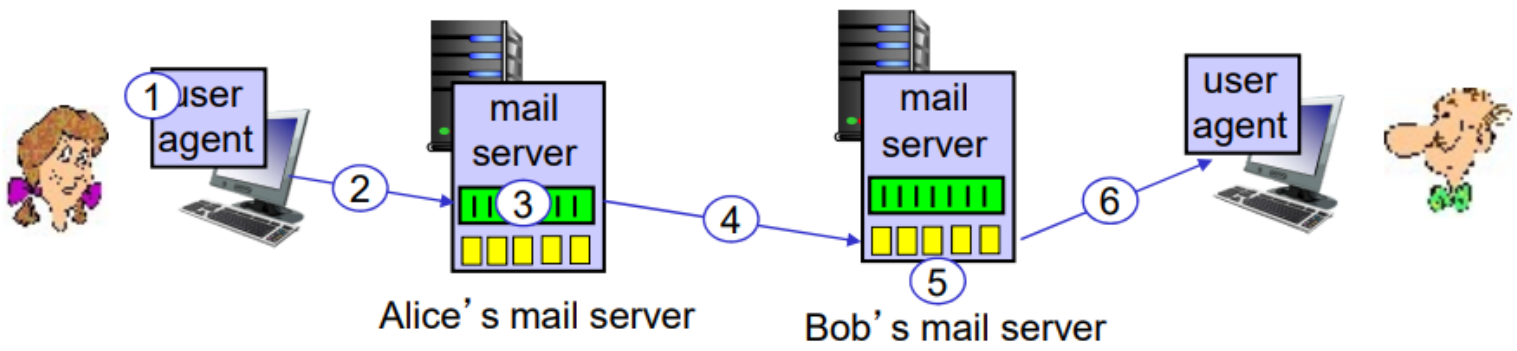
使用 RFC 5321：

- SMTP 的 Port 通常都是 25。
- SMTP 通常都是直接傳送，從 Client 傳送到 Server。
- 三階段傳送：Handshaking、傳送訊息、Closure。
- command/response 互動模式（如同 HTTP），包含以 ASCII 文字的 Command，與包含 Status Vode 以及 Phrase 的 Response

The process of mail transfer

以下演示了如何透過 STMP 協定進行寄信。

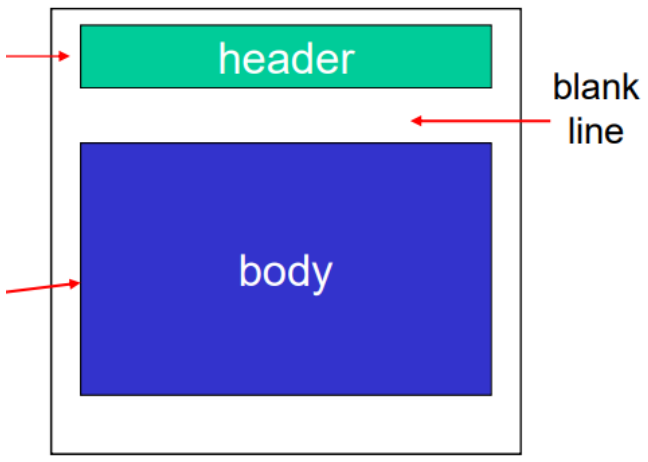
1. Alice 使用 User Agent 來寫一封信，寄給 Bob。
2. Alice 使用 mail server 把信寄給 Bob，信被放在 mail server 的 mail queue 內，等待被發送。
3. Alice's mail server 利用 client side 的方式，開啟 Bob's mail server 的 server 端 TCP 連接。
4. Alice's mail server 使用 STMP 協定，透過 TCP 來將信件送至 Bob's mail server 內。
5. Bob's mail server 的 Mailbox 放置了 Alice 的信。
6. Bob 使用 User Agent 來讀信。



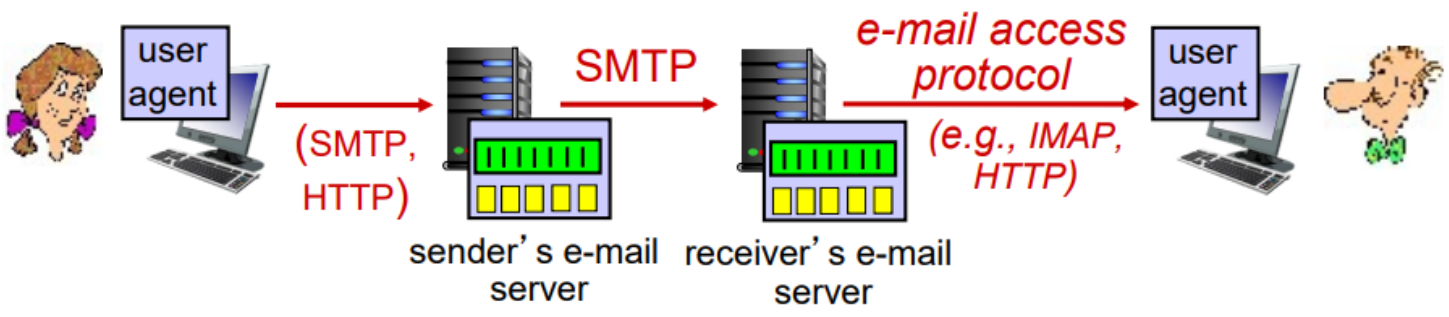
Mail message format

使用 RFC 8022 的定義方式：

- 信件分成 header 與 body，間隔一行來分隔。
- header 具有 To、From、Subject 的欄位，分別是收件人、寄件人與主旨。
- body 存放信件內容，必須是 ASCII 字元，



Mail acccess



在信件的傳輸中，過程可能提供多種不同的協定，這些協定大致的功能如下：

- SMTP：負責寄信／存信到收件人的伺服器。
- IMAP：Internet Mail Access Protocol，用來從本地端去存取遠端伺服器上的郵件的協定。
- HTTP：提供給一個基於網頁的 SMTP 與 IMAP 去存取信件內容。