NETWORK APPLICATION MODELS

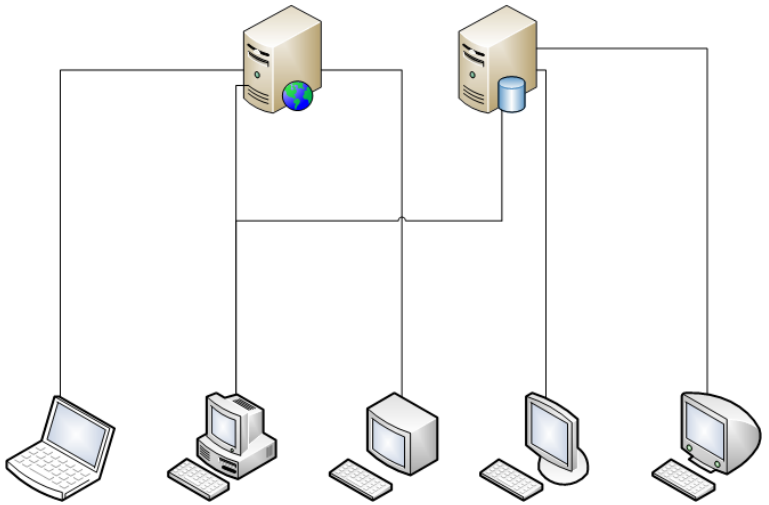
- CLIENT/SERVER MODEL
  - TELNET
  - SMTP
  - HTTP
- PEER-TO-PEER MODEL

JAVA SOCKET

การโปรแกรมระบบเครือข่าย

- ในการพัฒนาโปรแกรมระบบเครือข่ายสามารถแบ่งออกเป็น 2 สถาปัตยกรรมใหญ่ๆ คือ
  - Client/Server Architecture
  - Peer to Peer (P2P) Architecture
- ในการพัฒนาโปรแกรมแบบ Client/Server Architecture นั้นจะประกอบไปด้วย โปรแกรม 2 ประเภท
  - Client → ผู้ขอใช้บริการ และ
  - Server → ผู้ให้บริการ

Client/Server Architecture



- **หน้าที่การทำงานของโปรแกรม Client**
  - จะเป็นผู้เริ่มต้นติดต่อ Server เพื่อขอใช้บริการ
  - รอการตอบรับจาก Server เพื่อเริ่มใช้บริการ
- **หน้าที่การทำงานของโปรแกรม Server**
  - รอการขอใช้บริการจาก Client
  - เมื่อ Client ขอใช้บริการ และ Server พร้อมที่จะให้บริการจะตอบกลับหา Client เพื่อเริ่มให้บริการ
- ในสถาปัตยกรรมแบบ Client/Server โปรแกรม Client จะเชื่อมต่อและแลกเปลี่ยนข้อมูลกับ Server เท่านั้น โปรแกรม Client จะไม่สามารถเชื่อมต่อแลกเปลี่ยนข้อมูลกับ Client อื่นๆ ได้

ข้อดีและข้อเสียของ Client/Server Architecture

- **ข้อดี**
  - เมื่อบริการต่างๆที่ Client จะขอใช้งานอยู่ที่ Server ทำให้การบำรุงรักษา เช่น upgrade โปรแกรม, เปลี่ยนอุปกรณ์ สามารถทำที่ Server อย่างเดียวไม่จำเป็นต้องไปยุ่งกับเครื่อง Client
  - ในการเก็บข้อมูลถ้าข้อมูลถูกเก็บใน Server ทั้งหมดแล้ว Server สามารถตั้งความปลอดภัยในการเข้าถึงข้อมูลนั้นๆ ได้
  - ง่ายต่อผู้ดูแลระบบในการตรวจสอบการให้บริการของโปรแกรมระบบเครือข่าย
  - Server สามารถตั้งความปลอดภัยในการให้บริการต่างๆได้ง่าย
- **ข้อเสีย**
  - ภาระการทำงานตกอยู่กับเครื่อง Server เป็นส่วนใหญ่
  - เมือ Server มีปัญหาอาจทำให้ผู้ใช้งานทั้งระบบเครือข่ายไม่สามารถทำงานได้

Peer-to-Peer Architecture

- ส่วนใหญ่จะเขียนย่อว่า P2P
- จะเรียกแทนระบบโปรแกรมของเครือข่าย ที่เครื่องทุกเครื่องมีหน้าที่ในการทำงานเท่าเทียมกัน คือ เป็นได้ทั้ง เซิฟเวอร์ และ ไคลเอ็น ในแต่ละช่วงเวลา
- บางครั้ง P2P จะหมายถึงระบบที่มีการเปลี่ยนแปลงตลอดเวลาโดยไม่มีผลกระทบต่อระบบรวม
- ถูกประยุกต์ใช้ในช่วงเวลาที่ผ่านมาไม่นานหลังจาก **internet** ได้รับความนิยม เนื่องจากในการทำงานในระบบ **internet** บางอย่างไม่สามารถที่จะหาผู้รับผิดชอบ **server** ได้

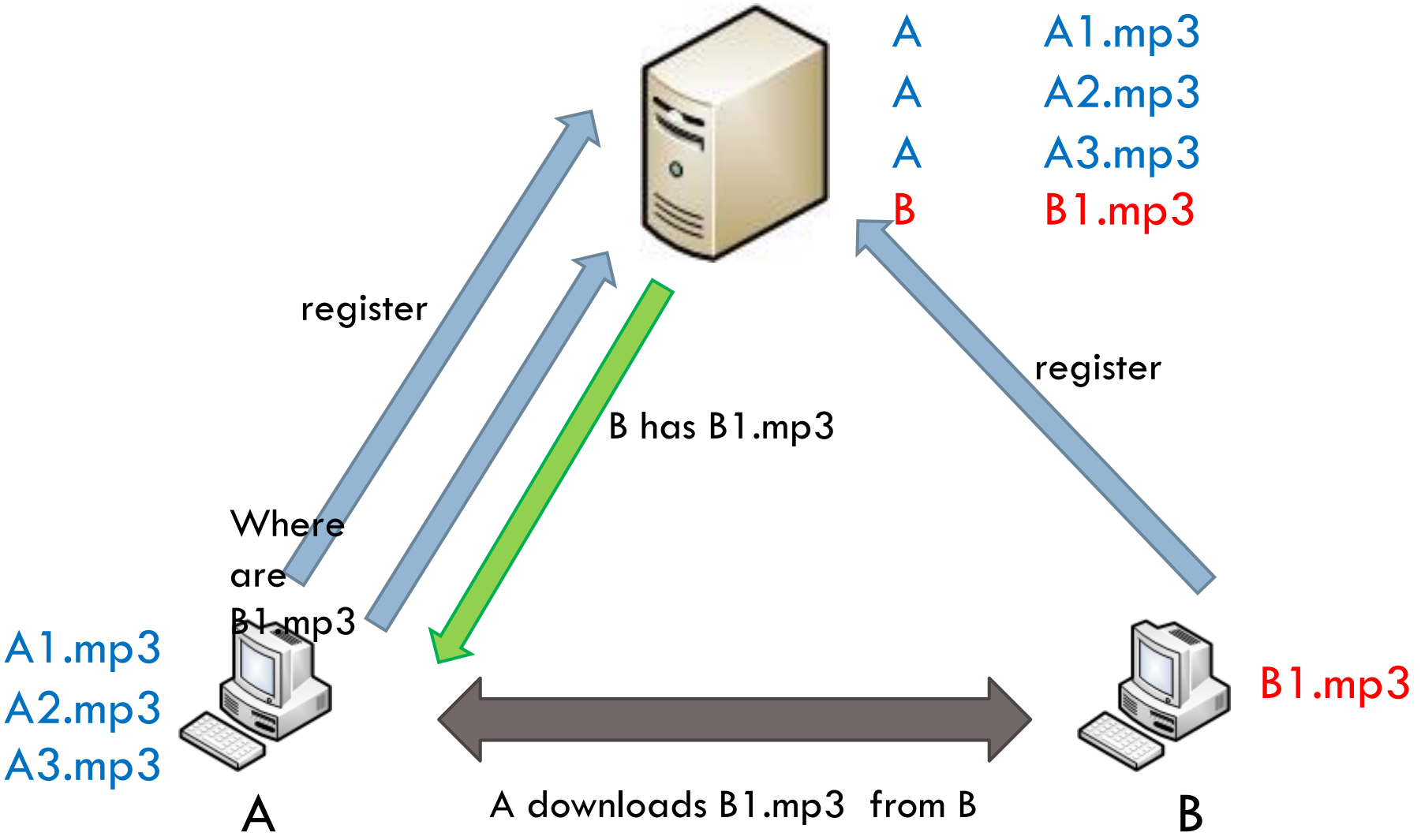
ประเภทของ Peer-to-Peer

- P2P แบ่งออกเป็น 2 พวกใหญ่ๆคือ
  - **Centralized** แบบรวมศูนย์กลาง
  - **Decentralized** แบบกระจาย ซึ่งแบ่งออกอีก 3 ประเภทคือ
    - Unstructured แบบไร้โครงสร้าง
    - Structured แบบมีโครงสร้าง
    - Hybrid แบบผสม

Peer-to-Peer Architecture

- Application แรกๆที่ทำให้สถาปัตยกรรมแบบ P2P โด่งดังก็คือ การแชร์ไฟล์ (file sharing)
- ซึ่งเริ่มต้นที่ประมาณปี 1999 ได้มีการพัฒนาโปรแกรมที่ทำการแชร์ไฟล์ mp3 โดยเป็นแบบ centrailized ชื่อว่า napster
- Napster
  - เป็นโปรแกรมที่พัฒนาเพื่อแลกเปลี่ยนไฟล์ mp3 ของผู้ใช้ตามบ้านทั่วไป
  - Napster ถือว่าเป็น P2P แบบ centralized เพราะจำเป็นต้องมี server ที่ทำหน้าที่ เหมือนสมุดหน้าเหลือง ที่เก็บตำแหน่งที่อยู่ของ ไฟล์ mp3

การทำงานของ Napster



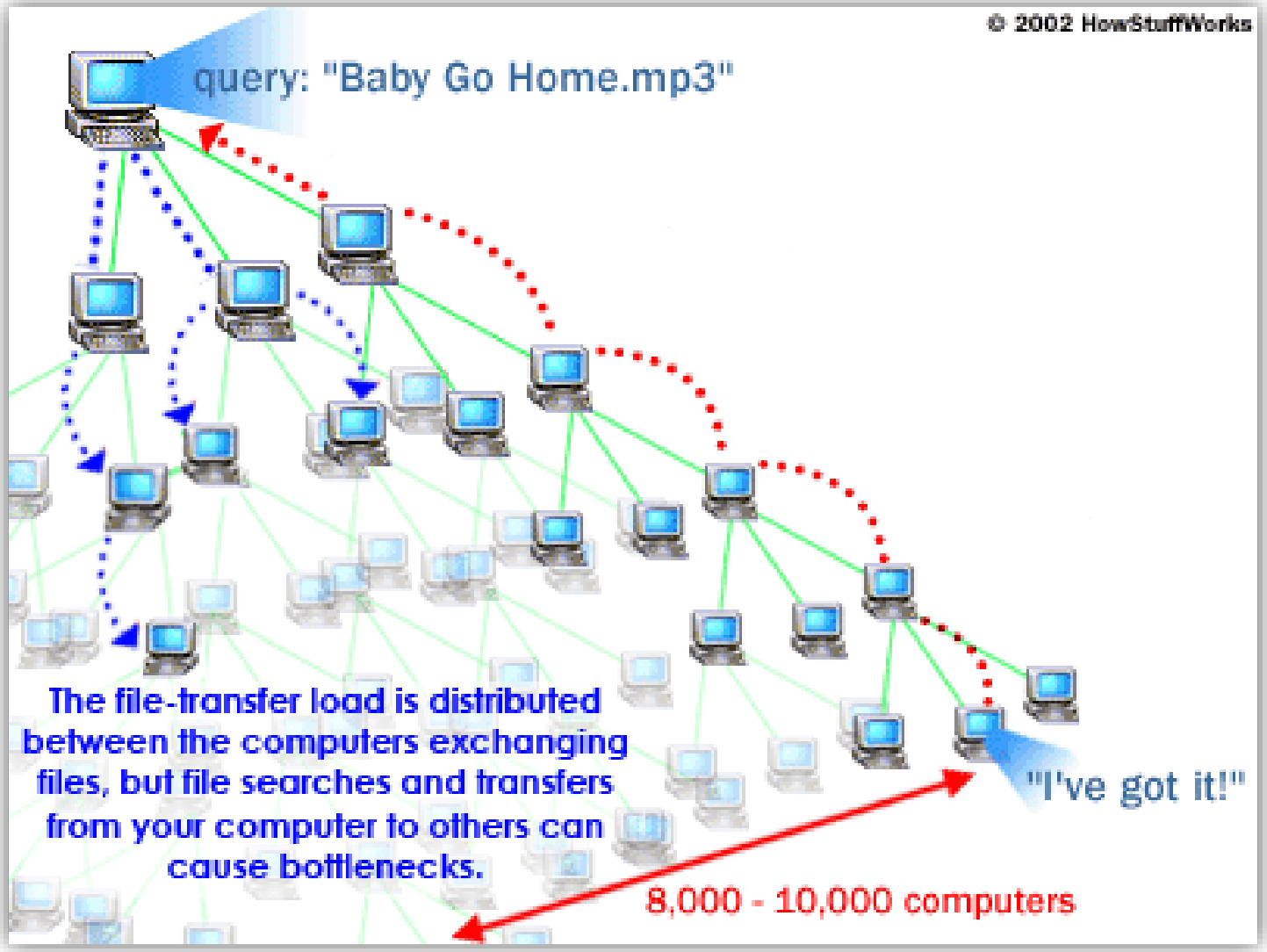
ข้อดีและข้อเสียของระบบ Napster

- **ข้อดี**
  - กระจายเนื้อที่ในการเก็บ mp3 และ bandwidth เพราะใครจะออกเงินซื้อ server ที่มี harddisk ขนาดใหญ่เพื่อจุ mp3 และเช่าสายสัญญาณ internet เพื่อรองรับคนทั้งโลก
  - ระบบทำงานเร็วเพราะ server แค่ค้นหา IP address หรือชื่อโฮทส์ที่มี mp3 ที่ต้องการ
- **ข้อเสีย**
  - ผู้ใช้จำเป็นต้องติดต่อกับ server เพื่อจะทราบว่าจะ download ที่ไหน ถ้า server มีปัญหา ก็จะทำให้ระบบล่มทั้งระบบ
  - จากข้อเสียข้อนี้ทำให้ napster หยุดการให้บริการเพราะศาลสั่งให้ปิดเซิฟเวอร์ เนื่องจาก กฎหมาย copyright ของ mp3

Gnutella

- หลังจาก Napster ถูกปิด ทางออกใหม่สำหรับการแชร์ไฟล์คือ การพัฒนาระบบ P2P แบบ decentrailized ซึ่งไม่จำเป็นต้องมี server
- Gnutella เป็น 1 ใน application ต้นๆ ที่ได้ใช้ระบบ decentralized P2P ถูกพัฒนาขึ้นประมาณปี 2001
- Gnutella เป็น decentralized P2P system แบบ unstructured
- การค้นหาข้อมูลใช้วิธีการที่เรียกว่า Flooding

Gnutella


© 2002 HowStuffWorks
query: "Baby Go Home.mp3"
The file-transfer load is distributed between the computers exchanging files, but file searches and transfers from your computer to others can cause bottlenecks.
"I've got it!"
8,000 - 10,000 computers

Gnutella

- **ข้อดี**
  - กระจายเนื้อที่ในการเก็บ mp3 และ bandwidth เพราะใครจะออกเงินซื้อ server ที่มี harddisk ขนาดใหญ่เพื่อจุ mp3 และเช่าสายสัญญาณ internet เพื่อรองรับคนทั้งโลก
  - ไม่มี server ที่เก็บข้อมูลกลางทำให้ไม่มีใครสามารถปิดระบบได้
- **ข้อเสีย**
  - วิธีการ flood อาจทำให้เครื่องผู้ใช้ทำงานมากกว่าปกติ
  - ไม่มีการรับประกันว่าข้อมูลที่ถูกหาจะสามารถหาเจอ แม้ว่าข้อมูลนั้นจะมีอยู่จริงในระบบ
  - การค้นหาข้อมูลจะช้ากว่าแบบ centralized ภายหลังจึงมีการเพิ่มส่วนที่เรียกว่า supernode เพื่อเก็บ cache

CHORD, PASTRY

- ในด้านการวิจัยได้พัฒนาระบบ P2P แบบ decentralized ที่มีโครงสร้างขึ้น (structured)
- โครงสร้างพื้นฐานจะเป็นรูปวงแหวน (ring) โดยแต่ละ node จะมี nodeID
- วัตถุที่ต้องการจะเก็บเข้าในระบบ จะมี objectID
- ระบบจะพยายามวาง objectID ลงใน nodeID ที่มีค่าใกล้เคียงกันที่สุด
- ทำให้การค้นหาข้อมูลสามารถหาได้ง่ายและรวดเร็ว

Bittorrent

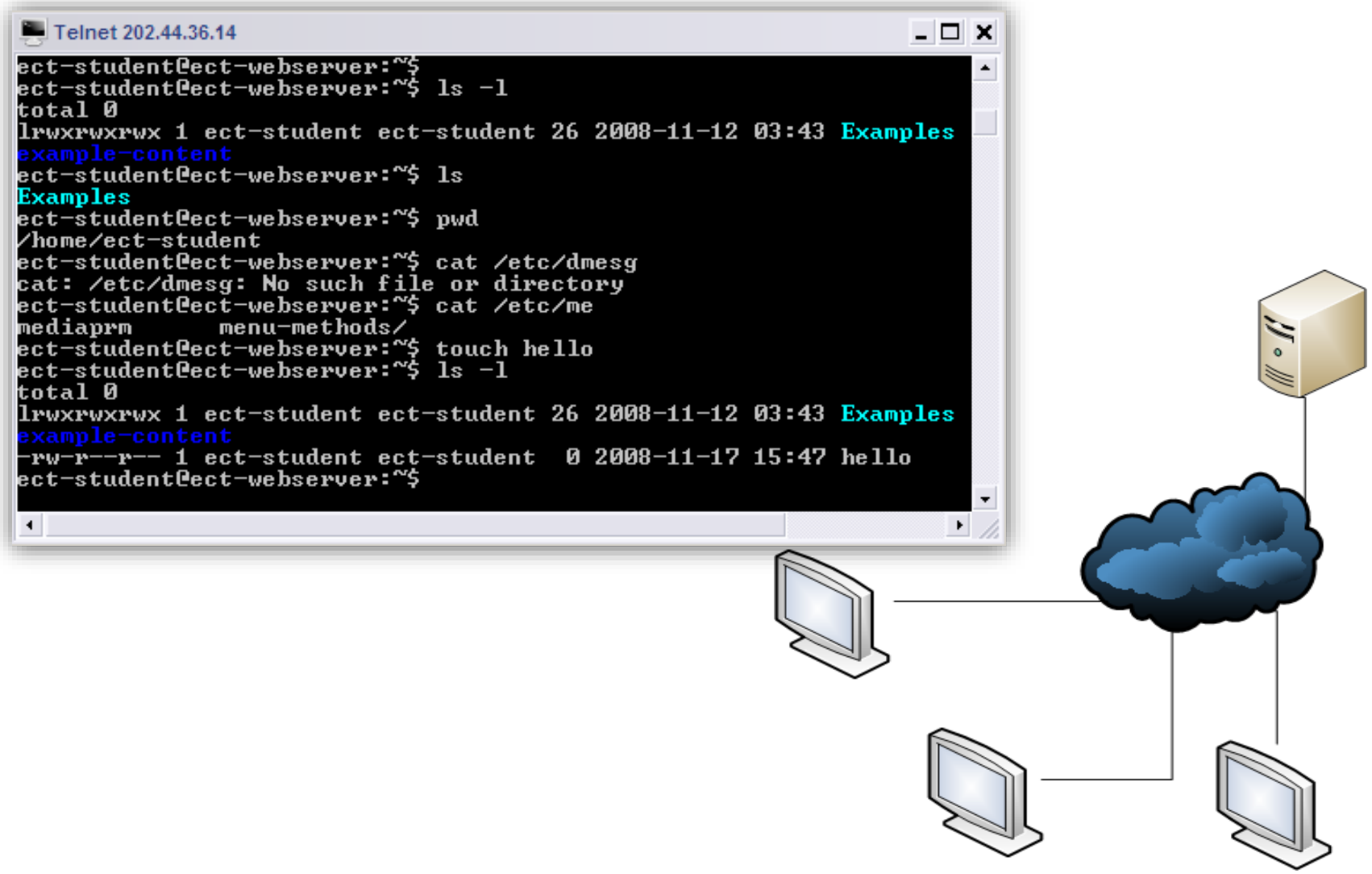
- เป็นการผสมผสานระบบแบบ **centralized** และ **decentralized** เรียกว่า **hybrid**
- ใน **bittorrent** จะมี **tracker** ที่ทำหน้าที่เหมือนเป็น **server** ที่เก็บ **IP address** ของเครื่องที่กำลังแชร์ไฟล์นั้นอยู่
- **Client** จะติดต่อกับ **tracker** เพื่อขอ **IP address** ของเครื่องที่แชร์ไฟล์นั้นๆ แล้วติดต่อกับ **client** อื่นเพื่อ **download** ไฟล์ข้อมูลนั้น
- ได้แทรกระบบ **tit-and-tat** ยิ่งแชร์มากยิ่ง **download** เร็วเพื่อป้องกันปัญหาคนที่เอาแต่ **download** แต่ไม่ยอมแชร์คืนให้ระบบ

Telnet (**Tel**ecommunication **net**work)

ประวัติ

- ก่อนที่ PC จะแพร่หลาย ผู้ใช้คอมพิวเตอร์จะใช้บริการผ่านทางเทอร์มินัลที เรียกว่า **Dumb terminal**
- ที่ต่อเชื่อมตรงอยู่กับ Server โดยคำสั่งต่างๆ เมื่อพิมพ์จะถูกส่งไปหา Server โดยตรง
- Server จะทำหน้าที่ทั้งเก็บข้อมูล และประมวลผลคำสั่ง และส่งผลลัพธ์กลับมาหา **terminal** โดยจะไม่มีการประมวลผลคำสั่งและเก็บข้อมูลใน **terminal**
- เมื่อระบบเครือข่ายแพร่หลาย สถานีส่วนบุคคลมีความสามารถในการประมวลผลมากขึ้น แต่การขอเข้าไปทำงานใน Server ก็ยังมีการใช้งานอยู่จึงเป็นที่มาของ **telnet** ซึ่งจะหน้าที่เสมือน PC ต่อตรงอยู่กับ Server

รูปแบบการทำงานของ Telnet



การใช้งาน Telnet

- การเชื่อมโยงด้วย **Telnet** จะทำโดย
  - เครื่อง **Client** ขอสถาปนาการเชื่อมต่อด้วย **TCP** กับเครื่อง **Server** ที่ **port** หมายเลข 23
  - การส่งข้อมูลจะอยู่ในรูป **ASCII code**

- ใน **Windows** และ **Linux** มีคำสั่ง **telnet** มาให้อยู่แล้ว และสามารถเรียกใช้งานได้เลย ด้วยคำสั่ง

telnet <ชื่อ telnet server> หรือ

telnet เพื่อเข้าสู่ telnet prompt

การใช้งาน Telnet (2)

- ในปัจจุบัน บริการ **Telnet** แทบจะไม่เปิดให้บริการแล้ว เนื่องจาก ข้อมูลที่ส่งผ่านเครือข่ายไม่มีการเข้ารหัส ทำให้ผู้บุกรุกระบบเครือข่ายสามารถดักข้อมูลที่ส่งผ่านกันในเครือข่าย และ ขโมยรหัสผ่านได้ เปลี่ยนมาใช้ **SSH** แทน

- **Telnet** สามารถจำลองการทำงานของ **TCP** ได้โดยการบังคับหมายเลข **port** เหมาะสำหรับ **debug application** อื่นๆ ที่แลกเปลี่ยนข้อมูลเป็น **text** และใช้ **TCP** เช่น
  - **telnet mail.kmutnb.ac.th 25** คือการต่อเข้า **server : mail.kmutnb.ac.th** ที่ **port** หมายเลข 25 ด้วย **TCP**

SMTP (Simple Maill Transfer Protocol)

- เป็นบริการส่ง **mail** แบบง่ายๆ
- ให้บริการแบบ **TCP** ที่ **port** หมายเลข 25
- มีคำสั่งอยู่ในรูป **ASCII**
- เราสามารถใช้ **telnet** เชื่อมต่อเข้าที่ **port** หมายเลข 25 เพื่อส่ง **mail** ได้
  - **telnet** <ชื่อ **mail server**> **25**

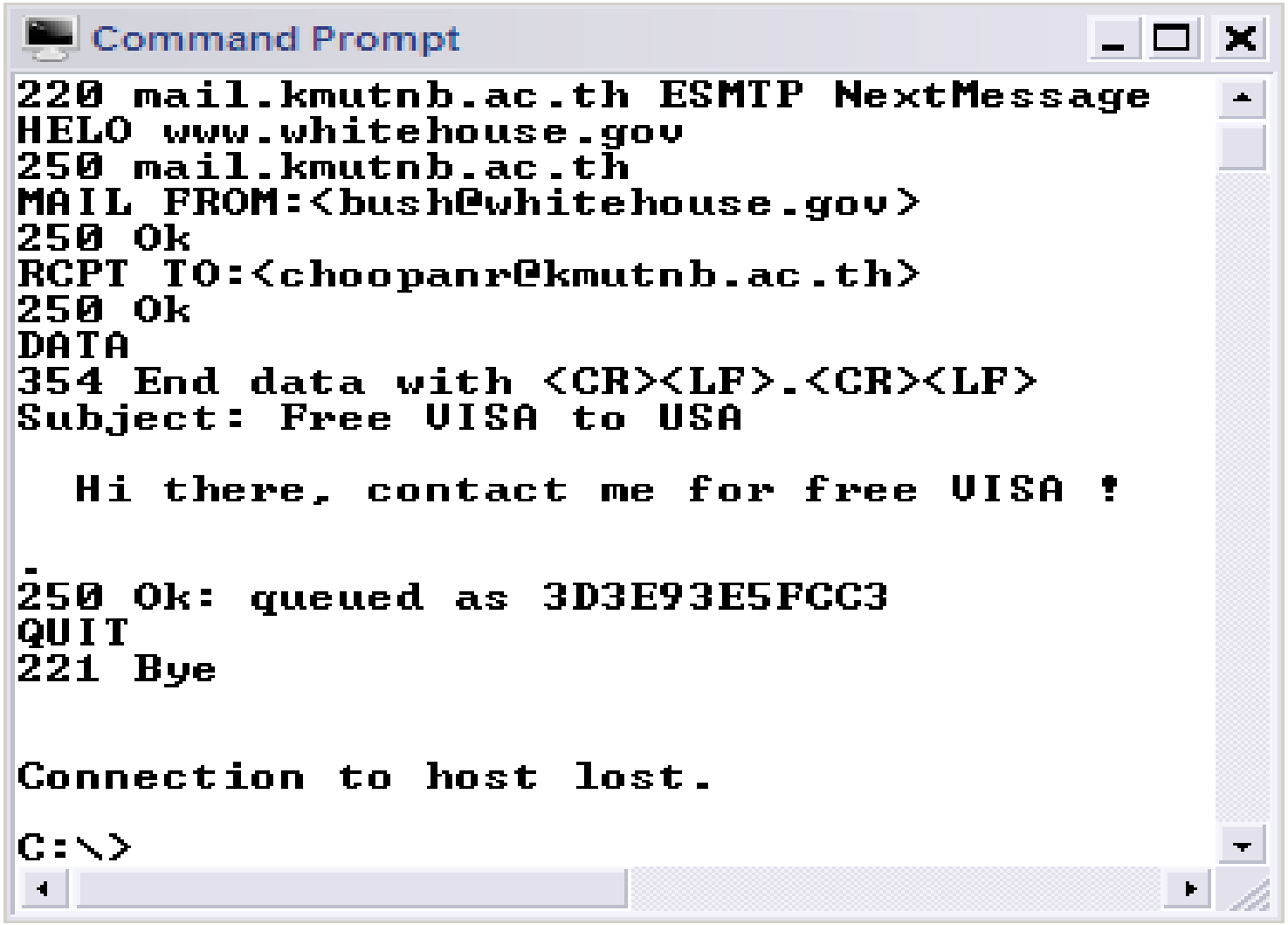
Protocol การส่ง mail ของ SMTP

- เริ่มต้นด้วยการใช้คำสั่ง HELO หรือ EHLO แล้วแต่ version ของ SMTP server ตามด้วยชื่อ host ที่จะรับ mail ตอบกลับ
  - HELO www.microsoft.com
- ใส่ชื่อผู้ส่งด้วยคำสั่ง MAIL FROM:<***email ผู้ส่ง***>
  - MAIL FROM:<billgates@microsoft.com>
- ใส่ชื่อผู้รับด้วยคำสั่ง RCPT TO:<***email ผู้รับ***>
  - RCPT TO:<stevejobs@apple.com>

การส่ง mail ทาง SMTP

- พิมพ์ DATA แล้ว **enter** เพื่อบอกว่าต่อไปจะเป็นเนื้อหาของ **email**
- เมื่อเขียน **email** เรียบร้อยแล้วให้ **enter** 1 ทีเพื่อให้บรรทัดว่างแล้วพิมพ์จุด (.) ที่คอลัมแรกของแถวใหม่แล้วกด **enter**
- จากนั้นพิมพ์ QUIT เพื่อปิดการเชื่อมต่อกับ **mail server**

ตัวอย่างการใช้ SMTP



Fakemail, mailbomb

- สมัยก่อน การติดต่อเข้าใช้บริการ SMTP สามารถทำ **mail** ปลอม หรือที่เรียกว่า **fakemail** โดยการปลอมชื่อ **email** ผู้ส่งแปลกๆ ซึ่งเปลี่ยนตรงส่วน MAIL FROM:<***ชื่อ email แปลก ๆ***>
- **Mailbomb** คือการเขียนโปรแกรมวนลูปให้เชื่อมต่อใช้บริการSMTP เพื่อส่ง **fakemail** เป็นจำนวนมาก
- แต่ในสมัยนี้ SMTP server จะมีการป้องกันความปลอดภัยเอาไว้คือจะกรองให้ผู้ส่งต้องทำการ **authentication** ก่อนจะส่ง **email** ข้ามออกจาก **server**
- และในปัจจุบันสามารถตรวจสอบและค้นหาผู้ที่ส่ง **fakemail** ได้ดังนั้น **โปรดระวังในการใช้งาน**

HyperText Transfer Protocol (HTTP)

- เป็นบริการเกี่ยวกับ web page โดยเป็น protocol ที่ใช้ในการติดต่อระหว่าง HTTP client (web browser IE, firefox) และ HTTP server หรือที่เรียกกันว่า web server
- ตามมาตรฐานแล้ว Web server ให้บริการบน TCP ที่ port หมายเลข 80
- ตอนนี้มีใช้กัน 2 version คือ HTTP 1.0 และ HTTP 1.1
- สามารถใช้ telnet ติดต่อไปยัง port หมายเลข 80 ของ web server เพื่อ debug ก็ทำงานของ HTTP ได้
  - ในกรณีที่ใช้ Windows ผู้ใช้จะ**ไม่สามารถ**เห็นข้อความที่พิมพ์
  - ในกรณีที่ใช้ Linux ผู้ใช้สามารถเห็นข้อความที่พิมพ์

HTTP version 1.0

- คำสั่งที่ใช้ในการ **download** ทรัพยากร **(resources)** จาก **web server** คือ

GET <ชื่อทรัพยากร> HTTP/1.0

  - **GET** เป็นคำสั่งที่ใช้สำหรับของโหลดทรัพยากรจาก **web server**
  - **ชื่อทรัพยากร** จะรวมถึง **path** ที่เข้าถึงทรัพยากร
  - **HTTP/1.0** เป็นการระบุบอก **web server** จะติดต่อแบบ HTTP version 1.0

Example : GET

- **ตัวอย่าง**
  - ถ้าต้องการโหลดหน้าเวป

    http://www.somehost.com/path/file.html

1. ใช้คำสั่ง telnet www.somehost.com 80
2. พิมพ์

   GET /path/file.html HTTP/1.0
3. กด Enter 2 ที

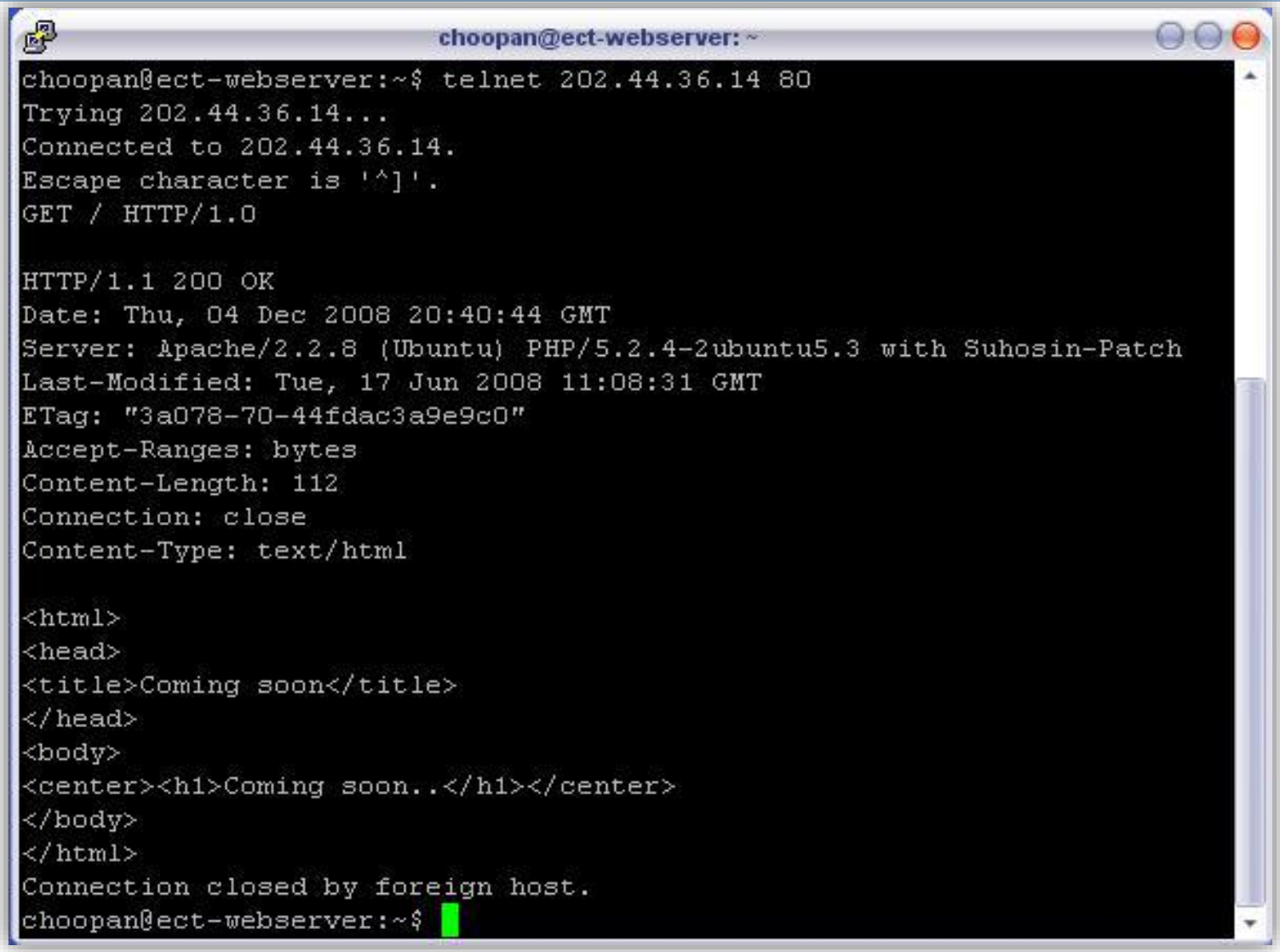
HTTP version 1.0

- ส่วนขยายของการ **request** มีคำสั่งอื่นๆ เช่น
  - **From :** บ่งบอก **email** ของผู้ติดต่อกับ **web server** ใช้เฉพาะในกรณีพิเศษ
  - **User-Agent :** ใช้ในการบ่งบอกว่า **web browser** คืออะไร
- รหัสที่ตอบสนองจาก **web server** หลังจากที่ **client** ติดต่อขอ **download** ทรัพยากร
  - 2XX สำเร็จ
  - 3XX **redirect** ไปที่หน้าอื่น
  - 4XX มีปัญหาที่ตัวของ **client**
  - 5XX มีปัญหาที่ตัวของ **server**

HTTP version 1.0

- นอกเหนือจากรหัสในการตอบกลับแล้ว **web server** ยังมีข้อความบางอย่างกลับมาด้วยเพื่อเป็นข้อมูล เช่น
  - **Server :** บ่งบอกว่า **web server** ใช้โปรแกรมชื่อว่าอะไร
  - **Last-Modified :** บอกว่าทรัพยากรที่ถูก **request** นี้แก้ไขครั้งล่าสุดเมื่อใด
  - **Content-type :** บอกว่าทรัพยากรที่ถูก **request** นี้เป็นไฟล์ประเภทไหน (MIME-type) เช่น **text/html, image/gif**
  - **Content-length :** บอกว่าทรัพยากรที่ถูก **request** มีขนาดกี่ไบต์

Example : HTTP/1.0



HTTP version 1.0

- นอกเหนือจาก GET แล้วใน HTTP version 1.0 ยังมีคำสั่งอื่นๆอีก เช่น
  - **HEAD** ใช้สำหรับดึงหัวข้อมูลอย่างเดียว โดยไม่ download ทรัพยากรที่ request มาจริงๆ
  - **POST** ใช้สำหรับส่งค่าให้กับทรัพยากรนั้น เช่น ส่งค่าให้กับ CGI เช่น PHP
- หลังจาก HTTP/1.0 ได้รับความนิยมจึงมีการพัฒนา HTTP version 1.1 ขึ้นมาเพื่อรองรับการทำงานที่มากขึ้น และให้การตอบสนองรวดเร็วขึ้น

HTTP version 1.1

- ใน HTTP/1.1 มีการรองรับ **multi-homed** หมายถึง การที่ **web server** เครื่องเดียวสามารถให้บริการหลายเวปไซต์ เช่น
  - ให้บริการ **www.host1.com** และ **www.host2.com** ในเครื่องเดียวคือมี IP เดียว

- ดังนั้นใน HTTP/1.1 จึงบังคับให้ใส่ชื่อ **host** ทุกครั้งหลังจากใช้คำสั่ง GET ด้วยการใช้คำสั่ง **Host :**

Example : HTTP/1.1

- **ตัวอย่าง**
  - ถ้าต้องการโหลดหน้าเวป

    http://www.somehost.com/path/file.html

1. ใช้คำสั่ง telnet www.somehost.com 80
2. พิมพ์

   GET /path/file.html HTTP/1.1

   Host : www.somehost.com

   User-Agent: choopan
3. กด Enter 2 ที

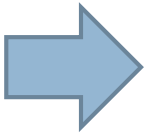
HTTP version 1.1

- ใน HTTP version 1.0 การเชื่อมต่อจะปิดลงทุกครั้ง หลังจากการ transfer ทรัพยากร เสร็จสิ้นลง
- การขอเปิดการเชื่อมต่อ และ ปิดการเชื่อมต่อทุกครั้งเพื่อ download ทรัพยากรเพียง 1 อย่าง จะเพิ่มโหลดให้กับ web server อย่างมาก และทำให้การตอบสนองช้า
- HTTP version 1.1 จึงได้พัฒนาคือ การเชื่อมต่อจะไม่ปิดตัวลงหลังจาก download ทรัพยากรเสร็จสิ้น เพื่อความเร็วในการตอบสนอง และ ลดภาระโหลดให้กับ web server

HTTP version 1.1

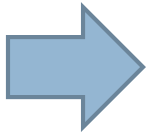
- อย่างไรก็ตาม **HTTP version 1.1** ก็ยังให้อำนาจกับ **client** ในการเลือกรูปแบบการเชื่อมต่อ ว่าจะให้ปิดการเชื่อมต่อ หลังจาก **download** ทรัพยากรหรือไม่
- **ตัวอย่าง :** จะโหลด **http://www.somehost.com**

Default : ไม่ปิดการเชื่อมต่อหลังจากโหลดหน้าเวปเสร็จ



```
GET      /      HTTP/1.1
Host :   www.somehost.com
```

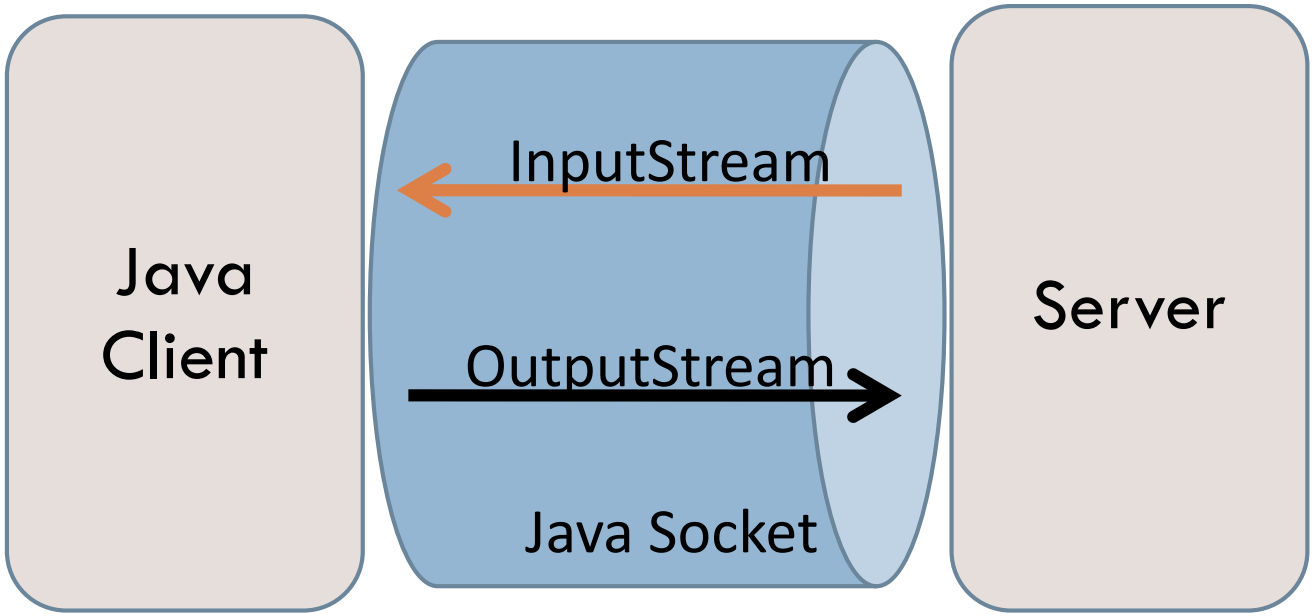
ปิดการเชื่อมต่อหลังจากโหลดหน้าเวปเสร็จ



```
GET      /      HTTP/1.1
Host :   www.somehost.com
Connection : close
```

Java Socket

- ทบทวน
  - การสร้าง socket ใน java คือการใช้ Class Socket
  - Socket s = new Socket(“ชื่อโฮสท์”, หมายเลขพอร์ต);



การอ่านค่าและส่งค่าผ่าน socket

- จาก Object ของ Socket ที่เราได้สามารถเรียกใช้ method ดังนี้
  - การอ่านค่าจาก Socket จะทำผ่าน InputStream

    public InputStream **getInputStream()** *throws IOException*

  - การส่งค่าลงไปใน Socket จะทำผ่าน OutputStream

    public OutputStream **getOutputStream()** *throws IOException*

  - เมื่อสิ้นสุดการทำงานทุกครั้ง จะต้องเรียก method close(); เพื่อปิดการใช้งานของ InputStream และ OutputStream

ตัวอย่างการเอา InputStream และ OutputStream ออกจาก Socket

```
1  import java.io.*;
2  import java.net.*;
3
4  public class TestOne {
5      public static void main(String[] args)  {
6          try {
7              Socket s = new Socket("127.0.0.1",  80);
8              InputStream in   = s.getInputStream();
9              OutputStream out = s.getOutputStream();
10             in.close();
11             out.close();
12             s.close();
13         } catch (Exception e) {
14             e.printStackTrace();
15         }
16     }
17 }
```

ประยุกต์ใช้ BufferedReader และ PrintWriter

```
1  import java.io.*;
2  import java.net.*;
3
4  public class TestTwo {
5      public static void main(String[] args)  {
6          try {
7              Socket s = new Socket("127.0.0.1",  80);
8              InputStream in   = s.getInputStream();
9              OutputStream out = s.getOutputStream();
10
11
12             BufferedReader sin = new BufferedReader(
13                                          new InputStreamReader(in));
14
15             PrintWriter sout = new PrintWriter(out);
16
17             sin.close();
18             sout.close();
19             in.close();
20             out.close();
21             s.close();
22         } catch (Exception e) {
23             e.printStackTrace();
24         }
25     }
26 }
```