Introduction to assignment 2 and socket programming

TDTS04: Computer networks and distributed systems

January 2025

LINKÖPING UNIVERSITY

Lecture outline

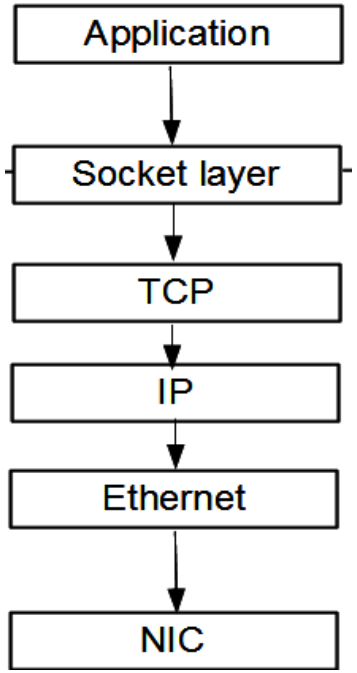
- General lab assignment information
- Assignment 2
- HTTP & Proxy
- Socket programming
- Questions

General lab assignment information

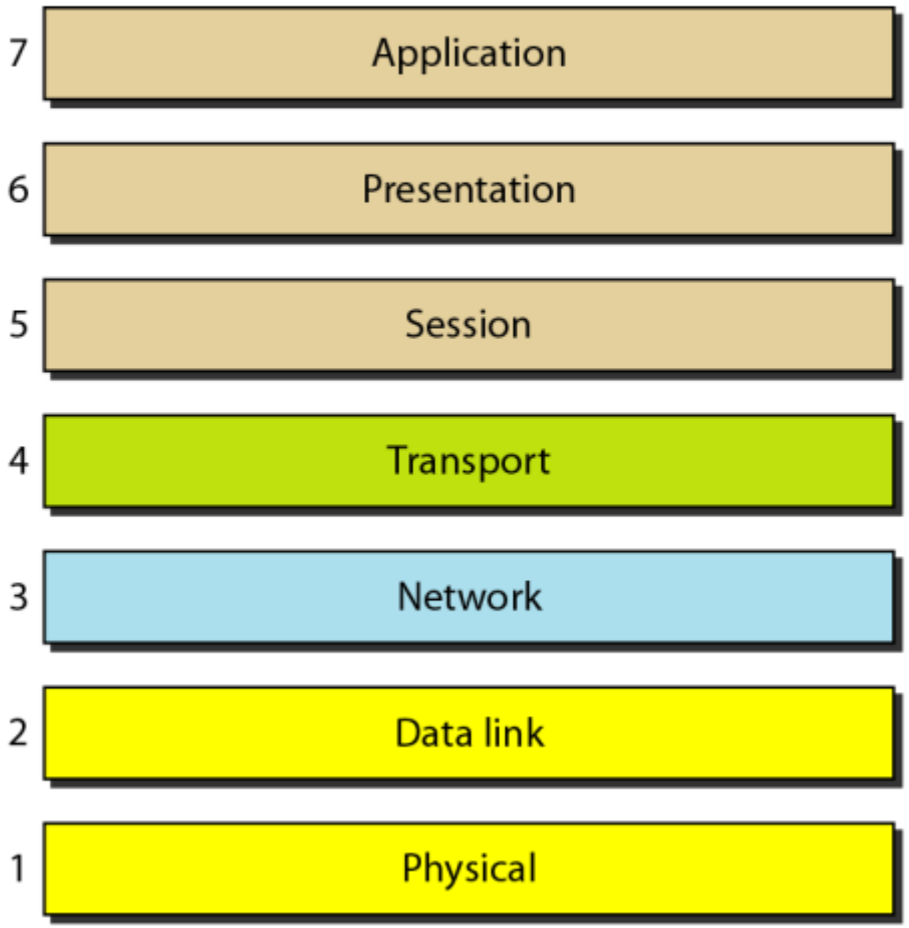
- Assignment 1 should be finished by 31st Jan. 2025
- Assignment 2 takes time, and have a soft deadline due: 20th of Feb. 2025
- Third assignment is more in the style of the first and shouldn't take too much time
- The last assignment (4) needs a little more time so don't put it off
- deadlines and last time to demonstrate are stated in course webpage
- Check with the TA if you plan to use languages other than those prescribed

Assignment 2 – what will we do?

- Learn about WWW and HTTP
- Learn TCP/IP socket programming to understand HTTP and WWW better
- Build a simple proxy



Seven layers of the OSI model



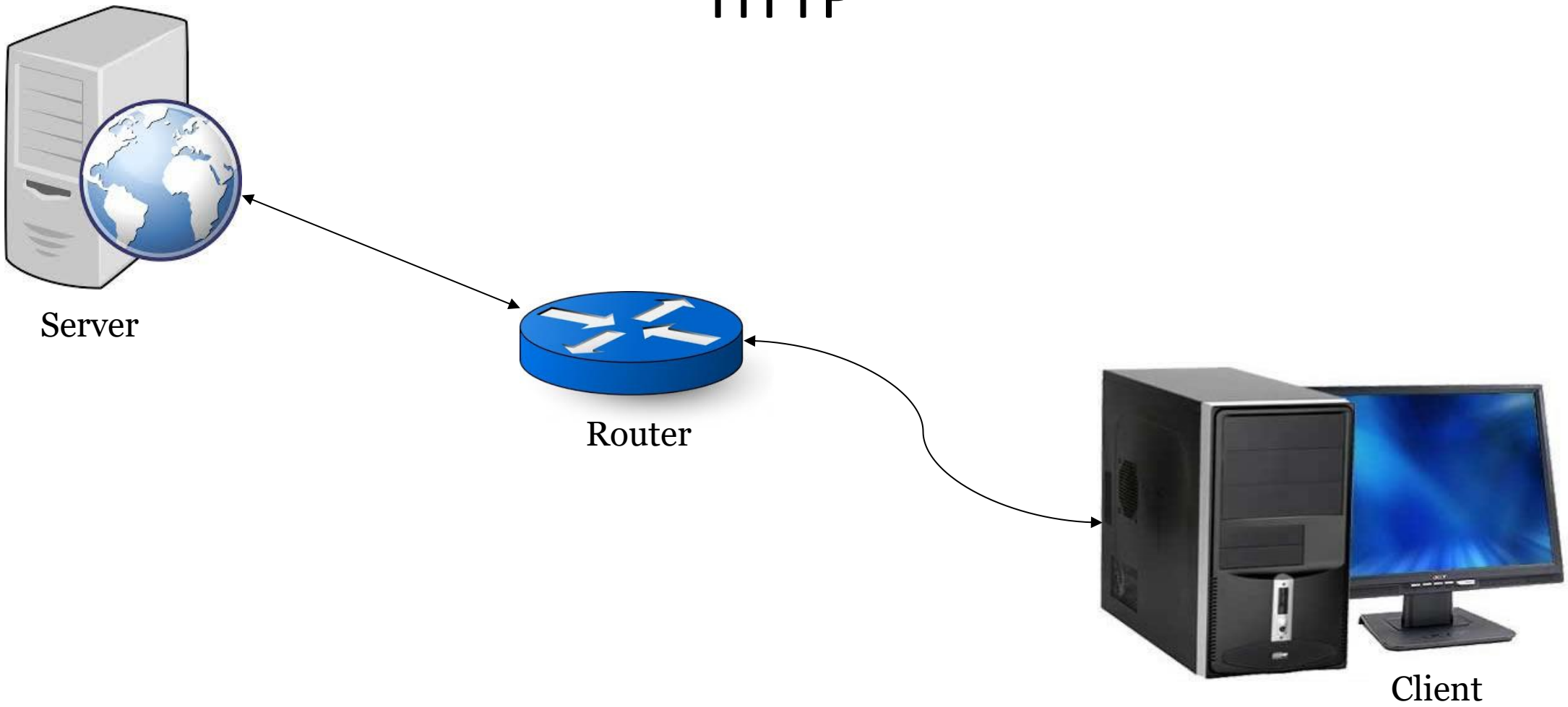
What is WWW?

- It is a world-wide system of interconnected servers which distribute a special type of document
- Documents are marked-up to indicate formatting (Hypertexts)
- This idea has been extended to embed multimedia and other content within the marked-up page

What is HTTP?

- HTTP is WWW's application layer protocol
- HyperText Transfer Protocol (HTTP) to transfer HyperText Markup Language (HTML) pages and embedded objects
- Works on a client-server paradigm
- Needs reliable transport mechanism (TCP)

HTTP



HTTP

Note: HTTP server often runs on port 80

Server

Router

Client

HTTP

Server

Note: HTTP server often runs on port 80

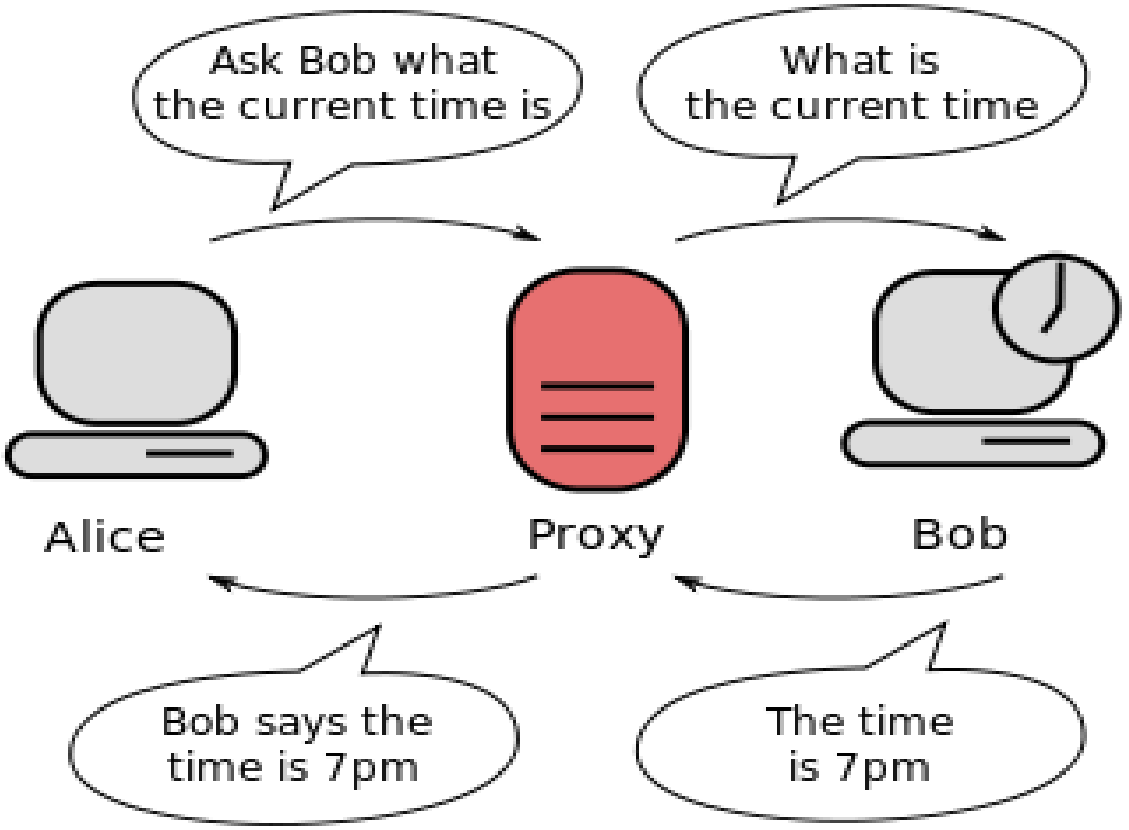
Router

Note: Client can use any unrestricted port
Generally >1024

Client

Proxy

- Acts as intermediary between client and server.



Benefits of a proxy

- Hide your internal network information (such as host names and IP addresses)
- You can set the proxy to require user authentication
- The proxy provides advanced logging capabilities
- Proxy helps you control which services users can access
- Proxy-caches can be used to save bandwidth
- Get access to blocked resources

A video to understand the concept
https://www.youtube.com/watch?v=5cPIukqXe5w

Risks of using a Proxy

Free proxy server risks

Browsing history log

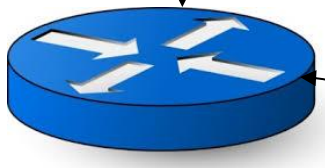
No encryption

HTTP with proxy

Server

Note: HTTP server often runs on port 80

Proxy listens on a port (>1024) and talks to server on another (>1024)



Router

Note: Client can use any unrestricted port
Generally >1024

Proxy

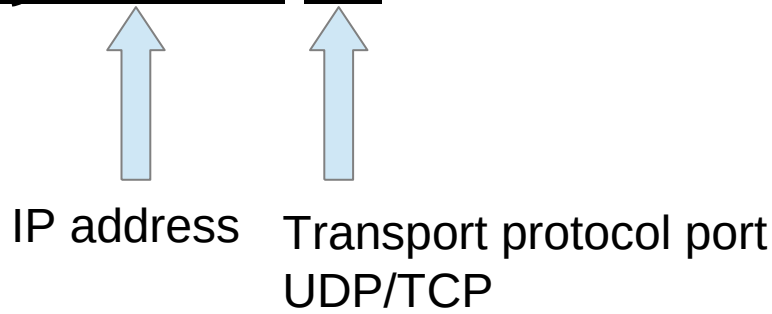
Client

What is a port?

- A port is an application-specific or process-specific software construct serving as a communications endpoint
- The purpose of ports is to uniquely identify different applications or processes running on a single computer and thereby enable them to share a single physical connection to a packet-switched network like the Internet

Ports continued

- Port only identifies processes/applications
- With regard to the Internet, ports are always used together with IP
- Notation 192.168.1.1:80



Socket programming

- These are software constructs used to create ports and perform operations on them
- We will talk about these types of sockets:
    - Datagram socket
    - Stream socket
    - SSL sockets

Datagram sockets

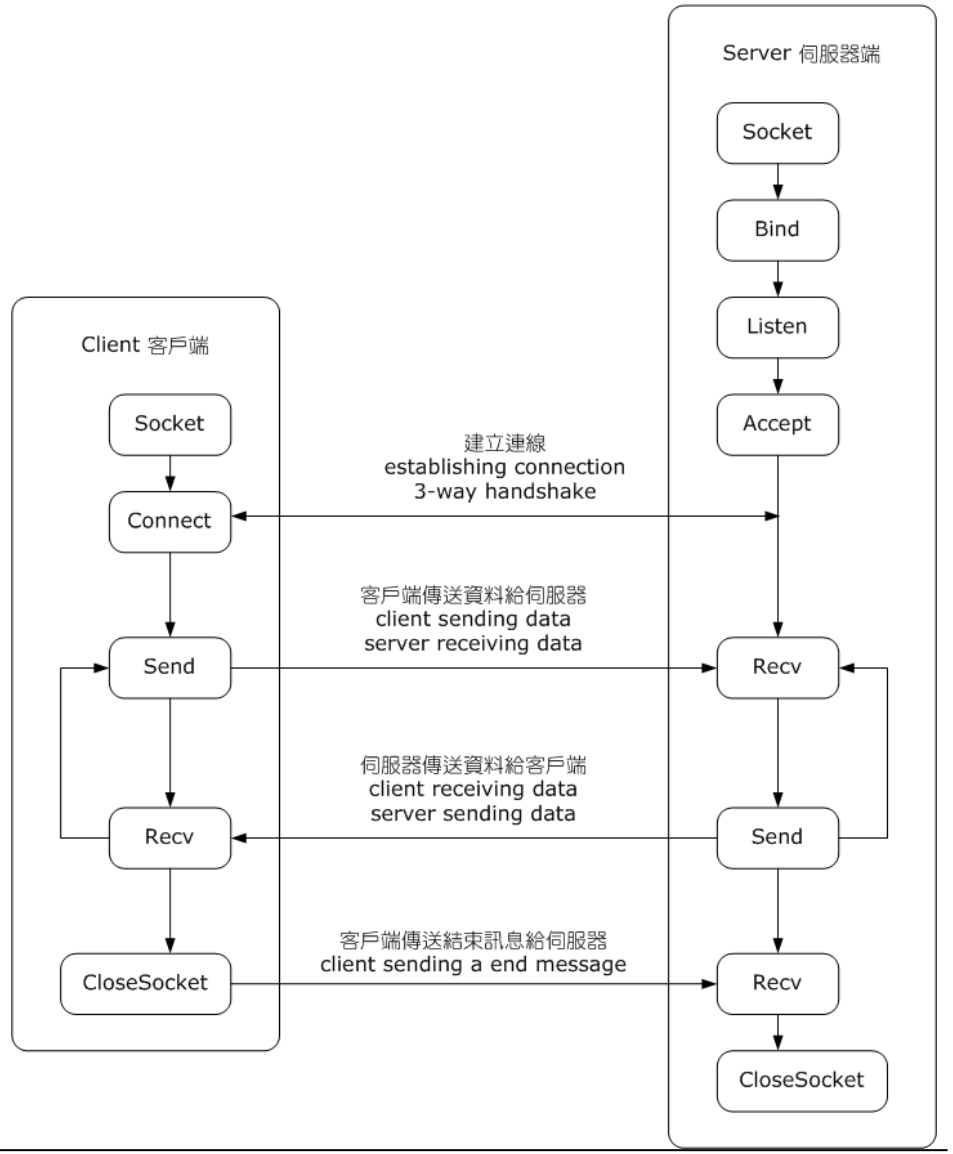
- Datagram sockets use UDP
- They are connectionless
- Do not guarantee in order delivery
- No form of loss recovery
- No congestion control
- No flow control

Stream sockets

- Stream sockets use TCP protocol
- Connection oriented sockets
- In order and guaranteed delivery
- Error identification and recovery
- Congestion control
- Flow control
- SSL sockets are similar to stream sockets, but include functions to handle encryption

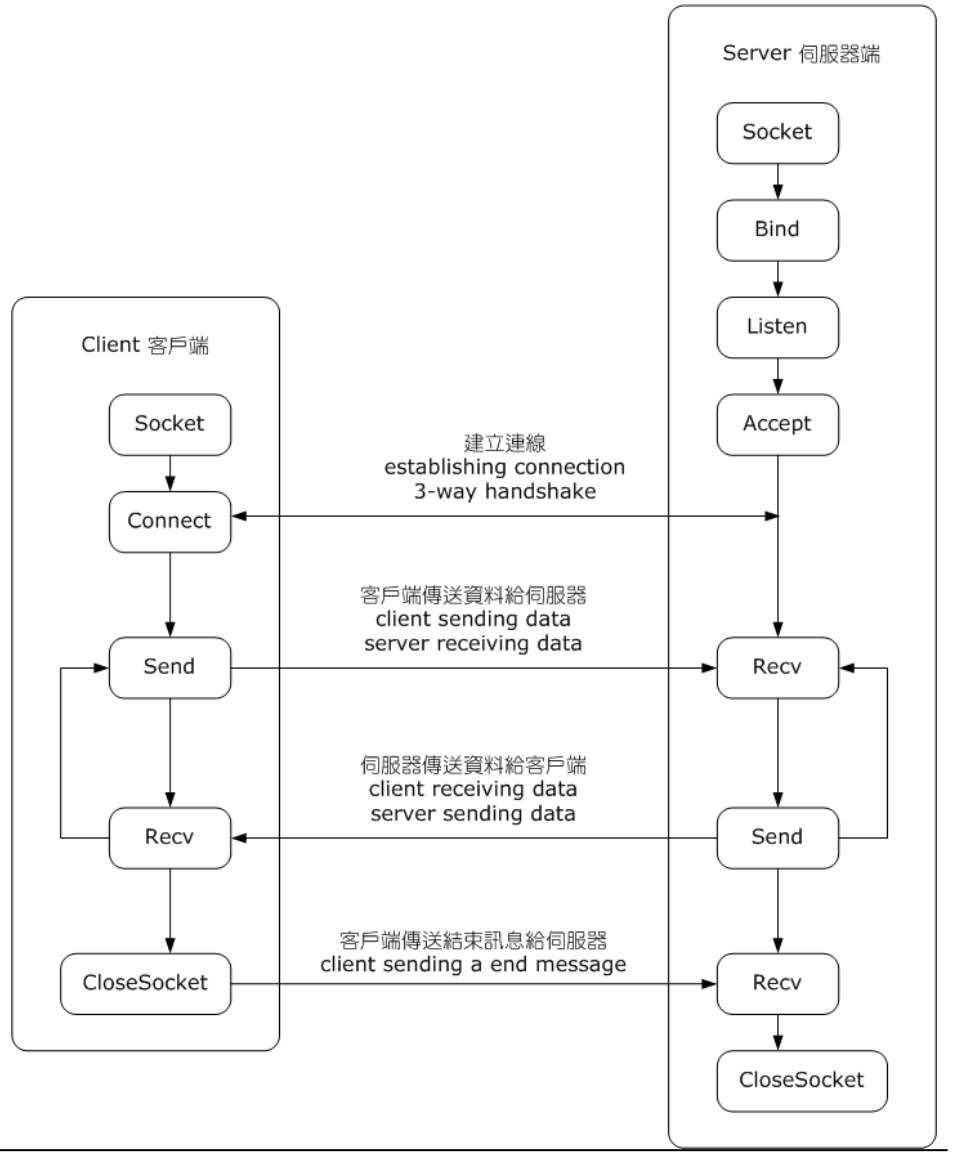
Important socket calls

- socket
- bind
- listen
- accept
- connect
- send
- recv



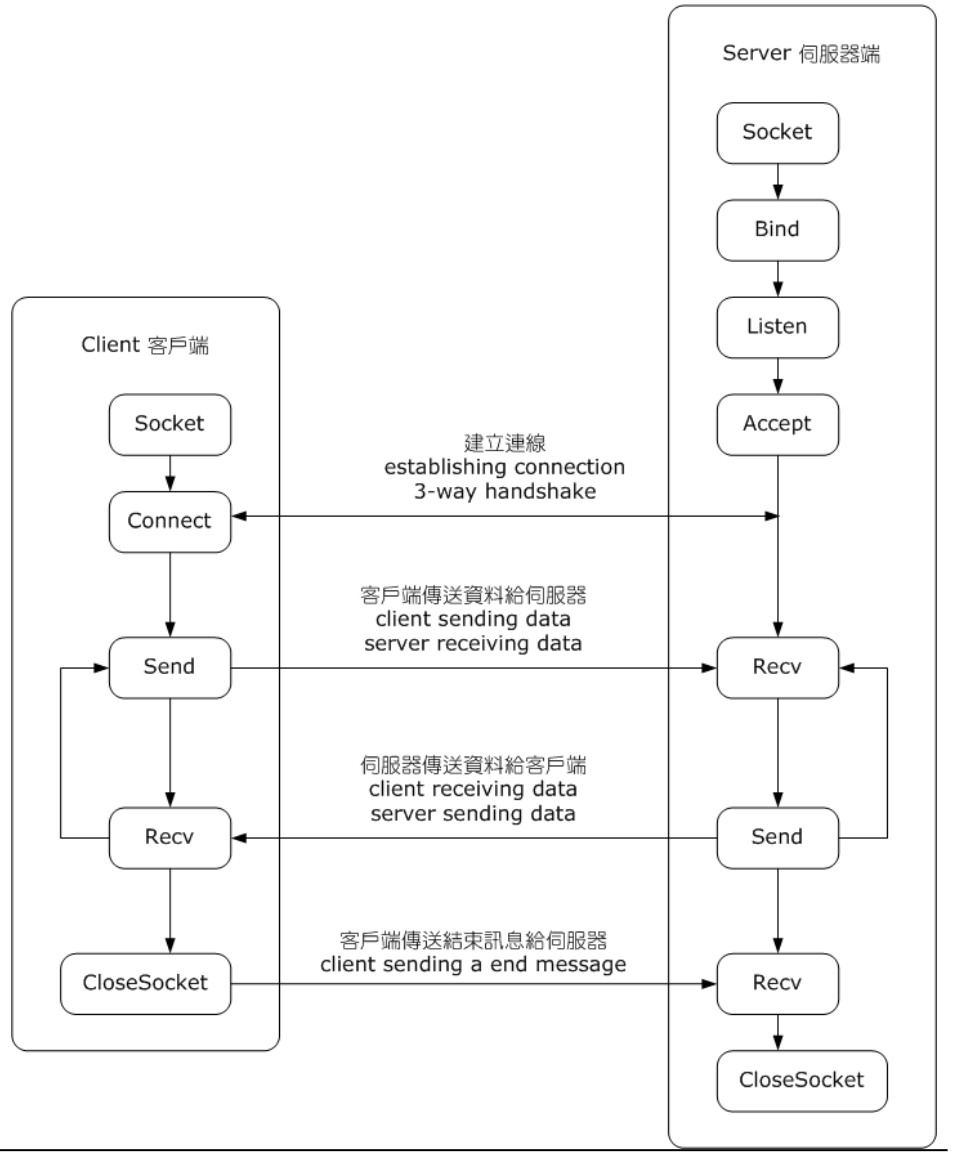
Socket programming calls

- **socket()**
  - Takes as input
    - Address family (=AF_INET)
    - Socket type (=SOCK_STREAM)
  - Returns
    - A socket object



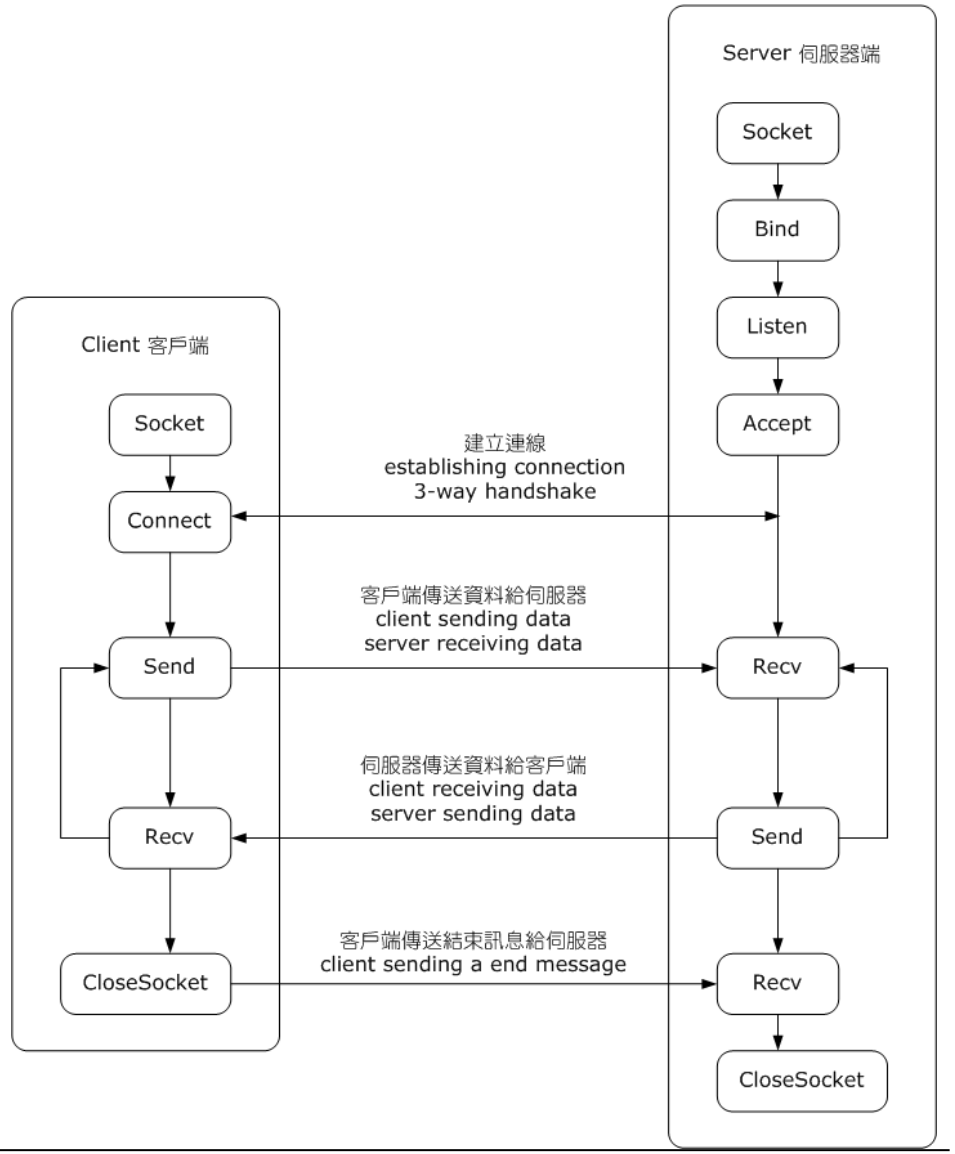
Socket programming calls

- **bind()**
  - Takes as input
    - address/port tuple (for AF_INET)
- What does this do?
  - Associate the socket with an address/port tuple



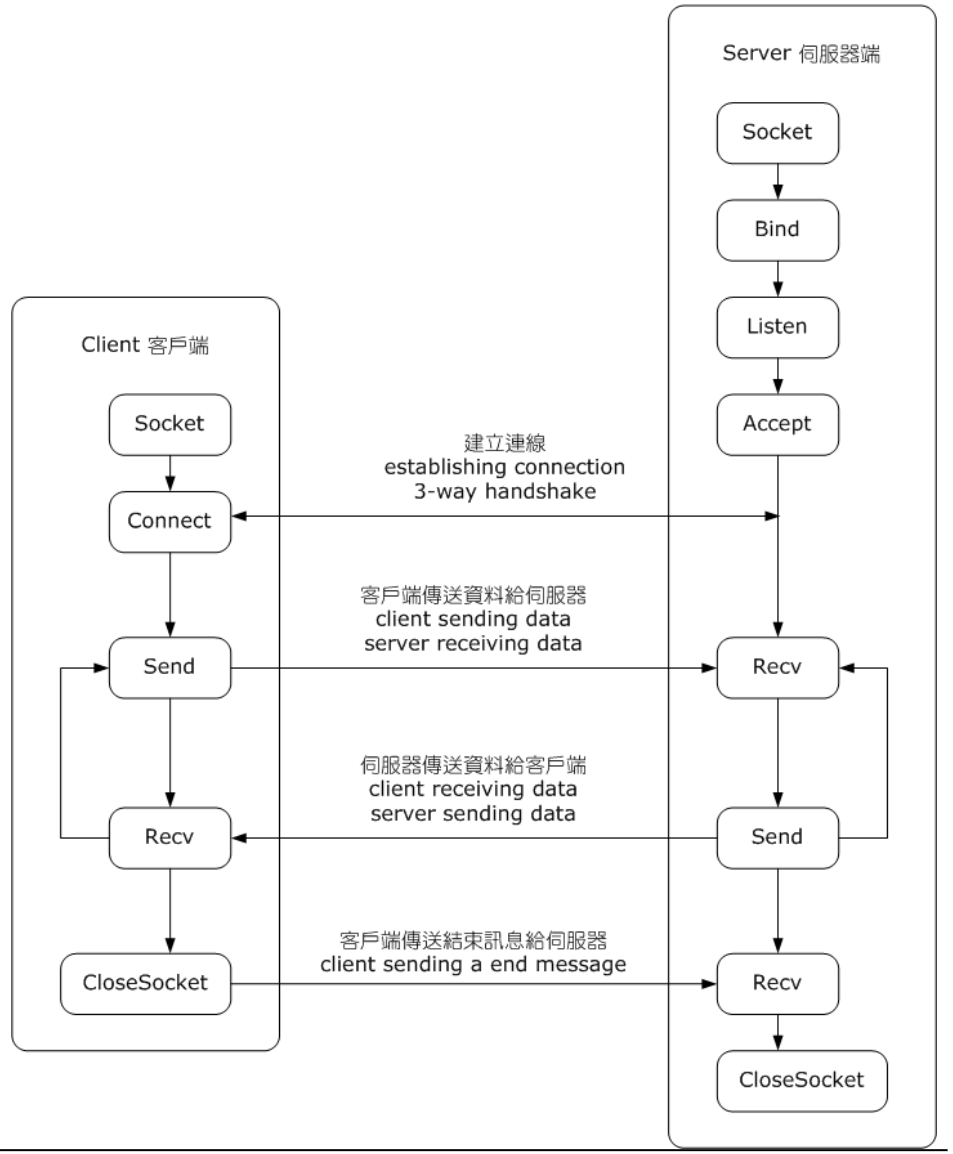
Socket programming calls

- **listen()**
  - Takes as input
    - Backlog (max queue of incoming connection)
- This must run at the server side to listen to incoming connection



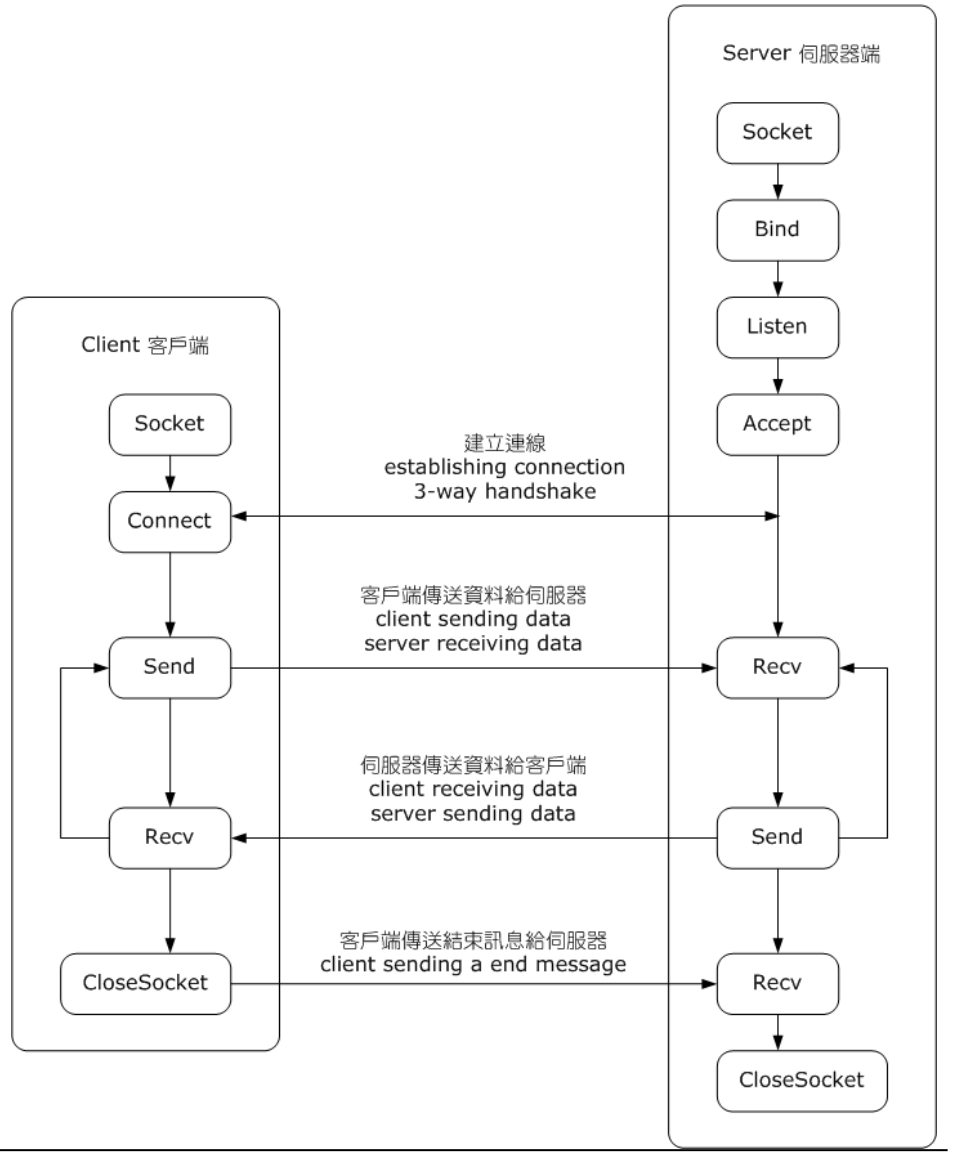
Socket programming calls

- **connect()**
  - Takes as input
    - Address/port tuple
- What does this do?
  - Attempts to setup a connection with the other end



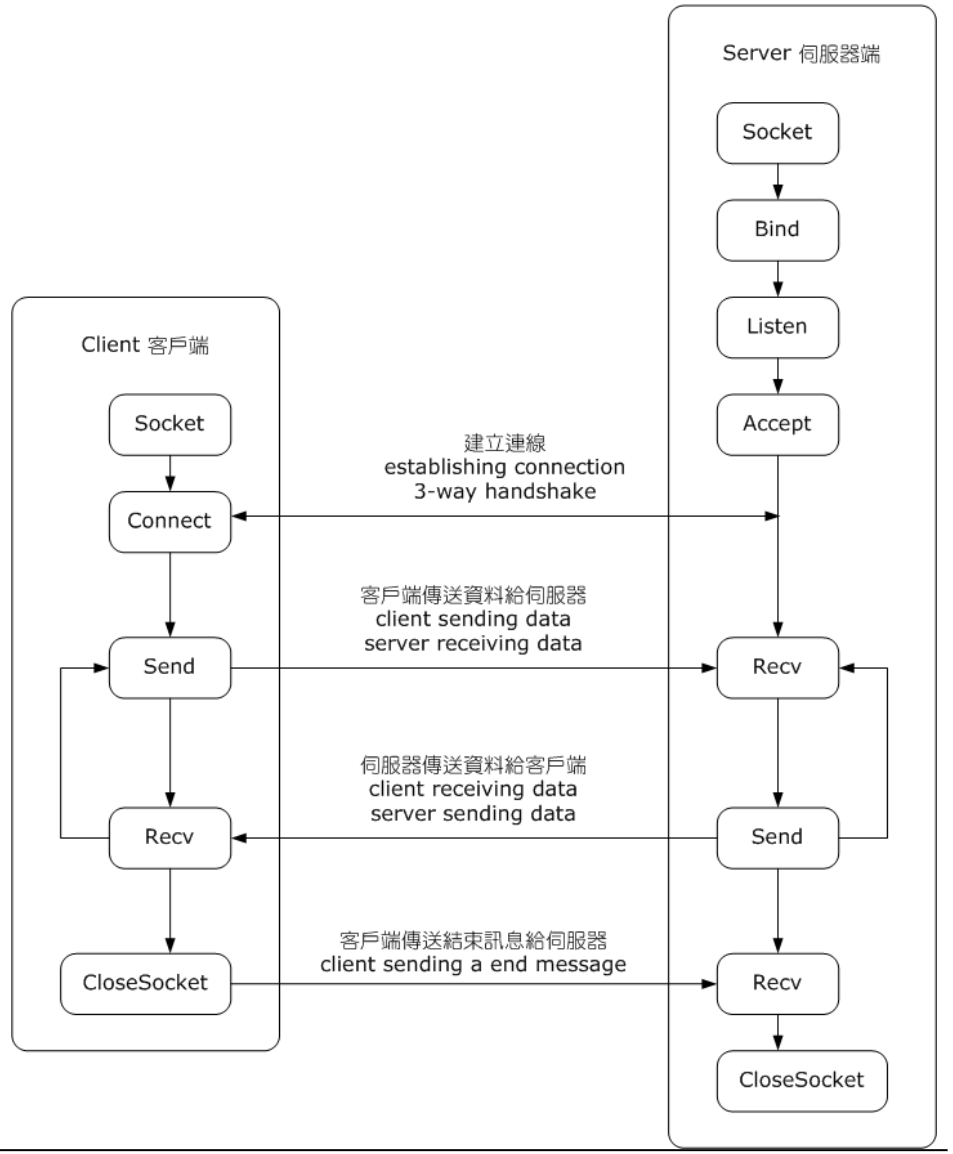
Socket programming calls

- **accept()**
  - No input
  - Returns
    - conn - a new socket object
    - address - address/port tuple
- Reads through the backlog and picks one from the list to connect to it.
- Runs at the server side



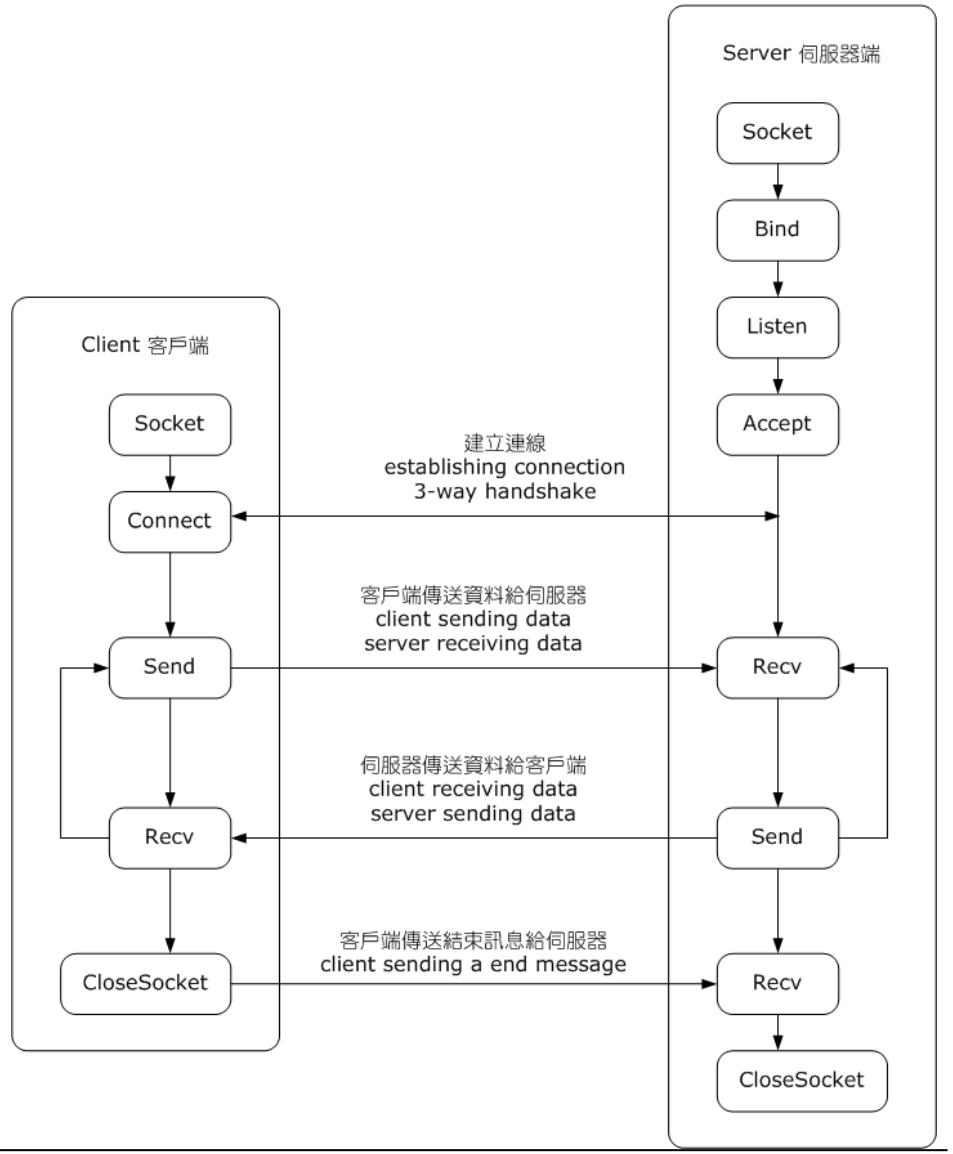
Socket programming calls

- **send()**
  - Takes as input
    - Message
  - Returns
    - Number of bytes sent
- Send is always best effort. If it cannot send the whole message, the returned value is smaller.



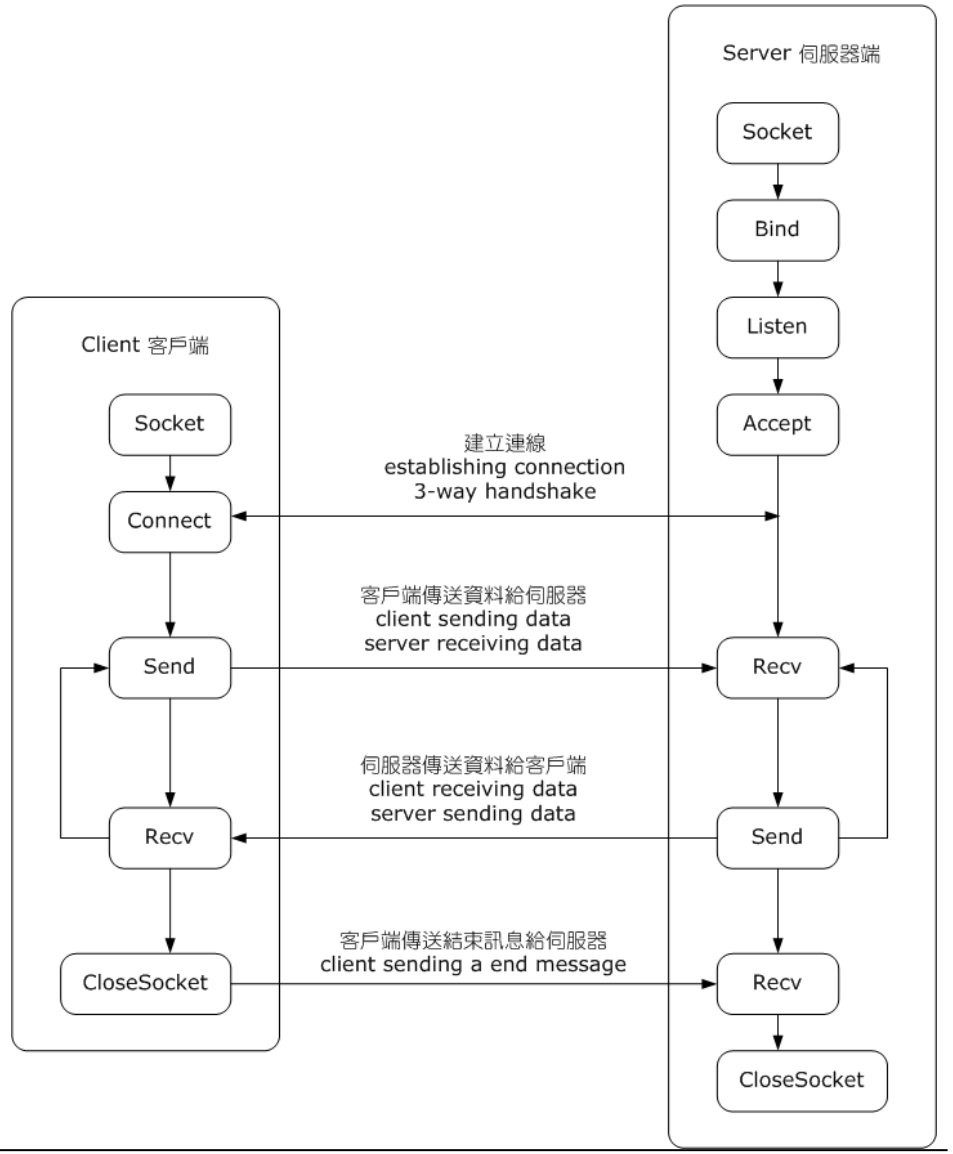
Socket programming calls

- **recv()**
  - Takes as input
    - Max buffer length
  - Returns
    - bytes object representing the data received



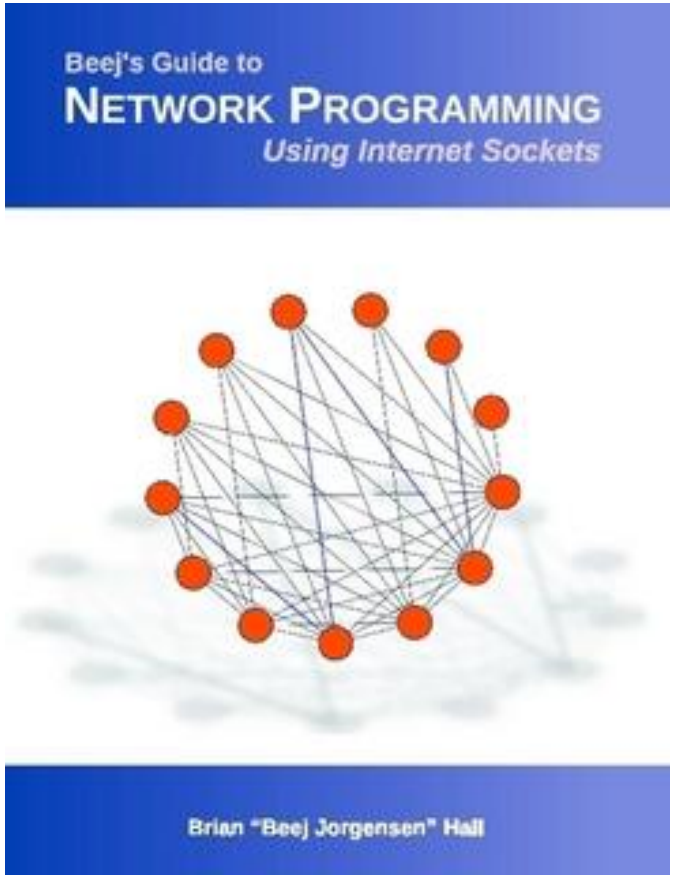
Socket programming calls

- **close()**
  - No input
- Marks the socket as closed



Socket programming resource

- Helpful guide linked from the assignment text: Beej's Guide to Network Programming
- Based on C, but can be used as a foundation for other languages



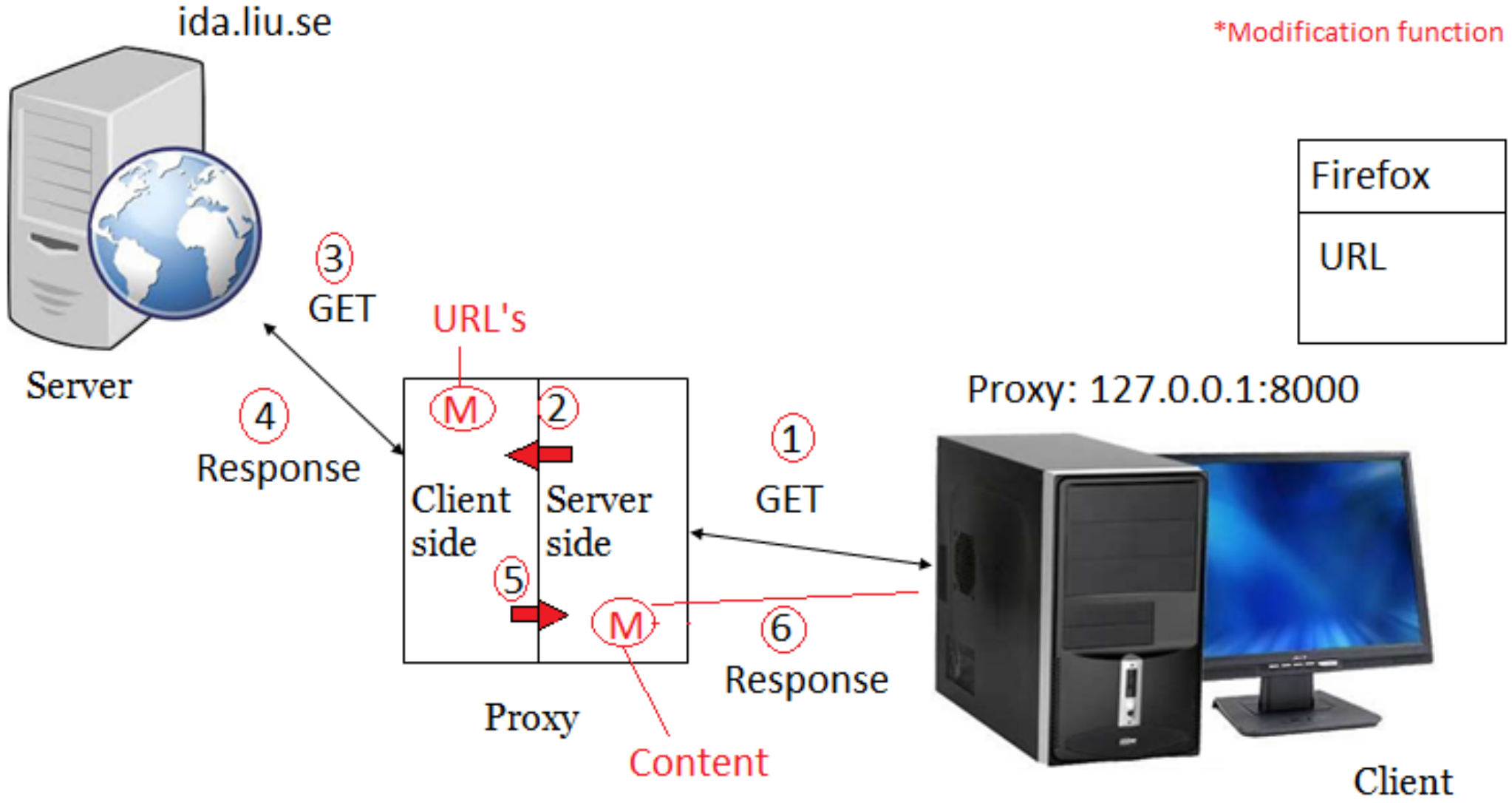
Assignment 2: Simple Web (HTTP) proxy

- Build a properly functioning Web proxy for simple Web pages, and then use your proxy to change some of the content before it is delivered to the browse

- Change all occurrences of "Smiley" on a Web page into "Trolly", and all occurrences of "Stockholm" into "Linköping". And if you find any JPG images of Smiley (linked or embedded), then you should replace them with your favorite troll image file (JPG, GIF, or PNG) from the Internet.

- For the sake of simplicity, we will restrict ourselves only to HTTP (not HTTPS), and consider only basic text and HTML pages with a few images.

General overlay



Assignment 2: Description

- Socket programming is the key
- Build a proxy to which a user can connect to
- The proxy connects to the web server on user's behalf (recollect how proxy works)
- Proxy receives the response from the web server
- Proxy forwards the HTTP response (from the web server) to the user with all occurrences of *"Smiley"* replaced by *"Trolly"*, and all occurrences of *"Stockholm"* replaced by *"Linköping"*

Assignment 2: requirements

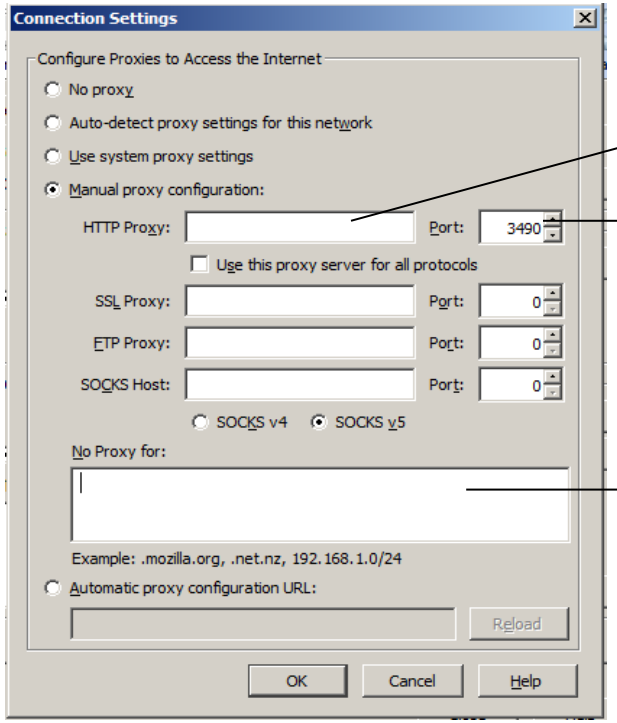
1. The proxy should support both HTTP/1.0 and HTTP/1.1.
2. Handles simple HTTP GET interactions between client and server
3. Consider how your proxy handles commonly occurring HTTP response codes, such as 200 (OK), 304 (Not Modified), and 404 (Not Found)
4. Imposes no limit on the size of the transferred HTTP data
5. Use only the *basic* libraries available for socket programming

Assignment 2: requirements

6. Is compatible with all major browsers (e.g. Internet Explorer, Mozilla Firefox, Google Chrome, etc.) without the requirement to tweak any advanced feature
7. Allows the user to select the proxy port (i.e. the port number should not be hard coded)
8. Is smart in selection of what HTTP content should be searched for the forbidden keywords. For example, you probably agree that it is not wise to search inside compressed or other non-text-based HTTP content such as graphic files, etc.
9. You do not have to relay HTTPS requests through the proxy

Browser configuration

- Proxy listens on a particular port



127.0.0.1

Proxy's port number

Make sure it is blank

HTTP basics

- Recollect lab 1. It contains things what you need in lab 2.
- HTTP request
    - Get
        - Syn, SynAck, Ack

```
⊞ Transmission Control Protocol, Src Port: 50139 (50139), Dst Port: http (80), Seq: 1, Ack: 1, Len: 276
⊟ Hypertext Transfer Protocol
  ⊞ GET /vod/final_1.3.f4m HTTP/1.1\r\n
    Host: 130.236.182.199\r\n
    Connection: keep-alive\r\n
    User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.103 Safari/537.36\r\n
    Accept-Encoding: gzip,deflate,sdch\r\n
    Accept-Language: en-US,en;q=0.8,ms;q=0.6\r\n
    \r\n
    [Full request URI: http://130.236.182.199/vod/final_1.3.f4m]
```

HTTP basics

- HTTP response
  - OK

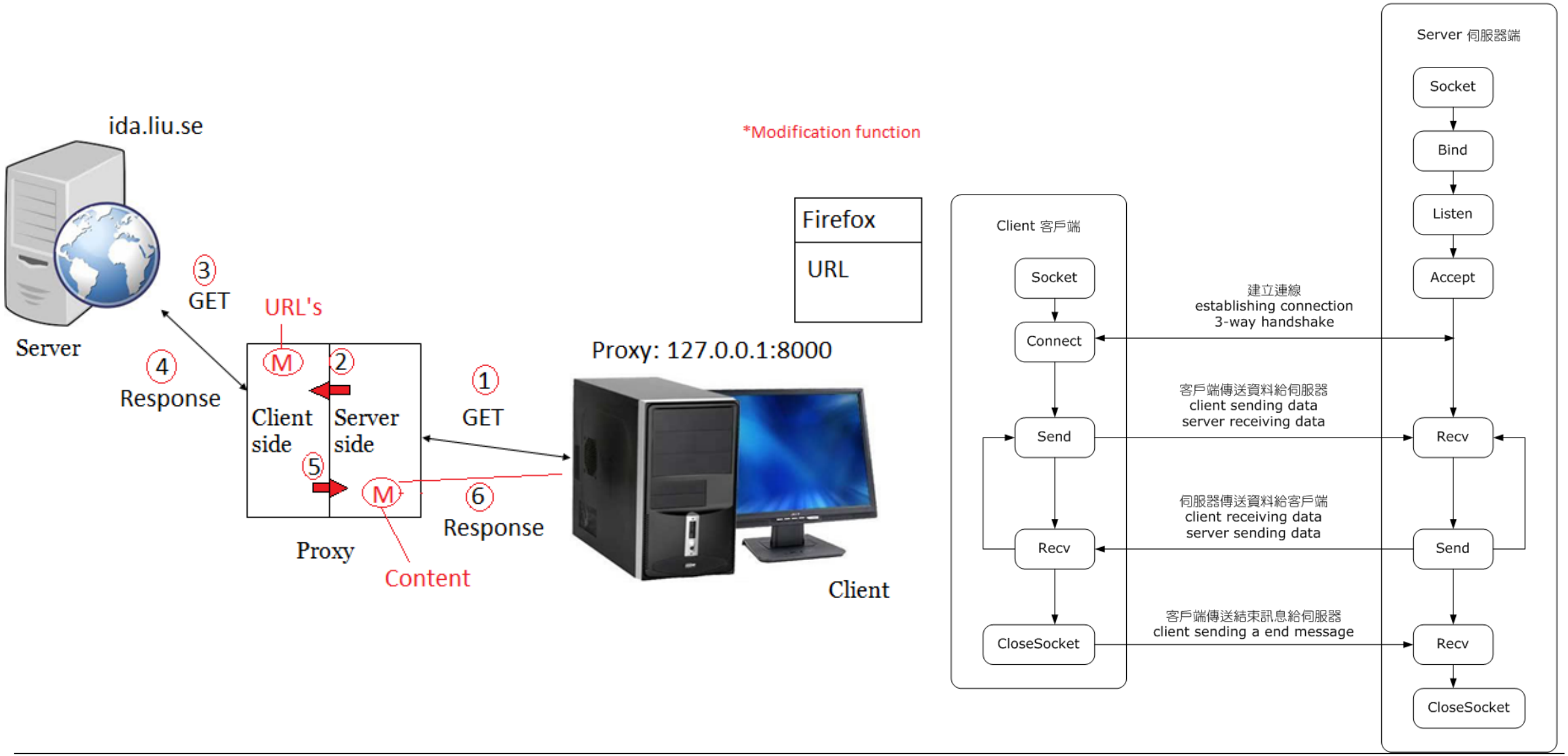
```
⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: 50139 (50139), Seq: 4381, Ack: 277, Len: 1215
⊞ [4 Reassembled TCP Segments (5595 bytes): #248(1460), #249(1460), #251(1460), #252(1215)]
⊟ Hypertext Transfer Protocol
  ⊞ HTTP/1.1 200 OK\r\n
    Date: Sun, 07 Sep 2014 10:06:36 GMT\r\n
    Server: Apache/2.2.17 (Unix) DAV/2\r\n
  ⊞ Content-Length: 5354\r\n
    Last-Modified: Tue, 04 Feb 2014 12:25:40 GMT\r\n
    Keep-Alive: timeout=15, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/xml\r\n
    \r\n
```

HTTP basics

- HTTP 1.0 vs HTTP 1.1
    - Many differences read http://www8.org/w8-papers/5c-protocols/key/key.html
    - For this assignment
        - Connection: close
            - Handshake-Get-response-OK-Teardown
        - Connection: keep-alive
            - Handshake-Get-response-OK-wait-Get-response
- What should you use for the proxy?

How to handle connections

- With connection: keep-alive, the connection is kept open. You are responsible to figure out when the response is completed.
- With connection: close, the server closes the connection after the response is sent.
- How can you enforce connection: close on HTTP 1.1?


ida.liu.se
Server
3
GET
4
Response
URL's
M
2
Client side
Server side
5
M
Proxy
Content
*Modification function
Firefox
URL
Proxy: 127.0.0.1:8000
1
GET
6
Response
Client
Client 客戶端
Socket
Connect
Send
Recv
CloseSocket
建立連線
establishing connection
3-way handshake
客戶端傳送資料給伺服器
client sending data
server receiving data
伺服器傳送資料給客戶端
client receiving data
server sending data
客戶端傳送結束訊息給伺服器
client sending a end message
Server 伺服器端
Socket
Bind
Listen
Accept
Recv
Send
Recv
CloseSocket

Various Steps involved:

Proxy (Server Side)

- Allocate IP address and Port No. (Tuple)
- Binding the socket
- Listen for incoming connections
- Add proxy settings in the web browser
- Receive Request(s) from the user (browser)
- Decode and parse through received GET
- Modify URL (depends)
- Encode (Opt) and send the request to Proxy (Client Side)
- Receive response from Proxy (Client Side) and decode information (if not done earlier)
- Modify text (not image file name) (if large, store all of them in a temporary buffer)
- Encode and send to browser
- Close the connection

Proxy (Client Side)

- Create another socket for proxy-actual server communication
- Prepare GET, encode, and send to actual server
- Receive response from Actual Server, decode (optional), and send to Proxy (Server Side)
- Close the connection.

Some Common errors.

- Content length modification (Linkoping and Stockholm)
- Be careful while modifying information (issues with images)
- To use proxy for one port multiple times, use reuseaddress command
- While loop can be a good choice to collect large data
- Do not encode and decode the images
- Issue with the firefox, then try with other browser like chrome
- Clear the cache to see the updated outcomes

General overlay

Server

Server side: listens on a port, accepts, receives, forwards to client side

Client side | Server side

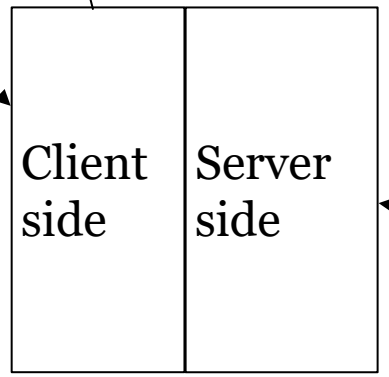
Proxy

Client

General overlay

Client side: connects to the server, send request, receive response,
Forwards to server side

Server



Proxy

Client

Content filtering

- Need to be able to filter both based on URL and content
- In which of the two halves of the proxy will you implement filtering based on URL?
- In which of the two halves of the proxy will you implement content filtering?
- How to actually do content filtering?

Content filtering

- Response from the server comes in segments
- Remember TCP segmentation?
- Reconstruct the message in a temporary buffer
- No dynamic sizing of buffer, chose a value and stick with it
- Do not type-cast non-text data!
- Then run filtering only on the text message

Text vs binary data

- Content-type header
- Differentiate content type
    - Run/don't run filtering

```
⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: 50139 (50139), Seq: 4381, Ack: 277, Len: 1215
⊞ [4 Reassembled TCP Segments (5595 bytes): #248(1460), #249(1460), #251(1460), #252(1215)]
⊟ Hypertext Transfer Protocol
  ⊞ HTTP/1.1 200 OK\r\n
    Date: Sun, 07 Sep 2014 10:06:36 GMT\r\n
    Server: Apache/2.2.17 (Unix) DAV/2\r\n
  ⊞ Content-Length: 5354\r\n
    Last-Modified: Tue, 04 Feb 2014 12:25:40 GMT\r\n
    Keep-Alive: timeout=15, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/xml\r\n
    \r\n
```

Debugging advice

- Stick to simple web pages initially
- Debug incrementally
- Check and double check request string for formatting and completeness
- Source of many errors like 'server closed connection unexpectedly'
- If developing on own computers, use Wireshark to debug. Can save a lot of time!

Debugging advice

- HTTP vs HTTPS
    - Requirements do not ask for a proxy which works with HTTPS
    - Avoid testing on any site to which you are signed in
    - Restrict yourselves to simple sites and basic test cases

Debugging advice

- Header manipulation
    - First thing to check at a proxy is the URL that it sends out to the server
    - It might require different manipulations based on the site. Be sure that you test for all sites mentioned in the test scenario
    - If you change some fields in the header, the packet length has to be changed or brought back to the original length

Basic Example Socket Programming Server Side

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('127.0.0.1', 12349))
s.listen(5)

while True:
    # now our endpoint knows about the OTHER endpoint.
    clientsocket, address = s.accept()
    print(f"Connection from {address} has been established.")
    clientsocket.send(bytes("Hello", "utf-8"))
```

Basic Example Socket Programming Client Side

```
import socket


s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('127.0.0.1', 12349))


msg = s.recv(1024)
print(msg.decode("utf-8"))
```

Questions?

e-mail your TA (subject “TDTS04 …”)

www.liu.se

LINKÖPING UNIVERSITY