Introduction to Lab 2 and Socket Programming

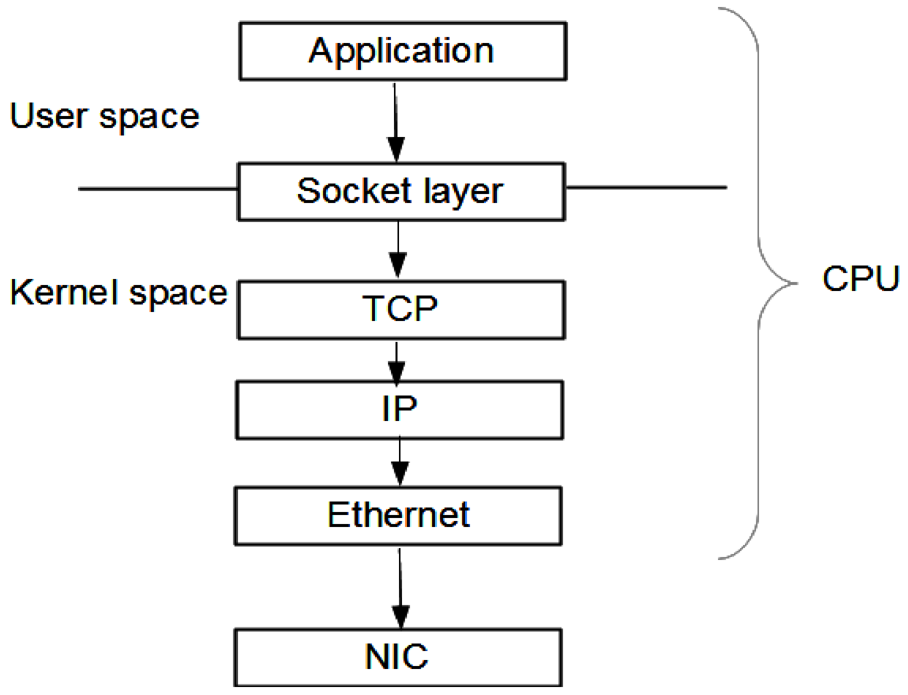
-Vengatanathan Krishnamoorthi

Before we start..

- Soft deadline for lab 2- February 13
- Finish assignment 1 as soon as possible if you have not yet.
- Hard deadline for assignments- March 11
- If you still have issues with login, registration or group write an email immediately to your TA.
- For change of group/no group-mate remember to cc both new and old group-mates.

What will we do in lab 2?

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



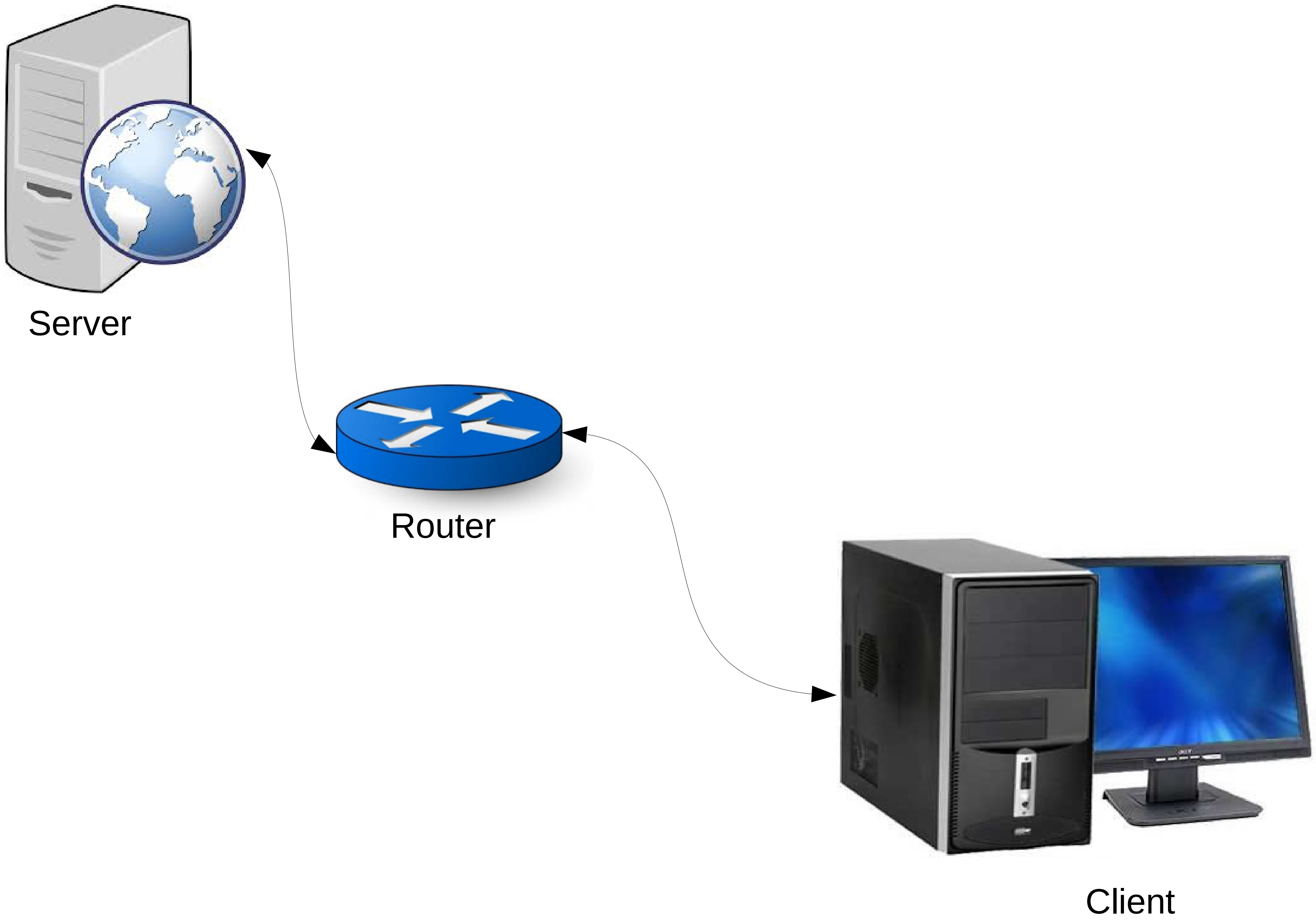
What is WWW?

- 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?

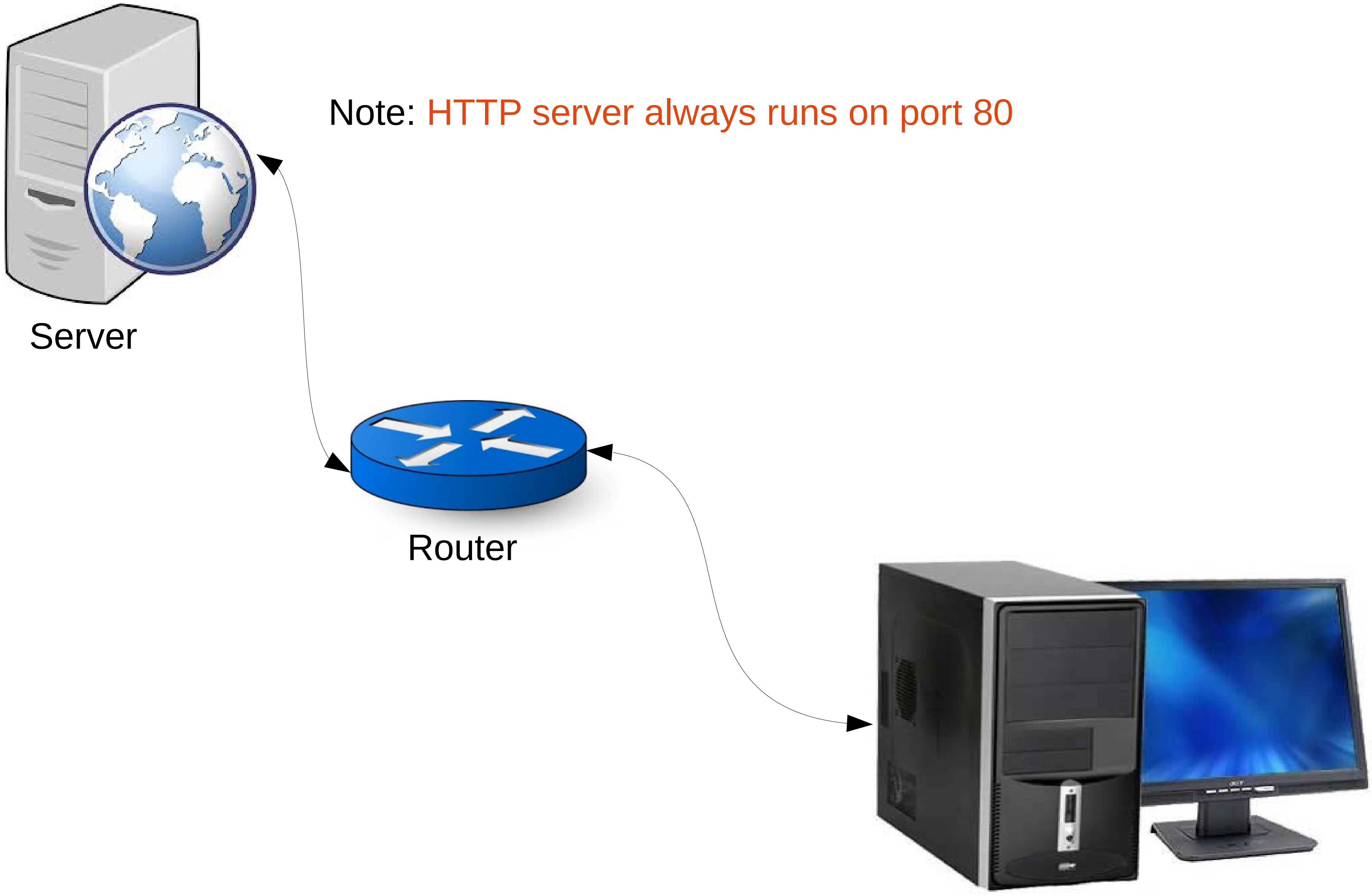
- HTTP is WWW's application layer protocol.
- HyperText Transfer Protocol (HTTP) to transfer HyperText Markup (HTML) pages and embedded objects.
- Works on a client-server paradigm.
- Needs reliable transport mechanism (TCP).

HTTP



HTTP

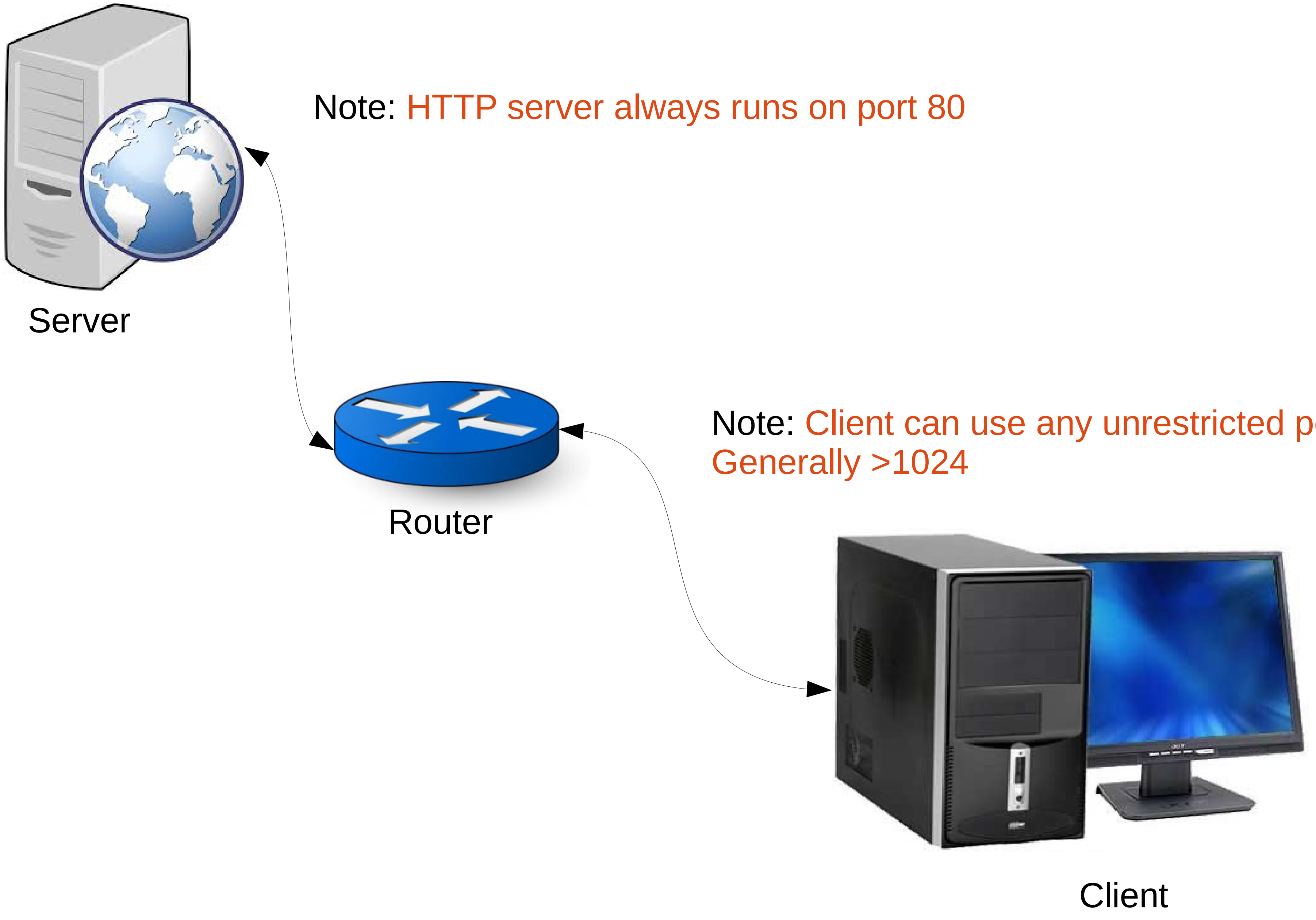
Note: HTTP server always runs on port 80



HTTP

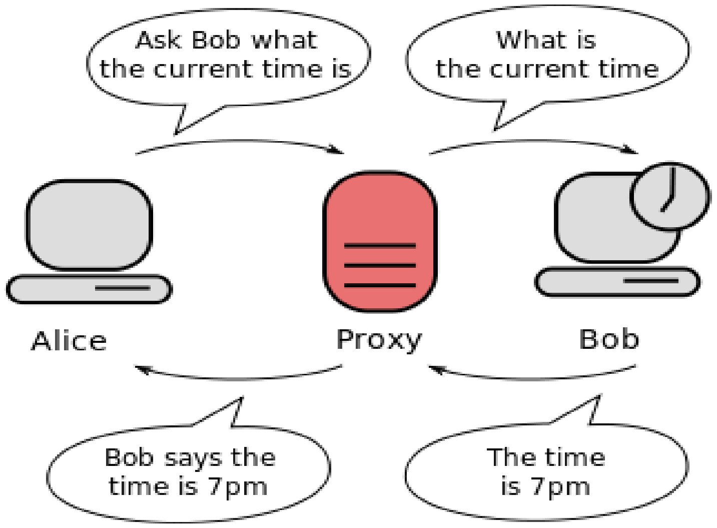
Note: HTTP server always runs on port 80

Note: Client can use any unrestricted port
Generally >1024



Proxy

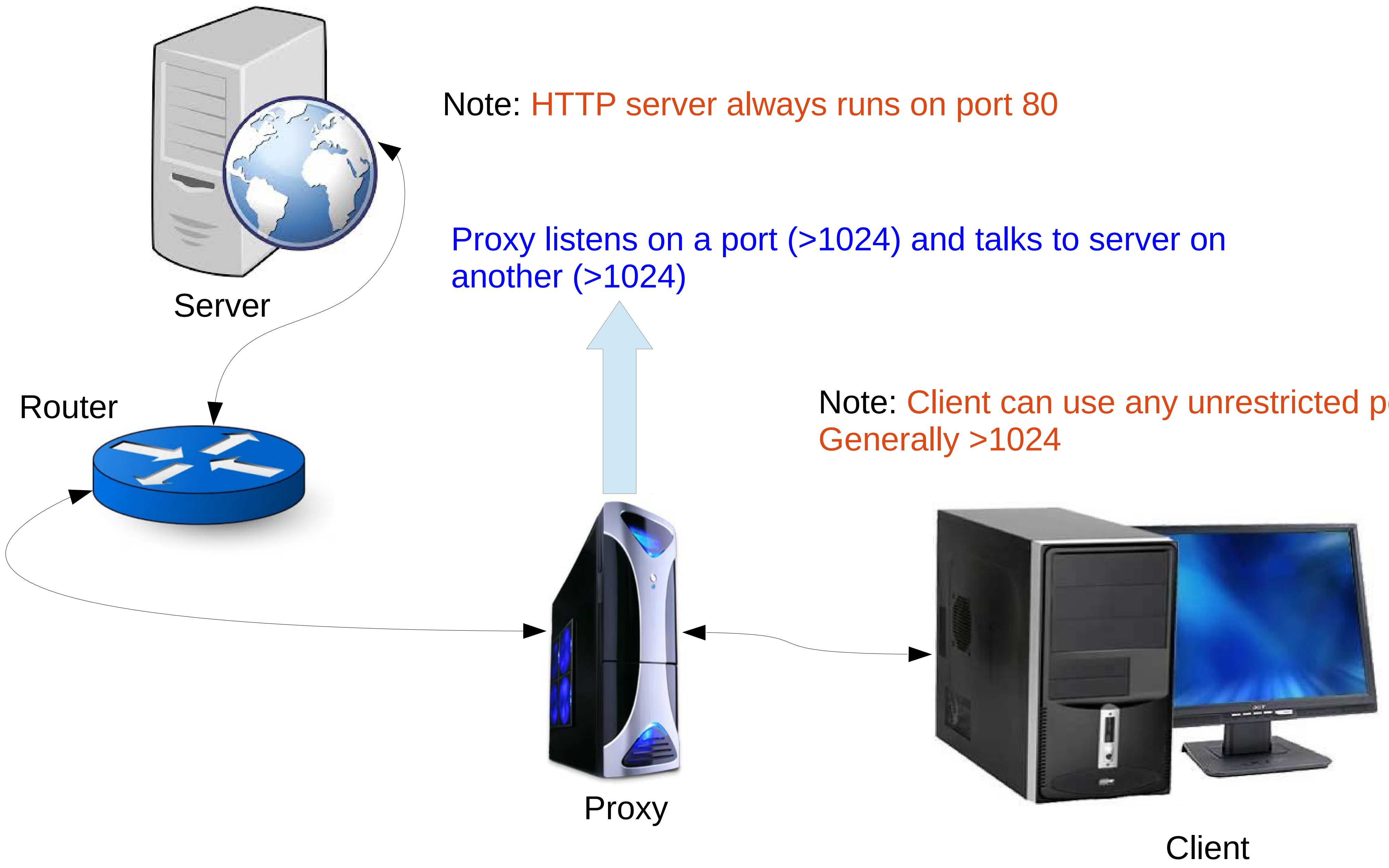
- Acts as intermediary between client and server.



Benefits of a proxy

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

HTTP with proxy



Note: HTTP server always runs on port 80

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

Note: Client can use any unrestricted port
Generally >1024

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.

Port cont..

- Port only identifies processes/applications.
- W.r.t. Internet, ports are always used together with IP.
- Notation 192.168.1.1:80

IP address Transport protocol port
UDP/TCP

Socket programming

- These are software constructs used to create ports and perform operations on them.
- It is a way to speak to other programs using standard Unix file descriptors.
- We will talk about two types of sockets:
  - Datagram socket
  - Stream socket

Datagram sockets

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

Datagram sockets

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

Stream sockets

- Connection oriented sockets
- In order and guaranteed delivery
- Error identification and recovery
- Congestion control
- Flow control
- Stream sockets use TCP protocol

Structs

- Structs are used to pass values to most socket functions.
- Read up on the following structs
    - addrinfo (contains address related information)
    - sockaddr (contains socket address)

```
struct addrinfo {
    int              ai_flags;     // AI_PASSIVE, AI_CANONNAME, etc.
    int              ai_family;    // AF_INET, AF_INET6, AF_UNSPEC
    int              ai_socktype;  // SOCK_STREAM, SOCK_DGRAM
    int              ai_protocol;  // use 0 for "any"
    size_t           ai_addrlen;   // size of ai_addr in bytes
    struct sockaddr *ai_addr;      // struct sockaddr_in or _in6
    char            *ai_canonname; // full canonical hostname

    struct addrinfo *ai_next;      // linked list, next node
};
```

```
struct sockaddr {
    unsigned short    sa_family;    // address family, AF_xxx
    char              sa_data[14];  // 14 bytes of protocol address
};
```

Socket programming calls

- **getaddrinfo()**
  - Get address information
  - Takes as input
    - Host name
    - Service type (HTTP) or only port number if local
    - Information about IP family(v4 or v6), type of socket. (struct addrinfo)
  - Returns
    - A pointer to a linked list. Lets call this 'result'

Socket programming calls

- **socket()**
  - Takes as input
    - Address family
    - Socket type
    - Protocol
  - Returns
    - File descriptor

Socket programming calls

- **bind()**
  - Takes as input
    - File descriptor number
    - Address information obtained from getaddrinfo()
    - Address length
  - Returns
    - -1 on error
- What does this do?
  - Associate the socket with a port number

Socket programing calls

- **connect()**
  - Takes as input
    - File descriptor number
    - Address information obtained from getaddrinfo()
    - Address length
  - Returns
    - -1 on error
- What does this do?
  - Attempts to setup a connection with the other end

Socket programing calls

- **listen()**
  - Takes as input
    - File descriptor (fd for the socket/port to listen)
    - Backlog (max queue of incoming connection)
  - Returns
    - -1 on error
- This must run at the server side to listen to incoming connection

Socket programing calls

- **accept()**
  - Takes as input
    - File descriptor number
    - Address information obtained from getaddrinfo()
    - Address length
  - Returns
    - -1 on error
- Reads through the backlog and picks one from the list to connect to it.
- Runs at the server side

Socket programing calls

- **send()**
  - Takes as input
    - File descriptor number
    - Message
    - Length
  - Returns
    - Number of bytes sent
- Send is always best effort. If it cant send the whole message, the value returned is smaller.

Socket programing calls

- **recv()**
  - Takes as input
    - File descriptor number
    - Buffer
    - Max buffer length
  - Returns
    - Number of bytes received
    - Or -1 on error

Socket programing calls

- **close()**
  - Takes as input
    - File descriptor
- Closes the stream socket (TCP connection tear down)

Assignment description

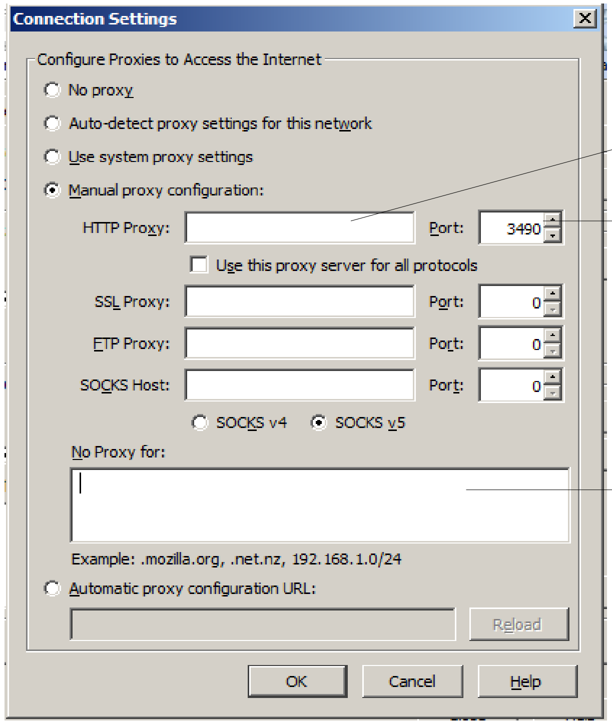
- Imagine that you are a conscientious Internet user who wishes to protect your friends and family from viewing inappropriate Web content. In particular, you want them to avoid any Web pages that might insult their intelligence. Specific examples that may come to mind are Web pages that mention SpongeBob, Britney Spears, Paris Hilton, or Norrköping. You must do your best to prevent their Web browsers from viewing these sites.

How do you do that?

- Well socket programming of course...
- Build a proxy to which an user can connect to
- The proxy connects to the server on user's behalf (recollect how proxy works)
- Proxy receives the response from the server
- Forwards only 'good' responses to the user
- Redirects in other case

Browser configuration

- Proxy listens on a particular port



Connection Settings

Configure Proxies to Access the Internet

- ○ No proxy
- ○ Auto-detect proxy settings for this network
- ○ Use system proxy settings
- [✓ Manual proxy configuration:]

HTTP Proxy:		Port:	3490

☐ Use this proxy server for all protocols

SSL Proxy:		Port:	0
FTP Proxy:		Port:	0
SOCKS Host:		Port:	0

○ SOCKS v4 [✓ SOCKS v5]

No Proxy for:

Example: .mozilla.org, .net.nz, 192.168.1.0/24

○ Automatic proxy configuration URL:

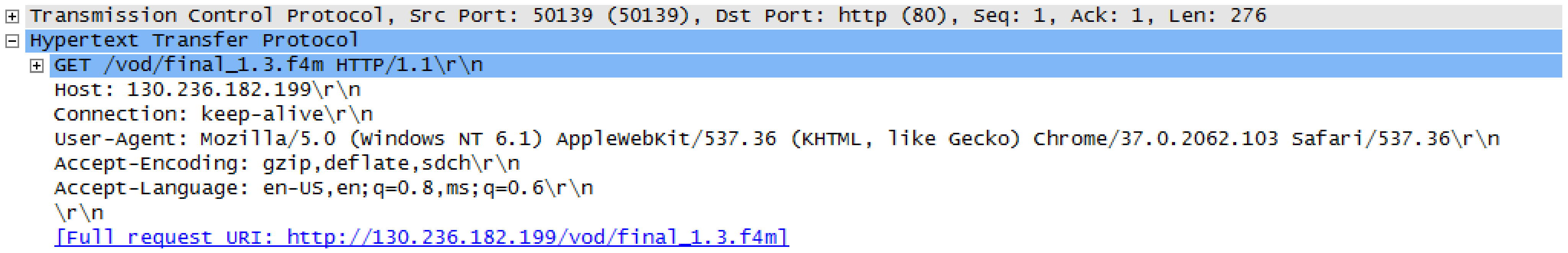
127.0.0.1

Proxy's port number

Make sure it is blank

HTTP basics

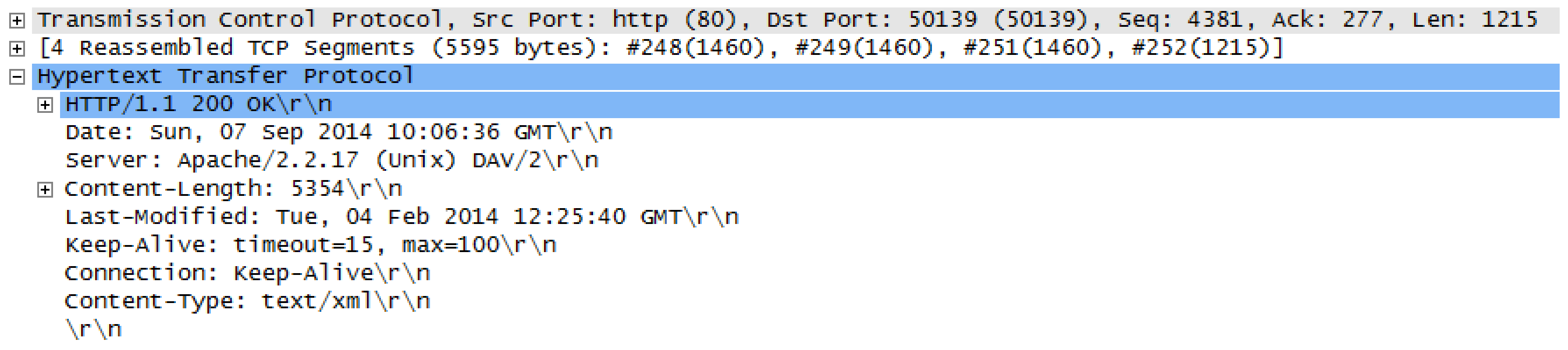
- Recollect lab 1. It contains things that you need in lab 2.
- HTTP request
  - Get
    - Syn, SynAck, Ack



```
⊞ 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?

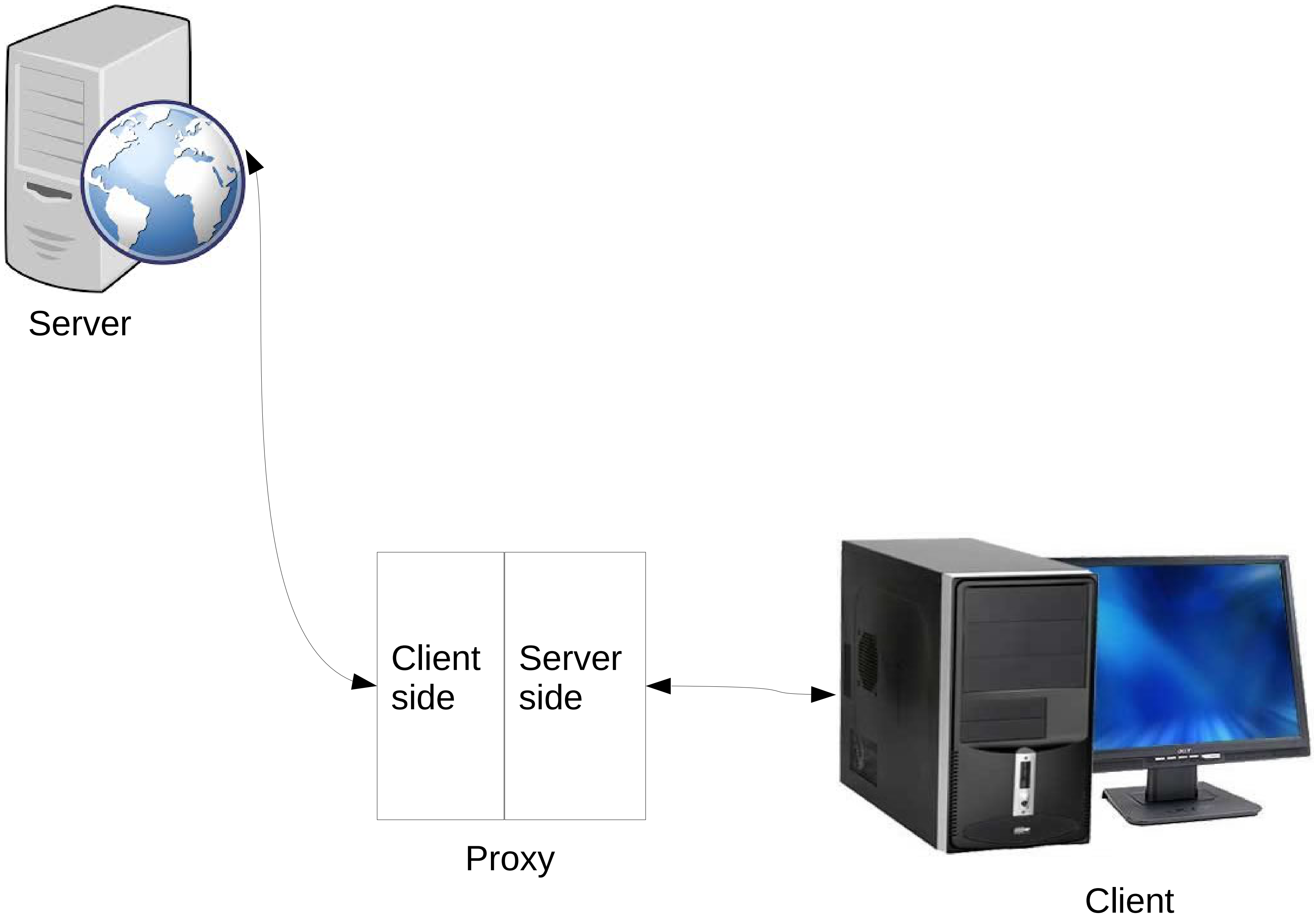
How to handle connections

- 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.

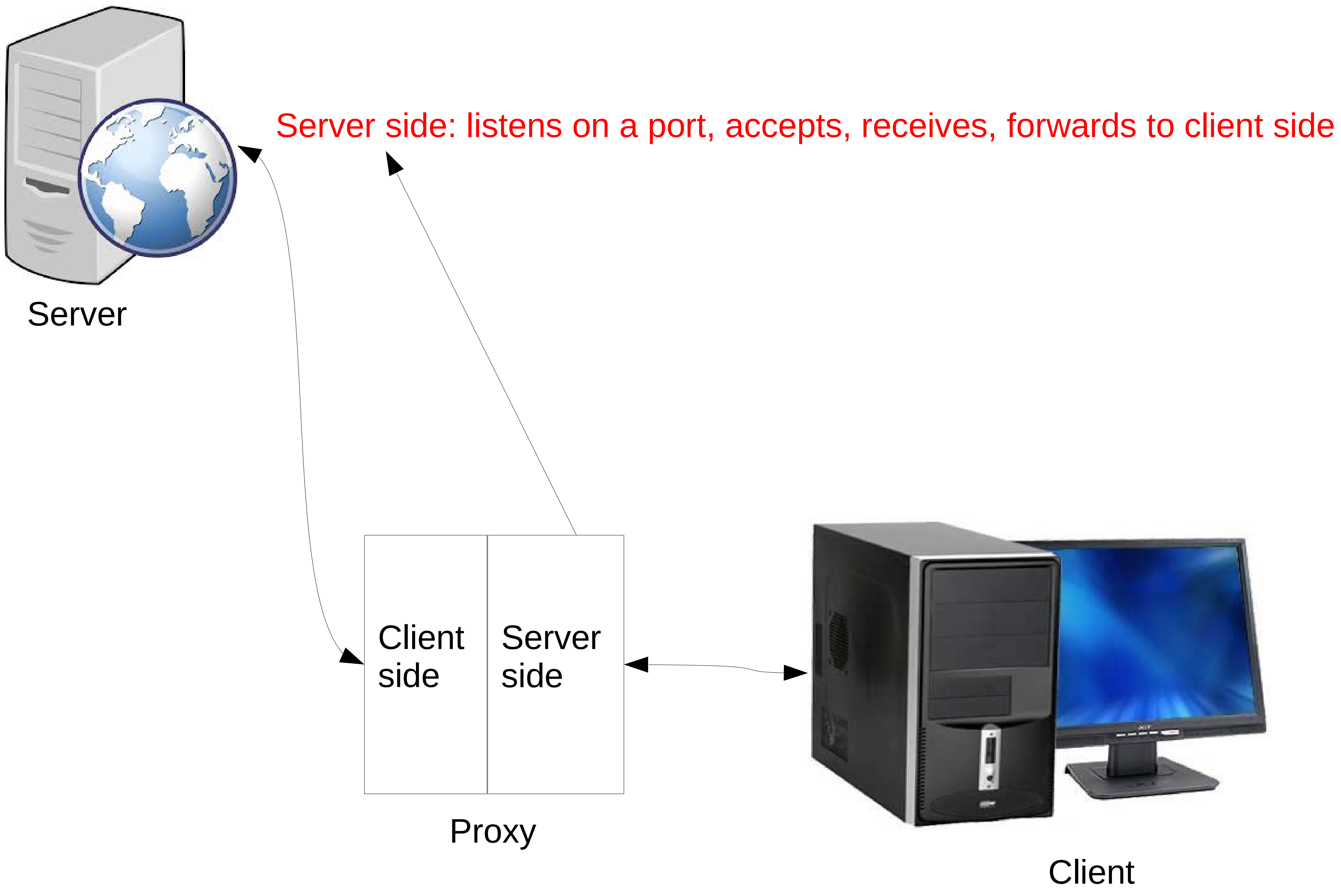
How to handle connections

- 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?

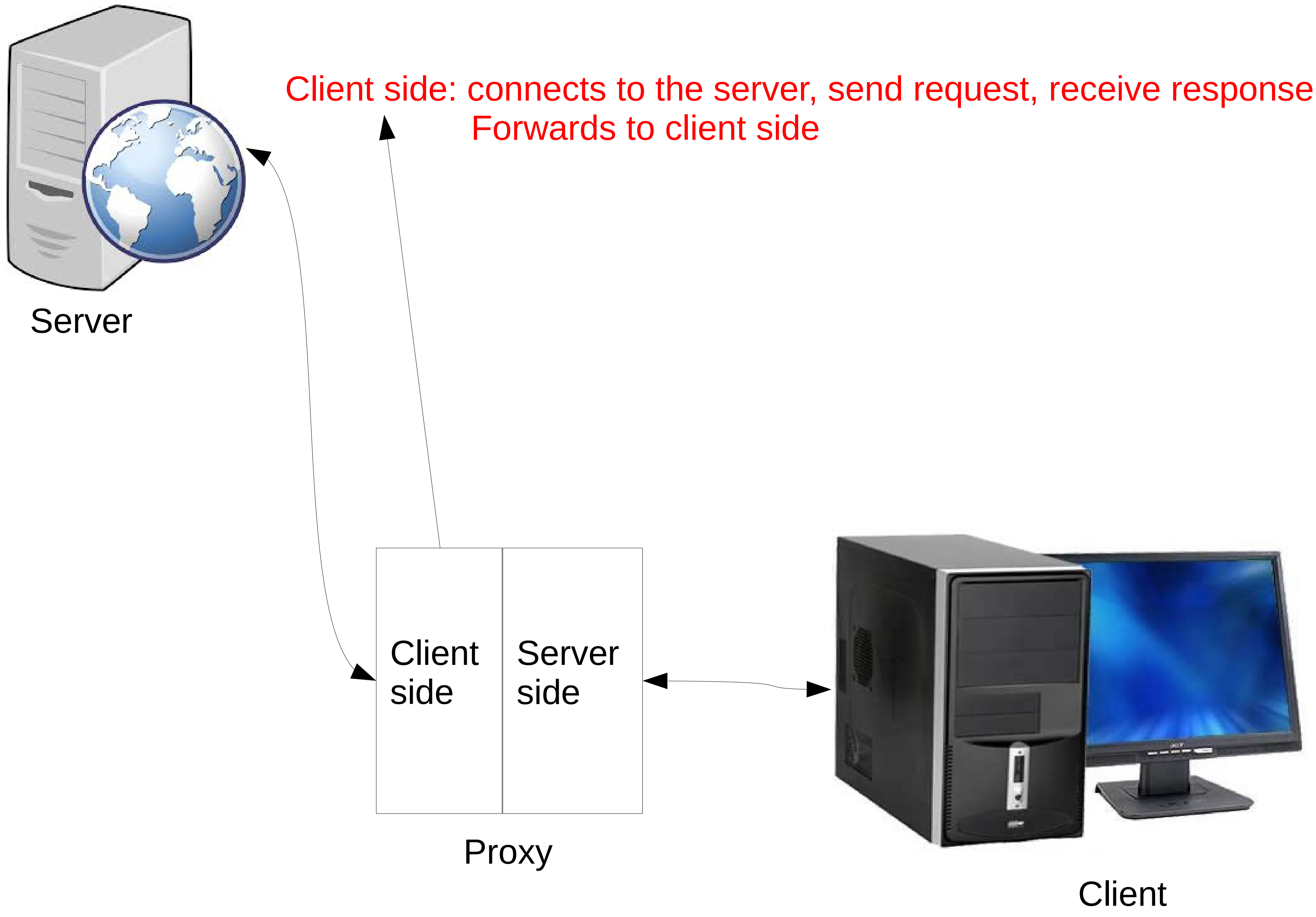
General overlay



General overlay


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

General overlay


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

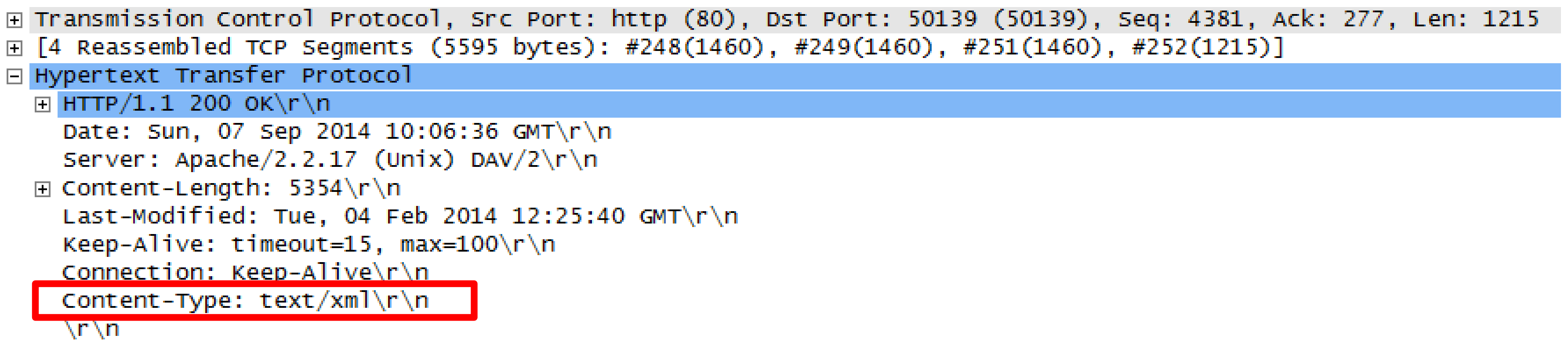
Content filtering

- Response from the server comes in segments
- Remember TCP segmentation?
- Reconstruct the message in a temporary buffer
- Then run filtering on the message

Text vs other binary data

- What is the requirement for filtering w.r.t binary data?
  - Only that you have to be smart in handling any data type
- What will happen if you attempt to reconstruct an image or video and try to filter it?
- Solutions?

Text vs binary data



```
⊞ 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
```

- Content-type header
- Differentiate content type
  - Run/don't run filtering
  - Send data or block the client

How to block specific content

- You are supposed to return a specific response based on URL filtering or content filtering

HTTP redirect

- If filtering confirms presence of inappropriate words

HTTP/1.1 301 Moved Permanently

- Else send response

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

Questions?

Good Luck!!