Computer Systems
Introduction to Networks
Summative Assignment

Jan Minář `<rdancer@rdancer.org>`

November 27, 2010

Chapter 1

Client-Server Architecture

In a client-server architecture, the application is distributed over two or more separate platforms. The servers offer services which are utilized by the clients. One functional unit can act both as a client and a server at the same time (e.g. a web server that is at the same time a client to a database server). Clients and servers communicate over shared network. (Vaughn 1994, pp3–11)

There is a vast number of applications that follow the client-server model (most of the ports assigned by IANA correspond to client-server applications (*Port Numbers* 2009)). Alternatives to the client-server architecture include *application* architecture and *peer-to-peer* architecture. (Kurose & Ross 2008, p110)

There is typically one or a few servers, serving a large number of clients (Kurose & Ross 2008, p110). It is not possible for clients to communicate with each other directly; all communication between clients must be realized through the server (for example, two Mail User Agents can indeed send e-mails to each other, but always via e.g. SMTP and IMAP mail servers).

Maintaining a server can be "infrastructure-intensive", when the server has to withstand very many requests. Often a server farm is deployed, so that the load is shared between multiple machines. (Kurose & Ross 2008, p110)

1.1 Transport Layer View

The client initiates the communication by sending a request to the server. The server only *responds*, and can never *initiate* communication. If the initial request is to succeed, the server process must have had requested the operating system to listen on a given (TCP or UDP) port. The port numbers and their corresponding services are maintained by a central registry (*Port Numbers* 2009), so that for example a POP3 client will be able to connect to a POP3 server just by knowing the server IP address. The port number only has to be specified when it differs from the default. Once the client sends in a request to the correct port and address, request is received by the operating system of the server machine, the operating system passes the request on to the server process. The server process responds to the request, and two-way communication ensues.

1.2 Thick and Thin Clients

A thin client system does most of the data processing on the server side. This allows the client to be relatively low-powered, which can mean lower costs (network terminals used instead of full-blown PCs, with the applications running on an application server), or perhaps longer battery life (mobile phone or a PDA running a video-processing application client, with the actual computationally intensive video re-encoding performed on the server accessed over the mobile network). Early web browsers were thin clients.

A thick client does most of the data processing itself. This approach does not suffer from the limitations of the network, such as latency. For example, high quality video playback is a bandwidth-intensive application, and it is often more practical to transport the video over the network in a compressed form, and have the client do the computationally intensive decompression, thus saving bandwidth. Contemporary web browsers are thick clients.

Chapter 2

Key Features of Virtual Circuit and Datagram Packet Switching Networks

Virtual circuit (VC; also called *virtual call*) networks came from the traditional pre-existing voice telephone network, which was usually a state-wide network, with interstate and overseas links. The word "datagram" itself derives from "telegram" (Russell 1989, p141). On the other hand, *packet switching* was developed in the 1970s as means of efficient transmission of data over long distances. Nowadays, datagram networking takes over traditional mainstays of VC, however, VC is still being used. It is possible to use a mixed approach.

2.1 Virtual Circuit Lifetime

Connection via VC has three phases:

1. set-up
    - the path through the network is determined
    - resources such as bandwidth/time slot are reserved, and everything is set
    - this can take some amount of time, creating a set-up delay
2. data transfer
    - the path does not changed for the duration of the connection
    - resources remain reserved even when not actually needed
    - the latency is negligible, because there are no delays introduced by the management of the link, unlike with datagram networks

3. tear-down
    - the routing table entries are purged
    - bandwidth/slots are released for use for future VCs

2.2 Routing in Datagram Networks

Datagram networks route each packet individually. When a datagram router receives a packet, it needs to decide which next hop it should send it to, i.e. which interface to forward it via. It would be impractical to store this information for every possible destination address, and therefore the routers mostly store only aggregate routes (multiple adjacent addresses represented by a common prefix). Often an address is comprised of a prefix, and a host part. The routes are then decided with respect to the network prefixes, not the individual addresses. A router can maintain two different routes for two network prefixes in such a way that one prefix is contained in the other one. In that case, the longer prefix takes precedence. It is said that the corresponding route is more specific.

As using hard-coded, or static routing would not be feasible for busy routers with many connections, automatic routing protocols have been devised that change routing table typically every few minutes. In contrast to that, in a VC network, the routing table constantly changes with every VC setup/tear-down, many times a second.

2.3 Comparison

Table 2.1 compiled from data in (Kurose & Ross 2008, Russell 1989), (Stallings 2007, p298–299).

2.4 Hybrid Approach

“[V]irtual circuit can be built on top of the datagram service” (Russell 1989, p141). Many currently deployed networks behave like a VC towards the end-hosts, and are really internally working as datagram networks—the advantage of having a comfortable interface presented to the host is married with the flexibility and cost-efficiency of a datagram network.

Virtual Circuit	*Datagram*
complexity is kept within the network, which allows end-systems to be simple, even "dumb" (Kurose & Ross 2008, p349)	designed to connect sophisticated hosts, the network is deliberately "as simple as possible"; complex functionality is implemented by the *transport* and *application* layers (Kurose & Ross 2008, pp349–351)
maintains a dedicated path through the network between the two end-hosts	routes each packet separately, each packet contains routing information header; path can change in-between packets
guarantees delivery, low latency and reserved bandwidth	best-effort, with packet loss/duplicity, jitter, data corruption, shared bandwidth
its all-or-nothing approach means it will simply fail hard and tear down the connection in case of an error	routes around a malfunctioning router or network path and recovers from errors
naturally maps onto traditional voice networks, which it was developed from and for	developed in the 1970s; has not fundamentally changed since, one of few technologies of choice for data transmission over long distances (Stallings 2007, p298–299)
good for voice, video & similar real-time, low-latency applications that require approximately the same amount of bandwidth all the time	good for applications that transmit data in bursts
upfront delay while the virtual circuit is set up; latency negligible afterwards	considerable latency due to intrinsic delays

Table 2.1: Comparison of Virtual Circuit and Datagram Networks

Chapter 3

Hypertext Transfer Protocol

The Hypertext Transfer Protocol is a public domain client-server stateless application layer protocol that communicates over TCP. (Berners-Lee, Fielding & Frystyk 1996, Fielding, Gettys, Mogul, Frystyk, Masinter, Leach & Berners-Lee 1999), (Kurose & Ross 2008, pp122–124)

HTTP is the most often used data transport application protocol on the World Wide Web.

3.1 Function

HTTP retrieves and manipulates objects denoted by Uniform Resource Identifiers (URIs). (Stallings 2007) HTTP methods GET, PUT, DELETE, MOVE, LINK, UNLINK are used for this purpose. HTTP has auxiliary capabilities represented by the methods HEAD OPTIONS, TRACE, CONNECT and others.

HTTP has built-in support for caches. Cache is an intermediate web server that takes the HTTP requests and if possible serves the objects from a local repository, thereby providing speedier response and saving Internet bandwidth.

3.2 Behaviour

HTTP uses several powerful technologies, some of them predating its invention, some of them invented because of HTTP. Underlying character set is human-readable ASCII. Reliable transport is provided by TCP. The protocol leverages the very powerful concept of URIs/URLs (Berners-Lee, Masinter & McCahill 1994) to identify the objects that are to be manipulated. HTTP also uses MIME, and other technologies to negotiate the presentation of the object identified by the URI.(Fielding et al. 1999) HTTP then retrieves the object.

3.3 Design

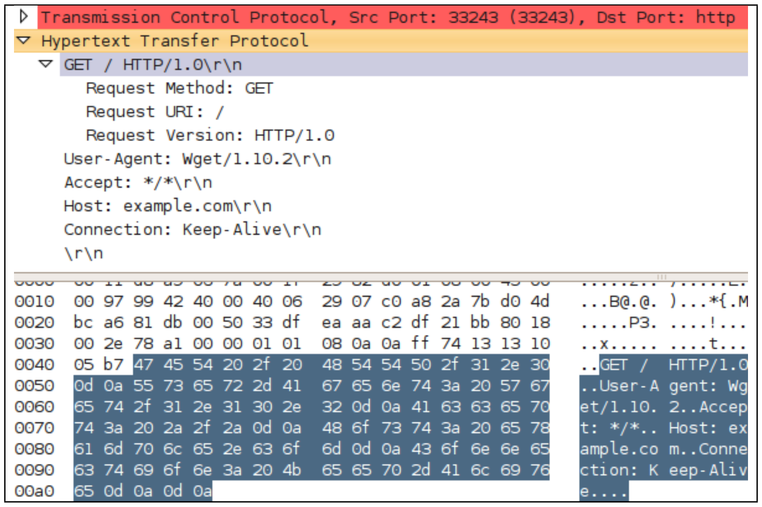
HTTP was designed to transfer simple textual web pages on the World Wide Web efficiently. Because a typical web session has the user retrieving pages in a quick succession from different servers, the protocol was designed to have low overhead, and be stateless. By the time HTTP 1.1 was designed, the Web changed radically, and a need for multi-object web pages was felt, as well as usefulness of tracking the user across visits. Therefore, persistent connections and pipelining was added, and HTTP cookies. Other functionality, such as the support for virtual servers (the Host header) has been added.

3.4 Packet Dissection

We have prepared two HTTP packets out of a TCP stream that was the result of retrieving a single document from a remote WWW server. The upper half of the images shows the interpretation, the lower half then the on-the-wire data in hexadecimal and ASCII representation. We used *Wireshark* 1.0.0 to perform the packet analysis.

Figure 3.1 shows the HTTP request. TCP packet sent to HTTP default port (denoted by *http*; 80), the well-known port number assigned to HTTP. The request line specifies the GET method, path /, the HTTP version used is 1.0. The User Agent (UA) claims to be Wget version 1.10.2, and it accepts any and all MIME types. URI host part is example.com. The UA wishes to use persistent connections. Entity body is empty.

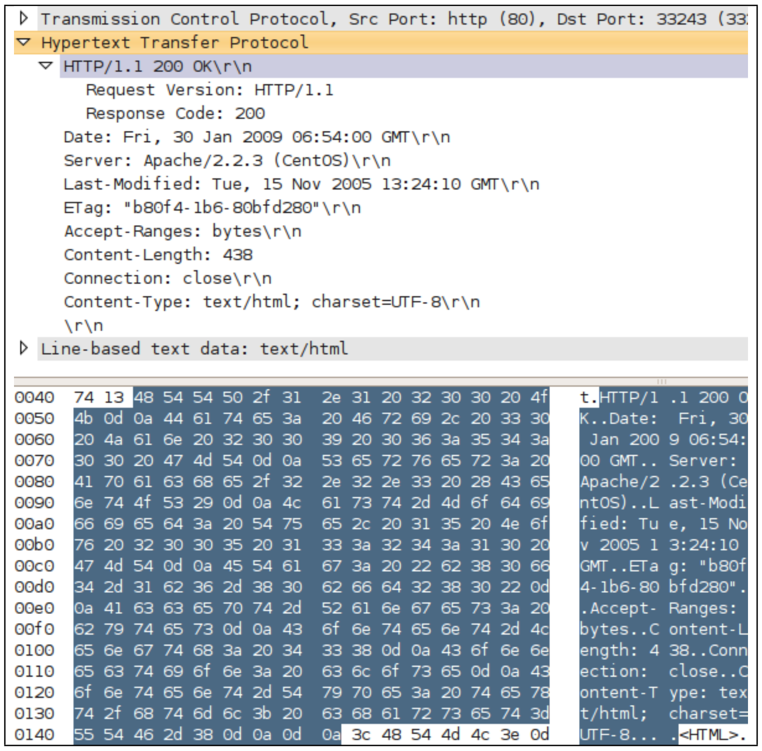
Figure 3.2 shows the HTTP response. The status line shows the server uses HTTP version 1.1, status code 200 means that the request has been successfully processed. The response was generated on 2009-01-30 at 06:54:00 GMT. Server claims to be Apache version 2.2.3 running on CentOS. The requested object has not been modified since 2005-11-15 13:24:10 GMT. The entity tag (ETag) is given. Server accepts byte-range requests. Length of the entity body is 438 octets. Server will close the TCP connection when the entity body will have been transmitted. The MIME type of the object is text/html, character set UTF-8. The entity body contains the requested object (the HTML page).



```
▷ Transmission Control Protocol, Src Port: 33243 (33243), Dst Port: http
▽ Hypertext Transfer Protocol
  ▽ GET / HTTP/1.0\r\n
      Request Method: GET
      Request URI: /
      Request Version: HTTP/1.0
    User-Agent: Wget/1.10.2\r\n
    Accept: */*\r\n
    Host: example.com\r\n
    Connection: Keep-Alive\r\n
    \r\n

0010  00 97 99 42 40 00 40 06  29 07 c0 a8 2a 7b d0 4d   ...B@.@. )...*{.M
0020  bc a6 81 db 00 50 33 df  ea aa c2 df 21 bb 80 18   .....P3. ....!...
0030  00 2e 78 a1 00 00 01 01  08 0a 0a ff 74 13 13 10   ..x..... ....t...
0040  05 b7 47 45 54 20 2f 20  48 54 54 50 2f 31 2e 30   ..GET /  HTTP/1.0
0050  0d 0a 55 73 65 72 2d 41  67 65 6e 74 3a 20 57 67   ..User-A gent: Wg
0060  65 74 2f 31 2e 31 30 2e  32 0d 0a 41 63 63 65 70   et/1.10. 2..Accep
0070  74 3a 20 2a 2f 2a 0d 0a  48 6f 73 74 3a 20 65 78   t: */*.. Host: ex
0080  61 6d 70 6c 65 2e 63 6f  6d 0d 0a 43 6f 6e 6e 65   ample.co m..Conne
0090  63 74 69 6f 6e 3a 20 4b  65 65 70 2d 41 6c 69 76   ction: K eep-Aliv
00a0  65 0d 0a 0d 0a                                     e....
```

Figure 3.1: HTTP request



```
▷ Transmission Control Protocol, Src Port: http (80), Dst Port: 33243 (33
▽ Hypertext Transfer Protocol
  ▽ HTTP/1.1 200 OK\r\n
      Request Version: HTTP/1.1
      Response Code: 200
    Date: Fri, 30 Jan 2009 06:54:00 GMT\r\n
    Server: Apache/2.2.3 (CentOS)\r\n
    Last-Modified: Tue, 15 Nov 2005 13:24:10 GMT\r\n
    ETag: "b80f4-1b6-80bfd280"\r\n
    Accept-Ranges: bytes\r\n
    Content-Length: 438
    Connection: close\r\n
    Content-Type: text/html; charset=UTF-8\r\n
    \r\n
▷ Line-based text data: text/html

0040  74 13 48 54 54 50 2f 31  2e 31 20 32 30 30 20 4f   t.HTTP/1 .1 200 O
0050  4b 0d 0a 44 61 74 65 3a  20 46 72 69 2c 20 33 30   K..Date:  Fri, 30
0060  20 4a 61 6e 20 32 30 30  39 20 30 36 3a 35 34 3a    Jan 200 9 06:54:
0070  30 30 20 47 4d 54 0d 0a  53 65 72 76 65 72 3a 20   00 GMT.. Server: 
0080  41 70 61 63 68 65 2f 32  2e 32 2e 33 20 28 43 65   Apache/2 .2.3 (Ce
0090  6e 74 4f 53 29 0d 0a 4c  61 73 74 2d 4d 6f 64 69   ntOS)..L ast-Modi
00a0  66 69 65 64 3a 20 54 75  65 2c 20 31 35 20 4e 6f   fied: Tu e, 15 No
00b0  76 20 32 30 30 35 20 31  33 3a 32 34 3a 31 30 20   v 2005 1 3:24:10 
00c0  47 4d 54 0d 0a 45 54 61  67 3a 20 22 62 38 30 66   GMT..ETa g: "b80f
00d0  34 2d 31 62 36 2d 38 30  62 66 64 32 38 30 22 0d   4-1b6-80 bfd280".
00e0  0a 41 63 63 65 70 74 2d  52 61 6e 67 65 73 3a 20   .Accept- Ranges: 
00f0  62 79 74 65 73 0d 0a 43  6f 6e 74 65 6e 74 2d 4c   bytes..C ontent-L
0100  65 6e 67 74 68 3a 20 34  33 38 0d 0a 43 6f 6e 6e   ength: 4 38..Conn
0110  65 63 74 69 6f 6e 3a 20  63 6c 6f 73 65 0d 0a 43   ection:  close..C
0120  6f 6e 74 65 6e 74 2d 54  79 70 65 3a 20 74 65 78   ontent-T ype: tex
0130  74 2f 68 74 6d 6c 3b 20  63 68 61 72 73 65 74 3d   t/html;  charset=
0140  55 54 46 2d 38 0d 0a 0d  0a 3c 48 54 4d 4c 3e 0d   UTF-8... .<HTML>.
```

Figure 3.2: HTTP response

Chapter 4

Characteristics of Connection-Oriented and Connectionless Transport Layer Services

The two most prolific Internet Protocol Suite transport layer protocols are UDP and TCP. We will show the characteristics of connection-oriented and connectionless services using UDP and TCP as examples and describing briefly how they work.

4.1 Connectionless Service—User Datagram Protocol

UDP is defined in (Postel 1980). It is the datagram transport in the Internet Protocol Suite. It is a very thin layer on top of IP, as it only provides a checksum and port number (enabling multiplexing, i.e. having more than one connection from one host to another via UDP). (Cf. Fig. 4.1) (Socolofsky & Kale 1991)

The connectionless service only provides the bare minimum necessary for a transport layer to provide. This can be a positive feature, because it means low overhead and low implementation complexity.

UDP provides the application layer with the ability to send and receive short fixed-length datagrams. Each datagram is a separate transmission; there is no state kept in-between. If desired, the application can implement some of the missing features in a higher layer.

```
0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+---------------- ...
```

Figure 4.1: User Datagram Protocol Header (Postel 1980, p1)

4.2 Connection-Oriented Service—Transport Control Protocol

TCP (Postel 1981) is the most commonly used transport protocol in the Internet Protocol Suite. In order to provide a connection-oriented service, the protocol is necessarily more complex than its connectionless counterpart. The complexity comes at cost in bandwidth and processing overhead.

TCP provides in-order reliable delivery (with automatic retransmission of lost packets) and automatic congestion sensing and adaptation. Like UDP, TCP provides corruption checking using checksum, and multiplexing using ports. TCP provides the application layer with an abstraction of a transparent end-to-end duplex virtual circuit (byte stream). (Socolofsky & Kale 1991)

Bibliography

Berners-Lee, T., Fielding, R. & Frystyk, H. (1996), 'Hypertext Transfer Protocol – HTTP/1.0', RFC 1945 (Informational).
URL: *http://www.ietf.org/rfc/rfc1945.txt*

Berners-Lee, T., Masinter, L. & McCahill, M. (1994), 'Uniform Resource Locators (URL)', RFC 1738 (Proposed Standard). Obsoleted by RFCs 4248, 4266, updated by RFCs 1808, 2368, 2396, 3986.
URL: *http://www.ietf.org/rfc/rfc1738.txt*

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T. (1999), 'Hypertext Transfer Protocol – HTTP/1.1', RFC 2616 (Draft Standard). Updated by RFC 2817.
URL: *http://www.ietf.org/rfc/rfc2616.txt*

Kurose, J. F. & Ross, K. W. (2008), *Computer Networking — A Top-Down Approach*, fourth edn, Pearson Education, Inc., Boston, Massachusetts. International Edition.

Port Numbers (2009). Internet Assigned Numbers Authority.
URL: *http://www.iana.org/assignments/port-numbers*

Postel, J. (1980), 'User Datagram Protocol', RFC 768 (Standard).
URL: *http://www.ietf.org/rfc/rfc768.txt*

Postel, J. (1981), 'Transmission Control Protocol', RFC 793 (Standard). Updated by RFCs 1122, 3168.
URL: *http://www.ietf.org/rfc/rfc793.txt*

Russell, D. (1989), *The Principles of Computer Networking*, Cambridge University Press, Cambridge, United Kingdom.

Socolofsky, T. & Kale, C. (1991), 'TCP/IP tutorial', RFC 1180 (Informational).
URL: *http://www.ietf.org/rfc/rfc1180.txt*

Stallings, W. (2007), *Data and Computer Communications*, eighth edn, Pearson Education, Inc., Upper Saddle River, New Jersey.

Vaughn, L. T. (1994), *Client/Server System Design and Implementation*, McGraw-Hill, Inc.