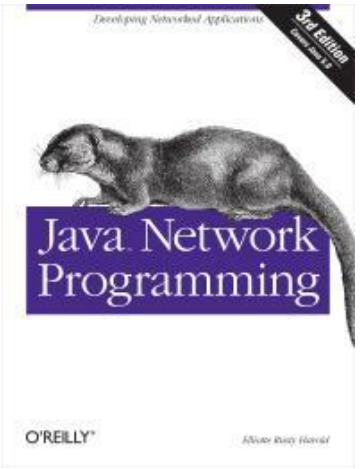
- DA343A -
Objektorienterad programutveckling, trådar och datakommunikation

Föreläsning 10
Network Communication

Ben Blamey
Based on previous slides by: Gion Koch Svedberg

Föreläsning 10 (Datakom 7)

- Java and the Internet
  - General construction of the Internet as a network of networks
  - Java's view of the Internet
    - InetAddress
    - TCP/IP Model with 4 layers
    - HTTP as an example of an application protocol
    - URL
    - URLConnection
- Communicating via the Internet network
  - HTTP-server, ”request and response”

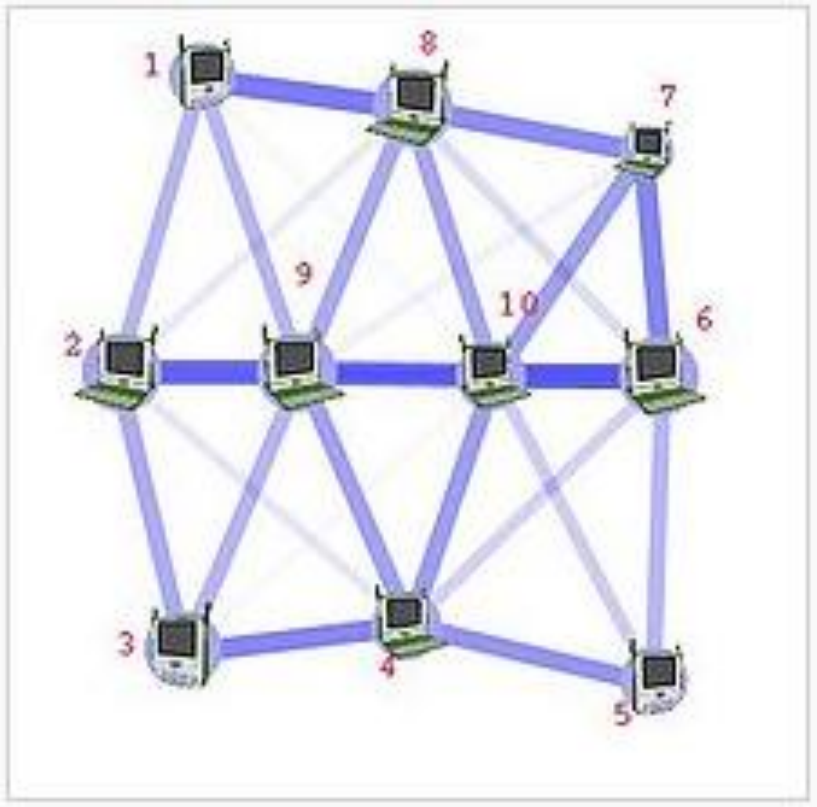


eBook:
Harold, Elliotte Rusty (2008),
Java Network Programming,
Sebastopol:O’Reilly Media

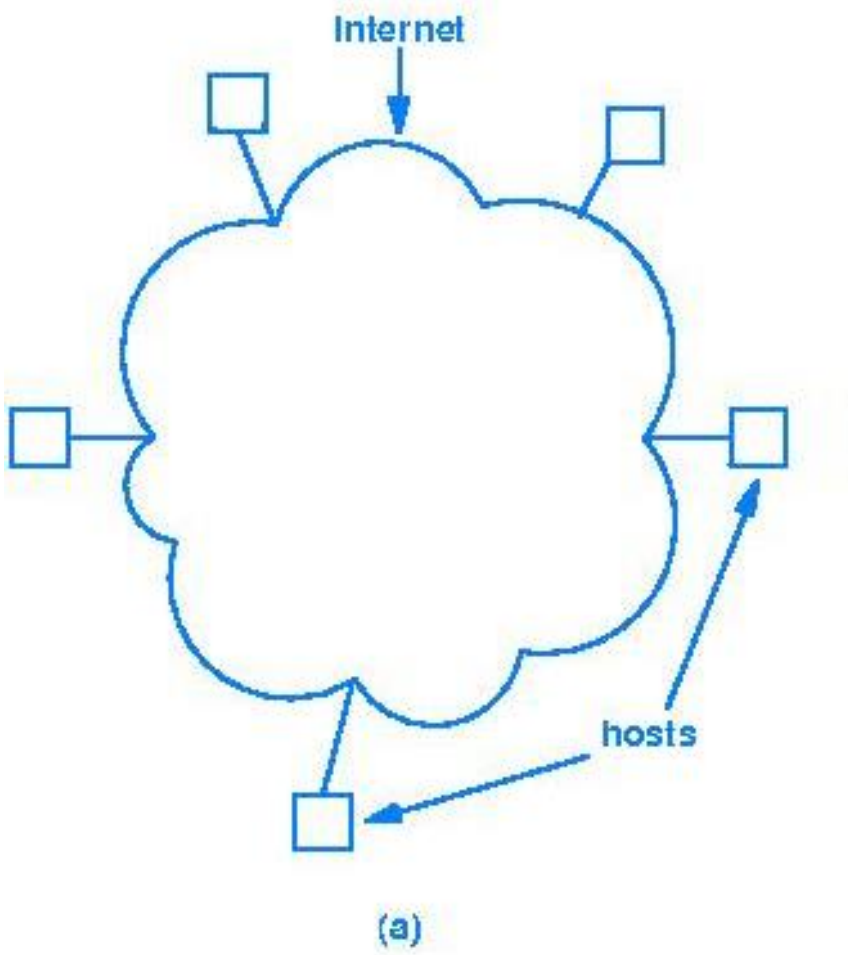
https://proxy.mau.se/login?url=https://ebookcentral.proquest.com/lib/malmo/detail.action?docID=540711

Network

- A network is a number of computers and other devices that can send and receive data from each other.
- A protocol defines how computers communicate with each other, e.g.
  how addressing is done and how data is packaged.
- Each computer on the network is a node, and each node has a unique address.
- Data is sent over the network as packets. A packet contains the address of the recipient and the sender.
- The path a packet takes from sender to receiver depends, among other things, on load in different parts of the network.

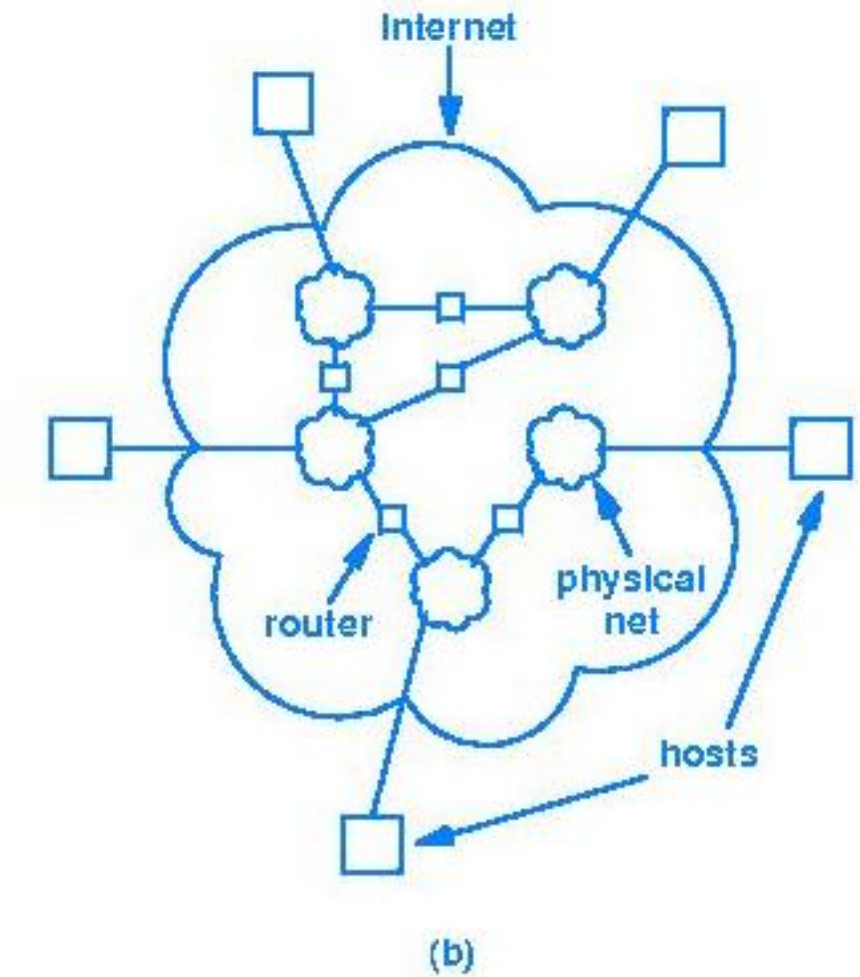


The Internet as a Network of Networks



(a)

User's view of the Internet: Each computer appears to be connected to the same, large network.



(b)

The structure of physical networks with routers that build up a network of different networks.

Complexity!

- **Security** – How can we be sure we know who eachother are? Can we encrypt our data to keep it secret? Can we be sure it has not been modified by someone else?
- **Application Protocols** – what protocol should we use to communicate with different applications – web servers, email servers, print servers, file sharing, …?
- **Addressing** – how do we know where to send data? How do we represent addresses?
- **Ordering** – Do we need to split large messages into smaller pieces? if we have many packets of data, how do we put them back in the correct order at the other end? Do we know it was received?
- **Congestion** – How do we deal with network congestion? Can we ask data senders to slow down?
- **Corruption & Lost Data** – how do we ensure our data has not been corrupted en route? Should it be re-sent?
- **Routing** – how should it get there? (perhaps around the world!) What route should it take? How do send data back again?
- **Which kind of network(s) do we use on the way?** Wifi? Ethernet? …?
- **How do we send data on that network?** e.g. Ethernet -> what voltages for 1 or 0? How fast?

Solution… Encapsulation! (Lecture 5)

Solution… Encapsulation in multiple layers!

Layers of TCP/IP 'Stack'

Lästips
Java Network Programming
Chapter 2 ("Basic Network Concepts")

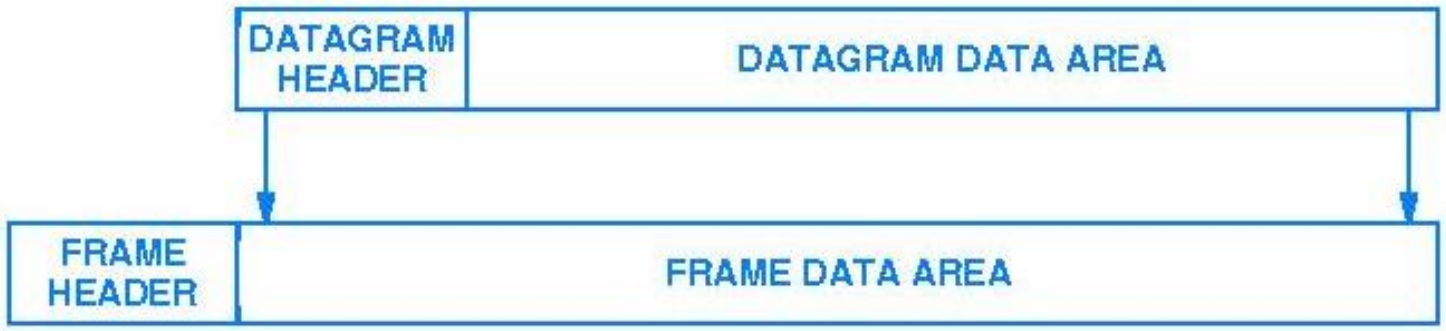
TCP/IP Stack

Layer	Unit of Data	Protocols	
Application Layer	Message / Data	HTTP SMTP FTP DNS (proprietary)	→ Web Request + Response → Transfer of Emails → Transfer of Files → Domain Name System → (application specific protocols)
Transport Layer	Segment Datagram	TCP UDP	→ Connection-orientated, guaranted delivery, congestion, ordering. → Connectionless, no control.
Nework Layer (IP Layer)	Datagram / Packet	IP Protocol, Routing	Routing
Data-Link Layer	Frame	Ethernet, WiFi (802.11), Point-to-Point Protocol (PPP)	Binary data sent between nodes.
Physical Layer	Bits	e.g. Ethernet. Wifi: 802.11	Ethernet: Which color wires used for what? Voltages, frequency. WiFi: Format of radio messages? What radio frequency?

- Security and addressing is a concern of all the layers.

Why The Internet Is So Flexible And Adaptable

1. Encapsulation –
    1. Data for each protocol is pre-pended (at the start) as a header when sending.
    2. …and then removed when receiving
    3. Each layer leaves the rest of the data (for the other layers) unchanged
    4. Different names are used for the unit of data in different ”layers”



M: Application layer message
Ht: transport-layer header information
Hn: network-layer header information
Hl: link-layer header information

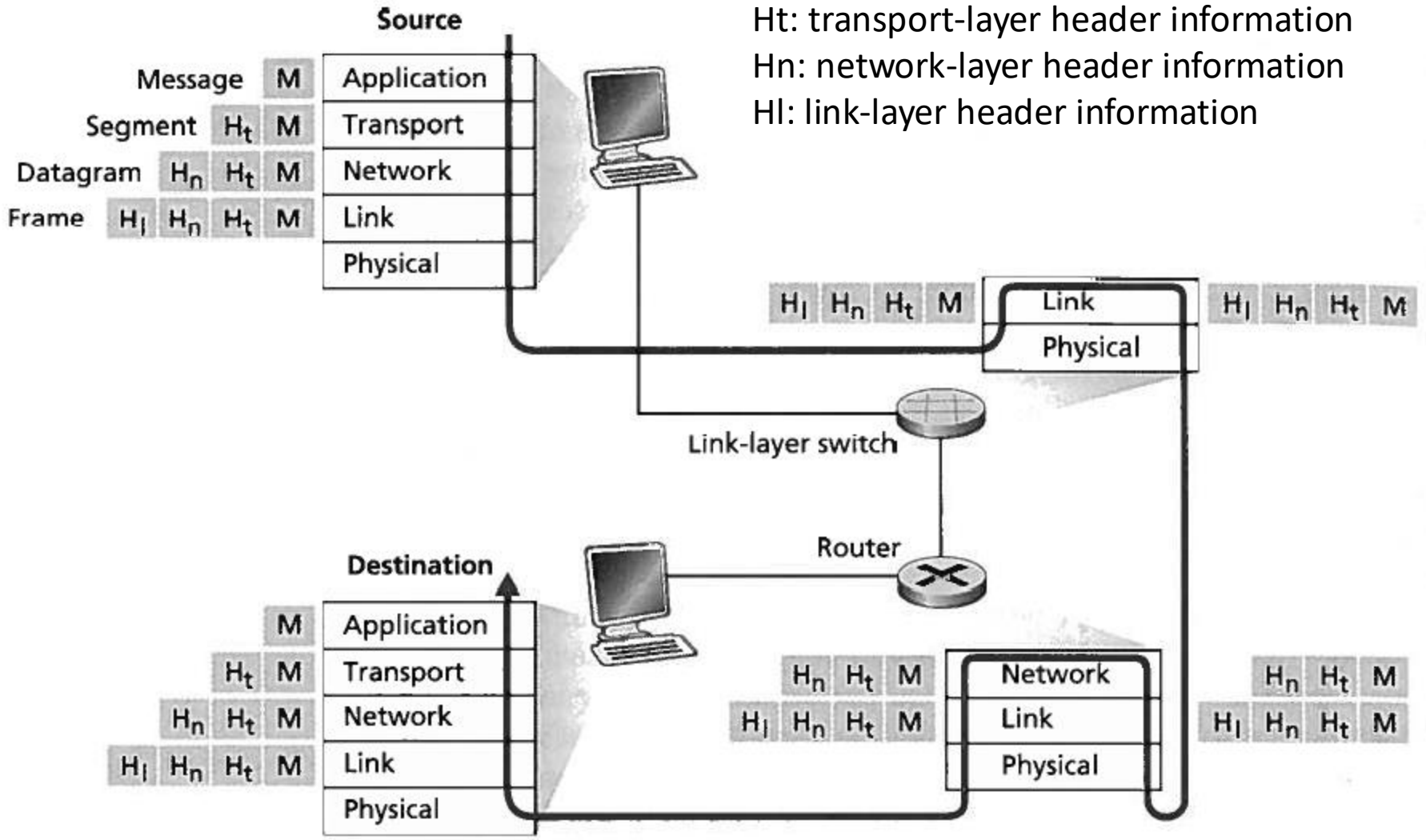


Figure 1.20 ♦ Hosts, routers, and link-layer switches; each contains a different set of layers, reflecting their differences in functionality.

Why The Internet is so Flexible And Adaptable

2. Different forms of addressing in different layers:

TCP/IP Stack

Layer	Unit of Data	Protocols	Concept of Address
Application Layer	Message		**e.g. DNS names (mau.se), URLs (https://mau.se/bibliotek) (also IP address)**
Transport Layer	Segment	TCP UDP	**IP Address (e.g. 188.144.120.34)**
Nework Layer (IP Layer)	Datagram	IP Protocol, Routing	
Data-Link Layer	Frame	Ethernet, WiFi (802.11), Point-to-Point Protocol (PPP)	**MAC "Physical" Addresses (e.g. 08:36:71:ec:41:1d)**
Physical Layer	Bits	e.g. Ethernet. Wifi: 802.11	--

Example: HTTP (HyperText Transfer Protocol)

An example of a *Application Layer* protocol used with TCP/IP

Layers of TCP/IP ’Stack’

Java

Layer	Unit of Data	Protocols	
Application Layer	Message	HTTP SMTP FTP DNS (proprietary)	→ Web Request + Response → Transfer of Emails → Transfer of Files → Domain Name System → (application specific protocols)
Transport Layer	Segment Datagram	TCP UDP	→ Connection-orientated, guaranted delivery, congestion, ordering. → Connectionless, no control.
Nework Layer (IP Layer)	Datagram/ Packet	IP Protocol, Routing	Routin
Data-Link Layer	Frame	Ethernet, WiFi (802.11), Point-to-Point Protocol (PPP)	Binary data sent between nodes.
Physical Layer	Bits	e.g. Ethernet. Wifi: 802.11	Ethernet: Which wires used for what? Voltages, frequency. WiFi: Format of radio messages? What radio frequency?

InetAddress

Reading:
Java Network Programming
Chapter 6 (”Looking Up Internet Addresses”)

- The InetAddress class encapsulates the address when communicating. Each node in the network is identified by a unique address.
- The Current Object is an instance of either the Inet4Address or Inet6Address subclasses depending on the network the address will be used in. The object is immutable – its contents cannot be changed.
- https://docs.oracle.com/javase/8/docs/api/java/net/InetAddress.html
- Example: Ex1

- IPv4
  - The address consists of 4 bytes, written with a period in between, e.g. 92.122.190.114
  - 4 bytes means about 4 billion addresses, which does not cover today's needs.
- IPv6
  - The address consists of 16 bytes printed in pairs of hexadeximal with colons in between, e.g. 34E2:AFFE:1544:F553:A44F:100D:5C7A:BE72
  - Around 45 percent of access to Google is via IPv6 (2025), see: https://www.google.com/intl/en/ipv6/statistics.html

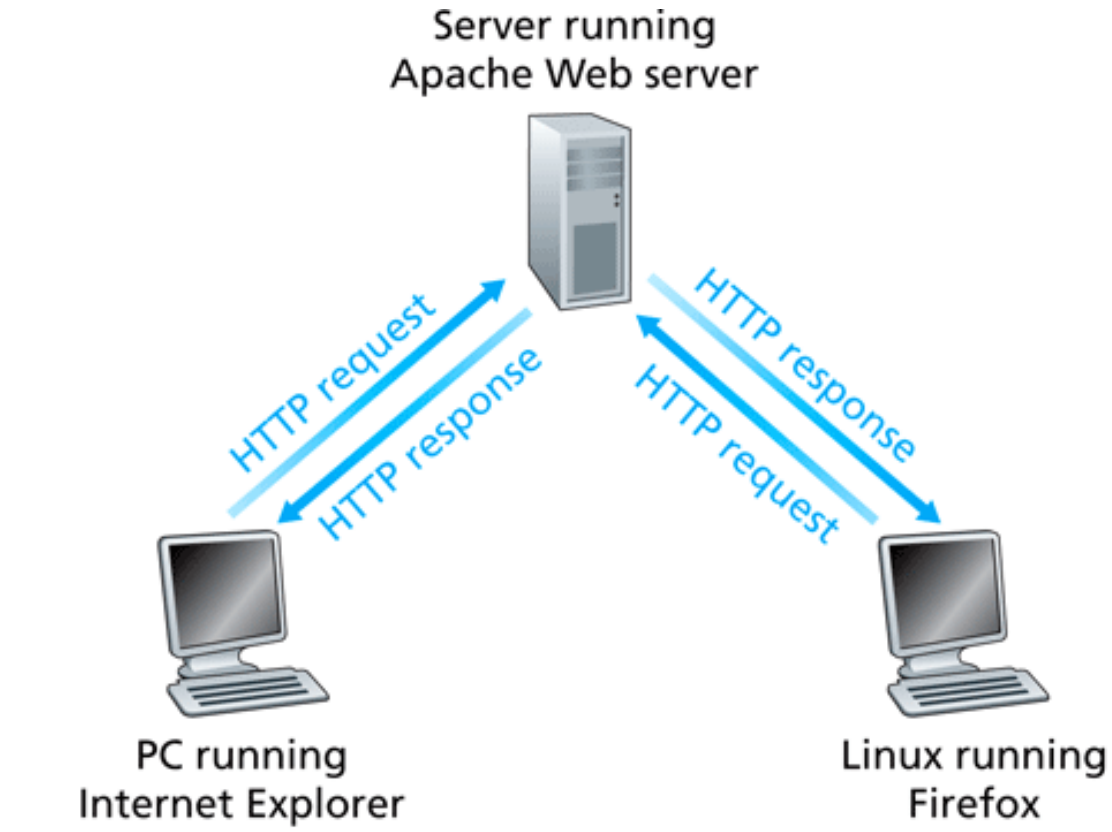
HTTP: example of application layer protocol

Layer	Unit of Data	Protocols	
Application Layer	Message	**HTTP** SMTP FTP DNS (proprietary)	→ **Web Request + Response** → Transfer of Emails → Transfer of Files → Domain Name System → (application specific protocols)

- The computer (client) connects to a web server to retrieve data, such as::
  - Web servers and browsers for the 'world wide web (www)', in the form of hypertext html pages.
  - REST API. Web servers or web services that provide access to data from a resource, such as:
    - Surveillance cameras, Canvas, sensor networks, data storage in the clouds (e.g. activity watches), Google maps, Google street view, Twitter, ...
    - There are thousands of APIs to web services with different data available through HTTP(s) calls. An excellent introduction to the YouTube REST API can be found here: https://www.youtube.com/watch?v=7YcW25PHnAA The problem is that the video is from 2014 and many of the displayed APIs have since introduced authentication requirements.

HTTP: example of application layer protocol

- HTTP is the most common protocol, especially for web servers
  - In turn, it uses secure, connection-based communication with TCP
  - The client must send the requests for information first ("HTTP request") to which the server reacts and sends back the response with the data ("HTTP response")
  - The server does not save previous requests from the client (="stateless")
  - Bi-directional communication between server and client, with initial information exchange of details, such as character sets to be used for file transfer.



HTTP request-response behavior

The format of "request" and "response" messages

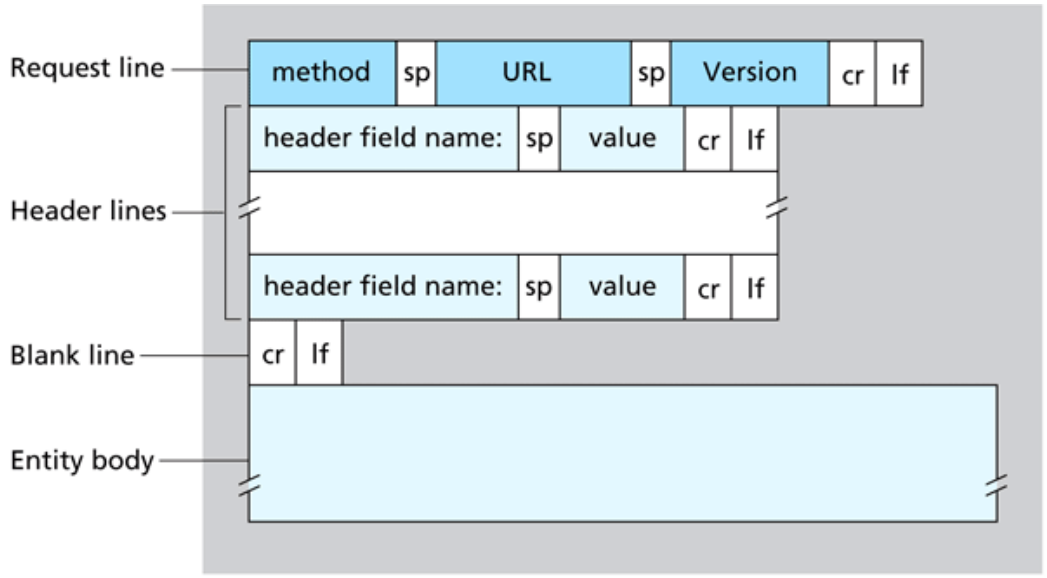


Figure 2.8 ♦ General format of an HTTP request message

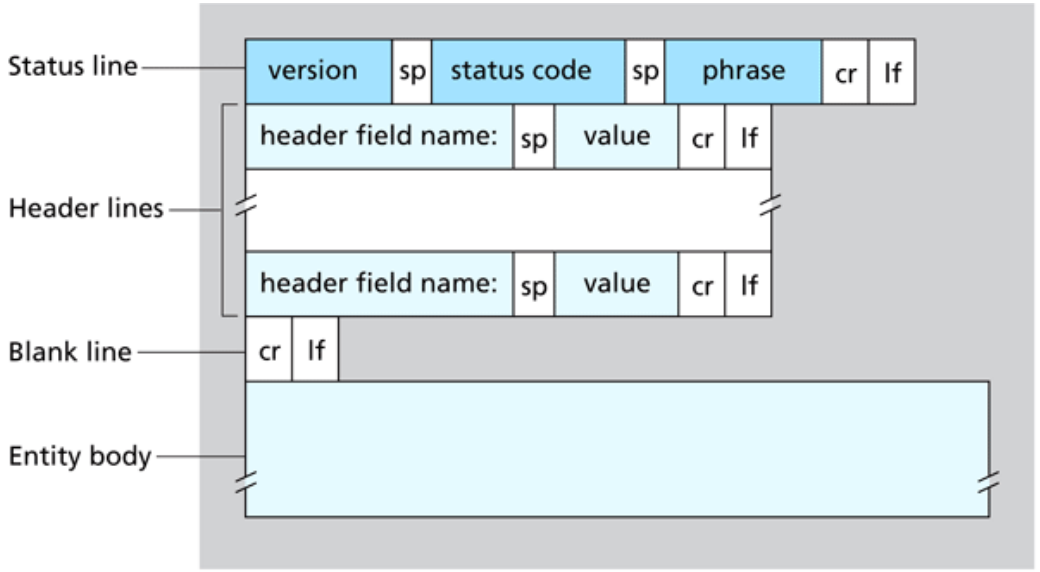


Figure 2.9 ♦ General format of an HTTP response message

The format of "request" and "response" messages

http:// *hostname* [: *port*] */path* [;*parameters*] [*? query*]

1	**GET** The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
2	**HEAD** Same as GET, but it transfers the status line and the header section only.
3	**POST** A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
4	**PUT** Replaces all the current representations of the target resource with the uploaded content.
5	**DELETE** Removes all the current representations of the target resource given by URI.
6	**CONNECT** Establishes a tunnel to the server identified by a given URI.
7	**OPTIONS** Describe the communication options for the target resource.
8	**TRACE** Performs a message loop back test along with the path to the target resource.

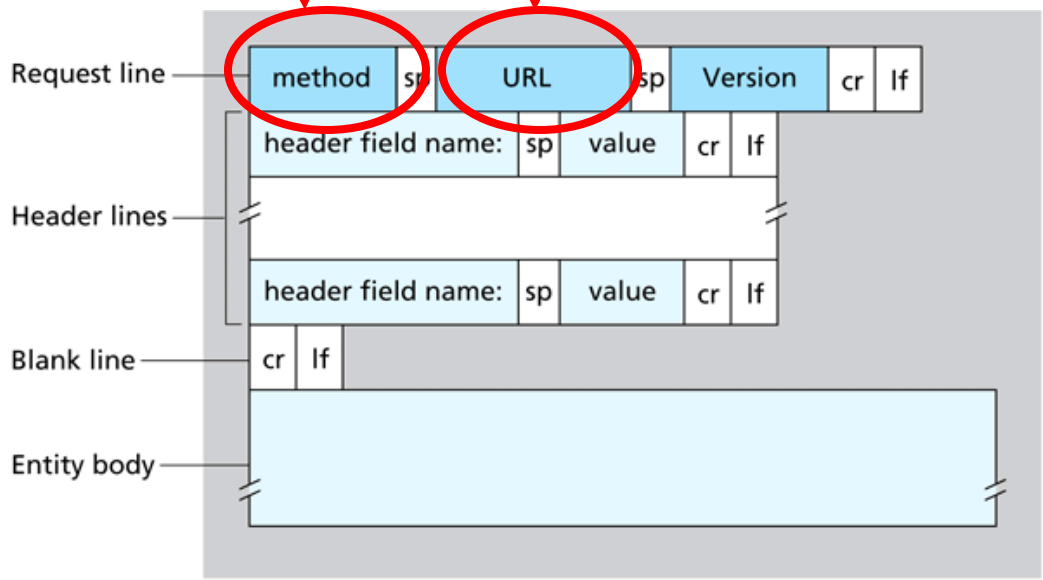
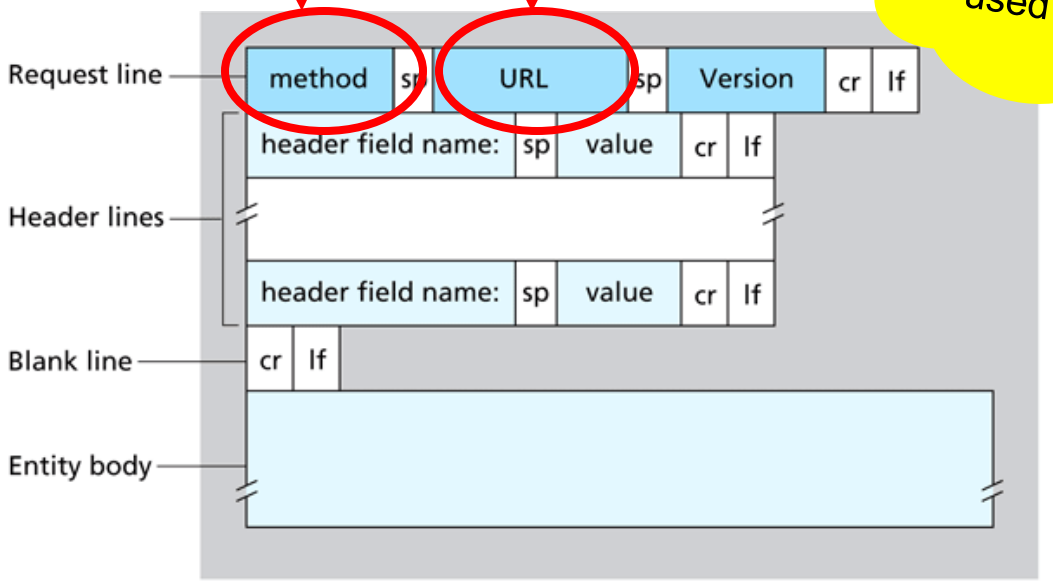


Figure 2.8 ♦ General format of an HTTP request message

The format of "request" and "response" messages

1	**GET** The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
2	**HEAD** Same as GET, but it transfers the status line and the header section only.
3	**POST** A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
4	**PUT** Replaces all the current representations of the target resource with the uploaded content.
5	**DELETE** Removes all the current representations of the target resource given by URI.
6	**CONNECT** Establishes a tunnel to the server identified by a given URI.
7	**OPTIONS** Describe the communication options for the target resource.
8	**TRACE** Performs a message loop back test along with the path to the target resource.

http:// *hostname* [: *port*] */path* [;*parameters*] [*? query*]



How does this compare to the application protocol used in Assignment 2?

Figure 2.8 ♦ General format of an HTTP request message

URL class in Java

Lästips
Java Network Programming
Chapter 7 (”URLs and URIs”)
Chapter 15 (”URLConnections”)

- The URL class is the easiest way for a Java application to find and retrieve data from the web.
- The programmer doesn't have to worry about the details of the protocol, formatting, or communication:
  - The "URL" and "URLConnections" classes use communication connections (TCP) with the current "InputStream" that are opened using the openStream() method.
  - Because the URL starts with http:// the HTTP application layer protocol is used, with a GET request.
  - The response can be found in "InputStream". The class manages sends the request, and waits for the response.
  - It blocks the current thread while we wait for the response.
- It is also possible to write to the server with the output stream that you get with getOutputStream().

- Examples: ex2 Ex3 Ex4

WireShark Example – Request (client to server)

```
Frame 71: 693 bytes on wire (5544 bits), 693 bytes captured (5544 bits) on interface en0, id 0
Ethernet II, Src: Apple_cb:04:98 (6c:7e:67:cb:04:98), Dst: CASwell_ec:41:1c (08:35:71:ec:41:1c)
Internet Protocol Version 4, Src: 10.3.5.244, Dst: 188.184.100.182
Transmission Control Protocol, Src Port: 51361, Dst Port: 80, Seq: 1, Ack: 1, Len: 627
Hypertext Transfer Protocol
  GET /hypertext/WWW/TheProject.html HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET /hypertext/WWW/TheProject.html HTTP/1.1\r\n]
    Request Method: GET
    Request URI: /hypertext/WWW/TheProject.html
    Request Version: HTTP/1.1
  Host: info.cern.ch\r\n
  Connection: keep-alive\r\n
  Cache-Control: max-age=0\r\n
  Upgrade-Insecure-Requests: 1\r\n
  DNT: 1\r\n
  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatio
  Accept-Encoding: gzip, deflate\r\n
  Accept-Language: sv-SE,sv;q=0.9,en-GB;q=0.8,en-SE;q=0.7,en;q=0.6,en-US;q=0.5\r\n
  If-None-Match: "8a9-291e721905000"\r\n
  If-Modified-Since: Thu, 03 Dec 1992 08:37:20 GMT\r\n
  \r\n
  [Full request URI: http://info.cern.ch/hypertext/WWW/TheProject.html]
  [HTTP request 1/1]
  [Response in frame: 73]
```

WireShark Example – Response (server to client)

```
> Frame 145120: 1130 bytes on wire (9040 bits), 1130 bytes captured (9040 bits) on interface en0, id 0
> Ethernet II, Src: CASwell_ec:41:1c (08:35:71:ec:41:1c), Dst: Apple_cb:04:98 (6c:7e:67:cb:04:98)
> Internet Protocol Version 4, Src: 188.184.100.182, Dst: 10.3.5.244
> Transmission Control Protocol, Src Port: 80, Dst Port: 52735, Seq: 1387, Ack: 559, Len: 1064
> [2 Reassembled TCP Segments (2450 bytes): #145119(1386), #145120(1064)]
v Hypertext Transfer Protocol
  v HTTP/1.1 200 OK\r\n
    > [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      Response Version: HTTP/1.1
      Status Code: 200
      [Status Code Description: OK]
      Response Phrase: OK
    Date: Mon, 12 Feb 2024 10:57:54 GMT\r\n
    Server: Apache\r\n
    Last-Modified: Thu, 03 Dec 1992 08:37:20 GMT\r\n
    ETag: "8a9-291e721905000"\r\n
    Accept-Ranges: bytes\r\n
  > Content-Length: 2217\r\n
    Connection: close\r\n
    Content-Type: text/html\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.034997000 seconds]
    [Request in frame: 145116]
    [Request URI: http://info.cern.ch/hypertext/WWW/TheProject.html]
    File Data: 2217 bytes
v Line-based text data: text/html (73 lines)
    <HEADER>\n
    <TITLE>The World Wide Web project</TITLE>\n
    <NEXTID N="55">\n
    </HEADER>\n
    <BODY>\n
    <H1>World Wide Web</H1>The WorldWideWeb (W3) is a wide-area<A\n
    NAME=0 HREF="WhatIs.html">\n
    hypermedia</A> information retrieval\n
    initiative aiming to give universal\n
    access to a large universe of documents.<P>\n
    Everything there is online about\n
```

Questions?