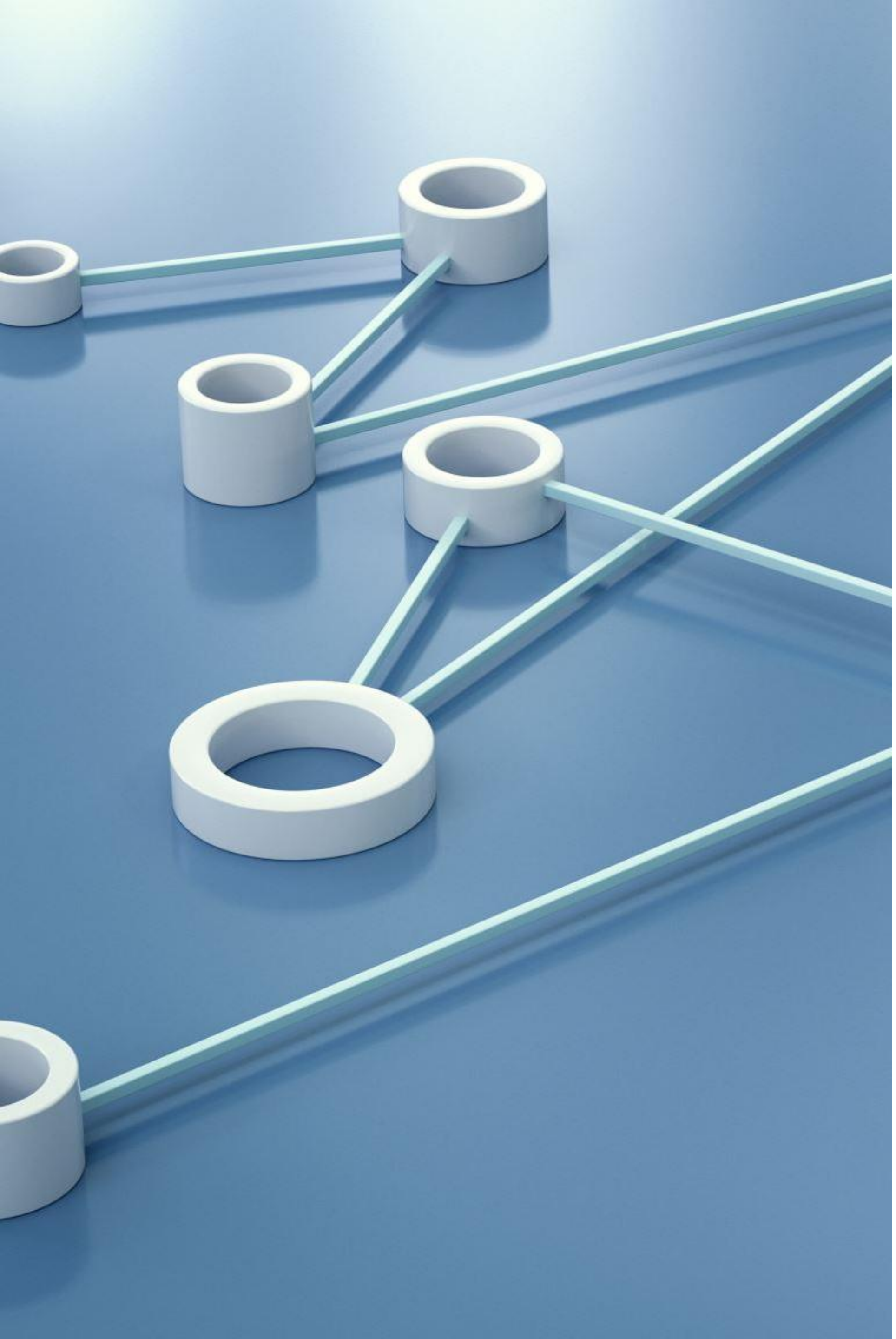
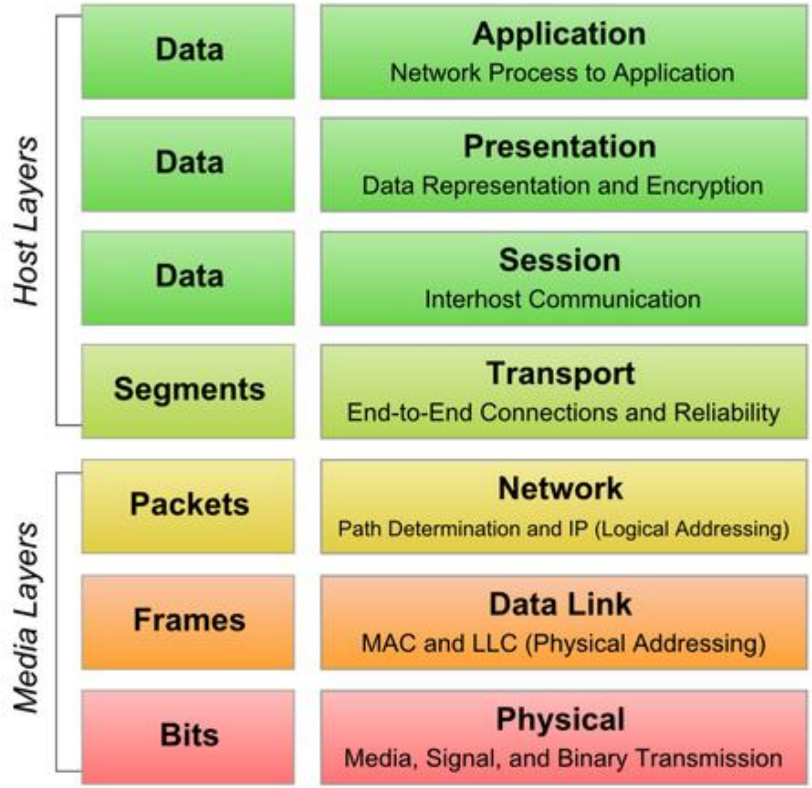
Complex IT-Systems Section 2

Henrik Bulskov

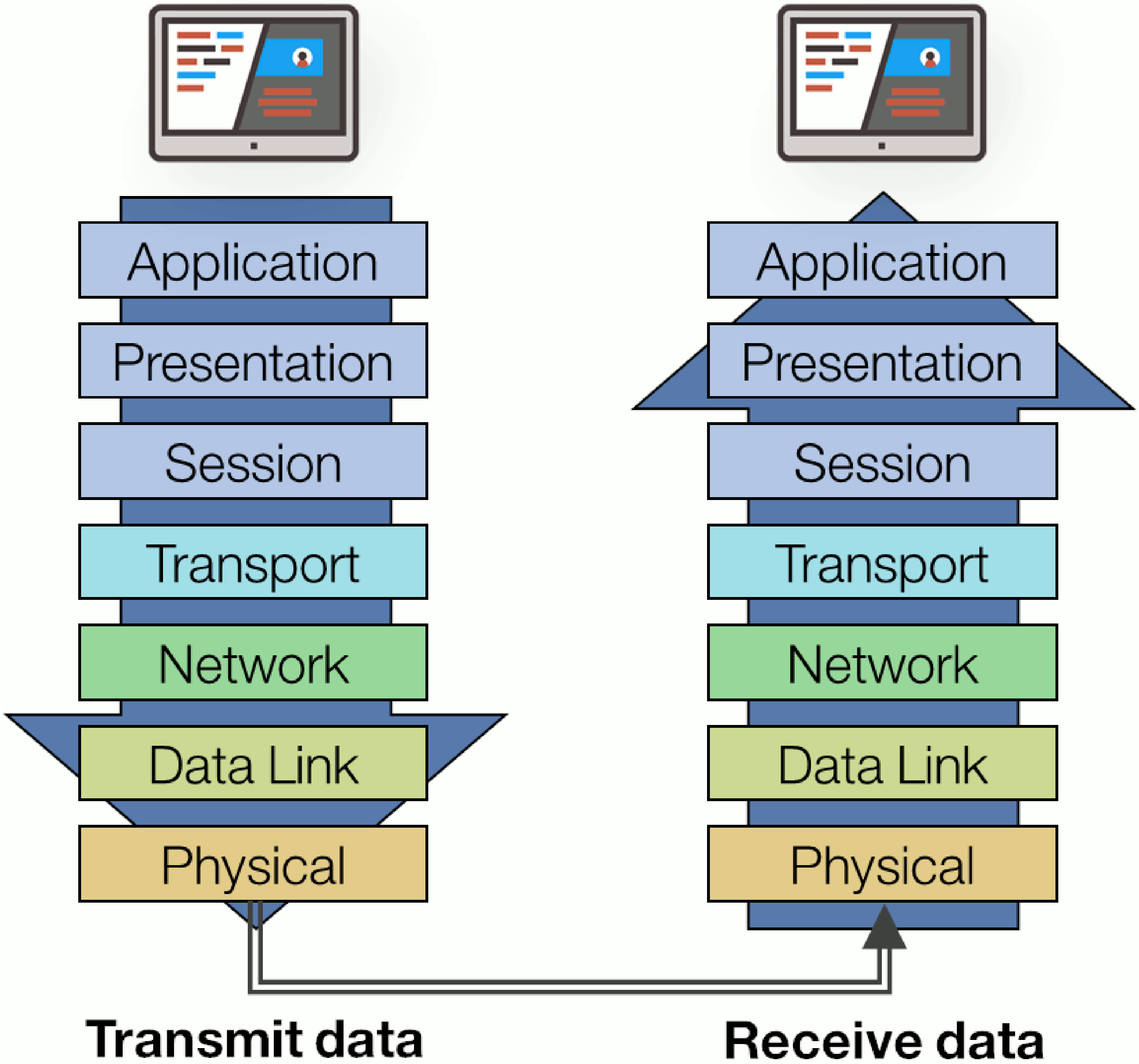


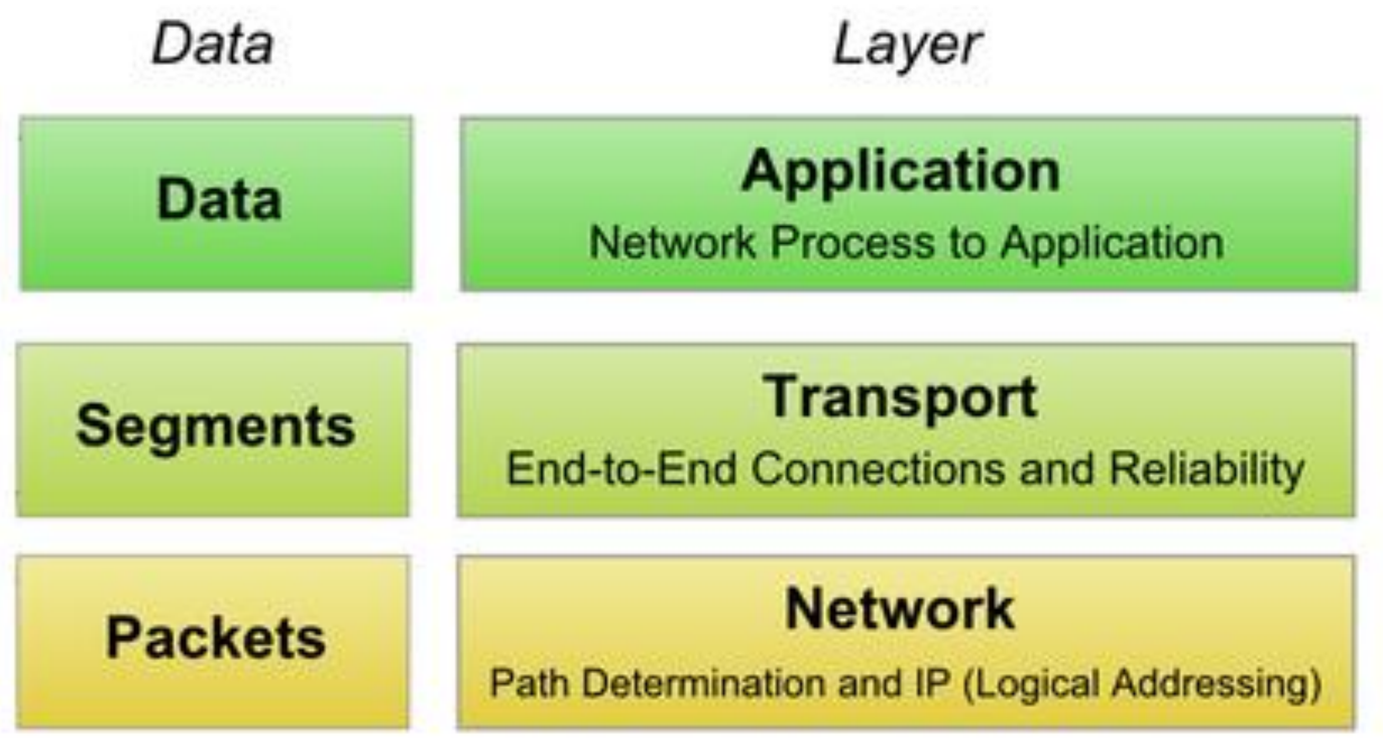
Networks

- Overview
- IP
- TCP
- HTTP
- C# Networking (and Threading briefly)



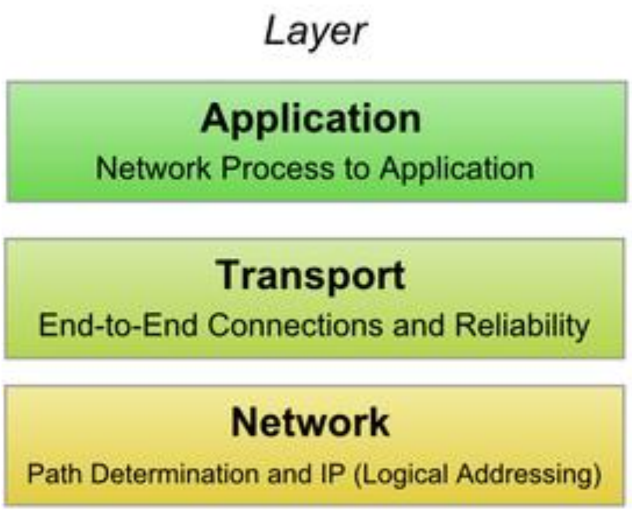
LAYERS
THE OSI MODEL

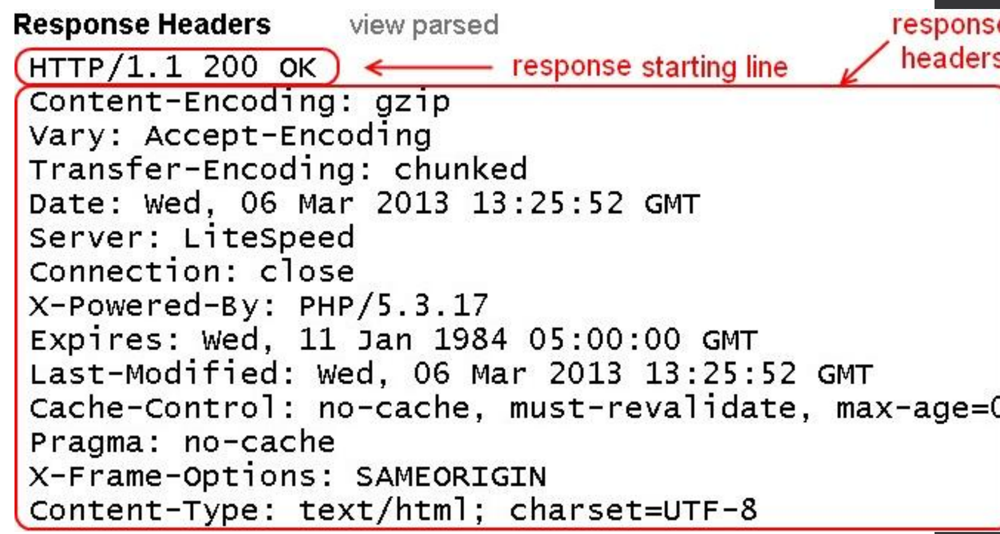

Application
Presentation
Session
Transport
Network
Data Link
Physical
Application
Presentation
Session
Transport
Network
Data Link
Physical
Transmit data
Receive data

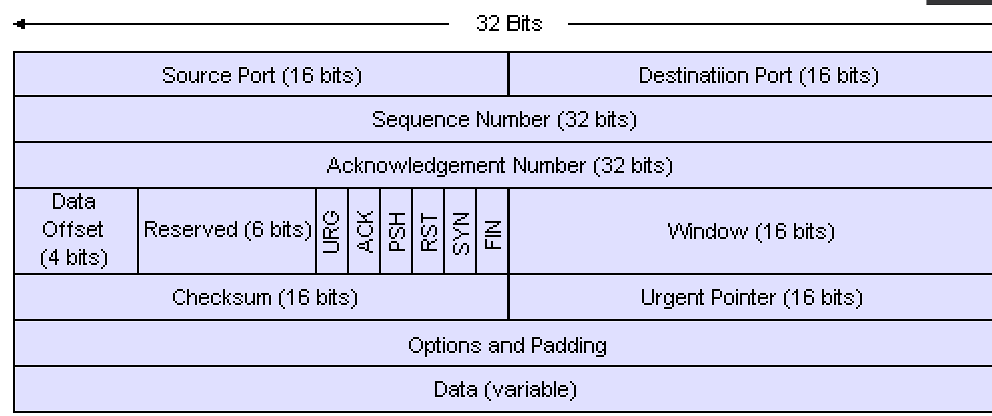

Data
Layer
Data
Application
Network Process to Application
Segments
Transport
End-to-End Connections and Reliability
Packets
Network
Path Determination and IP (Logical Addressing)

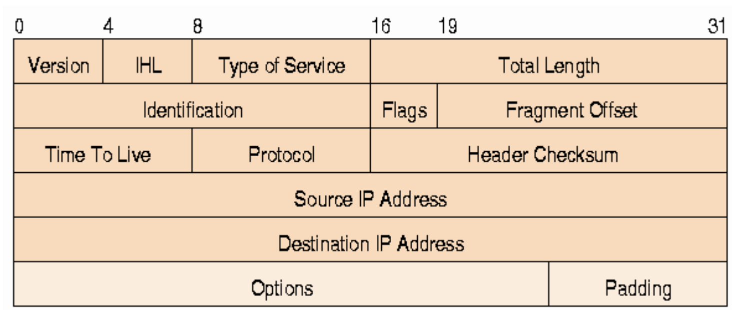
Layers

Protocols


Layer
Application
Network Process to Application
Transport
End-to-End Connections and Reliability
Network
Path Determination and IP (Logical Addressing)


Response Headers
view parsed
response headers
HTTP/1.1 200 OK
response starting line
Content-Encoding: gzip
Vary: Accept-Encoding
Transfer-Encoding: chunked
Date: Wed, 06 Mar 2013 13:25:52 GMT
Server: LiteSpeed
Connection: close
X-Powered-By: PHP/5.3.17
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Last-Modified: Wed, 06 Mar 2013 13:25:52 GMT
Cache-Control: no-cache, must-revalidate, max-age=0
Pragma: no-cache
X-Frame-Options: SAMEORIGIN
Content-Type: text/html; charset=UTF-8


32 Bits
Source Port (16 bits)
Destinatiion Port (16 bits)
Sequence Number (32 bits)
Acknowledgement Number (32 bits)
Data Offset (4 bits)
Reserved (6 bits)
URG
ACK
PSH
RST
SYN
FIN
Window (16 bits)
Checksum (16 bits)
Urgent Pointer (16 bits)
Options and Padding
Data (variable)

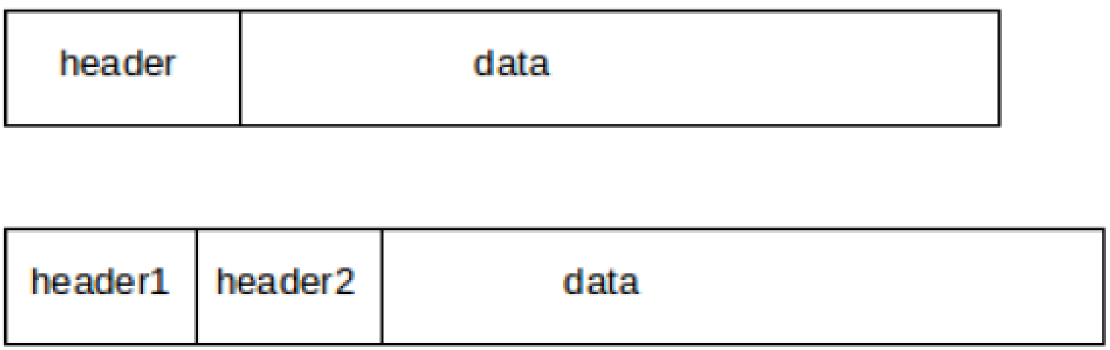

0
4
8
16
19
31
Version
IHL
Type of Service
Total Length
Identification
Flags
Fragment Offset
Time To Live
Protocol
Header Checksum
Source IP Address
Destination IP Address
Options
Padding

Core Concepts

- Bandwidth and Throughput
  - Data rate (the rate at which bits are transmitted)
  - Throughput (overall effective transmission rate)
  - Bandwidth ?
  - Goodput ?

Core Concepts

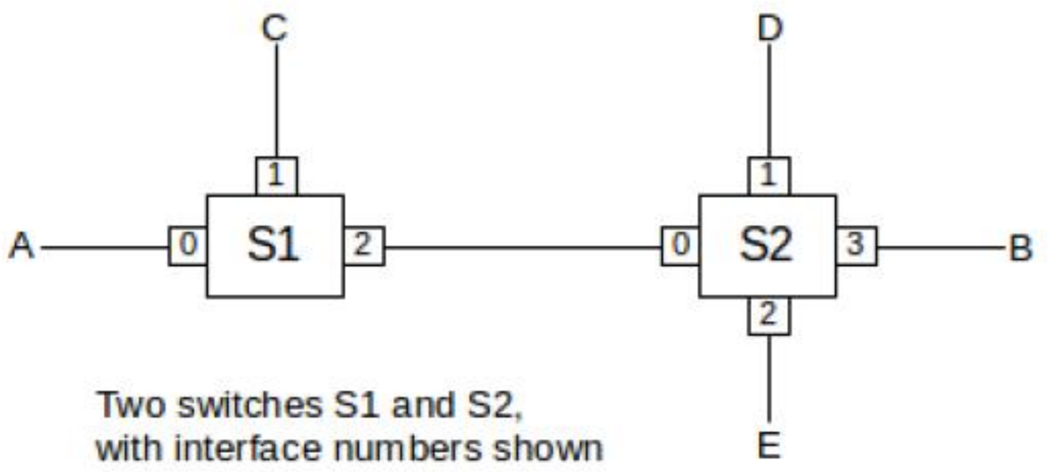
- Packets
  - Packets are modest-sized buffers of data, transmitted as a unit through some shared set of links



Single and multiple headers

Core Concepts

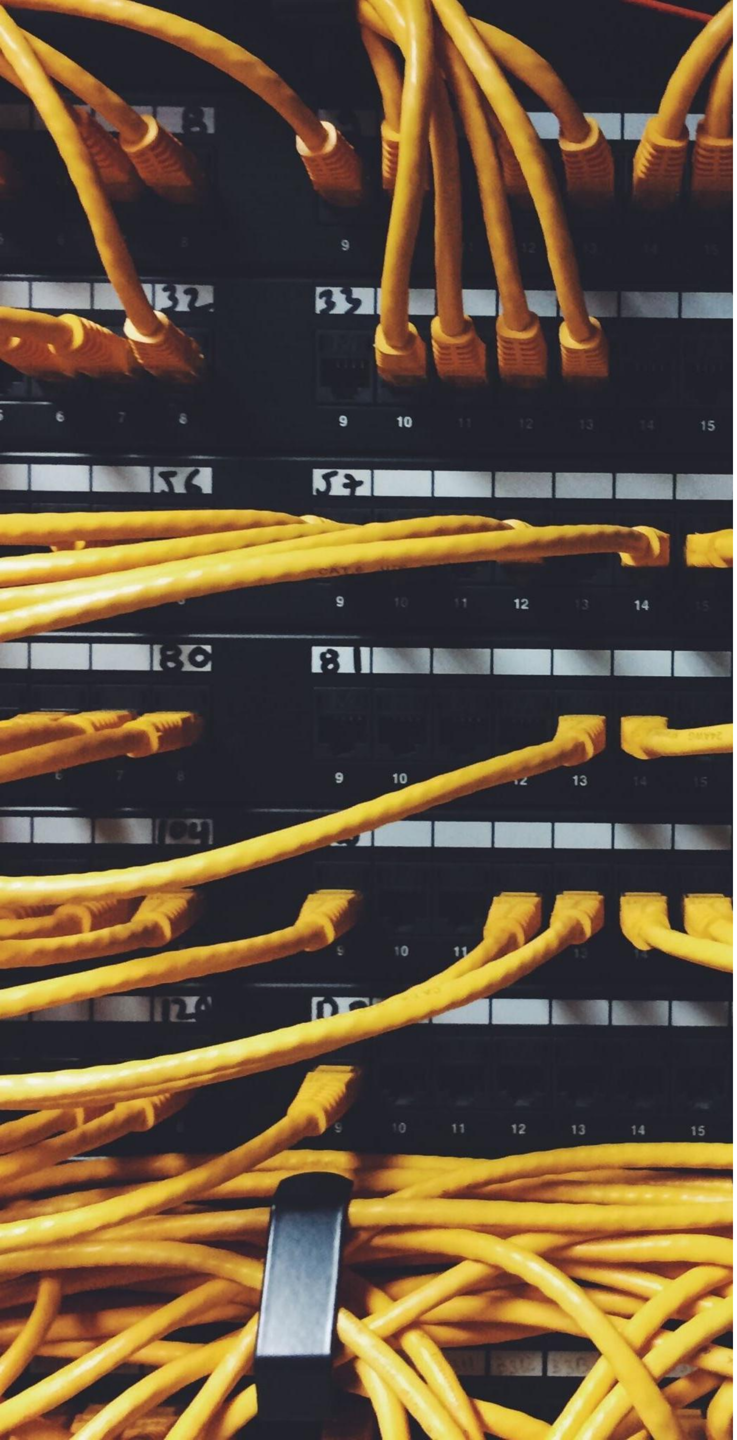
- Routing and Switching
  - In the datagram-forwarding model of packet delivery, packet headers contain a destination address. It is up to the intervening switches or routers to look at this address and get the packet to the correct destination



Two switches S1 and S2, with interface numbers shown

Core Concepts

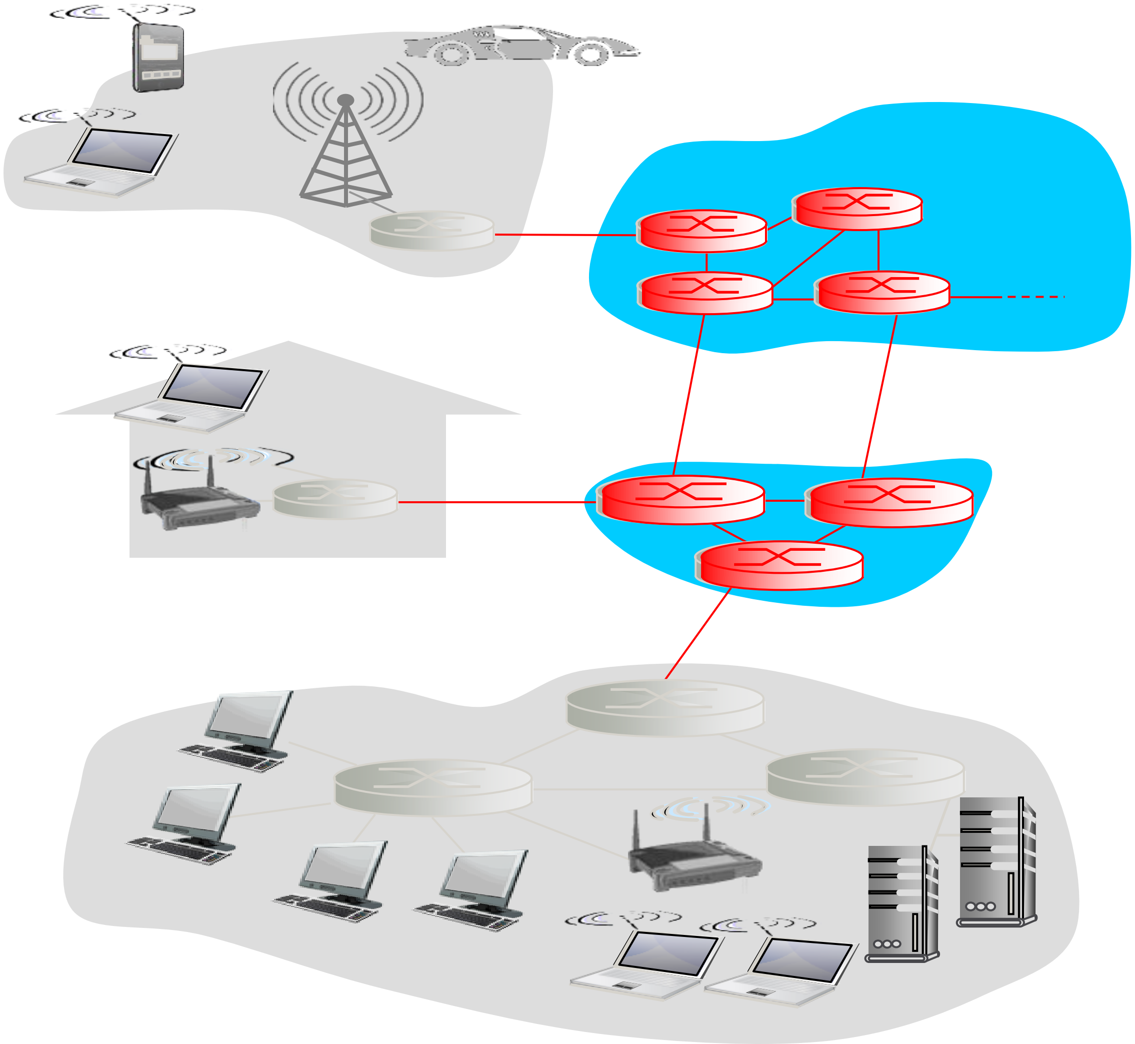
- Congestion
  - Problem
    - packets arriving faster than they can be sent out
    - multiple inputs and all destined for the same output
  - Solution
    - queue incoming packets
    - drop packets



Core Concepts

- LAN
  - physical links that are, ultimately, serial lines
  - common interfacing hardware connecting the hosts to the links
  - protocols to make everything work together

- Ethernet
  - hardware address or MAC (Media Access Control) address
  - broadcast
  - unicast
  - switched ethernet



The Internet

What is a protocol?

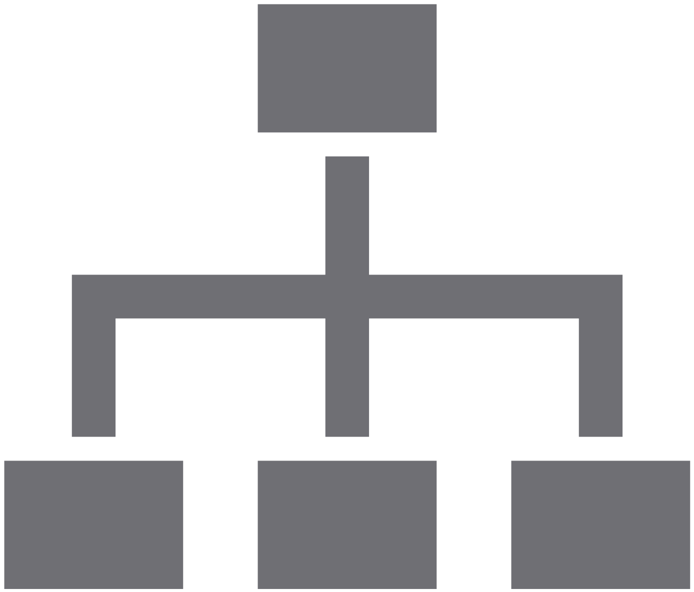
In diplomatic circles, a protocol is the set of rules governing a conversation between people

The client and server carry on a machine-to-machine conversation

A network protocol is the set of rules governing a conversation between a client and a server

Network protocols

- The details are only important to developers.
- The rules are defined by the inventor of the protocol – may be a group or a single person.
- The rules must be precise and complete so programmers can write programs that work with other programs.
- The rules are often published as an RFC along with running client and server programs.



RFC = request for comments

https://www.rfc-editor.org/rfc/rfc1945.txt

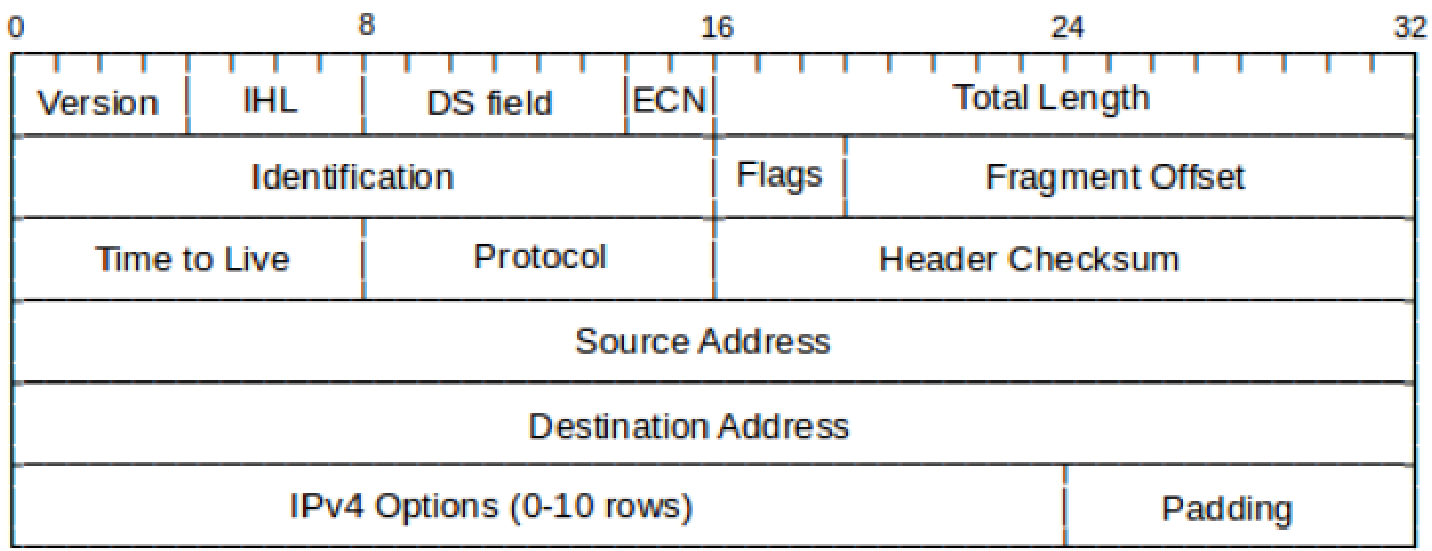
Network Layer

IP Protocol

- point-to-point links
- support universal connectivity (everyone can connect to everyone else)
- IPv4 (4 bytes)
- IPv6 (16 bytes)

IP Protocol

- destination and source adresses
- indication of ipv4 versus ipv6
- a Time To Live (TTL) value, to prevent infinite routing loops
- a field indicating what comes next in the packet (e.g. TCP v UDP)



IP Address

- IP provides a global mechanism for addressing and routing
- An essential feature of IPv4 (and IPv6) addresses is that they can be divided into a “network” part (a prefix) and a “host” part (the remainder).

first few bits	first byte	network bits	host bits	name	application
0	0-127	8	24	class A	a few very large networks
10	128-191	16	16	class B	institution-sized networks
110	192-223	24	8	class C	sized for smaller entities

- IP addresses, unlike Ethernet addresses, are administratively assigned

IP Address (IPv4)

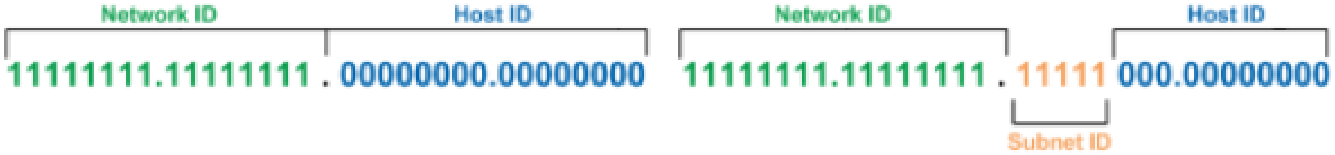
- A 32 bit address that is used to uniquely identify a computer on a network
- The Network ID portion of the IP Address identifies the network where the computer sits
- The Host ID portion of the IP Address uniquely identifies the computer on its network

IP Address:	192.168.10.1	192.168.10.1	192.168.10.1
Subnet Mask:	255.255.255.0	255.255.0.0	255.0.0.0
Addresses:	254	65,534	16,777,214
Class:	C	B	A

Classful vs Classless IP networks

IP Address:	192.168.10.1	192.168.10.1	192.168.10.1
Subnet Mask:	255.255.255.0	255.255.0.0	255.0.0.0
Addresses:	254	65,534	16,777,214
Class:	C	B	A
CIDR:	192.168.10.1/24	192.168.10.1/16	192.168.10.1/8
	255.255.255.0	255.255.0.0	255.0.0.0

Subnets



11000000.10101000.00001010.00000000	192.168.10.0
11000000.10101000.00001010.00000001	192.168.10.1
11000000.10101000.00001010.00000010	192.168.10.2
11000000.10101000.00001010.00000011	192.168.10.3
11000000.10101000.00001010.00000100	192.168.10.4
11000000.10101000.00001010.00000101	192.168.10.5
11000000.10101000.00001010.00000110	192.168.10.6
11000000.10101000.00001010.00000111	192.168.10.7

Mask: 255.255.255.248

Subnets

~~11000000.10101000.00001010.00000000~~	~~192.168.10.0~~
11000000.10101000.00001010.00000001	192.168.10.1
11000000.10101000.00001010.00000010	192.168.10.2
11000000.10101000.00001010.00000011	192.168.10.3
11000000.10101000.00001010.00000100	192.168.10.4
11000000.10101000.00001010.00000101	192.168.10.5
11000000.10101000.00001010.00000110	192.168.10.6
~~11000000.10101000.00001010.00000111~~	~~192.168.10.7~~

Network ID:	192.168.10.0
Host ID's:	192.168.10.1 – 192.168.10.6
Broadcast ID:	192.168.10.7

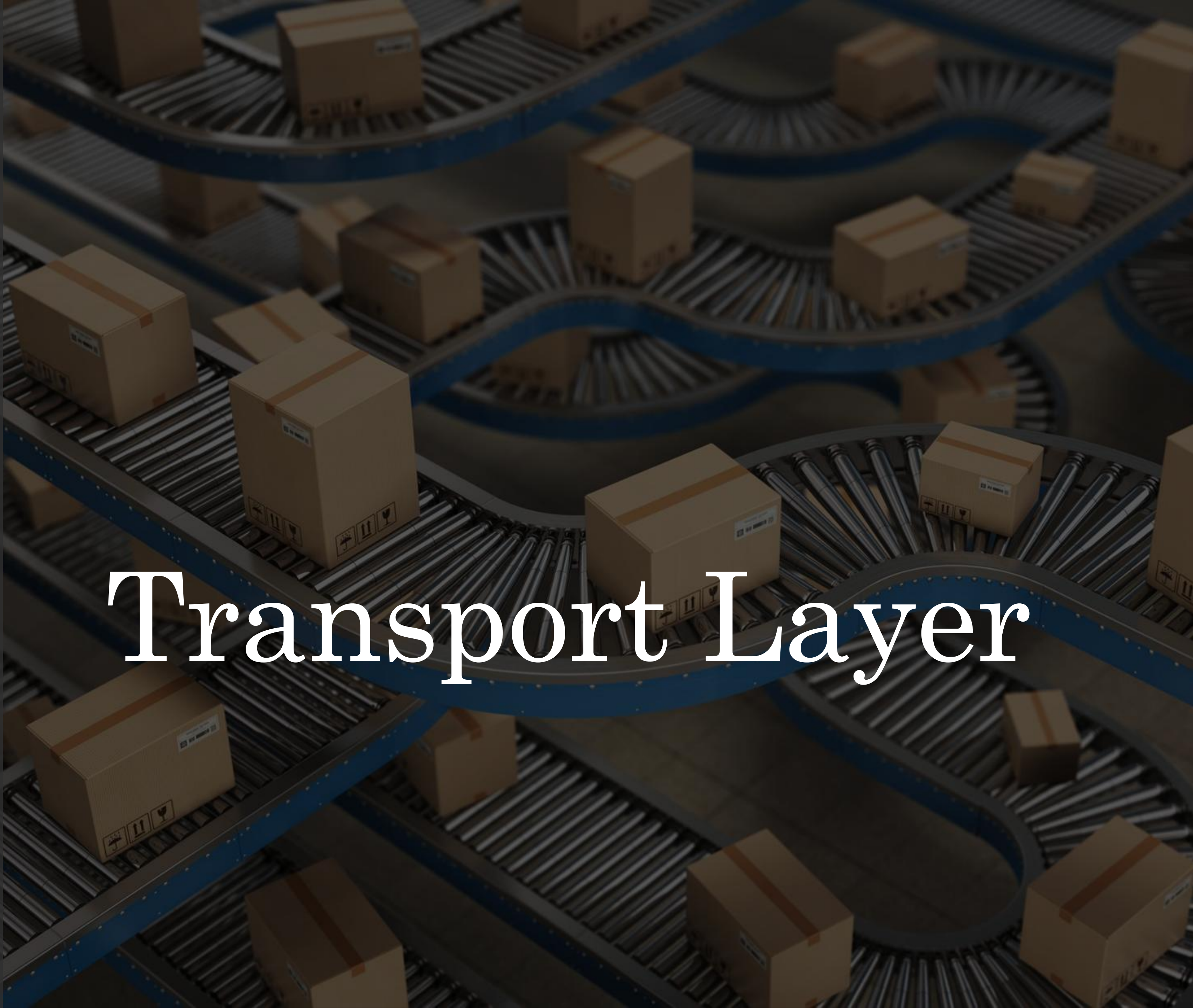
Subnets

~~11000000.10101000.00001010.00001000~~	~~192.168.10.8~~
11000000.10101000.00001010.00001001	192.168.10.9
11000000.10101000.00001010.00001010	192.168.10.10
11000000.10101000.00001010.00001011	192.168.10.11
11000000.10101000.00001010.00001100	192.168.10.12
11000000.10101000.00001010.00001101	192.168.10.13
11000000.10101000.00001010.00001110	192.168.10.14
~~11000000.10101000.00001010.00001111~~	~~192.168.10.15~~

Network ID:	192.168.10.8
Host ID's:	192.168.10.9 – 192.168.10.14
Broadcast ID:	192.168.10.15

IPv6 – Internet Protocol Version 6

- **Why IPv6?**
  - Designed to replace IPv4 due to address exhaustion.
  - Supports vastly more devices in the modern internet.
- **Key Features:**
  - **128-bit address space** → ~3.4×10^{38} unique addresses.
  - **Example address:** 2001:0db8:85a3:0000:0000:8a2e:0370:7334
  - **Simplified header** for faster processing.
  - **Built-in security** via IPsec.
  - **Auto-configuration** (no need for DHCP).
- **Benefits:**
  - Virtually unlimited IP addresses.
  - Better support for mobile and IoT devices.
  - Eliminates the need for NAT (Network Address Translation).
- **Challenges:**
  - Coexistence with IPv4 during transition.
  - Slow adoption in some regions and networks.



Transport Layer

What Is a Port in Networking?

- Definition:

  A port is a logical endpoint for communication.It helps identify specific services or processes on a device.

- How It Works:

  IP address identifies the device.Port number identifies the application/service.

- Format:

  Ports range from 0 to 65,535Example: 192.168.1.10:443 → IP +

- PortTypes of Ports:

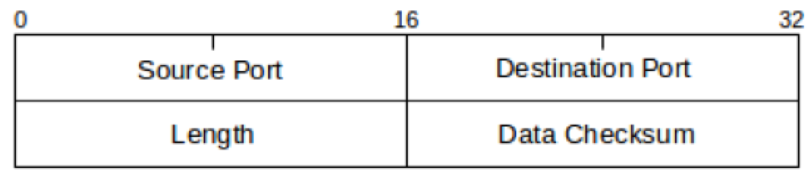
Port Range	Type	Examples
0–1023	Well-known ports	HTTP (80), HTTPS (443), FTP (21)
1024–49151	Registered ports	Custom apps/services
49152–65535	Dynamic/private	Temporary connections

- Use Case Example:

  When you visit a website over HTTPS, your browser connects to port 443 on the server.

UDP Protocol

- almost a null protocol



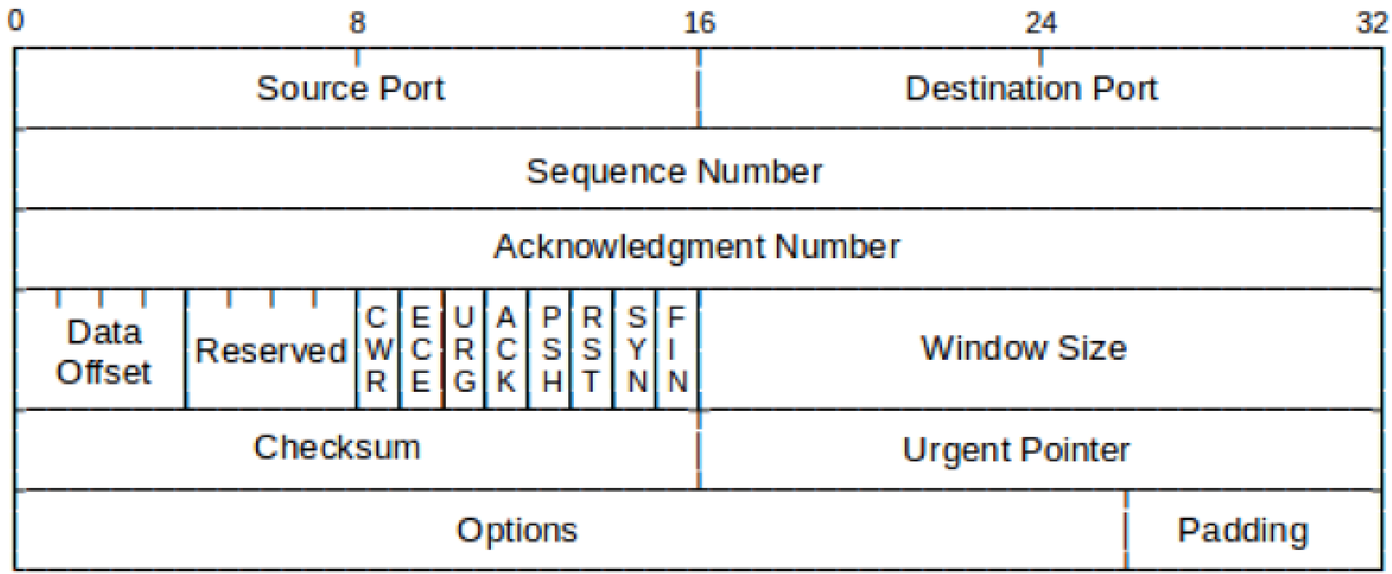
- unreliable
- common to use UDP as basis for a Remote Procedure Call
- well-suited for “request-reply” semantics
- popular for real-time transport

TCP Protocol

- **reliability**: TCP numbers each packet and keeps track of which are lost and retransmits them after a timeout and holds early-arriving out-of-order packets for delivery at the correct time. Every arriving data packet is acknowledged by the receiver; timeout and retransmission occurs when an acknowledgment isn't received by the sender within a given time.
- **connection-orientation**: Once a TCP connection is made, an application sends data simply by writing to that connection. No further application-level addressing is needed.
- **stream-orientation**: The application can write 1 byte at a time, or 100KB at a time; TCP will buffer and/or divide up the data into appropriately sized packets.
- **port numbers**: these provide a way to specify the receiving application for the data, and also to identify the sending application.
- **throughput management**: TCP attempts to maximize throughput, while at the same time not contributing unnecessarily to network congestion.

TCP Header

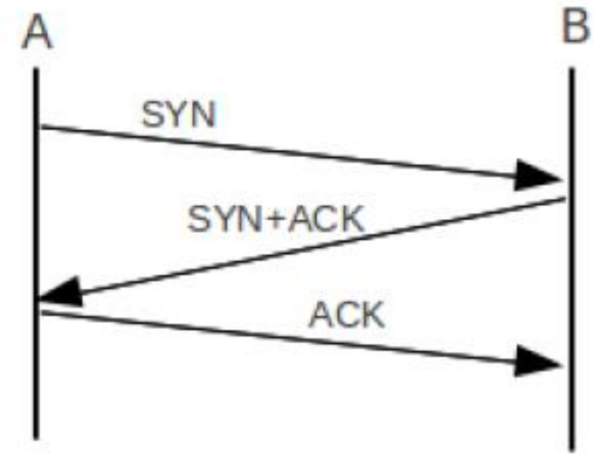
- sequence and acknowledgment numbers are for numbering the data



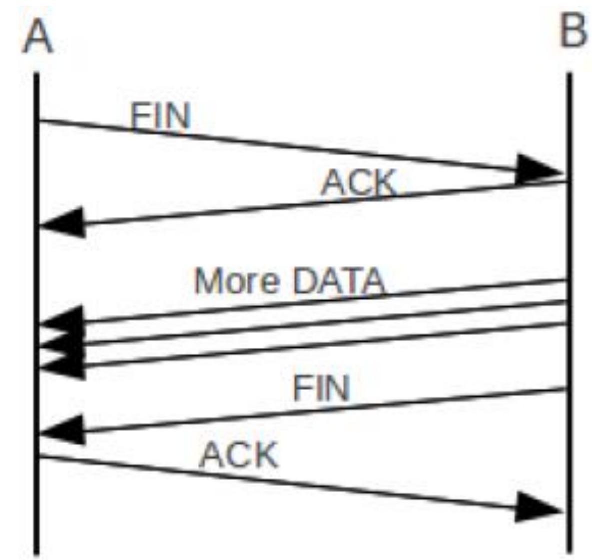
- flags:
  - SYN: for SYNchronize; marks packets that are part of the new-connection handshake
  - ACK: indicates that the header Acknowledgment field is valid; that is, all but the first packet

TCP Connection Establishment

- The handshake proceeds as follows
  - A sends B a packet with the SYN bit set (a SYN packet)
  - B responds with a SYN packet of its own; the ACK bit is now also set
  - A responds to B's SYN with its own ACK



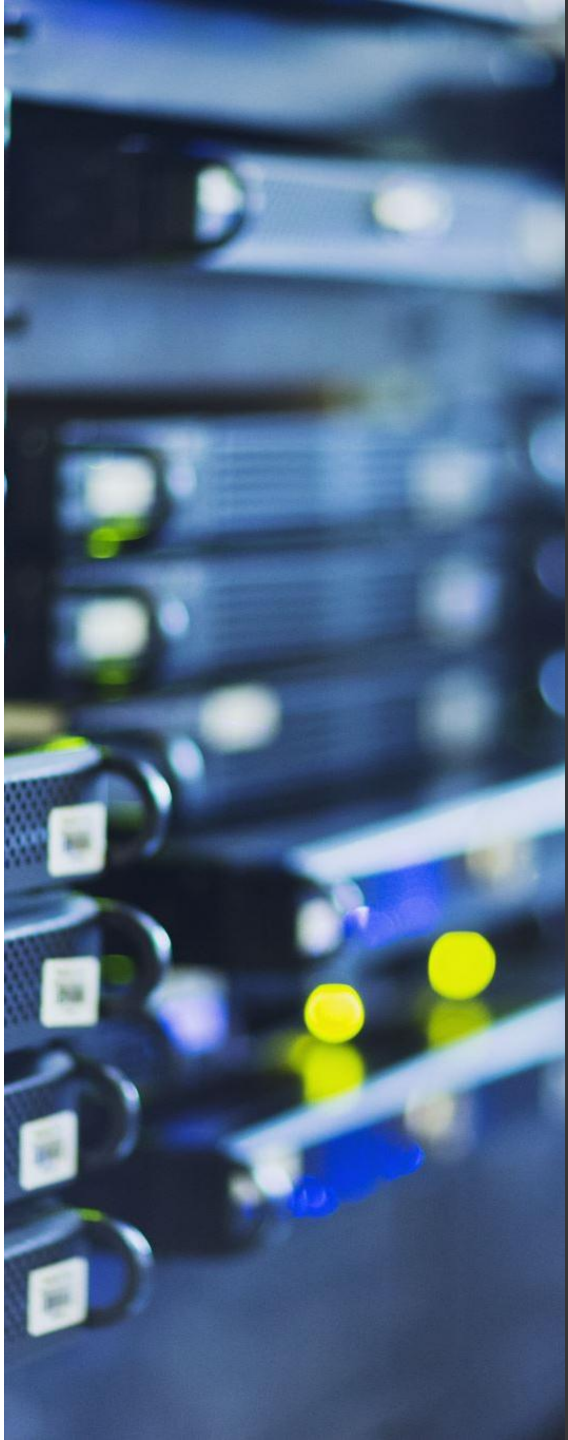
- Close the connection
  - A sends B a packet with the FIN bit set (a FIN packet), announcing that it has finished sending
  - B sends A an ACK of the FIN
  - When B is also ready to cease sending, it sends its own FIN to A
  - A sends B an ACK of the FIN; this is the final packet in the exchange

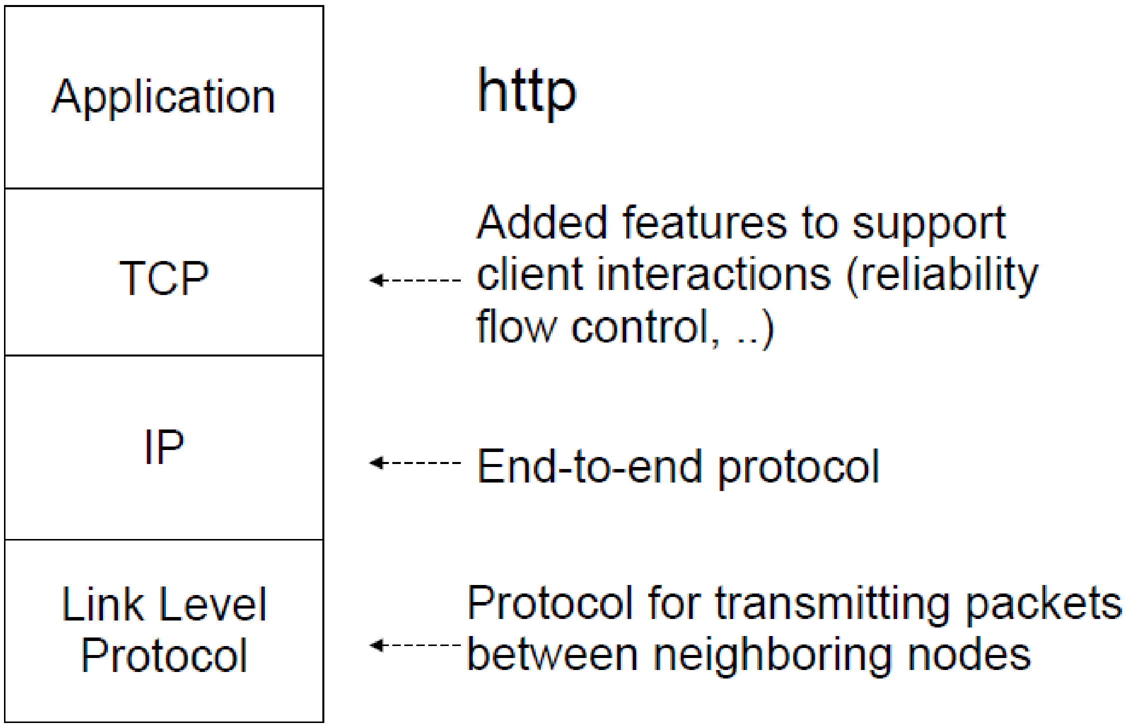


Application Layer

Hypertext Transfer Protocol

- HTTP, or Hypertext Transfer Protocol, is the foundation of data communication on the World Wide Web.
- A high-level protocol built on top of a TCP connection for exchanging messages (with *arbitrary content)*
  - Each (request) message from client to server is followed by a (response) message from server to client.
  - Facilitates the remote invocation of methods on the server.
- Web: A set of client and server processes on the Internet that communicate via HTTP.





PROTOCOL STACK

Clients and Servers: HTTP Request and Response

- An HTTP request is made by a client (usually a web browser) to a server, which then responds with the requested resource or an error message.
- This request-response cycle is fundamental to how the web operates
- Client: browser capable of displaying HTML pages.
- Web Server: stores pages for distribution to clients.
- Pages are identified by Uniform Resource Locator (URL).
  - *<protocol>: protocol to be used to communicate with host.*
  - Example - http, ftp

Start line: *<method>* *<URL>* *<protocol_version>* *CrLf*

Followed by: *<header>**

Followed by: CrLf

Followed by: *<data>*

there can be several header lines

<method> = GET | HEAD | POST | PUT | ….

<protocol_version> = HTTP/1.1 | ….

HTTP REQUEST FORMAT

Request Methods

- **GET** – response body contains data identified by argument URL
- **HEAD** – response header describes data identified by argument URL (no response body)
  - Use: has page changed since last fetched?
- **PUT** – request body contains page to be stored at argument URL
- **DELETE** – delete data at argument URL
- **POST** – request body contains a new object to be placed subordinate to object at argument URL

  – Use: adding file to directory named by URL

  – Use: information entered by user on displayed forms
- Others ….

Client/Server Interaction

1. User supplies URL (clicks on link)

   http://yourbusiness.com/~items/printers
   .html

2. Browser translates <*host_name*> *(yourbusiness.com)* to *host_ip_address*

3. Browser assumes a port number of 80 for http (if no port is explicitly provided as part of <*host_name*> *)*
   - Program at port 80 interprets http headers

4. Browser sets up TCP connection to yourbusiness.com at *(host internet address, 80)*

5. Browser sends http message
   GET ~items/printers.html HTTP/1.0

HTTP Response

Status line: <*HTTP_version*>
<*status_code*>

<*reason_line*> *CrLf*

Followed by: < *header* >*

Followed by: <*data*>

HTTP Response

<status_code> = 3 digits

Ex: 2xx -- success

4xx -- bad request from client

5xx -- server failed to fulfill valid request

<reason_line> = explanation for human reader

<header> = <field_name> : <value> CrLf

<field_name> =

Allowed | -- methods supported by URL

Date | -- creation date for response

Expires | -- expiration date for data

Last-Modified | -- creation date for object

Content-Length | Content-Type | ….

Client/Server Interaction

6. Server sends response message with requested html page to browser

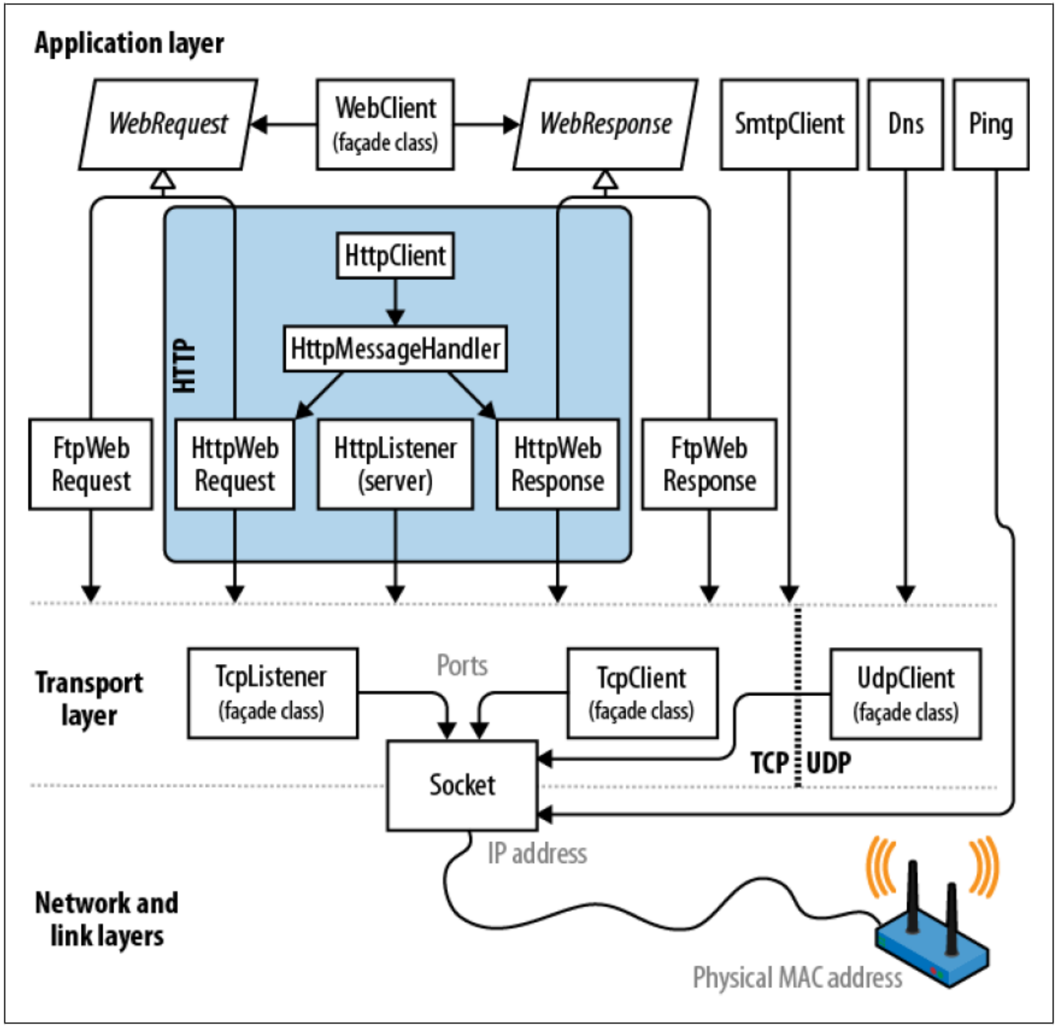
```
HTTP/1.0 200 Document follows
Date: <date>
Content-Type: text/html
Content-Length: integer
Expires: date

html document ~items/printers.html goes here
```

7. Server releases TCP connection (stateless)

8. Browser receives page and displays it

HTTP Performance

- HTTP/1.0 allowed one transaction per connection
  - TCP connection setup and teardown are expensive
  - TCP's *slow start* slows down the initial phase of data transfer
  - typical Web pages use between 10-20 resources (HTML + images)
  - typically, these resources are stored on the same server
- HTTP/1.1 introduces *persistent connections*
  - the TCP connection stays open for some time (10 sec is a popular choice)
  - additional requests to the same server use the same TCP connection
- HTTP/1.1 introduces *pipelined connections*
  - instead of waiting for a response, requests can be queued
  - the server responds as fast as possible
  - the order may not be changed (there is no sequence number)
- HTTP/2
  - introduces *multiplexing*
  - server push
  - binary protocol
  - header compression
- HTTP/3
  - QUIC protocol
  - UDP



C# Networking

C# Networking

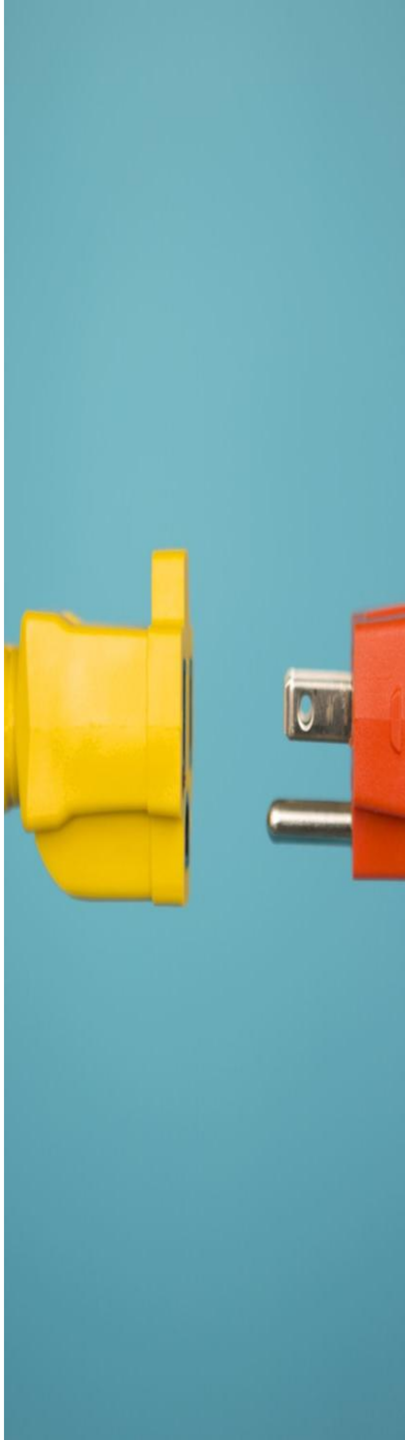
Sockets

TcpClient/TcpListener

HttpClient

Sockets

```
var host = "roskilde.dk";
var ipHostEntry = Dns.GetHostEntry(host);
var ipAddress = ipHostEntry.AddressList[0];
var ipEndPoint = new IPEndPoint(ipAddress, 80);
var socket = new Socket(
 AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
Socket.Connect(ipEndPoint);
```



TcpClient/TcpListener

```
using (TcpClient client = new TcpClient())
{
  client.Connect ("address", port);
  using (NetworkStream n = client.GetStream())
  {
    // Read and write to the network stream...
  }
}
```

```
TcpListener listener = new TcpListener (<ip address>, port);
listener.Start();

while (keepProcessingRequests)
  using (TcpClient c = listener.AcceptTcpClient())
  using (NetworkStream n = c.GetStream())
  {
    // Read and write to the network stream...
  }

listener.Stop();
```

Principles of Asynchrony

- Synchronous Versus Asynchronous Operations
  - A *synchronous operation* does its work *before* returning to the caller
  - An *asynchronous operation* does (most or all of) its work *after* returning to the caller

Concurrency and Asynchrony: Threads

- Threads allow multiple tasks to run concurrently, improving the responsiveness and performance of applications.
  - A *thread* is an execution path that can proceed independently of others

```
Thread t = new Thread (WriteY);
t.Start();

static void WriteY()
{
  for (int i = 0; i < 1000; i++) Console.Write ("y");
}
```

- Join and Sleep
  - You can wait for another thread to end by calling its Join method
  - Thread.Sleep pauses the current thread for a specified period

Thread Example

```
using System;
using System.Threading;

Thread t = new Thread (WriteY);          // Kick off a new thread
t.Start();                               // running WriteY()

// Simultaneously, do something on the main thread.
for (int i = 0; i < 1000; i++) Console.Write ("x");

void WriteY()
{
  for (int i = 0; i < 1000; i++) Console.Write ("y");
}

// Typical Output:
xxxxxxxxxxxxxxxxyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxyyyyyyyyyyyyy
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
yyyyyyyyyyyyyxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
...
```

Concurrency and Asynchrony: Tasks

- A task represents an asynchronous operation.
- Tasks are used for asynchronous programming and can be created using the Task class.
- Tasks provide a higher-level abstraction for managing asynchronous operations.
  - Tasks are managed by the Task Scheduler, which optimizes the use of system resources. Threads, on the other hand, require more manual management.
- They are often used with the async and await keywords to simplify asynchronous code. For example:

```
var result = await expression;
statement(s);
```

- Task.Delay
  - Task.Delay is the *asynchronous* equivalent of Thread.Sleep

```
Task.Run (() => Console.WriteLine ("Foo"));
new Thread (() => Console.WriteLine ("Foo")).Start();
```

Tasks

- Awaiting

```
var result = await expression;
statement(s);
```

```
var awaiter = expression.GetAwaiter();
awaiter.OnCompleted (() =>
{
  var result = awaiter.GetResult();
  statement(s);
});
```

- Async

```
public async Task<int> SomeMethod()
```