

The Internet

ARPANET (1969-1990)

International Network Working Group (1972)

NSFNet

Problem

Network Model

Services and Protocols

TCP/IP Model

OSI Model

Why do we need a model?

Advantages and Disadvantages of having international standards for network protocols.

Connection-oriented and Connectionless services

Connection Oriented (TCP)

Connectionless (UDP)

The protocol stack

World Wide Web

Components

Architecture

Application Layer

HTTP

Overview

Persistent vs. Non-persistent

Summary of key steps

HTTP Methods

HTTP Request

HTTP Response Codes

HTTP Response

HTTP Headers

Client side processing

Server side processing

Multi-threaded Web Server

Web Cache

Web proxy

Cookies

Tracking

URL/URI

HTML

XML & XSL

XHTML

Dynamic Content

Client-side Scripting

Domain Name System

Elements of DNS

Domain name

Conceptual division of DNS Namespace

Top-level domains

Resource Records

Inserting records into DNS

Name server zones

Types of name servers

Resolving a query

DNS Security

Email

User agents

Format

Message Header Fields

SMTP

MIME

Transfer

Delivery

POP3 – Post Office Protocol

IMAP – Internet Message Access Protocol

Presentation Layer

Session Layer

Transport layer

Role

Services

Encapsulation

Addressing

Port allocations

Multiplexing /Demultiplexing

UDP – User Datagram Protocol

Connectionless

Strengths

Weaknesses

Applications

UDP header

UDP Question

RPC - Remote Procedure Calls

How RPC works?

Challenges

UDP vs TCP for RPC

RTP – Real-Time Transport Protocol (RTP)

Which layer is RTP at?

RTP Header

RTP Playback

RTCP - Real-time Transport Control Protocol

TCP - Transmission Control Protocol

TCP Services

Service Model

TL;DR

TCP Segment

TCP Primitives

TCP Connection

Features

Issues

Connection establishment

Three-way handshake

Synchronisation

Retransmission

Closing

SYN Flooding

TCP Sliding Window

Send window

Receive window

Potential for deadlock

Congestion Control Window CWND

Slow-start

Congestion Avoidance (Additive increase)

Fast recovery

Further Optimisations

Macroscopic model

TCP Question

- Socket Programming
 - Berkeley Sockets
 - Using Sockets in C
 - Multi-threaded Web Server

Git

- What is Git?
- VCS History
- Architecture
- Storage
- Usage
- Risks
- Git Question

Original flow + loss control

- Go-back-N
 - Sender
 - Receiver
- Selective Repeat
 - Sender
 - Receiver
- Congestion collapse
 - Question

Internet (network) Layer

- Role
- Where
- Services
 - Connectionless("datagram" network)
 - Connection-oriented("virtual circuit" network)
- Routing
 - Flooding
 - Optimality Principle
 - Sink Tree
 - Shortest Path Algorithm
 - Link State Routing
 - Distance vector Routing
 - Border Gateway Protocol (BGP)
- Packet forwarding
 - Connectionless
 - Connection-oriented
 - MultiProtocol Label Switching
 - Connection-oriented vs Connectionless
- Quality of Service
 - Why is Quality of Service (QoS) important
- Internet Protocol
- IP Addressing
 - IPv4 Addresses
- Classes
 - Classless Inter Domain Routing
 - Prefixing
 - Route aggregation
 - Special IP Addresses
- IPv6
- IP Header
 - IP Version 4 Protocol
 - IP Version 6 Protocol
- Subnets
 - Subnet Routing
- Network Address Translation (NAT)
 - How NAT works

- [Advantages](#)
- [Criticisms](#)
- [Fragmentation](#)
 - [Fragmentation and IP Headers](#)
 - [Example](#)
 - [Downsides](#)
- [Path MTU discovery](#)
- [IPv4 vs. IPv6 Fragmentation](#)

The Internet

ARPANET (1969-1990)

- TCP/IP was developed at ARPANET

International Network Working Group (1972)

- Proposed a packet switched datagram based network standard but rejected
- designed the OSI
- By the late 80's the slow development of the OSI model was leading to increased frustration

NSFNet

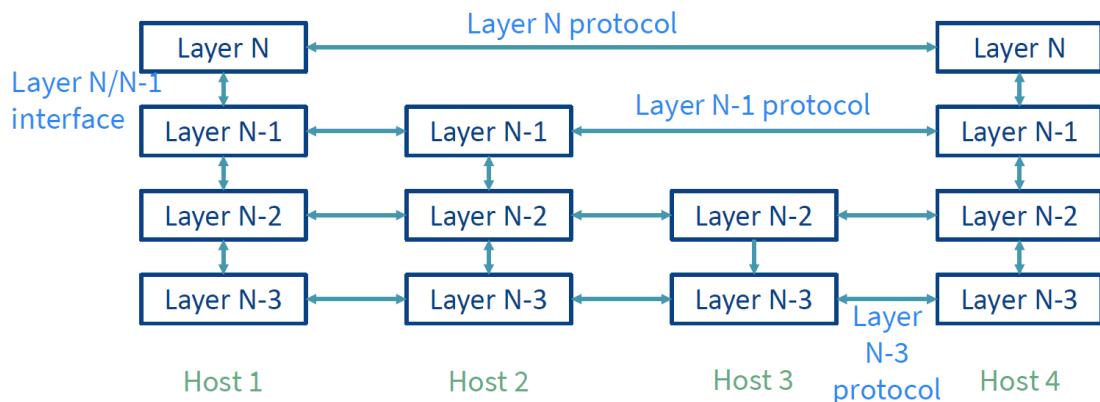
- National Science Foundation was created in 1986 to provide researchers access to supercomputer sites in the USA

Problem

- Many of the underlying protocols were designed without consideration for an adversary on the network
 - Security has been retrofitted, with many insecure protocols still widely used (DNS)
- The rivalry between the two groups led to TCP/IP working groups rejecting OSI concepts out of principle

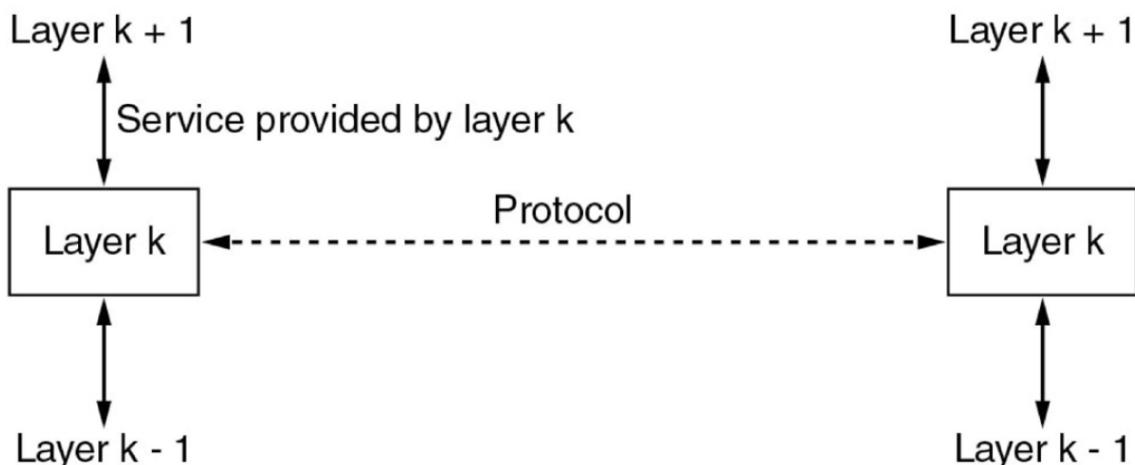
Network Model

- Model the network as a stack of layers. **Just a model**
- Each layer offers services to layers above it.
- Inter-layer exchanges are conducted according to a protocol.



- Model the network as a **stack** of layers.
- Each layer offers services to layers above it.
- Inter-layer exchanges are conducted according to a protocol.

Services and Protocols



- **Service:** set of primitives that a layer provides to a layer **above** it
 - interfaces between layers
- **Protocol:** rules which govern the format and meaning of packets that are exchanged by peers **within a layer**
 - packets sent between peer entities

Suppose the algorithms used to implement the operations at layer k are changed. Do the implementations of the operations at layers k - 1 and k + 1 need to change accordingly?

NO

- It is not necessary to change the layer k + 1 implementations as long as
 - layer k still provides the same services to layer k + 1
 - Since the role of layer k is to provide services to the layer k + 1 as long as the API's of the services have not been changed, there is no need for the layer k + 1 to change its implementations.

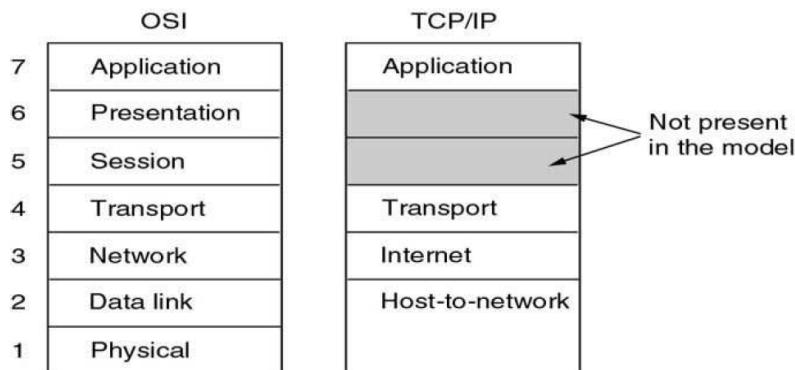
- o layer k still only uses services provided by layer k - 1
- The change in algorithm at layer k may change which services are used by layer k - 1 and how often they are made
 - o may change the costs of the services it provides to layer k + 1.
 - o these layers may choose to re-optimize the way they implement their services.

Suppose there is a change in the service (set of operations) provided by layer k.

How does this impact services at layers k - 1 and k + 1?

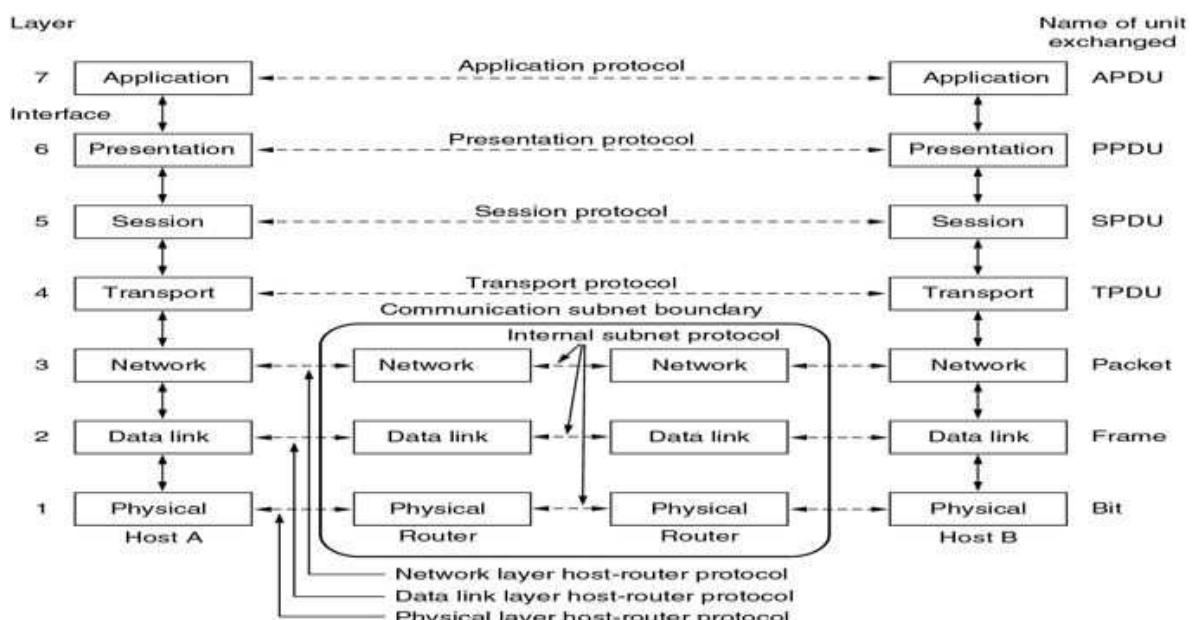
- k - 1: No impact
- k + 1: Have to be reimplemented

TCP/IP Model



- Transmission Control Protocol/Internet Protocol – was designed to be independent of data link and physical layers
- The TCP/IP model reflects what happens on the internet
- TCP/IP effectively standardised post implementation

OSI Model



- OSI standardised pre-implementation, but not widely implemented
- OSI helps reflect the thought process that should be followed when designing a network or diagnosing a fault
 - o It remains at the core of many accreditation schemes

- View the OSI model as idealised, but with a degree of flexibility
- **Layer:** A layer should be created where a different abstraction is needed.
- **Functionality:**
 - Each layer should perform a well defined function.
 - The function of each layer should be chosen with a view toward defining internationally standardised protocols
- **layer boundaries:** The layer boundaries should be chosen to minimise the information flow across the interfaces.
- **The number of layers** should be large enough that distinct functions need not to be thrown together in the same layer out of necessity, and small enough that the architecture does not become unwieldy.
- **Why did the OSI initiative fail?**
 - Politics? Poor design? Vested interests? Bad timing?
 - Design by committee – too open
 - In software engineering terms: Waterfall vs agile

Why do we need a model?

- Interoperability – Open, ideally not proprietary
- A reference model to develop and validate against independently
- Since networks are multi-dimensional, a reference model can serve to simplify the design process.
- It's engineering best practice to have an abstract reference model, and a reference model and corresponding implementations are always required for validation purposes

Advantages and Disadvantages of having international standards for network protocols.

Advantage

- if everyone uses the standard, everyone can talk to everyone.
- widespread use of any standard will give it economies of scale, as with VLSI chips.

Disadvantage

- Political compromises necessary to achieve standardization frequently lead to poor standards.
- Once a standard has been widely adopted, it is difficult to change, even if new and better techniques or methods are discovered.
 - Also, by the time it has been accepted, it may be obsolete.

Connection-oriented and Connectionless services

The choice of service type affects the reliability, quality and cost of the service itself.

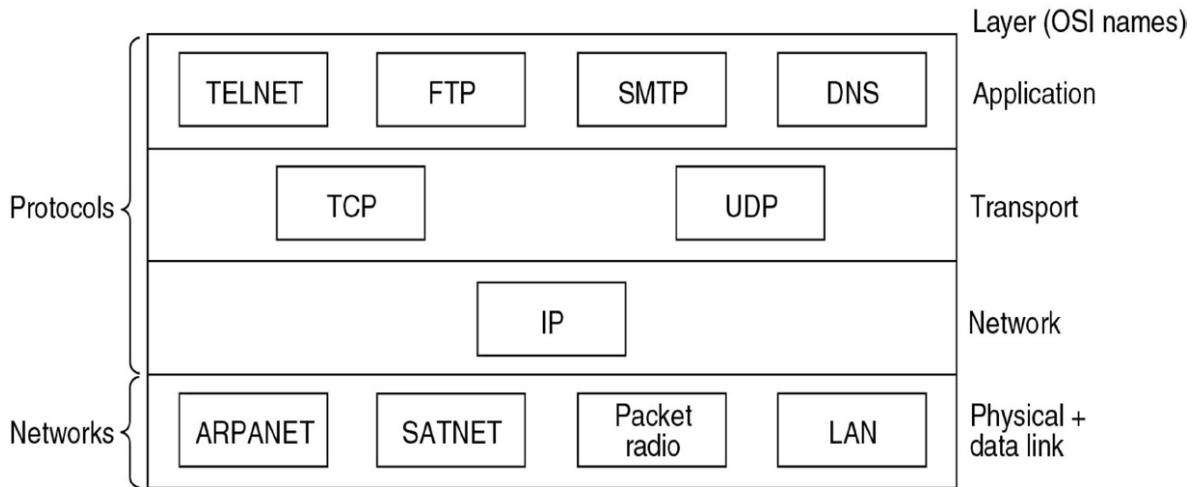
Connection Oriented (TCP)

- connect, use, disconnect
- negotiation inherent in connection setup similar to telephone service

Connectionless (UDP)

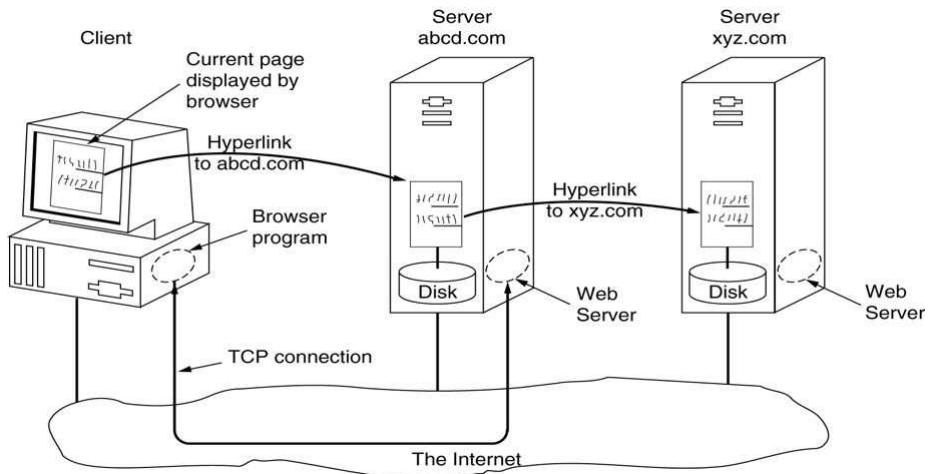
- Use message routed through intermediate nodes
- similar to postal service or text message

The protocol stack



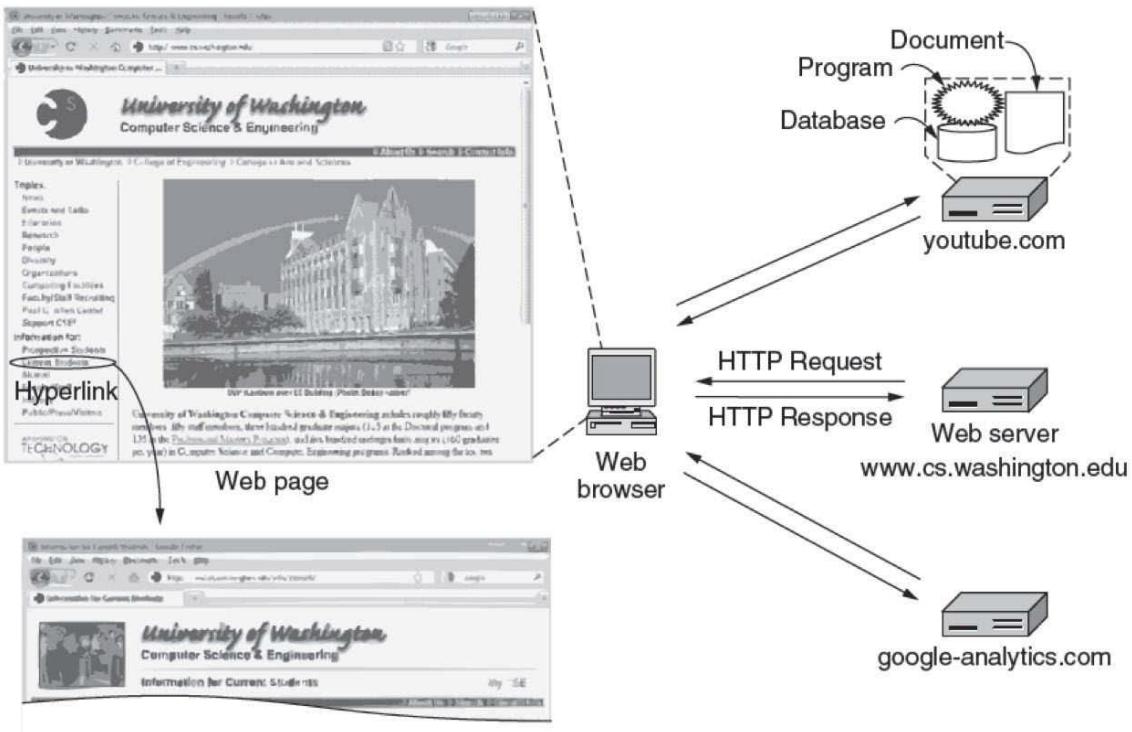
World Wide Web

Components



- Client - typically a browser based access to pages
- Server - daemon based content delivery of pages
- URL ≈ Protocol + DNS Name + file name

Architecture



Application Layer

HTTP

Overview

1. Client **initiates** TCP connection (creates socket) to server, port 80
2. Server **accepts** TCP connection from client
3. HTTP messages (application-layer protocol messages) **exchanged** between browser (HTTP client) and Web server (HTTP server)
4. TCP connection **closed**

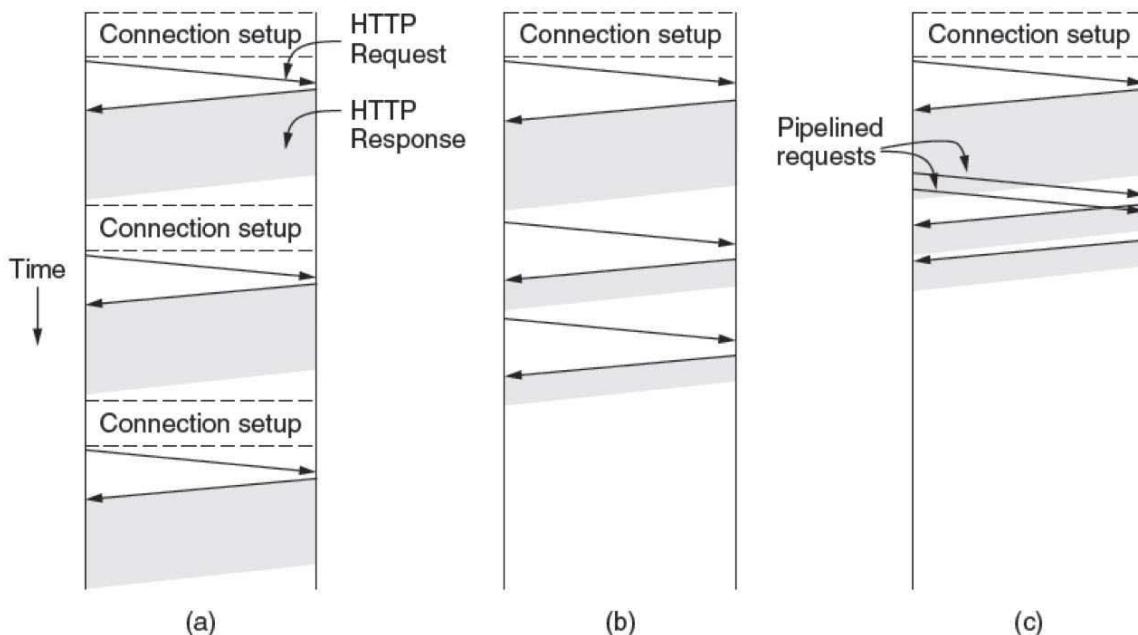
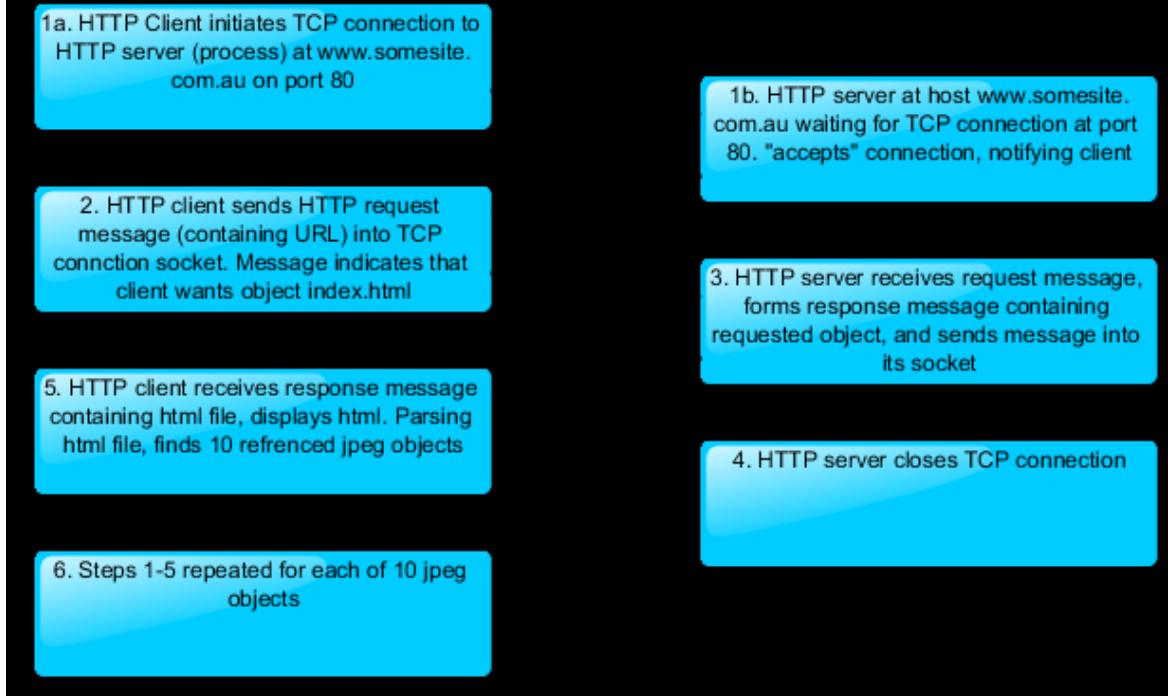
Connections

- HTTP 1.0 – single use connection
- HTTP 1.1 – persistent connections, additional headers
- HTTP/2 – 2015 – Further speed improvements (origins in SPDY)

Persistent vs. Non-persistent

Non-persistent	Persistent
requires 2 “response times” (one to initiate TCP connection and one for initial HTTP request) per object + file transmission time	server leaves connection open after sending response
OS overhead for each TCP connection	subsequent HTTP messages between same client/server sent over open connection
browsers often open parallel TCP connections to fetch referenced objects	client sends requests as soon as it encounters a referenced object, reducing overall response time

Suppose the user enters <http://www.somesite.com.au/index.html>
(the page contains text and references to 10 images)



- (a) multiple connections and sequential requests.
- (b) A persistent connection and sequential requests.
- (c) A persistent connection and pipelined requests.

Tutorial 2 Question 8

(c) With non-persistent connections between browser and origin server, it is possible for a single TCP connection to carry two distinct HTTP request messages.

False.

- In a non-persistent connection, the connection is terminated once the object has been successfully received by the client.
- The second HTTP request method will require a new TCP connection to be established.

- TL;DR one connection, one message, one response.

(b) Two distinct Web pages (for example, <http://cis.unimelb.edu.au/research/> and <http://cis.unimelb.edu.au/people>) can be sent over the same persistent connection.

True

- Multiple Web pages residing on the same server can be sent from the server to the same client over a single persistent TCP connection.

Summary of key steps

- Steps that occur when a link is selected
 1. Browser determines the URL
 2. Browser asks DNS for the IP address of the server (Resolving URL)
 3. DNS replies
 4. The browser makes a TCP connection
 5. Sends HTTP request for the page
 6. Server sends the page as HTTP response
 7. Browser fetches other URLs as needed
 8. The browser displays the page (progressively, as content arrives)
 9. The TCP connections are closed

HTTP Methods

HTTP Method	Safe	Idempotent	Cacheable
GET	Yes	Yes	Yes
HEAD	Yes	Yes	Yes
POST	No	No	Yes/No
PUT	No	Yes	No
DELETE	No	Yes	No
CONNECT	No	No	No
OPTIONS	Yes	Yes	No
TRACE	Yes	Yes	No
PATCH	No	No	No

- **Idempotent** 幕等性 : multiple identical requests have same effect
- **Safe:** Only for information retrieval, should not change state

HTTP Request

```
GET /somedir/page.html HTTP/1.1
Host: www.somesite.com.au
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr
Blank line (2 LF or 2 CR/LF) indicates end of message
```

Tutorial 2 Question 7

Consider the following string of ASCII characters that were captured by Wireshark when the browser sent an HTTP GET message (i.e., this is the actual content of an HTTP GET message). The characters crlf are carriage return and line-feed characters Answer the following questions, indicating where in the HTTP GET message below you find the answer.

```
GET /people/index.html HTTP/1.1<cr><lf>
Host: cis.unimelb.edu.au<cr><lf>
Connection: keep-alive<cr><lf>
Cache-Control: max-age=0<cr><lf>
Upgrade-Insecure-Requests: 1<cr><lf>
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.186
Safari/537.36<cr><lf>
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
image/webp,image/apng,*/*;q=0.8<cr><lf>
Accept-Encoding: gzip, deflate<cr><lf>
Accept-Language: en-AU,en;q=0.9<cr><lf>
<cr><lf>``
```

(a) What is the URL of the document requested by the browser?

http://cis.unimelb.edu.au/people/index.html

(b) What version of HTTP is the browser running?

HTTP/1.1

(c) Does the browser request a non-persistent or a persistent connection?

Connection: keep-alive

(d) What is the IP address of the host on which the browser is running?

Not presented in the given http request. Need to inspect IP header

(e) What type of browser initiates this message? Why is the browser type needed in an HTTP request message?

Google Chrome, but that isn't obvious because it identifies itself as four different browsers.

The browser type information is needed by the server to send different versions of the same object to different types of browsers.

Tutorial 2 Question 8

(a) A user requests a Web page that consists of some text and three images. For this page, the client will send one request message and receive four response messages.

False

- Each object requires a separate request message specifying the URI of the object.
- TL;DR: one request, one response

HTTP Response Codes

Code	Meaning	Examples
1xx	Information	100 – server agrees to handle client's request
2xx	Success	200 = request succeeded; 204 = no content present
3xx	Redirection	301 = page moved; 304 = cached page still valid
4xx	Client error	403 = forbidden page; 404 = page not found
5xx	Server error	500 = internal server error; 503 try again later 18

HTTP Response

```

HTTP/1.1 200 OK\r\n
Server: nginx\r\n
Content-Type: text/html; charset=UTF-8\r\n
Transfer-Encoding: chunked\r\n
Connection: keep-alive\r\n
Vary: Accept-Encoding\r\n
Cache-Control: max-age=86400, public\r\n
Date: Tue, 23 Jun 2020 12:16:31 GMT\r\n
X-Dynamic-Cache: MISS\r\n
Link: <http://www.voycn.com/article/gaobingfahexinjishu-midengxing-yu-fenbushisuo>; rel="canonical"\r\n
Link: <http://www.voycn.com/article/gaobingfahexinjishu-midengxing-yu-fenbushisuo>; rel="revision"\r\n
X-UA-Compatible: IE=edge\r\n
Content-Language: zh-hans\r\n
X-Content-Type-Options: nosniff\r\n
X-Frame-Options: SAMEORIGIN\r\n
Expires: Sun, 19 Nov 1978 05:00:00 GMT\r\n
Last-Modified: Tue, 23 Jun 2020 12:16:30 GMT\r\n
ETag: W/"1592914590"\r\n
Vary: Cookie\r\n
X-Generator: Drupal 8 (https://www.drupal.org)\r\n
X-Cache: HIT\r\n
Content-Encoding: gzip\r\n
\r\n

```

Tutorial 2 Question 8

(d) The Date: header in the HTTP response message indicates when the object in the response was last modified.

False.

- The Date: header line indicates the time and date when the HTTP response was created and sent by the server.
- The Last-Modified: header line indicates the time and date when the object was created or last modified.

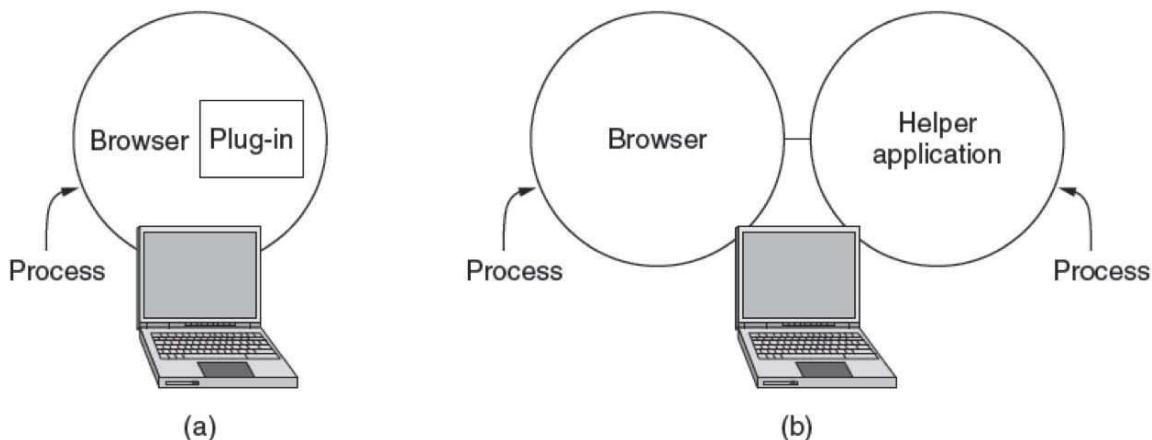
(e) HTTP response messages never have an empty message body.

False.

- For response messages, whether or not a message-body is included with a message is dependent on both the request method and the response status code.
- All responses to the HEAD request method **MUST NOT** include a message-body
- All 1xx (in-formational), 204 (no content), and 304 (not modied) responses **MUST NOT** include a message-body.
- All other responses do include a message-body, although it MAY be of zero length.

HTTP Headers

Client side processing

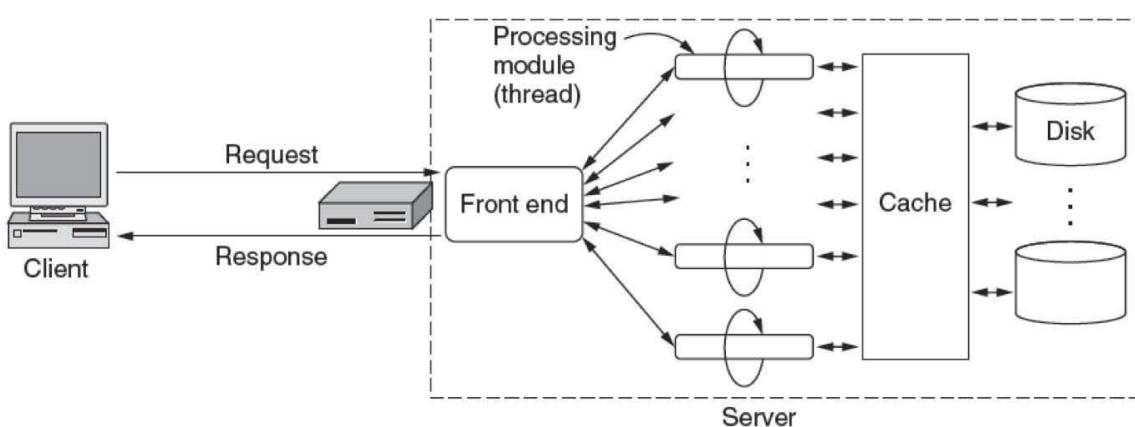


- Plugins/Extensions - integrated software module which executes inside the browser
 - direct access to online context
- Helper - separate program which can be instantiated by the browser, but can only access local cache of file content
 - application/pdf
 - application/msword

Server side processing

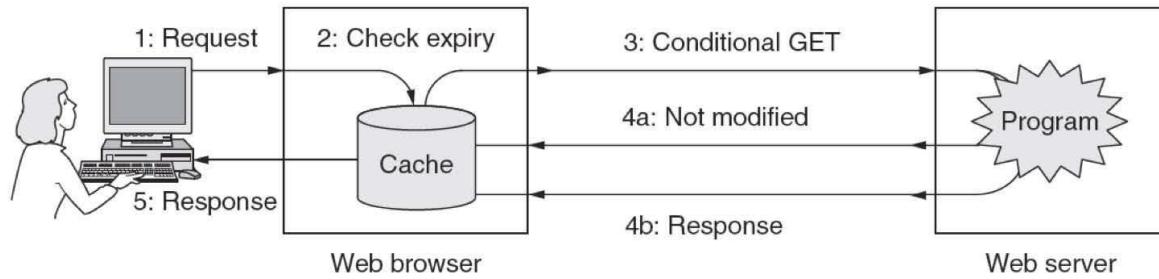
1. Accept TCP Connection from client (browser)
2. Identify the file requested
3. Get the specified file from the local storage (disk, RAM, ...)
4. Send the file to the client
5. Release the TCP connection

Multi-threaded Web Server



- A multithreaded Web server with a front end and processing modules.
- A processing module performs a series of steps:
 1. Resolve name of Web page requested.
 2. Perform access control on the Web page
 3. Check the cache
 4. Fetch requested page from disk or run program
 5. Determine the rest of the response
 6. Return the response to the client
 7. Make an entry in the server log

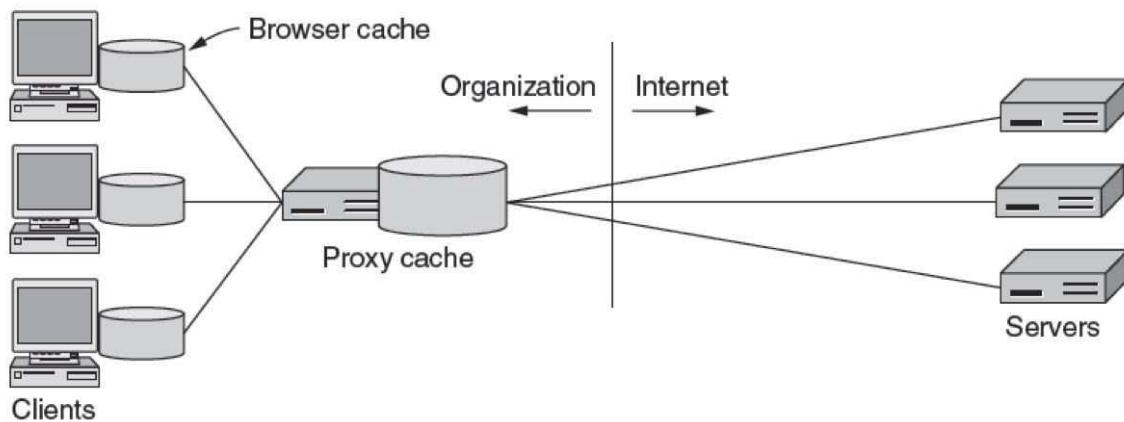
Web Cache



Goal: satisfy client request without involving origin server - reduce response time.

Web proxy

- **Purpose:** used for **caching, security and IP address sharing**
- The browser sends all HTTP requests to the proxy.
 - The proxy returns objects in its cache
 - or the proxy requests object from origin server, then returns object to client.
- Note: the proxy server acts as both client and server.



Cookies

Application	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Priority
prov	68b67a44-a758-f68c-b2c5-fd4aad7e347d	stackoverfl...	/	2020-11-24T1...	40	✓			Medium
prov	cebd18e2-3239-a285-fd62-9f828d25409a	serverfault...	/	2020-12-20T1...	40	✓			Medium
optimizelyEndUserId	c0048944cfba4d1cdcb	.stackoverfl...	/	Session	38				Medium

- HTTP is a stateless protocol
 - The network stores no state about web sessions

- Cookies can place small amount (<4Kb) of information on the users computer and re-use deterministically (RFC 2109)
- **Cookies have 5 fields**
 - domain
 - path
 - content
 - expiry
 - security
- **How to keep state**
 - maintain state at sender/receiver over multiple transactions
 - http messages carry “state”

Tracking

- Questionable mechanism for tracking users and learning about user behaviour
- Tracking companies have expanded beyond simple cookies
 - Plug-in, browser fingerprinting

Consider an e-commerce site that wants to keep a purchase record for each of its customers. Describe how this can be done with cookies.

- When the user first visits the site, the server
 - creates a unique identification number
 - creates an entry in its back-end database
 - returns this identification number as a cookie number.
- This cookie number is stored on the users host and is managed by the browser.
- During each subsequent visit (and purchase), the browser sends the cookie number back to the site.
- Thus the site knows when this user (more precisely, this browser) is visiting the site.

URL/URI

- Uniform Resource Locator
 - Defined in original HTTP specification
 - An address for a resource
 - Can be relative “./nextpage.html” or absolute
[“<http://www.google.com>”](http://www.google.com)
- Separate specification by W3C in 1998 for URI
 - Uniform Resource Identifier
 - `scheme://[user[:password]@]host[:port][/path][?query][#fragment]`
 - `abc://username:password@example.com:123/path/data?key=value#fragid1`

HTML

- a simple mark-up language designed to encode both content and presentational information.
- Plain text encoding, with browser based rendering
- **Components**

- Head ...
- Body
- Syntactically Restricted Tag Sets
- Attributes & Values

Item	HTML 1.0	HTML 2.0	HTML 3.0	HTML 4.0	HTML 5.0
Hyperlinks	X	X	X	X	X
Images	X	X	X	X	X
Lists	X	X	X	X	X
Active maps & images		X	X	X	X
Forms		X	X	X	X
Equations			X	X	X
Toolbars			X	X	X
Tables			X	X	X
Accessibility features				X	X
Object embedding				X	X
Style sheets				X	X
Scripting				X	X
Video and audio					X
Inline vector graphics					X
XML representation					X
Background threads					X
Browser storage					X
Drawing canvas					X

XML & XSL

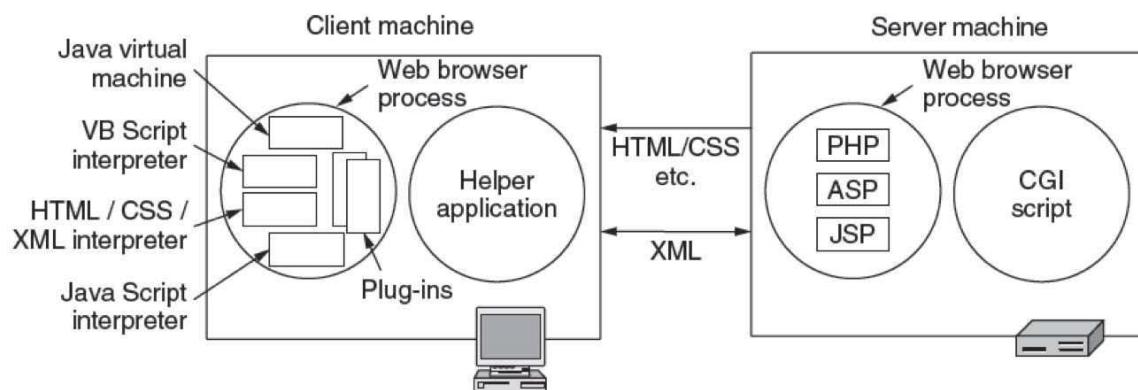
XML (Extensible Markup Language) & XSL (Extensible Stylesheet Language)

- Primary feature: separation of content and presentational markup
- Stringent validation requirements

XHTML

- Essentially an expression of HTML 4.0 as valid XML
- Major differences to HTML 4.0 are the requirements for conformance(一致性), case folding, well-formedness, attribute specification, nesting and embedding, and inclusion of a document type identifier

Dynamic Content



```

<html>
<body>
<form action="action.php" method="post">
<p> Please enter your name: <input type="text" name="name"> </p>
<p> Please enter your age: <input type="text" name="age"> </p>
<input type="submit">
</form>
</body>      (a)
</html>

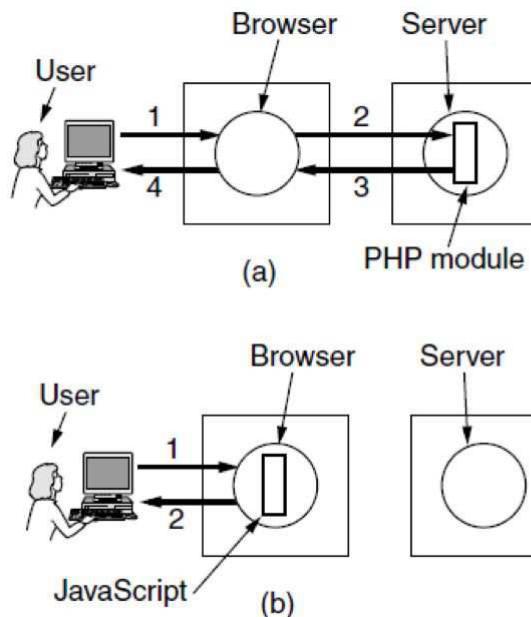
<html>
<body>
<h1> Reply: </h1>
Hello Barbara.
Prediction: next year you will be 33
</body>
</html>          (c)

<html>
<body>
<h1> Reply: </h1>
Hello <?php echo $name; ?>.
Prediction: next year you will be <?php echo $age + 1; ?>
</body>
</html>      (b)

```

![image-20200623233011163](Note.assets/image-20200623233011163.png)

Client-side Scripting



Technologies for producing interactive web applications include:

- JavaScript
- Java Applets - compiled Java code (platform independent)
- ActiveX - compiled code for Windows
- AJAX
 - HTML and CSS: present information as pages.
 - DOM: change parts of pages while they are viewed.
 - XML: let programs exchange data with the server.
 - An asynchronous way to send and retrieve XML data.
 - JavaScript as a language to bind all this together

Domain Name System

- We use DNS to resolve the URL to an absolute IP address

- DNS is the technology behind mapping host.domain.com to an IP address.
- an IP addresses **should uniquely** identify a socket/jack (or wireless interface) on a computer
 - Often not the case today

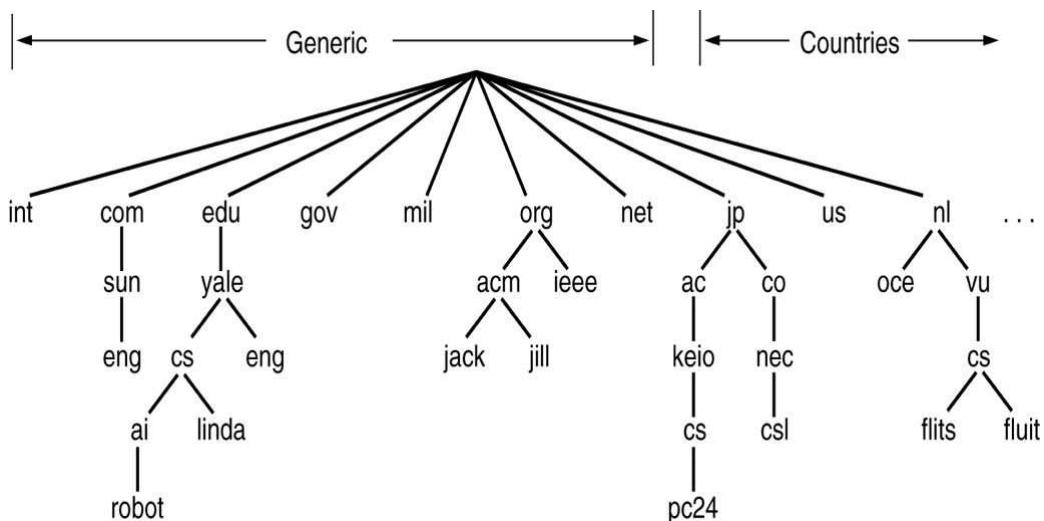
Elements of DNS

- **Domain name space:** DNS uses a tree-structured name space to identify resources on the Internet.
- **DNS database:** Each node/leaf in the name space tree names a set of information that is contained in a resource record (RR). The collection of all RRs is organized into a distributed database.
- **Name servers:** Server programs that hold information about a portion of the domain name tree structure and the associated RRs.
- **Resolvers:** These are programs that extract information from name servers in response to client requests.

Domain name

- are case insensitive
- can have up to 63 characters per constituent
- can have up to 255 chars per path
- can be internationalised (since 1999) – caused security problems
- Naming conventions often follow either organisational or physical boundaries e.g.,
 - au.ibm.com / uk.ibm.com (for email)
 - ibm.com.au / ibm.co.uk (for web)

Conceptual division of DNS Namespace



Top-level domains

Domain	Intended use	Start date	Restricted?
com	Commercial	1985	No
edu	Educational institutions	1985	Yes
gov	Government	1985	Yes
int	International organizations	1988	Yes
mil	Military	1985	Yes
net	Network providers	1985	No
org	Non-profit organizations	1985	No
aero	Air transport	2001	Yes
biz	Businesses	2001	No
coop	Cooperatives	2001	Yes
info	Informational	2002	No
museum	Museums	2002	Yes
name	People	2002	No
pro	Professionals	2002	Yes
cat	Catalan	2005	Yes
jobs	Employment	2005	Yes
mobi	Mobile devices	2005	Yes
tel	Contact details	2005	Yes
travel	Travel industry	2005	Yes
xxx	Sex industry	2010	No

Resource Records

Type	Meaning	Value
SOA	Start of authority	Parameters for this zone
A	IPv4 address of a host	32-Bit integer
AAAA	IPv6 address of a host	128-Bit integer
MX	Mail exchange	Priority, domain willing to accept email
NS	Name server	Name of a server for this domain
CNAME	Canonical name	Domain name
PTR	Pointer	Alias for an IP address
SPF	Sender policy framework	Text encoding of mail sending policy
SRV	Service	Host that provides it
TXT	Text	Descriptive ASCII text

```

; Authoritative data for cs.vu.nl
cs.vu.nl.      86400  IN  SOA   star boss (9527,7200,7200,241920,86400)
cs.vu.nl.      86400  IN  MX    1 zephyr
cs.vu.nl.      86400  IN  MX    2 top
cs.vu.nl.      86400  IN  NS    star

star           86400  IN  A     130.37.56.205
zephyr         86400  IN  A     130.37.20.10
top            86400  IN  A     130.37.20.11
www            86400  IN  CNAME star.cs.vu.nl
ftp             86400  IN  CNAME zephyr.cs.vu.nl

flits          86400  IN  A     130.37.16.112
flits          86400  IN  A     192.31.231.165
flits          86400  IN  MX    1 flits
flits          86400  IN  MX    2 zephyr
flits          86400  IN  MX    3 top

rowboat        IN  A     130.37.56.201
                IN  MX   1 rowboat
                IN  MX   2 zephyr

little-sister  IN  A     130.37.62.23

laserjet       IN  A     192.31.231.216

```

Tutorial 3 Question 5

Is it possible for an organization's Web server and mail server to have exactly the same alias for a hostname (for example, foo.com)? How does a DNS server distinguish these two types of mapping, namely, hostname-web server IP and hostname-mail server IP?

- Yes an organization's mail server and Web server can have the same alias for a host name.
- Instead of simply storing the (hostname, IP) mappings, DNS servers store a four tuple called resource record (RR): (Name, Value, Type, TTL) in their database. We ignore the old TTL here. Depending on the Type field, the tuple can have different meanings:
 - A:
 - Name is a hostname
 - Value is the IP address for the hostname.
 - Thus, a Type A record provides the standard hostname-to-IP address mapping. As an example, (relay1.bar.foo.com, 145.37.93.126, A) is a Type A record.
 - NS:
 - Name is a domain (such as foo.com) and
 - Value is the hostname of an authoritative DNS server that knows how to obtain the IP addresses for hosts in the domain.
 - This record is used to route DNS queries further along in the query chain. As an example, (foo.com, dns.foo.com, NS) is a Type NS record.
 - CNAME:
 - Value is a canonical hostname for the alias hostname Name.
 - This record can provide querying hosts the canonical name for a hostname. As an example, (foo.com, relay1.bar.foo.com, CNAME) is a CNAME record.
 - MX:
 - Value is the canonical name of a mail server that has an alias hostname Name. As an example, (foo.com, mail.bar.foo.com, MX) is an MX record.

- MX records allow the hostnames of mail servers to have simple aliases.
- Note that by using the MX record, a company can have the same aliased name for its mail server and for one of its other servers (such as its Web server). To obtain the canonical name for the mail server, a DNS client would query for an MX record; to obtain the canonical name for the other server, the DNS client would query for the CNAME record.

Inserting records into DNS

- Provide names, IP addresses of authoritative name server (primary and secondary)
- registrar inserts two RRs into com TLD server:
 - (networkkutopia.com, dns1.networkkutopia.com, NS)
 - (dns1.networkkutopia.com, 212.212.212.1, A)
- create authoritative server
 - Type A record for www.networkkutopia.com;
 - Type MX record for networkkutopia.com

Name server zones

Zones

- DNS namespace is divided into overlapping zones.
- The name servers are authoritative for that zone.
 - usually two name servers for a zone
- Name servers are arranged hierarchically from a set of root servers

Root name servers

- The root servers form the authoritative cluster for enquiries. The root servers are contacted by a local name server that can not resolve name.

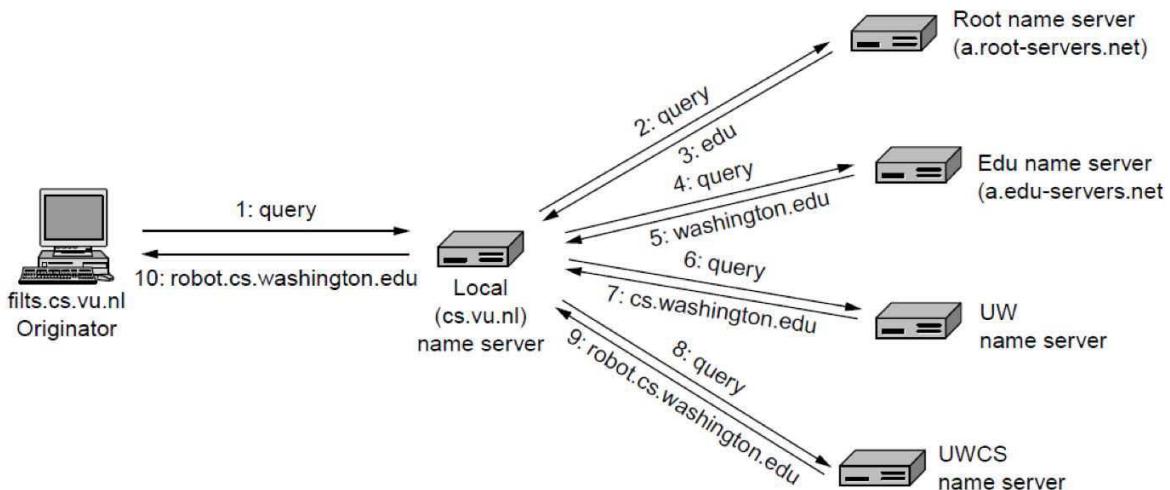
Types of name servers

- **Top-level domain DNS servers**
 - responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, au, jp
- **Authoritative DNS servers:**
 - organizations DNS servers, **providing authoritative hostname to IP mappings for organizations servers** (e.g., Web, mail).
 - Can be maintained by organization or service provider.
- **Local DNS server**(not strictly part of the hierarchy)
 - Typically, each ISP (residential ISP, company, university) has a "default name server" which handles DNS queries
 - a query is sent to its local DNS server acts as proxy, forwards query into hierarchy

Resolving a query

- A **resolver** client asks the local DNS for the domain to IP mapping
 - if answer is known by the local DNS, then it sends the answer.
 - if answer is not known, then the local DNS queries up the hierarchy to the top level (root) DNS for the domain and then relays the answer to the resolver client.

- Essentially, this is a recursive query mode. Queries are subject to timers to avoid longer than necessary response times.



DNS Security

- No security in original design
 - DNS spoofing
 - DNS** cache poisoning is the act of entering false information into a DNS cache, so that DNS queries return an incorrect response and users are directed to the wrong websites.
 - DNS flooding
 - A DNS flood is a type of [distributed denial-of-service attack \(DDoS\)](#) where an attacker floods a particular domain's DNS servers in an attempt to disrupt DNS resolution for that [domain](#).
- Solutions
 - DNSSEC
 - DNSSEC strengthens authentication in DNS using *digital signatures* based on *public key cryptography*.
 - With DNSSEC, it's not DNS queries and responses themselves that are cryptographically signed, but rather DNS data itself is signed by the owner of the data.
 - Root signing

Tutorial 3 Question 4

Consider a situation in which a cyberterrorist makes all the DNS servers in the world crash simultaneously. How does this change people's ability to use the Internet?

- The DNS servers provide a *mapping between domain names and IP addresses*, such that when a request for a Web page is received, the browser can look up in the DNS server the IP address corresponding to the domain name of the requested page, and then download the requested page from that IP address.
- If all the DNS servers in the world were to crash at the same time, nobody would be able to map between domain names and IP addresses.
- Therefore, the only way to access Web pages would be by using the IP address of the host server instead of the domain name.

- Since most of us do not know the IP addresses of the servers we access, this type of situation would make use of the Internet virtually impossible for most users.

Tutorial 3 Question 6

DNS uses UDP instead of TCP. If a DNS packet is lost, there is no automatic recovery.

Does

this cause a problem, and if so, how is it solved?

- DNS is idempotent.
- Operations can be repeated without harm.
- When a process makes a DNS request, it starts a timer. If the timer expires, it just makes the request again. No harm is done.

Email

User agents

- allow user to read and send email
- Basic functions: compose, report, display, dispose
- Envelope and contents: encapsulation of transport related information
- Header and body:
 - **header** user agent control info
 - **body** for human recipient
- User must provide message, destination, optional other parameters
- Addressing scheme user@dns-address

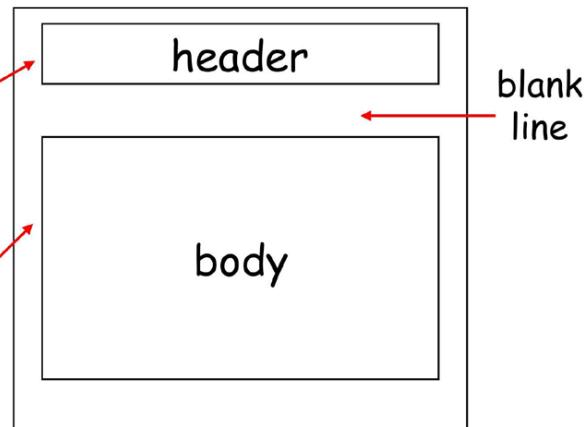
Message transfer agents: transport messages from source to destination

Format

SMTP: protocol for exchanging
email messages

RFC 822: standard for text
message format:

- header lines, e.g.,
 - To:
 - From:
 - Subject:
- body
 - the “message”, ASCII characters only



Message Header Fields

Headers the user usually sees	To:	Cc:	Bcc:
	From:	Date:	Subject:
	Message-Id:	In-Reply-To:	References:
	Reply-To:	Sender:	Return-Path:

What are they for?

SMTP

```

S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection

```

- SMTP uses **TCP** to reliably transfer email message from client to server, port 25
- three phases of transfer
 - handshaking (greeting),
 - transfer of messages
 - closure
- command/response interaction: commands in ASCII text and response consists of status code and phrase
- messages **must** be in 7-bit ASCII

MIME

Type	Example subtypes	Description
text	plain, html, xml, css	Text in various formats
image	gif, jpeg, tiff	Pictures
audio	basic, mpeg, mp4	Sounds
video	mpeg, mp4, quicktime	Movies
model	vrml	3D model
application	octet-stream, pdf, javascript, zip	Data produced by applications
message	http, rfc822	Encapsulated message
multipart	mixed, alternative, parallel, digest	Combination of multiple types

MIME has 5 additional message headers

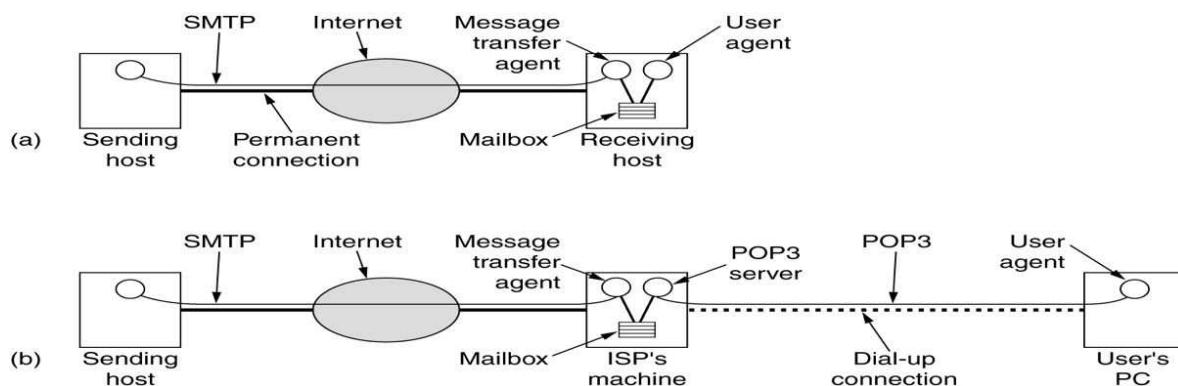
- **MIME-Version:** identifies the MIME version
- **Content-Description:** human readable describing contents
- **Content-Id:** unique identifier
- **Content-Transfer-Encoding:** how body is wrapped for transmission
- **Content-Type:** type and format of content

Transfer

- SMTP: delivery/storage to receiver's server
- SMTP is a '**push**' protocol so you cannot pull emails off the server using SMTP

Delivery

- **Local**
- **POP3** : Post Office Protocol; authorization (agent – server) and download
- **IMAP**
 - Internet Mail Access Protocol
 - more features (more complex);
 - provides for the manipulation of stored messages on server
- **HTTP**: gmail, Hotmail, Yahoo! Mail, etc.



(a) Sending and reading mail when the receiver has a permanent Internet connection and the user agent runs on the same machine as the message transfer agent (now rare).

(b) Current case: notebook/PC/phone is not an MTA.

POP3 – Post Office Protocol

- Three states of a POP3 transaction
 - Authorisation
 - Transactions
 - Update
- Syntax
 - USER / PASS
 - LIST
 - RETR / DELE
 - QUIT (update)
- Issue: "download and delete" mode does not allow messages to be re-read.

IMAP – Internet Message Access Protocol

- IMAP keeps user state across sessions

- Retain mailbox contents online (server) and allow manipulation of online and offline messages and mailbox folders
- Implications of server infrastructure to support high volume of IMAP users. This implies storage projections by the provider, and hence limitations.

Tutorial Week 3 Question 2

Describe the commonalities and differences between SMTP and (persistent) HTTP.

Commonalities:

- Both of them are protocols used for transferring files between two sides.
- Both of them are application layer protocol
- Both of them use persistent TCP connections.

Differences:

- SMTP is a push protocol, while HTTP is a pull protocol.
- SMTP requires all the contents in message to be in 7-bit ASCII format, while HTTP has no such a restriction.
- HTTP sends objects in a "one-in-one" manner, namely, each object is sent via a message. However, SMTP sends objects in a "all-in-one" manner, i.e., sending all the objects in one message.

Tutorial Week 3 Question 3

Suppose that John just set up an auto-forwarding mechanism on his work email address, which receives all of his business-related emails, to forward them to his personal email address, which he shares with his wife. John's wife was unaware of this, and activated a vacation agent on their personal account. Because John forwarded his email, he did not set up a vacation daemon on his work machine. What happens when an email is received at Johns work email address?

- Each message received in Johns work email inbox will be forwarded to his personal inbox, thereby generating an autoreply by the vacation agent, sent to his work inbox.
- This reply will be seen by the work computer as a new message, and thus be forwarded to the personal mailbox.
- Most vacation agents would note that this has the same From: address as the previous, and end the process there. However, the agent may be configured to reply to each new email.
- In this case it will send another reply to the work inbox. As a result there will be an endless string of messages for each message received in Johns work email address (unless the vacation agent is smart enough to reply just once to each sender it sees).

Presentation Layer

- OSI layer 6 to provide
 - Encryption
 - Compression

- Data conversion (e.g., mapping CR/LF to LF, .doc to .docx)
- Mapping between character sets (ASCII/EBCDIC, now UTF-8/BIG5/...)
- These services haven't vanished: done by applications in TCP/IP model
 - Why does IETF consider them "Application Layer"?
 - The protocol to negotiate encryption etc. is quite simple and separate from the algorithms
 - There aren't simple "common services" needed by all applications
 - The application is not in the kernel, and so much more flexible
 - "Layering violations"
 - Closest thing to presentation layer: Real time protocol (RTP)

Session Layer

OSI Layer 5 to provide

- Authentication
- Authorization
- Session restoration
 - continue a failed download
 - log back in to same point in an online purchase
- Examples:
 - Remote procedure call (RPC)
 - Point-to-point tunneling protocol (PPTP)
 - Password (/Extensible) Authentication Protocol (PAP/EAP)
- Often used between protocols called **layer 2** and **layer 3**
 - Layers are funny.
 - Ethernet is always called "layer 2", but has many properties of layer 3, and even some of layer 4.

Transport layer

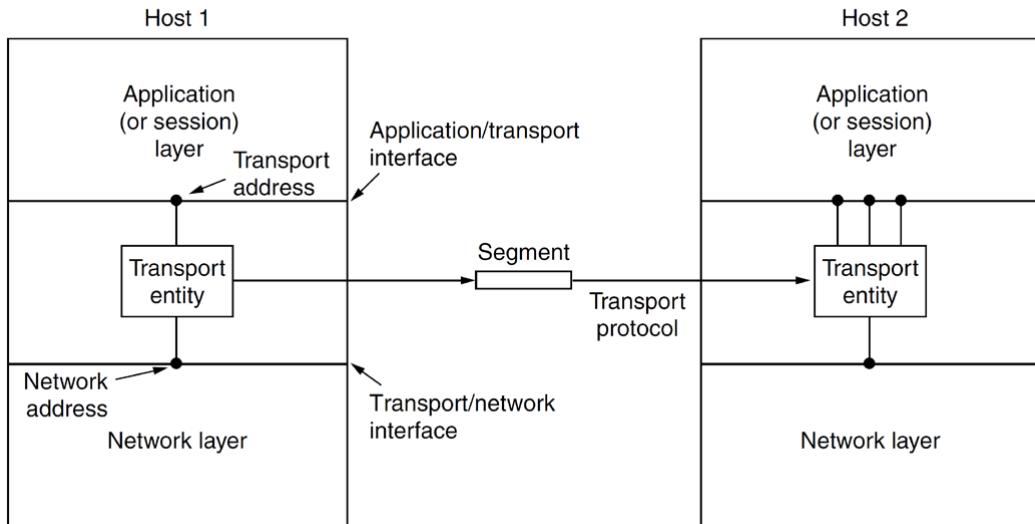
Role

- provide services needed by applications, using services available by the network layer.
- Application needs:
 - **Stream:** Data is a stream of bytes
 - **Multiplexing:** Data from one application is not mixed with those for another
 - **Reliability:** Data arrives reliably (or we know when a packet has been lost)
 - **In-order delivery:** Data arrives in order
 - **Rate Control:** Data doesn't arrive faster than we can handle
- Network provides
 - Get packets from host to host most of the time sometimes multiple copies

Services

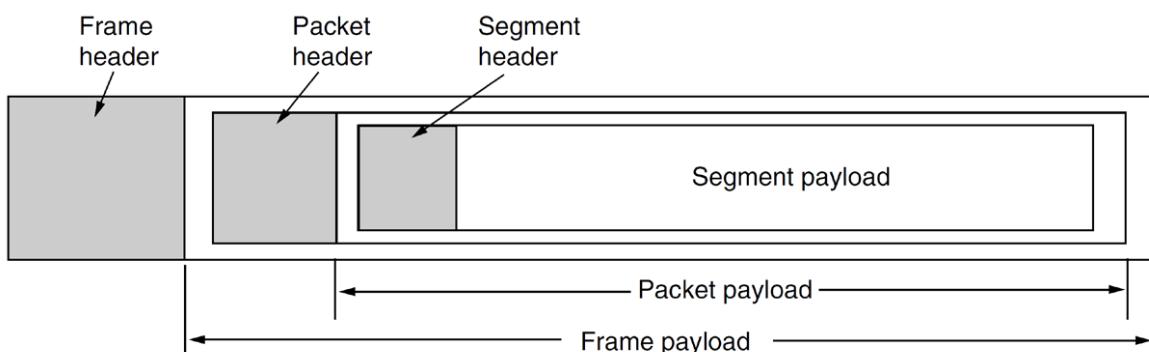
- The Transport layer services provide interfaces between the **Application layer** and the **Network/Internet layer**.

- **Location:** The Transport layer entities (the hardware or software which actually does the work e.g. OS kernel, processes, NIC) can exist in multiple locations.



- **Transport layer provides a “logical” communication channel between processes running on different hosts**
 - Connection-oriented
 - Connection establishment, data transfer, connection release (TCP)
 - **Does Provide** a reliable service on top of an unreliable network to the application layer
 - **Doesn’t** provide privacy, isochrony (preserving delay between packets)
 - Hides acknowledgements, congestion control, lost packets
 - Connectionless: data transfer (UDP)
 - Provides multiplexing between different processes

Encapsulation



- Abstract representation of messages sent to and from transport entities
- Encapsulation of segments (transport layer units) in packets (network/internet layer units) in frames (data/link layer units)

Packet	Meaning
Message	sent at the application layer
Segments	sent at the transport layer
Packets	sent at the internet/network layer
Frames	sent at the link/data link layer

Addressing

- Addressing in the Transport layer is typically done using **port numbers**
- Full address is a 5-tuple
 - source IP address
 - source port
 - destination IP address
 - destination port
 - protocol)

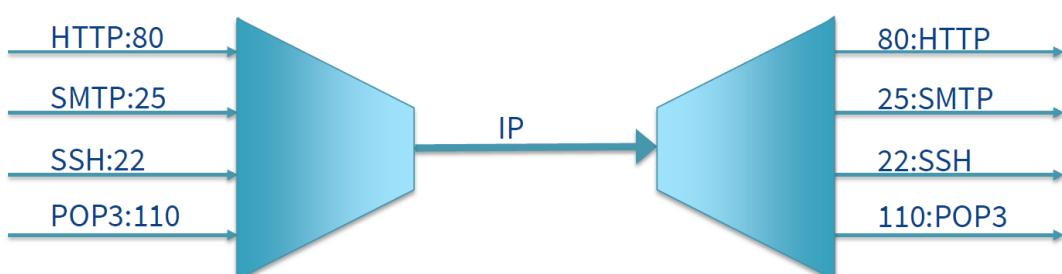
Port allocations

- Range:** 0-65535 (16 bits)
- Allocated by:** Internet Assigned Numbers Authority (IANA)
- classified into 3 segments:**
 - Well Known Ports (0-1023)
 - 21 FTP
 - 22 SSH
 - 23 Telnet
 - 25 SMTP
 - 80 HTTP
 - 110 POP3
 - 119 NNTP
 - Registered Ports (1024-49151)
 - Also called “user ports” but still registered with IANA or similar body
 - Dynamic Ports (49152-65535)

Multiplexing /Demultiplexing

- Multiplexing(MUXING):** combining multiple distinct streams into a single shared stream
- Demultiplexing(DEMUXING):** splitting distinct streams out from a single shared stream

![image-20200624014246763](Note.assets/image-20200624014246763.png)



UDP – User Datagram Protocol

Connectionless

The User Datagram Protocol provides a protocol whereby applications can transmit encapsulated IP datagrams **without** establishing a connection.

Strengths

- Simple and efficient
- provide multiplexing/de-multiplexing
- no delay waiting to recover lost packets
- **Advantage over raw IP:** the ability to specify ports for source and destination pairs.

Weaknesses

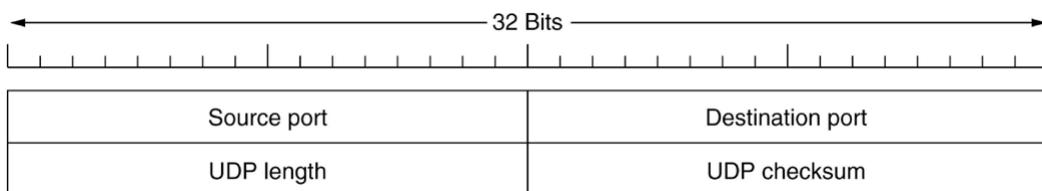
- No flow control
- No error control
- No retransmission of bad segments

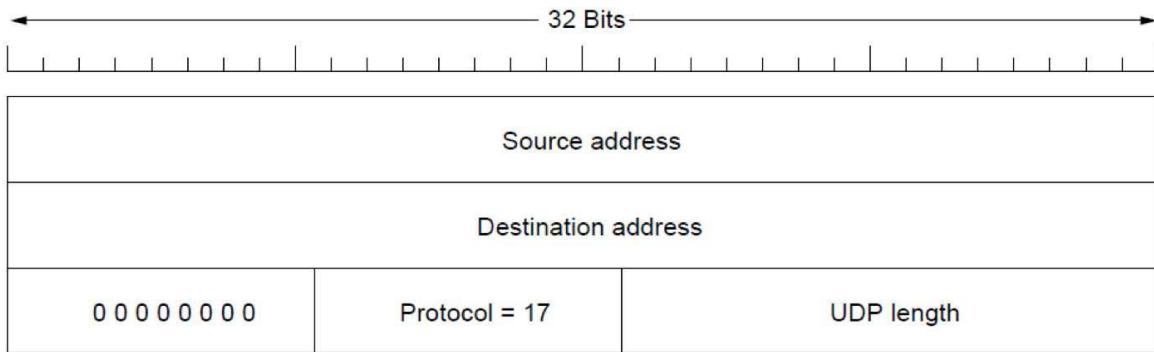
Applications

- Suitable for applications require a precise level of control over packet flow/error/timing
- Suitable for **timeout and resend** client – server settings
 - Clients sends a short request to the server, expects a short response
 - If that does not occur (request or response is lost) client timeouts and resends
 - Simple to code, and fewer messages, one in each direction
 - DNS is a good example
- Suitable for real-time services (e.g., VoIP)
 - If a packet is lost, we don't want to wait for it to be resent
 - **Loss concealment**(隐藏): fill in the time with our “best guess” sound

UDP header

- UDP transmits in segments consisting of a header followed by the payload
- UDP headers contain **source** and **destination** ports, payload is handed to the process which is attached to the particular port at the destination (using BIND primitive or similar)
 - **destination port** allows initial routing for incoming segments
 - **source port** allows reply routing for outgoing segments.





UDP Question

Tutorial Week 4 Question 1

Suppose a process in Host C has a UDP socket with port number 6789. Suppose both Host

A and Host B each send a UDP segment to Host C with destination port number 6789.

- Will both of these segments be directed to the same socket at Host C?
- Yes, both segments will be directed to the same socket.
- This is because UDP receivers only use listening sockets, which are identified by a **3-tuple (protocol, local IP address, local port)**, instead of the 5-tuple used for a connection-oriented socket.
- If so, how will the process at Host C know that these two segments originated from two different hosts?
- For each received segment, at the socket interface, the operating system will provide the process with the sender's IP address and port number to determine the origins of the individual segments.
- If the sender also has a UDP listening socket on its sending port, then the original receiver can send it a reply by sending a UDP packet with the original sender's IP address and port.

Tutorial Week 4 Question 2

Is it possible for an application to enjoy reliable data transfer even when the application runs over UDP? If so, how?

- Yes.
- The application can implement its own retransmission scheme (transport layer functionality) on top of UDP. In this context, the application would be treating UDP as a network-layer protocol.
- Example: VLAN There are even 'virtual wires', in which IP packets carry bit streams, which represent actual ethernet frames.

Tutorial Week 4 Question 4

Why does UDP exist? Would it not have been enough to just let the user processes send raw IP packets?

- UDP provides multiplexing service to upper layer.
- IP packets contain IP addresses, which specify a destination machine.
 - Once such a packet arrived, how would the network handler know which process to give it to?

- UDP packets contain a destination port number. This information is essential so they can be delivered to the correct process.
- UDP is sometimes called a "process-to-process", as distinct from a "host-to-host" protocol like IP.

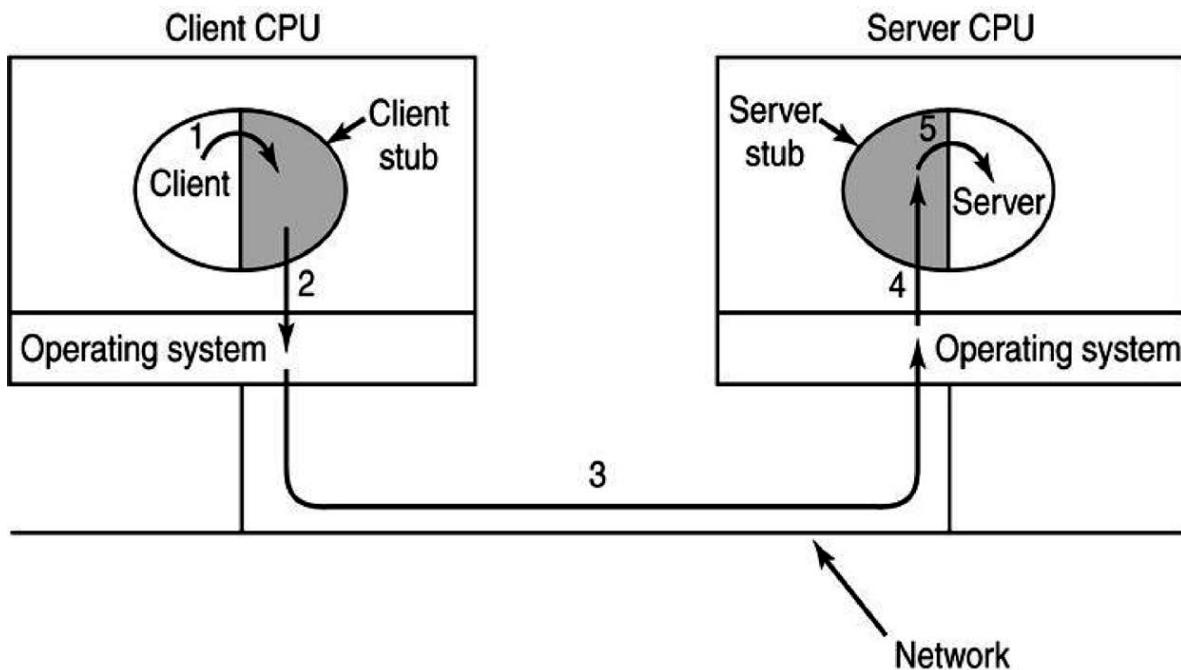
RPC - Remote Procedure Calls

Remote Procedure Calls

- Allow calling procedures on a remote server as if they are local to the client
- Hides the networking aspects from the programmer (proxy pattern haha)
 - To hide the networking, the client and server must be bound to respective stubs
 - Client stub – operates in the client address space
 - Server stub – operates in the server address space
 - From the perspective of the client and server processes all the calls are local
- Parameters can be passed and returned
 - **Marshalling** – convert the in-memory data structure to a form that can be stored or transmitted
 - **Unmarshalling** – convert the stored or transmitted data into an in-memory data structure
- RPC isn't a single protocol/API. Dozens of variants exist gRPC, Dubbo

How RPC works?

1. Client process on Machine A calls procedure on Machine B
2. Process on machine A is suspended, whilst execution of the procedure takes place on Machine B
3. Machine B responds with result to Machine A, which then continues processing



Challenges

- Cannot pass pointers easily

- client and server are in different address spaces
- Possible to marshal and unmarshal underlying value and create a pointer in each address space, does not work for complex data structures
- Weakly typed languages like C can present problems, e.g. unknown array sizes
- Unable to deduce parameter types
- Global variables are not shared

UDP vs TCP for RPC

- **RPC over TCP**
 - TCP can be used for non-idempotent operations
- **RPC over UDP:**
 - Resending after timeout if no reply is received
 - a reply constitutes an acknowledgement of the request
 - Handling large parameter sizes that need to be split across multiple UDP segments
 - Caution must be used if operation is not idempotent(e.g., incrementing a bank balance)

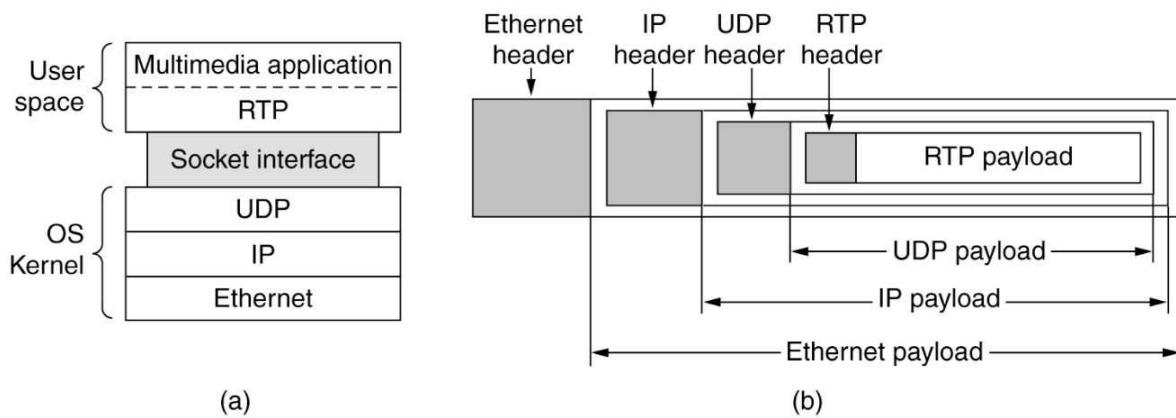
RTP – Real-Time Transport Protocol (RTP)

- RTP multiplexes several streams into a single stream of UDP segments



Which layer is RTP at?

- RTP typically runs on top of UDP.
- At the receiver side, the application receives the RTP packet from its socket interface.
- The application extracts the audio chunk from the RTP packet and uses the header fields of the RTP packet to properly decode and play back the audio chunk.

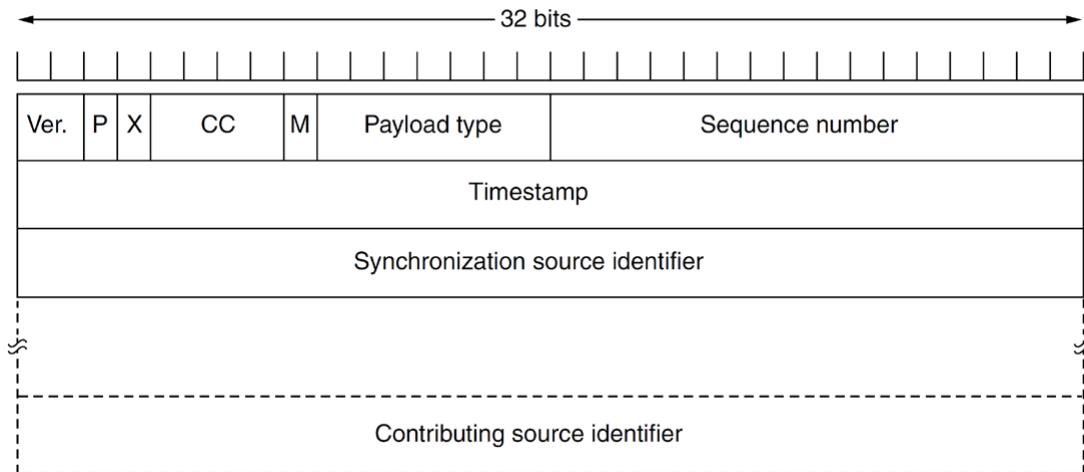


(a) The position of real-time protocol in the protocol stack

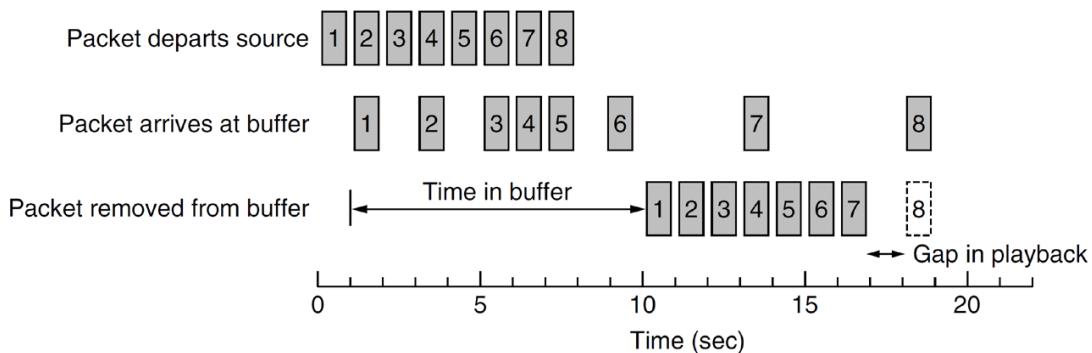
(b) Packet nesting

RTP Header

- Payload type – encoding used (MP3, etc.) – can vary each time
- Sequence Number – counter incremented on each packet
- Timestamp – Source controlled relative to start of the stream



RTP Playback



- Jitter – variation in delay of packets
 - Buffer at receiver to counter it
- Packet 8 too late, can wait or skip, depending on application
- Size of buffer is also application specific (VOIP = small buffer)

RTCP - Real-time Transport Control Protocol

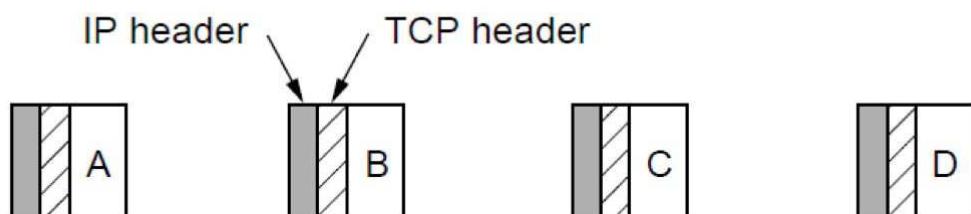
- Control protocol for RTP
 - Handles feedback, synchronization, and UI
- Feedback to source
 - Delay, jitter, bandwidth, congestion
 - Used by encoder to adaptively encode to suit network conditions
 - In multicast settings, feedback is limited to small percentage of media bandwidth
- Synchronization
 - Where different streams use different clocks/have different drift
- UI

- naming sources to show who is on a conference call

TCP - Transmission Control Protocol

TCP Services

- The Transmission Control Protocol provides a protocol by which applications can transmit IP datagrams within a connection-oriented framework, thus increasing reliability.
 - TCP transport entity manages TCP streams and interfaces to the IP layer
 - TCP entity accepts user data streams, and segments them into pieces
 - sends each piece as a separate IP datagram
 - Recipient TCP entities reconstruct the original byte streams from the encapsulation
-



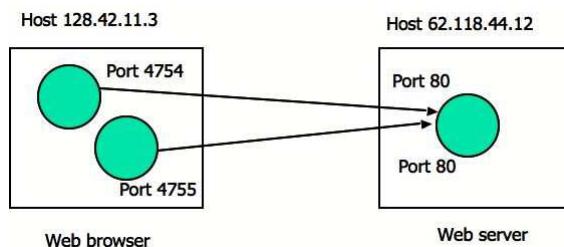
(a)



(b)

Service Model

- The sender and receiver both create sockets
 - A kernel data structure, named by the 5-tuple of IP address and port number of sender and receiver, and the protocol
 - For TCP service to be activated, connections must be explicitly established between a socket at a sending host (src-host, src-port) and a socket at a receiving host (dest-host, dest-port)
-



TL;DR

TCP provides

- Steam oriented :
 - Data is sent and received as a stream of bytes
 - message boundaries are not preserved

- **Multiplexing**
 - TCP does multiplexing and de-multiplexing at the sender and receiver ends respectively as a number of logical connections can be established between port numbers over a physical connection.
- **Reliability**
 - The delivery of the packet, i.e., no packet loss
 - TCP attempts to recover lost or corrupted packets by using re-transmission, acknowledgement policy and timers.
 - the integrity of the packet
 - uses checksum for error detection
 - corrupted packets will be retransmitted
 - It uses Acknowledge number, sequence number to ensure reliability
- **Congestion control**
- **Flow control:** limits the rate a sender transfers data to guarantee reliable delivery.
- **Ordered data transfer**
 - Guaranteed **in-order** delivery of data to the receiver
 - segments will arrive in order by the help of sequence number

TCP Segment

- TCP pairs each chunk of client data with a TCP header, thereby forming TCP segments.
- Data is exchanged between TCP entities in segments
 - each has a 20– 60 byte header, plus zero or more data bytes
- **Segment size:** TCP entities decide how large segments should be, given two constraints
 - IP payload < 65,515 byte
 - Maximum Transfer Unit (MTU) - generally 1500 bytes

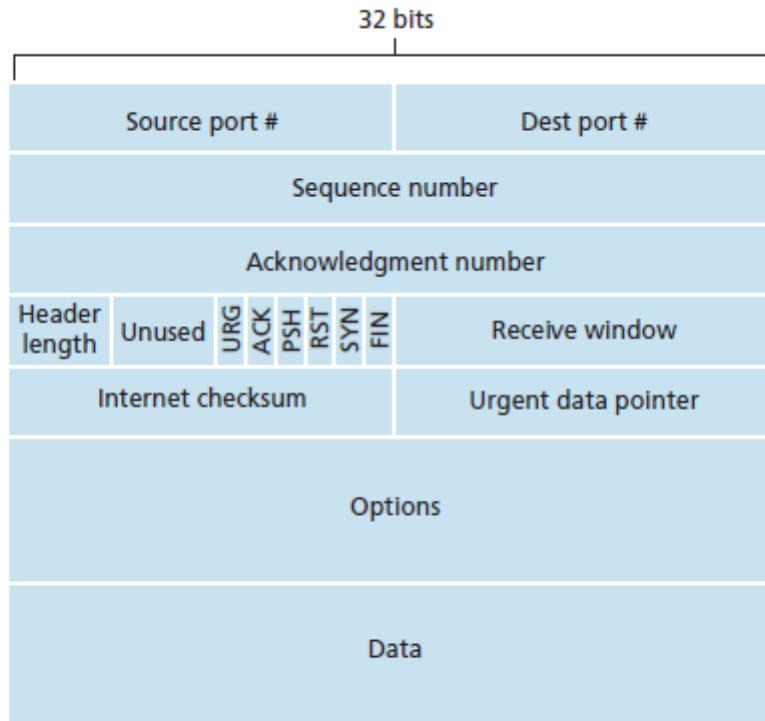


Figure 3.29 ♦ TCP segment structure

Name	Description
Source port	Sending port
Destination port	Receiving port
Sequence Number	If SYN =1: initial sequence number if SYN =0: is accumulated sequence number of the first data byte of this segment
Acknowledgement number	If ACK =1: next sequence number that the sender of the ACK is expecting
Data offset	Size of the TCP Header (20-60 bytes)
Flags	Single bit flags (SYN , ACK , RST , FIN , etc.)
Window size	Size of receive window – how much data the sender of this segment is willing to receive

TCP Primitives

core functions which allow interface with transport services (in particular TCP)

Primitive	Packet	Meaning
LISTEN	None	Block until something tries to connect
CONNECT	CONNECTION REQ	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	None	Block until DATA packet arrives
DISCONNECT	DISCONNECTION REQ	This sides wants to release the connection

TCP Connection

Features

- **Full duplex:** data in both directions simultaneously
- **End to end:** exact pairs of senders and receivers
- **Byte streams, not message streams:** message boundaries are not preserved
- **Buffer capable:**
 - TCP entity can choose to buffer prior to sending or not depending on the context
 - **PUSH:** indicates a transmission is not to be delayed, and should interrupt the receiving application
 - **URGENT:** indicates that transmission should be sent immediately, and that the receiver should send it to the application out-of-band

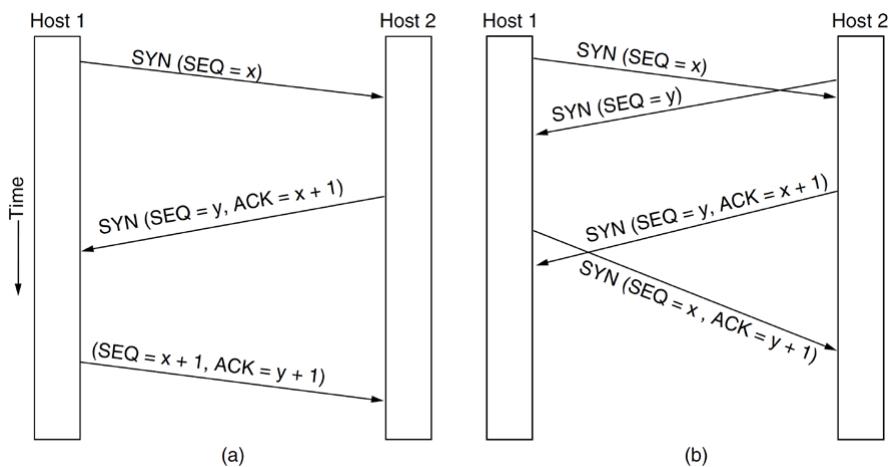
Issues

- TCP is a connection orientated protocol running over a **connectionless** network layer(IP)
- When networks can lose, store and duplicate packets, connection establishment can be complicated
 - **delay:** congested networks may delay acknowledgements
 - **duplicate:** incurring repeated multiple transmissions
 - any of which may not arrive at all or out of sequence – delayed duplicates

Connection establishment

Three-way handshake

- **Goals**
 - Ensure one (and **only one**) connection is established, even if some set-up packets get lost
 - Establish initial sequence numbers for sliding window
- Three-way handshake
 - avoids problems that can occur when both sides allocate **same sequence numbers** by accident
 - Sender and receivers exchange information about which sequencing strategy each will use, and agree on it before transmitting segments



- (a) Normal operation

- The first two segments carry no payload, that is, no application-layer data;
- the third of these segments may carry a payload.
- (b) Simultaneous connection attempts
 - Two simultaneous connection attempts results in only one connection
 - Basically, a TCP receiving duplicate SYNs will SYN,ACK them, but it will receive a RST for the duplicate.
- At end, Host 1 and Host 2 have agreed on respective sequence numbers

Synchronisation

- `SYN` is used for synchronization during connection establishment
 - Sending `SYN` or `FIN` causes sequence number to be incremented by 1
- `SYN` is used in both `CONNECTION_REQUEST` and `CONNECTION_ACCEPTED`, `ACK` bit distinguishes between the two
 - Connection request has `SYN=1, ACK=0`
 - Connection reply has `SYN=1, ACK=1`
- **Sequence Number**
 - first byte of this segments payload
 - 1 + data sent prior to this segment
 - Offset by a random number
 - initial value is arbitrary
 - offset will be reflected in both Sequence and Acknowledgement numbers
- **Acknowledgement Number**
 - next byte the sender expects to receive
 - data successfully received + 1
 - Bytes received without gaps
 - a missing segment will stop this incrementing, even if later segments have been received

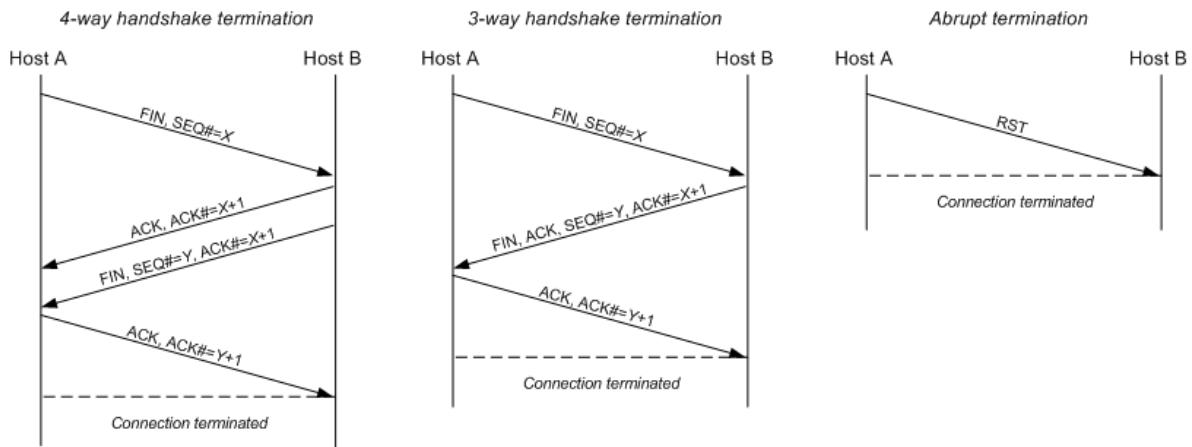
Retransmission

- RTO(retransmission timer)
 - Initialised with a default value and updated based on network performance
 - If the timer expires before an ACK is received the segment is resent
- Fast retransmission(3 DupACKs)
 - Receiver receives segment with a sequence number higher than expected (i.e. segment has been lost)
 - Receiver sends ACK with sequence number it is expecting
 - This is a duplicate of the previously sent acknowledgement (DupACK)
 - After receiving 3 DupACKs the sender resends the lost segment

Closing

- The `FIN` flag is used to signify a request to close a connection
- Each FIN is **directional**, once acknowledged no further data can be sent from the sender to the receiver
 - Data can continue to flow in the other direction

- Sender of FIN will **STILL** retransmit unacknowledged segments
- Typically requires 4 segments to close, 1 FIN and 1 ACK for each direction
 - **Can be optimised to 3 segments**
 - Host A send FIN request
 - Host B responds with ACK of Host A FIN request, and sends FIN request of its own
 - Host A send ACK of Host B's FIN request and connection is closed
- The **RST** flag is used to signify a hard close of a connection
 - Basically states the sender is closing the connection and will not listen for any further messages
 - **RST** is sent in response to receiving a packet for a closed socket.
 - Can be used to close a connection, but **FIN** is greatly preferred because **it is an orderly shutdown of the connection**, as opposed to a reset



SYN Flooding

- **Why:** To establish a connection, the server needs to remember an initial Sequence number for each received SYN request
- **How:** An attacker would make initial SYN requests then not send the appropriate ACK, causing the server to gradually fill up its queue with sequence numbers for now defunct connections
- **Solution**
 - Rather than store the sequence number it is derived from connection information and a timer that creates a stateless SYN queue using cryptographic hashing
 - Incurs performance cost in validating SYN Cookies, but preferable to being unresponsive
 - typically only enabled when under attack

TCP Sliding Window

- **Window**
 - Sender and receiver maintain buffers to send and receive data independently of the application
 - No guarantee that data is immediately sent or read from the respective buffers. i.e. data can be buffered
 - Other windows are maintained for congestion control

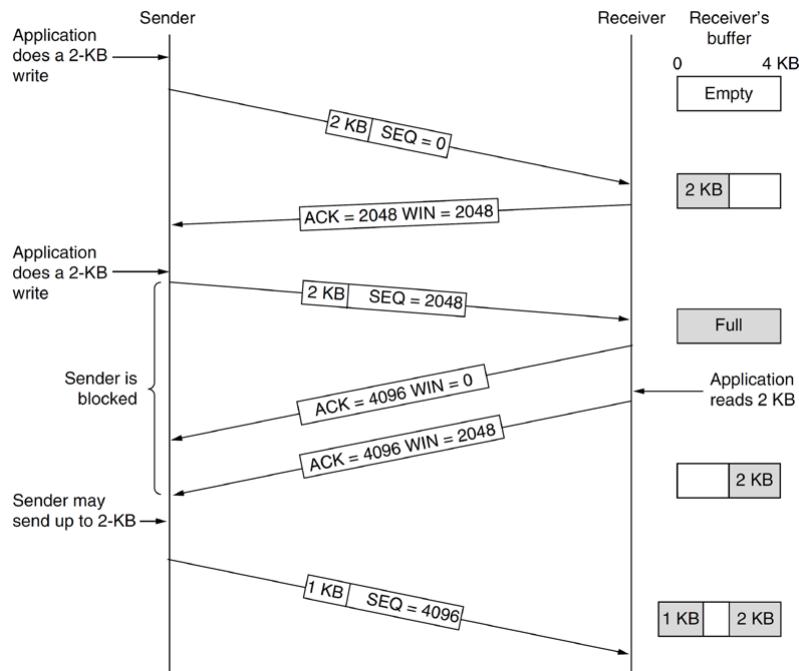
Send window

- What data the sender is able to send

- unacknowledged segments and unsent data that will fit into the receive window

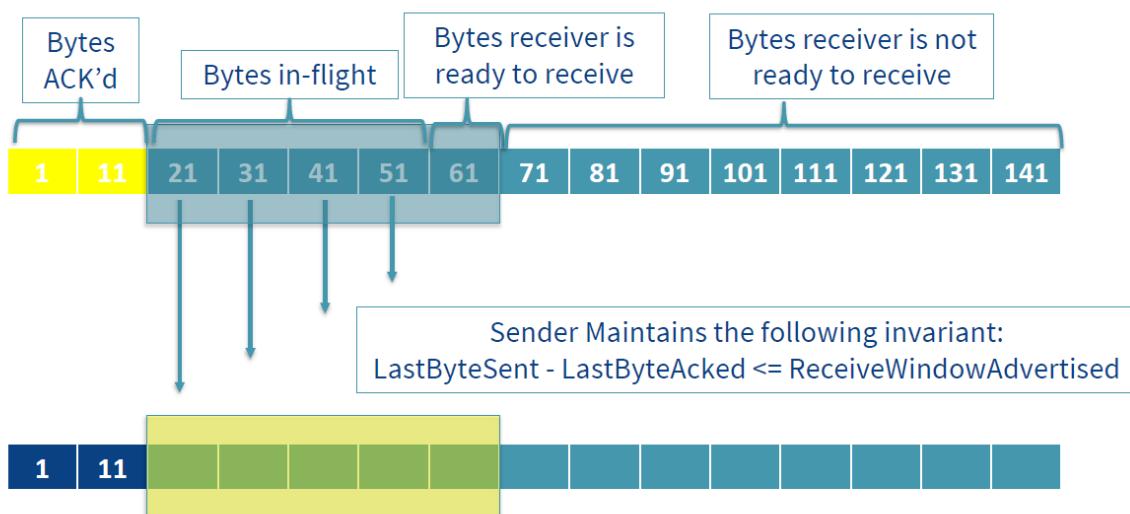
Receive window

- **controlled by receiver:** Sliding window is controlled by receiver
- Determines **amount of data the receiver is able to accept**
 - window size in ACK
- **Zero Receive Window:** When the window is 0 the sender should not send any data
 - the sender can send URGENT data
 - the sender can send “**zero window probe**”
 - 0 byte segment that causes the receiver to re-announce the next expected byte and window size (window probe) this is designed to prevent deadlock
- Senders may delay sending data, e.g. instead of sending the 2KB immediately, could wait for a further 2KB to fill the 4KB receive window
- Sender Maintains the following invariant: `LastByteSent - LastByteAcked <= ReceivewindowAdvertised`





TCP Sliding Window



Potential for deadlock

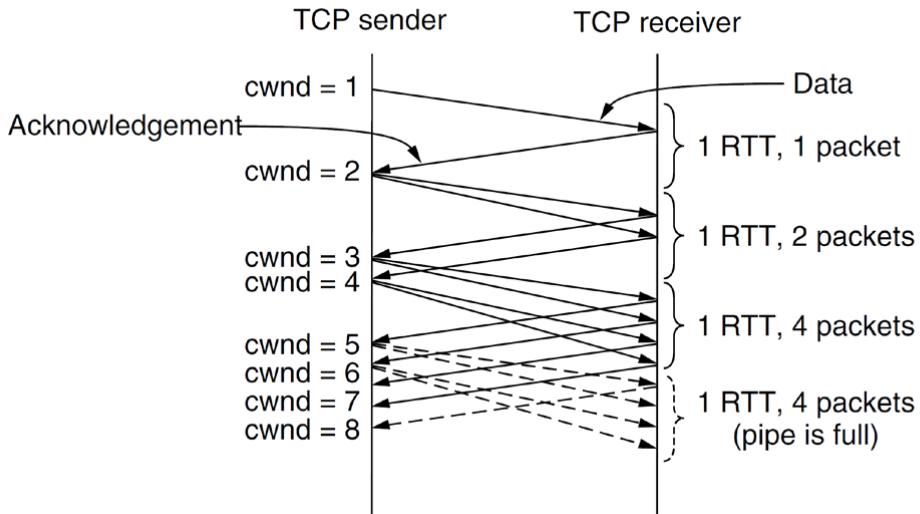
- Sender won't send any more data (window size is 0)
 - Sender can initiate an update by sending a ZeroWindowProbe
- Receiver won't receive anything, so won't send any ACKs to increase window size
 - Receiver can initiate an update by sending a WindowUpdate

Congestion Control Window CWND

- dynamically adjusted window size based on network performance to aid efficient transfer
- **CWND maintained** by the **sender**
 - Also only used at the sender
 - the **sliding window** that is controlled by the **receiver**
 - No changes to packet formats to send additional field

Slow-start

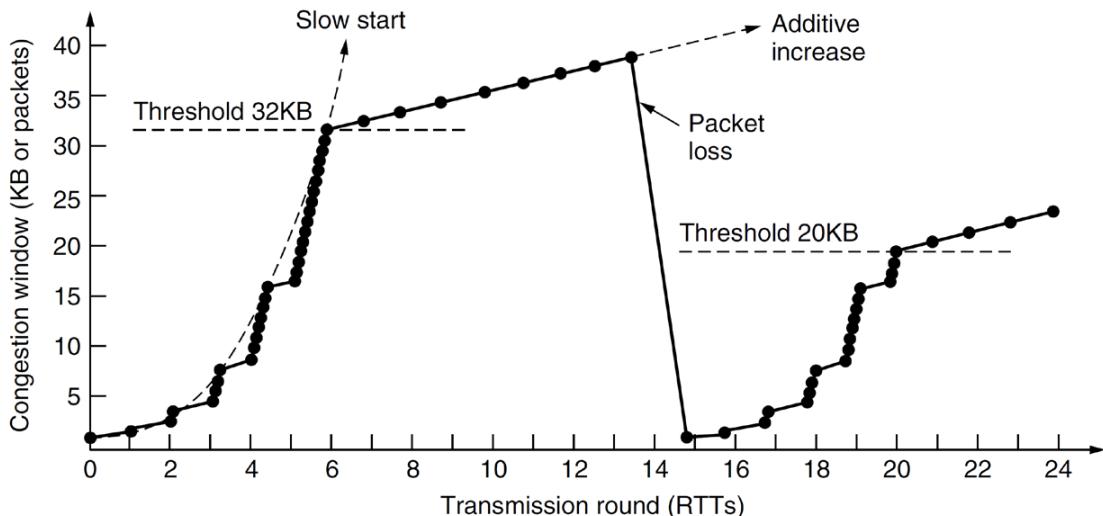
- Although it has a far from slow start, it is **exponential** in growth
- **Initial size:** $2 \times \text{maximum segment size}$
- Slow start keeps increasing the size of `ssthresh` until
 - a timeout occurs or
 - `ssthresh` is set to half the current congestion window size, start slow-start
 - the threshold is reached
 - start Additive increase
- **As each new segment is acknowledged, the congestion window is increased by one maximum segment size**
 - `CWND++`



- Each full window of acknowledgements doubles the congestion window which grows until
 - a timeout
 - It reaches a threshold, SSthresh

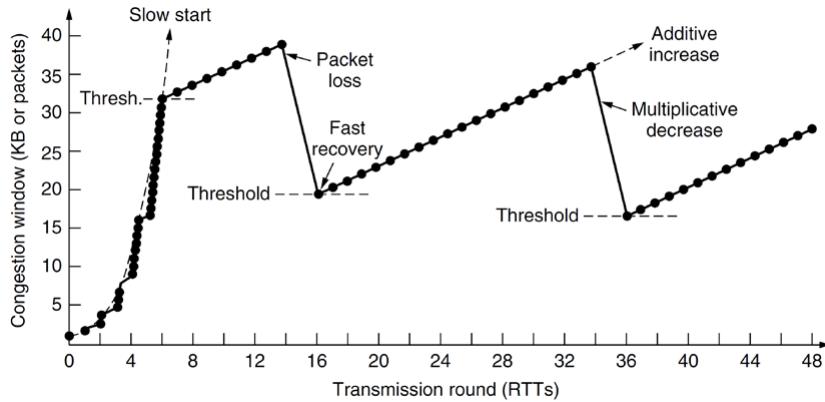
Congestion Avoidance (Additive increase)

- Known as TCP Tahoe (1988)
- Once `ssthresh` is reached the growth is slowed to linear, by adding 1 MSS to the congestion window for each successful ACK.
- Can also react to known lost segments via fast retransmission



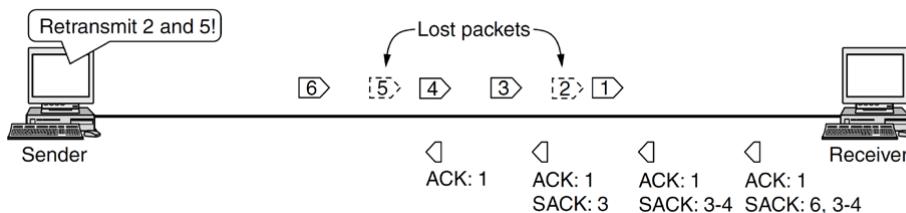
Fast recovery

- Fast recovery using duplicate acknowledgement counts
- Starts from new `ssthresh` instead of original starting value
 - effectively avoiding the slow start phase and going straight to additive increase



Further Optimisations

- **SACK (Selective Acknowledgements)**
 - provides greater ability to track segments in-flight, by allowing up to 3 ranges of bytes received to be specified



- **ECN Explicit Congestion Notification (ECE & CWR bits set during SYN)**
 - Allows IP Layer to indicate congestion is occurring without dropping the segment by setting an ECN flag
 - Receiver indicates this to sender via ECE (ECN Echo) flag
 - Sender acknowledges this by
 - setting the Congestion Window Reduced Flag (CWR),
 - reacts as if a segment has been lost, without actually having lost it

Macroscopic model

- These packet-level rules affect many things we care about
 - Fairness between flows
 - Response to long round-trip times (RTTs)
 - Response to random packet loss

W increases once per window

- Approximation: Each packet that arrives increases W by $1/W$

When a loss occurs, W is halved

- With probability p , $W \leftarrow W/2$

The average increase in window size is

$$\frac{(1-p)}{W} - p \frac{W}{2}$$

To be in equilibrium (balance), the average increase must be 0

$$W \approx \sqrt{2/p}$$

- The window is sent \leq once per round trip time (RTT), T
- Rate is $\frac{W}{T} \approx \frac{1}{T} \sqrt{2/p}$
- For a given packet loss rate, longer RTTs get less rate
 - “RTT unfairness”
- If RTT is small, TCP *forces* the packet loss rate to be high

Why: TCP is forcing the packet loss rate to be high in order to keep the window size small enough.

Model Limitation

- Window only responds to one packet loss per RTT
- Packet losses are clustered

TCP Question

Tutorial Week 4 Question 3

Service	TCP	UDP
Reliable data transfer between processes	Yes	No
Minimum data transmission rate between processes	No	No
Congestion-controlled data transfer between processes	Yes	No
A guarantee that data will be delivered within a specified amount of time	No	No
Preserve application-level message boundaries.	No	Yes
Guaranteed in-order delivery of data to the receiver	Yes	No

Tutorial Week 4 Question 5

Both UDP and TCP use port numbers to identify the destination entity when delivering a message. Give two reasons for why these protocols invented a new abstract ID (port numbers), instead of using process IDs, which already existed when these protocols were designed?

- process IDs are OS-specific
- not good for an OS-independent protocol
- A single process may establish multiple channels of communications.
- Single process ID (per process) as the destination identifier cannot be used to distinguish between these channels.
- Example: FTP
 - control connection
 - data connection
- Having processes listen on well known ports is easy, e.g. 80, but well-known process IDs are impossible.

Tutorial Week 4 Question 7

A process on host 1 has been assigned port p, and a process on host 2 has been assigned port q.

- Is it possible for there to be two or more TCP connections between these two ports at the same time?
 - No, A connection is identified only by its sockets.
 - One and only one connection can be established between a unique pair of ports on two hosts
- Is it possible for there to be more than one TCP connection on port p of host 1 at a time?
 - Yes, 5 tuple is used to identify a socket.
 - Connections can be identified as long as they all have different remote IP addresses or different remote ports.

Tutorial Week 5 Question 1

Consider a TCP connection between Host A and Host B. Suppose that the TCP segments traveling from Host A to Host B have source port number x and destination port number y. What are the source and destination port numbers for the segments traveling from Host B to Host A?

Host A: Source x, Destination y

Host B: Source y, Destination x

Tutorial Week 5 Question 4

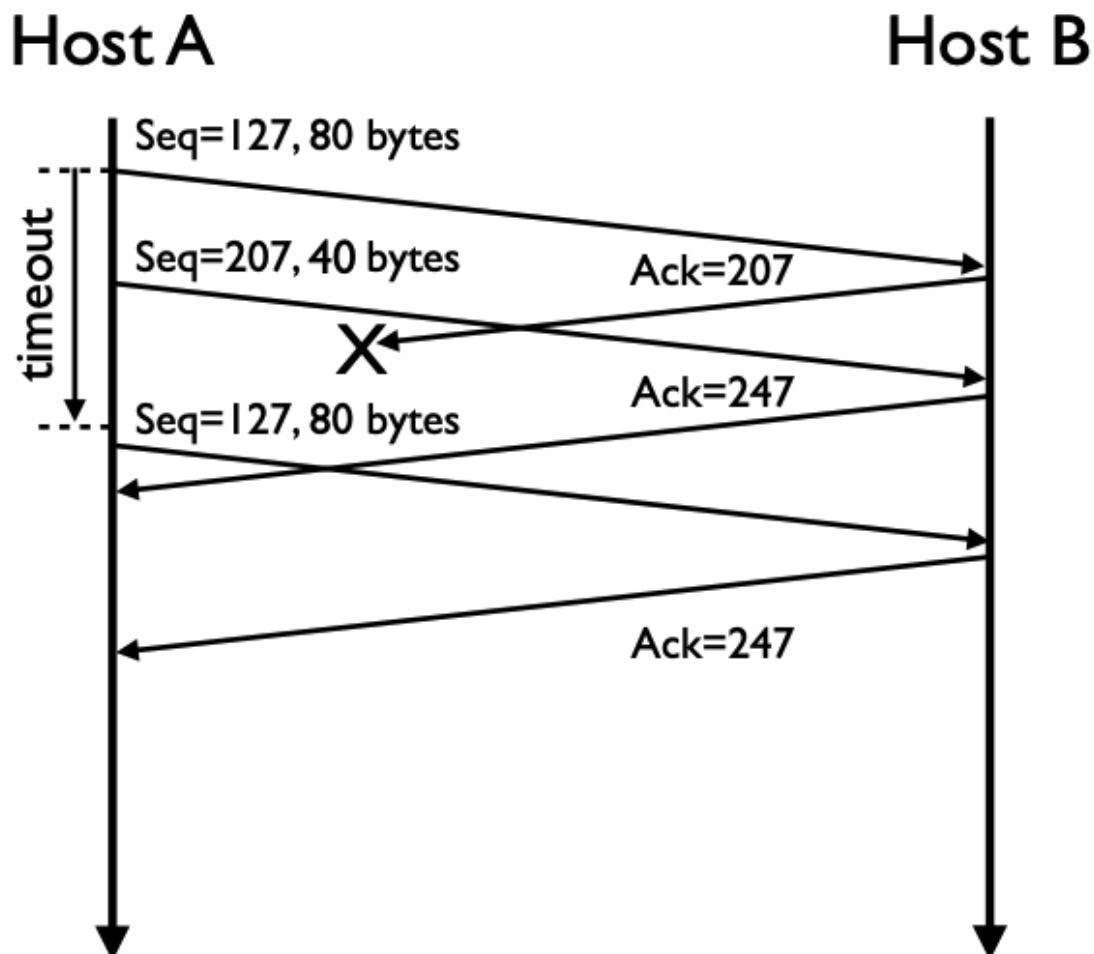
Suppose Host A sends two TCP segments back to back to Host B over a TCP connection. The first segment has sequence number 900; the second has sequence number 1024.

- How much data is in the 1st segment?
- $1024 - 900 = 124$ bytes
- Suppose the first segment is lost but the second segment arrives at B. In the acknowledgement that Host B sends to Host A, what will be the acknowledgement number?
- ACK = 900
- Suppose that now the third segment with sequence number of 1124 from Host A arrives to Host B, and the second segment is still missing. What will be the acknowledgement number?
- It is still 900.
- The acknowledgement number is always up to the byte-number in the byte-stream of the first missing byte.

Tutorial Week 5 Question 5

Host A and B are communicating over a TCP connection, and Host B has already received from A all bytes up through byte 126. Suppose Host A then sends two segments to Host B back-to-back. The first and second segments contain 80 and 40 bytes of data, respectively. In the first segment, the sequence number is 127, the source port number is 302, and the destination port number is 80. Host B sends an acknowledgment whenever it receives a segment from Host A.

- In the second segment sent from Host A to B, what are the sequence number, source port number, and destination port number?
 - Source port: 302
 - Dest port: 80
 - Seq # = 127 + 80 = 207
- If the 1st segment arrives before the second segment, in the acknowledgment of the 1st arriving segment, what is the acknowledgment number, the source port number, and the destination port number?
 - Source port: 80
 - Dest port: 302
 - Ack # = 127 + 80 = 207
- If the second segment arrives before the 1st segment, in the acknowledgment of the 1st arriving segment, what is the acknowledgment number?
 - ACK # = 127
- Suppose the two segments sent by A arrive in order at B. The first acknowledgment is lost and the second acknowledgment arrives after the first timeout interval. Draw a timing diagram, showing these segments and all other segments and acknowledgments sent. (Assume there is no additional packet loss.) For each segment in your figure, provide the sequence number and the number of bytes of data; for each acknowledgment that you add, provide the acknowledgment number.



Tutorial Week 5 Question 6

A reliable data transfer should provide two guarantees:

- the delivery of the packet, i.e., no packet loss;

- protection against accidental errors, i.e., packets corrupted by random changes of bits will not be delivered.

It does not guarantee in-order delivery; we say "TCP provides reliable, in-order delivery" because reliable does not imply in-order. What other notions of "reliability" are there, and at what layers are they provided?

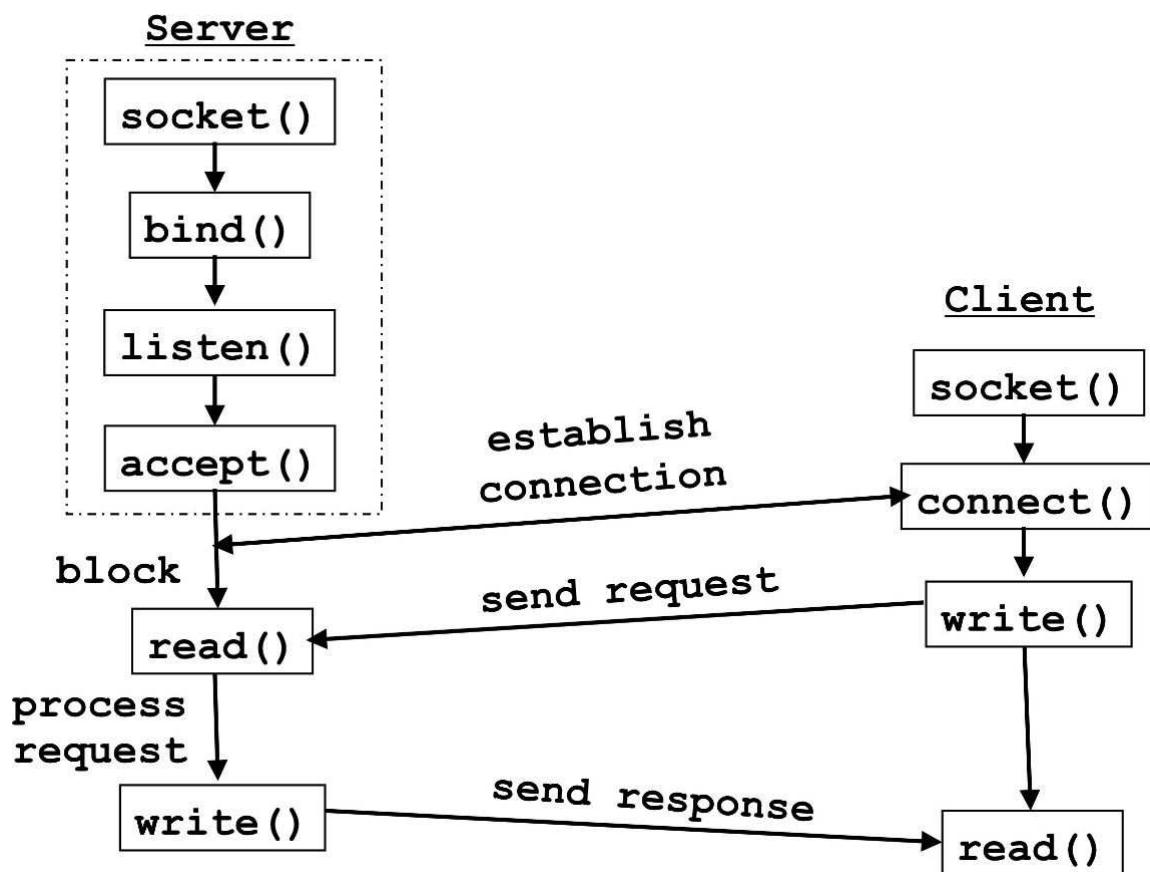
- The data hasn't been maliciously corrupted along the way. That is provided by **digital signatures**, usually at layer 7, but possibly at layer 6.
- The data was sent by the person it claims to have been sent by. That is provided by **authentication**, at layers 7, 5 (e.g., TLS) or 2 (e.g., WiFi).
- The data is private cannot be seen by others. That is provided by **encryption**, again at layer 7 or layer 6.

Socket Programming

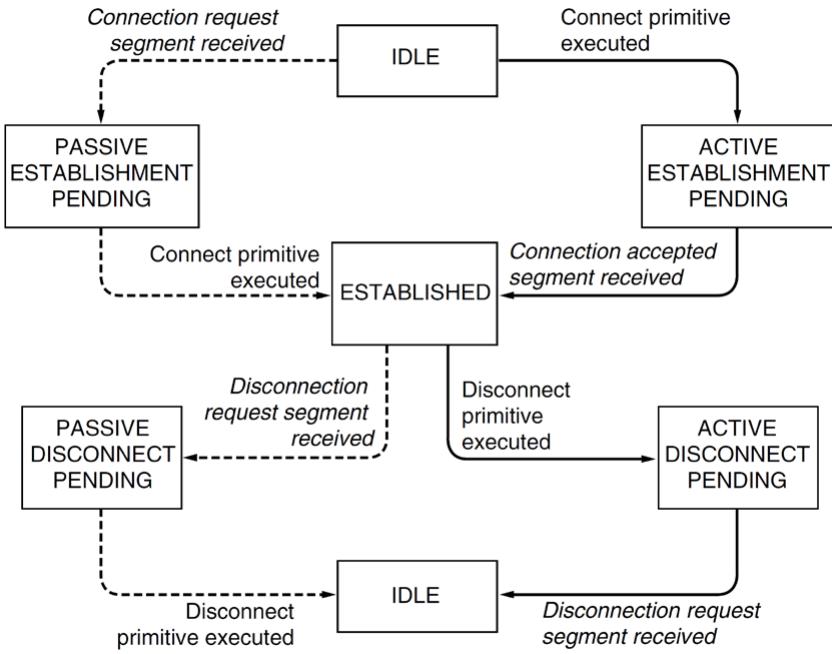
Berkeley Sockets

- Socket interface
 - originally provided in Berkeley UNIX
 - later adopted by all popular operating systems
 - simplifies porting applications to different OSes
- In UNIX, everything is like a file
 - all input is like reading a file
 - all output is like writing a file
 - file is "addressed" by an integer file descriptor
- API implemented as system calls
 - examples include `connect()`, `read()`, `write()`, `close()`

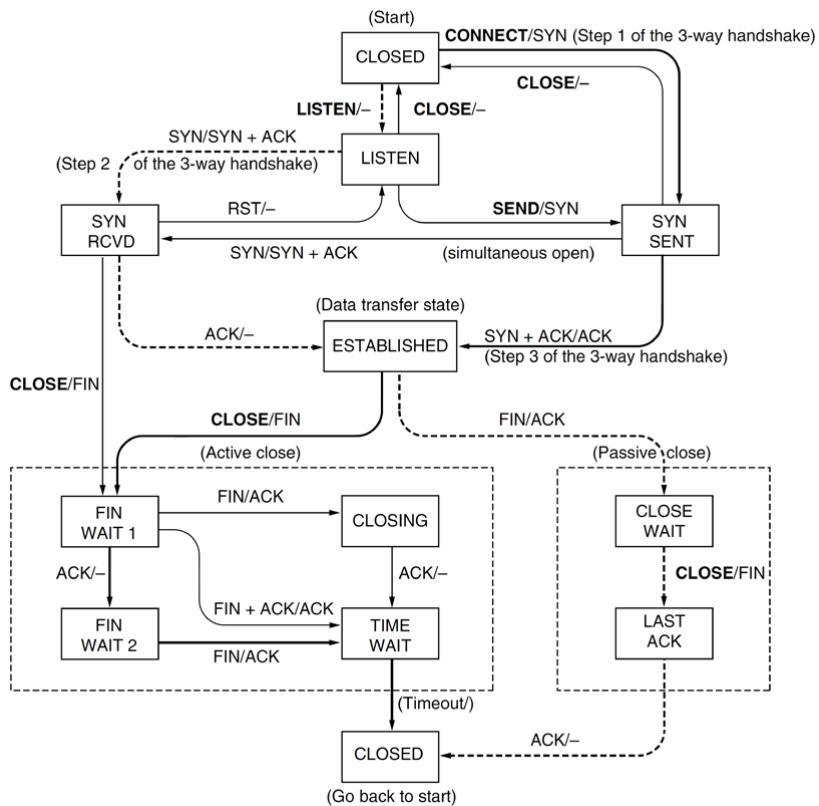
Using Sockets in C



State	Description
SOCKET	Creates a new communication endpoint
BIND	Associate a local address with a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Passively establish an incoming connection
CONNECT	Actively attempt to establish a connection
SEND	Send some data over a connection (write())
RECEIVE	Receive some data from the connection (read())
CLOSE	Release the connection



State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIME WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off



```

// Headers
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>

// Variables
int listenfd = 0, connfd = 0;
char sendBuff[1025];
char* host = "www.google.com";
struct sockaddr_in serv_addr;

// DNS Lookup
/* lookup the ip address */
server = gethostbyname(host);

// Create a socket
listenfd = socket(AF_INET, SOCK_STREAM, 0);

//initialise server address
memset(&serv_addr, '0', sizeof(serv_addr));
//initialise send buffer
memset(sendBuff, '0', sizeof(sendBuff));
//Type of address - internet IP
serv_addr.sin_family = AF_INET;
//Listen on ANY IP Addr
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
//Listen on port 5000
serv_addr.sin_port = htons(5000);

/* Server side bind and listen*/
// Bind and listen

```

```

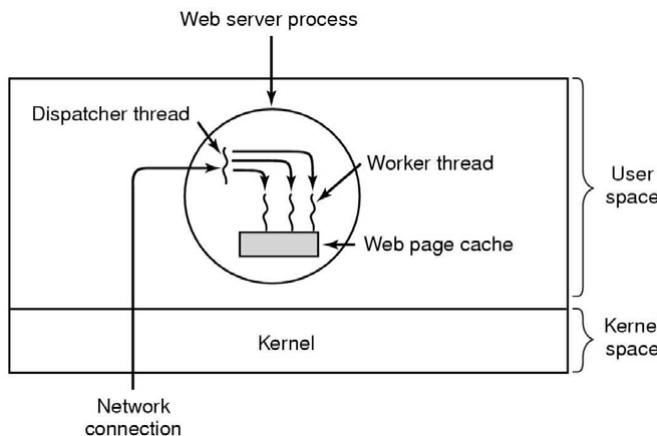
bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
// maximum number of client connections to queue
listen(listenfd, 10);
// Accept
connfd = accept(listenfd, (struct sockaddr*)NULL, NULL);
// send
snprintf(sendBuff, sizeof(sendBuff), "Hello world!");
write(connfd, sendBuff, strlen(sendBuff));

/* Client side connect to server */
connect(*sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr))
/* Receive */
while ( (n = read(connfd, recvBuff, sizeof(recvBuff)-1)) > 0) {
    //process received buffer
}
/* Close connection */
close(connfd);

```

Multi-threaded Web Server

- Clearly a web server needs to be able to handle concurrent connections from multiple clients
- This can be achieved through the usage of a multi-threaded web server



```

while (TRUE) {
    get_next_request(&buf);
    handoff_work(&buf);
}

```

(a)

```

while (TRUE) {
    wait_for_work(&buf)
    look_for_page_in_cache(&buf, &page);
    if (page_not_in_cache(&page)
        read_page_from_disk(&buf, &page);
    return_page(&page);
}

```

(b)

Tutorial Week 5 Question 2

Suppose that a Web server runs in Host C on port 80. Suppose this Web server uses persistent connections, and is currently receiving requests from two different Hosts, A and B. Are all of the requests being sent through the same socket at Host C? If they are being passed through different sockets, do both of the sockets have port 80? Discuss and explain.

- When the requests from A and B are the "connection requests" at the beginning of the three-way handshake, these requests are sent to the same welcome socket at Host C, which will be handled by the dispatcher thread.

- However, once the TCP connections are established between A and C, and B and C, each of A and B is allocated a dedicated connection socket, which is identified by a four-tuple: (source IP address, source port number, destination IP address, destination port number).
- Therefore, the requests from A and B are then sent through different sockets as at least the source IP addresses of those sockets are different. Moreover, all those sockets have destination port number 80.

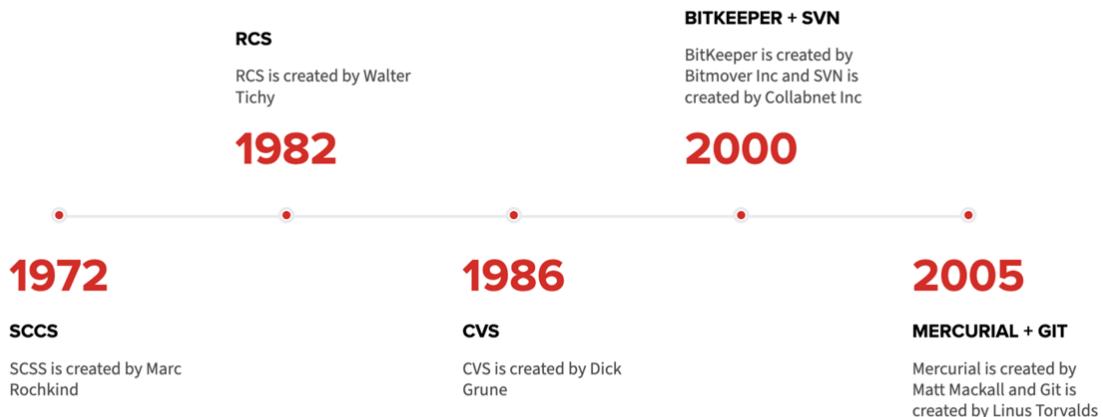
Git

What is Git?

- **Git** track and control changes over files
- Why use it?
 - Collaboration
 - Revision history
 - Audit changes
 - Backup

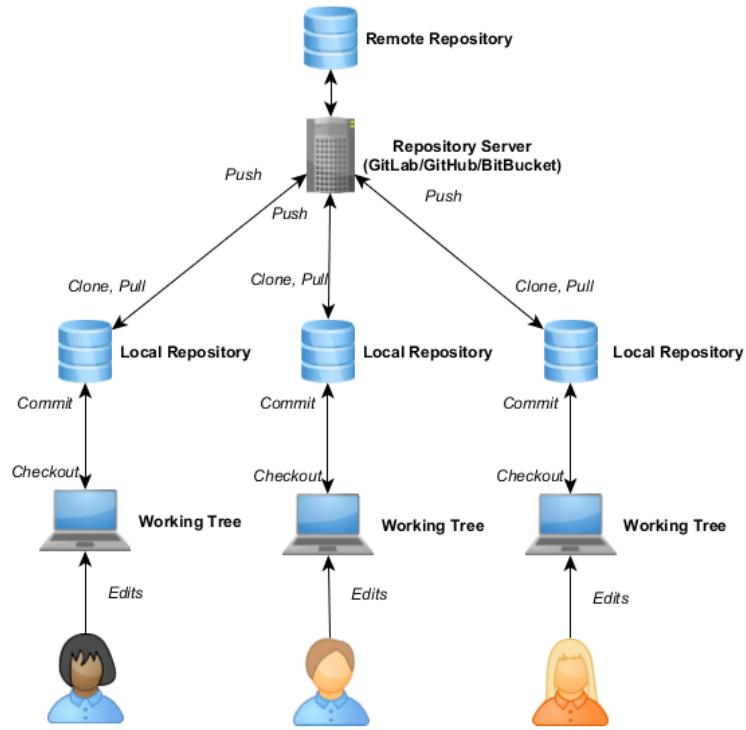
VCS History

- Source Code Control System (SCCS) 1972 - Unix
- Revision Control Systems – 1982
- Concurrent Version Systems - 1986
- BitKeeper – 2000 proprietary
- Git – developed by Linus Torvalds

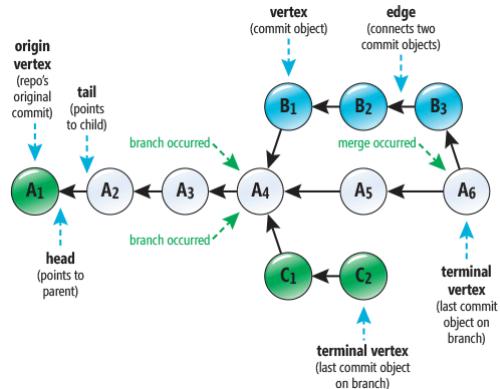


Architecture

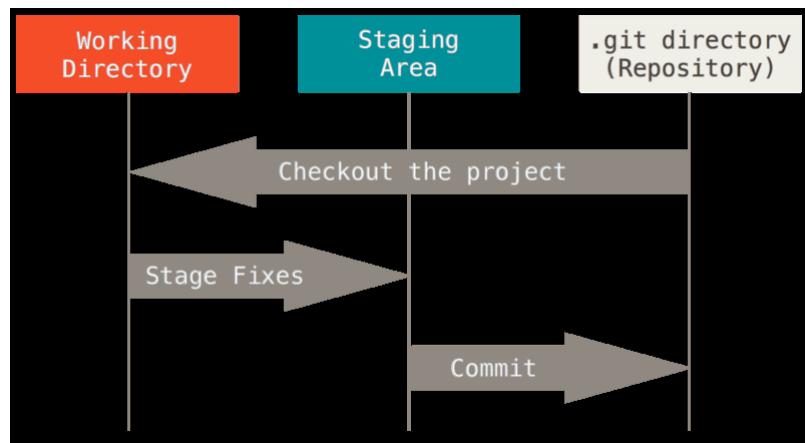
- Distributed version control
 - Git stores changes remotely by syncing repositories



- **Data structure:** Directed acyclic graph(DAG)



- **Working directory:** top level folder with single version of the files
- **Staging area** – file in .git that records files to be added to next commit
- `.git` directory – configuration and repository database



Storage

- Git does not store deltas (**changes**), it takes snapshots of the file system

- Files that are the same are referenced, not duplicated
- This is a big difference compared to other VCS
 - Makes branching easier and cheaper
- Git occasionally packs those snapshots into single files, and uses delta compression to **save space**
 - `git gc --aggressive`

Usage

- **init:** initialise a git repo
- **Staging** is the process of marking the file as to be included in the next commit
- **Checks out:** switches the working tree to a particular commit/branch
- **Branches** are just text files pointing to a different commit
- **Pull:** Fetch retrieves any changes from the remote (origin)
- **Merge:** combines those changes into the local repo
- **Push:** transfers your local commits to the remote repo
 - `Git push <remote> <branch>`
 - Creates a branch in the destination repository
 - Push only allowed for fast-forward merge
 - Non fast-forward merge: cannot make changes on the remote without losing commits
- **remote:** Adds remote to a local repository – useful when creating a new repository
 - `git remote add remotename host:/dir.git`
 - Can have multiple remotes for a single repository
- **Tags:** can be used to add more informative names to commits, i.e. releases 1.0
- **Binary File:**
 - Git automatically packs file changes using delta compression
 - Increases the size of the clone operation
 - Binary files typically exhibit large changes
- **Fork**
 - Link remains to parent
 - Useful for large changes, keep development work out of main repository
 - Remains even if original repository is removed

Risks

- Build processes that depend on external repositories are fragile
 - A user deletes the repository -> build process breaks
 - Risk of unknown code changes in builds
 - Better to fork the repository
 - Higher support costs – need to pull changes
- Github allows recreating deleted users
 - Serious security concern – inject malicious code
- **Credentials Leak** Be careful what you store in public repositories
 - Never store credentials AWS keys

Git Question

Tutorial 5 Question 7

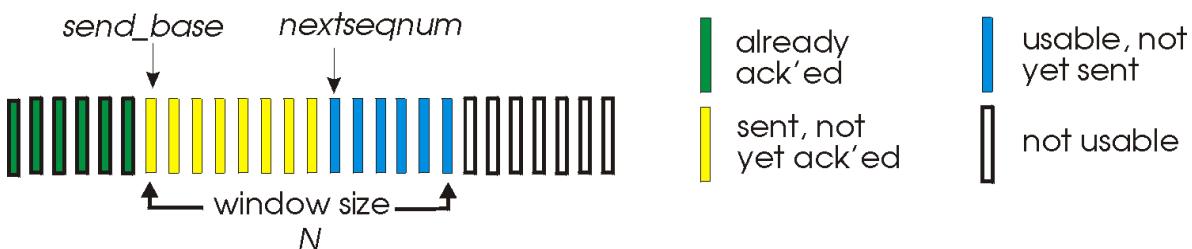
Imagine you were going to reimplement git. What algorithm would you use to merge two sets of changes to a text file? Think about how fast your scheme would be, and how accurate it would be.

- compare each new version to the shared parent. A line-based difference like diff would be suitable.
- This finds the line numbers of the lines that are inserted, deleted and changed, but doesn't consider which characters within a line are changed.
- "replay" the two sets of inserts and deletes on the original file
- An insertion by one difference will change the line numbers of subsequent lines in the combined file, and so the merger would need to keep track of that.
- If a line is changed in only one of the new files, then the line in the output will be the line from the corresponding new file.
- If a line is changed in both new files, then either the line is the same in both new files, or it is different.
 - If it is the same, then the line in the output will be the line from the new files.
 - Otherwise, there is a "conflict", which has to be resolved either manually or by a more advanced algorithm.

Original flow + loss control

- Flow+loss control had existed on single point-to-point links for a long time
- TCP originally used the experience from the link layer
 - Caused bad design decision

Go-back-N



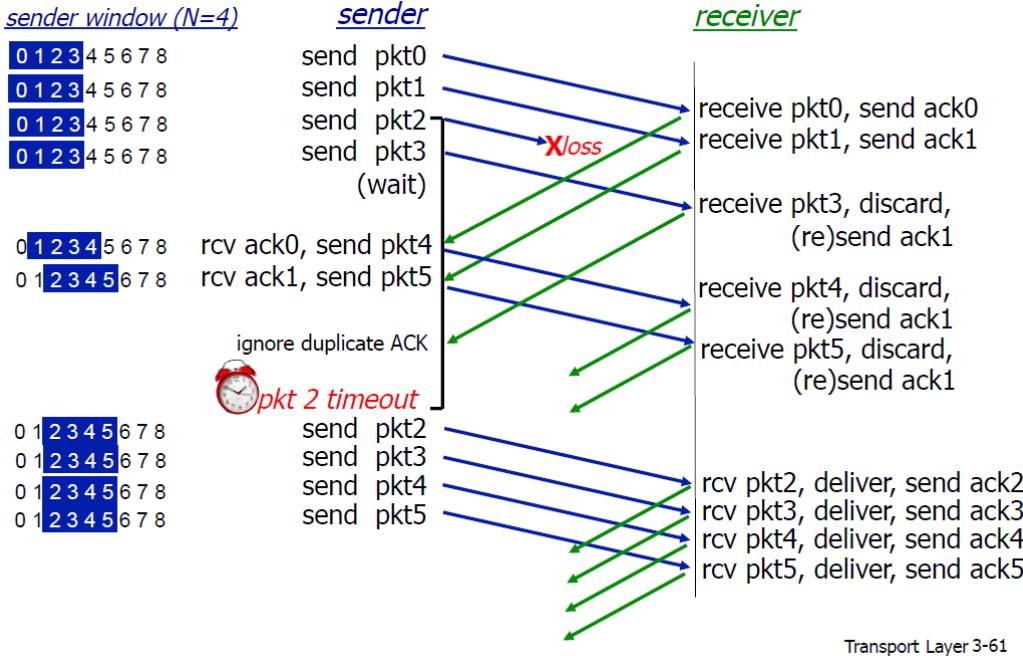
Sender

- sender can have up to N unacked packets in pipeline
- sender has **ONE** timer for oldest unacked packet
 - when timer expires, retransmit packet n and all higher seq # packets in window

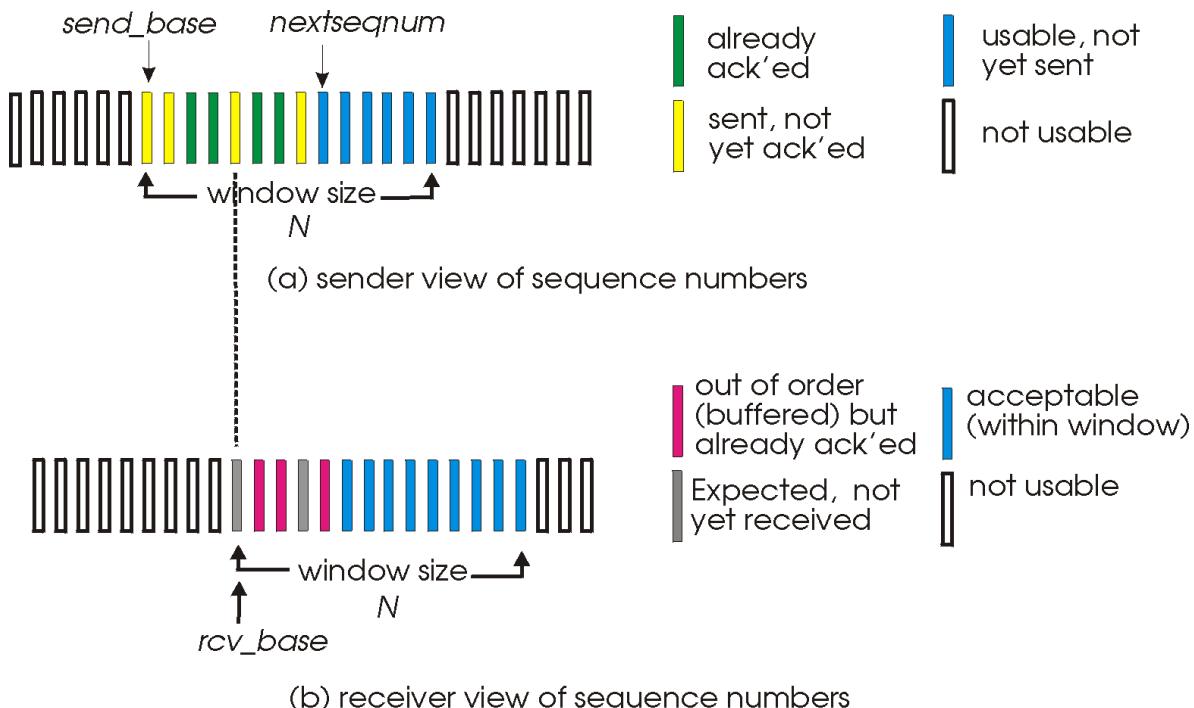
Receiver

- receiver always send ACK for correctly-received pkt with highest in-order seq
 - **cumulative ack:** ACKs all pkts up to, including # n
 - may generate duplicate ACKs

- doesn't ack packet if there's a gap
- receiver **discard** out-of-order packets
 - discard (don't buffer): no receiver buffering!
 - re-ACK pkt with highest in-order seq #
- **Advantage:** Receiver doesn't need to store/reorder packets
- **Disadvantage:** wasteful, causing Congestion collapse



Selective Repeat

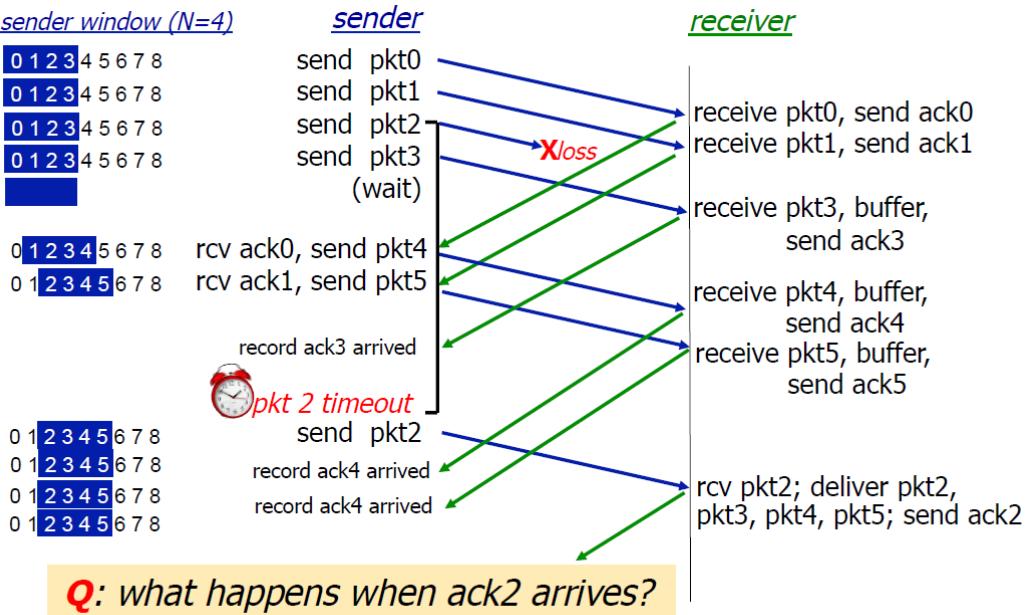


Sender

- sender can have up to N unack'd packets in pipeline
- sender maintains timer for each unacked packet
 - when timer expires, retransmit only that unacked packet

Receiver

- receiver sends individual ack for each correctly received packet
- receiver buffers out-of-order packets



Transport Layer 3-65

Advantage: efficient - only retransmit the lost packet

Disadvantage:

- need to buffer out-of-order packets
- more complex, only helps if loss is common

Congestion collapse

- **When:** In the late 1980s, the internet had “congestion collapse”
- **What:** Router buffers were overflowing, causing high loss
- **Why:** Senders were doing go-back-N, so that every packet loss caused N more packets to enter the system
- **Solution:** Selective repeat (“fast retransmit”)

Question

Tutorial 5 Question 3

A negative acknowledgement tells the sender that a particular packet has not been received, rather than that a packet has been received. Consider a reliable data transfer protocol that uses only negative acknowledgments.

- Suppose the sender sends data only infrequently. Would a NAK-only protocol be preferable to a protocol that uses ACKs? Why?
- In a NAK only protocol, the loss of packet x is only detected by the receiver when packet x+1 is received.
- If there is a long delay between the transmission of x and the transmission of x+1, then it will be a long time until x can be recovered, under a NAK only protocol.
- Now suppose the sender has a lot of data to send and the end-to-end connection experiences few losses. In this second case, would a NAK-only protocol be preferable to a protocol that uses ACKs? Why?

- if data is being sent often, then recovery under a NAK- only scheme could happen quickly.
- Moreover, if errors are infrequent, then NAKs are only occasionally sent (when needed), and ACK are never sent a significant reduction in feedback in the NAK-only case over the ACK only case.

Internet (network) Layer

Role

- **Goal:** Get data from the source all the way to the destination
 - May not be in a single hop (point-to-point link)
- **Efficiency:** Route Traffic efficiently
- **Network layer addressing:** Nodes must be given names (addresses)

Where

- Most Network Layer code runs on routers
- We will refer to the protocol data units as packets

Services

Connectionless("datagram" network)

- Packet switching (Internet Protocol - IP)
- Minimum required service: "send packet"

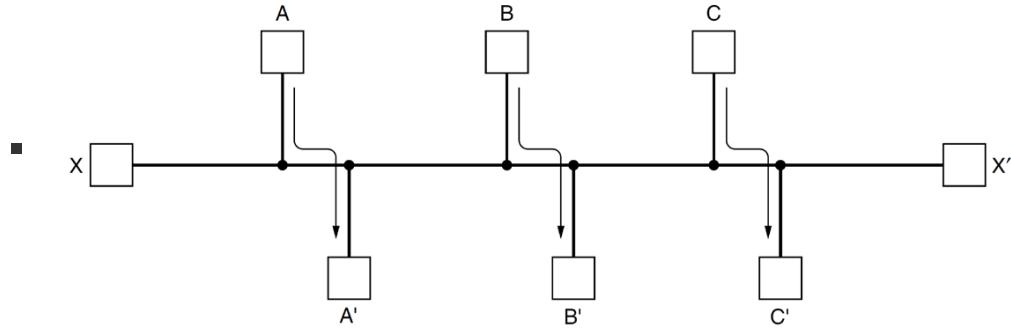
Connection-oriented("virtual circuit" network)

- (virtual) Circuit Switching
 - Asynchronous Transfer Mode – ATM
 - MultiProtocol Label Switching MPLS
- These usually act as a single "link" of an IP network

Routing

- **Routing algorithm:** decides which output line an incoming packet should be transmitted on
 - Combination of
 - an algorithm local to each router
 - a protocol to gather the network information needed by the algorithm
 - Properties of a good routing algorithm
 - **Correctness:** finds a valid route between all pairs of nodes
 - **Simplicity**
 - **Robustness:** a router crash should not require a 'network' reboot
 - **Stability:** a stable algorithm reaches equilibrium and stays there
 - **Fairness**
 - **Efficiency**
 - **Flexibility** to implement policies
- Routing **Trade-off**

- Fairness vs. Efficiency



- Delay vs. Bandwidth

- Minimise the number of hops a packet has to make
 - Tends to reduce per packet bandwidth and improve delay
 - Hopefully also reduces the distance travelled – but not guaranteed
- Actual algorithms give a cost to each link
 - More flexible, but still cannot express all routing preferences

	Non-adaptive (static routing)	Adaptive
Adaptivity	Does not adapt to the network topology	Dynamic routing, adapts to changes in topology and potentially even traffic levels
	Calculated 'offline' and uploaded to the router at boot	may get information from adjacent routers, or all routers in the network
Response to Failure	Does not respond to failure	respond to failure
When to use	Reasonable where there is a clear or implicit choice	-
Optimization	-	distance, hops, estimated transit time, etc

Flooding

- **Guarantees** shortest distance and minimal delay
- Useful benchmark in terms of speed
- **Advantage:** Extremely robust
 - if there is a path it will find it
- **Disadvantage:** Highly inefficient
 - generates many duplicate packets
- Have to have a way of discarding packets (TTL)
 - If unknown can be set to diameter of network

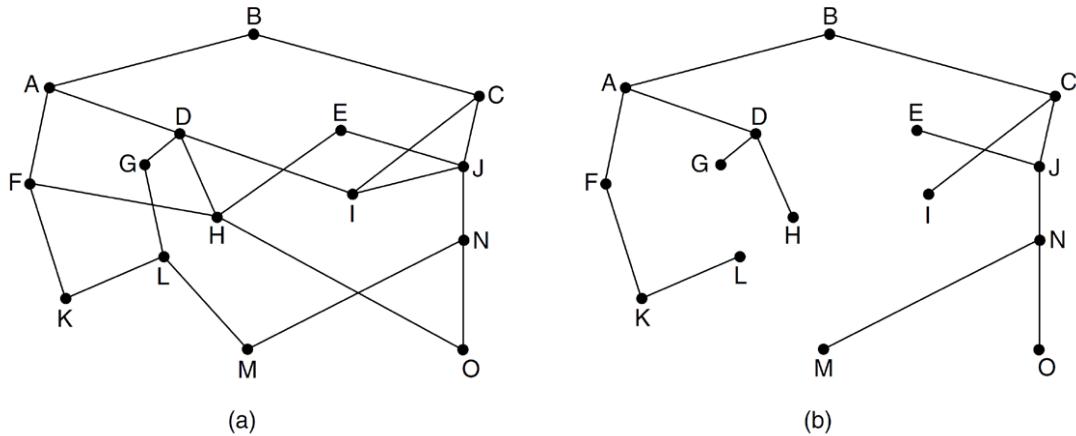
Optimality Principle

- If router J is on the optimal path from router I to K, then the optimal path from J to K also falls along the same route.



Sink Tree

- The optimality principle means that a set of optimal routes from all sources form a tree rooted at the destination



Shortest Path Algorithm

- View as a labelled graph
 - Label weight based on delay, distance, cost, etc.
- At each step, each node is labelled with its distance (sum of costs on edges) from the source and the best known path
-

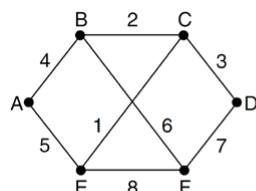
Link State Routing

- Replaced Distance Vector Routing ("Bellman-Ford") that had problems with **converging quickly enough**
- Link state routing distributes the topology, Everyone then performs **centralised routing**
- 5 step process each router must follow
 - Discover its neighbours** and learn their network address
 - To discover neighbours, a router on boot sends out a HELLO packet on each interface.
 - The router on the other end must reply with its unique ID
 - Set the distance** or cost metric to each of its neighbours
 - Cost can be set automatically or manually
 - Common technique is bandwidth 1 Gbps = 1, 100 Mbps = 10
 - Could use delay as well – calculated using an ECHO packet

- **Traffic engineering:** Many networks manually choose preferred routes and then look for link costs to make those routes the shortest.

3. Construct a packet containing all it has just learned

- Link State Packet consists of ID, sequence number, age, and a list of neighbours and their respective costs
 - Building the packet is easy, deciding **when to build them is difficult**
 - At intervals?
 - When a change occurs – link disconnect?



(a)

Link	State	Packets			
A	B	C	D	E	F
Seq.	Seq.	Seq.	Seq.	Seq.	Seq.
Age	Age	Age	Age	Age	Age
B 4	A 4	B 2	C 3	A 5	B 6
E 5	C 2	D 3	F 7	C 1	D 7
	F 6	E 1		F 8	E 8

(b)

4. Send the packet to, and receive packets from, all other routers

- **reliable flooding** is used to send packets to all other routers
 - acknowledgements are used to guarantee every other router receives the packet
 - **comparing seq #**
 - **Length:** Sequence numbers are 32 bits to avoid wrap-around
 - When a router receives a Link State Packet it compares the sequence number to the one it previously received
 - If the sequence number is not larger it discards it and does not forward on the flood
 - **age**
 - age is reduced by 1 each second
 - when the age hits zero the information is discarded
 - resolve the problem that if a router crashes and restarts its sequence number from 0

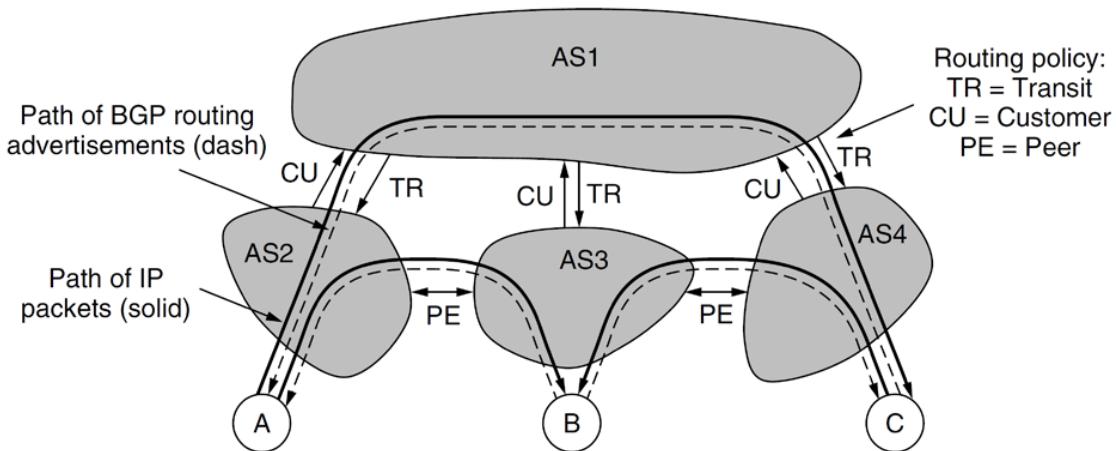
Distance vector Routing

- Distance vector routing uses a **true distributed algorithm**
 - Nodes announces the distance from themselves to each destination
 - When they receive a new announcement of others' distances,
 - they update their own estimates
 - and make a new announcement
 - Bellman Ford algorithm

Border Gateway Protocol (BGP)

- **Autonomous Systems (AS)**: collections of routers under the same administrative control
 - Each network will have
 - A protocol for internal routing (usually based on linked state)
 - A protocol for external routing between ASes

- Must be the same for all ASes
- In contrast to the internal routing, BGP needs to consider **politics** as well
 - Companies not willing to have their network used by others
 - ISPs not wanting other ISPs' traffic on their networks
 - Not carrying commercial traffic on academic networks
 - Use one provider over another because they are cheaper
 - Don't send traffic through certain companies or countries because we don't trust them
- Can't always say one route is "better" than another
 - Better in some respects, worse in others
 - Bellman's optimality principle doesn't always apply
- Roles in BGP
 - **Customer/provider:** pay you for "transit" of traffic I send or receive
 - **Provider** advertises routes for the entire internet
 - **Customer** only advertises routes for their network to avoid transiting other traffic
 - **peering agreements:** we carry each others' traffic without charge



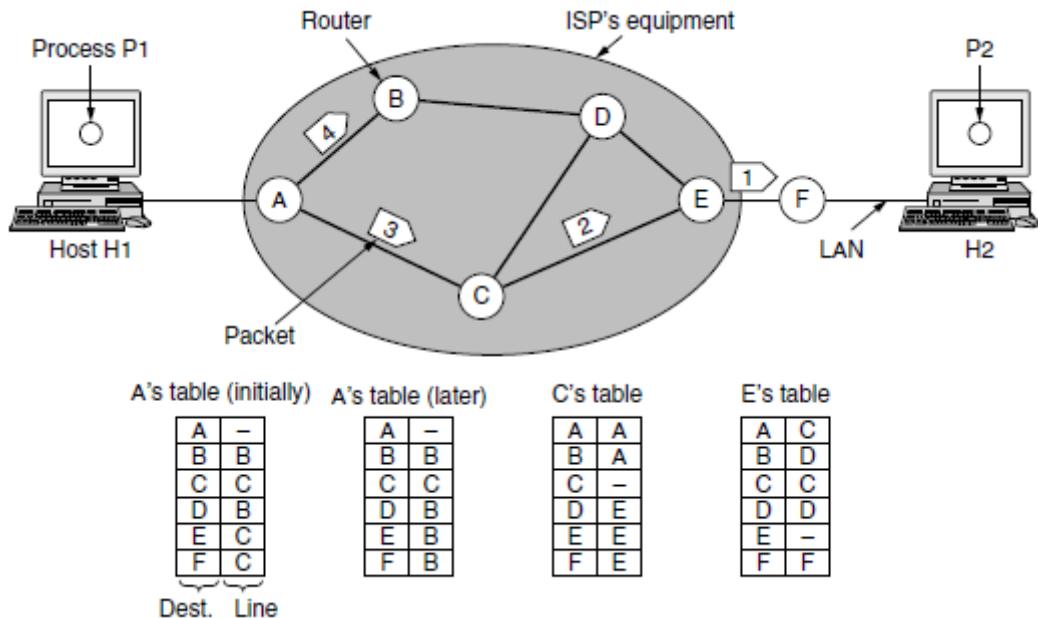
- **Attack:**
 - A malicious AS can advertise routes for networks at extremely low cost, causing traffic to re-route through that AS
 - 2017 Russian AS advertised routes for Google, Apple, Facebook, Microsoft, Twitter, etc.
 - 2008 Pakistan attempted to block YouTube, but inadvertently blocked it for the entire internet
 - an effective way to divert traffic for monitoring or disruption

Packet forwarding

- Each router has a forwarding table (or routing table).
 - This maps destination addresses to outgoing interfaces.
- Upon receiving a packet
 - inspect the destination IP address in the header
 - index into the table
 - determine the outgoing interface
 - forward the packet out that interface
- The packet travels along the path to the destination.

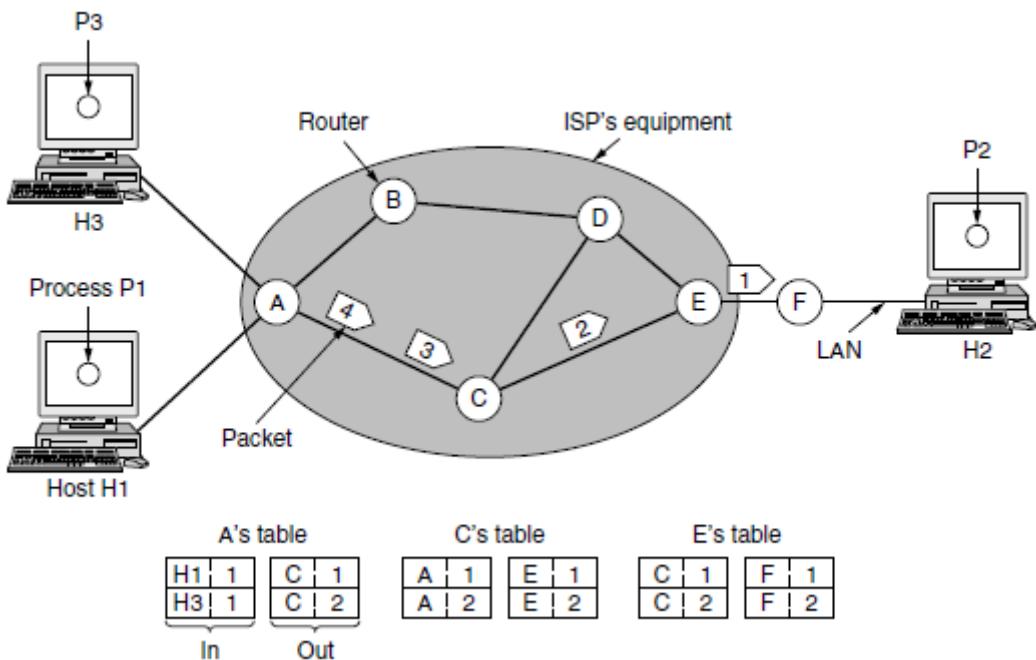
Connectionless

- The internet is a packet switched network
- Host H1 wants to send a packet to H2
 1. Transmits it to the nearest router (A)
 2. The packet is buffered while it is arriving, and the checksum is verified
 3. If valid, the packet is stored until the outgoing interface is free
 4. The router forwards the packet onto the next router in the path
 5. Repeat 2-4
- Paths can change for packets in the same transport layer connection
- **Forwarding tables** (also called routing tables)



Connection-oriented

- Forwarding table:
 - “In”: connection ID
 - “Out”: next hop and new connection number
- Connection number is local to a hop



MuliProtocol Label Switching

- Widely deployed Virtual Circuit (connection-oriented) Network Layer Protocol (below the internet sublayer)
- Purpose is **Quality of Service**
 - Prioritising traffic
 - Service Level Agreements for network performance
 - Reliable connectivity with known parameters
- Popular with businesses that want to connect multiple sites and phone companies carrying voice traffic

Connection-oriented vs Connectionless

Issue	Datagram Network	Virtual Circuit
Type	Connectionless	Connection-oriented
Addressing	Each packet has full source and destination	Each packet contains a short VC number
State	Routers do not hold state information about connections	Each VC requires router table space Router table will be lost if router reboots
Routing	Each packet independently	Defined at set-up
Quality of Service	Difficult	Easy if enough resources
Congestion control	Difficult	Easy if enough resources
Cost	Relatively cheap	Expensive

Quality of Service

Why is Quality of Service (QoS) important

- Not all services are equally important or robust to network delay
 - VoIP vs file downloads
 - VPN connections vs web browsing
- Within your own network, or within a single administered network (ISP), services can be prioritised
 - Own network - typically explicitly
 - the Differentiated Services header can be used to define classes of traffic
 - Shared network - typically implicitly (ISP traffic shaping)

Internet Protocol

- **Responsibility:** Responsible for moving the packets through the various networks from source to destination host
- **Paths**
 - Multiple paths through the network
 - Important for redundancy
 - Routing algorithms are used to determine best path
- **Guarantee:** IP is an best effort protocol
 - no guarantee on performance
- **Principles:** Designed with a number of principles in mind, including
 - **OK** is better than an ideal standard "in progress"
 - Keep it simple
 - Be strict when sending and tolerant when receiving
 - Make clear choices
 - Avoid static options and parameters – negotiate them at runtime
 - scalability

IP Addressing

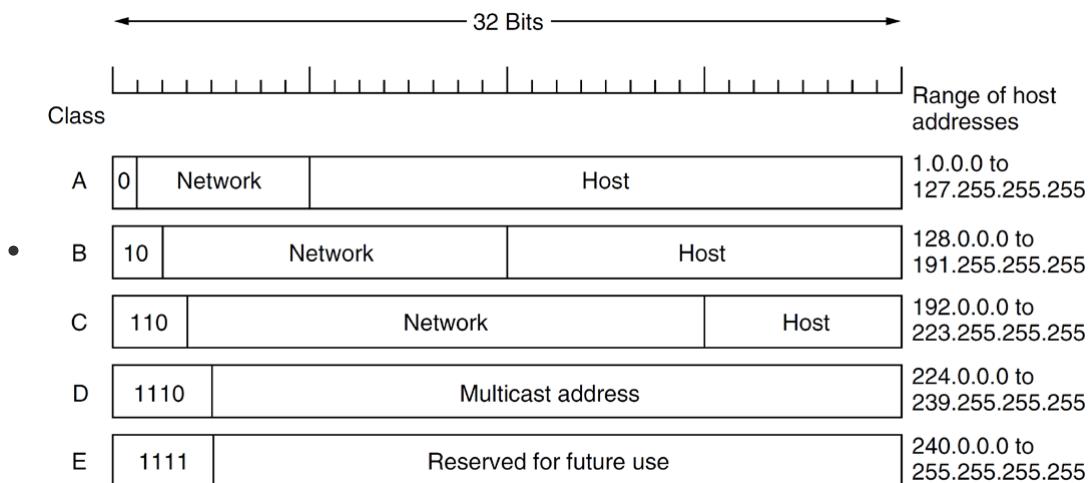
IPv4 Addresses

- **Length:** 32-bit number
- **Lowest:** 0.0.0.0
- **Highest:** 255.255.255.255
- **Allocation**
 - Overall IP allocation responsibility of Internet Corporation for Assigned Names and Numbers (ICANN) by delegation to IANA and Regional Internet Registries (RIR's)
- **Interfaces not hosts** : addresses are given to interfaces not hosts
- **Supply:** Supply of IPv4 addresses has basically been exhausted

Classes

- Originally IP addresses were allocated based on classes

- Routing was performed based on the class, which could be derived from the first part of the address

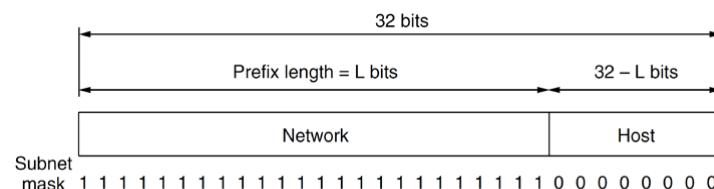


Classless Inter Domain Routing

- Classes simplify routing
- Wasteful. Networks often much bigger than needed
 - Network with 260 nodes must be class B with 65,536 addresses
- Classless Inter Domain Routing
 - Each interface / route explicitly specifies which bits are the “network” field
 - Network with 260 nodes only needs 9 bits for “host” field
 - 512 addresses
 - Can have 128 times as many of these networks as class B networks
- Hierarchical - encodes the network and host number
 - **Network** in top bits
 - **Host** in bottom bits
 - **disadvantages**
 - can be quite wasteful if not carefully assigned

Prefixing

- **Prefix:** A network corresponds to a contiguous block of IP address space, called a prefix
 - Prefixes are written as the lowest IP address followed by a slash and the size of the network portion - 192.0.2.0/24
 - Can also be written as a subnet mask, a binary mask of 1's



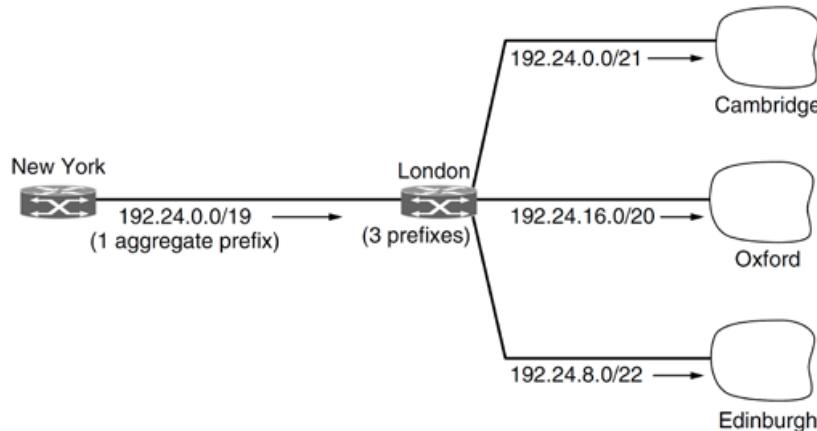
- **Blocks**

IP	Network	Host	Subnet mask
192.0.2.0/24	24 bits are for the network 192.0.2.0	8 bits for hosts – up to 256 addresses	255.255.255.0
10.0.0.0/8	8 bits are for the network 10.0.0.0	24 bits for hosts – up to 16,777,216 addresses	255.0.0.0

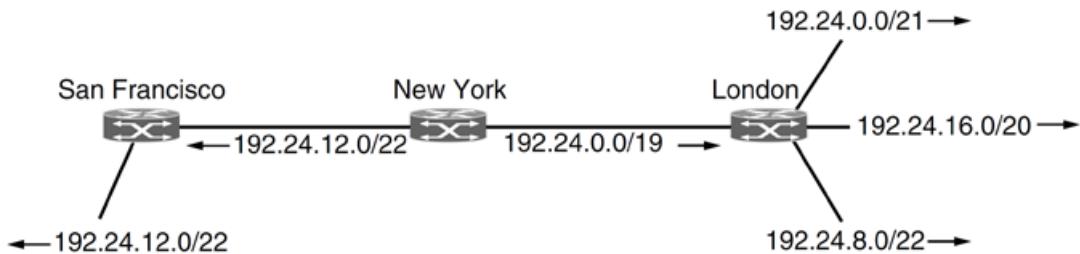
- **Network number** = network mask (bitwise-AND) IP address
- **Prefixing** is crucial for efficient routing on the internet
 - Since networks are assigned in blocks, intermediary routers need only maintain routes for the prefixes, not every individual host
 - Only when the packet arrives at the destination network does the host portion need to be read

Route aggregation

- Aggregation is performed automatically
- Currently it roughly halves the size of the routing table



- Prefixes can overlap, in which case the **longest matching prefix is selected**

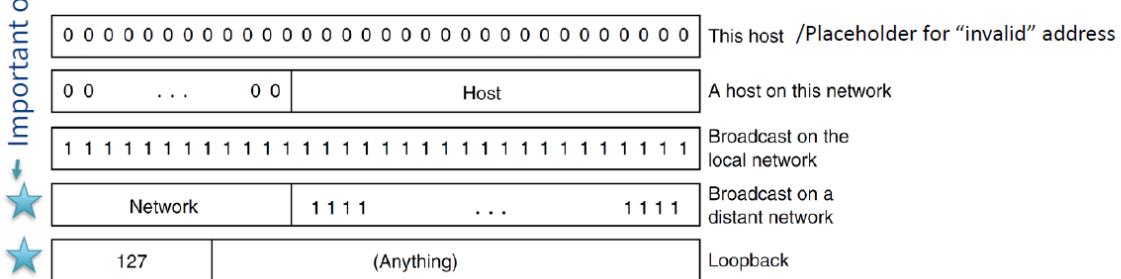


Special IP Addresses

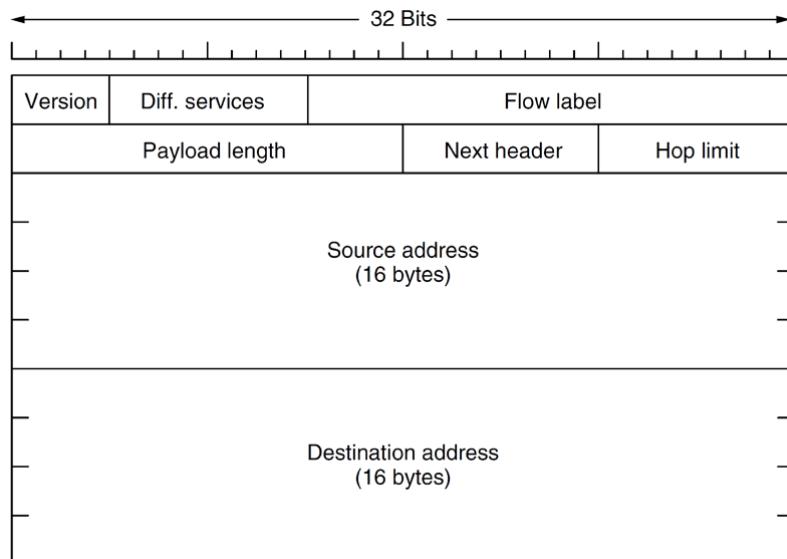


Special IP Addresses

Private address range	Prefix & Mask	Available Addresses
★ 10.0.0.0 – 10.255.255.255	10.0.0.0/8 (255.0.0.0)	16,777,216
★ 172.16.0.0 – 172.31.255.255	172.16.0.0/12 (255.240.0.0)	1,048,576
★ 192.168.0.0 – 192.168.255.255	192.168.0.0/16 (255.255.0.0)	65,536
↑ Link local / zero config		
169.254.0.0 – 169.254.255.255	169.254.0.0/16 (255.255.0.0)	65,536

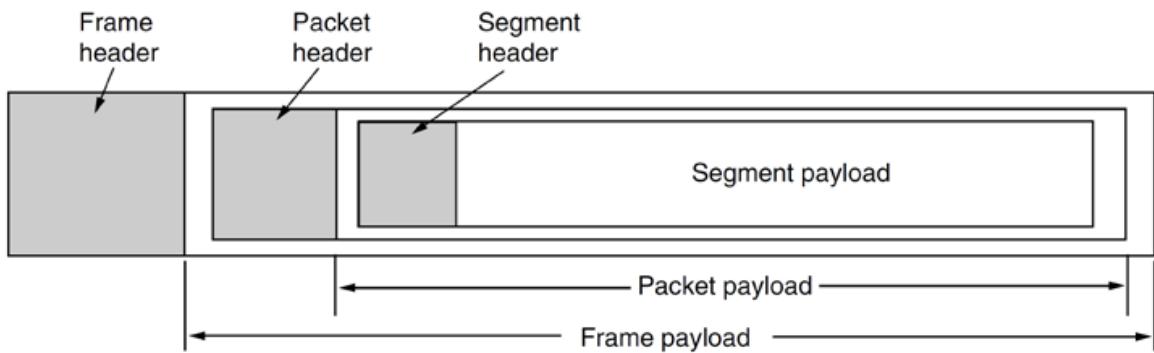


IPv6

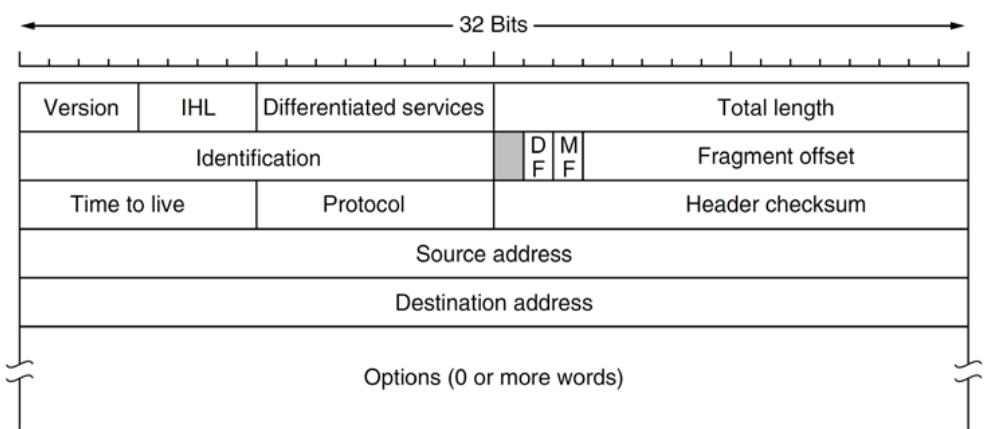


- **Length:** IPv6 addresses are **128** bits
 - Written as 8 groups of 4 hex digits
 - 8000:0000:0000:0000:0123:4567:89AB:CDEF
 - Can be optimised by stripping one group of consecutive 0's
 - 8000::123:4567:89AB:CDEF
 - Backwards compatibility with IPv4
 - ::ffff:192.31.2.46 (note the mix of hex with decimal)
- **Problem to address:** Designed to address the problem of exhausting the IPv4 address space
- Whilst solving that problem some other changes were made
 - Simpler header – allows faster processing
 - Improved security – now back-ported to IPv4
 - Further Quality of Service support

IP Header

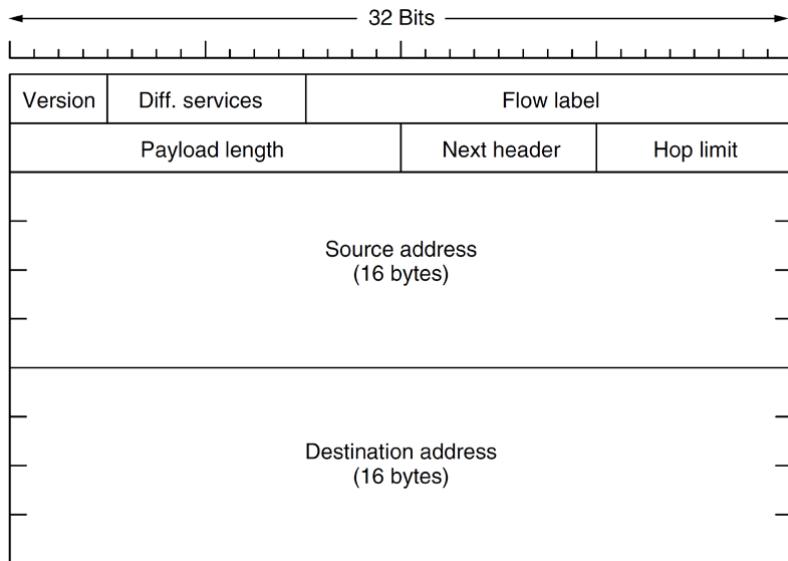


IP Version 4 Protocol



Field	Usage
Version	Protocol version 4
IHL	Header length in 32 bit words(min 5, max 15)
Differentiated services	6 bits for service class + 2 bits for congestion control (ECN)
Total length	Including payload, max 65,535
Identification, DF, MF, Fragment Offset	Used in the handling of fragmentation
Time to live (TTL)	Countdown of hops, at zero packet is discarded
Protocol	Transport layer service (TCP/UDP/SCTP/DCCP/etc.)
Source and Destination	IPv4 address
Options	Rarely used and poorly supported

IP Version 6 Protocol

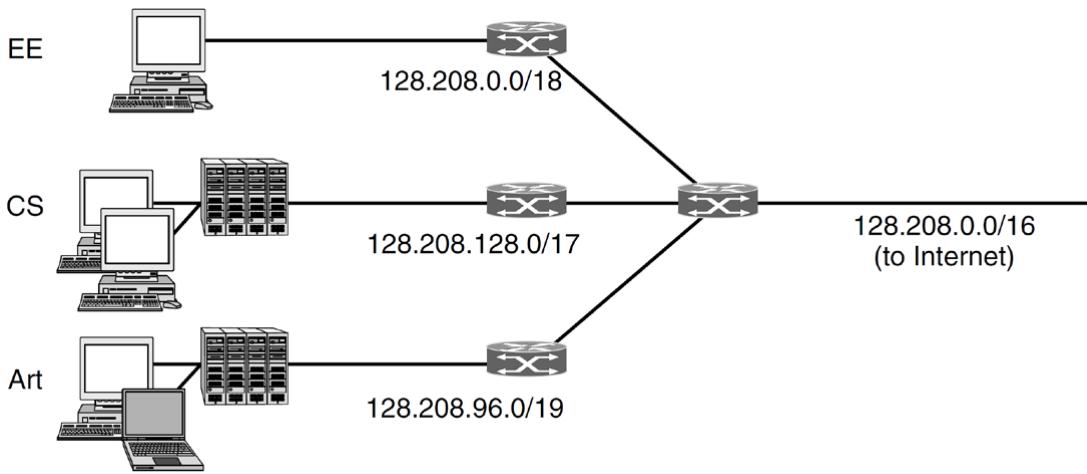


Field	Usage
Version	6
Differentiated services	6 bits for service class + 2 bits for congestion control (ECN)
Flow label	Pseudo-Virtual Circuit identifier
Payload length	Bytes after the 40 byte header
Next header	Used to specify additional headers or Protocol (TCP/UDP)
Hop limit	Same as TTL (Time To Live)
Source	16 bytes IPv6 addresses
Destination	16 bytes IPv6 addresses

Subnets

- Prefix can be used to indicate different destination within an organisation to maximise the use of their assigned IP prefix
- Subnetting
 - allows networks to be split into several parts for internal use
 - acting like a single network for external use
 - allows future changes without any external impact
 - No need to request additional IP address allocation
 - Routing on the internet does not change, only internally
 - is hierarchical
 - ISP -> organisations -> departments -> teams
 - Very similar to route aggregation among “top-level” networks
- **Subnet masks** can be written using:
 - “dotted decimal”(e.g. 255.255.255.128) or
 - “slash”notation (e.g. /25)
- Splits don’t need to be equal, but bits must be aligned to split the address into “network” and “host” portions

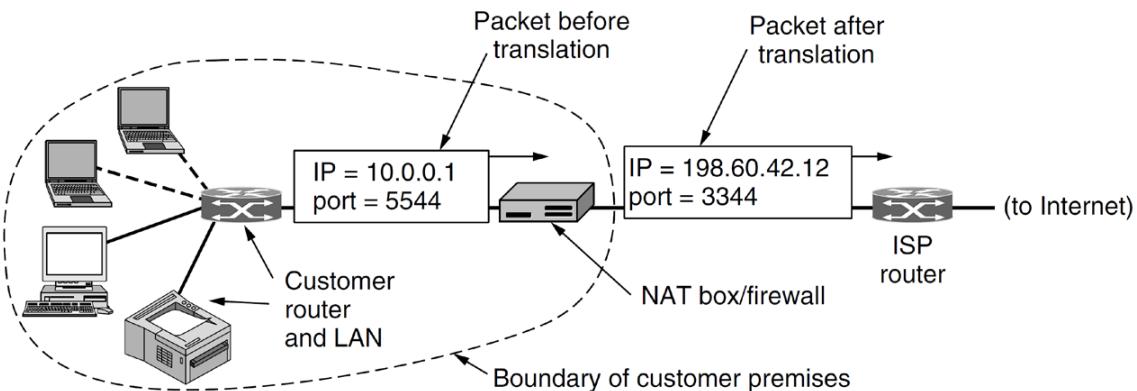
Subnet Routing



- When a packet arrives from the internet, the router can use the subnet masks (bitwise AND) to find which subnet it should send the packet to, without knowing all hosts on the subnet
 - Incoming:** `10000000.11010000.00000010.10010111`
 - CS Subnet Mask:** `11111111.11111111.10000000.00000000`
 - Result:** `10000000.11010000.00000000.00000000`
 - CS Network:** `10000000.11010000.10000000.00000000`

Network	Prefix	Network Address (binary)	Subnet Mask	Binary Subnet Mask
EE	128.208.0.0/18	<code>10000000.11010000.00000000.00000000</code>	255.255.192.0	<code>11111111.11111111.11000000.00000000</code>
CS	128.208.128.0/17	<code>10000000.11010000.10000000.00000000</code>	255.255.128.0	<code>11111111.11111111.10000000.00000000</code>
Arts	128.208.96.0/19	<code>10000000.11010000.01000000.00000000</code>	255.255.224.0	<code>11111111.11111111.11100000.00000000</code>

Network Address Translation (NAT)



- NAT **allows** an IP address to handle many more
 - an ISP with a /16 allocation can only handle 65,534 hosts
- NAT was that stop gap to solve the exhaustion of IPv4 address
- Each customer/home is assigned one "**public**" IP address
- Internally hosts/interfaces are issued **private** IP addresses
 - Internal IP addresses are used for communicating amongst hosts in the Local Area Network (LAN)
 - They must never be used on the public internet
 - When a packet is heading out of the network (to the ISP) the internal address is "translated" to the public IP address

How NAT works

- **Assumption:** TCP or UDP (+few others), in particular source and destination ports
- Send:
 - NAT box replaces TCP source address (10.x.y.z) with public IP address
 - TCP source port replaced with index of entry in NAT translation table
 - One of 65,536 entries (16 bits – same as TCP port field)
 - Each entry contains: private IP, private source port, public source port
 - IP and TCP checksums are recalculated because headers have been changed
- Receive:
 - When a packet arrives from the internet at the NAT box it looks up the destination port from the TCP header in the translation table
 - Retrieves original source port and source IP address, updates headers and checksums and sends to the internal host

Advantages

- Significant **security advantage**
 - Since packets can only be received once an outgoing connection has been created, the internal network is greatly shielded from attacks from incoming unsolicited packets
- Likely to remain in use even after IPv6 is widely deployed and there is no longer a scarcity of IP addresses

Criticisms

- **Violates IP architectural model:** every interface on the network should have a unique IP address
 - millions of interfaces connecting to the internet have 10.0.0.1
- **Breaks end-to-end connectivity**
 - an interface in the private network can only receive packets once it has sent packets out and created a table entry
- Changes internet from **connectionless to partly connection-oriented**
 - NAT maintains connection state, if it crashes all connections are lost
- **Violates layer model** by assuming nature of payload contents
 - initially only worked for TCP and UDP
- Causes problems with FTP and other protocols that **use multiple connections** in a prescribed way
- **Limits number of outgoing connection**

Fragmentation

- Fragmentation allows network gateways to meet size constraints
 - All networks have a maximum size for packets
 - 1500 bytes for Ethernet
 - 2304 bytes for 802.11 (WiFi)
 - Hosts want to transmit large packets
 - Reduces workload for them, as each packet causes overhead

- When that packet transits other networks that may not support such a large packet size

- Motivation**

- hardware
- OS
- protocols
- standards compliance
- desire to reduce transmissions due to errors
- desire for efficiency in communication channel

- Maximum Transmission Unit(MTU)**

- Maximum size for that network or protocol

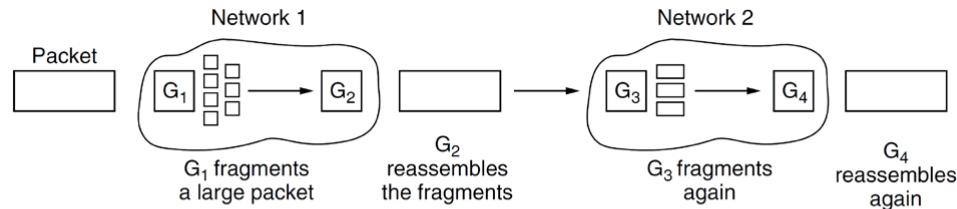
- Path MTU**

- Maximum size for the path through the network
- Min of MTU on each link / network
- Hard to determine path MTU in a connectionless network

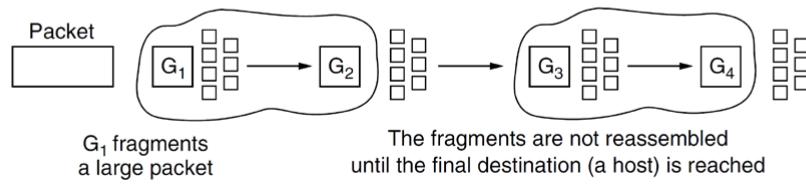
- Problem:** breaking a large packet into smaller fragments is easy, putting them back together again is hard

- Approaches**

- Transparent Fragmentation
 - reassembly is performed at next router
 - subsequent routers are unaware fragmentation has taken place
- Non-transparent Fragmentation
 - reassembly is performed at the destination host

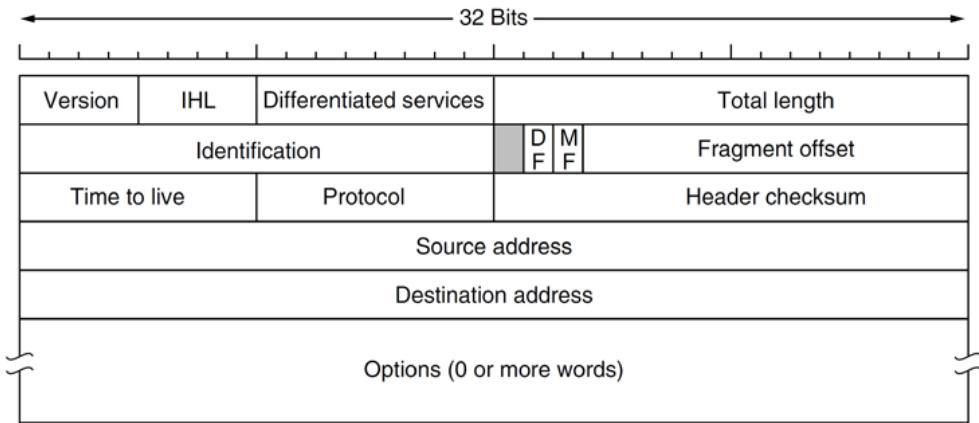


(a)



(b)

Fragmentation and IP Headers



- **Identification**: used to identify a packet
- **Flags**
 - DF=Don't Fragment
 - MF=More Fragments)
- **Fragment offset**: offset in 8 byte blocks
 - 13 bits – max offset $(2^{13} - 1) * 8 = 65,528$
 - must be on an 8 byte boundary
 - Cannot send single byte fragments, except last fragment
 - allows the receiving host to reconstruct out-of-order fragments in a buffer
- If a packet is fragmented
 - Identification stays the same for all fragments
 - MF=1 for all fragments, except the last
 - Fragment offset – appropriately set for each fragment

Example

If we have a payload of 1700 bytes. MTU=1500 bytes, ID=1:

- 1st Fragment : ID = 1, DF=0, MF=1, FO=0
- 2nd Fragment: ID = 1, DF=0, MF=0 , FO = 185 (185*8=1480 + 20 byte header = 1500)
- Minimum IP header length = 20 bytes = five 32-bit increments

Downsides

- Overhead from fragmentation is incurred from the point of fragmentation all the way to the host
 - 20 byte header for each fragment
- If a single fragment is lost the entire packet has to be resent
- Overhead on hosts in performing reassembly higher than expected

Path MTU discovery

- Each packet is sent with the DF bit set – don't fragment
- If a router cannot handle the packet size it sends an ICMP (Internet Control Message Protocol) to the sender host telling it to fragment its packets to a smaller size

IPv4 vs. IPv6 Fragmentation

- IPv4 allows for either **non-transparent** fragmentation, or path **MTU discovery**

- **Minimum MTU:** IPv4 minimum accept size 576 bytes
- IPv6 expects hosts to discover the optimal path MTU, routers will not perform fragmentation in IPv6
 - **Minimum MTU:** Pv6 minimum accept size 1280 bytes
- Caution
 - ICMP messages are sometimes dropped by networks, causing Path MTU discovery to fail
 - in such circumstances a connection will work for low volume, fails at high volume
 - if in doubt send at the minimum accept size