Pages may be referenced according to the following grid. (i.e. pag. 5.3 means bottom left corner of page 5)

1	4	7
2	5	8
3	6	9

Table of Contents

Made by

Miguel Cabral Pinto (Autor)

Nuno Batista (Contribuidor)

Tiago Silva (Contribuidor)

What's the Internet?

billions of connected computing devices:
- *hosts* = *end systems*
- running *network apps*

communication links
- fiber, copper, radio, satellite
- transmission rate: *bandwidth*

packet switches: forward packets (chunks of data)
- *routers* and *switches*

Internet: "network of networks"
- Interconnected ISPs

protocols control sending, receiving of messages
- e.g., TCP, IP, HTTP, Skype, 802.11

Internet standards
- RFC: Request for comments
- IETF: Internet Engineering Task Force

a "service view"

infrastructure that provides services to applications:
- Web, VoIP, email, games, e-commerce, social nets, ...

What's a protocol?

human protocols:
- "what's the time?"
- "I have a question"
- introductions

... specific messages sent

... specific actions taken when messages received, or other events

network protocols:
- machines rather than humans
- all communication activity in Internet governed by protocols

protocols define format, order of messages sent and received among network entities, and actions taken on message transmission, receipt

A closer look at network structure:

network edge:
- hosts: clients and servers
- servers often in data centers

network core:
- interconnected routers
- network of networks

access networks, physical media: wired, wireless communication links

Q: How to connect end systems to edge router?
- residential access nets
- institutional access networks (school, company)
- mobile access networks

Enterprise access networks (Ethernet)

- typically used in companies, universities, etc.
- 10 Mbps, 100Mbps, 1Gbps, 10Gbps transmission rates
- today, end systems typically connect into Ethernet switch

Wireless access networks

shared *wireless* access network connects end system to router
- via base station aka "access point"

wireless LANs:
- within building (100 ft.)
- 802.11b/g/n (WiFi): 11, 54, 450 Mbps transmission rate

wide-area wireless access
- provided by telco (cellular) operator, 10's km
- between 1 and 10 Mbps
- 3G, 4G: LTE, (5G!)

Host: sends *packets* of data

host sending function:
- takes application message
- breaks into smaller chunks, known as *packets*, of length *L* bits
- transmits packet into access network at *transmission rate R*
  - link transmission rate, aka link *capacity, aka link bandwidth*

$$\text{packet transmission delay} = \text{time needed to transmit } L\text{-bit packet into link} = \frac{L \text{ (bits)}}{R \text{ (bits/sec)}}$$

Physical media

- bit: propagates between transmitter/receiver pairs
- physical link: what lies between transmitter & receiver
- guided media:
  - signals propagate in solid media: copper, fiber, coax
- unguided media:
  - signals propagate freely, e.g., radio

twisted pair (TP)
- two insulated copper wires

Physical media: coax, fiber

coaxial cable:
- two concentric copper conductors
- Up to 500m
- bidirectional
- broadband:
  - multiple channels on cable
  - HFC (Hybrid fibe-coaxial)

fiber optic cable:
- glass fiber carrying light pulses, each pulse a bit
- Up to 40 km (single mode)
- high-speed operation:
  - high-speed point-to-point transmission (e.g., 10's-100's Gbps transmission rate)
- low error rate:
  - repeaters spaced far apart
  - immune to electromagnetic noise

radio

- signal carried in electromagnetic spectrum
- no physical "wire"
- bidirectional
- propagation environment effects:
  - reflection
  - obstruction by objects
  - interference

radio link types:
- terrestrial microwave
  - e.g. up to 45 Mbps channels
- LAN (e.g., WiFi)
  - 54 Mbps (and more)
- wide-area (e.g., cellular)
  - 4G cellular: ~ 10 Mbps
  - 5G cellular: up to 20 Gbps
- satellite
  - Kbps to 45Mbps channel (or multiple smaller channels)
  - 270 msec end-end delay

Submarine communications

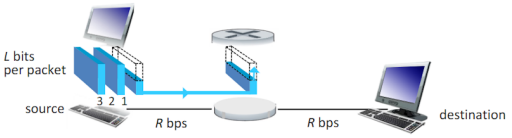
- The first submarine communications cable became operational in1858, and carried telegraphy traffic
- Modern cables use optical fiber technology to carry digital data, which includes telephone, Internet and private data traffic
- Currently 99% of the data traffic that is crossing oceans is carried by undersea cables
- Although very expensive, submarine cables transport terabits per second, while satellites typically offer only 1,000 megabits per second and display higher latency

The network core

mesh of interconnected routers
- forward packets from one router to the next, across links on path from source to destination
- each packet transmitted at full link capacity

packet-switching: hosts break application-layer messages into *packets*

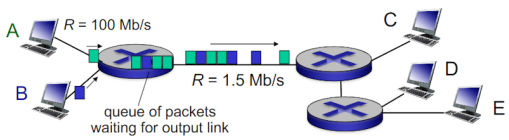
Packet-switching: store-and-forward



- takes *L*/*R* seconds to transmit (push out) *L*-bit packet into link at *R* bps
- *store and forward:* entire packet must arrive at router before it can be transmitted on next link
- end-end delay = 2*L*/*R* (assuming zero propagation delay)

one-hop numerical example:
- *L* = 7.5 Mbits
- *R* = 1.5 Mbps
- one-hop transmission delay = 5 sec

Packet Switching: queueing delay, loss

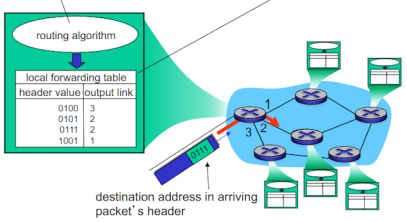


queuing and loss:
- if arrival rate (in bits) to link exceeds transmission rate of link for a period of time:
  - packets will queue, wait to be transmitted on link
  - packets can be dropped (lost) if memory (buffer) fills up
  - Bandwidth shared on demand: statistical multiplexing.

Two key network-core functions

routing: determines source-destination route taken by packets
- *routing algorithms*

forwarding: move packets from router's input to appropriate router output

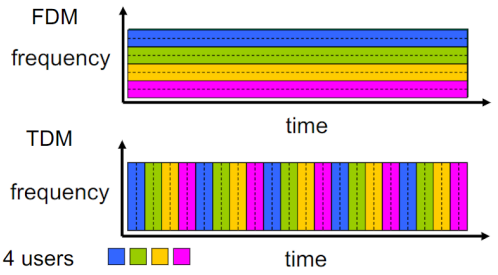


Alternative core: circuit switching

end-end resources allocated to, reserved for "call" between source & dest:

- in diagram, each link has four circuits.
  - call gets 2nd circuit in top link and 1st circuit in right link.
- dedicated resources: no sharing
  - circuit-like (guaranteed) performance
- circuit segment idle if not used by call *(no sharing)*
- commonly used in traditional telephone networks

Circuit switching: FDM versus TDM



Packet switching vs. circuit switching

packet switching allows more users to use network!

example:

- 1 Mb/s link
- each user:
  - 100 kb/s when "active"
  - active 10% of time
- *circuit-switching:*
  - 10 users
- *packet switching:*
  - with 35 users, probability > 10 active at same time is around .0004
  - with 10 or fewer active users data packets flow through the link without delay

is packet switching superior?

- Packet switching is easier and more affordable than circuit switching
- Packet switching is great for "bursty" data
  - resource sharing
  - simpler, no call setup (simpler infrastructure)
- excessive congestion possible: packet delay and loss
  - end-to-end delays are unpredictable (due primarily to queuing delays)
  - protocols needed for reliable data transfer, congestion control
- *Q:* How to provide circuit-like behavior?
  - bandwidth guarantees needed for audio/video apps
  - still an unsolved problem (later in the course)

Internet structure: network of networks

- End systems connect to Internet via access ISPs (Internet Service Providers)
  - residential, company and university ISPs
- Access ISPs in turn must be interconnected.
  - so that any two hosts can send packets to each other
- Resulting network of networks is very complex
  - evolution was driven by economics and national policies

Question: given *millions* of access ISPs, how to connect them together?

Option: connect each access ISP to every other access ISP?

connecting each access ISP to each other directly *doesn't scale:* $O(N^2)$ connections.

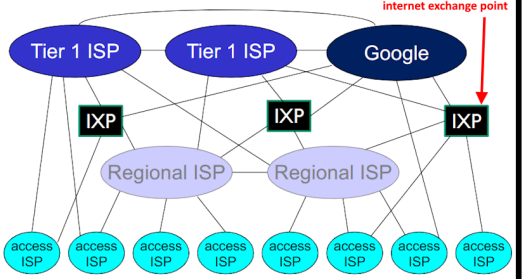
Option: connect each access ISP to one global transit ISP?

Customer and provider ISPs have economic agreement.

But if one global ISP is viable business, there will be competitors …. which must be interconnected

… and regional networks may arise to connect access nets to ISPs

… and content provider networks (e.g., Google, Microsoft, Akamai) may run their own network, to bring services, content close to end users



at center: small # of well-connected large networks

- "tier-1" commercial ISPs (e.g., Level 3, Sprint, AT&T, NTT), national & international coverage
- content provider network (e.g., Google): private network that connects it data centers to Internet, often bypassing tier-1, regional ISPs

Protocol "layers"

Networks are complex, with many "pieces":

- hosts
- routers
- links of various media
- applications
- protocols
- hardware, software

Internet (TCP/IP) protocol stack

- *application:* supporting network applications
  - FTP, SMTP, HTTP
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.111 (WiFi), PPP
- *physical:* bits "on the wire"

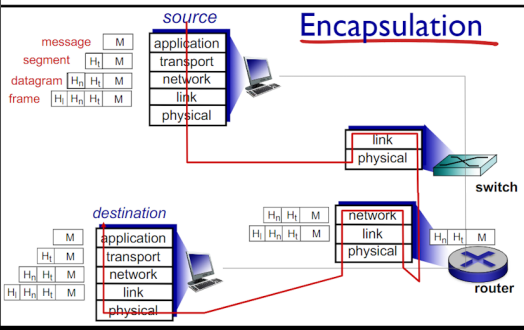
application
transport
network
link
physical

ISO/OSI reference model

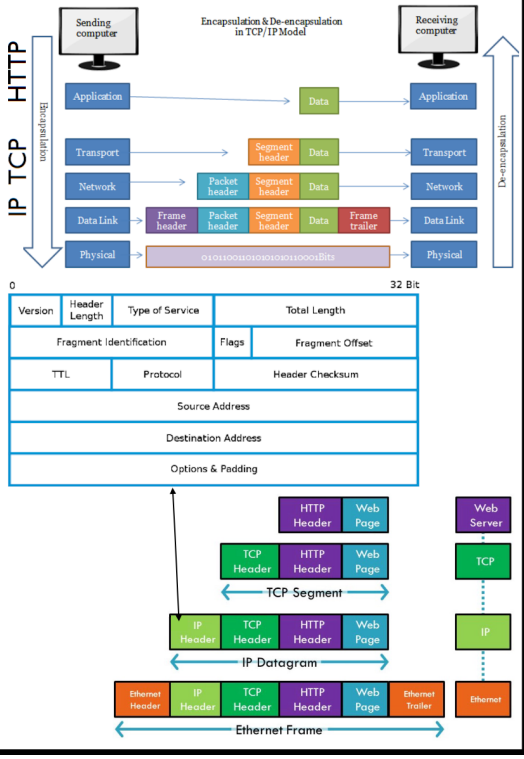
- *presentation:* allow applications to interpret meaning of data, e.g., encryption, compression, machine-specific conventions
- *session:* synchronization, checkpointing, recovery of data exchange
- Internet stack "missing" these layers!
  - these services, *if needed,* must be implemented in application

application
presentation
session
transport
network
link
physical

Encapsulation



Application layer Encapsulation (yes, again!)



Creating a network app

write programs that:

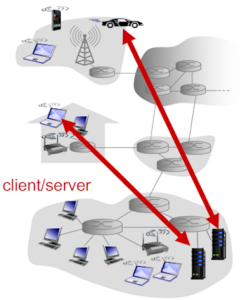
- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software

no need to write software for network-core devices

- network-core devices do not run user applications (operate at the network-layer and below)
- applications on end systems allows for rapid app development, propagation

Application architectures

Client-server architecture



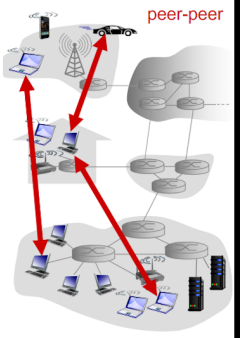
server:

- always-on host
- permanent IP address
- data centers for scaling

clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other
- Examples: Web, FTP, SSH, e-mail

P2P architecture



- *no* always-on server
- arbitrary end systems communicate directly
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management
- Example: BitTorrent, Skype, IPTV, …
- Also, some applications use *hybrid architectures* (e.g. in messaging)

Processes communicating

process: program running within a host

- within same host, two processes communicate using inter-process communication (defined by OS)
- processes in different hosts communicate by exchanging messages.
- IP communications may also take place within same host (via loopback interface, 127.0.0.1)

clients, servers

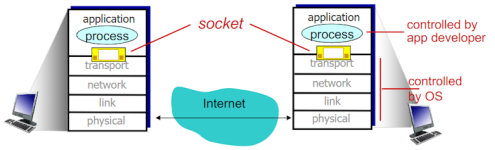
client process: process that initiates communication

server process: process that waits to be contacted

- aside: applications with P2P architectures have client processes & server processes
- In some applications a process can be both a client and a server

Sockets

- process sends/receives messages to/from its socket
- A socket is the software interface between the process and the computer network (between the application layer and the transport layer)
- Is also referred to as the Application Programming Interface (API) between the application and the network
- The application chooses the transport protocol (e.g. UDP or TCP) to use the transport-layer services provided by the protocol



Addressing processes

- to receive messages, processes must have *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?
  - *A:* no, *many* processes can be running on same host

- *identifier* includes both IP address and port numbers associated with process (the socket) on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80

App-layer protocol defines

types of messages exchanged,
- e.g., request, response

message syntax:
- what fields in messages & how fields are delineated

message semantics
- meaning of information in fields

rules for when and how processes send & respond to messages

open protocols:
- defined in RFCs
- allows for interoperability
- e.g., HTTP (HyperText Transfer Protocol), SMTP (Simple Mail Transfer Protocol)

proprietary protocols:
- e.g., Skype

What transport service does an app need?

data integrity
- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

timing
- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

throughput
- some apps (e.g., multimedia) require minimum amount of throughput to be "effective" (e.g. multimedia)
- other apps ("elastic apps") make use of whatever throughput they get (e.g. e-mail, web, file transfer)

security
- encryption, data integrity, …

Transport service requirements:

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
"real-time" audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	
interactive games	loss-tolerant	few kbps up	yes, few secs
text messaging	no loss	elastic	yes, 100's msec yes and no

Internet transport protocols services

TCP (protocol) service:
- *reliable transport* between sending and receiving process (data delivered without errors and in the correct order)
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantee, security
- *connection-oriented:* setup required between client and

UDP (protocol) service:
- *A "no-frills" lightweight transport protocol with minimal services*
- *unreliable data transfer* between sending and receiving process
- *does not provide:* reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup

A programmer need to decide to use UDP or TCP…

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

Securing TCP

TCP & UDP
- no encryption
- cleartext passwds sent into socket traverse Internet in cleartext

SSL
- provides encrypted TCP connection
- data integrity
- end-point authentication

SSL is at app layer
- apps use SSL libraries, that "talk" to TCP

SSL socket API
- cleartext passwords sent into socket traverse Internet encrypted
- more about network security later in the course!..

Web and HTTP

web page consists of *objects*

object can be HTML file, JPEG image, Java applet, audio file,…

web page consists of *base HTML-file* which includes *several referenced objects*

each object is addressable by a *URL*, e.g.,

HTTP overview

HTTP: hypertext transfer protocol
- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, (using HTTP protocol) and "displays" Web objects
  - *server:* Web server sends (using HTTP protocol) objects in response to requests

HTTP is "stateless"
- server maintains no information about past client requests

uses TCP:
- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

protocols that maintain "state" are more complex!
- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP connections

persistent HTTP
- multiple objects can be sent over single TCP connection between client, server
- (is the default mode)

non-persistent HTTP
- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

Non-persistent HTTP

suppose user enters URL: `www.someSchool.edu/someDepartment/home.index` (contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

Non-persistent HTTP: response time

RTT (Round-trip delay): time for a small packet to travel from client to server and back (considering also delays)

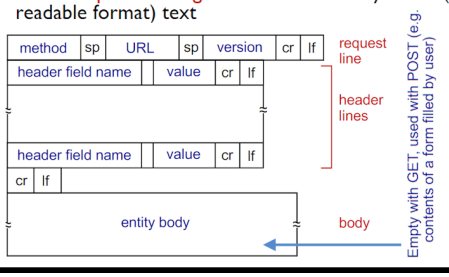
RTT may be measured using a "ping": a command-line tool that bounces a request off a server and calculates the time taken to reach a user device

HTTP response time:
- one RTT to initiate TCP connection
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time
- non-persistent HTTP response time = 2RTT+ file transmission time

non-persistent HTTP issues:
- requires 2 RTTs per object
- OS overhead for *each* TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

Persistent HTTP
- *server* leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects
- An entire web page (or various) may be sent over a *single* (persistent) TCP connection

HTTP request message
- two types of HTTP messages: *request, response*
- HTTP request message: written in ordinary ASCII (human-readable format) text



Uploading form input

POST method:
- web page often includes form input
- input is uploaded to server in entity body

URL method:
- uses GET method
- input is uploaded in URL field of request line:

Method types

HTTP/1.0:
- GET
- POST
- HEAD
  - asks server to leave requested object out of response (frequently used for debugging)

HTTP/1.1:
- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

HTTP response status codes
- status code appears in 1st line in server-to-client response message.
- some sample codes:

`200 OK`
- request succeeded, requested object later in this msg

`301 Moved Permanently`
- requested object moved, new location specified later in this msg (using the Location: header)

`400 Bad Request`
- request msg not understood by server

`404 Not Found`
- requested document not found on this server

`505 HTTP Version Not Supported`

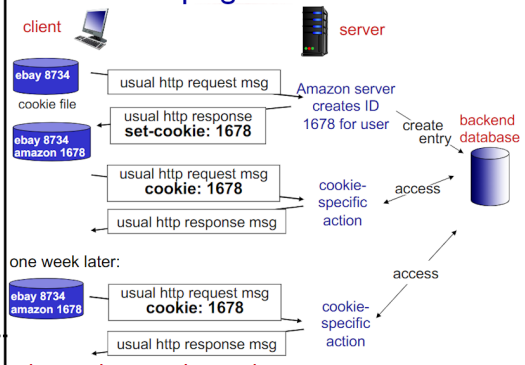
User-server state: cookies

many Web sites use cookies

four components:
1) cookie header line of HTTP *response* message (set-cookie)
2) cookie header line in next HTTP *request* message (cookie)
3) cookie file kept on user's host, managed by user's browser
4) back-end database at Web site

example:
- Susan always access Internet from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies: keeping "state"



what cookies can be used for:
- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

aside

cookies and privacy:
- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

how to keep "state":
- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies create a "user session layer" on to of stateless HTTP
- cookies may be considered an invasion of privacy!

Web caches (proxy server)

goal: satisfy client request without involving origin server
- user sets browser: Web accesses via cache (proxy)
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client

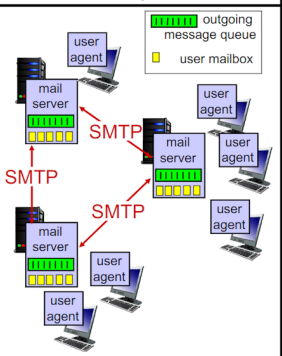
More about Web caching

- cache acts as both client and server
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)
- Example: Squid Web Cache

why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link
- Increasingly important in the Internet because of CDNs (Content Distribution Networks, e.g. Google, Akamai, etc).

Electronic mail



Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server

mail servers:

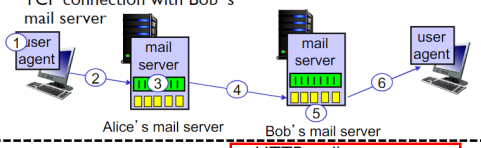
- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages
- *SMTP protocol* between mail servers to send email messages
  - client: sending mail server
  - "server": receiving mail server
  - Every mail server is an SMTP client and server

Electronic Mail: SMTP [RFC 2821]

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

Scenario: Alice sends message to Bob

1) Alice uses UA to compose message "to" `bob@someschool.edu`
2) Alice's UA sends message to her mail server; message placed in message queue
3) client side of SMTP opens TCP connection with Bob's mail server
4) SMTP client sends Alice's message over the TCP connection
5) Bob's mail server places the message in Bob's mailbox
6) Bob invokes his user agent to read message



- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses `CRLF.CRLF` to determine end of message

comparison with HTTP:

- HTTP: pull
- SMTP: push
- *both* have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response message
- SMTP: multiple objects sent in multipart message (e.g. body of message, images, videos, etc.)

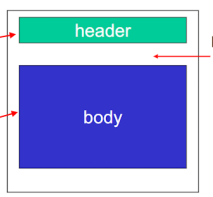
Mail message format

SMTP: protocol for exchanging email messages

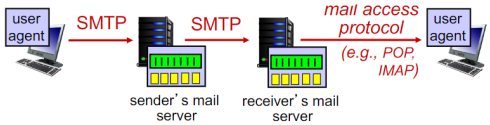
RFC 822: standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:

  different from SMTP MAIL FROM, RCPT TO: commands! (so called "envelope" headers)
- Body: the "message"
  - ASCII characters only



Mail access protocols



SMTP: delivery/storage to receiver's server

mail access protocol: retrieval from server

- POP: Post Office Protocol [RFC 1939]: authorization, download
- IMAP: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored messages on server
- HTTP (web mail): GMail, Hotmail, Yahoo! Mail, etc.

POP3 protocol

authorization phase

- client commands:
  - `user:` declare username
  - `pass:` password
- server responses
  - `+OK`
  - `-ERR`

transaction phase, client:

- `list:` list message numbers
- `retr:` retrieve message by number
- `dele:` delete
- `quit`

more about POP3

- previous example uses POP3 "download and delete" mode
  - Bob cannot re-read e-mail if he changes client
- POP3 "download-and-keep": copies of messages on different clients
- POP3 is *stateless* across sessions

IMAP

- keeps all messages in one place: at server
- allows user to organize messages in folders
- keeps *user state* across sessions:
  - names of folders and mappings between message IDs and folder name

DNS: domain name system

people: many identifiers:

- SSN, name, passport #

Internet hosts, routers:

- IP address (32 bit) - used for addressing datagrams
- "name", e.g., www.yahoo.com - used by humans

Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)
- core Internet function, implemented as application-layer protocol

services, structure

DNS services

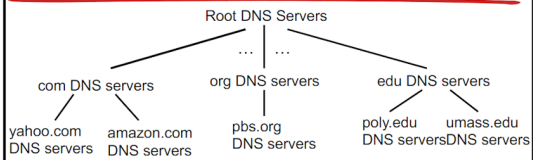
- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - Example: replicated Web servers (many IP addresses correspond to one name)

why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance (updates)

A: doesn't scale!

DNS: a distributed, hierarchical database



client wants IP for www.amazon.com; example:

- client queries root server to find .com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: root name servers

- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server

TLD, authoritative servers

top-level domain (TLD) servers:

- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
- Network Solutions maintains servers for .com TLD
- Educause for .edu TLD

authoritative DNS servers:

- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

Local DNS name server

does not strictly belong to hierarchy

each ISP (residential ISP, company, university) has one

- also called "default name server"

when host makes DNS query, query is sent to its local DNS server

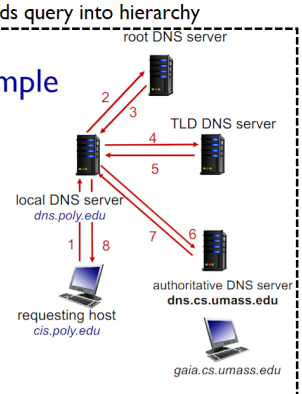
- has local cache of recent name-to-address translation pairs (but may be out of date!)
- acts as proxy, forwards query into hierarchy

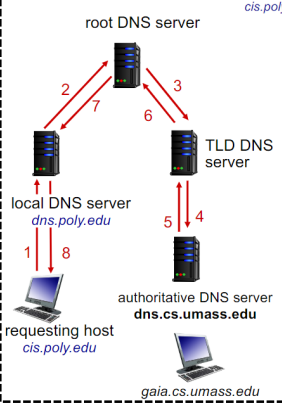
DNS name resolution example

- host at cis.poly.edu wants IP address for gaia.cs.umass.edu

iterated query:

- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"





recursive query:

- puts burden of name resolution on contacted name server

DNS: caching, updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- cached entries may be *out-of-date* (best effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire
- update/notify mechanisms proposed IETF standard
  - RFC 2136

DNS records

DNS: distributed database storing resource records (RR)

RR format: `(name, value, type, ttl)`

type=A

- `name` is hostname
- `value` is IP address

type=NS

- `name` is domain (e.g., foo.com)
- `value` is hostname of authoritative name server for this domain

type=CNAME

- `name` is alias name for some "canonical" (the real) name
- `value` is canonical name

type=MX

- `value` is name of mailserver associated with `name`

type=PTR

- `IP to name` translation

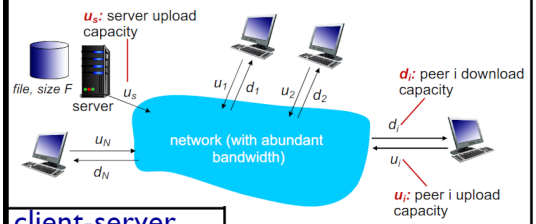
Pure P2P architecture

- *no* need to rely on always-on server
- arbitrary end systems communicate directly

examples: • VoIP (Skype)

- peers are intermittently connected and change IP addresses
- file distribution (BitTorrent)

File distribution: client-server vs P2P



client-server

server transmission: must sequentially send (upload) *N* file copies:

- time to send one copy: F/u_s
- time to send *N* copies: NF/u_s

client: each client must download file copy

- d_{min} = min client download rate
- min client download time: F/d_{min}

time to distribute F to N clients using client-server approach $D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$

increases linearly in N (number of peers)

P2P

server transmission: must upload at least one copy (at the beginning only the server has the file)

- time to send one copy: F/u_s

client: each client must download file copy

- min client download time: F/d_{min}

clients: as aggregate must download *NF* bits

- max upload rate (limiting max download rate) is $u_s + \Sigma u_i$

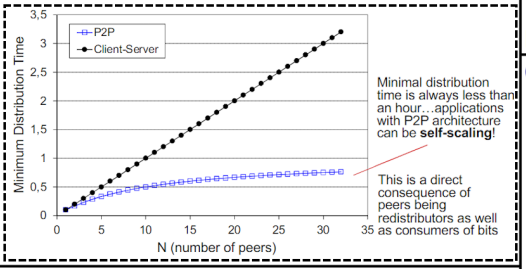
time to distribute F to N clients using P2P approach $D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$

increases linearly in *N* …

… but so does this, as each peer brings service capacity

(lower bound is achievable with a scheme where each peer can redistribute a bit as soon as it receives it, and a good approximation when redistributing chunks of the file)

client upload rate = *u*, F/u = 1 hour (a peer can transmit the file in 1 hour), $u_s = 10u$ (server transmission rate is 10 x the peer upload rate), $d_{min} \geq u_s$



Minimal distribution time is always less than an hour…applications with P2P architecture can be **self-scaling**!

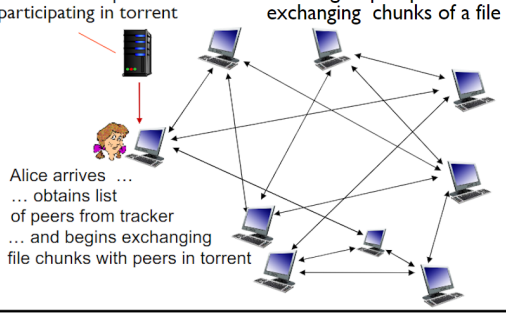
This is a direct consequence of peers being redistributors as well as consumers of bits

P2P file distribution: BitTorrent

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks

tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file



Alice arrives …
… obtains list of peers from tracker
… and begins exchanging file chunks with peers in torrent

Video Streaming and CDNs: context

video traffic: major consumer of Internet bandwidth

- Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
- ~1B YouTube users, ~75M Netflix users

challenge: scale - how to reach ~1B users?

- single mega-video server won't work (why?)

challenge: heterogeneity

- different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)

solution: **distributed, application-level infrastructure**

Multimedia: video

video: sequence of images displayed at constant rate

- e.g., 24 images/sec

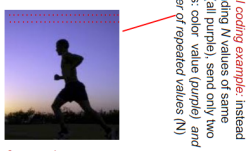
digital image: array of pixels

- each pixel represented by bits

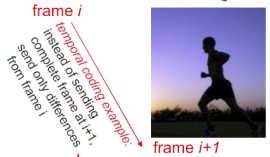
coding: use redundancy *within* and *between* images to decrease # bits used to encode image

- spatial (within image)
- temporal (from one image to next)

- CBR: (constant bit rate): video encoding rate fixed
- VBR: (variable bit rate): video encoding rate changes as amount of spatial, temporal coding changes



spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (purple) and number of repeated values (*N*)



- examples:
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (often used in Internet, < 1 Mbps)

Streaming multimedia: DASH

DASH: Dynamic, Adaptive Streaming over HTTP

server:

- divides video file into multiple chunks
- each chunk stored, encoded at different rates
- *manifest file:* provides URLs for different chunks

client:

- periodically measures server-to-client bandwidth
- consulting manifest, requests one chunk at a time
  - chooses maximum coding rate sustainable given current bandwidth
  - can choose different coding rates at different points in time (depending on available bandwidth at time)

"intelligence" at client: client determines

- *when* to request chunk (so that buffer starvation, or overflow does not occur)
- *what encoding rate* to request (higher quality when more bandwidth available)
- *where* to request chunk (can request from URL server that is "close" to client or has high available bandwidth)

Examples: MPEG-DASH in YouTube and Netflix

Content distribution networks

challenge: how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users around the world?

Example: a billion hours of video watched on YouTube every day!

Option 1: single, large "mega-server"

- single point of failure
- point of network congestion
- long path to distant clients (bottleneck links)
- multiple copies of video (e.g. popular videos) sent over outgoing link, which Internet video company has to pay

..quite simply: this solution *doesn't scale!*

option 2: store/serve multiple copies of videos at multiple geographically distributed sites *(CDN)*. Two different server placement strategies:

- *enter deep:* push CDN servers deep into many access networks
  - get close to end users
  - used by Akamai, 1700 locations
- *bring home:* smaller number (10's) of larger clusters in POPs near (but not within) access networks (Tier-1 ISPs)
  - used by Limelight

Content Distribution (or Delivery) Networks (CDN) may be *private* or *third-party*

A *private* CDN is owned by the content provider, e.g. Google's CDN distributes YouTube Videos

A *third-party* CDN distributed content on behalf of multiple content-providers, e.g. Limelight or Akamai for Netflix and Hulu

CDN: stores copies of content at CDN nodes

- e.g. Netflix stores copies of MadMen

subscriber requests content from CDN

- directed to nearby copy, retrieves content
- may choose different copy if network path congested

FIM DE APP LAYER!

Transport Layer

Transport services and protocols

provide *logical (rather than physical) communication* between app processes running on different hosts

- *From an application's perspective, it is as if the hosts running the processes were directly connected!*

application processes use the logical communication provided by the transport layer to exchange messages

- *Application processes do not have to worry with the details of the physical infrastructure used to carry the messages*

transport protocols run in end systems (not in network routers)

- send side: breaks app messages into *segments*, passes to network layer
- rcv side: reassembles segments into messages, passes to app layer
- network routers act only on the network-layer fields of the datagram

more than one transport protocol available to apps

- Internet: TCP and UDP (different services to applications)

Transport vs. network layer

network layer: logical communication between **hosts**

transport layer: logical communication between **processes**

- relies on, enhances, network layer services

Internet transport-layer protocols

reliable, in-order delivery (TCP)

- congestion control
- flow control
- connection-oriented
- connection setup

unreliable, unordered delivery: UDP

- connecionless
- no-frills extension of "best-effort" IP

services not available:

- delay guarantees
- bandwidth guarantees

Do not get confused!

Applications send **messages**

Transport layer packets are referred to as **segments**

Network layer (IP) packets are referred to as **datagrams** or **packets**

Data link packets are referred to as **frames**

Although, in the Internet literature (e.g. in RFCs) the term datagram may refer to IP packets but also to UDP packets!

A few words about the network layer

Network layer has a name: IP (Internet Protocol) layer

IP service model is "best effort", no guarantees of:

- segment delivery
- orderly delivery of segments
- integrity of the data in segments

UDP and TCP extend host-to-host delivery of IP to process-to-process delivery of the transport layer:

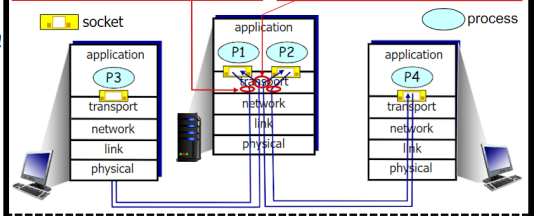
- UDP only provides process-to-process data delivery and error checking
- TCP provides a reliable data transport service between processes

Multiplexing/demultiplexing

A process (which is part of an application) can have one or more sockets, "doors" though which they exchange data with the network

multiplexing at sender: handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver: use header info to deliver received segments to correct socket

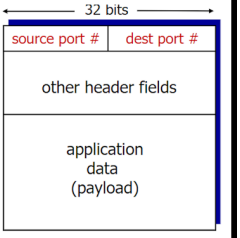


How demultiplexing works

host receives IP datagrams

- each datagram has source IP address, destination IP address
- each datagram carries one transport-layer segment
- each segment has source, destination port number

host uses *IP addresses & port numbers* to direct segment to appropriate socket (and process)



TCP/UDP segment format (common structure)

Connection-oriented demux

UDP socket identified by 2-tupple:

- dest IP address
- dest port number

TCP socket identified by 4-tuple:

- source IP address
- source port number
- dest IP address
- dest port number

demux: receiver uses all four values to direct segment to appropriate socket

server host may support many simultaneous TCP sockets:

- each socket identified by its own 4-tuple

web servers have different sockets for each connecting client

- example: non-persistent HTTP will have different socket for each request

UDP: User Datagram Protocol [RFC 768]

- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out-of-order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others
- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP
- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

Why UDP?

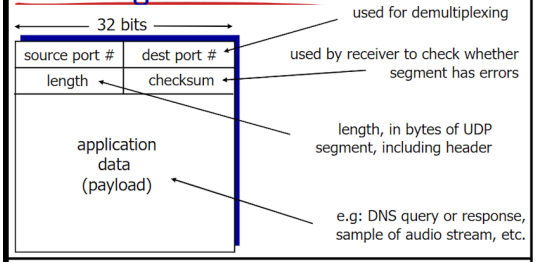
Isn't TCP always preferable, since TCP provides a reliable data transfer service, while UDP does not? No!

- Many applications are better suited for UDP:
  - Finer application-level control over what data is sent, and when: UDP sends data immediately, TCP implements congestion control
  - Some applications tolerate some loss, while TCP uses retransmission
  - Additional functionality may be implemented at the application
- No connection establishment:
  - No additional delay prior to start communicating
  - Example: DNS uses UDP
- No connection state:
  - Server using UDP may support many more clients than when using TCP
- Small packet header overhead:
  - Less overhead in communications

Examples of applications using UDP:

- RIP updates: updates of routing tables are periodic, thus lost updates will be replaced by more recent information
- Network management using SNMP: UDP is better in congested networks
- DNS: IP and name resolutions are faster using UDP, without TCP's connection establishment delays

UDP: segment header



UDP checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment (e.g. errors due to noisy links, or while stored in a router)

sender:

- treat segment contents, including header fields, as sequence of 16-bit integers
- checksum: addition (one's complement sum) of segment contents, with any overflow wrapper around
- sender puts checksum value into UDP checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value
  - NO - error detected
  - YES - no error detected

example: add two 16-bit integers

```
                1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
                1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
wraparound    1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
sum             1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
checksum        0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1
```

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

Is error checking in UDP useful?

IP is supposed to run over just about any layer-2 protocol

There is no guarantee that all the links between source and destination provide error checking

Errors may also occur while segment is stored in a router's memory

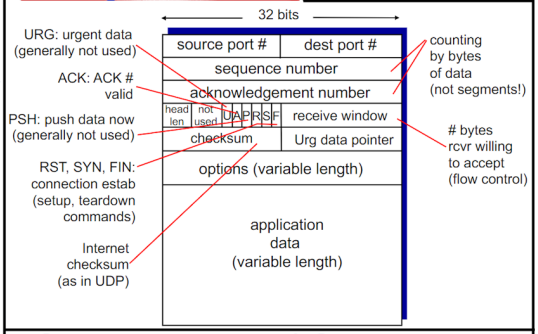
Neither link-by-link reliability nor in-memory error detection is guaranteed: UDP provides *error detection at the transport layer, on and end-to-end basis*

UDP does nothing to recover from an error, segment may be discarded or passed to the application with a warning

TCP: Overview RFCs: 793,1122,1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte stream:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver
  - Sliding-window protocol

TCP segment structure



TCP seq. numbers, ACKs

sequence numbers:

- byte stream "number" of first byte in segment's data

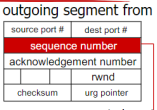
acknowledgements:

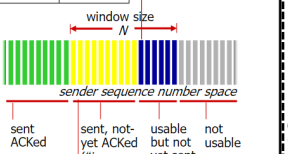
- seq # of next byte expected from other side
- cumulative ACK (acknowledges bytes up to the first missing byte in the stream)

Q: how receiver handles out-of-order segments

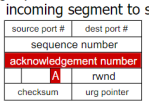
- A: TCP spec doesn't say, - up to implementor

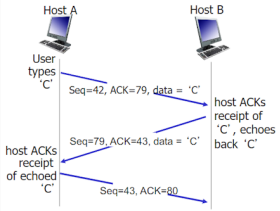
outgoing segment from sender





incoming segment to sender





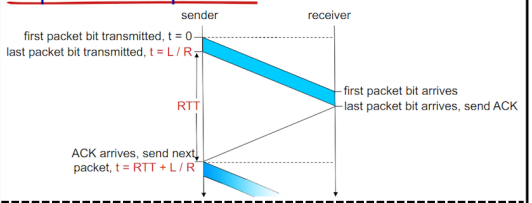
simple telnet scenario

Acknowledgements contain the sequence number of next byte expected from other side

Sequence number contain the byte stream "number" of first byte in segment's data

"Piggybacking": Using data segments to transport ACKs

Stop-and-wait operation



Example: Considering R=1Gbps (10^9 bps), L=1000 bytes (packet size) and an RTT of 30 msec

Sender is busy only 2.7 hundredths of one percent of the time!

$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

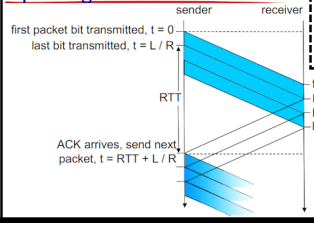
Pipelined protocols

pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged packets

- An appropriate range of sequence numbers must be used
- buffering at sender and/or receiver is required

two generic forms of pipelined protocols: *go-Back-N, selective repeat*

Pipelining: increased utilization



3-packet pipelining increases utilization by a factor of 3!

$$U_{sender} = \frac{3L/R}{RTT + L/R} = \frac{.0024}{30.008} = 0.00081$$

Pipelined protocols: overview

Go-back-N:

- sender can have up to N unacked packets in pipeline
- receiver only sends *cumulative ack*
  - doesn't ack packet if there's a gap
- sender has timer for oldest unacked packet
  - when timer expires, retransmit *all* unacked packets

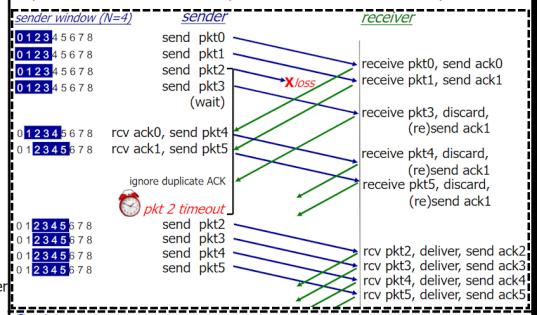
Selective Repeat:

- sender can have up to N unacked packets in pipeline
- rcvr sends *individual ack* for each packet
- sender maintains timer for each unacked packet
  - when timer expires, retransmit only that unacked packet

Go-Back-N: sender

- k-bit seq # in pkt header
- "window" of up to N, consecutive unack'ed pkts allowed

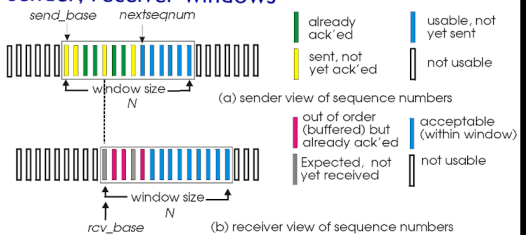
- ACK(n): ACKs all pkts up to, including seq # n - *"cumulative ACK"*
  - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt
- *timeout(n):* retransmit packet n and all higher seq # pkts in window
- Simplicity: receiver need not buffer *any* out-of-order packets, just need to maintain seq number of next in-order packet

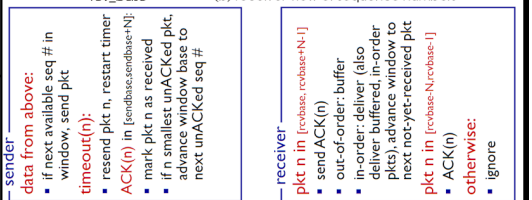


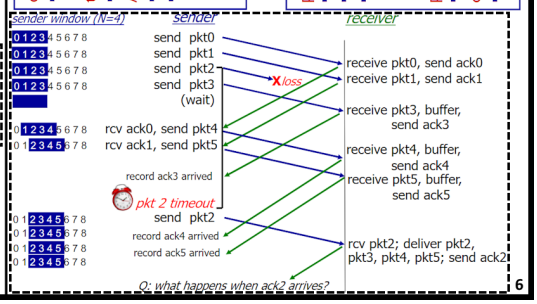
Selective repeat

- receiver *individually* acknowledges all correctly received pkts
  - buffers pkts, as needed, for eventual in-order delivery to upper layer
- sender only resends pkts for which ACK not received
  - sender timer for *each* unACKed pkt
- sender window
  - *N* consecutive seq #'s
  - limits seq #s of sent, unACKed pkts

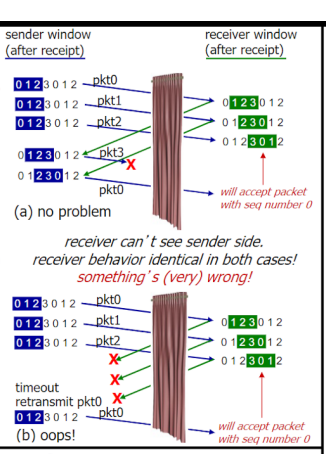
sender, receiver windows







Selective repeat: dilemma



example:

- seq #' s: 0, 1, 2, 3
- window size=3
- receiver sees no difference in two scenarios!
- duplicate data accepted as new in (b)

Q: what relationship between seq # size and window size to avoid problem in (b)?

A: *window size must be less than or equal to half the seq #'s space*

Window size and performance

The window size relates directly with the performance of TCP connections. Considering:

W: Window size (in bytes)

C: Transmission Rate (in bps)

D: Propagation delay (in s)

S: Normalized throughput

Bits transmitted before a confirmation may arrive: 2CD

Bytes transmitted: 2CD/8 = CD/4

Normalized throughput is 1 if window size is larger than number of bytes transmitted before a confirmation may be received by sender:

S = 1 , for W > CD/4

Otherwise normalized throughput is obtained by dividing windows size by the number of bytes that could be transmitted before confirmation arrives:

S = 4W/(CD) , for W < CD/4

TCP Options

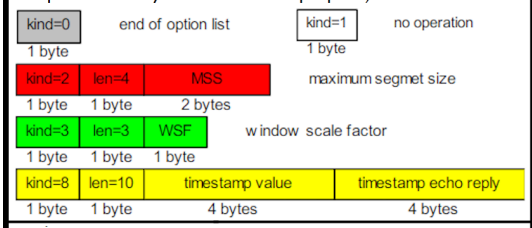
Maximum Segment Size (MSS): may be used during the connection establishment phase to negotiate the maximum size of the TCP segments that entity can receive (in bytes)

Window Scale Factor (WSF): may be used during the connection establishment phase to set larger window sizes (required for high bandwidth network links). If F is the value stored in this field (F<15), window size is multiplied by 2^F

Timestamp: this option is set in data segments and copied to confirmation segments, and allows for continuous monitoring of the RTT between the client and server

TCP Option fields

- Kind: identifies the option.
- Length: total size of the option field
- Options are aligned in multiple 4-byte fields (the option "no operation" may be used for this purpose)



TCP round trip time, timeout

Q: how to set TCP timeout value?

- longer than RTT
  - but RTT varies
- *too short:* premature timeout, unnecessary retransmissions
- *too long:* slow reaction to segment loss

Q: how to estimate RTT?

- `SampleRTT`: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- `SampleRTT` will vary, want estimated RTT "smoother"
  - average several *recent* measurements, not just current `SampleRTT`

```
EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT
```

- exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: α = 0.125 (RFC 6298)
- Puts more weight on recent samples than on old samples
- timeout interval: `EstimatedRTT` plus "safety margin"
  - large variation in `EstimatedRTT` -> larger safety margin
- estimate SampleRTT deviation from EstimatedRTT:

```
DevRTT = (1-β)*DevRTT + β*|SampleRTT-EstimatedRTT|
```

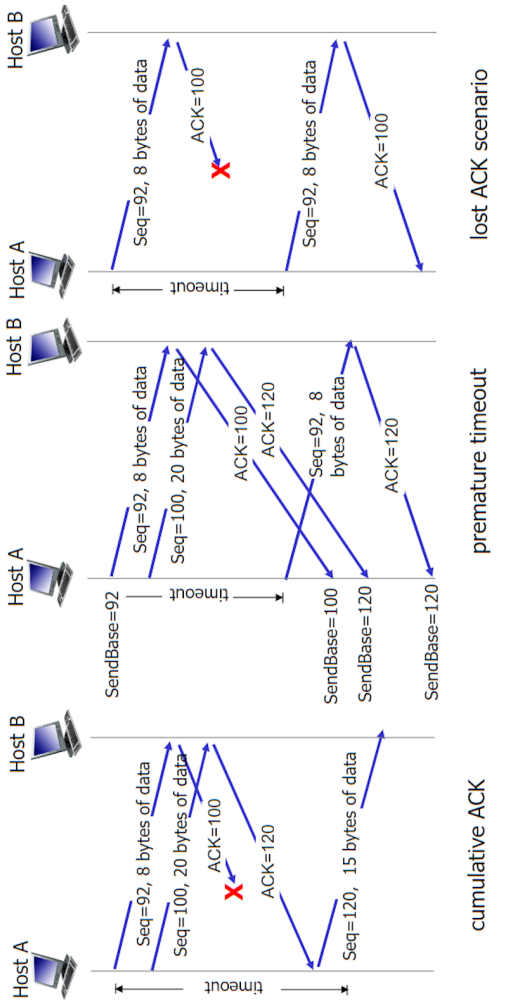
(typically, β = 0.25)

DevRTT is small if SampleRTT values have little fluctuation, large otherwise

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

estimated RTT | "safety margin"

TCP: retransmission scenarios



TCP ACK generation [RFC 1122, RFC 2581]

event at receiver	*TCP receiver action*
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately send single cumulative ACK, ACKing both in-order segments
arrival of out-of-order segment higher-than-expect seq. # . Gap detected	immediately send *duplicate ACK*, indicating seq. # of next expected byte
arrival of segment that partially or completely fills gap	immediate send ACK, provided that segment starts at lower end of gap

TCP fast retransmit

time-out period often relatively long:

- long delay before resending lost packet

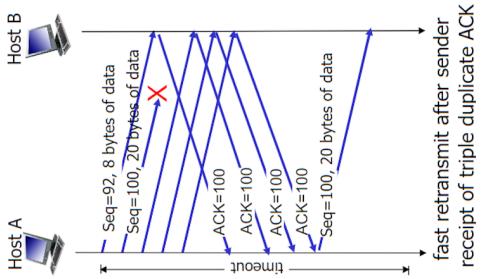
detect lost segments via duplicate ACKs:

- sender often sends many segments back-to-back
- if segment is lost, there will likely be many duplicate ACKs

TCP fast retransmit

if sender receives 3 ACKs for same data ("triple duplicate ACKs"), resend unacked segment with smallest seq #

- likely that unacked segment lost, so don't bother wait for timeout



Is TCP Go-Back-N or Selective Repeat?

TCP ACKs are cumulative

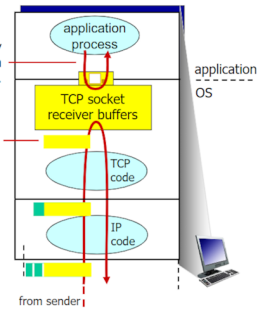
Correctly received but out-of-order segments are not individually ACKed by the receiver

But many implementations buffer such segments

Retransmissions may be of only a single lost segments if subsequent ACKs arrive before timeout

TCP's error recovery is best categorized as a **hybrid** of GBN and SR!

TCP flow control



flow control

receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

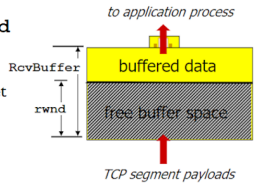
Flow control it's a speed-matching service

receiver "advertises" free buffer space by including `rwnd` value in TCP header of receiver-to-sender segments

- `RcvBuffer` size set via socket options (typical default is 4096 bytes)
- many operating systems autoadjust `RcvBuffer`

sender limits amount of unacked ("in-flight") data to receiver's `rwnd` value

guarantees receive buffer will not overflow



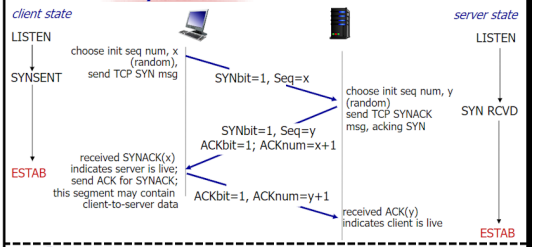
receiver-side buffering

Connection Management

before exchanging data, sender/receiver "handshake":

- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters

TCP 3-way handshake



TCP: closing a connection

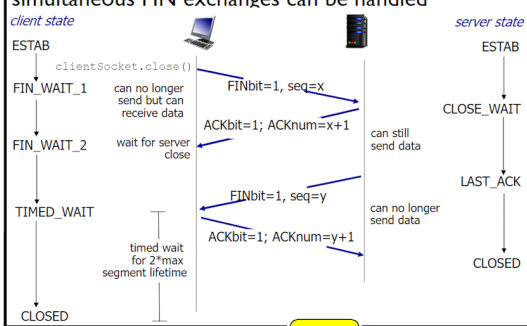
client, server each close their side of connection

- send TCP segment with FIN bit = 1

respond to received FIN with ACK

- on receiving FIN, ACK can be combined with own FIN

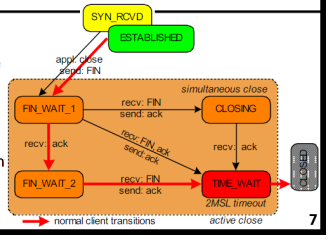
simultaneous FIN exchanges can be handled



2MSL timer

Maximum segment lifetime (MSL) is the time a TCP segment can exist in the internetwork system

The purpose of TIMED_WAIT is to prevent delayed packets from one connection being accepted by a later connection



Principles of congestion control

congestion:

- informally: "too many sources sending too much data too fast for *network* to handle"
- different from flow control!
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- a top-10 problem!

Approaches

We can distinguish among congestion-control approaches by whether the network layer provides any explicit assistance :

End-to-end congestion control: no explicit support from the network layer, congestion is inferred from network behavior

- Used in TCP, sender limits the rate at which it sends traffic as a function of *perceived network congestion*
- In case of loss (timeout) or triple ACKs, TCP decreases its (congestion) window size accordingly
- May also consider increasing RTT as indicators of increased congestion

Network-assisted congestion control

- Routers provide explicit feedback
- As used in IBM SNA, DEC DECnet, ATB ABR

Congestion and receiver window

The TCP congestion-control mechanism operating at the sender keeps track of an additional variable: the congestion window (cwnd)

The congestion window imposes a constraint on the rate at which TCP sender can send data into the network

At any given time, the amount of acknowledged data at sender may not exceed the minimum of *cwnd* and *rwnd*:

LastByteSent – LastByteAcked ≤ min { rwnd, cwnd }

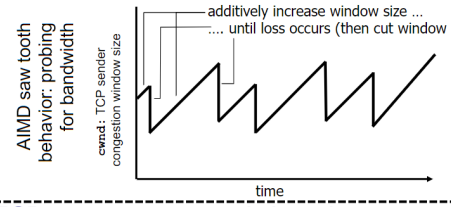
Receiver window (flow control) — Congestion window (congestion control)

AIMD principle

(additive increase multiplicative decrease)

approach: sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs

- *additive increase:* increase **cwnd** by 1 MSS every RTT until loss detected
- *multiplicative decrease:* cut **cwnd** in half after loss



TCP congestion control algorithm

Standardized in RFC 5681

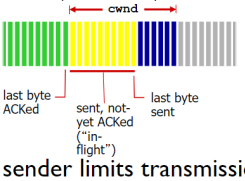
Is based on three major components:

Slow start: set initial transmission rate slow but ramp up exponentially fast (until loss is detected)

Congestion avoidance: on entering this state the value of cwnd is approximately half its value when congestion was last encountered

Fast recovery: when 3 ACKs are received sender performs fast retransmit of missing segment, proceeds in congestion avoidance mode (network is still capable of delivering segments)

details



TCP sending rate:

- *roughly:* send cwnd bytes, wait RTT for ACKS, then send more bytes

$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

sender limits transmission:

LastByteSent-LastByteAcked ≤ cwnd

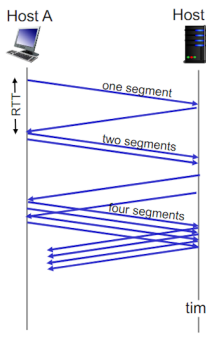
cwnd is dynamic, function of **perceived** network congestion

TCP Slow Start

when connection begins, increase rate exponentially until first loss event:

- initially **cwnd** = 1 MSS
- double **cwnd** every RTT
- done by incrementing **cwnd** for every ACK received

summary: initial rate is slow but ramps up exponentially fast



TCP: detecting, reacting to loss

loss indicated by *timeout*:

- **cwnd** set to 1 MSS;
- window then grows exponentially (as in slow start) to threshold, then grows linearly

loss indicated by *3 duplicate ACKs*:

- dup ACKs indicate network capable of delivering some segments
- Fast retransmit missing segment
- **cwnd** is cut in half window then grows linearly

TCP: switching from slow start to CA

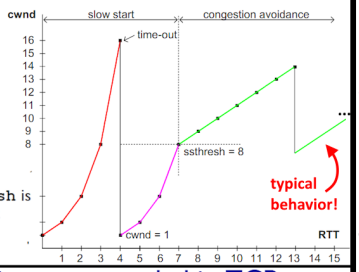
Q: when should the exponential increase switch to linear?

A: when **cwnd** gets to 1/2 of its value before timeout.

Implementation:

variable **ssthresh**

on loss event, **ssthresh** is set to 1/2 of **cwnd** just before loss event



Fast recovery is recommended in TCP, but not required

Early version of TCP (Tahoe) unconditionally cut cwnd to 1 MSS and enters slow-start after either a timeout of triple ACKs received

TCP Reno (more recent) incorporates fast recovery

TCP *flavours*	RFC 793 Postel 81	Tahoe Jacobson 88	Reno Jacobson 90	Vegas Barkmo 94	SACK S. Floyd 2000
Slow start		√	√	√	√
Congestion avoidance		√	√	√	√
Fast retransmit		√	√	√	√
Fast recovery			√	√	√
Selective ACK					√

Network layer !!!

transport segment from sending to receiving host

on sending side encapsulates segments into datagrams

on receiving side, delivers segments to transport layer

network layer protocols in *every* host, router

router examines header fields in all IP datagrams passing through it

Two key network-layer functions

network-layer functions:

- *forwarding:* move packets from router's input interface to appropriate router output interface
- *routing:* determine route taken by packets from source to destination

analogy: taking a trip

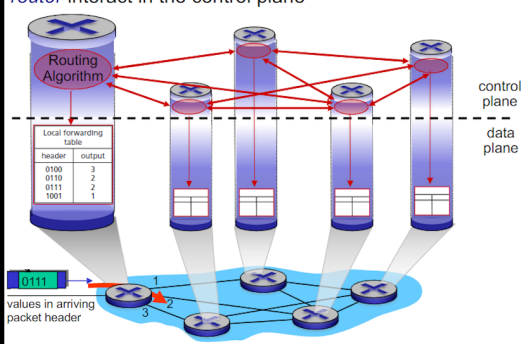
- *forwarding:* process of getting through single interchange
- *routing:* process of planning trip from source to destination

→ *routing algorithms*

→ *implemented in various routers*

Routing algorithms and forwarding tables

Individual routing algorithm components *in each and every router* interact in the control plane



Routing algorithms: important aspects

Algorithms compute the *forwarding tables* that are used to forward packets through the network

Router receives *routing control messages* which are used to configure its *forwarding table*

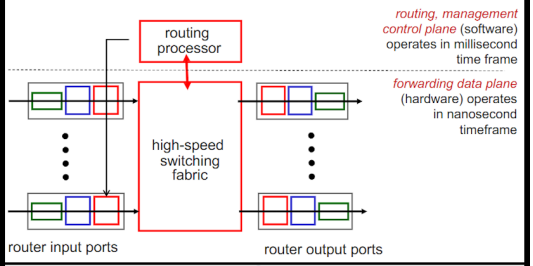
Routing algorithms may be:

- Centralized: algorithms executes on central site and routing information is downloaded to each of the routers
- Decentralized: a piece of the routing algorithm runs in each router

The term "*packet switch*" *may refer to either:*

- Link-layer switch: forwarding decision is based on fields of the link-layer frame (layer 2 devices)
- Router or layer 3 switch: forwarding decision is based on fields of the network-layer (IP) header

Router architecture overview



Destination-based forwarding

forwarding table

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

Longest prefix matching

longest prefix matching

when looking for forwarding table entry for given destination address, use *longest* address prefix (longest subnet mask) that matches destination address.

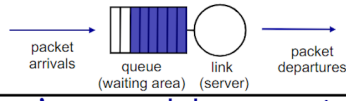
Destination Address Range	Link interface
11001000 00010111 00010*** ********	0
11001000 00010111 00011000 ********	1
11001000 00010111 00011*** ********	2
otherwise	3

Scheduling mechanisms

scheduling: choose next packet to send on link

FIFO (first in first out) scheduling: send in order of arrival to queue

- *discard policy:* if packet arrives to full queue: who to discard?
  - *tail drop:* drop arriving packet
  - *priority:* drop/remove on priority basis
  - *random:* drop/remove randomly



Internet's network layer service

Internet's network layer offers a **best-effort** service:

Ordering: any order possible

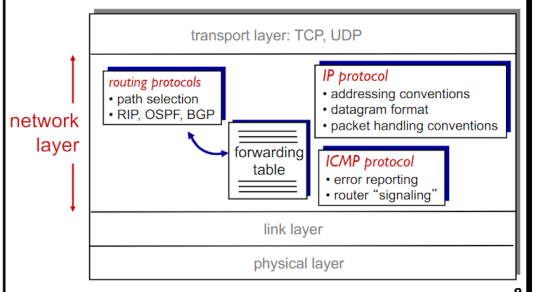
Timing between packets: not maintained

Congestion indication: None

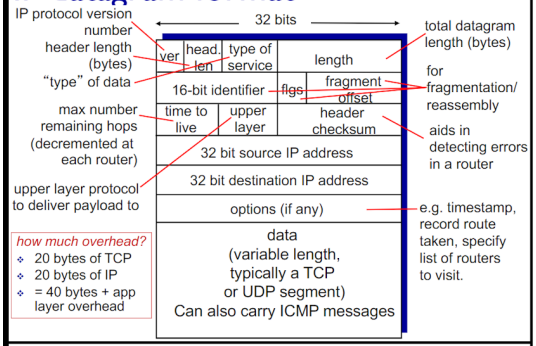
Service model: best effort

Bandwidth guarantee: None

No-Loss guarantee: None



IP datagram format



how much overhead?
- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead

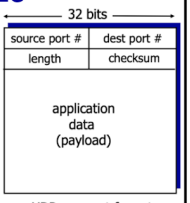
Revisiting UDP: packet size

Minimum size of a UDP packet is 8 bytes.

The maximum size of a UDP packet is 65,535 bytes (or 2^16 - 1 bytes) minus the size of the IP (20 bytes) and UDP (8 bytes) headers.

Most networks have a lower maximum transmission unit (MTU) size, which limits the maximum size of a UDP packet that can be transmitted without fragmentation.

It is generally recommended to keep UDP packets small to minimize the chances of packet loss.



UDP segment format

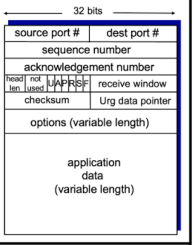
Revisiting TCP: packet size

The minimum size of a TCP packet is 20 bytes.

The maximum size of a TCP packet is determined by the Maximum Segment Size (MSS) value negotiated between the two endpoints during the TCP connection setup process.

The MSS value is typically determined by the maximum size of the data that can be transmitted without fragmentation, based on the MTU of the underlying network.

In practice, the maximum size of a TCP packet can range from 536 bytes (the minimum MSS value) to around 65,535 bytes (the maximum TCP payload size), depending on the MTU of the network and the negotiated MSS value.

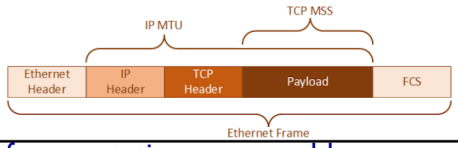


MTU in TCP

The maximum size of an IP packet is determined by the Maximum Transmission Unit (MTU) of the underlying network.

The MTU is the largest size of a packet that can be transmitted over a network without being fragmented. In practice, the MTU can vary depending on the type of network and the network equipment being used.

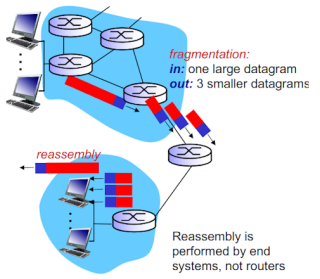
For example, the MTU for Ethernet networks is typically 1,500 bytes, which means that the maximum size of an IP packet (including header) that can be transmitted over an Ethernet network is 1,500 bytes.



IP fragmentation, reassembly

network links have MTU (max.transfer size) - largest possible link-level frame
- different link types, different MTUs

large IP datagram divided ("fragmented") within net
- one datagram becomes several datagrams
- "reassembled" only at final destination (UDP and TCP are expecting to received complete segments)
- IP header bits used to identify, order related fragments



example:
- 4000 byte datagram
- MTU = 1500 bytes

one large datagram becomes several smaller datagrams

	length	ID	fragflag	offset
	=4000	=x	=0	=0
1480 bytes in data field	=1500	=x	=1	=0
offset = 1480/8	=1500	=x	=1	=185
	=1040	=x	=0	=370

IP addresses: how to get one?

Q: How does a *host* get IP address?

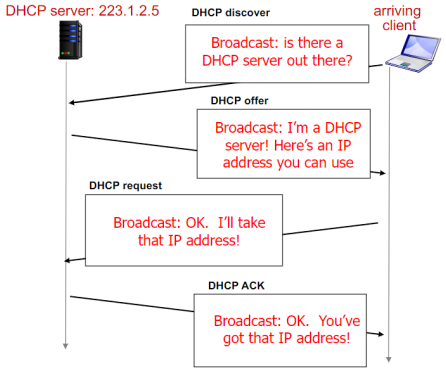
hard-coded by system admin in a file
- Windows: control-panel->network->configuration->tcp/ip->properties
- UNIX: /etc/rc.config

DHCP: Dynamic Host Configuration Protocol: dynamically get address from as server
- "plug-and-play"

DHCP: Dynamic Host Configuration Protocol

goal: allow host to *dynamically* obtain its IP address from network server when it joins network
- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/"on")
- support for mobile users who want to join network (more shortly)

DHCP overview:
- host broadcasts "DHCP discover" msg [optional]
- DHCP server responds with "DHCP offer" msg [optional]
- host requests IP address: "DHCP request" msg
- DHCP server sends address: "DHCP ack" msg



DHCP can return more than just allocated IP address on subnet:
- address of first-hop router for client
- name and IP address of DNS sever
- network mask (indicating network versus host portion of address)

- connecting laptop needs its IP address, addr of first-hop router, addr of DNS server: use DHCP
- DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.1 Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demuxed to IP demuxed, UDP demuxed to DHCP
- DCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulation of DHCP server, frame forwarded to client, demuxing up to DHCP at client
- client now knows its IP address, name and IP address of DNS server, IP address of its first-hop router

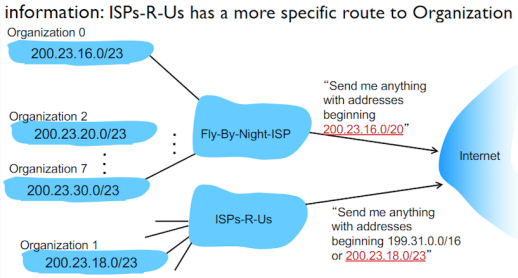
Q: how does *network* get subnet part of IP addr?

A: gets allocated portion of its provider ISP's address space

ISP's block	11001000 00010111 00010000 00000000	200.23.16.0/20
Organization 0	11001000 00010111 00010000 00000000	200.23.16.0/23
Organization 1	11001000 00010111 00010010 00000000	200.23.18.0/23
Organization 2	11001000 00010111 00010100 00000000	200.23.20.0/23
...	.....	....
Organization 7	11001000 00010111 00011110 00000000	200.23.30.0/23

Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information: ISPs-R-Us has a more specific route to Organization 1



Q: how does an ISP get block of addresses?

A: ICANN: Internet Corporation for Assigned Names and Numbers http://www.icann.org/
- allocates addresses
- manages DNS
- assigns domain names, resolves disputes

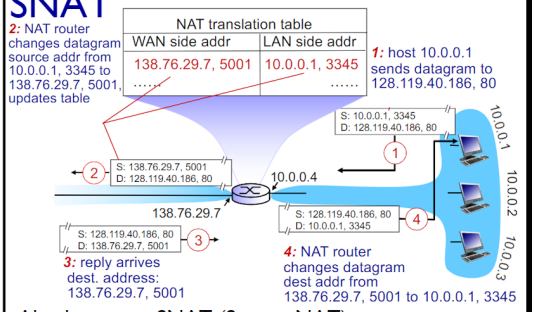
NAT: network address translation

Class	Private address range start address	Private address range finish address	Class	Public address range start address	Public address range finish address
A	10.0.0.0	10.255.255.255	A	0.0.0.0	126.255.255.255
B	172.16.0.0	172.31.255.255	B	128.0.0.0	191.255.255.255
C	192.168.0.0	192.168.255.255	C	192.0.0.0	223.255.255.255
			D	224.0.0.0	239.255.255.255
			E	240.0.0.0	254.255.255.255

motivation: local network uses just one IP address as far as outside world is concerned:
- range of addresses not needed from ISP: just one IP address for all devices
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network
- devices inside local net not explicitly addressable, visible by outside world (a security plus)

implementation: NAT router must:
- *outgoing datagrams: replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
  . . . remote clients/servers will respond using (NAT IP address, new port #) as destination addr
- *remember (in NAT translation table)* every (source IP address, port #) to (NAT IP address, new port #) translation pair
- *incoming datagrams: replace* (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

SNAT



Also known as SNAT (Source NAT)

16-bit port-number field:
- 60,000 simultaneous connections with a single LAN-side address!

NAT is controversial:
- routers should only process up to layer 3
- address shortage should be solved by IPv6
- violates end-to-end argument
  - NAT possibility must be taken into account by app designers, e.g., P2P applications
- NAT traversal: what if client wants to connect to server behind NAT?

NAT Traversal: DNAT

- What if client wants to connect to server behind NAT?
- DNAT (Destination NAT): destination IP and port # changed for packets from the external to the internal network

Network-layer functions

Recall: two network-layer functions:

forwarding: move packets from router's input to appropriate router output

routing: determine route taken by packets from source to destination

data plane *control plane*

Routing algorithm classification

Q: global or decentralized information?

global:
- all routers have **complete topology**, link cost info
- "link state" algorithms

decentralized:
- router knows **only physically-connected neighbors** and link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- "distance vector" algorithms

Q: static or dynamic?

static:
- routes change slowly over time

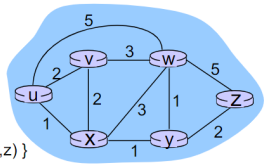
dynamic:
- routes change more quickly
  - periodic update
  - in response to link cost or status changes

Graph abstraction of the network

graph: G = (N,E)

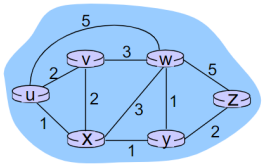
N = set of routers or vertices = { u, v, w, x, y, z }

E = set of links ={ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }



aside: graph abstraction is useful in other network contexts, e.g., P2P, where *N* is set of peers and *E* is set of TCP connections

Graph abstraction: costs



c(x,x') = cost of link (x,x')
e.g., c(w,z) = 5

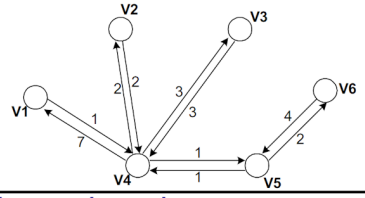
- cost could always be 1, or inversely related to bandwidth, or inversely related to congestion, etc.
- costs can also be different for two directions of communicatior

cost of path $(x_1, x_2, x_3, \ldots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

key question: what is the least-cost path between u and z ?
routing algorithm: algorithm that finds that least cost path

Spanning tree (of a graph)

A **spanning tree** is a subset of a graph which has all the vertices covered and with minimum possible number of edges



Dijkstra algorithm

Created by Edsger Dijkstra (1959)

Algorithm to find shortest path from a given vertex to all other known vertices (routers)

The resulting set of paths forms a spanning tree

For a graph with V vertices requires V-1 iterations

In each iteration the number of operations is proportional to V

Iterative: of $O(V^2)$ complexity for a graph with V vertices

Used in link state routing protocols: requires information about *topology* and *link costs*

Given:

- N: set of vertices in graph
- S: origin vertex
- T: set of vertices (already) added by the algorithm
- w(i,j): cost of path from i to j
- L(n): cost of the path with lower cost from s to n, already added by the algorithm

The algorithm uses 3 steps

- Step 1: initialization
- Step 2 and 3 are repeated until T = N (until spanning tree starting at origin vertex is formed)

Step 1 – Initialization

1.1 T = {s} 1.2 L(n) = w(s,n)

L(s) = **0**

L(i) = **w(s,i)** (to neighbouring nodes of *s*)

L(n) = **infinite** (to other nodes which are not neighbors of *s*)

Step 2 – chooses next vertex to add

2.1 Find x not belonging to T for which L(x) = min L(j), for all j not in T

2.2 Add x to T 2.3 Add link to x to T

Step 3 – update least cost paths

L(n) = min [L(n), L(x)+w(x,n)], for all n not in T

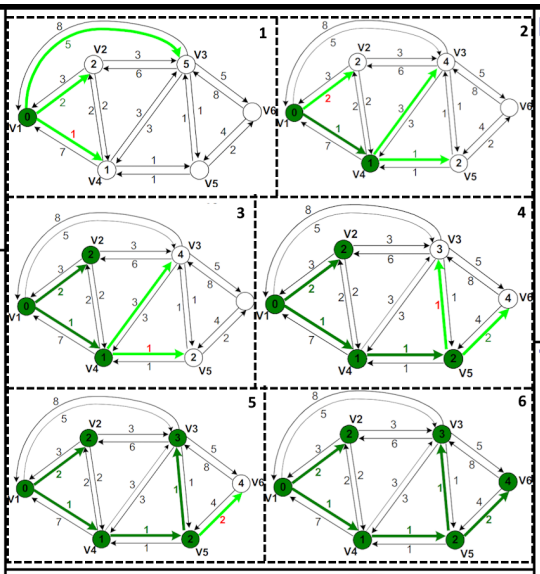
Distance-Vector Protocols

Each router only knows directly-connected neighbours (reachable via the same physical link), and link costs to its neighbours

Contrary to link-state protocols, each router possesses a limited vision of the topology

Each router sends periodic *updates* to its neighbours (even if no changes in the network have to be reported)

The updates include the complete contents of the routing table of the router



Routing protocols (link-state and distance-vector)

Protocolo	Tipo de encaminhamento		Algoritmo
	interior	exterior	
RIP (*Routing Information Protocol*), v1	•		*distance-vector*
RIP (*Routing Information Protocol*), v2	•		*distance-vector*
IGRP (*Interior Gateway Routing Protocol*)	•		*distance-vector*
EIGRP (*Enhanced Interior Gateway Routing Protocol*)	•		*distance-vector +link-state*
OSPF (*Open Shortest Path First*)	•		*link-state*
EGP (*Exterior Gateway Protocol*)		•	–
BGP (*Border Gateway Protocol*)		•	–

Router Information Protocol

RIP-2 – Defined in RFC 1723 (1994)

Router *broadcasts* routing information at each 30 s

Sends immediate update upon detecting change on a link

Router uses information received from its neighbours to calculate the shortest paths to all reachable (and known) destinations

Uses *hop count* as metric

Maximum hop count of 15 (16 is considered "infinite", or "unreachable")

Reports are broadcasted to neighbours

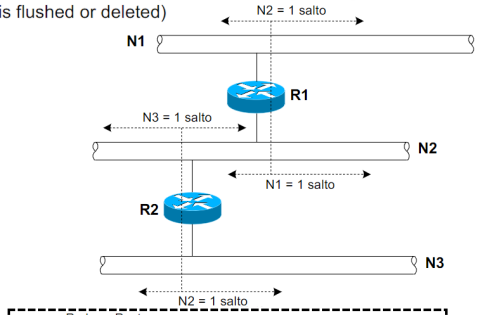
A route expires if no update is received for 180s

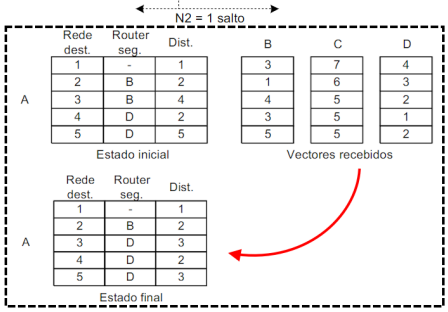
May store up to 6 equal cost paths to same destination

Supports load balancing using paths with the same cost

Timer values used by the protocol:

- **Update interval**: 30s (periodic update of routes sent to neighbours)
- **Invalid timer**: 180s (time since last update for route, upon which route is marked as invalid and put "on hold": *hop count* 16 or "infinite")
- **Flush timer**: 240 s (time since last update upon which route is flushed or deleted)





ICMP: internet control message protocol

used by hosts & routers to communicate network-level information

- error reporting: unreachable host, network, port, protocol
- echo request/reply (used by ping)

network-layer "above" IP:

- ICMP msgs carried in IP datagrams

ICMP message: type, code plus first 8 bytes of IP datagram causing error

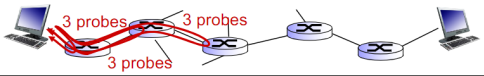
Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

Traceroute and ICMP

- source sends series of UDP segments to destination
  - first set has TTL =1
  - second set has TTL=2, etc.
  - unlikely port number
- when datagram in *n*th set arrives to nth router:
  - router discards datagram and sends source ICMP message (type 11, code 0)
  - ICMP message include name of router & IP address
- when ICMP message arrives, source records RTTs

stopping criteria:

- UDP segment eventually arrives at destination host
- destination returns ICMP "port unreachable" message (type 3, code 3)
- source stops



Link layer:

terminology:

- hosts and routers: nodes
- communication channels that connect adjacent nodes along communication path: links
  - wired links
  - wireless links
  - LANs
- layer-2 packet: frame, encapsulates datagram

data-link layer has responsibility of transferring datagram from one node to *physically adjacent* node over a link

datagram transferred by *different link protocols* over *different links*:

- e.g., Ethernet on first link, frame relay on intermediate links, 802.11 on last link

transportation analogy:

- trip from Princeton to Lausanne
  - limo: Princeton to JFK
  - plane: JFK to Geneva
  - train: Geneva to Lausanne
- tourist = datagram
- transport segment = communication link
- transportation mode = link layer protocol
- travel agent = routing algorithm

each link protocol provides different services

- e.g., may or may not provide rdt over link

Link layer services

framing, link access:

- encapsulate (network layer) datagram into frame, adding header, trailer
- Medium Access Control (MAC) protocol defines rules for channel access if shared medium
- "MAC" addresses used in frame headers to identify source, destination
  - different from IP address!

reliable delivery between adjacent nodes

- we learned how to do this already (network layer)!
- seldom used on low bit-error link (fiber, some twisted pair)
- wireless links: high error rates
  - *Q:* why both link-level and end-end reliability?

flow control:

- pacing between adjacent sending and receiving nodes

error detection:

- errors caused by signal attenuation, noise.
- more sophisticated than at network and transport layer (Internet checksum) and implemented at the hardware
- receiver detects presence of errors:
  - transmitting node includes error-detection bits in frame, receiving node performs an error check
  - signals sender for retransmission or drops frame

error correction:

- receiver identifies *and corrects* bit error(s) without resorting to retransmission

half-duplex and full-duplex

- with half duplex, nodes at both ends of link can transmit, but not at same time

Where is the link layer implemented?

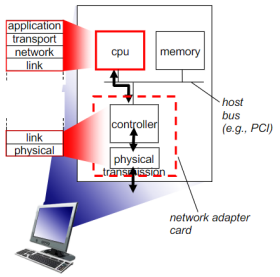
in each and every host

link layer implemented in "adaptor" (aka *network interface card* NIC) or on a chip

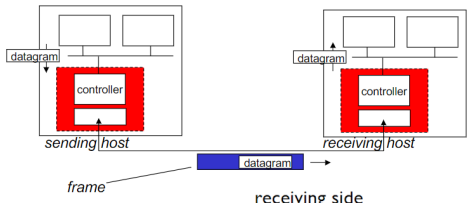
- Ethernet card, 802.11 card; Ethernet chipset
- implements link, physical layer

attaches into host's system buses

combination of *hardware, software, firmware*



Adaptors communicating



sending side:

- encapsulates datagram in frame
- adds error checking bits, flow control, etc.

receiving side

- looks for errors, flow control, etc.
- extracts datagram, passes to upper layer at receiving side

Multiple access links

two types of "links":

- point-to-point
  - PPP for dial-up access
  - point-to-point link between Ethernet switch, host
- *broadcast (shared wire or medium)*
  - **broadcast**: when a node transmits a frame each of the other nodes in the link receives a copy
  - old-fashioned Ethernet
  - 802.11 wireless LAN

Multiple access protocols

single shared broadcast channel

two or more simultaneous transmissions by nodes: interference

- *collision* if node receives two or more signals at the same time
- bandwidth is wasted during collisions

multiple access protocol

- distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
- communication about channel sharing must use channel itself!
  - no out-of-band channel for coordination

MAC protocols: taxonomy

three broad classes:

- *channel partitioning*
  - divide channel into smaller "pieces": time-division multiplexing (TDM), frequency-division multiplexing (FDM), code division multiplexing (CDMA)
  - allocate piece to node for exclusive use
- *random access*
  - channel not divided, allow collisions
  - "recover" from collisions
- *"taking turns"*
  - nodes take turns, but nodes with more to send can take longer turns

Channel partitioning MAC protocols: TDMA

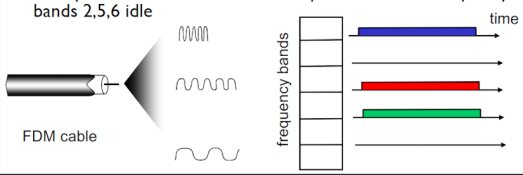
TDMA: time division multiple access

- access to channel in "rounds"
- each station gets fixed length slot (length = packet transmission time) in each round
- unused slots go idle
- example: 6-station LAN, 1,3,4 have packets to send, slots 2,5,6 idle
- eliminates collisions and is perfectly fair, but..
- limited transmission rate per node even if other nodes are not transmitting, node always waits for its turn to transmit

Channel partitioning MAC protocols: FDMA

FDMA: frequency division multiple access

- channel spectrum divided into frequency bands
- each station assigned fixed frequency band
- unused transmission time in frequency bands go idle
- example: 6-station LAN, 1,3,4 have packet to send, frequency bands 2,5,6 idle



Random access protocols

when node has packet to send

- transmit at full channel data rate R.
- no *a priori* coordination among nodes

two or more transmitting nodes ➜ "collision",

random access MAC protocol specifies:

- how to detect collisions
- how to recover from collisions (e.g., via delayed retransmissions): *each node selects an independent random delay before retransmitting*

examples of random access MAC protocols (hundreds are available in the literature!):

- slotted ALOHA
- ALOHA
- CSMA, CSMA/CD, CSMA/CA

CSMA collisions

collisions *can* still occur: propagation delay means two nodes may not hear each other's transmission

collision: entire packet transmission time wasted

- distance & propagation delay play role in in determining collision probability

(collision detection)

CSMA/CD: carrier sensing, deferral as in CSMA

- collisions *detected* within short time
- colliding transmissions aborted, reducing channel wastage

- collision detection: measure signal strengths, compare transmitted, received signals
  - May be used in wired networks (e.g. Ethernet)
  - Very hard in wireless networks (Wi-Fi 802.11): use CSMA/CA

Ethernet CSMA/CD algorithm

1. NIC receives datagram from network layer, creates frame
2. If NIC senses channel idle, starts frame transmission. If NIC senses channel busy, *waits until channel idle*, then transmits.
3. If NIC transmits entire frame without detecting another transmission, NIC is done with frame !
4. If NIC detects another transmission while transmitting, aborts
5. After aborting, NIC enters *binary (exponential) backoff:*
   - after *m*th collision, NIC chooses *K* at random from $\{0,1,2,\ldots,2^m-1\}$. NIC waits $K \cdot 512$ bit times, *returns to Step 2*
   - longer backoff interval with more collisions

"Taking turns" MAC protocols

channel partitioning MAC protocols:

- share channel *efficiently* and *fairly* at high load

inefficient at low load: delay in channel access, 1/N bandwidth allocated even if only 1 active node!

random access MAC protocols

- efficient at low load: single node can fully utilize channel
- high load: collision overhead

"taking turns" protocols

look for best of both worlds!

polling:

- master node "invites" slave nodes to transmit in turn
- typically used with "dumb" slave devices
- concerns:
  - polling overhead
  - latency
  - single point of failure (master)

token passing:

- control *token* passed from one node to next sequentially.
- token message
- concerns:
  - token overhead
  - latency
  - single point of failure (token)

Summary of MAC protocols

channel partitioning, by time, frequency or code

- Time Division, Frequency Division

random access (dynamic),

- ALOHA, S-ALOHA, CSMA, CSMA/CD
- carrier sensing: easy in some technologies (wire), hard in others (wireless)
- CSMA/CD used in Ethernet
- CSMA/CA used in 802.11

taking turns

- polling from central site, token passing
- Bluetooth, FDDI, token ring

MAC addresses and ARP

32-bit IP address:

- *network-layer* address for interface
- used for layer 3 (network layer) forwarding

MAC (or LAN or physical or Ethernet) address:

- function: *used 'locally" to get frame from one interface to another physically-connected interface (same network, in IP-addressing sense)*
- 48 bit MAC address (for most LANs) burned in NIC ROM, also sometimes software settable
- e.g.: 1A-2F-BB-76-09-AD

LAN addresses and ARP

each adapter on LAN has unique *LAN* address

MAC address allocation administered by IEEE

manufacturer buys portion of MAC address space (to assure uniqueness)

Portability:

- can move LAN card from one LAN to another

IP hierarchical address *not* portable

- address depends on IP subnet to which node is attached

ARP: address resolution protocol

Question: how to determine interface's MAC address, knowing its IP address?

ARP table: each IP node (host, router) on LAN has table

- IP/MAC address mappings for some LAN nodes: < IP address; MAC address; TTL>
- TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

ARP protocol: same LAN

A wants to send datagram to B

- B's MAC address not in A's ARP table.

A broadcasts ARP query packet, containing B's IP address

- destination MAC address = FF-FF-FF-FF-FF-FF
- all nodes on LAN receive ARP query

B receives ARP packet, replies to A with its (B's) MAC address

- frame sent to A's MAC address (unicast)

A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (times out)

- soft state: information that times out (goes away) unless refreshed

ARP is "plug-and-play":

- nodes create their ARP tables *without intervention from net administrator*

Addressing: routing to another LAN

walkthrough: send datagram from A to B via R

- focus on addressing – at IP (datagram) and MAC layer (frame)
- assume A knows B's IP address
- assume A knows IP address of first hop router, R (how?)
- assume A knows R's MAC address (how?)

- A creates IP datagram with IP source A, destination B
- A creates link-layer frame with R's MAC address as destination address, frame contains A-to-B IP datagram

Ethernet

"dominant" wired LAN technology:

- single chip, multiple speeds (e.g., Broadcom BCM5761)
- first widely used LAN technology
- simpler, cheap
- kept up with speed race: 10 Mbps – 10 Gbps

physical topology

- *bus:* popular through mid 90s
- all nodes in same collision domain (can collide with each other)
- *star:* prevails today
  - active *switch* in center
  - each "spoke" runs a (separate) Ethernet protocol (nodes do not collide with each other)

Ethernet frame structure

sending adapter encapsulates IP datagram (or other network layer protocol packet) in Ethernet frame

preamble:

- 7 bytes with pattern 10101010 followed by one byte with pattern 10101011
- used to synchronize receiver, sender clock rates

addresses: 6 byte (48 bit) source, destination MAC addresses

- if adapter receives frame with matching destination address, or with broadcast address (e.g. ARP packet), it passes data in frame to network layer protocol
- otherwise, adapter discards frame

type: indicates higher layer protocol (mostly IP but others possible, e.g., Novell IPX, AppleTalk)

CRC: cyclic redundancy check at receiver

- error detected: frame is dropped

Ethernet: unreliable, connectionless

connectionless: no handshaking between sending and receiving NICs

unreliable: receiving NIC doesn't send acks or nacks to sending NIC

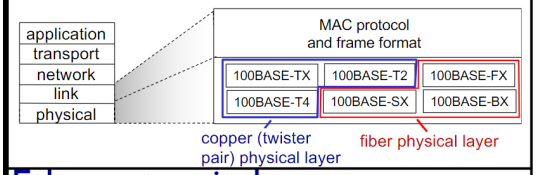
- data in dropped frames recovered only if initial sender uses higher layer rdt (e.g., TCP), otherwise dropped data lost

Ethernet's MAC protocol: unslotted *CSMA/CD with exponential backoff*

802.3 Ethernet standards: link & physical layers

many different Ethernet standards

- common MAC protocol and frame format
- different speeds: 2 Mbps, 10 Mbps, 100 Mbps, 1Gbps, 10 Gbps, 40 Gbps
- different physical layer media: fiber, cable



Ethernet switch

link-layer device: takes an *active* role

- store, forward Ethernet frames
- examine incoming frame's MAC address, selectively forward frame to one-or-more outgoing links when frame is to be forwarded or segment, uses CSMA/CD to access segment

transparent

- hosts are unaware of presence of switches

plug-and-play, self-learning

- switches do not need to be configured

Switch: *multiple* simultaneous transmissions

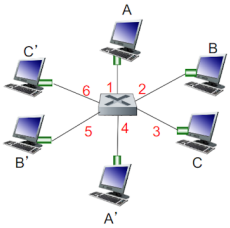
hosts have dedicated, direct connection to switch

switches buffer packets

Ethernet protocol used on *each* incoming link, but no collisions; full duplex

- each link is its own collision domain

switching: A-to-A' and B-to-B' can transmit simultaneously, without collisions



switch with six interfaces (1,2,3,4,5,6)

Switch forwarding table

Q: how does switch know A' reachable via interface 4, B' reachable via interface 5?

- *A:* each switch has a switch table, each entry:
  - (MAC address of host, interface to reach host, time stamp)
  - looks like a routing table!

self-learning

switch *learns* which hosts can be reached through which interfaces

- when frame received, switch "learns" location of sender: incoming LAN segment
- records sender/location pair in switch table

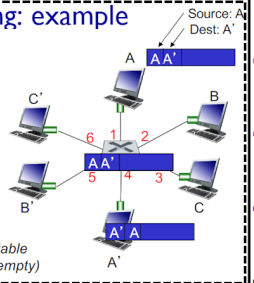
Switch: frame filtering/forwarding

when frame received at switch:

1. record incoming link, MAC address of sending host
2. index switch table using MAC destination address
3. if entry found for destination
   then {
   if destination on segment from which frame arrived
   then drop frame
   else forward frame on interface indicated by entry
   }
   else flood /* forward on all interfaces except arriving interface */

Self-learning, forwarding: example

- frame destination, A', location unknown: *flood*
- destination A location known: selectively send on just one link

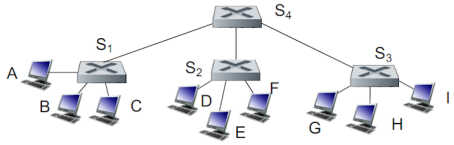


MAC addr	interface	TTL
A	1	60
A'	4	60

switch table (initially empty)

Interconnecting switches

self-learning switches can be connected together:



Q: sending from A to G - how does S_1 know to forward frame destined to G via S_4 and S_3?

- *A:* self learning! (works exactly the same as in single-switch case!)

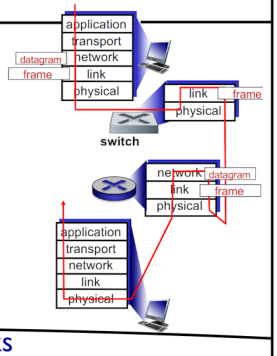
Switches vs. routers

both are store-and-forward:

- *routers:* network-layer devices (examine network-layer headers)
- *switches:* link-layer devices (examine link-layer headers)

both have forwarding tables:

- *routers:* compute tables using routing algorithms, IP addresses
- *switches:* learn forwarding table using flooding, learning, MAC addresses



Data center networks

10's to 100's of thousands of hosts, often closely coupled, in close proximity:

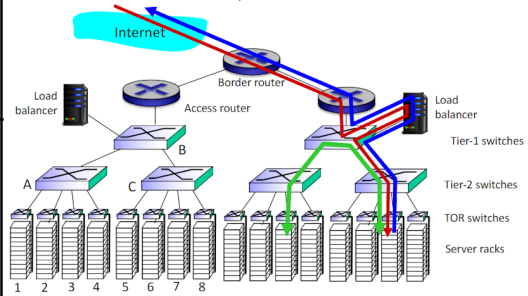
- e-business (e.g. Amazon)
- content-servers (e.g., YouTube, Akamai, Apple, Microsoft)
- search engines, data mining (e.g., Google)

challenges:

- multiple applications, each serving massive numbers of clients
- managing/balancing load, avoiding processing, networking, data bottlenecks

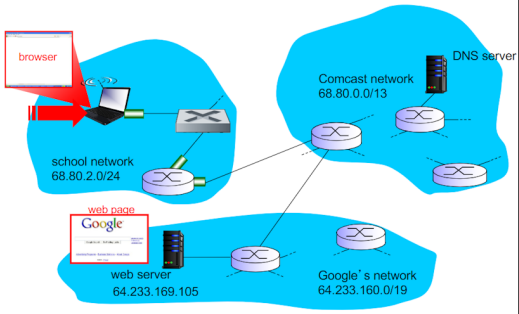
load balancer: application-layer routing

- receives external client requests
- directs workload within data center
- returns results to external client (hiding data center internals from client)



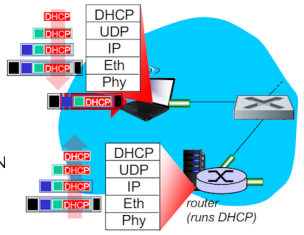
- rich interconnection among switches, racks:
  - increased throughput between racks (multiple routing paths possible)
  - increased reliability via redundancy

Synthesis: a day in the life of a web request



connecting to the Internet

- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use *DHCP*
- DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.3 Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN received at router running DHCP server
- Ethernet demuxed to IP demuxed, UDP demuxed to DHCP
- DHCP server formulates *DHCP ACK* containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server



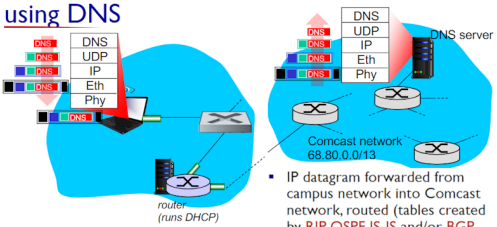
- encapsulation at DHCP server, frame forwarded (switch learning) through LAN, demultiplexing at client
- DHCP client receives DHCP ACK reply

Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router

ARP (before DNS, before HTTP)

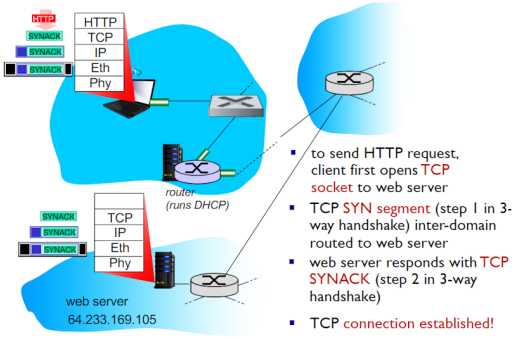
- before sending *HTTP* request, need IP address of www.google.com: *DNS*
- DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: *ARP*
- *ARP query* broadcast, received by router, which replies with *ARP reply* giving MAC address of router interface
- client now knows MAC address of first hop router, so can now send frame containing DNS query

using DNS



- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router
- IP datagram forwarded from campus network into Comcast network, routed (tables created by RIP, OSPF, IS-IS and/or BGP routing protocols) to DNS server
- demuxed to DNS server
- DNS server replies to client with IP address of www.google.com

TCP connection carrying HTTP



- to send HTTP request, client first opens TCP socket to web server
- TCP SYN segment (step 1 in 3-way handshake) inter-domain routed to web server
- web server responds with TCP SYNACK (step 2 in 3-way handshake)
- TCP connection established!

after...

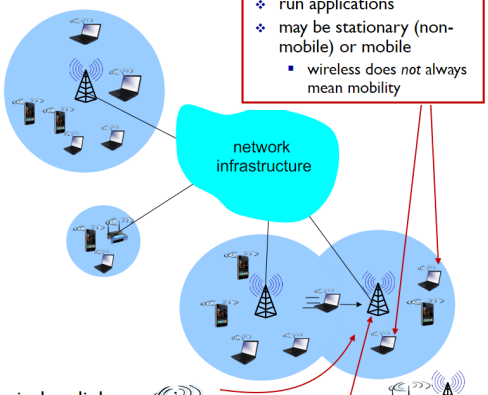
- HTTP request sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com
- web server responds with HTTP reply (containing web page)
- IP datagram containing HTTP reply routed back to client
- web page finally (!!!) displayed

Wireless Networks

Background:

- # wireless (mobile) phone subscribers now exceeds # wired phone subscribers (5-to-1)!
- # wireless Internet-connected devices exceeds # wireline Internet-connected devices
  - laptops, Internet-enabled phones promise anytime untethered Internet access
- two important (but different) challenges
  - *wireless:* communication over wireless link
  - *mobility:* handling the mobile user who changes point of attachment to network

Elements of a wireless network



wireless hosts
- laptop, smartphone
- run applications
- may be stationary (non-mobile) or mobile
  - wireless does *not* always mean mobility

wireless link
- typically used to connect mobile(s) to base station
- also used as backbone link
- multiple access protocol coordinates link access
- various data rates, transmission distance

base station
- typically connected to wired network
- relay - responsible for sending packets between wired network and wireless host(s) in its "area"
  - e.g., cell towers, 802.11 access points

ad hoc mode
- no base stations
- nodes can only transmit to other nodes within link coverage
- nodes organize themselves into a network: route among themselves

infrastructure mode
- base station connects mobiles into wired network
- handoff: mobile changes base station providing connection into wired network

2G vs 3G vs 4G vs 5G

Comparison	2G	3G
Introduced in year	1993	2001
Technology	GSM	WCDMA
Access system	TDMA, CDMA	CDMA
Switching type	Circuit switching for voice and packet switching for data	Packet switching except for air interference
Internet service	Narrowband	Broadband
Bandwidth	25 MHz	25 MHz
Advantage	Multimedia features (SMS, MMS), internet access and SIM introduced	High security, international roaming
Applications	Voice calls, short messages	Video conferencing, mobile TV, GPS

4G	5G
2009	2018
LTE, WiMAX	MIMO, mm Waves
CDMA	OFDM, BDMA
Packet switching	Packet switching
Ultra broadband	Wireless World Wide Web
100 MHz	30 GHz to 300 GHz
Speed, high speed handoffs, global mobility	Extremely high speeds, low latency
High speed applications, mobile TV, wearable devices	High resolution video streaming, remote control of vehicles, robots, and medical procedures

Wireless network taxonomy

	single hop	multiple hops
infrastructure (e.g., APs)	host connects to base station (WiFi, WiMAX, cellular) which connects to larger Internet	host may have to relay through several wireless nodes to connect to larger Internet: *mesh net*
no infrastructure	no base station, no connection to larger Internet (Bluetooth, ad hoc nets)	no base station, no connection to larger Internet. May have to relay to reach other a given wireless node MANET, VANET

Wireless Link Characteristics

important differences from wired link ….
- *decreased signal strength:* radio signal attenuates as it propagates through matter (path loss)
- *interference from other sources:* standardized wireless frequencies (e.g., 2.4 GHz) shared by other devices (e.g., phone); devices (motors) interfere as well
- *multipath propagation:* radio signal reflects off objects/ground, reaching destination at slightly different times

…. make communication across (even a point to point) wireless link much more "difficult"

IEEE 802.11 Wireless LAN

802.11b
- 2.4-5 GHz unlicensed spectrum
- up to 11 Mbps
- direct sequence spread spectrum (DSSS) in physical layer

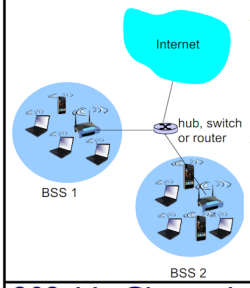
802.11a
- 5-6 GHz range
- up to 54 Mbps

802.11g
- 2.4-5 GHz range
- up to 54 Mbps

802.11n: multiple antennae
- 2.4-5 GHz range
- up to 200 Mbps

- all use CSMA/CA for multiple access
- all have base-station and ad-hoc network versions

802.11 LAN architecture



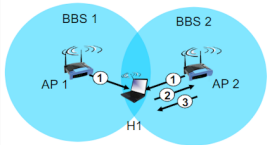
- wireless host communicates with base station
  - base station = access point (AP)
- Basic Service Set (BSS) (aka "cell") in infrastructure mode contains:
  - wireless hosts
  - access point (AP): base station
  - ad hoc mode: hosts only

802.11: Channels, association

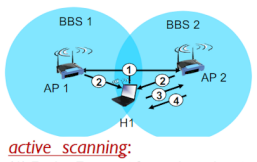
802.11b: 2.4GHz-2.485GHz spectrum divided into 11 channels at different frequencies
- AP admin chooses frequency for AP
- interference possible: channel can be same as that chosen by neighboring AP!

host: must *associate* with an AP
- scans channels, listening for *beacon frames* containing AP's name (SSID) and MAC address
- selects AP to associate with
- may perform authentication
- will typically run DHCP to get IP address in AP subnet

802.11: passive/active scanning



passive scanning:
(1) beacon frames sent from APs
(2) association Request frame sent: H1 to selected AP
(3) association Response frame sent from selected AP to H1



active scanning:
(1) Probe Request frame broadcast from H1
(2) Probe Response frames sent from APs
(3) Association Request frame sent: H1 to selected AP
(4) Association Response frame sent from selected AP to H1

IEEE 802.11: multiple access

avoid collisions: 2^{+} nodes transmitting at same time

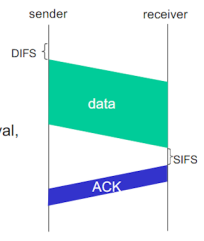
802.11: CSMA - sense before transmitting
- don't collide with ongoing transmission by other node

802.11: *no* collision detection!
- difficult to receive (sense collisions) when transmitting due to weak received signals (fading)
- goal: *avoid collisions:* CSMA/C(ollision)A(voidance)

IEEE 802.11 MAC Protocol: CSMA/CA

1 if sense channel idle for **DIFS** (Interframe space) then
  transmit entire frame (no CD)
2 if sense channel busy then
  start random backoff time
  timer counts down while channel idle
  transmit when timer expires
  if no ACK, increase random backoff interval, repeat 2

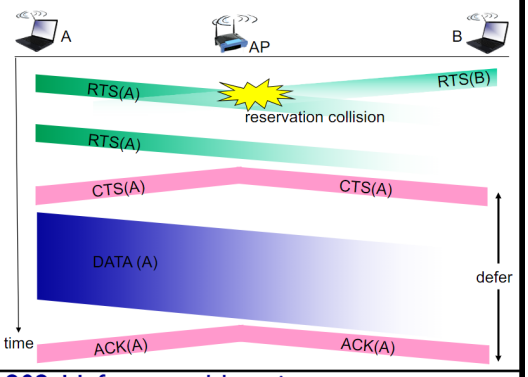
802.11 receiver
- if frame received OK
  return ACK after **SIFS** (Short Interframe Space)



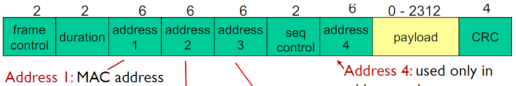
Avoiding collisions (more)

idea: allow sender to "reserve" channel rather than random access of data frames: avoid collisions of long data frames
- sender first transmits *small* request-to-send (RTS) packets to BS using CSMA
  - RTSs may still collide with each other (but they're short)
- BS broadcasts clear-to-send CTS in response to RTS
- CTS heard by all nodes
  - sender transmits data frame
  - other stations defer transmissions

avoid data frame collisions completely using small reservation packets!



802.11 frame: addressing

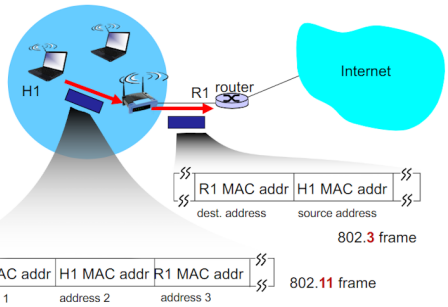


Address 1: MAC address of wireless host or AP to receive this frame

Address 2: MAC address of wireless host or AP transmitting this frame

Address 3: MAC address of router interface to which AP is attached

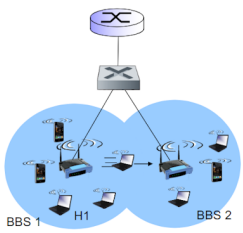
Address 4: used only in ad hoc mode



802.11: mobility within same subnet

H1 remains in same IP subnet: IP address can remain same

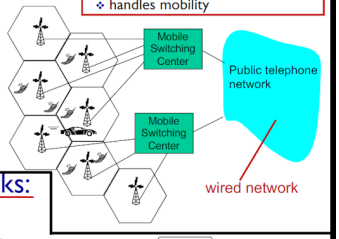
switch: which AP is associated with H1?
- self-learning (T05): switch will see frame from H1 and "remember" which switch port can be used to reach H1



Components of cellular network architecture

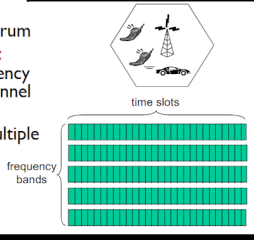
cell
- covers geographical region
- *base station* (BS) analogous to 802.11 AP
- *mobile users* attach to network through BS
- *air-interface:* physical and link layer protocol between mobile and BS

MSC
- connects cells to wired tel. net.
- manages call setup
- handles mobility



Cellular networks: the first hop

Two techniques for sharing mobile-to-BS radio spectrum
- combined FDMA/TDMA: divide spectrum in frequency channels, divide each channel into time slots
- CDMA: code division multiple access



What is an IP address?

IP addresses allow to *uniquely* identify and communicate with hosts (servers, laptops, smartphones, sensors, etc) in the Internet

IPv4 addresses are written as four numbers separated by periods (Dotted-decimal notation), each number can be 0 to 255

Alternatively, may be written in hexadecimal notation

IPv4

- 32 bits used for addresses
- Provides a total of 4,294,967,296 addresses

IPv6

- 128 bits used for addresses
- Provides a total of 340,282,366,920,938,463,374,607,431,768,211,456 addresses
- That's about 3.7×10^{21} addresses per square inch of the earth's surface!

IPv4 classful addressing

32-bit IP addresses are divided into five sub-classes: A, B, C, D and E

Each of the classes has a valid range of IP addresses

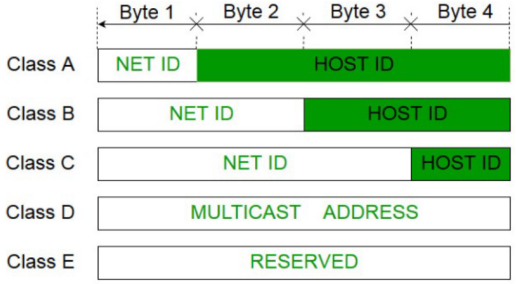
The order of bits in the first octet (byte) determine the classes of IP addresses

Each IP address if divided in the Network ID and Host ID

The class of IP address determines the number of total networks and hosts possible in that particular class.

IP addresses are globally managed by the Internet Assigned Numbers Authority (IANA) and Regional Internet Registries (RIR)

Each ISP or network administrator assigns IP address to each device that is connected to its network



Class A IPv4 addresses:

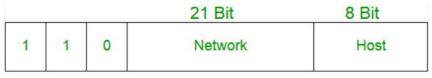
- Assigned to networks that contain a large number of hosts
- Network ID is 8 bits long, Host ID is 24 bits long
- Higher order bit of the first byte is always 0
- From 1.0.0.1 to 126.255.255.254

Class B IPv4 addresses:

- Assigned to medium-sized to large-sized networks
- Network ID is 16 bits long, Host ID is 16 bits long
- Higher order bits of the first byte is always 10
- From 128.1.0.1 to 191.255.255.254

Class C IPv4 addresses:

- Assigned to small-sized networks (most common)
- Network ID is 24 bits long, Host ID is 8 bits long
- Higher order bits of the first byte is always 110
- From 192.0.1.1 to 223.255.255.254



Class D IPv4 addresses:

- Reserved for multicast communications
- Higher order bits of the first byte is always 1110
- From 224.0.0.0 to 239.255.255.255

Class E IPv4 addresses:

- Experimental and research purposes
- Higher order bits of the first byte is always 1111
- From 240.0.0.0 to 255.255.255.254

CLASS	LEADING BITS	NET ID BITS	HOST ID BITS	NO. OF NETWORKS	ADDRESSES PER NETWORK	START ADDRESS	END ADDRESS
CLASS A	0	8	24	2^7 (128)	2^{24} (16,777,216)	0.0.0.0	127.255.255.255
CLASS B	10	16	16	2^{14} (16,384)	2^{16} (65,536)	128.0.0.0	191.255.255.255
CLASS C	110	24	8	2^{21} (2,097,152)	2^8 (256)	192.0.0.0	223.255.255.255
CLASS D	1110	NOT DEFINED	NOT DEFINED	NOT DEFINED	NOT DEFINED	224.0.0.0	239.255.255.255
CLASS E	1111	NOT DEFINED	NOT DEFINED	NOT DEFINED	NOT DEFINED	240.0.0.0	255.255.255.255

Special addresses

Lookback addresses (127.0.0.1 to 127.0.0.8):

- Most common is 127.0.0.1
- Used for communications between applications in the same system
- Used for diagnostic testing of the local TCP/IP installation

Automatic private IP addressing addresses (169.254.0.0 to 169.254.255.255)

- Self-assigned IP addresses, when computer is unable to get an address from the network

Network address:

- First address in the range (**Host ID bits are all set to 0**), used to **represent the local network**

Broadcast address:

- Last address in the range (**Host ID bits all set to 1**), used to **communicate with all systems in the local network**

IP addresses and netmasks

Netmask designate which bits of an IP address represent the network portion (bits with 1) of the address and which bits represent the host portion (bits with 0)

- Class A: 255.0.0.0 (or /8)
- Class B: 255.255.0.0 (or /16)
- Class C: 255.255.255.0 (or /24)

The netmask also allows to know the network size (how many hosts can be addressed)

Netmask for a class C network (alternative representations):

255.255.255.0
11111111 11111111 11111111 00000000
/24

Further subdivision of the addressing space is possible and very useful!

This is known as "subnetting"

Subnet Mask	CIDR	Subnet Mask	CIDR
255.128.0.0	/9	255.255.240.0	/20
255.192.0.0	/10	255.255.248.0	/21
255.224.0.0	/11	255.255.252.0	/22
255.240.0.0	/12	255.255.254.0	/23
255.248.0.0	/13	255.255.255.0	/24
255.252.0.0	/14	255.255.255.128	/25
255.254.0.0	/15	255.255.255.192	/26
255.255.0.0	/16	255.255.255.224	/27
255.255.128.0	/17	255.255.255.240	/28
255.255.192.0	/18	255.255.255.248	/29
255.255.224.0	/19	255.255.255.252	/30

Reserved IP addresses

Designed to be used on a private network behind a NAT (Network Address Translation) device (e.g. firewall or router)

Cannot be used to communicate directly with other systems over the Internet

Common usage in home, office and academic networks

Class A IP Range	Subnet Mask
10.0.0.0 – 10.255.255.255	255.0.0.0
172.16.0.0 – 172.31.255.255	255.240.0.0
192.168.0.0 – 192.168.255.255	255.255.0.0

Internet (IP) routing

A router is responsible for the forwarding of IP packets across different (physical) networks

A router (and also a host) uses a routing table to select the next destination (hop) for an IP packet

The information in the routing table is referred to as routes

Routes may be added manually (static routing) or by specializes routing processes/protocols (dynamic routing)

IP subnetting

Further subdivision (subnetting) of the addressing space is possible

As different physical networks need to use different IP address networks this is a very common operation

/24 supports 256 addresses (254 addresses for hosts)
/25 supports 128 addresses (126 addresses for hosts)

/25 = 11111111 11111111 11111111 10000000

We can now use the new bit in the netmask to create two subnetworks of the original network

If the bit is "0":

Network: 192.168.11.0/25
Netmask: 255.255.255.128
Network address: 192.168.11.0
Broadcast address: 192.168.11.127
IP address range: 192.168.11.1 to 126

If the bit is "1":

Network: 192.168.11.128/25
Netmask: 255.255.255.128
Network address: 192.168.11.128
Broadcast address: 192.168.11.255
IP address range: 192.168.11.129 to 254

Socket programming (UDP)

UDP is connectionless

The client does not form a connection with the server like in TCP, instead just sends a datagram to the server

The server need not accept a connection and just waits for datagrams to arrive

Datagrams upon arrival contain the address and port of sender, which the server may use to reply

The application invokes the **socket** function to create an UDP or TCP socket

int socket (int domain, int type, int protocol)

- Returns socket file descriptor
- Communication domain: AF_INET for IPv4, AF_INET6 for IPv6
- Type of socket to be created: SOCK_STREAM for TCP, SOCK_DGRAM for UDP
- Protocol to be used by socket: IPPROTO_UDP for UDP, IPPROTO_TCP for TCP

The **bind** function is used by the server to assign an address to the unbound socket

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)

- File descriptor of socket to be binded
- Structure in which address to be binded to is specified
- Size of addr structure

Socket address structures

Most socket functions require a pointer to a socket address structure

The names of these structures begin with *sockaddr_* and end with a unique suffix for each protocol suite

Generic structure (used in declarations, to deal with address structures from any supported protocol families)

```
#include <arpa/inet.h>
struct in_addr {
    in_addr_t  s_addr;          /* 32-bit IPv4 address */
                                /* network byte ordered */
};
struct sockaddr_in {
    uint8_t       sin_len;      /* length of structure (16) */
    sa_family_t   sin_family;   /* AF_INET */
    in_port_t     sin_port;     /* 16-bit TCP or UDP port number */
                                /* network byte ordered */
    struct in_addr sin_addr;    /* 32-bit IPv4 address */
                                /* network byte ordered */
    char          sin_zero[8];  /* unused */
};
```

The **sendto** function is used to send a message on the socket

- File descriptor of socket
- Application buffer containing the data to be sent
- Size of buf application buffer
- Bitwise OR of flags to modify socket behavior

ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen)

- Structure containing address of destination
- Size of dest_addr structure

The **recvfrom** function is used to receive a message from the socket

- File descriptor of socket
- Application buffer in which to receive data
- Size of buf application buffer
- Bitwise OR of flags to modify socket behaviour

ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr *src_addr, socklen_t *addrlen)

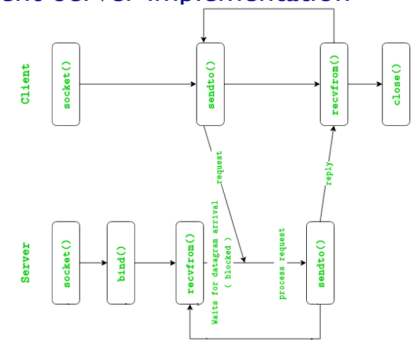
- Structure containing source address is returned
- Variable in which size of src_addr structure is returned

The **close** function is used to close a file descriptor

int close(int fd)

- Return code (is 0 if success)
- File descriptor to close

Client-Server implementation



Socket programming (TCP)

TCP is a connection-oriented transport-layer protocol, it provides guaranteed, in-order and error-free packet communications between applications on *hosts*

Before exchanging information, the client and server must establish a TCP connection

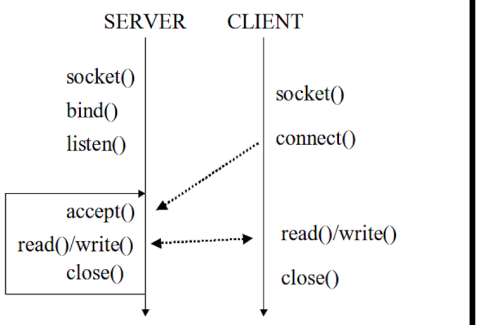
When one side wants to send data to the other side it just drops data into the TCP connection via its socket:

- Remember that with UDP you must attach a destination address to the packet before sending
- A TCP socket is already identified by a 4-tupple (destination address and port, source address and port)

A TCP Server uses two different types of sockets:

- A *welcoming socket*, where new connections are received by the server: remains open so that server can receive connections from multiple clients
- A *connection socket:* created/opened and closed as required, to handle communications with clients

Client-Server implementation



The **listen** function is used in the server to prepare the socket to receive new connections

int listen(int sockfd, int backlog)

File descriptor of socket to listen for new connection requests

How many clients are keep in the queue waiting for accept

The **accept** function is used in the server to accept a new connection (wait if none in the queue)

int accept(int sockfd, const struct sockaddr *address, socklent_t *address_len)

Returns file descriptor of socket for new connection

File descriptor of welcoming socket

Structure containing information about the new connection is returned

Variable in which size of address structure is returned

The **connect** function is used in the client to initiate a new TCP connection

int connect(int sockfd, const struct sockaddr *address, socklent_t address_len)

File descriptor of socket to initiate a new connection

Structure containing information about the address and port of the server

Variable with size of the address structure

Convert host name to IP address:

struct hostent * gethostbyname(const char* name)

Return structure with IP address corresponding to name

Name of host (server)

Network byte order is *big endian*. Convert to network byte order when sending data and using addresses and ports:

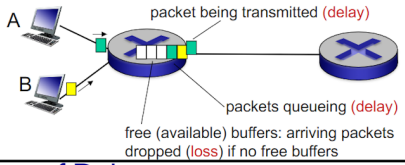
uint32_t htonl(uint32_t hostlong); **uint32_t ntohl(uint32_t netlong);**
uint16_t htons(uint16_t hostshort); **uint16_t ntohs(uint16_t netshort);**

Delay, loss and throughput in packet-switched networks

How do loss and delay occur?

packets *queue* in router buffers

- packet arrival rate to link (temporarily) exceeds output link capacity
- packets queue, wait for turn



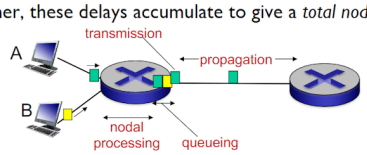
Types of Delay

The performance of many applications (search, web browsing, maps, email, etc.) is greatly affected by *network delays*

A packet suffers from several types of delays at *each* node along the path from the source to the destination

- Node processing delay
- Queuing delay
- Transmission delay
- Propagation delay

Together, these delays accumulate to give a *total nodal delay*



$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$

d_{proc}: nodal processing
- check bit errors
- determine output link
- typically < msec

d_{queue}: queueing delay
- time waiting at output link for transmission
- depends on congestion level of router
- Is a function of the intensity and nature of traffic arriving at the queue
- Can vary from packet to packet

d_{trans}: transmission delay:
- Time required to transmit the packet's bits into the link
- *L*: packet length (bits)
- *R*: link *bandwidth (bps)*
- $d_{trans} = L/R$

d_{trans} and d_{prop} *very* different

d_{prop}: propagation delay:
- *d*: length of physical link
- *s*: propagation speed of the communications link (~2×10^8 m/sec)
- $d_{prop} = d/s$

Transmission vs. propagation delay

The transmission delay is the amount of time required for the router to push out (transmit) the packet

- It's a function of the packet's length and the transmission rate of the link (bandwidth)
- Has nothing to do with the distance between the two routers

The propagation delay is the time it takes a bit to propagate from one router to the next

- Is a function of the distance between the two routers
- Has nothing to do with the packet's length or the transmission rate of the link

Total nodal delay

$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$

The contribution of the four delays can vary significantly:

- d_{prop} *can be negligible for a short link connecting two routers, or of hundreds of milliseconds for two routers interconnected by a geostationary satellite link*
- d_{trans} *can be negligible for transmission rates of 10 Mbps or higher, or hundreds of milliseconds for large packets sent over low-speed links (e.g. dial-up modems)*
- d_{proc} *is often negligible, but strongly influences a router's maximum throughput (maximum rate at which it forwards packets)*
- d_{queue} *is the most complicated (and interesting) component*

"Real" Internet delays and routes

`traceroute` program: provides delay measurement from source to router along end-end Internet path towards destination. For all *i*:

- sends three packets that will reach router *i* on path towards destination
- router *i* will return packets to sender
- sender times interval between transmission and reply.

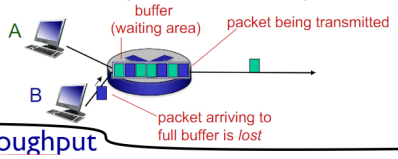
Packet loss

queue (aka buffer) preceding link in buffer has finite capacity

packet arriving to full queue dropped (aka lost)

lost packet may be retransmitted by previous node, by source end system, or not at all

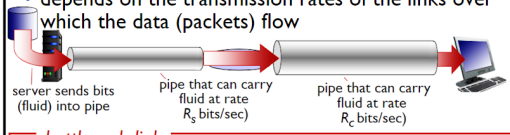
Must be handled by communications protocols



Throughput

throughput: rate (bits/time unit) at which bits transferred between sender/receiver

- *instantaneous:* rate at given point in time
- *average:* rate over longer period of time
- depends on the transmission rates of the links over which the data (packets) flow



bottleneck link

link on end-end path that constrains end-end throughput

Transmission of data

Data transmission involves converting bits in electric, electromagnetic or optical signals

Signals may suffer from:

- Attenuation: reduction of the amplitude of the signals
- Delay distortion: different speeds for different frequencies

The transmission of binary data involves:

- Modulation (AM, FM, PM, mixing)
- Power of the information signal vs Power of the noise
- A propagation delay related with the transmission medium

Maximum Data Rate (channel capacity)

Data rate governs the speed of data transmission

Depends upon 3 factors:

- Bandwidth available
- Number or levels in digital signal (resulting from modulation)
- Quality of the channel – the level of noise

Two theoretical formulas were developed to calculate the data rate in digital communication systems:

Nyquist formula

Assumes a noise-free channel

Defines the theoretical maximum bit rate, from:

- Bandwidth available (in Hz)
- Number or levels in the digital signal (resulting from modulation)

Bandwidth is fixed, so data rate is directly proportional to the number of signal levels

Nyquist formula

Factor of "2" in the formula arises from the need to sample the signal at least twice per cycle (according to the Nyquist sampling theorem) in order to accurately reconstruct the original signal.

$C = 2\ B \log_2 M$ bps

C – Maximum Bit Rate
B – Bandwidth
M – Levels of signalling

Shannon formula

Channels in reality are always noisy

Defines the theoretical highest data rate for a noisy channel, from:

- Bandwidth available (in Hz)
- The signal-to-noise (S/N) ratio
- Bandwidth is fixed, so data rate is directly proportional to S/N

$C = B \log_2 (1+S/N)$ bps

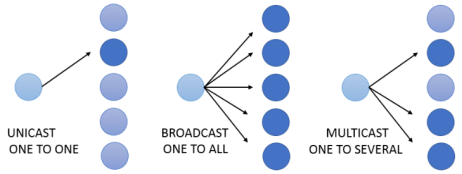
S – Signal power (Watt)
N – Noise Power (Watt)

or...
C = B log2(1 + 10^(SNR(dB)/10))
If S/N ratio in dB

- SNR may be expressed in dB (decibels), given by:
  $SNR(dB) = 10 * \log_{10} (S/N)$
  Example: For a S/N of 1000: $10 * \log_{10} (1000) = 30$ dB

Communications with IP (multicast)

IP SUPPORTS THE FOLLOWING SERVICES:



IP MULTICAST ALSO SUPPORTS A MANY TO MANY SERVICE
IP MULTCAST REQUIRES SUPPORT OF OTHER PROTOCOLS (IGMP, MULTICAST ROUTING)

Network Programming (Linux)

IP multicasting provides the capability for an application to send a single IP datagram that a group of hosts in a network can receive. The hosts in the group may reside on a single subnet or may be on different subnets (connected by multicast capable routers).

Hosts may join and leave groups at any time. There are no restrictions on the location or number of members in a host group. A class D Internet address in the range 224.0.0.1 to 239.255.255.255 **identifies a host group**.

An application program can send or receive multicast datagrams by using the socket() API and connectionless SOCK_DGRAM type sockets.

When a socket of type SOCK_DGRAM is created, an application can use the **setsockopt()** function to control the multicast characteristics associated with that socket.

The **setsockopt()** function accepts the following IPPROTO_IP level flags:

- IP_ADD_MEMBERSHIP: Joins the multicast group specified
- IP_DROP_MEMBERSHIP: Leaves the multicast group specified
- IP_MULTICAST_IF: Sets the interface over which outgoing multicast datagrams are sent
- IP_MULTICAST_TTL: Sets the Time To Live (TTL) in the IP header for outgoing multicast datagrams. By default it is set to 1. TTL of 0 are not transmitted on any sub-network. Multicast datagrams with a TTL of greater than 1 may be delivered to more than one sub-network, if there are one or more multicast routers attached to the first sub-network.
- IP_MULTICAST_LOOP: Specifies whether or not a copy of an outgoing multicast datagram is delivered to the sending host as long as it is a member of the multicast group.

sending a multicast datagram

1. Create an AF_INET, SOCK_DGRAM type socket
2. Initialize a *sockaddr_in* structure with the destination group IP address and port number
3. Set the IP_MULTICAST_LOOP socket option according to whether the sending system should receive a copy of the multicast datagrams that are transmitted
4. Set the IP_MULTICAST_IF socket option to define the local interface over which you want to send the multicast datagrams
5. Send the datagram

receiving a multicast datagram

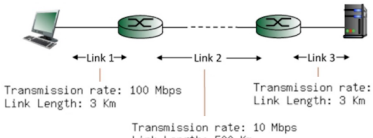
1. Create an AF_INET, SOCK_DGRAM type socket
2. Set the SO_REUSEADDR option to allow multiple applications to receive datagrams that are destined to the same local port number
3. Use bind() to specify the local port number. Specify the IP address as INADDR_ANY in order to receive datagrams that are addressed to a multicast group
4. Use the IP_ADD_MEMBERSHIP socket option to join the multicast group that receives the datagrams. When joining a group, specify the class D group address along with the IP address of a local interface.
5. Receive the datagram

Consider the following scenario, in which a single router is transmitting packets of lenght L = 8000 bits, over a single link with transmission rate R = 100 Mbps to another router:

1) What is the transmission delay (the time needed to transmit all of a packet's bits into the link)? L/R = 8000 bits / 100 Mbps = 0.08 msec

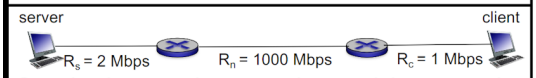
2) What is the maximum number of packets per second that can be transmitted by the link? 100 Mbps / 8000 bits = 12500 packets/sec

Considering the scenario below, find the end-to-end delay considering the transmission delays and propagation delays on each of the three links, but *ignoring queueing delays and processing delays*. The speed of light propagation delay on each link is 3x10**8 m/sec, assume a packet length of 8000 bits.



Link 1 transmission delay = L/R = 8000 bits / 100 Mbps = 0.080000 msec.
Link 1 propagation delay = d/s = 3 Km / 3*10**8 m/sec = 0.010000 msec.
Link 2 transmission delay = L/R = 8000 bits / 10 Mbps = 0.800000 msec.
Link 2 propagation delay = d/s = 500 Km / 3*10**8 m/sec = 1.666667 msec.
Link 3 transmission delay = L/R = 8000 bits / 10 Mbps = 0.800000 msec.
Link 3 propagation delay = d/s = 3 Km / 3*10**8 m/sec = 0.010000 msec.

Thus, the total end-to-end delay is the sum of these six delays: 3.366667 msecs.



Consider the network scenario above, and that you need to download a file of 32 million bits from the server to the client:

1. Assuming no other traffic in the network, what is the end-to-end throughput for the file transfer?

1 Mbps (transmission rate of the bottleneck link)

2. What is the time needed to transfer the file (ignoring any end-to-end delays)?

32 Mbits / 1 Mbps = 32 sec

Nyquist formula

- Consider a noiseless channel with a bandwidth of 3000 Hz transmitting a signal with two signal levels. What can be the maximum (theoretical) bit rate?

BitRate = 2 * 3000 * log2(2) = 6000bps

- We need to send 265 kbps over a noiseless channel with a bandwidth of 20 kHz. How many signal levels do we need?

265000 = 2 * 20000 * $\log_2(L)$
$\log_2(L)$ = 6.625
L = $2^{6.625}$ = 98.7 levels

Increasing the number of signal levels...also increases the complexity of the transmission and reception processes, and may require higher signal-to-noise ratios for reliable communication.

Shannon formula

A telephone line normally has a bandwidth of 3000 Hz (300 to 3300 Hz) assigned for data communication. The SNR is usually 3162. What will be the capacity for this channel?

C = 3000 * $\log_2$(1 + SNR) = 3000 * 11.62 = 34860 bps

The SNR is often given in decibels. Assume that SNR(dB) is 36 and the channel bandwidth is 2 MHz. Calculate the theoretical channel capacity.

SNR(dB) = 10 * $\log_{10}$(SNR)
SNR = $10^{(SNR(dB)/10)}$
SNR = $10^{3.6}$ = 3981
Hence, C = 2 * 10^6 * $\log_2$(3982) = 24 MHz

FICHA 4!

Despreze os tempos de processamento da informação;
Indique sempre as unidades que utiliza;
Use 1kB = 1024 bytes
Velocidades de propagação:
- Use a velocidade de $2x10^8$ m/s para comunicações através de condutores metálicos;
- Use a velocidade de $3x10^8$ m/s para comunicações pelo ar;

1) Considere que pretende enviar um ficheiro que ocupa 1.5 MBytes para 3 máquinas diferentes.

- Máquina A: Distância de 0.2 Km por cabo, usando um canal com uma largura de banda de 0.5 MHz e com 8 níveis por elemento de sinalização;
- Máquina B: Distância de 100 Km por ondas rádio, usando um canal com uma largura de banda de 0.5 GHz e uma codificação que usa 3 bits por elemento de sinalização;
- Máquina C: Ligação por satélite a uma altura de 45 000 Km, usando um canal com uma largura de banda de 200 KHz, uma relação S/N de 15 dB e com 32 níveis por eleamento de sinalização;

a) Supondo que não existem erros na transmissão e admitindo as velocidades máximas de transmissão teóricas, qual das máquinas irá receber o ficheiro primeiro?

A Nyquist: $C = 2 \times B \log_2(N) = 2 \times 0,5 \times 10^6 \times \log_2 8 = 3 \times 10^6$ bps
$T_f = \frac{1,5 \times 2^{23}}{3 \times 10^6} = 4,194304$
$T_p = \frac{0,2 \times 10^3}{2 \times 10^8} \simeq 10^{-6}$
$T_{total} = T_f + T_p + 0 + 0 = 4,194305$ seg

B Nyquist: $C = 2 \times 0,5 \times 10^9 \times 3 = 3 \times 10^9$ bps
$T_f = \frac{1,5 \times 2^{23}}{3 \times 10^9} = 4,194304 \times 10^{-3}$
$T_p = \frac{100 \times 10^3}{3 \times 10^8} = 3,333 \times 10^{-4}$
$T_{total} = T_f + T_p + 0 + 0 = 4,527637 \times 10^{-3}$ seg

C Nyquist: $2 \times 2 \times 10^5 \times \log_2 32 = 2 \times 10^6$ bps
$T_f = \frac{1,5 \times 2^{23}}{2 \times 10^6} = 6,291456$
$T_p = \frac{45 \times 10^6}{3 \times 10^8} = 0,15 \times 2 = 0,3$
$T_{total} = T_f + T_p + 0 + 0 = 6,441456$ seg

Shannon: $2 \times 10^5 \log_2(1 + 31,622777) = 1,005562 \times 10^6$
$SNR_{db} = 10 \log_{10}(S/N) \Leftrightarrow 10^{15/10} = S/N \simeq 31,622777$
$T_f = \frac{1,5 \times 2^{23}}{C} = 12,513338$
$T_p = 0,15 \times 2 = 0,30$
$T_{total} = 12,813338$ seg

2 – Suponha que vai enviar um ficheiro de 20MBytes de uma máquina A para uma máquina D. Para chegar a D o ficheiro terá de passar sequencialmente pelas máquinas B e C. Tendo em conta a informação seguinte, que detalha cada uma das ligações individuais, determine o Tempo de transmissão, o Tempo de propagação e a Velocidade máxima da transmissão para cada ligação. Suponha ainda que os dados do ficheiro só serão retransmitidos por uma máquina após terem sido completamente recebidos por essa mesma máquina.

- Máquina A->B: Distância de 90Km por transmissão rádio, usando um canal com largura de banda de 1.5MHz em que são transmitidos 8 bits por elemento de sinalização;
- Máquina B->C: Ligação por satélite geoestacionário situado a uma altitude de 35250Km usando um canal com uma largura de banda de 50MHz e uma relação sinal ruído de 15dB;
- Máquina C->D: Ligação rádio a uma distância de 10Km usando um canal com uma largura de banda de 100MHz e uma relação sinal ruído de 15.

a) Quanto tempo demora o ficheiro desde que é enviado até ser completamente recebido pela máquina D?

A→B Nyquist: $C = 1,5 \times 10^6 \times 8 \times 2 = 2,4 \times 10^7$ bps
$T_f = \frac{20 \times 2^{23}}{2,4 \times 10^7} = 6,990507$
$V_{máx} = \frac{90 \times 10^3}{6,990507} = 12874,60$ m/s
$T_p = \frac{90 \times 10^3}{3 \times 10^8} = 0,0003$
$T_{total} = T_f + T_p = 6,990807$

B→C Shannon: $C = B \log_2(1 + S/N)$
$SNR_{db} = 10 \log_{10}(S/N) \Leftrightarrow 15 = 10 \log_{10}(S/N)$
$\Leftrightarrow 10^{15/10} = S/N \Leftrightarrow S/N = 31,622777$
$C = 50 \times 10^6 \times \log_2(32,62277) = 2,51401269 \times 10^8$ bps
$T_f = \frac{20 \times 2^{23}}{C} = 0,6673481$ s
$T_p = \frac{35250 \times 10^3}{3 \times 10^8} = 0,1175$
$V_{max} = \frac{35250 \times 10^3}{0,6673481} \simeq 52821009,01$ m s^{-1}
$T_{total} = 0,7848481$

C→D
d = 10^4 m
B = 100 x 10^6 Hz
S/R = 15
Shannon: $C = B \log_2(1 + S/N)$
$C = 100 \times 10^6 \times \log_2(16) = 100 \times 10^6 \times 4 =$
$= 4 \times 10^8$ bps
$T_f = \frac{1}{C} = \frac{20 \times 2^{23}}{4 \times 10^8} = 0,4194304$
$T_p = \frac{d}{V_p} = \frac{10^4}{3 \times 10^8} = 3,(3) \times 10^{-5}$
$V_f = 23841,85791$ m s^{-1}
$T_{total} = 0,4194637$

3 – Suponha que é estabelecida uma ligação de 100 Mbps entre uma base terrestre e o *space shuttle* em órbita da terra a 320Km de altitude.

a) Sabendo que a largura de banda do canal existente é de 20MHz, que número de níveis por elemento de sinalização são necessários para assegurar que se conseguem obter os 100 Mbps?

B = 20MHz = 20×10^6 Hz — nº de níveis
$10^8 = 20 \times 10^6 \times 2 \log_2(N)$
$\frac{10^8}{40 \times 10^6} = \log_2 N \Leftrightarrow \log_2 N = 2,5 \Leftrightarrow N \simeq 6$ níveis

b) Qual o mínimo RTT (*Round-trip time*) para a ligação?

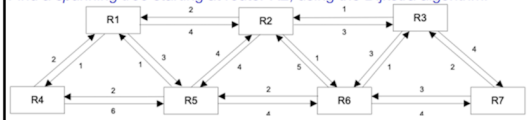
$RTT = 2 \times T_p = 2 \times \frac{d(m)}{V_{Prop.}} = 2 \times \frac{320 \times 10^3}{3 \times 10^8} = 2 \times \frac{320}{3 \times 10^5} = 2,1333333 \times 10^{-3}$

c) O controlo da missão quer fazer download de 25MB de dados do *space-shuttle*. Para isso faz um pedido ao *space-shuttle* de 100KB. Qual o tempo que decorre entre o início do pedido e a chegada de todos os dados à estação terrestre?

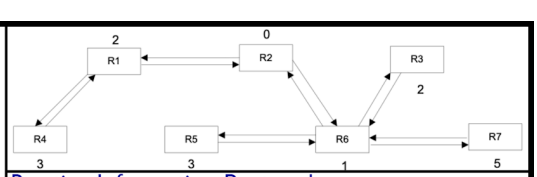
1) Enviar 100 KB a 100 Mbps: 2) RTT = 2,133 ms
$T_{trans} = \frac{100 \cdot 1024 \cdot 8}{100 \cdot 10^6} \simeq 0,008$ s = 8 ms
3) Receber 25 MB a 100 Mbps: $T_{trans} = \frac{25 \cdot 1024 \cdot 1024 \cdot 8}{100 \cdot 10^6} \simeq 2,097$ s
$T_{total} = 8$ ms + 2,133 ms + 2,097 s = 10,133 ms + 2,097 s = 2,107 s

Dijkstra algorithm

Find a *spanning tree* starting at router R2, using the Dijkstra algorithm:

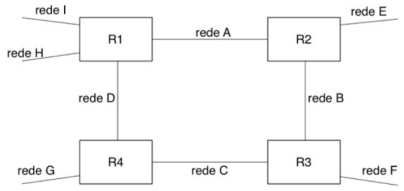


T={R2}
T={R2, R6} R2,R6
T={R2,R6,R1} R2,R1 (or R6,R3)
T={R2,R6,R1,R3} R6,R3 (ou R2, R1)
T={R2,R6,R1,R3,R5} R6,R5 (ou R1,R4)
T={R2,R6,R1,R3,R5,R4} R1,R4 (ou R6,R5)
T={R2,R6,R1,R3,R5,R4,R7}=N R6,R7



Routing Information Protocol

Consider that RIP is being used in the following network, and that all routing tables are already stabilized. Indicate the routing table for router R4 using the following syntax: **<destination network>,<next router>,<distance>**



Please note: With RIP directly connected routes are at a distance of "1", and unreachable networks at a distance of "16"

(solution):

C, - , 1	D, - , 1	G, - , 1
I, R1, 2,	H,R1,2	A,R1,2
F,R3,2	B,R3,2	
E,R3,3	(ou E,R1,3)	

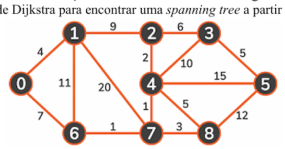
Consider now that network (link C) is down. Indicate the routing table for router R4 after the routing information has stabilized (after change has propagated to other routers using RIP):

(solution):

C, - , 16	D, - , 1	G, - , 1
I, R1, 2,	H,R1,2	A,R1,2
F,R1,4	B,R1,3	
E,R1,3		

EN 2023

Considere o cenário de rede ilustrado na figura seguinte, na qual os valores indicados representam o custo da comunicação entre os *routers*. Considere igualmente que se pretende utilizar o algoritmo de Dijkstra para encontrar uma *spanning tree* a partir do *router* 4.



Represente, com recurso a uma figura, a *spanning tree* final, resultante da aplicação do algoritmo de Dijkstra. Nessa figura deverá representar, igualmente, para cada *router*, o custo da comunicação a partir do *router* 4 até esse *router*.

Deverá indicar, igualmente, as arestas selecionadas pelo algoritmo em cada uma das iterações, pela ordem com que são selecionadas e recorrendo ao seguinte formato:

<*router* de origem>, <*router* de destino>

	node	dist	prev. n.
visit: 4, 7, 6, 2, 8, 3, 0, 1, 5	0	~~inf.~~ 9	6
	1	~~inf.~~ ~~21~~ ~~13~~ 11	~~7~~ 6 2
	2	~~inf.~~ 2	4
	3	~~inf.~~ ~~10~~ 8	~~4~~ 2
{4, 7}, {7, 6}, {4, 2}, {7, 8},	4	0	
{2, 3}, {6, 0}, {2, 1}, {3, 5}	5	~~inf.~~ ~~15~~ 13	~~4~~ 3
	6	~~inf.~~ 2	7
	7	~~inf.~~ 1	4
	8	~~inf.~~ ~~5~~ 4	~~4~~ 7

Considere que a Rede A utiliza o endereço 10.10.0.128/27 e a Rede B utiliza o endereço 10.20.0.192/28. Indique, para esta rede, a gama de endereços utilizáveis para endereçar *hosts*, o endereço da rede e o endereço de *broadcast*:

Rede:	Gama de endereços:	Endereço da rede:	Endereço de *broadcast*:
Rede A	10.10.0.129 10.10.0.158	10.10.0.128	10.10.0.159
Rede B	10.20.0.193 10.20.0.206	10.20.0.192	10.20.0.207

Considere que pretende ativar, no *router* R3, SNAT (Source NAT) para todas as comunicações IP com origem na Rede D e destinadas às outras redes do cenário. Indique os comandos que adicionaria à configuração desse *router* para esse efeito. Deverá considerar que dispõe da Rede 10.30.0.128/26 para endereçar as Redes C e D.

Rede C: 10.30.0.128/27 Rede D: 10.30.0.160/27

```
config terminal
access-list 30 permit 10.30.0.160 0.0.0.31
ip nat inside source list 30 interface Ethernet0/1 overload
interface FastEthernet0/1
ip address [IF address] [netmask]
no shutdown
ip nat outside
exit
interface FastEthernet0/0
ip address [IF address] [netmask]
no shutdown
ip nat inside
end
```

do lado da rede C

se IPs das IFs ainda não configurado...

do lado da rede D!

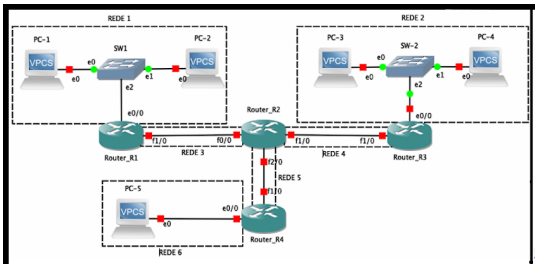
Considere uma sessão de comunicação por TCP através de um link em que a taxa de transmissão é de 10Mbps e o *delay* de propagação é de 10ms. Ignorando outros delays, calcule o tamanho mínimo da janela que o TCP deve utilizar, para a mesma não limite o desempenho da comunicação.

Window size = RTT * B = 2 * 10ms * 10Mb/s = 200000b = 50000B

Considere um sistema de comunicação em que a largura de banda disponível é de 1000 Hz e a relação sinal-ruído é de 20 dB. Calcule a taxa máxima de transmissão (teórica) para este canal de comunicação (indique todos os cálculos necessários).

SNR = 10^(20/10) = 100 C = 1000 * log(2)(1 + 100) = 6658,2115 b/s

Encapsulamento: funciona de cima para baixo enquanto as mensagens são enviadas numa rede. Em cada camada superior são considerados dados dentro do protocolo encapsulador. PE, o segmento TCP é considerado dados do pacote IP. O desenc. é o processo de enc. revertido no host recetor, e é usado por um ou mais cabeçalhos de protocolo. Os dados são desenc. à medida que sobrem na pilha em direção à aplicação do utilizador final.



Cenário: Na resolução das questões seguintes deverá considerar o cenário ilustrado a seguir, no qual as redes "REDE 1", "REDE 2" e "REDE 6" encontram-se interligadas através das redes "REDE 3", "REDE 4" e "REDE 5".

Rede:	Gama de endereços:	Endereço da rede:	Endereço de *broadcast*:
REDE 1	10.1.144.1 10.1.159.254	10.1.144.0	10.1.159.255
REDE 2	10.10.10.1 10.10.10.254	10.10.10.0	10.10.10.255
REDE 6	10.20.20.1 10.20.20.62	10.20.20.0	10.20.20.63

Considere que dispõe da gama de endereços IPv4 172.16.1.176/28 para endereçar as Redes 3, 4 e 5 (as redes de interligação dos *routers* no cenário). Segmente esta gama de forma a que todas as redes fiquem com o mesmo número de endereços disponíveis e indique, para as redes indicadas na tabela seguinte, a seguinte informação:

	REDE 3:	REDE 5:
Endereço da Rede:	172.16.1.180	176.16.1.188
Endereço de *broadcast:*	172.16.1.183	176.16.1.191
Gama de endereços disponíveis para endereçar *hosts*:	172.16.1.181 -> 172.16.1.182	176.16.1.189 -> 176.16.1.190

Utilizando a gama de endereços IPv4 convencionada anteriormente para a REDE 6, atribua um endereço à interface *e0/0* do *router Router_R4* e uma configuração de rede adequada ao *Computador PC-5*, indicando a seguinte informação de configuração a utilizar:

	Computador PC-5:	Interface e0/0 do Router_R4:
Endereço IP:	10.20.20.1	10.20.20.62
Máscara de Rede:	255.255.255.192	(e0/0 de R4 e PC5 estão na Rede 6)
Endereço do *default gateway*:	10.20.20.62	

Considere que o *router Router_R2* utiliza o sistema IOS da Cisco. Tendo em consideração as configurações atribuídas anteriormente, caso tivesse que configurar as rotas no *router Router_R2* utilizando apenas encaminhamento estático e o comando "ip route", que comando(s) teria que usar?

Nota: a sintaxe deste comando é a seguinte:
ip route {destination_network} {subnet_mask} {default_gateway}

Resposta:
ip route 10.1.144.0 255.255.240.0 172.16.1.181
ip route 10.10.10.0 255.255.255.0 172.16.1.185
ip route 10.20.20.0 255.255.255.192 172.16.1.189

Considere as comunicações entre os *routers* Router_R1 e Router_R3 do cenário, através do *router* Router_R2. Considere igualmente que os *links* de comunicação apresentam as seguintes características:

Link R1-R2: distância: 10km, velocidade de transmissão: 10Mbps
Link R2-R3: distância: 20km, velocidade de transmissão: 20Mbps

a) Considere que o *delay* total de *queuing* e processamento dos pacotes IP nos *routers* é de 0.05 ms. Calcule o *delay* total das comunicações entre os *routers* Router_R1 e Router_R3, considerando a transmissão de um pacote com 10000 bits e a velocidade de propagação de $2x10^8$ m/s. Deverá considerar igualmente que os *routers* operam no modo *store and forward*.

R1 -> R2:
Ttrans = 10^4b / 10^7Mbps = 1ms
Tprop = 10^4m / 2*10^8m/s = 0,05ms
R2 -> R2
Ttrans = 10^4b / 2*10^7Mbps = 0,5ms
Tprop = 2*10^4m / 2*10^8m/s = 0,1ms

Ttotal = 1 + 0,05 + 0,5 + 0,1 + 0,05 = 0,0017s

b) Considere que se pretende transferir 20000 bits de informação entre os *routers* Router_R1 e Router_R3, e assuma que não há mais comunicações entre estes *routers*. Qual é a taxa de transmissão máxima para as comunicações entre os dois *routers*?

A taxa de trans. máxima é igual ao menor valor de velocidade de trans., sendo neste caso igual a 10Mbps.

Ignorando agora os *delays* nas comunicações entre os *routers* Router_R1 e Router_R3, qual é o tempo total para transmissão dos 20000 bits de informação do *router* Router_R1 para o Router_R3?

Igual a a), mas com um pacote de 20000b e sem o delay de queue + proc

Considere o protocolo TCP na camada de transporte. De que forma uma aplicação servidor deverá usar sockets e portos, por forma a garantir que o servidor suporta o acesso simultâneo de vários clientes? Justifique a sua resposta (nota: não é necessário apresentar código).

Um socket TCP é identificado por 4 parâmetros: endereço de origem, porto de origem, endereço IP de destino e porto de destino. Como cada socket é identificado por estes 4 parâmetros, o servidor TCP pode ter vários TCP sockets em simultâneo. Assim, de início, é feita a multiplexagem (adicionada informação sobre o cabeçalho) e no destino desmultiplexagem. Deste modo pelo simples facto de podermos ter diferentes portos de origem e o mesmo destino, devemos indicar os 4 parâmetros.

Considere que pretende implementar um servidor de música, cujo objetivo é o de permitir o acesso em *streaming*, por parte de clientes, aos ficheiros de música alojados no servidor. Considere igualmente que o protocolo de transporte a usar nas comunicações entre o servidor e os clientes é o UDP (User Datagram Protocol). De que forma a aplicação servidor deverá usar *sockets* e portos, por forma a garantir que o servidor suporta o acesso simultâneo de vários clientes? Justifique a sua resposta (nota: não é necessário apresentar código).

Um socket UDP é apenas caracterizado pelo IP de destino e porta de destini. Desta forma, o servidor tem um socket com uma porta com a qual todos os clientes comunicam. Assim, apenas os clientes e servidor devem ter sockets e portas diferentes. O servidor sabe o cliente a que tem de responder pq tem essa infornação no seu datagrama.

Suponha que pretende efetuar um *download* de dados entre uma estação terrestre e a Estação Espacial Internacional (EEI). Para cada ligação, a estação terrestre tem no máximo 30s para completar o *download* total dos dados. Supondo que a distância entre a estação terrestre e a EEI é de 400Km, que cada pedido de *download* enviado pela estação terrestre usa um pacote de tamanho total de 500bytes, que a resposta enviada pela EEI é um pacote com tamanho total de 20MB, que o tempo de processamento do pedido na EEI é de 2seg e que o canal usado tem uma largura de banda de 1MHz, qual será o número mínimo de estados de sinalização que teremos de usar para assegurar que a transmissão possa ser realizada no tempo pretendido? Suponha que os dados só poderão começar a ser transmitidos pela EEI após o pedido ter sido completamente recebido e que a velocidade de transmissão é a mesma em ambos os sentidos.

Notas:
- Suponha que não existem erros durante as transmissões e admita as velocidades máximas de transmissão teóricas;
- Assuma que a velocidade de propagação das ondas de radio-frequência através do ar é de $3x10^8$ m/s;
- Indique sempre as unidades que utiliza;
- Caso não consiga realizar todos os cálculos indique-os apenas.

500B = 4000b 20MB = 160 * 2^20b

Terrestre
Tp = 4 * 10^5m / (3 * 10^8m/s) = (4/3)ms
Tx = 4000b / (C = 2 * 10^6 * log(2)(M)) = 2 * 10^-2 / log(2)(M)

Espacial
Tp = (4/3)ms
Tx = (160 * 2^20 / C) = (2^23 / (10^5 * log(2)(M)))

30 = 0.004/3 + 0.02/log(2)(M) + 0.004/3 + (2^23 * ^10^-6 / log(2)(M) + 2 <=>
...

Para que serve o campo *Window* no cabeçalho dos pacotes TCP?

O campo "Window" no cabeçalho dos pacotes TCP é usado para o controlo de fluxo na comunicação TCP. Ele especifica a quantidade de dados que o recetor está disposto a receber.
O valor do campo "Window" é a quantidade de bytes que o recetor tem disponível em seu buffer para receber dados. Quando o emissor recebe um pacote de confirmação (ACK) do receptor, ele verifica o valor do campo "Window" para saber quanto mais pode enviar antes que o receptor fique sobrecarregado.
Se o campo "Window" for definido como zero, isso indica que o recetor está temporariamente sobrecarregado e não pode receber mais dados. O emissor então para de enviar dados até que receba um pacote com um valor de "Window" maior que zero.
Assim sendo, o campo "Window" é crucial para garantir que a comunicação TCP seja eficiente e que o receptor não seja sobrecarregado com mais dados do que pode processar.