Computer Network - collection of computers interconnected
- components: • hosts (computers)
• links
• switches / routers / hubs
• access points

unit of communication = frame (electrical signal)
LAN = only switches & hubs
TC
switch - amount of chit-chat is minimum
hub - signal is sent to everyone; scrambled signals can happen, collision span is larger
switches are usually better
- look up MAC addresses (physical) from the factory
host puts in the frame source and destination MAC address
LEDs signaling collisions => resend the signal
broadcast: computer talking to all computers linked to a switch (hub-like behaviour)
unicast: two hosts talking to each other
destination MAC for broadcast: FF FF FF FF FF FF
hub bandwidth < switch bandwidth
multicast: one host talking to a controlled subset of machines
broadcast is only possible in LAN, while multicast is possible out of LAN
routers - interconnect networks
frame - contains IP addresses in the datagram

TCP/IP

- host needs identifier => unique IP address
- port - identifies process; ex: HTTP 80, HTTPS 443
data loss, timing (delay), bandwidth
TCP - reliable transport, no data is lost, congestion control, no timing
UDP - unreliable transport, no timing, possible data loss, but very quick
TCP regulates traffic, UDP does not
there are apps that tolerate not hearing those things, like Skype's call part

socket - OS controlled interface
TCP: - server listening for clients and must have a socket welcoming the client
- client creates TCP socket and binds to it an IP and a port and connects to it
- creating socket is immediate, while accept, receive are blocking calls
1-1024 ports are reserved
AF_INET - address family (TCP/IP)
INADDR_ANY = 0.0.0.0 IP address is 4 bytes => integer
listen = creates a queue; at most that number of connections
loopback - allows to run network apps while not connected
send, recv - TCP/IP specific and give greater control to the programmer
write, read - more general
netstat to see used ports
socket types: UDP = SOCK_DGRAM, TCP = SOCK_STREAM
sockaddr has the field sa_data[14], which is opaque
this gets split into 3 parts in sockaddr_in: sin_port (2 bytes), sin_addr (4 bytes) and sin_zero[8] not used
union in C: chunk of memory used to store several variables (C polymorphism)
in_addr is a union; it can be filled with anything we want on 4 bytes
send returns the number of bytes sent and successfully received, recv returns the number of bytes received
both return -1 in case of an error
shutdown - close one of the send / receive ends of a socket

TCP - connection oriented
- whatever you send is received, like a queue
- guaranteed data delivery + guaranteed data ordering delivery
server types - iterative servers (blocking) - handle clients one after another
- concurrent servers (blocking) - fork, threads
- concurrent multiplexed servers (select)
blocked - until a call fails or succeeds, the app is stuck at that part of the code
DNS: get hostbyname, gethostbyaddr return hostent
hostent has a list of addresses and a list of aliases
a machine may have multiple IP addresses
uint32_t to ensure integers are 4 bytes
big endian = most significant byte first little endian least significant byte first
always send on socket on big endian because you don't know the architecture of the other machine
float to network: look at it like it's an integer and apply htonl
double to network: swap the bytes manually by looking at it like a char array

UDP - connectionless, datagram oriented
datagram = basic transfer unit; individual messages
recvfrom, sendto must specify the destination address
read can be used as well
no guarantee for delivery or ordering
less overhead than TCP, less latency, higher bandwidth, can broadcast
packet = more 64kb, send more ⇒ the rest is lost
CRC = cyclic redundancy check; it is a checksum
one party can overflow the other
no listen, no accept because there's no connection
sendto returns -1 only in case of locally detected errors
1 sendto matches 1 recvfrom
each packet has a priority; routers are the ones discarding packets
recvfrom is blocked until you get some data
socket options - SO_REUSEADDR - reuse local addresses; get rid of addresses after interrupting
SO_BROADCAST - enables broadcast
blocking ⇒ process is put to sleep
set socket to nonblocking ⇒ calls won't block, but will return EWOULDBLOCK instead
multiplexed (select) - allows to watch over multiple file descriptors, and be notified when something changes; process is sleeping for a controlled interval of time
signal i/o model - signaled when data is available
async read - returns control but data is being waited for
select - watches sockets for read / write / exceptions for a given interval of time
- returns -1 on error, 0 on timeout and positive count of ready descriptors otherwise
- sets is destructive with respect to the given sets
- NULL timeout ⇒ wait indefinitely

broadcast does not work on TCP because you must have an existing connection
broadcast destination address [255.255.255.255 (universal broadcast)
192.168.0.255 (LAN broadcast) = largest IP

nc - send pings
there can be an unlimited number of broadcasters
ping - Windows sends 4 pings, Linux sends until it is stopped
- checks whether a machine is up and running; you can ping a broadcast address
netstat - sockets connected
traceroute - route packets

reset flag = 1 => connect to TCP 2

protocol - an agreement about communication
- specifies format, meaning, rules of exchange and error handling
- possible problems: corrupted bits, entire packets lost, duplicated packets, packets out of order

network = stack of layers
network architecture = set of layers and protocols
layers: physical, data link, network, transport, session, presentation, application

physical layer: transmits a stream of raw bits over a communication channel
unit of transmission: signal

data link layer: turns raw transmissions into error-free communication
handles flow control
unit of transmission: frames = thousand of bytes

network layer: controls the operation of a subnet
routes packets from a source to a destination unit of transmission: packet
handles congestion control (saturated routers)
quality of service = data priorities
fragmentation = when data to be sent is too big

transport layer: accepts data from upper layers and splits it into packets
ensures that packets arrive correctly => reliability
end-to-end
unit of transmission: packets (UDP), segments/sessions (TCP)

session layer: allows establishing sessions
unit of transmission: control message

presentation layer: - data translator for the network
- ntohl, htonl, string encoding happens there

application layer: - this is what we implement
- unit of transmission: message (user data)

OPEN SYSTEM INTERCONNECTION
TCP/IP has 4 layers => mappings are not 1 to 1
RFC - request for comments

Network layer - Internet Protocol (IP)
IP address: 32-bit identifier for host/router interface => 2^{32} IP addresses
interface: connection between host/router and physical link
LAN = can physically reach each other without a router
all routers have a routing table => destination, gateway, interface
if the IP addresses were unstructured => routing tables $\approx 2^{32}$ entries => it can be very slow

5 classes of networks:
A: 1 byte network, 3 bytes hosts, network prefix: 0
B: 2 bytes network, 2 bytes hosts, network prefix: 10
C: 3 bytes network, 1 byte hosts, network prefix: 110
D: multicast with prefix 1110
E: experimental with prefix 1111
255.255.255.255 is not used
classfull - allocate to a network an entire class
network address - smallest IP in the network
broadcast address - greatest IP in the network

CIDR: classless interdomain routing
address format: a.b.c.d/x, x is the number of network bits
reduces sizes of routing tables because of subnetting
the beginning address must be divisible by the number of addresses in the block
masks are flexible => networks can be divided into smaller subnets
all classes have natural masks
classfull is limited
we cannot have subnets with 2 IPs
supernetting: multiple routing entries become one

127.0.0.0/8 - local host
0.0.0.0 is the default route for a router
ip datagram: fields, source and destination ip addresses, data
data link: MAC addresses
network: ip addresses
transport: ports

ARP - address resolution protocol
- when sending a packet to another network, ip destination is the same, but the MAC address is that of the router

ip datagram
- 20 bytes of ip + data
- version (4 bits) + header length (4 bits) + type of service (1 byte) + length (2 bytes)
16 bit identifier (2 bytes) + 13 bit offset for fragments + DF/MF (don't fragment / more fragments)
TTL (time to live, 1 byte) - to avoid packets that never die, decremented on each router
upper layer protocol (1 byte) - UDP, TCP, ICMP
checksum (2 bytes) - to check if there are any errors
each router recomputes the checksum because of TTL
checksum = 16 bit 1's complement of the 1's complement sum of all 16-bit words in the header
fragmentation: networks have MTU (max transfer unit)
regular ethernet: 1500 → 9000 bytes (jumbo frames)
ip splits data when necessary, but reconstructs it only at the destination
offset: MTU - 20 bytes

ARP broadcasts with source ip address, destination ip address, its MAC address and an empty MAC address and waits for it to be filled

NAT - network address translation
addresses are hidden
private ips - no router in internet will route them
go out in the internet with a private ip
router changes packet ip address to its public address
NAT translation table
- a different port for each different connection
- one ip address for all services ⇒ local addresses can change
- devices inside network not explicitly addressable
- ~60000 simultaneous connections for UDP + another 60000 for TCP
- routers should only process up to layer 3
- address shortage should be solved by IPv6

ICMP - signal things, error reporting (for UDP/TCP), echo request
- 4 bytes header: 1 byte type, 1 byte code, 2 bytes checksum

traceroute - UDP Linux, ICMP Windows
- uses IP TTL to trace paths ⇒ TTL=1 get first router; TTL=2 get second router; ...
- uses highly unlikely to be used UDP port

UDP - source port and destination port (2 bytes each)
- length (2 bytes) ≥ 8, checksum for the entire datagram (2 bytes)
- 28 bytes + data

TCP - each segment needs confirmation, info network and peer status
- source port and destination port (2 bytes each)
- sequence number (4 bytes) - incremented for each segment sent
- allows segments to be sent in order
- acknowledgement number (4 bytes) - helps confirm all segments we sent
- TCP header length + not used area + flags (2 bytes)

format, meaning. ...
possible problems; ...
network = stack of la...
network archit...
layers

acknowledge flag
- reset flag = 1 => connect to TCP ? no one listening
- syn, fin = 1 when setting and shutting down a connection
- window size (2 bytes) - how much data you're willing to accept
- urgent pointer (2 bytes) - not used
- options - setsockopt

TCP segments:
- data is split into segments; each segment is carried into one or multiple IP datagrams
- each needs acknowledgment
- buffers for sending and receiving
- state: seq. no. - amount of bytes sent
  last acknowledged byte
- connect: request, granted, ack
  seq - random number (local 0)
  syn = 1 during connection establishment
  ack =) acknowledge) got the state variables
- teardown: 4 steps
  client closes its half (FIN), server acknowledges it
  server sends the remaining data, server closes its half (FIN), client acknowledges it
- sequence numbers:
  - used to reassemble data
  - increments based on the number of bytes in the TCP data field
  - ACK indicates the next byte number to receive
- sliding window
  - all data in the window is allowed to be sent without an acknowledgement
  - when acknowledgment arrives, the window is shifted
  - if we sent all data and no acknowledgment has been received, TCP stops
- receiver buffer:
  - app may not be reading data as it comes
  - window size: still free space
  - sender is allowed to send the minimum between its sliding window size and the advertised window size
  - sender keeps sending TCP segments of size • when the advertised window is 0 (flow control)

TCP will retransmit a segment upon expiration of a timer
round trip time - send segment and await acknowledgement
estimated RTT = (1-α) estimated RTT + α sample RTT α ∈ (0,1), α = 0.125
dev RTT = (1-β) dev RTT + β | sample RTT - estimated RTT |
timeout Interval = estimated RTT + 4 dev RTT
first acknowledgement is lost => it is fine as long as the second one is received
congestion: too many sources sending too much data
  - lost packets, long delays, router just starts dumping packets
end-to-end congestion - congestion is inferred
congestion window:
- starts with one segment
- TCP can send min (sliding Window, advertised Window, congestion Window)
- congestion window is doubled until first loss
- loss: timeout or 3 duplicate acks
- ack is cumulative, but doesn't allow gaps
- 3 dup. acks: window is halfed, then it grows linearly () have some loss, but data is still travelling)
- timeout: window set to 1 (this is bad)
- TCP is fair: all computers use the same bandwidth approximatively

...application
...a communication channel
...transmission: signal
...turns raw transmissions into error-free ...
handles flow control
unit of transmission: fr...
network layer: controls the oper...
router ...

routing
- determine good paths
- static or dynamic
  static: routes change slowly
  dynamic: routes change quickly, periodic updates
- global: all routes have complete info
- decentralised: routes only knows its neighbours

Routing Information Protocol (RIP)
each router sends the routing table to its neighbours regularly and updates it accordingly
- v1 - classfull addressing
- v2 - CIDR
- when a link is down, there may be a loop happening = counting to infinity
- RIP ∞ = 16 to limit counting to infinity
- RIP v2 poisoned reverse - A does not send entries to B passing through B

Application level Protocols

DNS - domain name server
- names are easier to remember
- distributed database of domain addresses and their IP addresses
- hierarchical naming system
  host name . domain . TLD
  TLD - top level domain standardised
  domain + TLD - name that you buy

DNS Software
- resolver: name → info
- name server: DNS database
- originally one single central table

Servers
- primary / master - authoritative, updates regularly
- secondary / slaves - temporarily authoritative, read-only copies of the database
- forwarders - resolver only

queries - recursive or non-recursive

types of records:
- A - host to IP
- NS - who is the name server for a domain
- CNAME - alias
- MX - mail

TTL - time a record should be kept alive

DNS - 12 bytes UDP

FTP Protocol - clear text, text protocol, exchanging files between machines
- 2 channels: control channel + data channel
- modes: active, passive
- TCP

SMTP Protocol - mail exchange linked to DNS
- text protocol which allows encryption (TCP) and offline message exchanging

HTTP Protocol - text protocol, TCP, which allows you to connect to a website