Datamat NetværksArkitektur

- Note kompendium

Forord

Dette kompendium er skrevet på baggrund af noter lavet i forbindelse med DNA-kurset (Datamat- og Netværksarkitektur) på 4.semester – Informatik-uddannelsen ved Aalborg Universitet, forår 2008.

Kompendiet er udarbejdet til forelæsninger med John Hansen & Joseph C. Okkika af Rahuvaran Pathmanathan & rettet til af Michael Bønnerup. Illustrationer og figurer er hentet fra slideshows fra forelæsningerne, og rettighederne til brugen af disse findes hos forelæserne John & Joseph.

Kompendiet er gratis og ligger frit tilgængeligt for alle.

Aalborg, maj 2008

Indholdsfortegnelse

DNA

Computer Architecture

Content: This course provides an introduction to fundamental computer architectures. Topics to be covered include: from silicon to super computers, microinstructions, pipelining, instruction set architecture, memory-models, data types, instruction formats, etc.

Computer architecture has a strong relation to other semester courses, especially PPS and SPO. One example is how to make a compiler generate an effective assembly implementation. This is not possible without a clear understanding of what assembly code really is and how it is executed.

Network Architecture

Content: This course provides an introduction to fundamental concepts in the design and implementation of computer communication networks, their protocols, and applications. Topics to be covered include: overview of network architectures, applications, network programming interfaces (e.g., sockets), transport, congestion, routing, and data link protocols, addressing, local area networks and wireless networks. Examples will be drawn from the Internet (e.g., TCP, UDP, and IP) protocol suite. There will be exercises and hands-on labs after each lecture and of course, the final exam.

Network Architecture in 10 Steps: The course aims to give students a broad foundation in network architecture in ten steps (lectures). We shall read the book *Computer Networking: A Top-Down Approach* by James F. Kurose and Keith W. Ross. It is fourth edition of a text with lots of improvement from the previous editions. The book gives a broad coverage of its theme and encourages the reader to understand underlying concepts by treating network architecture in a top-down approach. It touches also on many related topics from WAN and multimedia.

Objectives: Upon successful completion of the course, the student should be able to:

- Describe the core and edge of the Internet
- Define and exemplify circuit-switched and packet switched networks
- Understand and explain application, transport, network and link layer protocols of the Internet
- Explain how Internet protocols such as TCP/IP provide for the end-to-end exchange between users on different networks having different network architectures.
- Compare and contrast TCP/IP and UDP transport protocols
- Write socket-based programs using HTTP, SMTP, and UDP protocols.
- Explain techniques for flow/congestion control, routing, and resource allocation that are used on individual networks and on the Internet.
- Understand wireless networking

Literature

The course is based on the following texts:

For Computer Architecture part, we shall follow the following text::

Structured Computer Organization (5e)

Edition: 5th edition

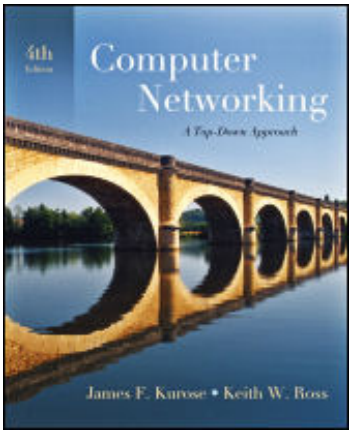
Author: Andrew S. Tanenbaum

Publisher: Prentice Hall

Language: English

ISBN: 0-131-48521-0

For Network Architecture part, we shall follow the following text:



Computer Networking: A Top-Down Approach (4e)

Edition: 4th edition

Author: James F. Kurose, Keith W. Ross

Publisher: Addison-Wesley

Language: English

ISBN-10: 0321497708

ISBN-13: 9780321497703

1. Computer architecture

Mikroprocessor Pentium 4 CPU (den fysiske del)

Mikroarkitektur Layout for hardwaren

Instruktionsarkitektur Fortæller hvad den skal gøre. Den er forskellige fra hardware/software og skal overholde ISA. De er såkaldt identiske alle sammen.

Computerarkitektur ISA & MP, danner tilsammen computerarkitekturen

Programmeringslag Der er i alt 5 generelle lag/niveauer

MMM lag Ud af de 5 niveauer , ser vi kun på de 2 første niveauer.

Harward vs. Neumann

Von Neumann CPU skal hente i en og samme chip.

Harward Lager til data og et program.

Når du i dag køber en Pentium 4 i dag, får du både en Von Neumann og Harward i den samme.

CISC vs. RISC

CISC Der er mange adresserings modes og mange operationer. Den er meget lille men også langsom.

RISC Hente og lagre, samt der er mange instruktioner derpå. Den er større men hurtigere. Modsat CISC på mange punkter.

De er også begge i Pentium i dag, men kører for det meste efter RISC, selvom den i dag kaldes CISC.

Arkitektur

X86 arkitektur Den blev oprettet i 1978. Hello World på en maskine fra 1978 kan i dag køres på en normal PC. I dag er alle chips nemlig bagud kombitabel.

RISC arkitektur

MIPS	Mikroprocessor dog uden pipelines
SPARC	Sun Micro Systems
ARM	Designer af chips og sælger dem
POWER:	Bliver brugt i PS3, Nintendo WII, Xbox og andre. Fra CISC til RISC.

Chips

Intel 86 En chip som findes i alle andre i dag.

Zilog 86 Dens arkitektur er anderledes, men da den overholde ISA reglerne kan den køre det.

Supercomputere

Intel er den første og største producent. Intel har omkring 80 % af markedsandelen.

Der bliver lavet omtrent 8 milliarder mikroprocessorer, lavet sidste år, hvor Intel kun har 2

procent af. Intel mikroarkitektur er i dag på 65nm, men med tiden bliver det måske bedre.

Woodcrest Bliver brugt til servere mv.

Conroe Bliver for det meste brugt i stationære

Merom Bliver for det meste brugt i mobile systemer.

Moores lov Hvert 2. År, doubles produktionen af transistorer. Næste generation er 45nm (penryn) og dernæst kommer der i 2012 på 22nm (gesher).

Cache Registrerprogram, hvis du har brug for mere, skal man endnu et level op, og det bliver billigere program på den måde, men dog bliver det langsommere. Dog kan vi ikke mærke det i dag i vores kodegenereringer, men jo større programmer desto mere tydeligere kan man se det.

Beregning af strøm **P**=$CV^{2}*F$

Embedded Mikroprocessorer

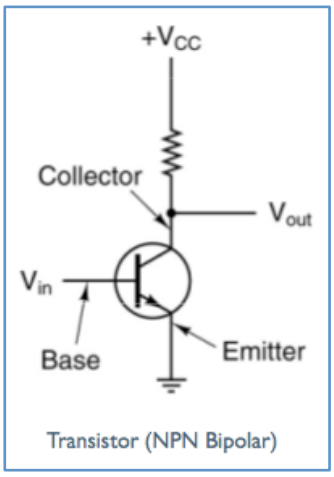
K750i ARM arkitektur,

HP iPAQ ARM arkitektur.

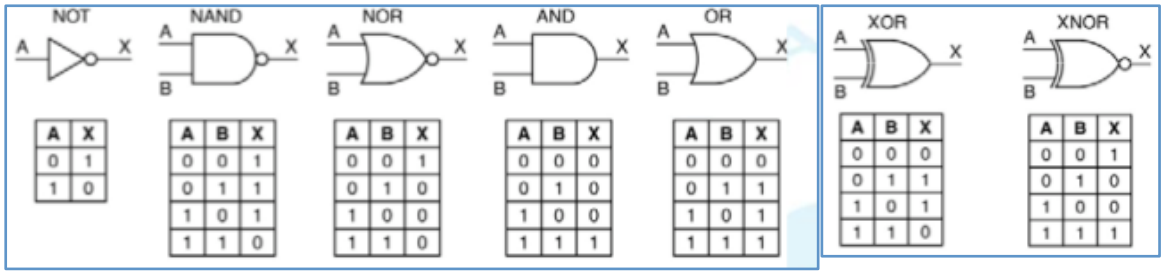
Texas instruments ARM arkitektur.

2. The Digital Logic Level

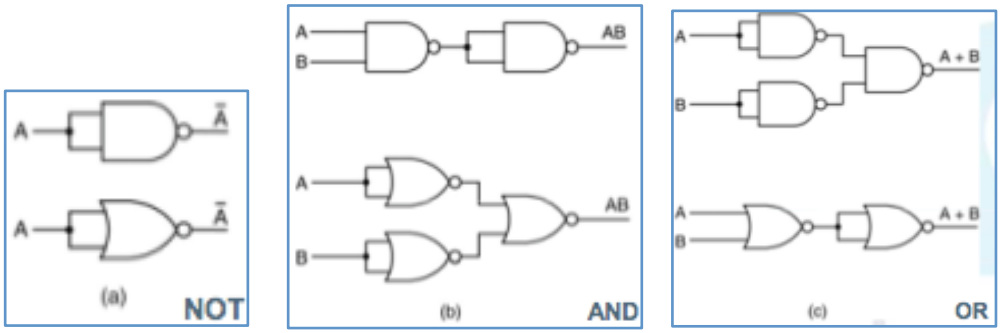
Transistor



De 5 + 2 Basale Logiske Gates

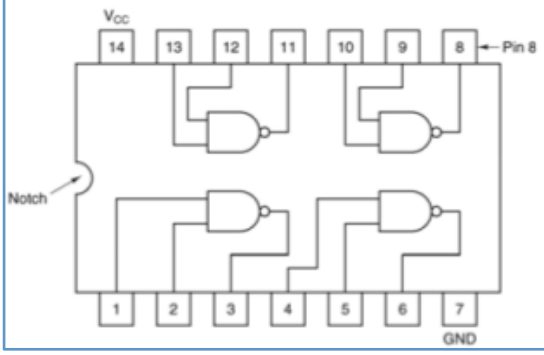


Boolesk algebra



Integrated Chips (IC)

Abbreviation	Name	Number of Gates
SSI	Small-Scale Integration	1 to 10
MSI	Medium-Scale Integration	10 to 100
LSI	Large-Scale Integration	100 to 100,000
VLSI	Very-Large-Scale Integration	more than 100,000



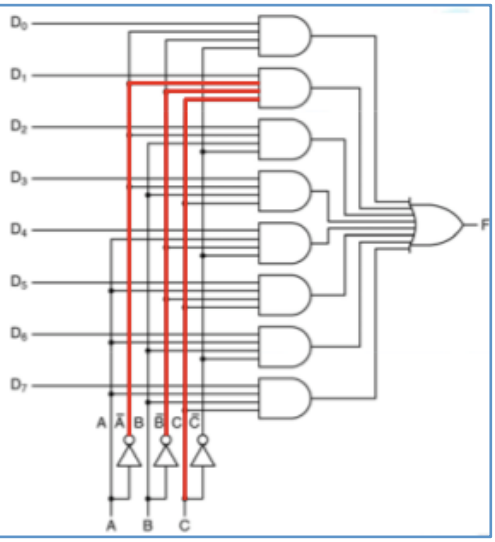
En SSI chip indeholdende 4 NAND gates.

En 8-input multiplexer:

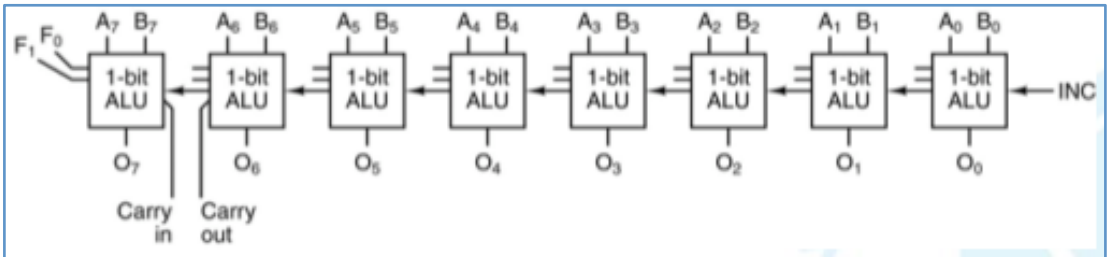
Input/output
- 2^3 data input (D0-D7)
- 3 kontrol input (A,B,C)
- 1 output (F)

Kontrol input selekterer nøjagtig et data input

Eksempel:
- (D0-D7) = 0100 0000
- (A,B,C) = 001
- F=?

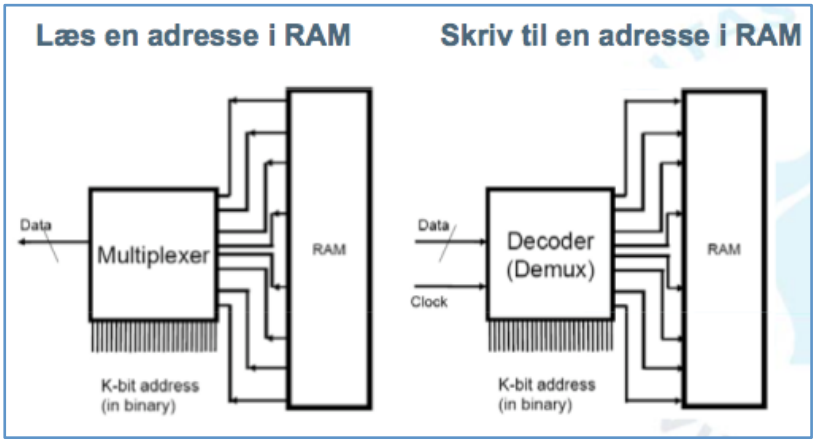


Opbygning af en 8-bits ALU



Otte 1-bits ALU'er sat i serie kan udgøre en 8-bits ALU.

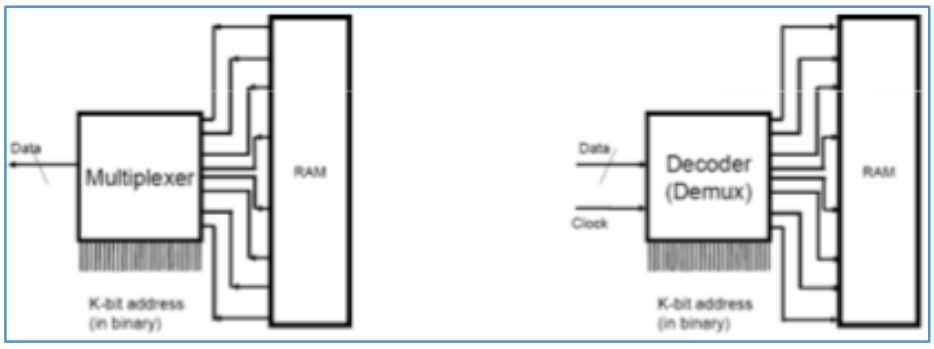
Anvendelse af multiplexer / demultiplexer



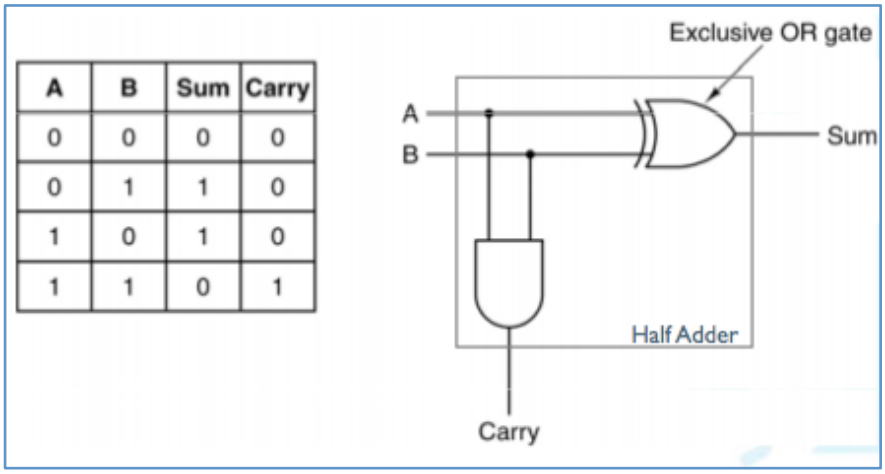
Definition

En multiplexer selekterer 1 linje bland 2^n linjer. En demultiplexer transmitterer data fra 1 linje til én blandt 2^n mulige output linjer.

Kort fortalt, en multiplexer selekterer en input linjer, en demultiplexer selekterer en output linjer..

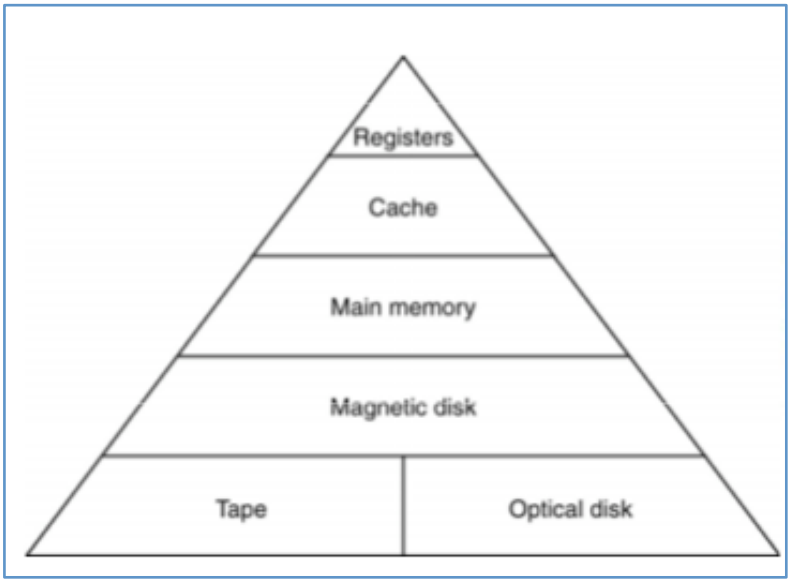


Aritmetiske kredsløb

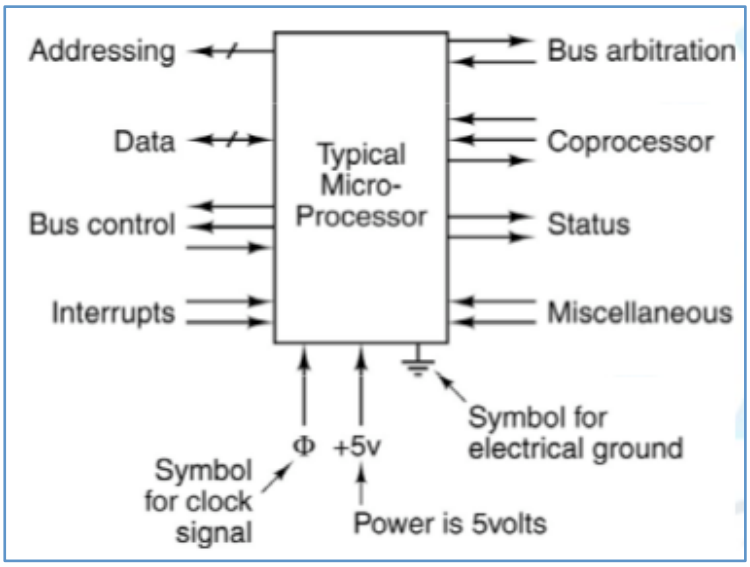


A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

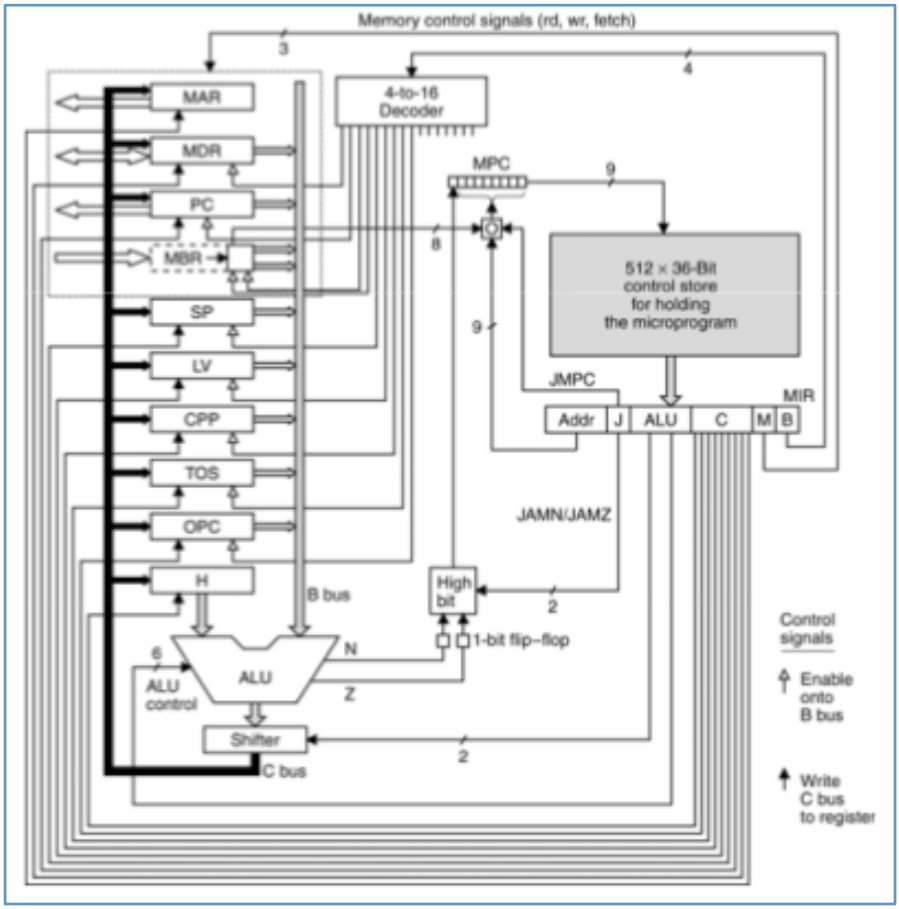
Memory


Registers
Cache
Main memory
Magnetic disk
Tape
Optical disk

CPU

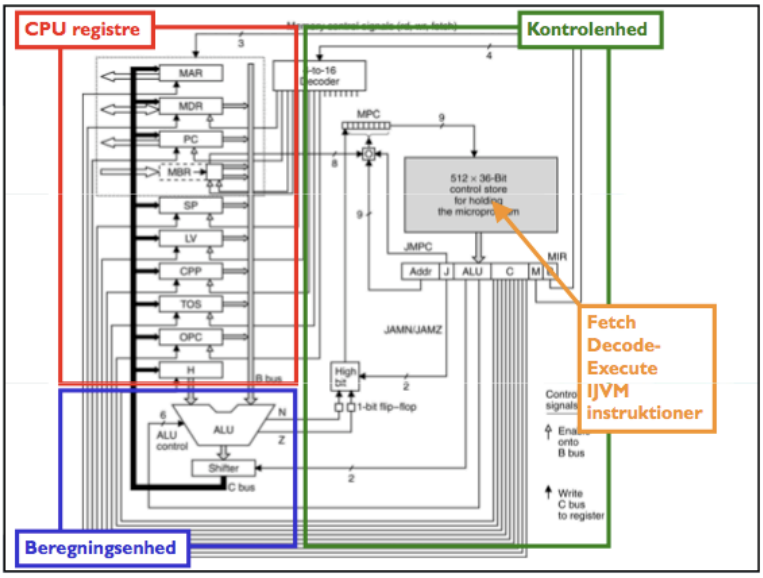

Addressing
Data
Bus control
Interrupts
Typical Micro-Processor
Bus arbitration
Coprocessor
Status
Miscellaneous
Symbol for electrical ground
Φ
+5v
Symbol for clock signal
Power is 5volts

Mic 1


Memory control signals (rd, wr, fetch)
3
4
MAR
MDR
PC
MBR
SP
LV
CPP
TOS
OPC
H
4-to-16
Decoder
MPC
9
8
9
512 × 36-Bit
control store
for holding
the microprogram
JMPC
MIR
Addr
J
ALU
C
M
B
JAMN/JAMZ
High
bit
2
1-bit flip–flop
B bus
6
ALU
control
ALU
N
Z
Shifter
2
C bus
Control
signals
Enable
onto
B bus
Write
C bus
to register

3. The micro architecture level I

Mic 1

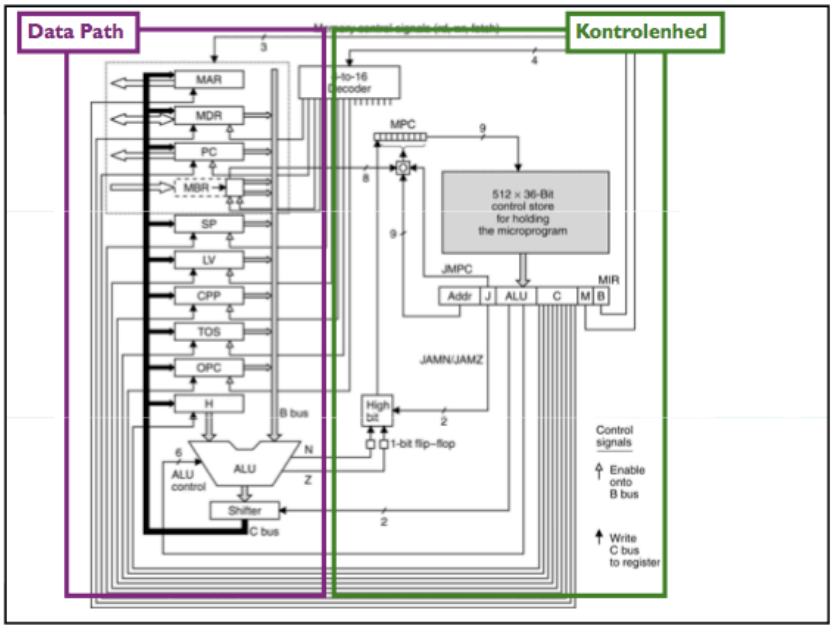


- **CPU registre**
- **Beregningsenhed**
- **Kontrolenhed**
- **FDEI (512 linier)**

FDEI = Der kommer en linie ind, som bliver valgt, og dernæst kommer ud til ALU igen.

Den kan deles i to dele, de første 256 og de næste 256. Den første, vil altid være 0, og dernæst 8 wildcards. Og i den anden vil den dog starte på 1 + otte wildcards.

Kan deles i to dele:



Datapath (Hvor lang tid det tager om at komme en tur rundt)

- Indeholder ALU'en med input og output.
- Holding (H) er A 2. Input til kontrolregistret

MAR = Læse eller tilskrivning af hukommelsen. Vi genererer og peger ind i hukommelsen.

Adresseringen på Word og Bit niveau. 2^{30} og hvis vi vil gøre det på fetch niveau er det 2^{32}.

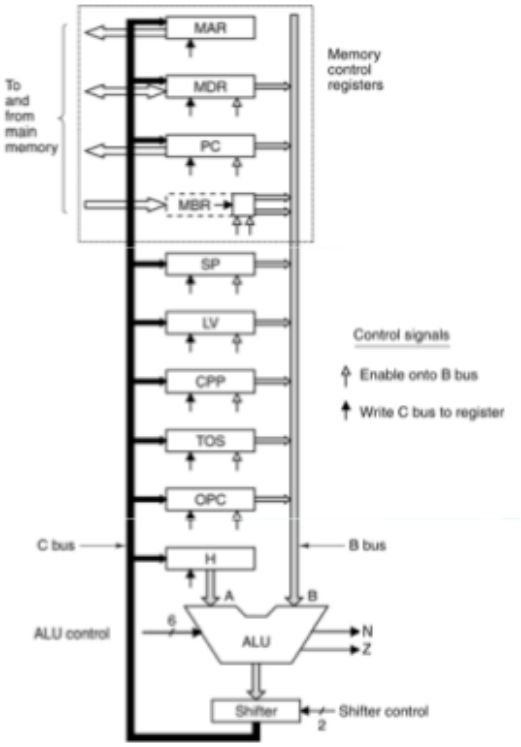
MDR = Hvis vi vil skrive til den, bliver det smidt.

MBR = Vi vil gerne putte to registreringer ud på B bussen. Hvis resultatet er negativt, er sign bittet er negativ. Og hvis den er positiv skal vi læse en usigned.

Hvis den bliver læst som et tal fra 0 til 256.

PC = Hvor er programmet kommet til

OPC= OpCode register. Aldrig bliver der hældt to registre ud på ALU. Der er kun 1 register som kan lægges ud af gangen på B.

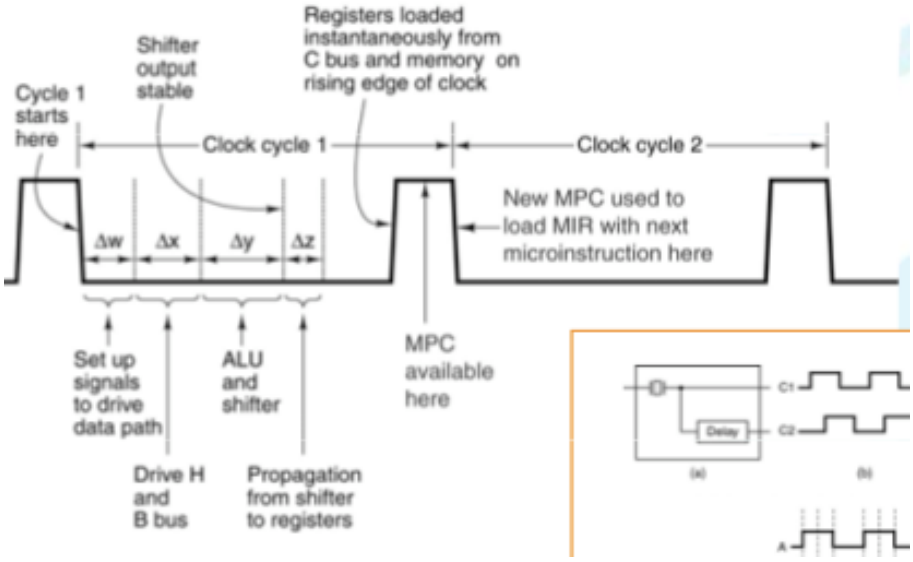


ALU = 6 kontrolsignaler, A og B, 6 ledninger, og et output. N(bit) og Z(bit) output. N er hvis resultatet er blevet negativ, og P hvis den er positiv. Den bruges til næste udregning. Hvis vi f.eks. har en loop. Den indeholder 32.bits ALU'er.

Most significant bit er på O31 = Den tjekker med N, og hvis den er negativ, udmunder den i negativ.

Under Funktion, kan man se hvad ALU'en kan gøre.

Data Path Timing



Asynkron clock – Første cycle, Hvor beregningen skifter fra ALU og shifter til registrene.

T 0

MPC peger på en bestemt adresse i 512 liniers kode.

T 0,Δw

Signalerne er kommet ud af microprogrammet, som sender videre. Det sætter signalerne op. Hvilke beregninger som skal udføres, og kontrolsignalerne til skifteren. Hvis beregningen er gået til negativ eller nul, hvor skal den så hen (JUMBIT sat) – Hvad er næste linie,

Adressefelt på 9 bit = Alle 512 fortæller om hvem som kommer efter sig.

T = Δ w; Δw+ Δx

Nu er signalerne klar på vores bus og står stabilt.

T= Δw +Δ x, ;Δw +Δx+ Δy

Nu beregnes der, og resultatet er kommet ud C bussen

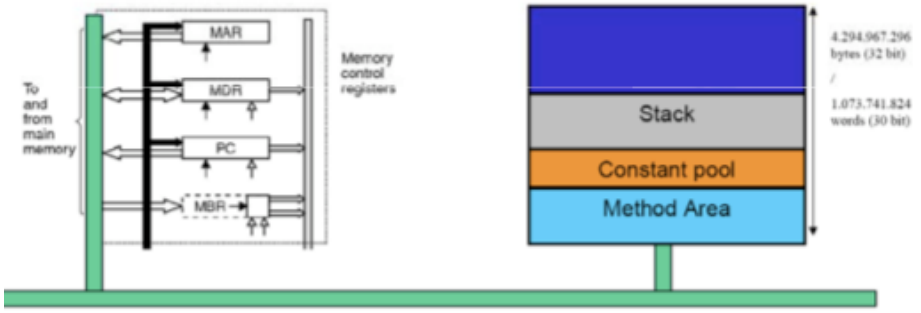
T= Δw + Δw, Δx, Δ y +Δz ;Δw+ Δx +Δy +Δz

Shifterne er registreres

T end of cycle

Når MPC er gået høj, er vi klar til at hente resultatet fra C bussen. På dette tidspunkt er den klar til at blive loadet ind. Der kommer derefter en ny værdi på MPC, og klar til næste beregning.

Lager tilgang



PC/MBR bruges til at tilgår method area ved brug af byte adresser.

MAR/MDR bruges til at tilgår stakken og constant pool ved brug af word adresser.

Timing: Hvis PC(MAR) sættes i cycle K indeholder MBR (MDR) indholdet af lagercellen i cycle K+2.

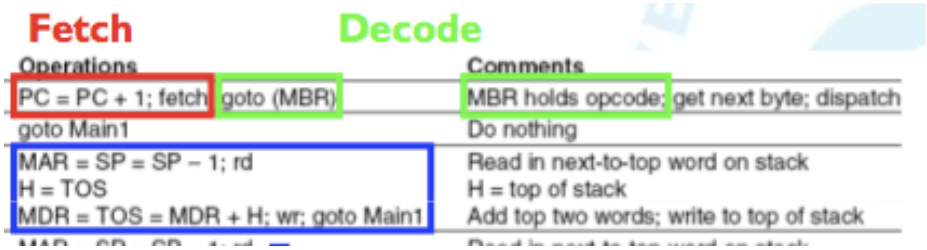
I **method area** har vi byte adressering, og i **Constant pool** og **Stack** arbejder vi med word adressering. Øverst har vi de antal bytes som er tilgængelig.

4. The micro architecture level II

Mic 1

Fetch-Decode-Execute

Eksempel på Assembly kode:



Det gælder om at reducere alle mikroinstruktioner per ISA instruktion. Vi skal derfor reducerer execution path. Det vil derfor gør er følgende:

Tidligere:

Label	Operations	Comments
pop1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
pop2		Wait for new TOS to be read from memory
pop3	TOS = MDR; goto Main1	Copy new word to TOS
Main1	PC = PC + 1; fetch; goto (MBR)	MBR holds opcode; get next byte; dispatch

Efter reducering:

Label	Operations	Comments
pop1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
Main1.pop	PC = PC + 1; fetch	MBR holds opcode; fetch next byte
pop3	TOS = MDR; goto (MBR)	Copy new word to TOS; dispatch on opcode

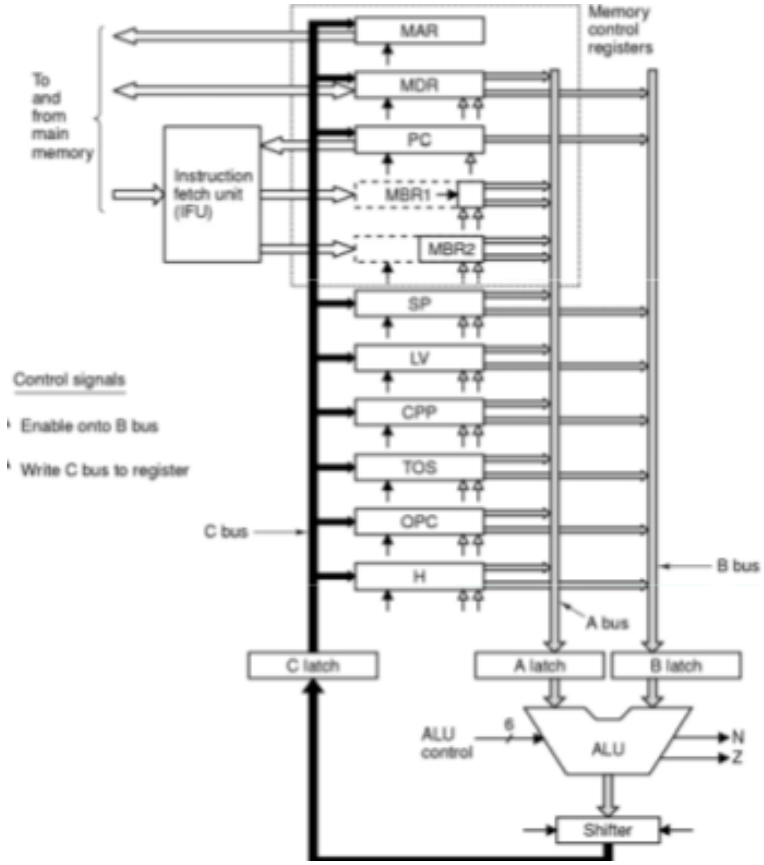
På samme tid, går man fra et 2-BUS design til et 3-BUS design. Man kan se slide 7.

Mic 2

Nye tiltag i MIC2, er et 3-Bus design + IFU. Nu bliver fetchen varetaget af IFU, samt PC tælles op i IFU, som dermed frigiver ALU cyklusser. A bussen eliminerer mange loads til H registeret. Der er dog brug for en mere kontrolenhed i mic2.

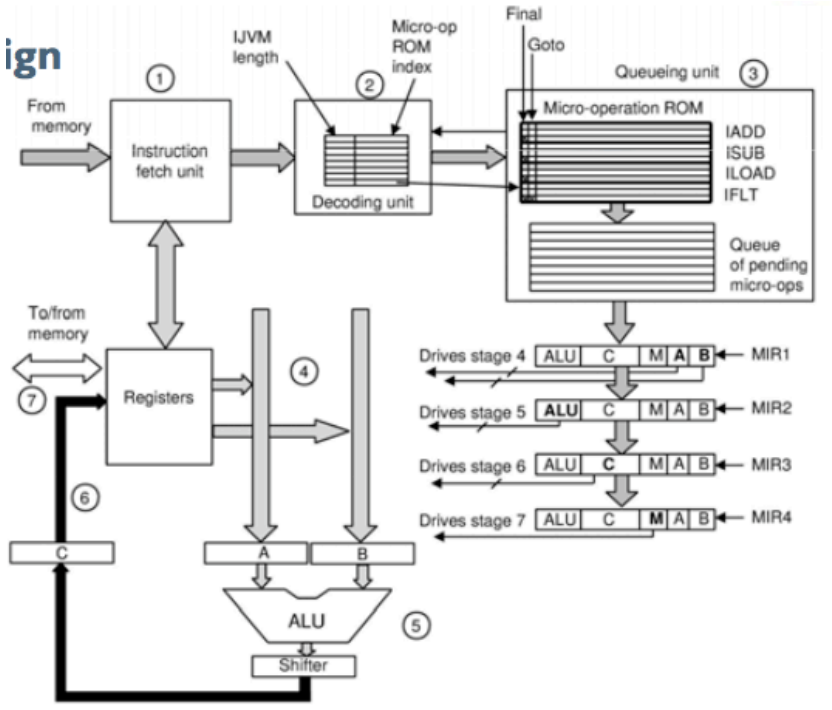
Mic 3

I Mic 3, er der igen et 3-BUS design med en IFU, og en register latches.

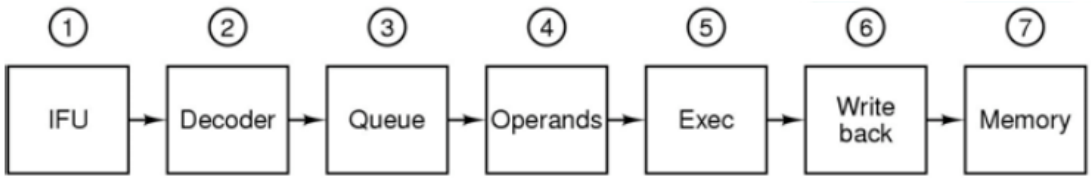


Mic 4

I mic 4, er der et multi-bus design, med en tilhørende IFU. Der er også en decoding enhed, som en Queueining enhed og multiple MIRs og igen Register latches.



Fra MIC 1 til MIC 4



True Dependence (RAW – Read After Write)
Når en mikroinstruktion vil læse et register, der endnu ikke er skrevet til.

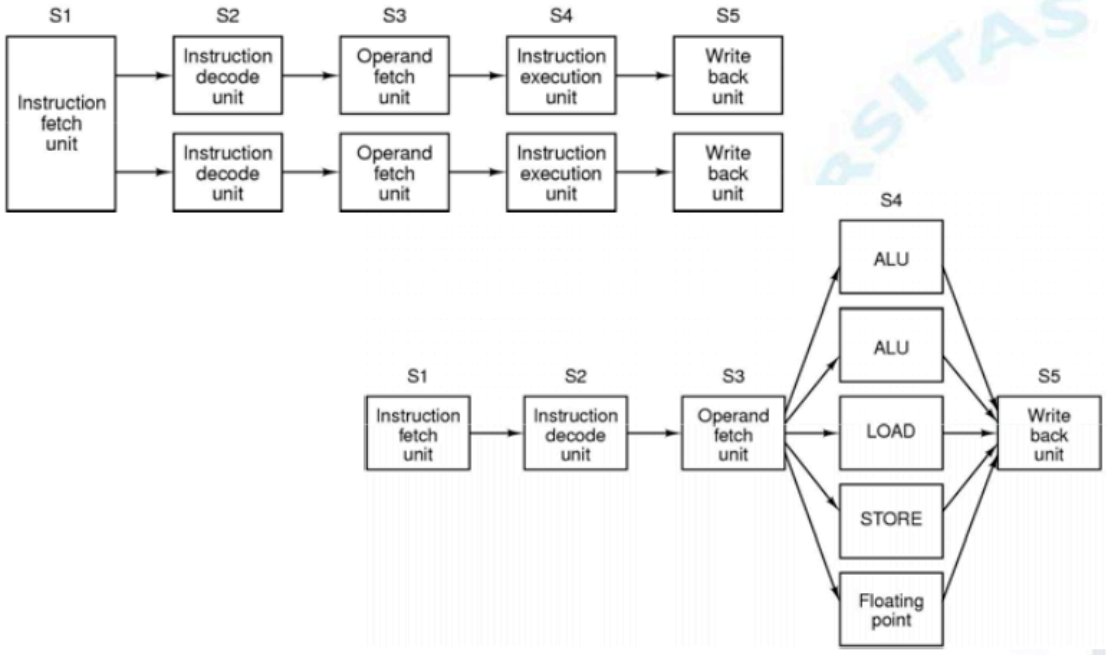
Anti Dependence (WAR – Write After Read)
Når en mikroinstruktion skriver til et register, der ikke er blevet læst.

Output Dependence (WAW – Write After Write)
Når to mikroinstruktioner vil skrive til samme register .

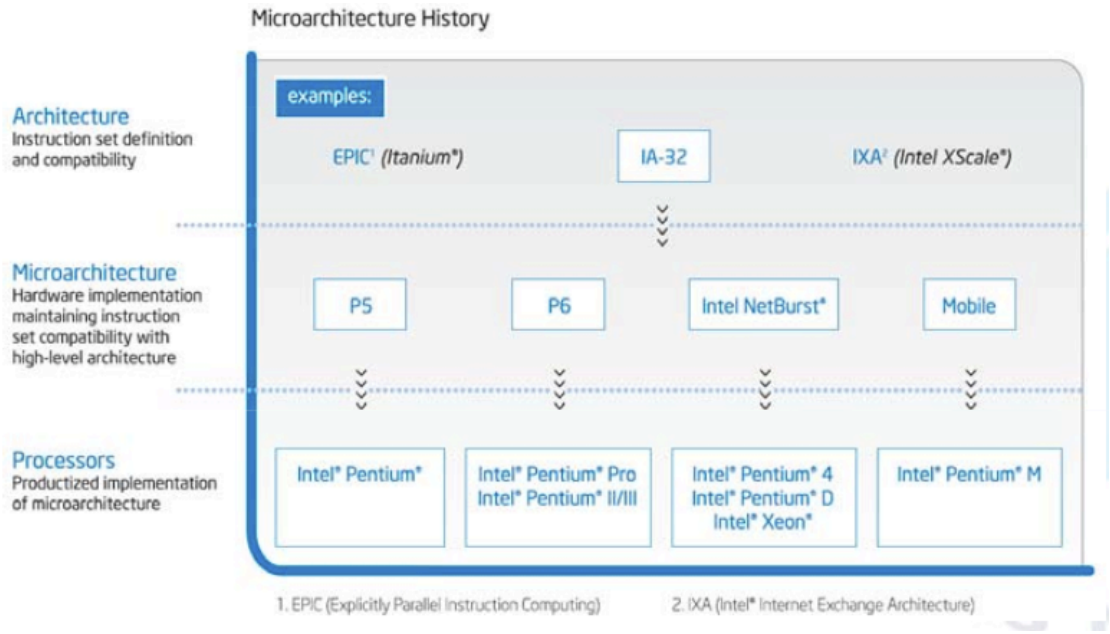
Structual Hazards
Ressource skal være tilgængelig for to eller flere instruktioner på samme tid

Branch hazards (eller Control Hazards)
Problemer med at holde IFU opdateret ved branches

Dual pipeline & superscalar processor

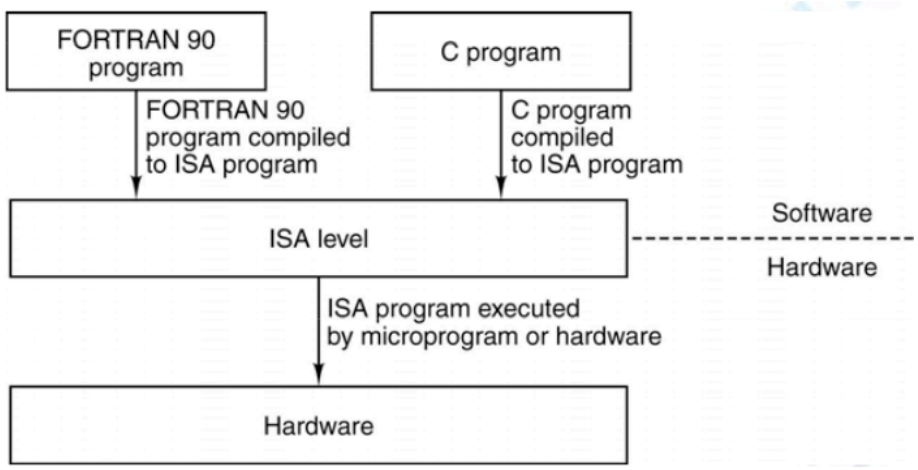


ISA vs. Mikro arkitektur vs. Processor



5. The ISA level

ISA laget



Et godt ISA

Programmability:

Et godt ISA er nemt at generere kode til

Et godt ISA er nemt at generere kode til

2. Implementability:

Et godt ISA kan implementeres simpelt og effektivt i hardware nu og i fremtiden (generelle løsninger er bedre end specielle løsninger).

Skal kunne understøtte forskellige design kriterier (low-power, high-reliability , low-cost).

3. Compatibility:

Nye processorer SKAL understøtte gammelt software

Registre

Special purpose registre

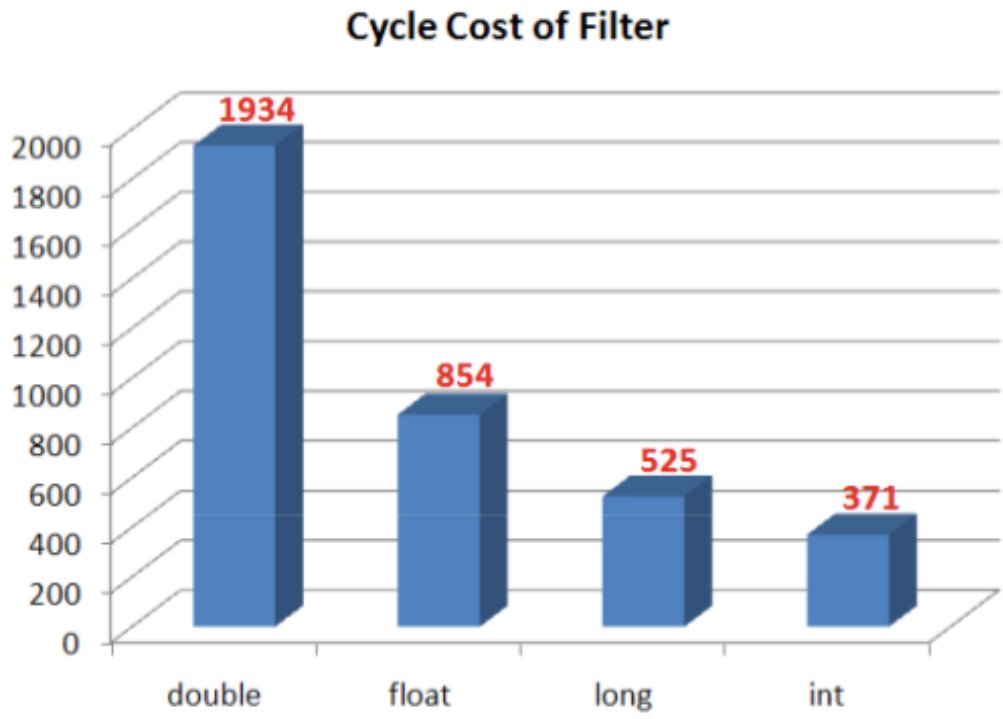
- PC (Program Counter)
- SP (Stack Pointer)

General purpose registre

- OPC (Opcode Register)
- Indeholder midlertidige resultater og hyppigt anvendte lokale variabler.
- RISC-maskiner har ≥ 32 registre af denne type

Registre er hurtigere end memory, så det gælder om at have så mange som muligt? (tradeoffmellem pris og antal)

Filter performance



Big vs. Little Endian

File	Endianness
BMP	Little Endian
GIF	Little Endian
JPEG	Big Endian
TIFF	Both Endian identifier encoded into file

,

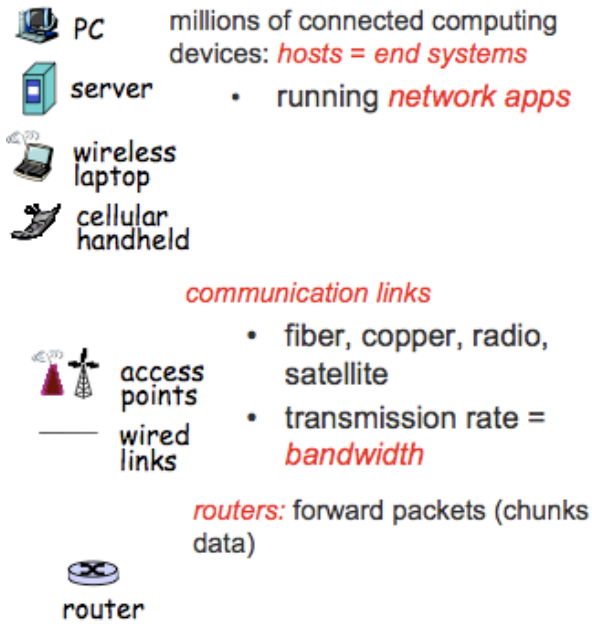
Instruktioner

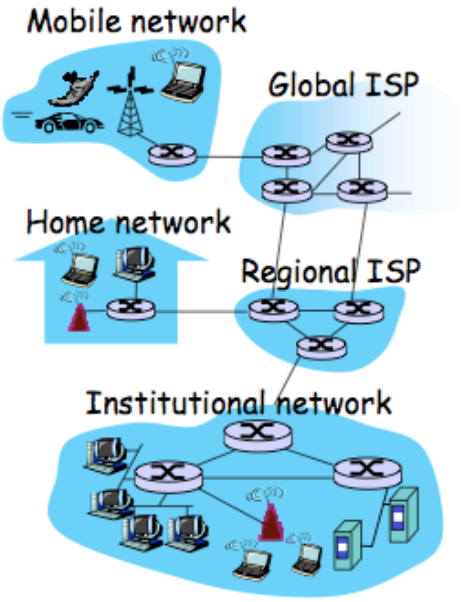
Set af maskin instruktioner

–Memory: LOAD, STORE, MOVE
–Arithmetic: ADD, SUB etc.
–Boolean: AND, OR, XOR
–Comparisons: CMP
–Branching: GOTO, BLE, BEQ
–Method invocation: INVOKE_VIRTUAL

6. Network architecture

Internet





Protocols is control sending, receiving of messages

- TCP, IP, http, Skype, Ethernet

Internet - Networks of networks

- looselyhierarchical
- public internet versus private internet

Internet standards

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

Forkortelser

ISP: Internet service provider

Host a service Clint: A clint program is a program running on one end system that request and receives from a server program running one end system.

TCP: Transmission Control Protocol

IP: Internet Protocol

API: Application Programming Interface

DSL: Digital Subscriber Line

HFC: Hybrid Fiber-Coaxial cable

Protocol

Human protocols

- Whats is the time
- You ask for a question (ex. In lecture)

Network protocols

- Sending messageswithcommunication of machines.
- More than 1 communicatingentry.

Network edge

- End systtems
- Client/server model
  - WhenyousearchonGoogle, youare the client, and Google the server.
- Peer-peer model
  - Youcanbe a server, and a client, and it changeswhileyouareon the network
  - Ex. Skype, EmuleorBittorrent

Point 2 Point Access

- Dialup via modem
- DSL – Digital Subsriber Line
  - Deployed by a telephonecompany
  - Up to 1 Mbpsupstream, 8 Mbpsdownstream
  - Dedicadedphysical line to telephone central office
- HFC – Hybrid fiber-coaxialcable
  - Asymmetric – Up to 2 Mbpsupstream, 30 Mbpsdownstream
- Netowrk – Cable and fiber attaches homes to ISP Router

Physcial media

Twisted pair (TP)

- Category 3: Traditionalphone wires – 10 Mbps Ethernet
- Category 5: 100 Mbps Ethernet

Coaxialcable

- Single chaneloncable
- Legacy Ethernet
- HFC

Fiber opticcable

- 10's – 100's Gps

Guided media: solid medium, such as fiber-optic, twisted-pair and coaxial.

Unguided media: wireless LAN and digital satellite

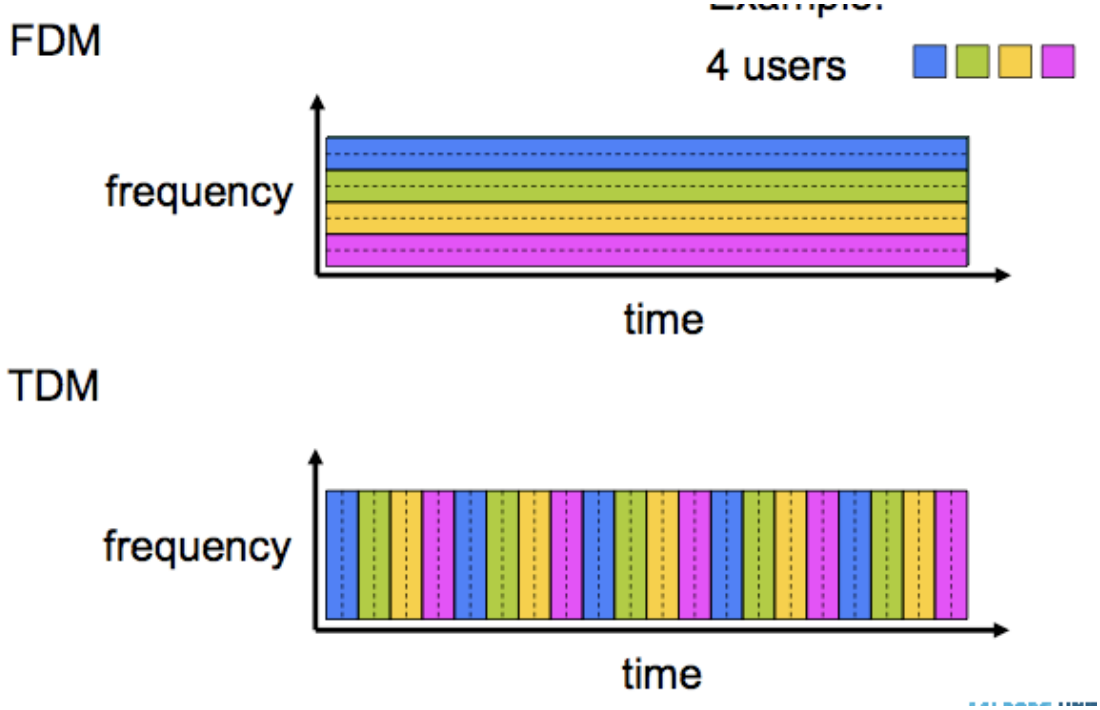
Network core

Circuitswitching

- Circuit per call – Link bandwitdth – Switch capacitydedicatedresurces – (nosharing)

Packetswitching

- Data sent throughnetwork – dividedintopieces–
  - Frequency division (FM radio)
  - Time division (time transfer packeging)



Packetswitching vs. circuitswitching

1 Mb/s link
each user:

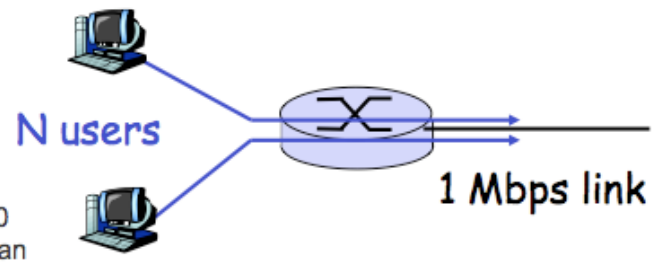
- 100 kb/s when "active"
- active 10% of time

circuit-switching:

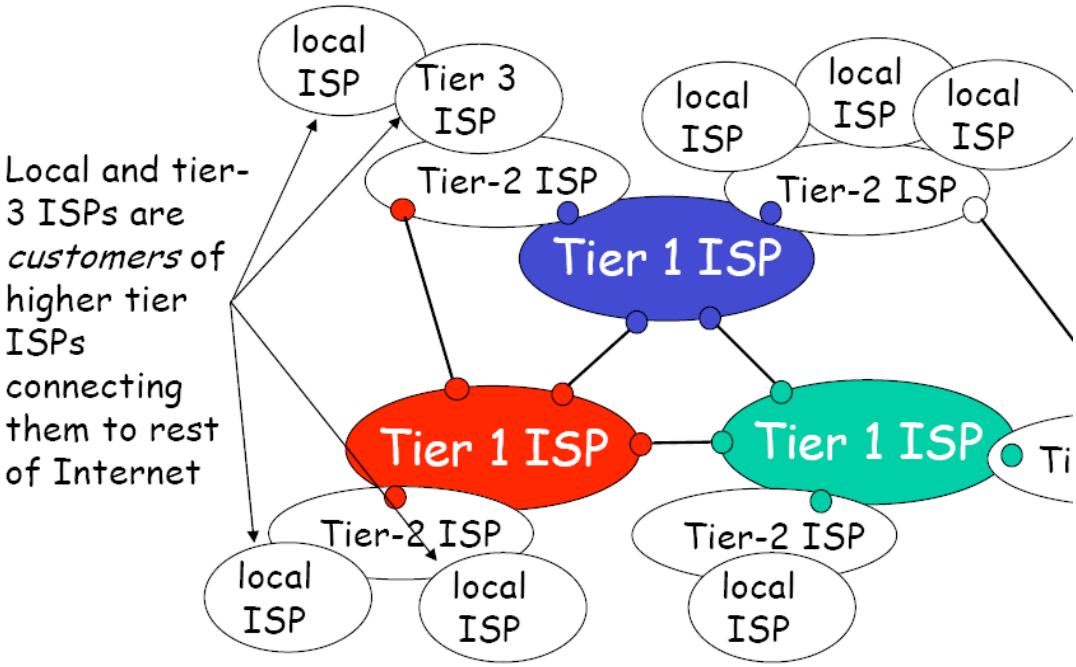
- 10 users

packet switching:

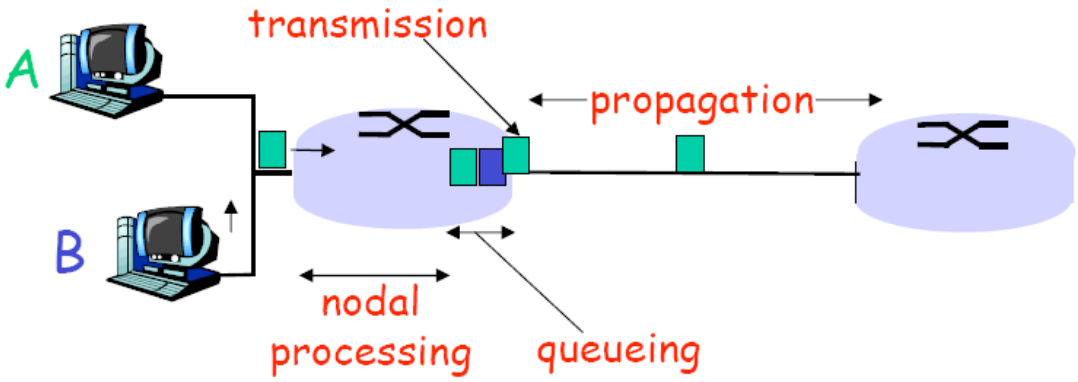
- with 35 users, probability > 10 active at same time is less than .0004



Network of Networks



Network delay



Nodal delay

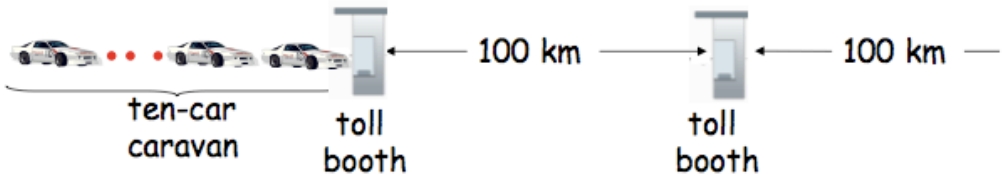
$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

d_{proc} = processing delay
- typically a few microsecs or less

d_{queue} = queuing delay
- depends on congestion

d_{trans} = transmission delay
- = L/R, significant for low-speed links

d_{prop} = propagation delay
- a few microsecs to hundreds of msecs



cars "propagate" at 100 km/hr
toll booth takes 12 sec to service car (transmission time)
car~bit; caravan ~ packet
Q: How long until caravan is lined up before 2nd toll booth?

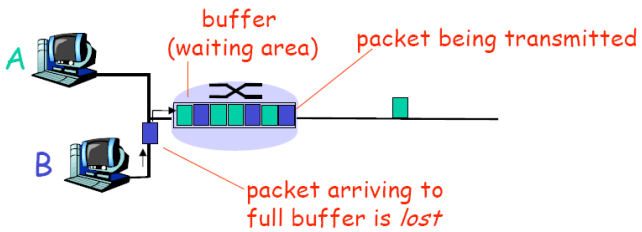
Time to "push" entire caravan through toll booth onto highway = 12*10 = 120 sec
Time for last car to propagate from 1st to 2nd toll both: 100km/(100km/hr)= 1 hr
A: 62 minutes

Packet loss

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

packet arriving to full queue dropped (aka lost)

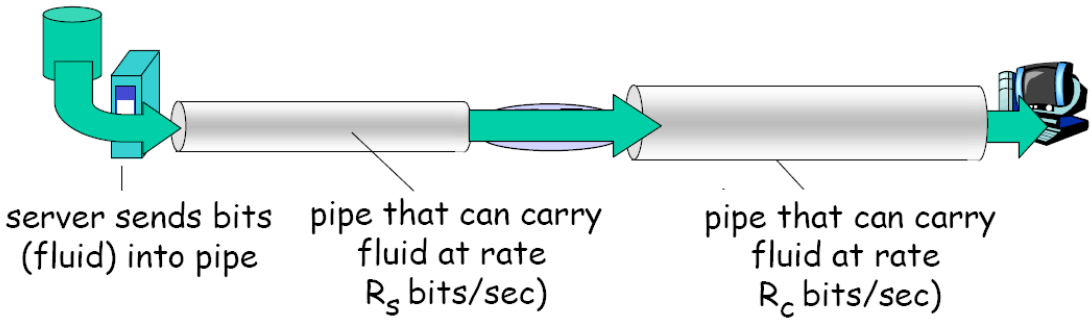
lost packet may be retransmitted by previous node, by source end system, or not at all



Throughput

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

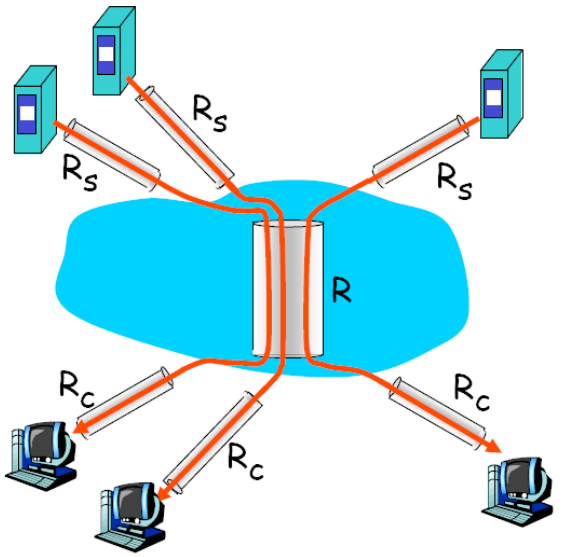
- *instantaneous:* rate at given point in time
- *average:* rate over longer period of time



Throughput: Internet scenario

per-connection end-end throughput: $\min(R_c, R_s, R/10)$

in practice: R_c or R_s is often bottleneck



10 connections (fairly) share backbone bottleneck link R bits/sec

Layer Protocols

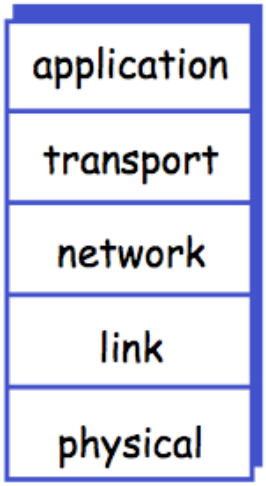
application: supporting network applications
- FTP, SMTP, HTTP

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
- PPP, Ethernet

physical: bits "on the wire"



7. Application layer I

Some network applications

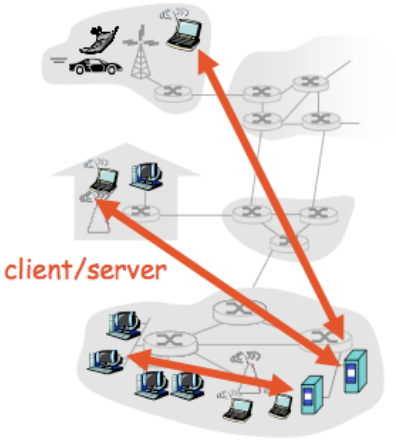
- E-mail
- Web
- Instantmessaging
- Remotelogin
- P2P file sharing
- VOIP

Creating a networkapplication

- *Write programs run on different end systems*
- *Noneed to write software for networkcoredevices.*

Priciples of network applications

Client-server



server:
- always-on host
- permanent IP address
- server farms for scaling

clients:
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Peer-to-peer (P2P)

no always-on server
arbitrary end systems directly communicate
peers are intermittently connected and change IP addresses

Highly scalable but difficult to manage

Hybrid of client-server and P2P

Skype
- voice-over-IP P2P application
- centralized server: finding address of remote party:
- client-client connection: direct (not through server)

Instant messaging
- chatting between two users is P2P
- centralized service: client presence detection/location
  - user registers its IP address with central server when it comes online
  - user contacts central server to find IP addresses of buddies

Processes Communicating

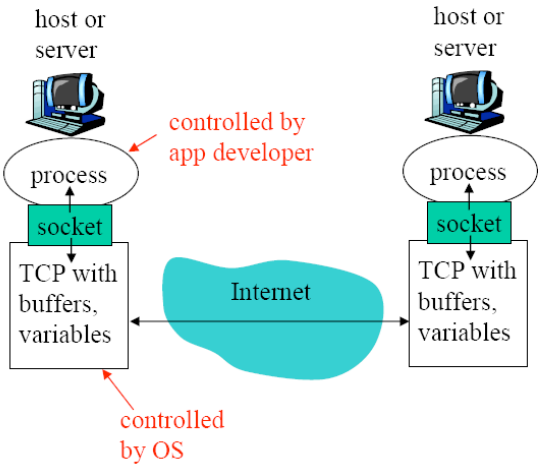
Process: program running within a host.
within same host, two processes communicate using inter-process communication (defined by OS).
processes in different hosts communicate by exchanging messages

Client process: process that initiates communication
Server process: process that waits to be contacted

Note: applications with P2P architectures have client processes & server processes

Sockets

process sends/receives messages to/from its socket
socket analogous to door
- sending process shoves message out door
- sending process relies on transport infrastructure on other side of door which brings message to socket at receiving process



What transport service does an application need?

Data loss
some applications (e.g., audio) can tolerate some loss
other applications (e.g., file transfer, telnet) require 100% reliable data transfer

Throughput
some applications (e.g., multimedia) require minimum amount of throughput to be "effective"
other apps ("elastic apps") make use of whatever throughput they get

Security
Encryption, data integrity, …

Timing
some applications (e.g., Internet telephony, interactive games) require low delay to be "effective"

Transport service requirements of common applications

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	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

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 (eg Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

TCP vs. UDP (Protocol for applications)

TCP service:
connection-oriented: setup required between client and server processes
reliable transport between sending and receiving process
flow control: sender won't overwhelm receiver
congestion control: throttle sender when network overloaded
does not provide: timing, minimum throughput guarantees, security

UDP service:
unreliable data transfer between sending and receiving process
does not provide: connection setup, reliability, flow control, congestion control, timing, throughput guarantee, or security

- TCP Service(Transmission Control Protocol)
  - Connection-oriented service
    - Handshaking (onslaught of packets)
    - Full-duplex
  - Reliable data transfer service
    - Rely on TCP to deliver all data sent without error and I proper order.
  - Real-time applications often run on over UDP rather than TCP. > TCP congestion control attempts to limit each TCP connection to its fair share of network bandwidth.

- UDP Services (User Datagram Protocol)
  - Connectionless
    - No handshaking
  - Unreliable data transfere
    - No guarantee that the message will ever reach the receiving process and they may be out of order
  - No congestion control.
  - Often block by firewalls.

- Addressing Processes
  - IP address: 32-bit quantity. A uniquely number identifying the host.
  - Port number: Specific port number. (Web servers port 80, SMTP (mail-servers) port 25)

identifier includes both IP address and port numbers associated with process on host.
Example port numbers:
- HTTP server: 80
- Mail server: 25

to send HTTP message to **fyr.cs.aau.dk** web server:
- IP address: **130.225.194.112**
- Port number: 80

Web and HTTP

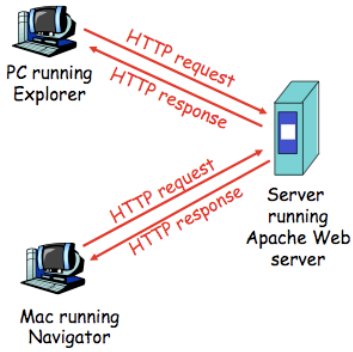
Web and HTTP

First some terminology
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
Example URL:

www.cs.aau.dk/dna/slide7.pdf

host name | path name

Http Overview



- Client (browser that requests, receives, displays web objects).
- Server (web server sends objects in response to requests)

- http uses TCP (TCP er den underliggende transport protokol som http benytter)
- http is "stateless"(http servere gemmer ikke oplysninger om klienten)

RTT (round trip tiden)

Nonpersistent http (At most one object is sent over TCP)

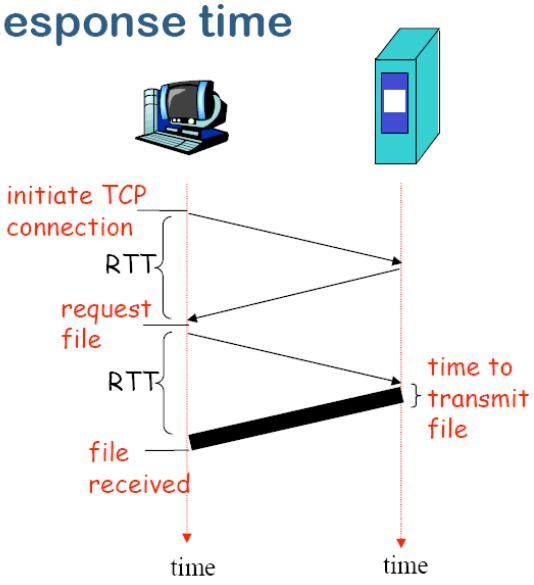
1. http clientinitiates TCP
2. http server at host
3. http clientsends http
4. http server receives
5. http server closes TCP
6. http clientrecievesresponse
7. Again for more objects

Non-Persistent HTTP: Response time

Definition of RTT: time for a small packet to travel from client to server and back.

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
total = 2RTT+transmit time



Persistent http (Multiple objects is sent over TCP)

1. Only does the steps one time.
2. Med og uden Pipelining

Persistent HTTP

Nonpersistent 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

HTTP request message

two types of HTTP messages: *request*, *response*
HTTP request message:

- ASCII (human-readable format)

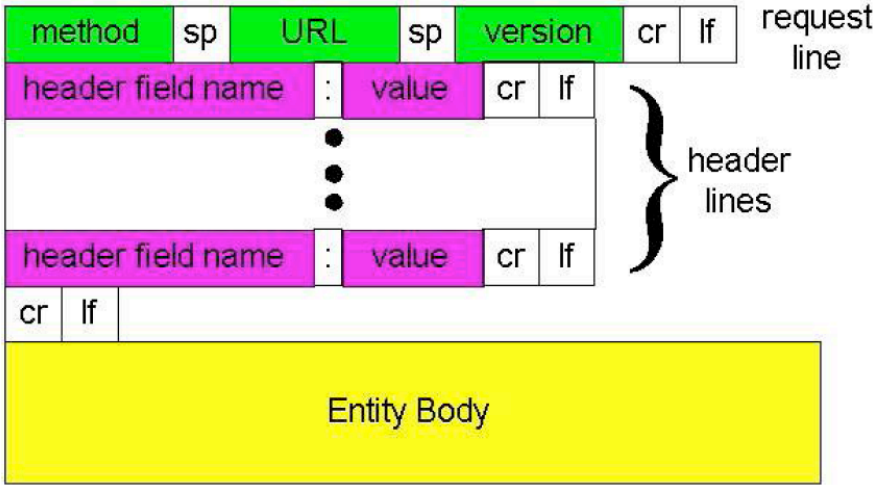
request line (GET, POST, HEAD commands)

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

header lines

Carriage return, line feed indicates end of message

(extra carriage return, line feed)



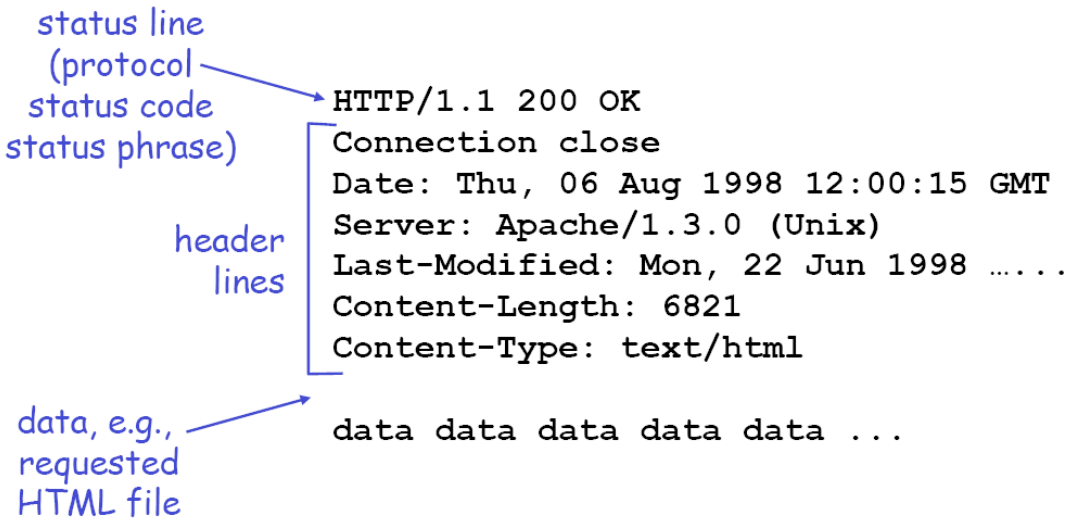
Post method:
Web page often includes form input
Input is uploaded to server in entity body

URL method:
Uses GET method
Input is uploaded in URL field of request line:

HTTP Methods

DELETE requests that the server delete the information referenced.
GET retrieves the information that it references.
HEAD is similar to GET, except that the response includes only its header, not the information referenced.
POST indicates that data is being transferred to the server to be used with the information referenced.
PUT requests that the server store the information accompanying it in the location given.

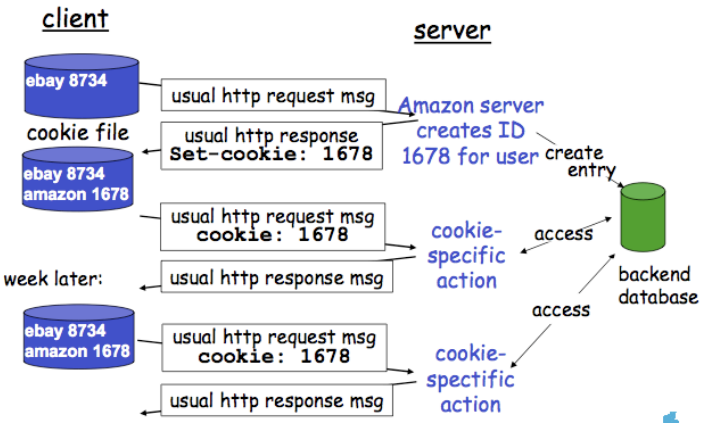
HTTP response message



Cookies

- header line of http responsemessage
- header line in http requestmessage
- file keptonuser's host managed by user's browser
- back-end database at website

Example



Cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state (web e-mail)

Cookies and privacy, permit sites to learn a lotaboutyou. Youmaysupplyname and email to sites.

Web caching

Cache acts as both client and servers. Normally cache is already installed by ISP providers.

- Reduce response time for client requests.
- Reduce traffic on an institution's access link
- Internet dense with caches: enables "poo" content
  Providers to effectively deliver content (but so does P2P file sharing)

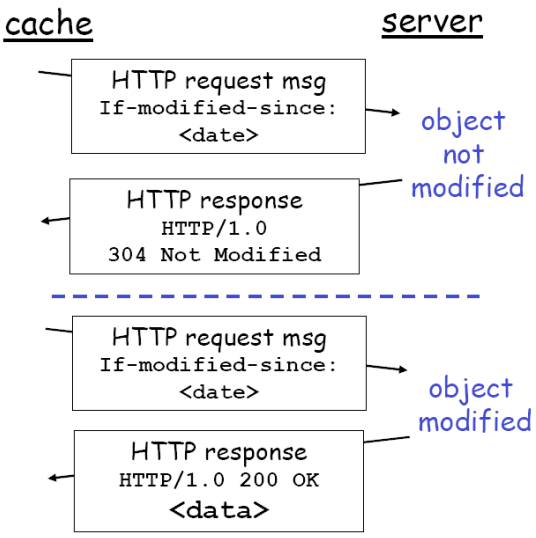
Conditional GET

Goal: don't send object if cache has up-to-date cached version
cache: specify date of cached copy in HTTP request

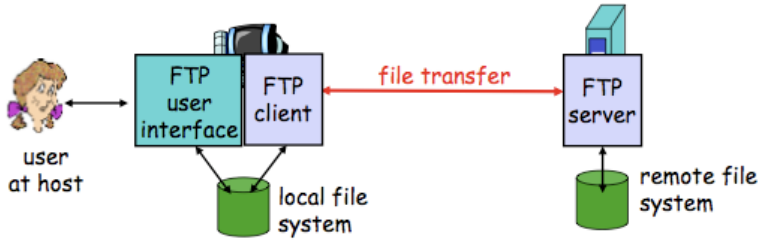
```
If-modified-since:
  <date>
```

server: response contains no object if cached copy is up-to-date:

```
HTTP/1.0 304 Not
  Modified
```



FTP



Transfer file to or from remote hosts. Client/server model.

- Client / side that initiates transfer
- Server / remote host
- ftp RFC 959
- ftp server : port 21

TCP control connection will always be port 21 and data connection will always be on port 20.

Er modsat HTTP ikke stateless, her er det nødvendigt at holde styr på brugerens status.

FTP: separate control, data connections

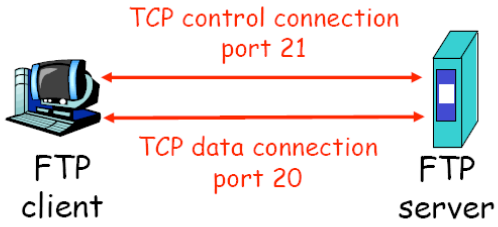
FTP client contacts FTP server at port 21, TCP is the transport protocol

client authorized over control connection

client browses remote directory by sending commands over control connection.

when server receives file transfer command, server opens 2nd TCP connection (for file) to client

after transferring one file, server closes data connection.



server opens another TCP data connection to transfer another file.

control connection: "out of band"

FTP server maintains "state": current directory, earlier authentication

Commands and return codes

Sample commands:
sent as ASCII text over control channel
USER *username*
PASS *password*
LIST return list of file in current directory
RETR **filename** retrieves (gets) file
STOR **filename** stores (puts) file onto remote host

Sample return codes
status code and phrase (as in HTTP)

```
331 Username OK, password required
125 data connection already open;
transfer starting
425 Can't open data connection
452 Error writing file
```

Electronic Mail

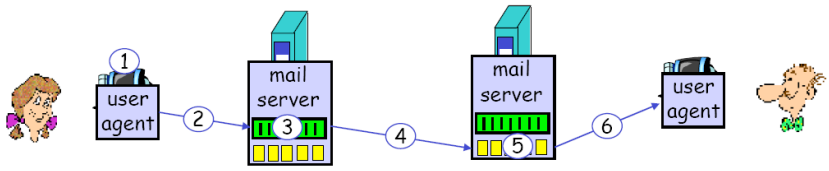
3 major components

- user agents (mail reader)
  - composing, editing, reading mail messages (apple mail, outlook, mozilla etc.)
- mail servers
  - contains incoming, messages for user.
- simple mail transfer protocol SMTP
  - client: sending mail server
  - Server: receiving mail server

It uses TCP, from client to server with port 25.

Example of sceneario

1) Alice uses UA to compose message and "to" bob@someschool.dk
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



Sample SMTP interaction

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

Final word

SMTP usespersistentconnections

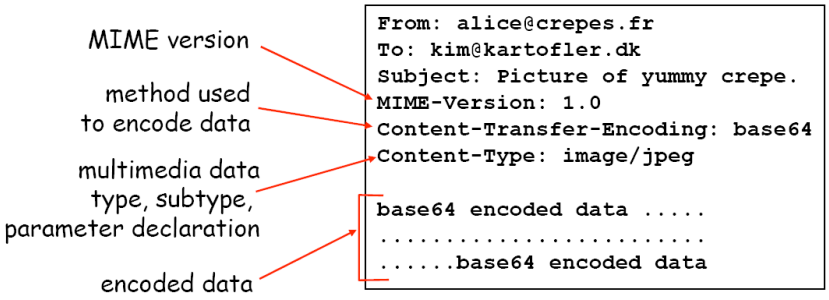
SMTP requiresmessage (header and body) to be in **7-bit ASCII**

SMTP server uses CRLF.CRLF to determine end of messages.

Comparison with HTTP:
HTTP: pull
SMTP: push
both have ASCII command/response interaction, status codes
HTTP: each object encapsulated in its own response msg
SMTP: multiple objects sent in multipart msg

Message format: multimedia extensions

MIME: multimedia mail extension, RFC 2045, 2056
additional lines in msg header declare MIME content type



Mail Access protocol

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`

```
S: +OK POP3 server ready
C: user kim
S: +OK
C: pass hungry
S: +OK user successfully logged on

C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

- POP (Post Office Protocol)
  - authorization (agent <-->server) and download
- IMAP (Internet Mail Access Protocol)
  - More features
  - Manipulation of stored messages on server.
- http
  - gmail, hotmail.com, Yahoo Mail.

8. Application layer II

DNS

DNS rolle er en ganske anden end WWW, filoverførsel og e-mail-applikationer. I modsætning er disse applikationer er DNS ikke en applikation, som er brugerne bruger interaktivt. I stedet er kernefunktionen at oversætte værtsnavnet til de underliggende IP-adresser til brug for brugerapplikationer og andet software på internettet.

A singel centralized database simply doesn't scale.

Domain Name System

People: Many identifiers (CPR #, name, passport #)

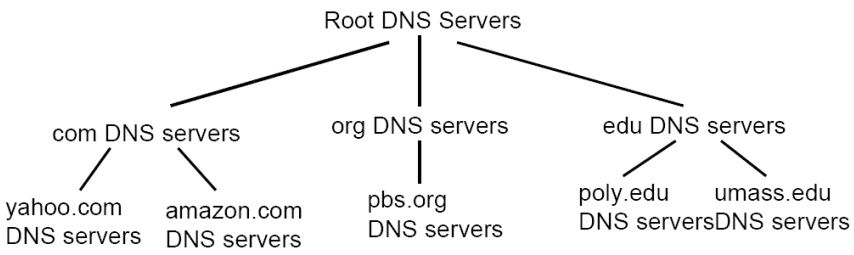
Internet host, routers: IP adress (32bit) – "name" e.g. www.inwire.dk (used by humans) , and IP-adresses. (used by hosts, Computers etc.)

DNS Services

- Hostname to IP addres translation (aau.dk has an IP adress)
- Host aliasing (Auc.dk to aau.dk)
- Mail server aliasing
- Load distribution (reducing the network traffic)

We **can not centralize** DNS because, the single point of failure, traffic volume, distant centralized database, and the maintenance is difficult. It will never ever be able to scale.

Distributed, Hierarchical Database



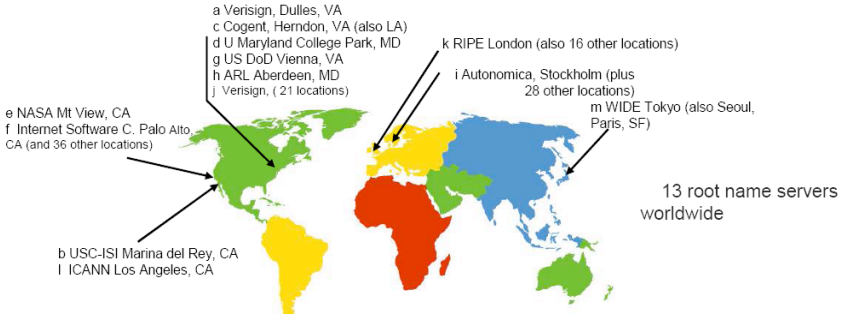
Client wants IP for www.amazon.com; 1st approx:
client queries a root server to find com DNS server
client queries com DNS server to get amazon.com DNS server
client queries amazon.com DNS server to get IP address for www.amazon.com

The root server, has 3 DNS servers, like .com , .org, and .edu.

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



We have 13 Root name servers Worldwide to day. Some of them is also located other places in the world, like the one in Stockholm, which is 28 other locations.

Top-level domain (TLD) servers:

- responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
- Network Solutions maintains servers for com TLD
- Educause for edu TLD

Authoritative DNS servers:

- organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web, mail).
- can be maintained by organization or service provider

Local Name Server

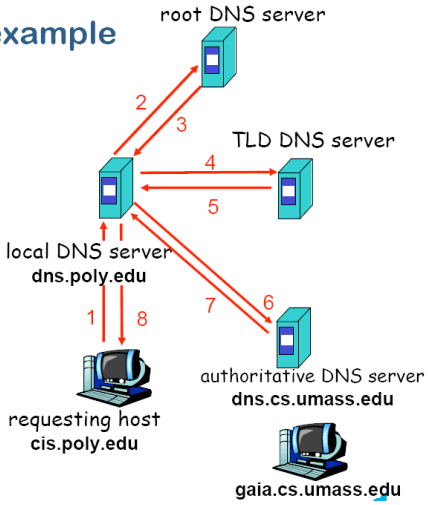
- Not belong to the hierarchy.
- Each ISP has one – default-name server
- When host makes DNS query, it is sent to the local DNS server
    - Proxy, forwards query into hierarchy.

Examples on DNS name resoulution

DNS name resolution example

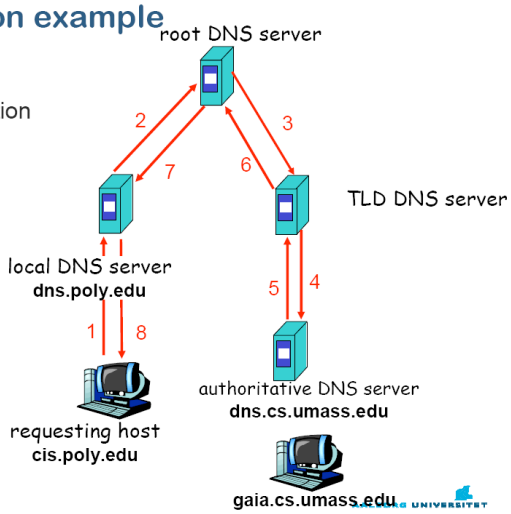
Host at cis.poly.edu wants IP address for gaia.cs.umass.edu

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



DNS name resolution example

recursive query:
puts burden of name resolution on contacted name server
heavy load?



DNS: caching and updating records

- Once name server learns mapping, it caches mapping
  - Entries timeout –(disappear) after some time.
  - TLD servers typically cached in LNS
  - Root name servers not often visited –
- Update/Notify mechanism - Design – RFC 2136

DNS Records

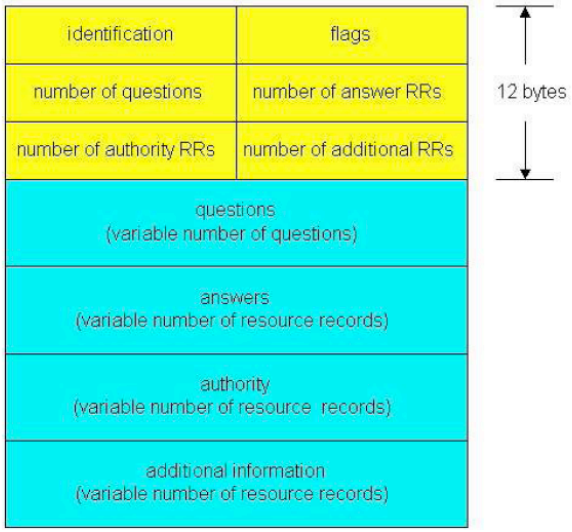
RR format: (name, value, type, ttl)

- Type A
  - **Name** is hostname
  - **Value** is IP Address
- Type=NS
  - **Name** is domain (inwire.dk)
  - **Value** is hostname of authoritative name server
- Type=CNAME
  - **Name** is alias name for canonical name
  - **Value** is canonical name
- Type=MX
  - **Value** is name of mailserver associated with **name.**

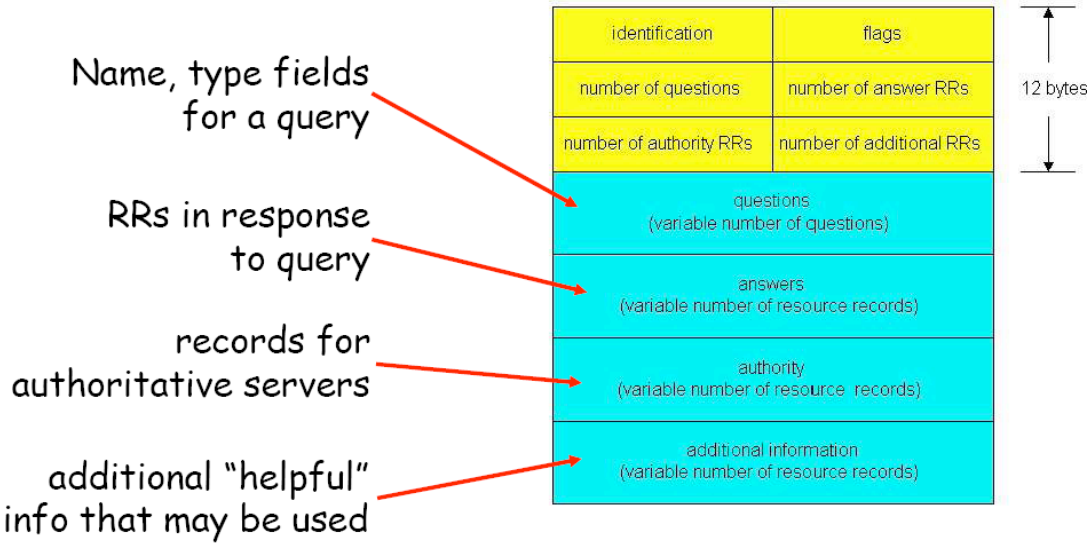
DNS protocol, messages

DNS protocol : *query* and *reply* messages, both with same *message format*

msg header
➢identification: 16 bit # for query, reply to query uses same #
➢flags:
- query or reply
- recursion desired
- recursion available
- reply is authoritative



DNS protocol, messages



Inserting records into DNS

Example: new startup "Network Utopia"
Register name networkuptopia.com at DNS registrar (e.g., Network Solutions)

- provide names, IP addresses of authoritative name server (primary and secondary)
- registrar inserts two RRs into com TLD server:
  (networkutopia.com, dns1.networkutopia.com, NS)
  (dns1.networkutopia.com, 212.212.212.1, A)

Create authoritative server Type A record for www.networkuptopia.com; Type MX record for networkutopia.com

How do people get IP address of your Web site?

They get IP address, because each host have this types RR from the network. You can't have an IP address, which isn't registered.

P2P Applications

- No always-on server
- Arbitrary end systems directly
- Communiation peers are intermittently connected and change IP-adress

- ***File distribution***
- ***Searching for information***
- ***Case study : Skype***

File Distribution (Server-client vs. P2P)

Time: Server-client

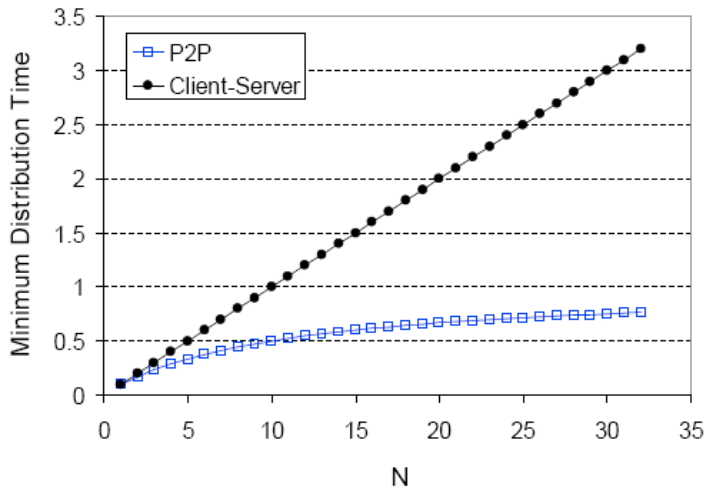
Time to distribute F to N clients using client/server approach $= d_{cs} = \max\{NF/u_s, F/\min_i(d_i)\}$

Time: P2P

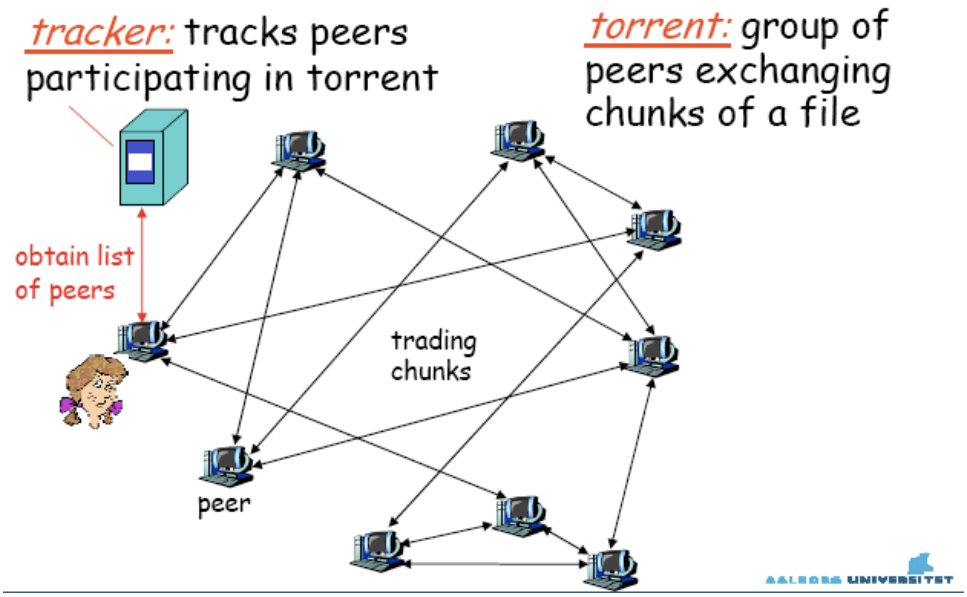
$$d_{P2P} = \max\{F/u_s, F/\min_i(d_i), NF/(u_s + \sum u_i)\}$$

Once it has sent one file out, the neighbours can sent the file through. It then means that the time is smaller than the server-client.

Client upload rate = u, F/u=1 hour, u_2=10u, $d_{min} >= u_s$



BitTorrent



A file is divided into 256kb (chunks) peer joining torrent:

- Has no chunks, but will accumulate them over time.
- Registers with tracker to get list of peers, connects to subset of peers (neighbors')

While downloading, peer uploads chunks to other peers

Peers may come and go

Pulling chunks

- any time different peers have different subsets of file chunks
- periodically a peer ask each neighbor for list of their chunks.
- The peer is sending requests for her missing chunks (Sender forespørgsel efter filer I en store fil).

Sending chunks: tit-for-tat

- Re-evaluate top 4 every 10 secs
- Newly chosen peer may join top 4
- "optimistically unchoke"

Searching for information

File sharing (eg e-mule)
- Index dynamically tracks the locations of files that peers share.
- Peers need to tell index what they have.
- Peers search index to determine where files can be found

Instant messaging
- Index maps user names to locations.
- When user starts IM application, it needs to inform index of its location
- Peers search index to determine IP address of user.

Centralized index

1. When peer connects, it informs central server
    a. IP address
    b. Content
2. Alice queries for "Hey Jude"
3. Alice request file from Bob.

Problems

- Single point of failure
- Performance bottleneck
- Copyright infringement: "target" of lawsuit is obvious.

Query Flooding

Here it is fully distributed with no central server

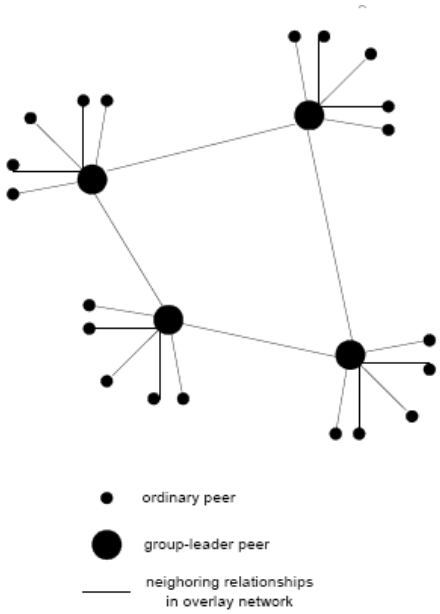
Gnutella uses it, and each peer indexes the files it makes available for sharing.

Problems

- Edge between peer X and Y, if there is a TCP connection
- All active peers and edges form overlay net.
- Virtual (not physical) link
- Given peer typically connected with <10 overlay neighbors.

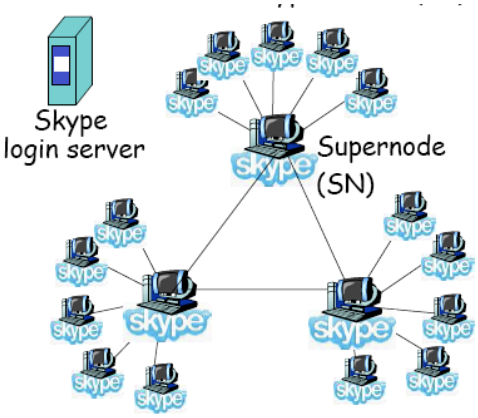
Hierarchical Overlay

- Between centralized index, query flooding approaces.
- Each peer is either a super node or assigned to a upper node
- Super node tracks content in its children.

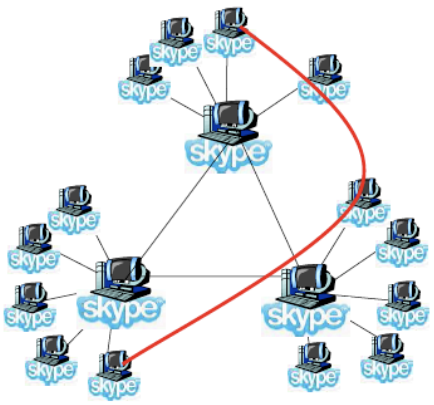


Case Study : Skype

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SNs
- Index maps usernames to IP addresses; distributed over SNs



- Problem when both Alice and Bob are behind "NATs".
  - NAT prevents an outside peer from initiating a call to insider peer
- Solution:
  - Using Alice's and Bob's SNs, Relay is chosen
  - Each peer initiates session with relay.
  - Peers can now communicate through NATs via relay



Socket Programming (TCP)

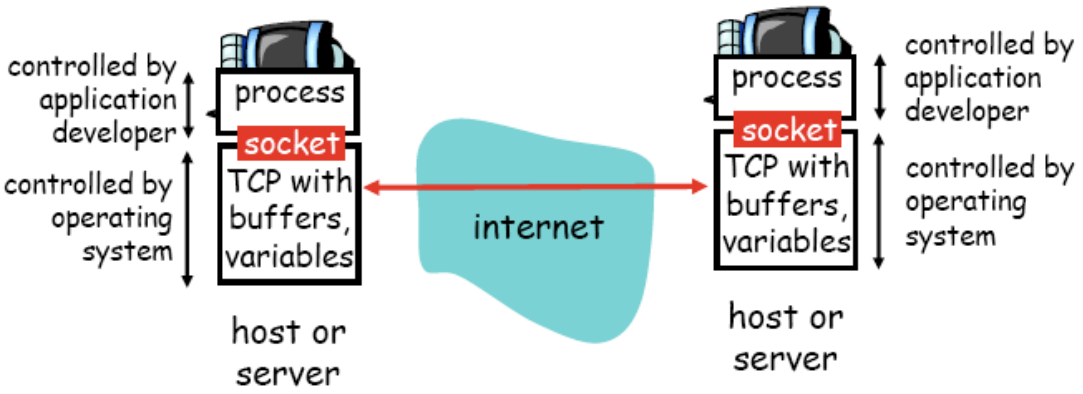
Socket API

- Unreliable datagram
- Reliable, byte stream-oriented

a *host-local*, *application-created*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another application process

Socket: A door between application process and end-end-trasport protocol (UDP or TCP)

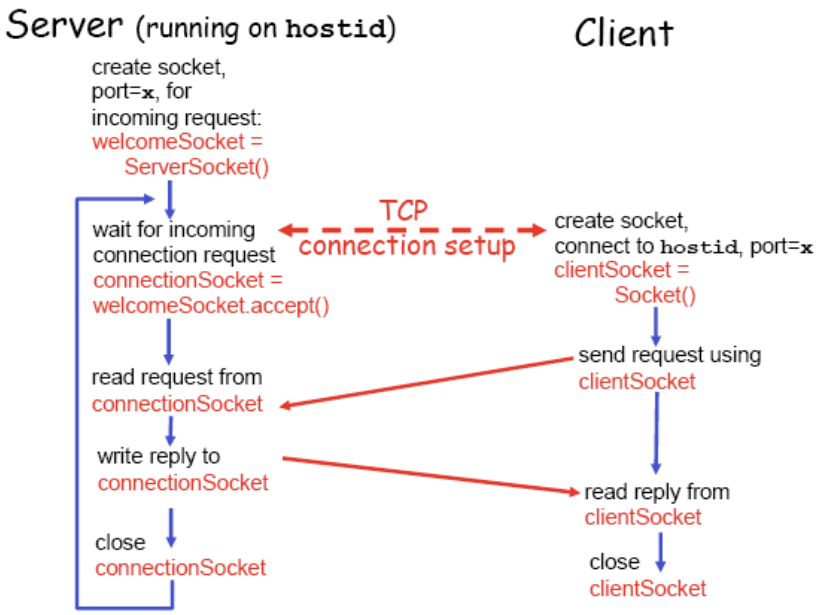
It is reliable transfer of bytes from one process to another.



- Client **must contact** server (server process must first be running, and then create a socket)
- **Client contacts** server by (client-local TCP Socket, specifying IP, port and establish connection when Socket is created.
- When contacted by client (server TCP creates new socket, for server process to communicatie with client).

Application viewpoint: // *TCP provides reliable, in-order transfer of bytes ("pipe") between client and server.*

Client/Server interaction (TCP)



Stream

A stream is a sequence of characters that flow into or out of a process.

- Input Stream
  - is attached to some input source for the process,
- Output Stream
  - is attached to an output source, e.g., monitor or socket.

Java Example (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

   public static void main(String argv[]) throws Exception
   {
      String sentence;
      String modifiedSentence;

      BufferedReader inFromUser =
       new BufferedReader(new InputStreamReader(System.in));

      Socket clientSocket = new Socket("hostname", 6789);

      DataOutputStream outToServer =
       new DataOutputStream(clientSocket.getOutputStream());

      BufferedReader inFromServer =
        new BufferedReader(new
        InputStreamReader(clientSocket.getInputStream()));

      sentence = inFromUser.readLine();

      outToServer.writeBytes(sentence + '\n');

      modifiedSentence = inFromServer.readLine();

      System.out.println("FROM SERVER: " + modifiedSentence);

      clientSocket.close();

   }
}
```

Create input stream

Create client socket, connect to server

Create output stream attached to socket

Create input stream attached to socket

Send line to server

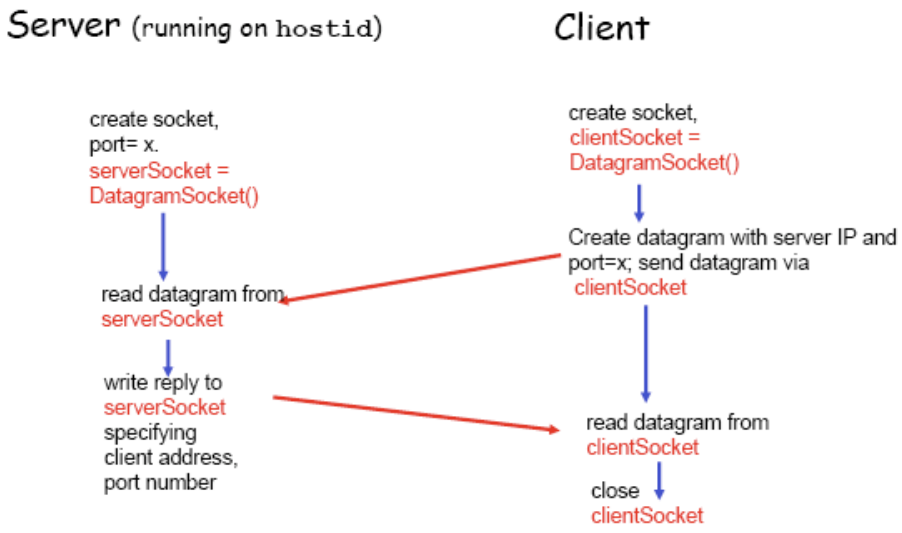
Read line from server

Socket Programming (UDP)

- No "Connection" between client and server.
- No handshaking, and sender explicitly attaches IP address and Port of destination to each packet
- Server must extract IP address, Port of sender from received packet.

Application viewpoint: // UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server.

Client/Server interaction (UDP)



Java Example (UDP)

```
import java.io.*;
import java.net.*;

class UDPClient {
   public static void main(String args[]) throws Exception
   {
      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      DatagramSocket clientSocket = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName("hostname");

      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];

      String sentence = inFromUser.readLine();
      sendData = sentence.getBytes();

      DatagramPacket sendPacket =
        new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

      clientSocket.send(sendPacket);

      DatagramPacket receivePacket =
        new DatagramPacket(receiveData, receiveData.length);

      clientSocket.receive(receivePacket);

      String modifiedSentence =
         new String(receivePacket.getData());

      System.out.println("FROM SERVER:" + modifiedSentence);
      clientSocket.close();
      }
}
```

Create input stream

Create client socket

Translate hostname to IP address using DNS

Create datagram with data-to-send, length, IP addr, port

Send datagram to server

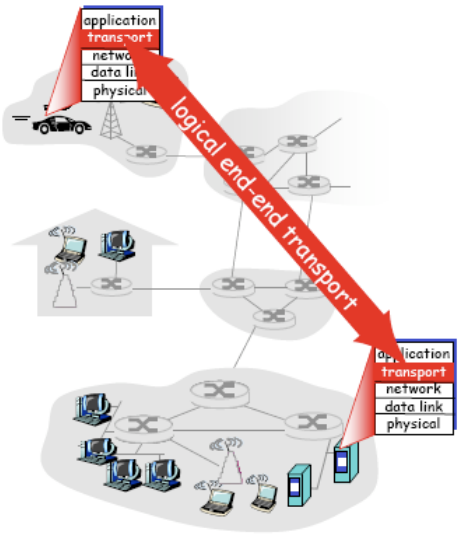
Read datagram from server

9. Transport Layer I

- Provide logical communication between application processes running on different hosts
- Transport protocols run in end systems
  - Send side (breaks application in to segments)
  - Receive side: reassembles segments into messages.
- More than one transport protocol available to applications
  - TCP and UDP

Transport layer vs. network layer

The network layer is logical communication between hosts, and the transport layers is between processes, which relies on, enhanced network layer services.



Example:
12 kids sending letters to 12 kids, where **processes** = kids and the **application messages** = letters in envelopes, and **hosts** = houses. The **transport protocol** = Ann and Bill in the **network-layer protocol** = postal service.

Internet transport-layer protocol

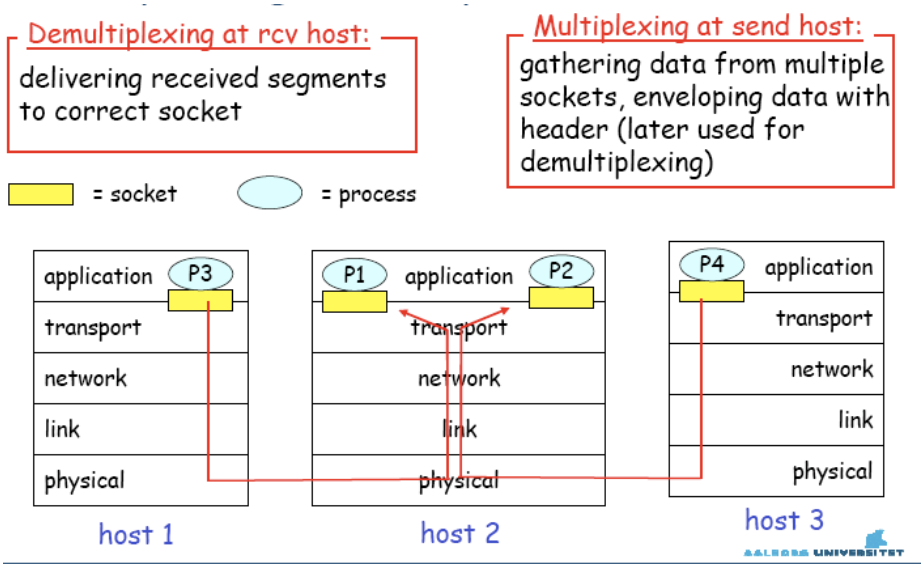
TCP (reliable – inorder delivery)

- Congestion control
- Flow control
- Connections setup

UDP (unreliable – unorder delivery)

- no-frills extension of "best effort" IP
- delay guarantees
- bandwidth guarantees

Multiplexiing and demultiplexing

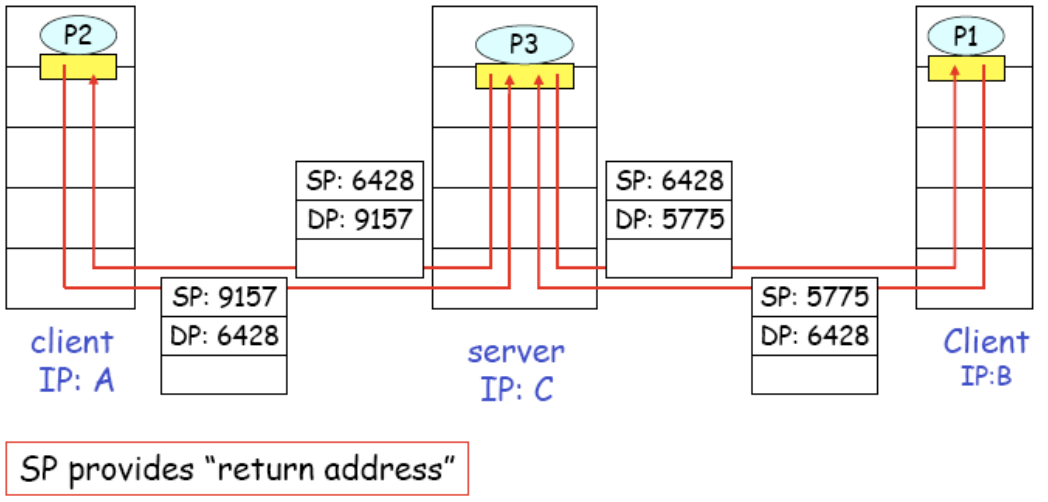


Work flow

Each datagram has source IP address, and it carries 1 transport-layer segment, and each segment has source destination port number. The host uses IP addresses and port number to direct segment to appropriate socket.

When a host receives UDP segment (with 2 tuple), it checks destination port number and directs UDP segment to socket with that number.

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```



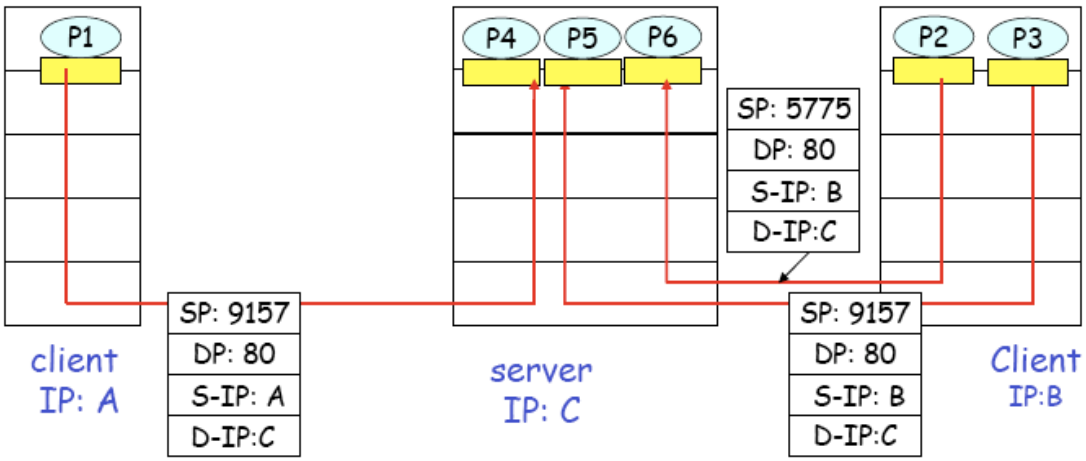
Connection-oriented demux

TCP socket identified by 4-tuple:
- source IP address
- source port number
- dest IP address
- dest port number

recv host 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
- non-persistent HTTP will have different socket for each

Connection-oriented demux (cont)



Connectionless transport (UDP)

- "no frills", "Bare bones"
- "best effort" service
  - lost
  - delivered out of order to application
- connectionless
  - no handshaking between sender and receiver.
  - Each UDP segment handled independently of others.

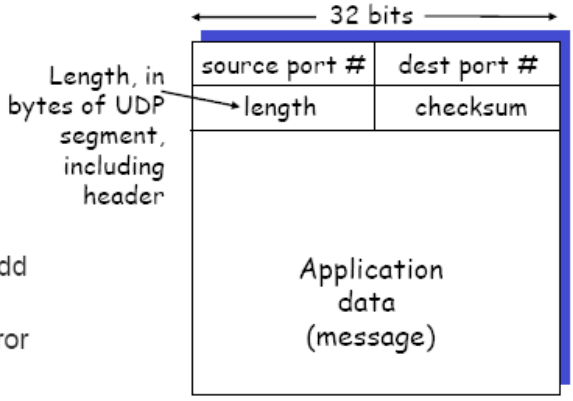
UDP is often used for streaming multimedia applicstions, because it is loss tolerant, and rate sensitive. It is being used by DNS and SNMP, and gives a reliable transfer over UDP, with application-specific error recovery.

UDP: more

often used for streaming multimedia applications
- loss tolerant
- rate sensitive

other UDP uses
- DNS
- SNMP

reliable transfer over UDP: add reliability at application layer
- application-specific error recovery!



UDP segment format

UDP Checksum

Sender:
treat segment contents as sequence of 16-bit integers
checksum: addition (1's complement sum) of segment contents
sender puts checksum value into UDP checksum field

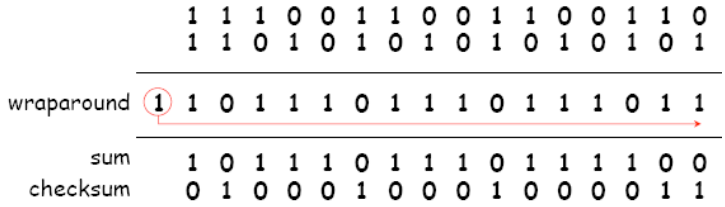
Receiver:
compute checksum of received segment
check if computed checksum equals checksum field value:
- NO - error detected
- YES - no error detected. *But maybe errors nonetheless?* More later ….

Internet Checksum Example

Note

- When adding numbers, a carryout from the most significant bit needs to be added to the result

Example: add two 16-bit integers



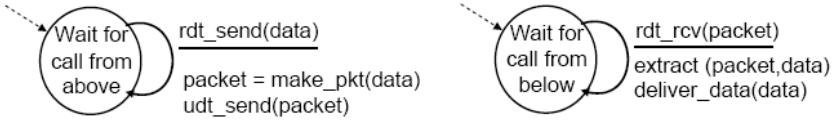
Principles of Reliable data transfer

- **important in app., transpor, link layers**
- **top10 list important networking topics**
- **characteristics of unreliable channel will determine complexity of rdt.**

RDT 1.0

➢underlying channel perfectly reliable
- no bit errors
- no loss of packets

➢separate FSMs for sender, receiver:
- sender sends data into underlying channel
- receiver read data from underlying channel



sender receiver

RDT 2.0

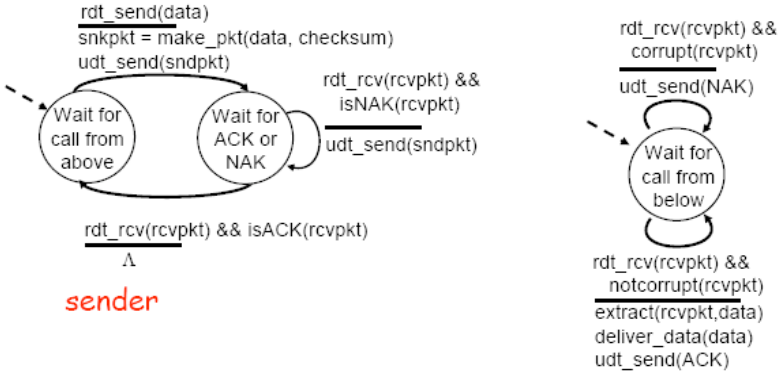
➢underlying channel may flip bits in packet
- checksum to detect bit errors

➢*the* question: how to recover from errors:
- *acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK
- *negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors
- sender retransmits pkt on receipt of NAK

➢new mechanisms in `rdt2.0` (beyond `rdt1.0`):
- error detection
- receiver feedback: control msgs (ACK,NAK) rcvr->sender

rdt2.0: FSM specification

receiver



sender

rdt2.0 has a fatal flaw!

What happens if ACK/NAK corrupted?

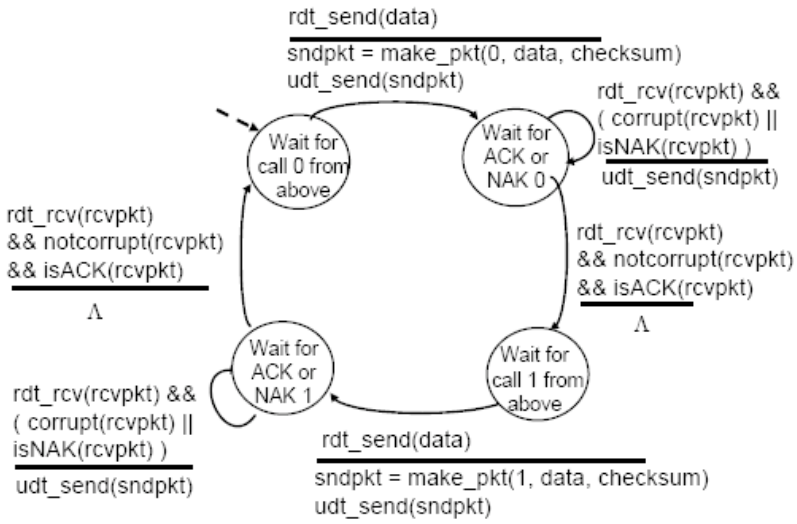
- sender doesn't know what happened at receiver!
- can't just retransmit: possible duplicate

Handling duplicates:

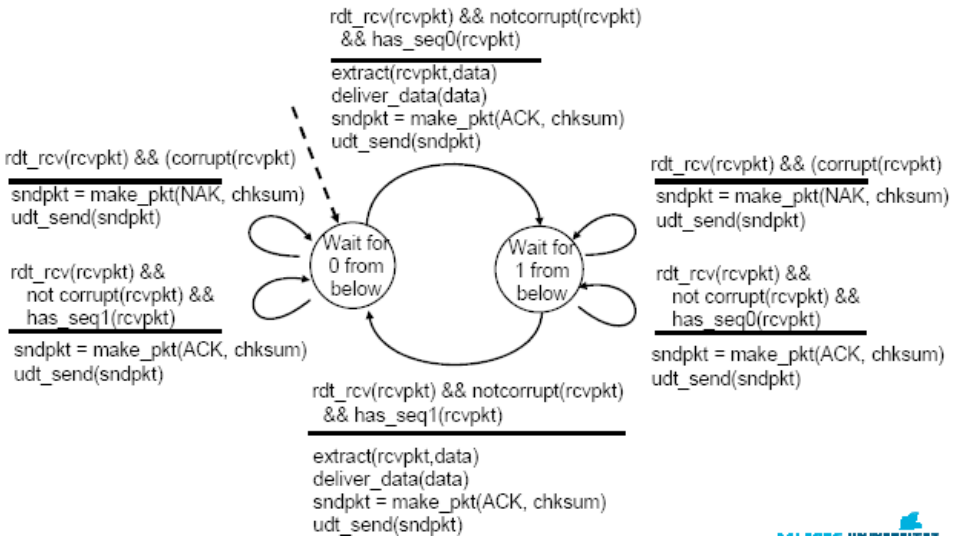
- sender retransmits current pkt if ACK/NAK garbled
- sender adds sequence number to each pkt
- receiver discards (doesn't deliver up) duplicate pkt

stop and wait
Sender sends one packet, then waits for receiver response

rdt2.1: sender, handles garbled ACK/I



rdt2.1: receiver, handles garbled ACK/NAKs



rdt2.1: discussion

Sender:

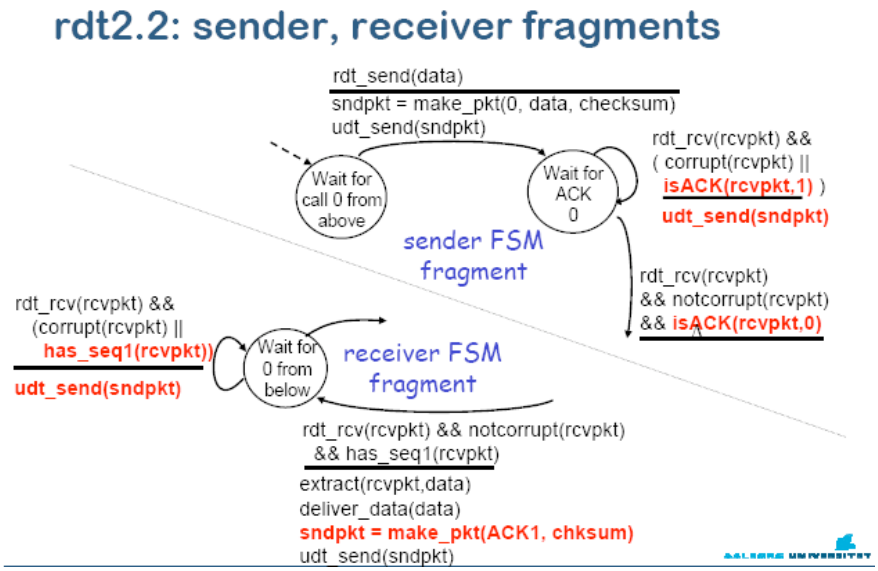
- seq # added to pkt
- two seq. #'s (0,1) will suffice. Why?
- must check if received ACK/NAK corrupted
- twice as many states
  - state must "remember" whether "current" pkt has 0 or 1 seq. #

Receiver:

- must check if received packet is duplicate
  - state indicates whether 0 or 1 is expected pkt seq #
- note: receiver can not know if its last ACK/NAK received OK at sender

rdt2.2: a NAK-free protocol

- same functionality as rdt2.1, using ACKs only
- instead of NAK, receiver sends ACK for last pkt received OK
  - receiver must explicitly include seq # of pkt being ACKed
- duplicate ACK at sender results in same action as NAK: retransmit current pkt



RDT 3.0

rdt3.0: channels with errors and loss

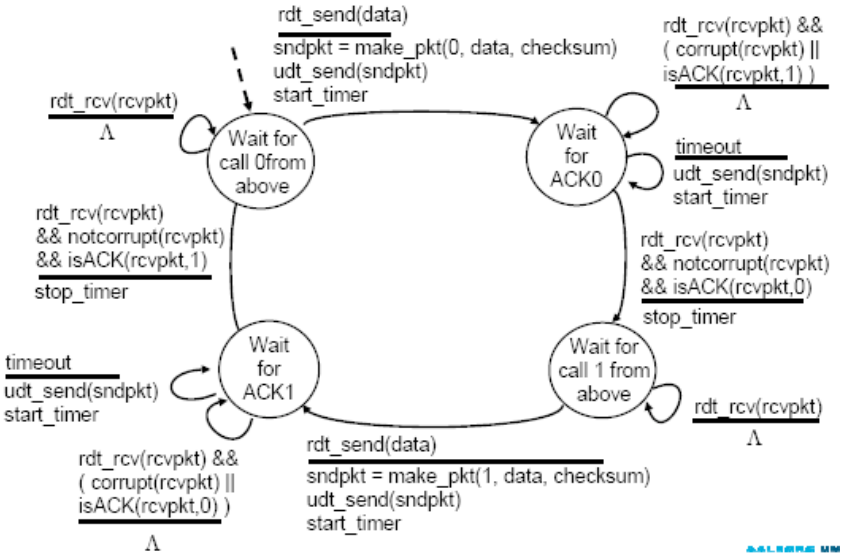
New assumption: underlying channel can also lose packets (data or ACKs)

- checksum, seq. #, ACKs, retransmissions will be of help, but not enough

Approach: sender waits "reasonable" amount of time for ACK retransmits if no ACK received in this time if pkt (or ACK) just delayed (not lost):

- retransmission will be duplicate, but use of seq. #'s already handles this
- receiver must specify seq # of pkt being ACKed requires countdown timer

rdt3.0 sender



Performance of rdt3.0

- rdt3.0 works, but performance stinks
- ex: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

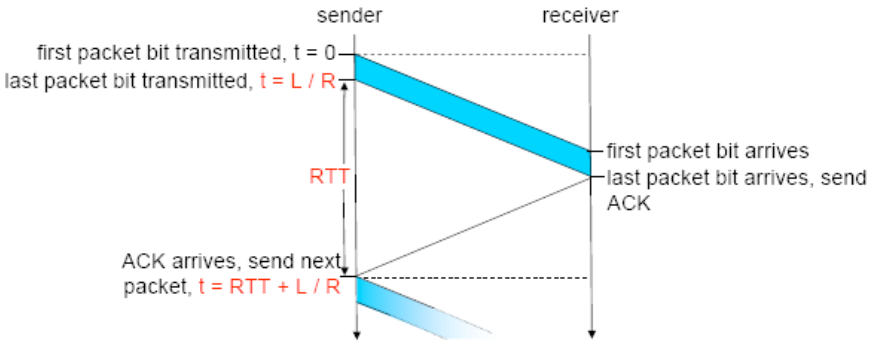
$$d_{trans} = \frac{L}{R} = \frac{8000\text{bits}}{10^9\text{ bps}} = 8\,\text{microseconds}$$

- U_{sender}: utilization – fraction of time sender busy sending

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

- 1KB pkt every 30 msec -> 33kB/sec thruput over 1 Gbps link
- network protocol limits use of physical resources!

rdt3.0: stop-and-wait operation



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

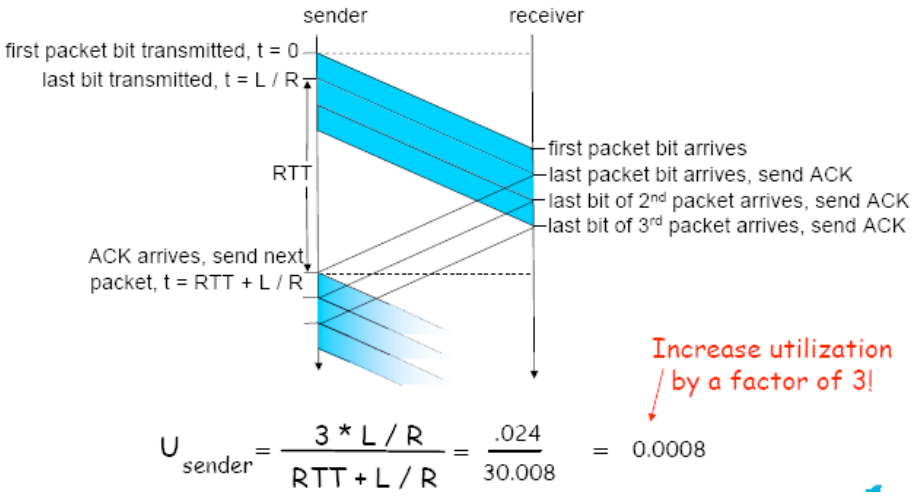
Pipelined protocols

Pipelining: sender allows multiple, "in-flight", yet-to-beacknowledged pkts

- range of sequence numbers must be increased
- buffering at sender and/or receiver

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

Pipelining: increased utilization



Pipelining Protocols

Go-back-N: big picture:
Sender can have up to N unacked packets in pipeline
Rcvr only sends cumulative acks

- Doesn't ack packet if there's a gap

Sender has timer for oldest unacked packet

- If timer expires, retransmit all unacked packets

Selective Repeat: big picture
Sender can have up to N unacked packets in pipeline
Rcvr acks individual packets
Sender maintains timer for each unacked packet

- When timer expires, retransmit only unack packet

Selective repeat: big picture

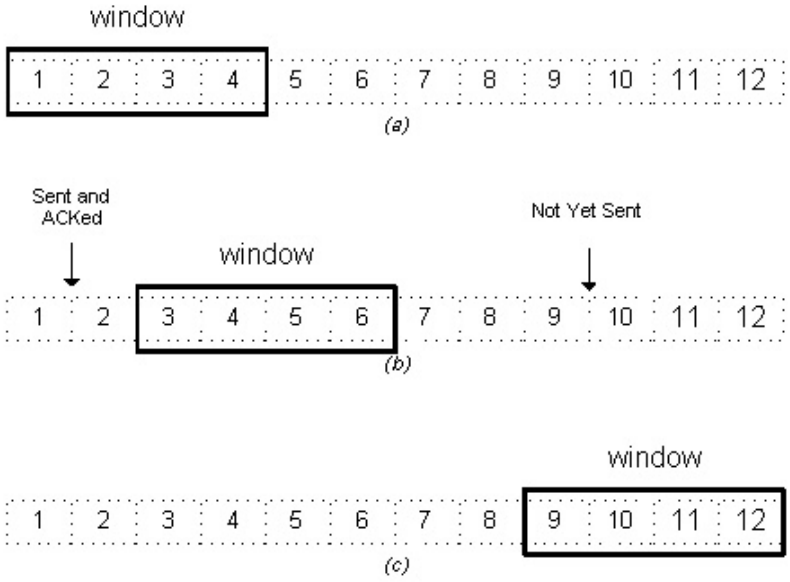
Sender can have up to N unacknowledged packets in pipeline
Receiver acknowledges individual packets
Sender maintains timer for each unacknowledged packet

- When timer expires, retransmit only unacknowledged packet

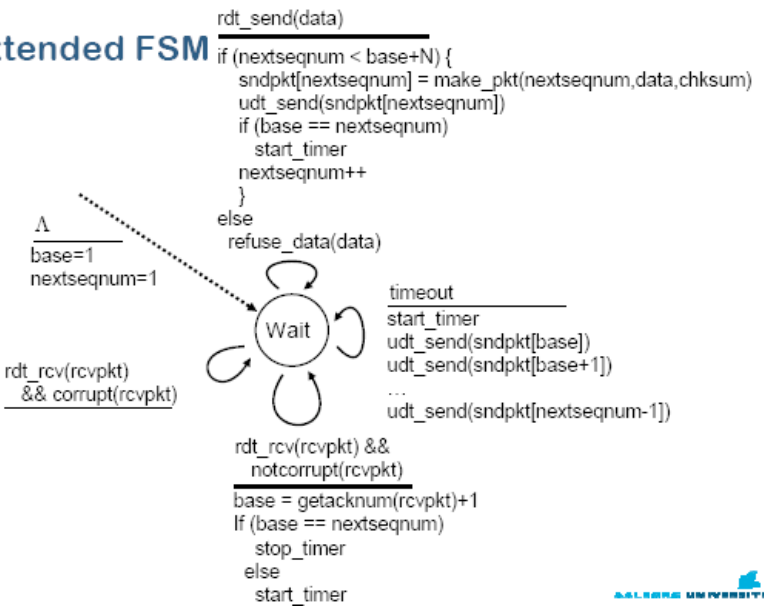
Go-Back-N (GBN protocol)

The sender is allowed to transmit multiple packets (when available) without waiting for acknowledgment, but is constrained to have no more than some maximum allowable number N, of unacknowledged packets in the pipeline.

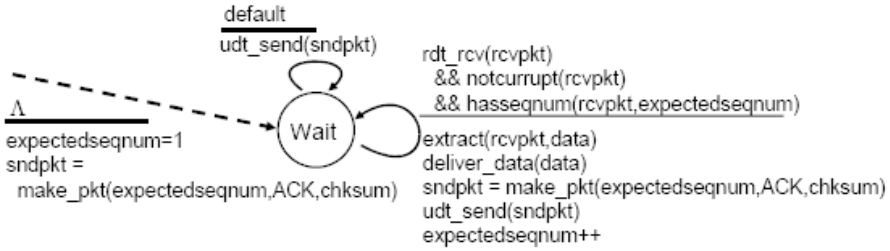
- N often referred to as the window size.
- GBN protocol often called Sliding-window protocol.



GBN: sender extended FSM



GBN: receiver extended FSM



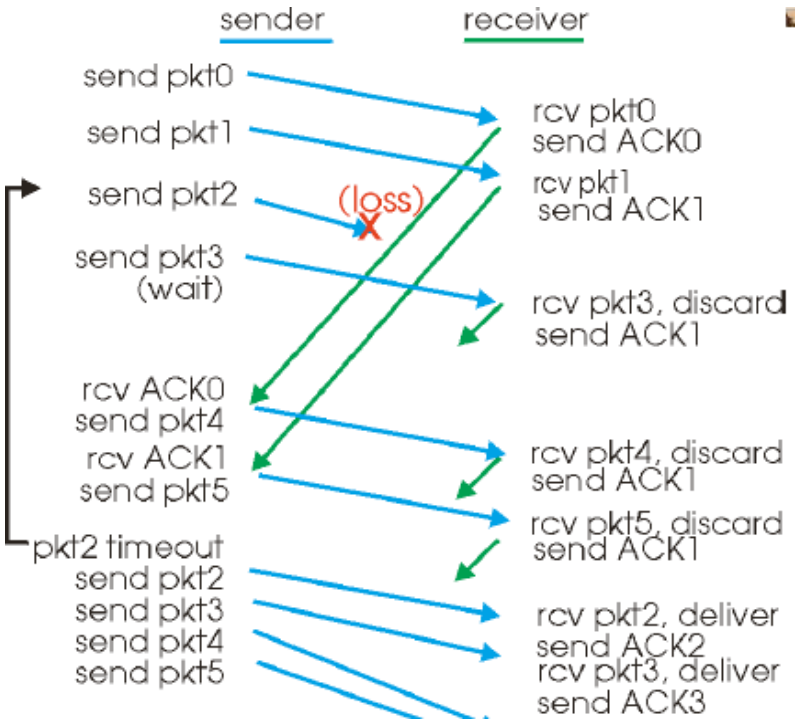
ACK-only: always send ACK for correctly-received pkt with highest *in-order* seq #

- may generate duplicate ACKs
- need only remember **`expectedseqnum`**

out-of-order pkt:

- discard (don't buffer) -> no receiver buffering!
- Re-ACK pkt with highest in-order seq #

GBN in action



Selective Repeat

receiver individually acknowledges all correctly received packets

- buffers packets, as needed, for eventual in-order delivery to upper layer

sender only resends pkts for which ACK not received

- sender timer for each unACKed packet

sender window

- N consecutive seq #'s
- again limits seq #s of sent, unACKed packets

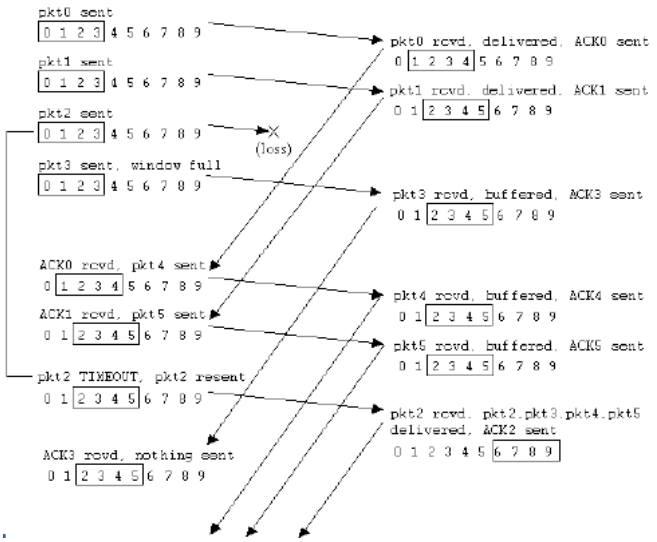
Selective repeat

sender

data from above :
if next available seq # in window, send pkt
timeout(n):
resend pkt n, restart timer
ACK(n) in [sendbase,sendbase+N]:
mark pkt n as received
if n smallest unACKed pkt, advance window base to next unACKed seq #

receiver

pkt n in [rcvbase, rcvbase+N-1]
send ACK(n)
out-of-order: buffer
in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt
pkt n in [rcvbase-N,rcvbase-1]
ACK(n)
otherwise:
ignore



Selective repeat: dilemma

Example:
seq #'s: 0, 1, 2, 3
window size=3

- receiver sees no difference in two scenarios!
- incorrectly passes duplicate data as new in (a)

Q: what relationship between seq# size and window size?

10. Transport Layer II

TCP : Connection Oriented Transport :

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

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- *send & receive buffers*

- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

TCP-header 20 byte long / **UDP**-header 8 byte long.

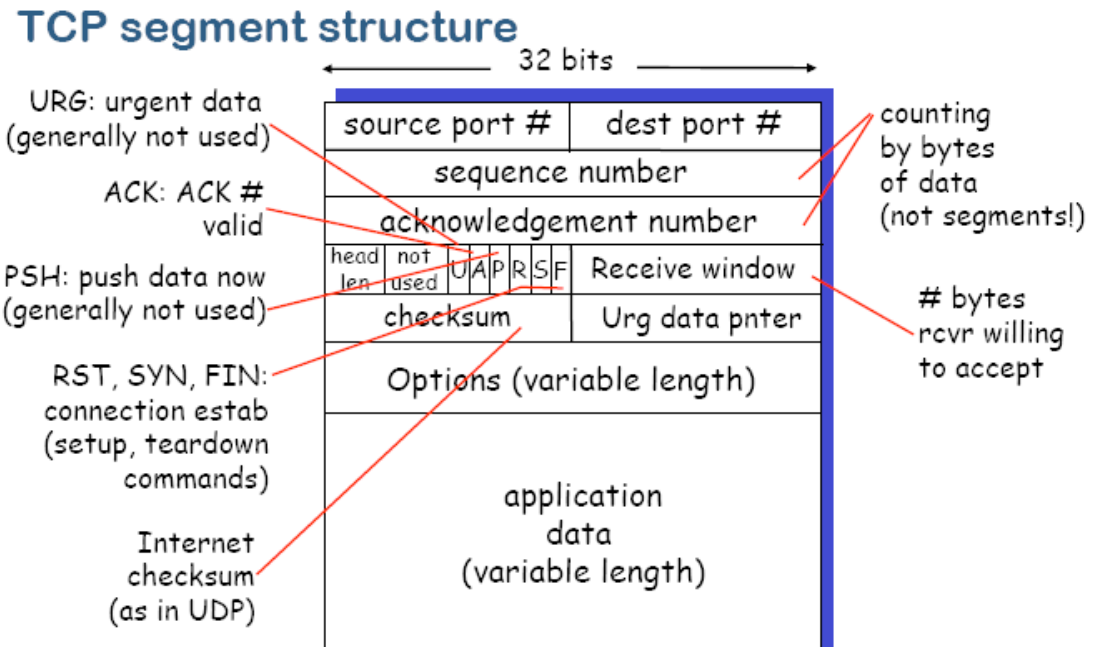
Maximum Segment Size (**MSS**)

Maximum transmission unit (MTU) typical values are 1460, 536, 512. Byts.

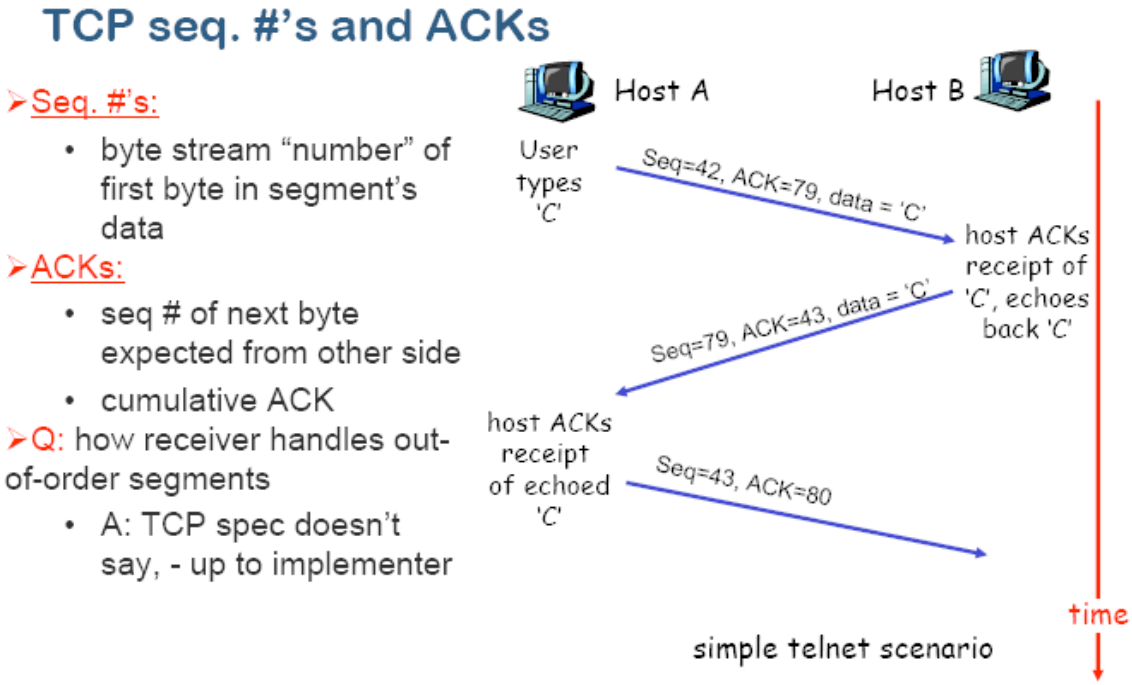
Ex. File consisting of 500.000 bytes, **MMS** is 1000 bytes. First sequence number is 0, next is 1000…

First sequence number is random.

Acknowledgement number is the next byte expected from other side.



There is many options to set up for the segment, besides the data which you want to send. It has 2 tuple and 4 tuple, where the 2 tuple is with 32 bits,



TCP Round Trip Time and 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

TCP Round Trip Time and Timeout

$$\text{EstimatedRTT} = (1-\alpha)*\text{EstimatedRTT} + \alpha*\text{SampleRTT}$$

Exponential weighted moving average
influence of past sample decreases exponentially fast
typical value: $\alpha = 0.125$

TCP Round Trip Time and Timeout

Setting the timeout

EstimtedRTT plus "safety margin"

- large variation in **EstimatedRTT** -> larger safety margin

first estimate of how much SampleRTT deviates from EstimatedRTT:

$$\text{DevRTT} = (1-\beta)*\text{DevRTT} + \beta*|\text{SampleRTT}-\text{EstimatedRTT}|$$

(typically, $\beta = 0.25$)

Then set timeout interval:

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4*\text{DevRTT}$$

TCP reliable data transfer

TCP sender events

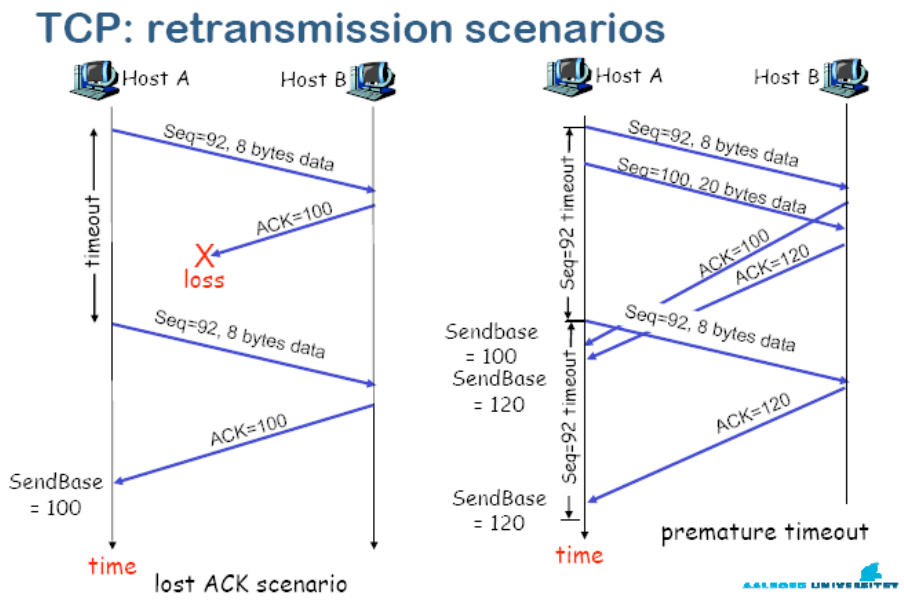
Data received from application:
Create segment with seq #
seq # is byte-stream number of first data byte in segment start timer if not already running (think of timer as for oldest unacked segment)
expiration interval: TimeOutInterval

Timeout:
retransmit segment that caused timeout
restart timer

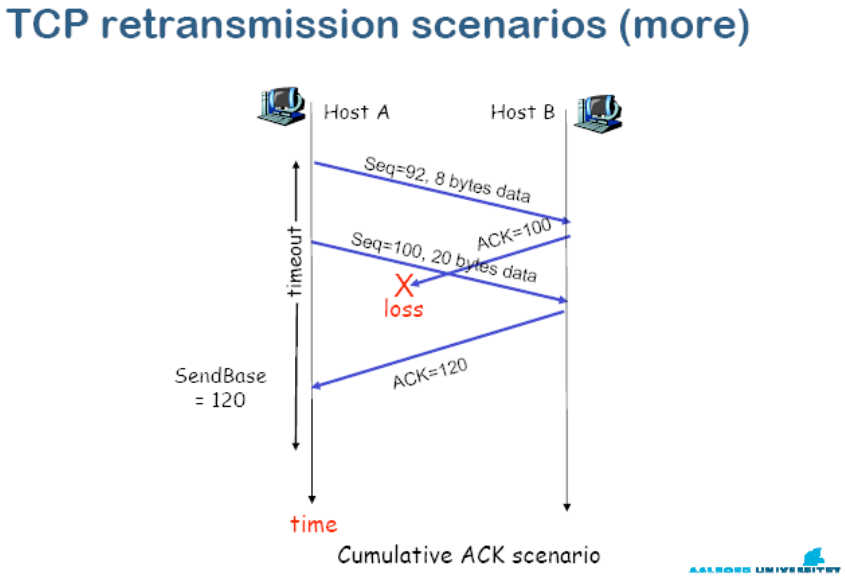
ACK received:
If acknowledges previously unacked segments
• update what is known to be acked
• start timer if there are outstanding segments

```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum
 loop (forever) {
   switch(event)

   event: data received from application above
       create TCP segment with sequence number NextSeqNum
       if (timer currently not running)
            start timer
       pass segment to IP
       NextSeqNum = NextSeqNum + length(data)

    event: timer timeout
       retransmit not-yet-acknowledged segment with
            smallest sequence number
       start timer
    event: ACK received, with ACK field value of y
       if (y > SendBase) {
            SendBase = y
           if (there are currently not-yet-acknowledged segments)
                 start timer
           }
 } /* end of loop forever */
```

Comment:
• SendBase-1: last cumulatively ack'ed byte
Example:
• SendBase-1 = 71; y= 73, so the rcvr wants 73+ ;
y > SendBase, so that new data is acked



The second picture, shows how the two sequences didn't come trough before the timeout, and it try again, and the one, with the sequence 92, 8 bytes, has the ACK on, 120.



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

Fast Retransmit

Time-out period often relatively long (see cumulative ACK scenario):

- long delay before resending lost packet

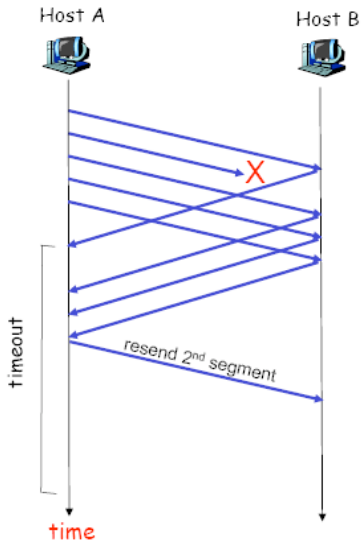
Detect lost segments via duplicate ACKs.

- Sender often sends many segments back-to-back
- If segment is lost, there will likely be many duplicate ACKs.

If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:

- fast retransmit: resend segment before timer expires

Resending a segment
after triple duplicate ACK



Fast retransmit algorithm:

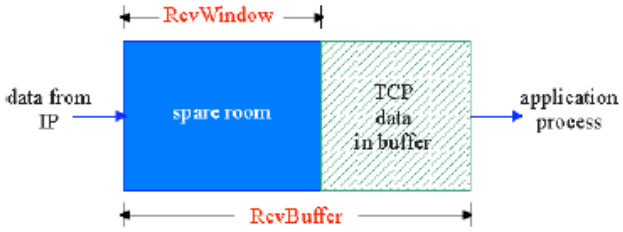
```
event: ACK received, with ACK field value of y
        if (y > SendBase) {
            SendBase = y
            if (there are currently not-yet-acknowledged segments)
                start timer
        }
        else {
            increment count of dup ACKs received for y
            if (count of dup ACKs received for y = 3) {
                resend segment with sequence number y
            }
```

a duplicate ACK for already ACKed segment

fast retransmit

TCP Flow Control

receive side of TCP connection has a receive buffer:

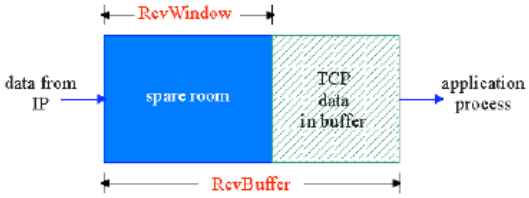


application process may be slow at reading from buffer

flow control
sender won't overflow receiver's buffer by transmitting too much, too fast

speed-matching service: matching the send rate to the receiving app's drain rate

TCP Flow control: how it works



Rcvr advertises spare room by including value of **`RcvWindow`** in segments
Sender limits unACKed data to **`RcvWindow`**

- guarantees receive buffer doesn't overflow

(Suppose TCP receiver discards out-of-order segments)
spare room in buffer
`= RcvWindow`
`= RcvBuffer-[LastByteRcvd - LastByteRead]`

TCP Connection Management

Recall: TCP sender, receiver establish "connection" before exchanging data segments
initialize TCP variables:

- seq. #s
- buffers, flow control info (e.g. **`RcvWindow`**)

client: connection initiator

```
Socket clientSocket = new Socket("hostname","port number");
```

Server : contacted by client

```
Socket connectionSocket = welcomeSocket.accept();
```

Three way handshake:

Step 1: client host sends TCP SYN segment to server

- specifies initial seq #
- no data

Step 2: server host receives SYN, replies with SYNACK segment

- server allocates buffers
- specifies server initial seq. #

Step 3: client receives SYNACK, replies with ACK segment, which may contain data

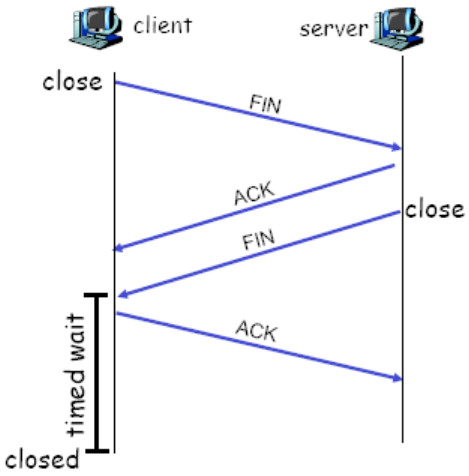
TCP Connection Management (cont.)

Closing a connection:

client closes socket:
`clientSocket.close();`

Step 1: client end system sends TCP FIN control segment to server

Step 2: server receives FIN, replies with ACK. Closes connection, sends FIN.



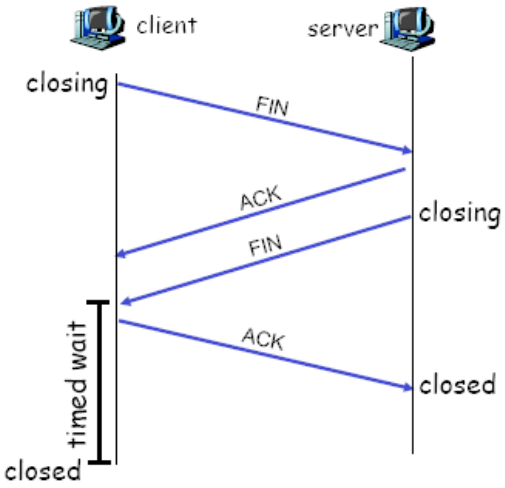
TCP Connection Management (cont.)

Step 3: client receives FIN, replies with ACK.

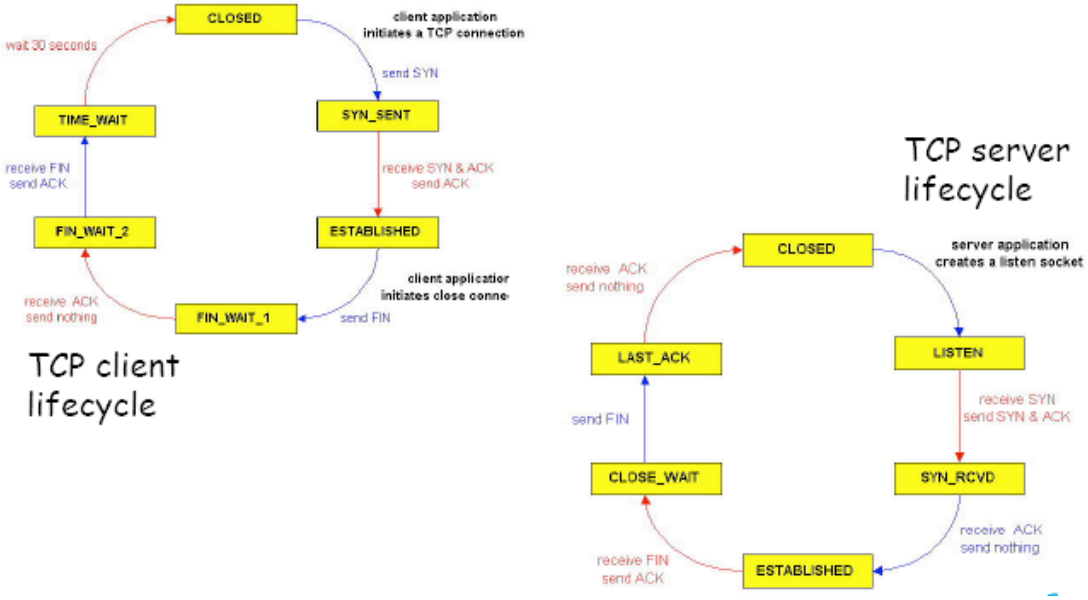
- Enters "timed wait" - will respond with ACK to received FINs

Step 4: server, receives ACK. Connection closed.

Note: with small modification, can handle simultaneous FINs.



TCP Connection Management (cont)



Principles of Congestion Control

Congestion:

Informally: "too many sources sending too much data too fast for network to handle" different from flow control!

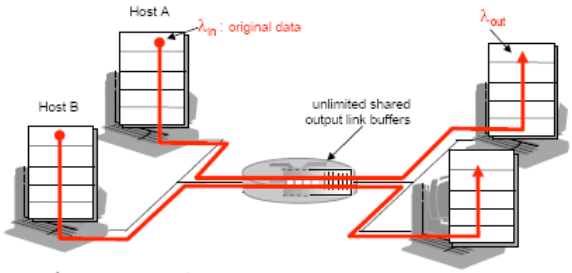
Manifestations:

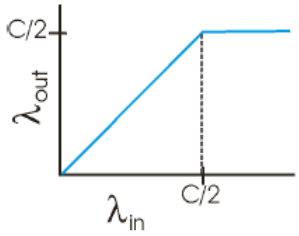
- lost packets (buffer overflow at routers)
- long delays (queueing in router buffers)

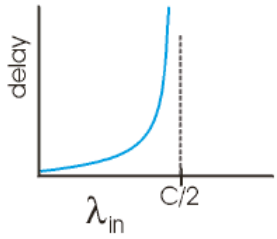
a top-10 problem!

Causes/costs of congestion: scenario 1

- two senders, two receivers
- one router, infinite buffers
- no retransmission



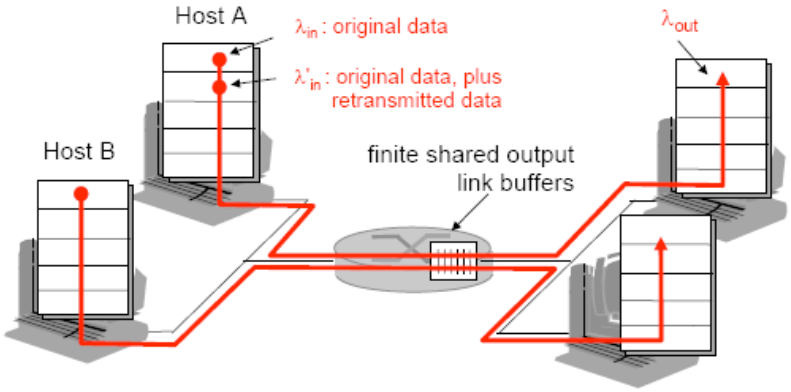




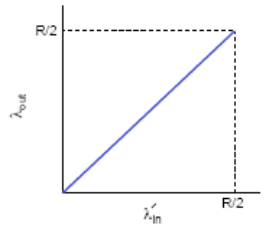
- large delays when congested
- maximum achievable throughput

Causes/costs of congestion: scenario 2

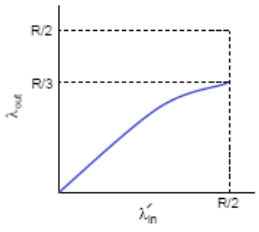
- one router, *finite* buffers
- sender retransmission of lost packet



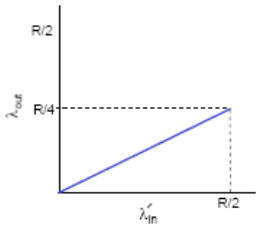
- always: $\lambda_{in} = \lambda_{out}$ (goodput)
- "perfect" retransmission only when loss: $\lambda'_{in} > \lambda_{out}$
- retransmission of delayed (not lost) packet makes λ'_{in} larger (than perfect case) for same λ_{out}



a.



b.



c.

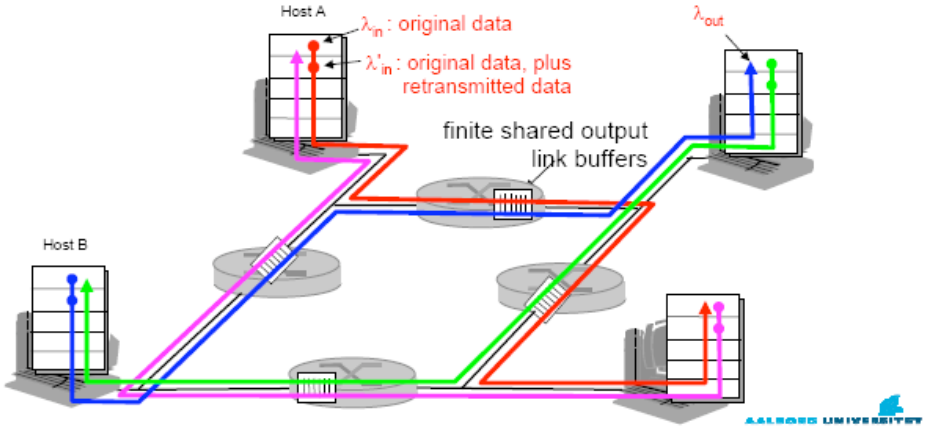
"costs" of congestion:

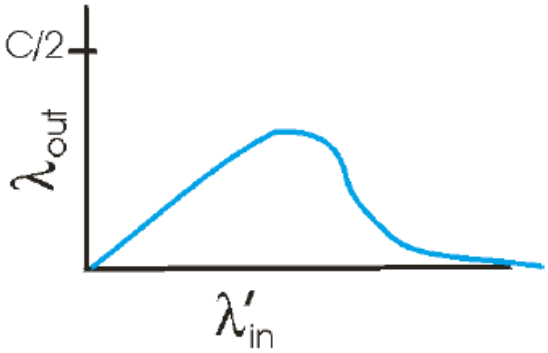
- more work (retrans) for given "goodput"
- unneeded retransmissions: link carries multiple copies of pkt

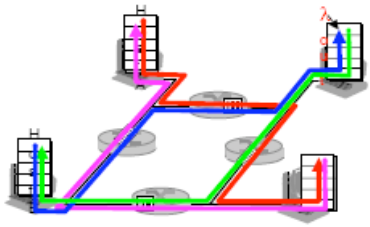
Causes/costs of congestion: scenario 3

- four senders
- multihop paths
- timeout/retransmit

Q: what happens as λ_{in} and λ'_{in} increase ?







Another "cost" of congestion:

- when packet dropped, any "upstream transmission capacity used for that packet was wasted!

2 broad approaches towards congestion control

End-end congestion control:

no explicit feedback from network

congestion inferred from end-system observed loss, delay

approach taken by TCP

Network-assisted congestion control:

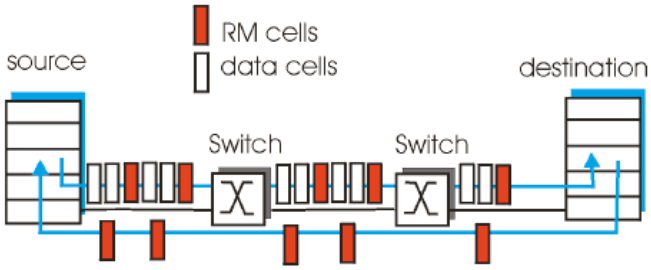
routers provide feedback to end systems

- single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
- explicit rate sender should send at

Case study: ATM ABR congestion control

ABR: available bit rate:
"elastic service"
if sender's path "underloaded":
• sender should use available bandwidth if sender's path congested:
• sender throttled to minimum guaranteed rate
RM (resource management) cells:
sent by sender, interspersed with data cells bits in RM cell set by switches ("network-assisted")
• NI bit: no increase in rate (mild congestion)
• CI bit: congestion indication
RM cells returned to sender by receiver, with bits intact

Case study: ATM ABR congestion control



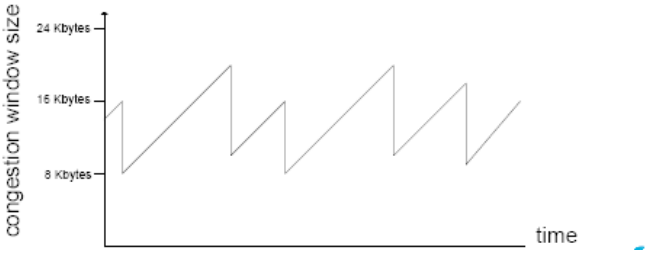
two-byte ER (explicit rate) field in RM cell
• congested switch may lower ER value in cell
• sender' send rate thus maximum supportable rate on path
EFCI bit in data cells: set to 1 in congested switch
• if data cell preceding RM cell has EFCI set, sender sets CI bit in returned RM cell

TCP congestion control: additive increase, multiplicative decrease

Approach: increase transmission rate (window size), probing for usable bandwidth, until loss occurs

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

Saw tooth behavior: probing for bandwidth



TCP Congestion Control: details

➢sender limits transmission:
LastByteSent-LastByteAcked ≤ CongWin

➢Roughly,

$$\text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}$$

➢**CongWin** is dynamic, function of perceived network congestion

How does sender perceive congestion?
loss event = timeout *or* 3 duplicate acks
TCP sender reduces rate (**CongWin**) after loss event
three mechanisms:

- AIMD
- slow start
- conservative after timeout events

TCP Slow Start

When connection begins, CongWin = 1 MSS

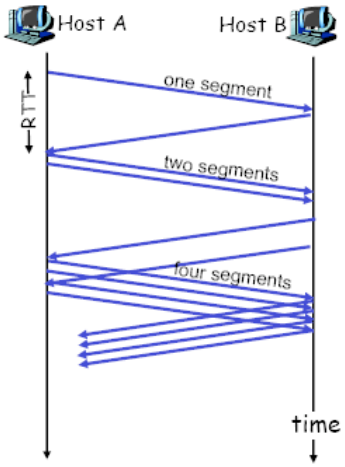
- Example: MSS = 500 bytes & RTT = 200 msec
- initial rate = 20 kbps available bandwidth may be >> MSS/RTT
- desirable to quickly ramp up to respectable rate

When connection begins, increase rate exponentially fast until first loss event

➢When connection begins, increase rate exponentially until first loss event:

- double **CongWin** every RTT
- done by incrementing **CongWin** for every ACK received

➢Summary: initial rate is slow but ramps up exponentially fast



Refinement: inferring loss

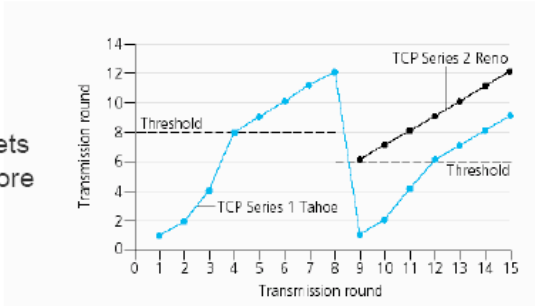
- After 3 dup ACKs:
  - `CongWin` is cut in half
  - window then grows linearly
- But after timeout event:
  - `CongWin` instead set to 1 MSS;
  - window then grows exponentially
  - to a threshold, then grows linearly

Philosophy:

- 3 dup ACKs indicates network capable of delivering some segments
- timeout indicates a "more alarming" congestion scenario

Refinement

Q: When should the exponential increase switch to linear?
A: When `CongWin` gets to 1/2 of its value before timeout.



Implementation:
Variable Threshold
At loss event, Threshold is set to 1/2 of CongWin just before loss event

Summary: TCP

When CongWin is below Threshold, sender in slowstart phase, window grows exponentially.

When CongWin is above Threshold, sender is in congestion-avoidance phase, window grows linearly.

When a triple duplicate ACK occurs, Threshold set to CongWin/2 and CongWin set to Threshold.

When timeout occurs, Threshold set to CongWin/2 and CongWin is set to 1 MSS.

TCP sender congestion control

State	Event	TCP Sender Action	Commentary
Slow Start (SS)	ACK receipt for previously unacked data	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to "Congestion Avoidance"	Resulting in a doubling of CongWin every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
SS or CA	Loss event detected by triple duplicate ACK	Threshold = CongWin/2, CongWin = Threshold, Set state to "Congestion Avoidance"	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
SS or CA	Timeout	Threshold = CongWin/2, CongWin = 1 MSS, Set state to "Slow Start"	Enter slow start
SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

TCP throughput

- What's the average throughout of TCP as a function of window size and RTT?
  - Ignore slow start
- Let W be the window size when loss occurs.
- When window is W, throughput is W/RTT
- Just after loss, window drops to W/2, throughput to W/2RTT.
- Average throughout: .75 W/RTT

TCP Futures: TCP over "long, fat pipes"

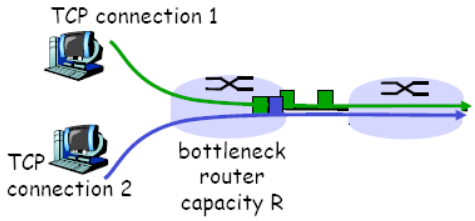
- Example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- Requires window size W = 83,333 in-flight segments
- Throughput in terms of loss rate:

$$\frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

- ➜ $L = 2 \cdot 10^{-10}$
- New versions of TCP for high-speed

TCP Fairness

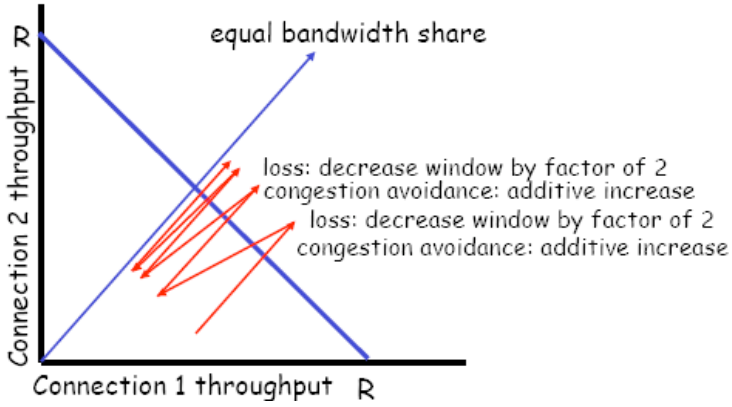
Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Why is TCP fair?

Two competing sessions:

- Additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally



Fairness (more)

Fairness and UDP

- Multimedia applications often do not use TCP
  - do not want rate throttled by congestion control
- Instead use UDP:
  - pump audio/video at constant rate, tolerate packet loss
- Research area: TCP friendly

Fairness and parallel TCP connections

- nothing prevents application from opening parallel connections between 2 hosts.
- Web browsers do this
- Example: link of rate R supporting 9 connections;
  - new application asks for 1 TCP, gets rate R/10
  - new application asks for 11 TCPs, gets R/2 !

11. Network Layer I

Goals

The goals of this lecture is to understand the

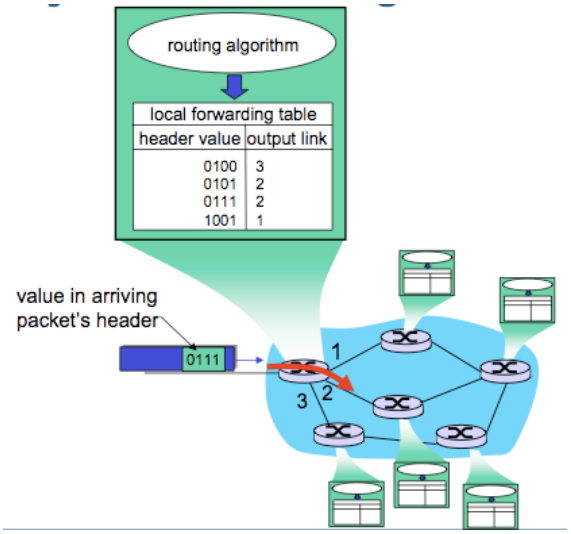
- Service models
- Forwarding packets

Transport segment from sending to receiving host
On sending side it encapsulates the segments into **datagram**
On receiving side, it delivers the segments to the **transport layer**
Network layer protocols in **every host and router**
Router examines header fields in **all IP datagram's passing** through it

An analogy

- *forwarding:* move packets from router's input to appropriate router output
- *routing:* determine route taken by packets from source to destination.
  - *routing algorithms*
- routing: process of planning trip from source to destination
- forwarding: process of getting through single interchange

Interplay between routing and forwarding
At every router, we have the router algorithm, and we have forwarding algorithm, where it uses the forwarding table.



Example
0111 (Bus no. 2) is arriving the router (Bus terminal).
Then I have to take another router(bus), to get to the destination. The routers are the different busses.

Connection setup

- Some network architectures (ATM, frame relay, X25)
- Routers establish virtual communication
- Network vs. transport layer connection service (network: between 2 hosts & transport: between two processes).

Datagrams

Individual datagram's

- Guarenteed delivery
- Guaranteed delivery with less than 40 msec delay

Flow of datagram's

- In-order datagram deliver (traversal)
- Guaranteed minimum bandwidth to flow
- Restrictions on changes in inter-packet spacing

Network layer service models (table)

Network Architecture	Service Model	Guarantees ?				Congestion feedback
		Bandwidth	Loss	Order	Timing	
Internet	best effort	none	no	no	no	no (inferred via loss)
ATM	CBR	constant rate	yes	yes	yes	no congestion
ATM	VBR	guaranteed rate	yes	yes	yes	no congestion
ATM	ABR	guaranteed minimum	no	yes	no	yes
ATM	UBR	none	no	yes	no	no

VC and datagram networks

Datagram network provides network-layer connectionless service
VC network provides network-layer connection service
Analogous to the transport-layer services, but the service is host-to host, and there isn't no choine, and the implementation is in the network core.

"source-to-dest path behaves much like telephone circuit"
- performance-wise
- network actions along source-to-dest path

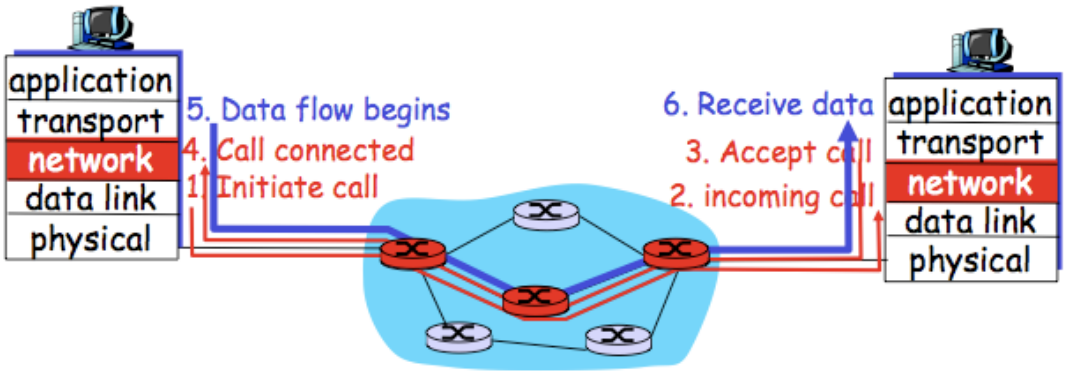
➢ call setup, teardown for each call *before* data can flow
➢ each packet carries VC identifier (not destination host address)
➢ *every* router on source-dest path maintains "state" for each passing connection
➢ link, router resources (bandwidth, buffers) may be *allocated* to VC (dedicated resources = predictable service)

VC implementation consist therefore

- Path from source to destination
- VC numbers, one number for each link along path
- Entries in forwarding tables in routers along path

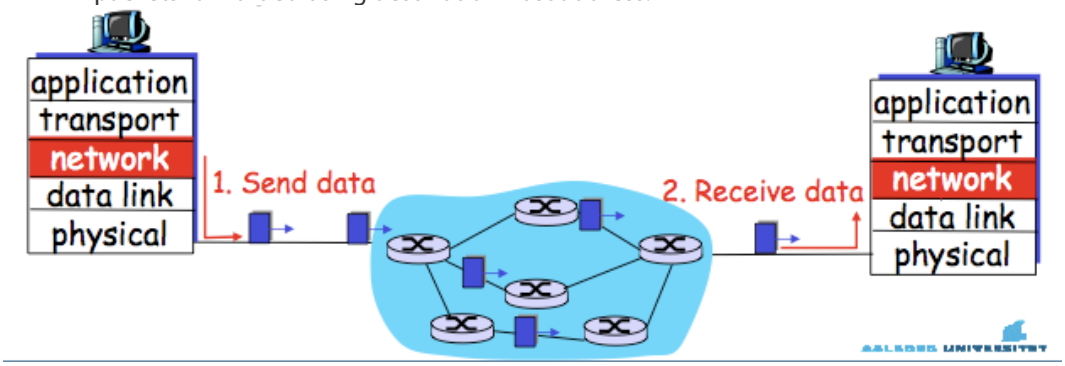
Example on VC

- Used to setup, maintain teardown VC, and used in ATM, frame-relay and not used in today's internet.



Example on Datagram networks

- used no call setup at network layer, and no network-level concept of "connection", and packets forwarded using destination host address.



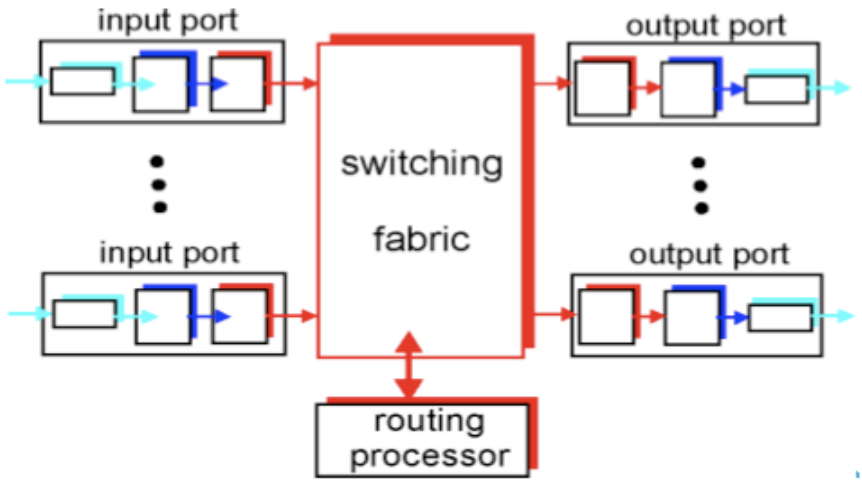
Datagram vs. VC network

Internet (datagram)
- ➢ data exchange among computers
  - "elastic" service, no strict timing req.
- ➢ "smart" end systems (computers)
  - can adapt, perform control, error recovery
  - simple inside network, complexity at "edge"
- ➢ many link types
  - different characteristics
  - uniform service difficult

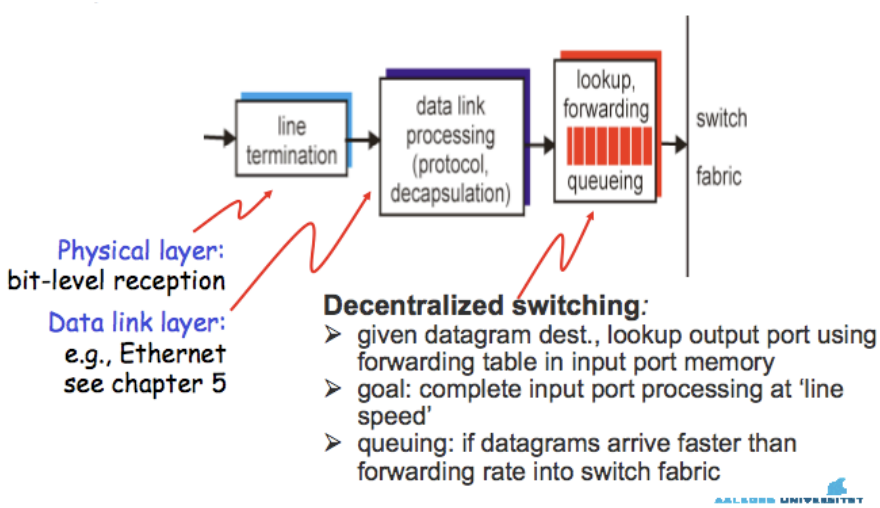
ATM (VC)
- ➢ evolved from telephony
- ➢ human conversation:
  - strict timing, reliability requirements
  - need for guaranteed service
- ➢ "dumb" end systems
  - telephones
  - complexity inside network

Router architecture

- it runs routering algorithms and protocols (RIP, OPSF, BGP)
- forwarding datagram's from incoming to outgoing link



Input port functions

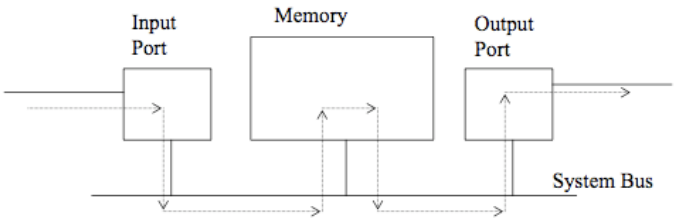


They can switch via **Memory**, **BUS** and **Crossbar**

Via Memory

First generation routers:

- traditional computers with switching under direct control of CPU
- packet copied to system's memory
- speed limited by memory bandwidth (2 bus crossings per datagram)



Via Bus

- datagram from input port memory to output port memory via a shared bus
- bus contention: switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access and enterprise routers

Via Crossbar / Interconnection network

overcome bus bandwidth limitations

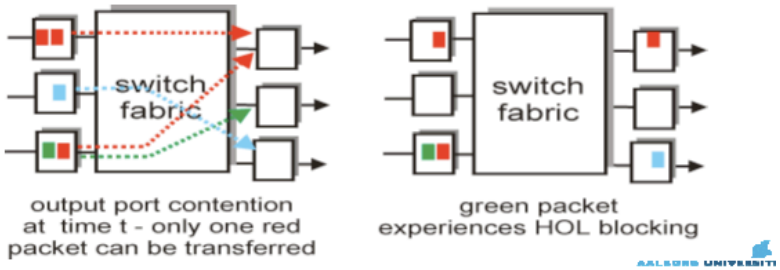
Banyan networks, other interconnection nets initially developed to connect processors in multiprocessor

advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.

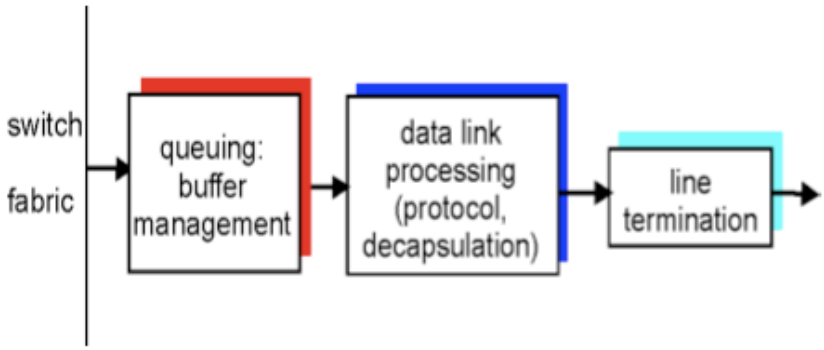
Cisco 12000: switches 60 Gbps through the interconnection network

Input port queueing

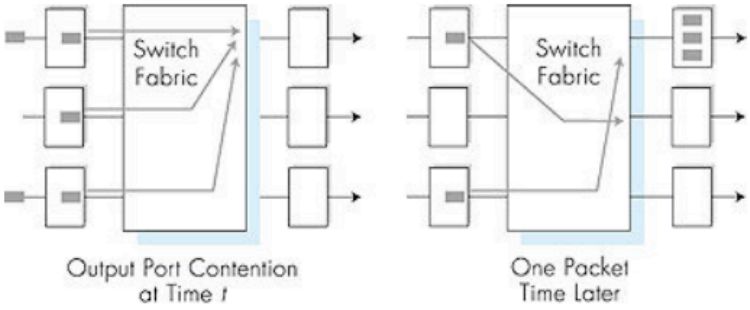
- Fabric slower than input ports combined -> queueing may occur at input queues
- Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward
- *queueing delay and loss due to input buffer overflow!*



Output port functions



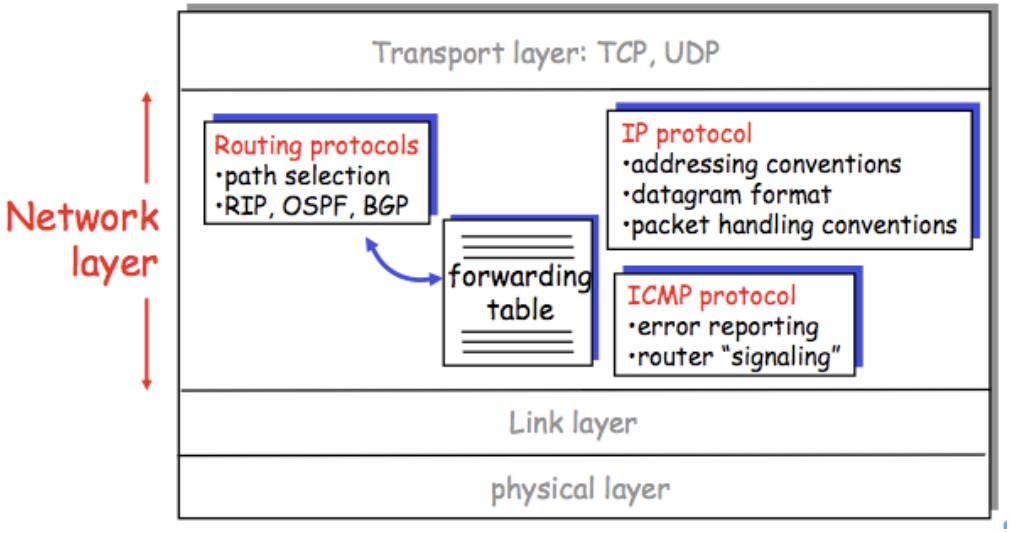
Output port queueing



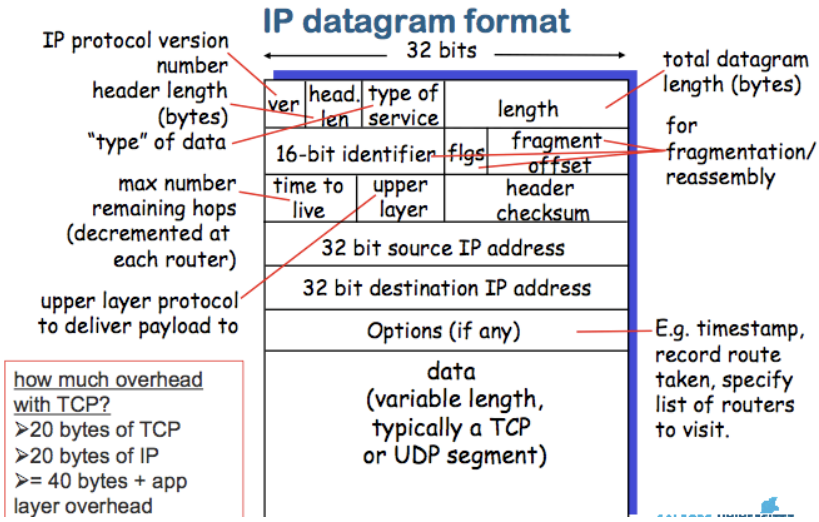
- buffering when arrival rate via switch exceeds output line speed
- *queueing (delay) and loss due to output port buffer overflow!*

IP: Internet Protocol

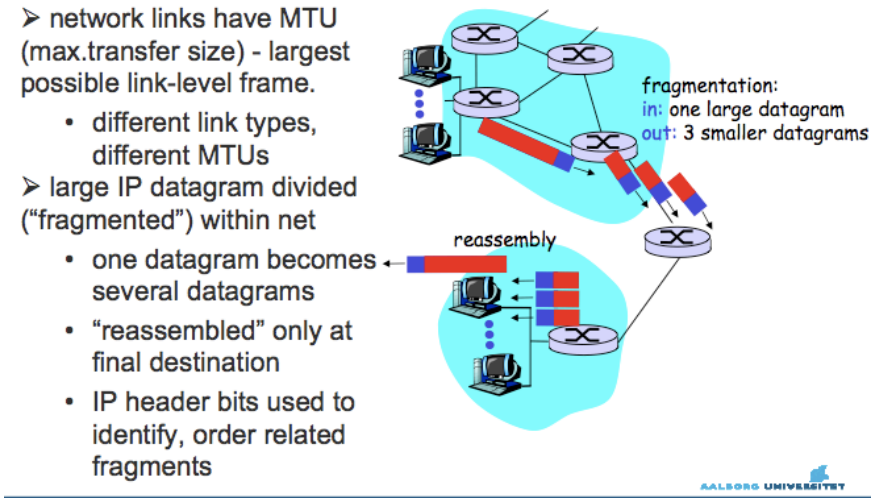
Host, router network layer functions



IP datagram format



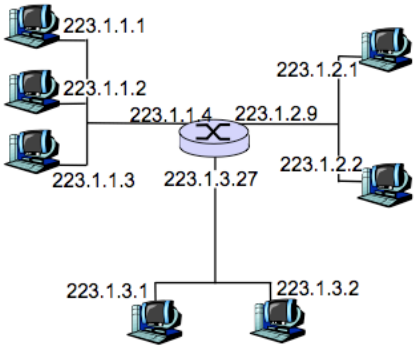
IP fragmentation & reassembly



IPV4

➢IP address: 32-bit identifier for host, router *interface*
➢*interface:* connection between host/router and physical link

- router's typically have multiple interfaces
- host typically has one interface
- IP addresses associated with each interface

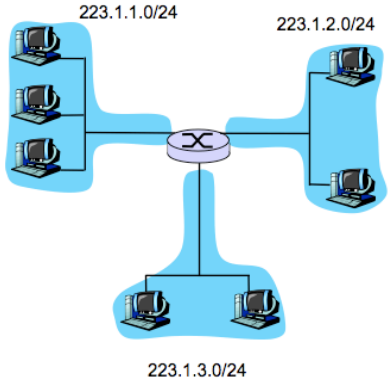


223.1.1.1 = 11011111 00000001 00000001 00000001

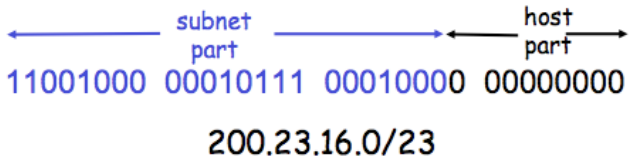
223 1 1 1

/Subnets

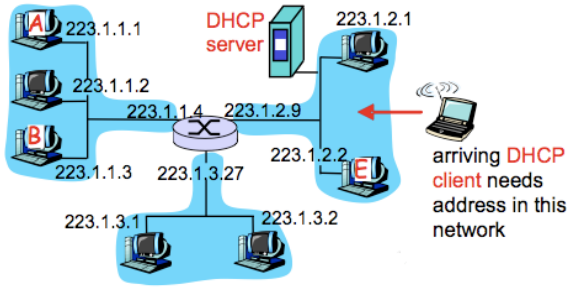
Recipe
To determine the subnets, detach each interface from its host or router, creating islands of isolated networks. Each isolated network is called a subnet.

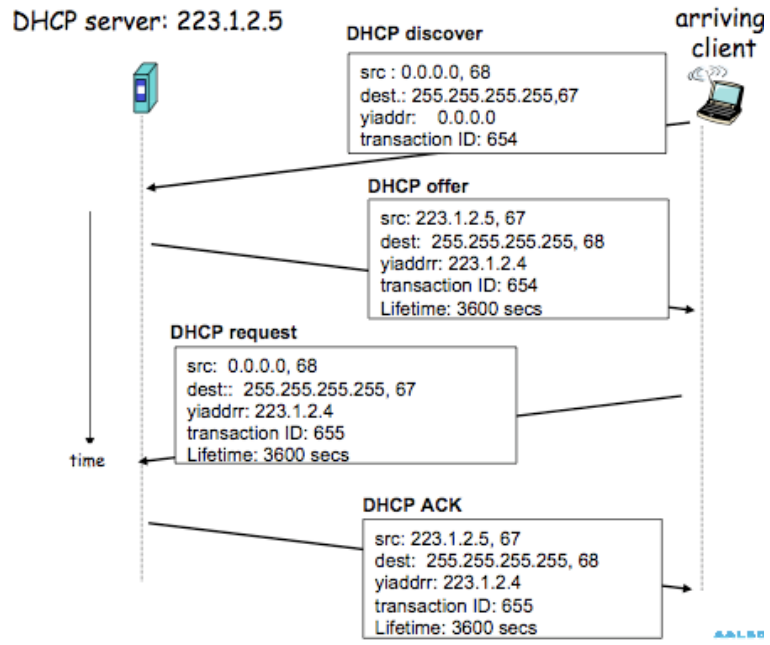


IP adresseing: CIDR

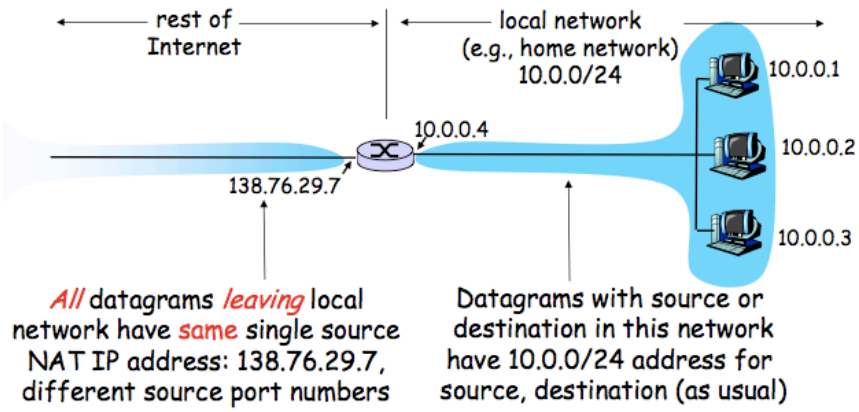


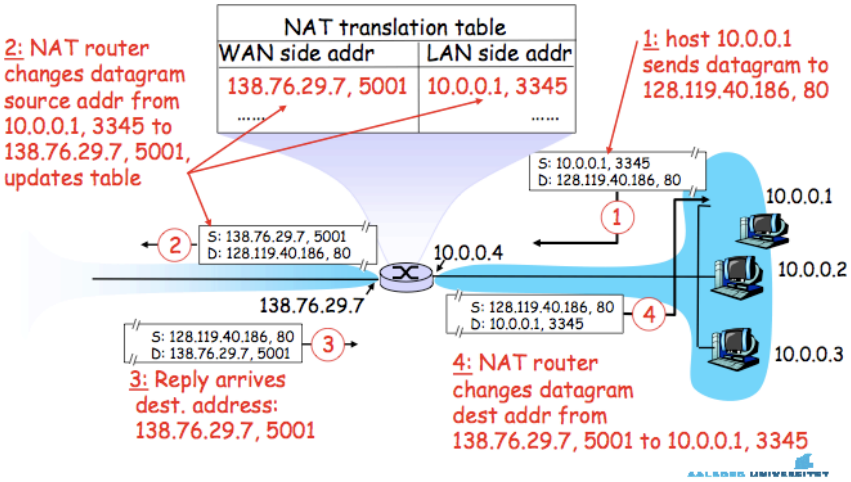
DHCP client-server





NAT: Network Address Translation

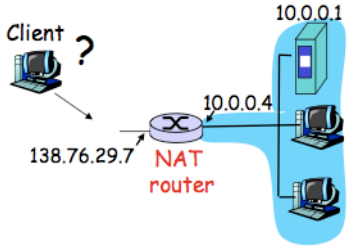




Traversal problems

I

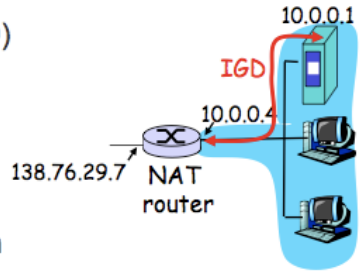
- client want to connect to server with address 10.0.0.1
  - server address 10.0.0.1 local to LAN (client can't use it as destination addr)
  - only one externally visible NATted address: 138.76.29.7
- solution 1: statically configure NAT to forward incoming connection requests at given port to server
  - e.g., (123.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000



II

- solution 2: Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATted host to:
  - learn public IP address (138.76.29.7)
  - enumerate existing port mappings
  - add/remove port mappings (with lease times)

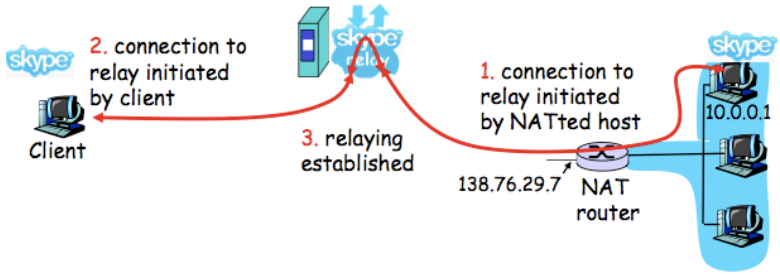
i.e., automate static NAT port map configuration



III

solution 3: relaying (used in Skype)

- NATed server establishes connection to relay
- External client connects to relay
- relay bridges packets between to connections



ICMP

- 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

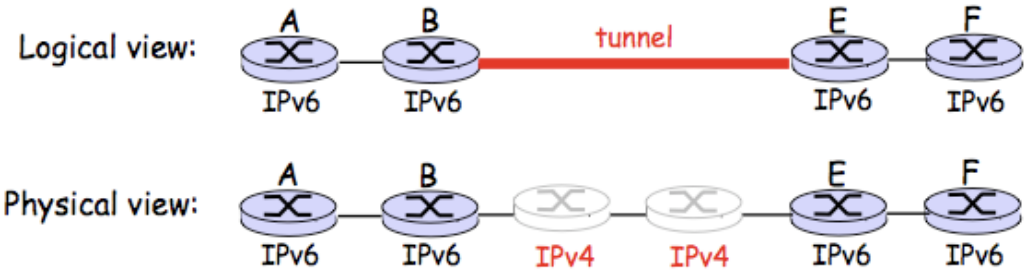
IPv6

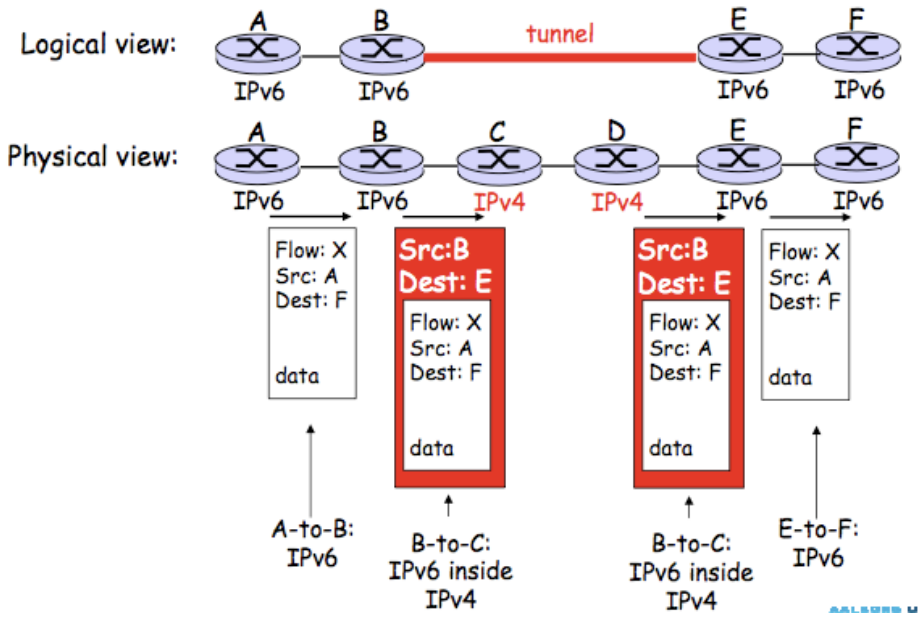
- Initial motivation: 32-bit address space soon to be completely allocated.
- Additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS

  IPv6 datagram format:
  - fixed-length 40 byte header
  - no fragmentation allowed

- *Checksum*: removed entirely to reduce processing time at each hop
- *Options:* allowed, but outside of header, indicated by "Next Header" field
- *ICMPv6:* new version of ICMP
  - additional message types, e.g. "Packet Too Big"
  - multicast group management functions

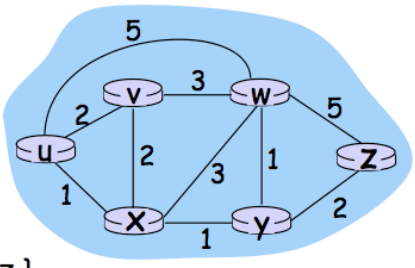
Tunneling


Logical view:
A
B
tunnel
E
F
IPv6
IPv6
IPv6
IPv6
Physical view:
A
B
E
F
IPv6
IPv6
IPv4
IPv4
IPv6
IPv6


Logical view:
A
B
tunnel
E
F
IPv6
IPv6
IPv6
IPv6
Physical view:
A
B
C
D
E
F
IPv6
IPv6
IPv4
IPv4
IPv6
IPv6
Flow: X
Src: A
Dest: F
data
Src:B
Dest: E
Flow: X
Src: A
Dest: F
data
Src:B
Dest: E
Flow: X
Src: A
Dest: F
data
Flow: X
Src: A
Dest: F
data
A-to-B:
IPv6
B-to-C:
IPv6 inside
IPv4
B-to-C:
IPv6 inside
IPv4
E-to-F:
IPv6

12. Network Layer II

The network is like graphs, where we have N to be the routers and the edges will be the links.



Graph: G = (N,E)

N = set of routers = { 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) }

Definition: C(x,x') = cost of link (x,x')
Example: C(w,z)=5.

Routing algorithm: Algorithm that finds least-cost pth.

Routing Algorithms

- Link State

Dijkstra algorithm

net topology, link costs known to all nodes

- accomplished via "link state broadcast"
- all nodes have same info

computes least cost paths from one node ('source") to all other nodes

- gives forwarding table for that node

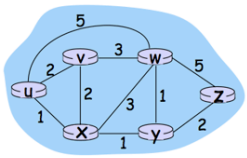
iterative: after k iterations, know least cost path to k dest.'s

Notation:
c(x,y): link cost from node x to y; = ∞ if not direct neighbors
D(v): current value of cost of path from source to dest. v
p(v): predecessor node along path from source to v
N': set of nodes whose least cost path definitively known

```
1  Initialization:
2   N' = {u}
3   for all nodes v
4     if v adjacent to u
5         then D(v) = c(u,v)
6     else D(v) = ∞
7
8  Loop
9    find w not in N' such that D(w) is a minimum
10   add w to N'
11   update D(v) for all v adjacent to w and not in N' :
12      D(v) = min( D(v), D(w) + c(w,v) )
13   /* new cost to v is either old cost to v or known
14    shortest path cost to w plus cost from w to v */
15 until all nodes in N'
```

Example

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



Complexity

Algorithm complexity: n nodes
each iteration: need to check all nodes, w, not in N
$n(n+1)/2$ comparisons: $O(n^2)$
more efficient implementations possible: O(nlogn)

Oscillations possible:
e.g., link cost = amount of carried traffic

Distance Vector

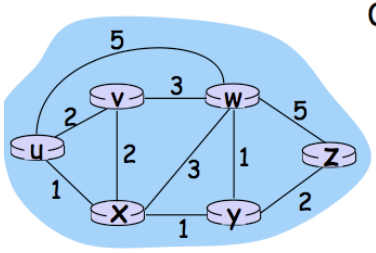
Next hop in the shortest path – forwarding table

Bellman-Ford Equation (dynamic programming)

Definition: $d_x(y)$:= cost of least-cost path from x to y.

Then: $d_x(y)=\min_v c(x,v) + d_v(y)$ where $\min_v$ is taken over all neighbors v of x.

Example



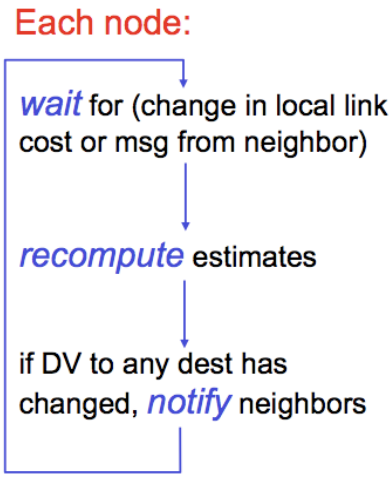
Clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$d_u(z) = \min \{ c(u,v) + d_v(z), \; c(u,x) + d_x(z), \; c(u,w) + d_w(z) \}$$
$$= \min \{2 + 5, \; 1 + 3, \; 5 + 3\} = 4$$

Node that achieves minimum is next hop in shortest path ➜ forwarding table

The 3 steps



Link State vs. Distance Vector

Message complexity
LS: with n nodes, E links, O(nE) msgs sent
DV: exchange between neighbors only
- convergence time varies

Speed of Convergence
LS: $O(n^2)$ algorithm requires O(nE) msgs
- may have oscillations

DV: convergence time varies
- may be routing loops
- count-to-infinity problem

Robustness: what happens if router malfunctions?
LS:
- node can advertise incorrect *link* cost
- each node computes only its *own* table

DV:
- DV node can advertise incorrect *path* cost
- each node's table used by others
  - error propagate thru network

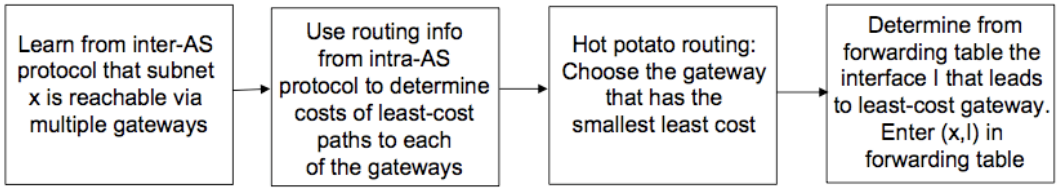
Hierarchical routing

Our routing study thus far - idealization
all routers identical
network "flat"
… *not* true in practice

scale: with 200 million destinations:
can't store all dest's in routing tables!
routing table exchange would swamp links!

administrative autonomy
internet = network of networks
each network admin may want to control routing in its own network

- now suppose AS1 learns from inter-AS protocol that subnet *x* is reachable from AS3 *and* from AS2.
- to configure forwarding table, router 1d must determine towards which gateway it should forward packets for dest x.
  - this is also job of inter-AS routing protocol!
- hot potato routing: send packet towards closest of two routers.



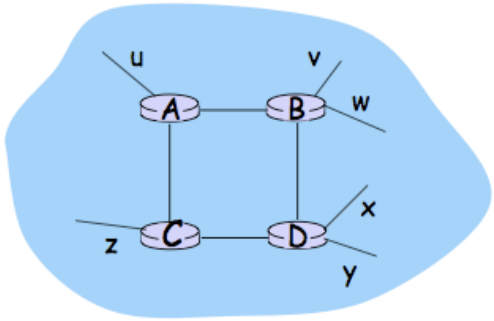
Routing in the Internet

most common Intra-AS routing protocols:

- RIP: Routing Information Protocol
- OSPF: Open Shortest Path First
- IGRP: Interior Gateway Routing Protocol (Cisco proprietary)

RIP

distance vector algorithm
included in BSD-UNIX Distribution in 1982
distance metric: # of hops (max = 15 hops)



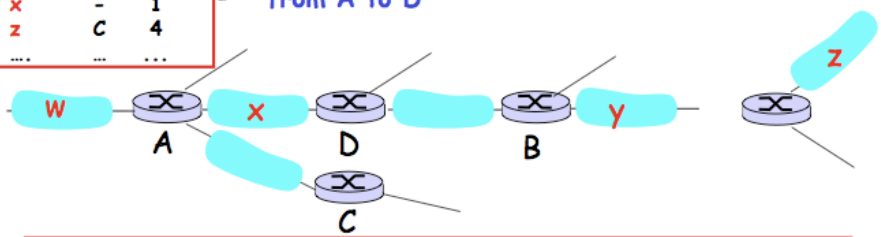
From router A to subsets:

destination	hops
u	1
v	2
w	2
x	3
y	3
z	2

Example

Dest	Next	hops
w	-	1
x	-	1
z	C	4
....	...	...

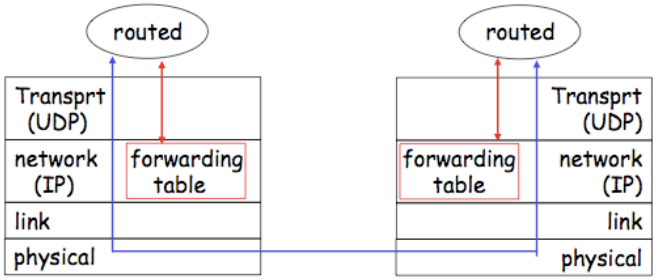
Advertisement from A to D



Destination Network	Next Router	Num. of hops to dest.
w	A	2
y	B	2
z	~~B~~ A	~~7~~ 5
x	--	1
....	....	....

Table processing

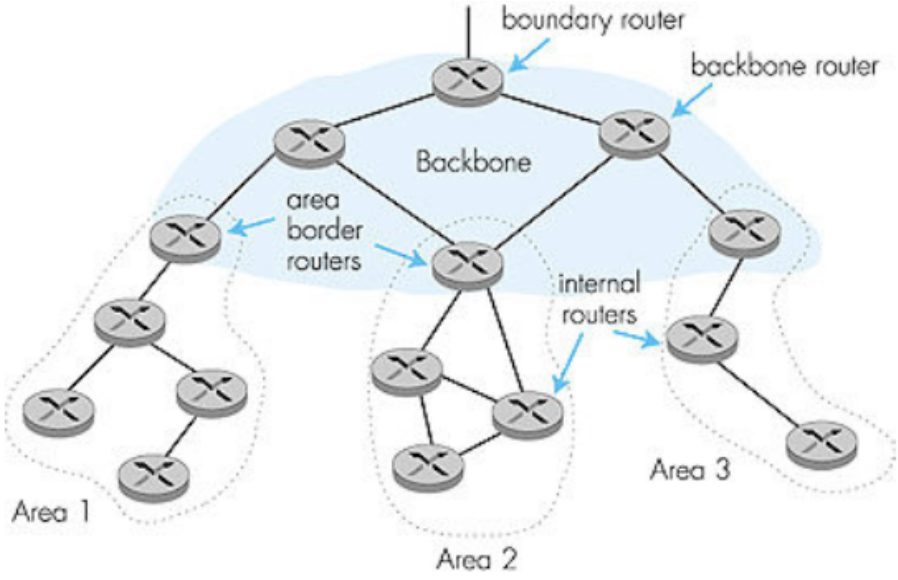
RIP routing tables managed by **application-level** process called route-d (daemon)
advertisements sent in UDP packets, periodically repeated



OPSF

- "open": publicly available
- uses Link State algorithm
  - LS packet dissemination
  - topology map at each node
  - route computation using Dijkstra's algorithm
- OSPF advertisement carries one entry per neighbor router
- advertisements disseminated to entire AS (via flooding)
  - carried in OSPF messages directly over IP (rather than TCP or UDP

Example



two-level hierarchy: local area, backbone.

- Link-state advertisements only in area
- each nodes has detailed area topology; only know direction (shortest path) to nets in other areas.

area border routers: "summarize" distances to nets in own area, advertise to other Area Border routers.
backbone routers: run OSPF routing limited to backbone.
boundary routers: connect to other AS's.

BGP

BGP (Border Gateway Protocol): *the* de facto standard
BGP provides each AS a means to:

1. Obtain subnet reachability information from neighboring ASs.
2. Propagate reachability information to all AS-internal routers.
3. Determine "good" routes to subnets based on reachability information and policy.

allows subnet to advertise its existence to rest of Internet: *"I am here"*

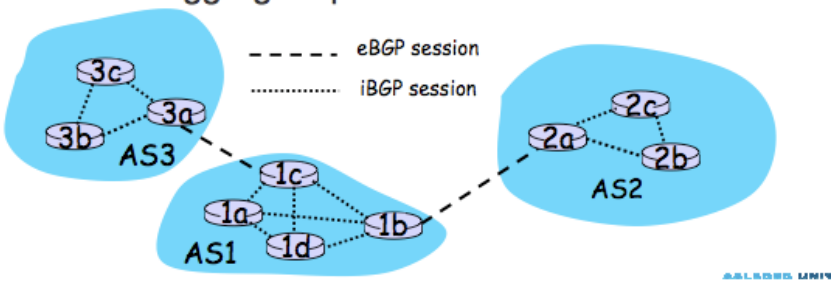
Example

pairs of routers (BGP peers) exchange routing info over semi-permanent TCP connections: BGP sessions

- BGP sessions need not correspond to physical links.

when AS2 advertises prefix to AS1:

- AS2 *promises* it will forward any addresses datagrams towards that prefix.
- AS2 can aggregate prefixes in its advertisement



Messages

BGP messages exchanged using TCP.
BGP messages:

- OPEN: opens TCP connection to peer and authenticates sender
- UPDATE: advertises new path (or withdraws old)
- KEEPALIVE keeps connection alive in absence of UPDATES; also ACKs OPEN request
- NOTIFICATION: reports errors in previous msg; also used to close connection

Intra vs. Inter

Policy:

- Inter-AS: admin wants control over how its traffic routed, who routes through its net.
- Intra-AS: single admin, so no policy decisions needed

Scale:

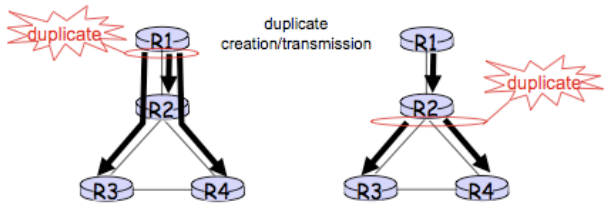
- hierarchical routing saves table size, reduced update traffic

Performance:

- Intra-AS: can focus on performance
- Inter-AS: policy may dominate over performance

Broadcast and multicast routing

deliver packets from source to all other nodes
source duplication is inefficient:



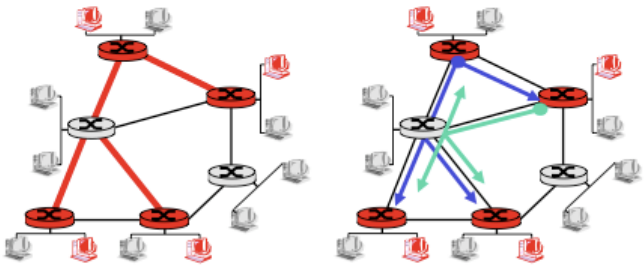
source duplication | in-network duplication

source duplication: how does source determine recipient addresses?

Goal

Goal: find a tree (or trees) connecting routers having local mcast group members

- *tree:* not all paths between routers used
- *source-based:* different tree from each sender to rcvrs
- *shared-tree:* same tree used by all group members



Shared tree | Source-based trees

DVMRP: distance vector multicast routing protocol, RFC1075
flood and prune: reverse path forwarding, source-based tree

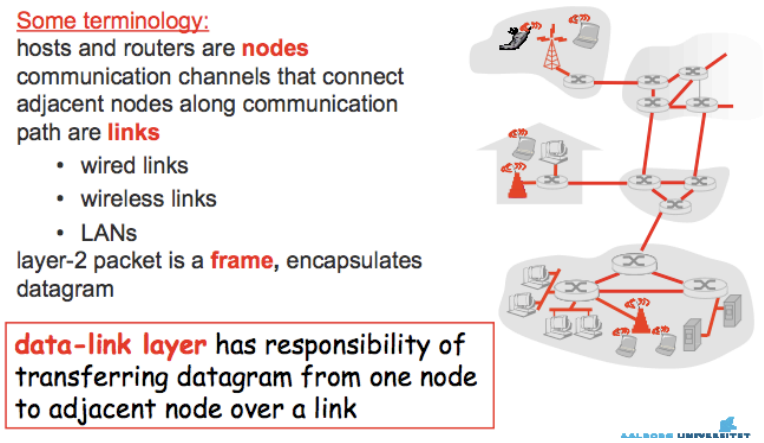
- RPF tree based on DVMRP's own routing tables constructed by communicating DVMRP routers
- no assumptions about underlying unicast
- initial datagram to mcast group flooded everywhere via RPF
- routers not wanting group: send upstream prune msgs

13. Link Layer I

Services provided by the link layer

2 different type of link layers

- First type consist broadcasting channels, such as LAN, W-LAN, HFC and satellite networks.
- The second type consist the point-to-point communication link, such as between routers or between residential dial-up modem and ISP router.



Example on the link layer:

- Tourist (datagram)
- The 3 tranportation segments (communication link)
- Tranportation mode (link layer protocol)
- Travel agent (routing protocol)

Link layer protocol includes:

- Framing (data field which include datagram)
- Link access (coordinating the frame transmission)
- Reliable delivery (guarentees transmission without error)
- Flow control (control the buffering flow)
- Error detection
- Error correction
- Half-duplex and full-duplex (Half is sending and receiving on same time, and full is sending it all on one time)

framing, link access:

- encapsulate datagram into frame, adding header, trailer
- channel access if shared medium
- "MAC" addresses used in frame headers to identify source, dest
  - different from IP address!

reliable delivery between adjacent nodes

- we learned how to do this already (chapter 3)!
- 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.
- receiver detects presence of errors:
  - signals sender for retransmission or drops frame

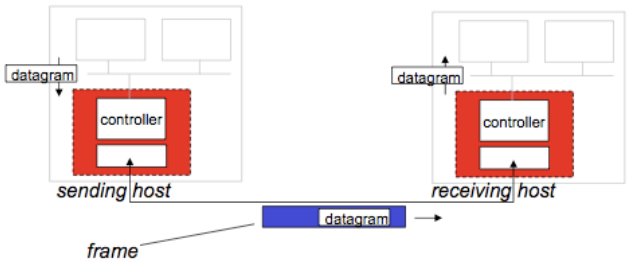
error correction:

- 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

Adaptors communicating



sending side:

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

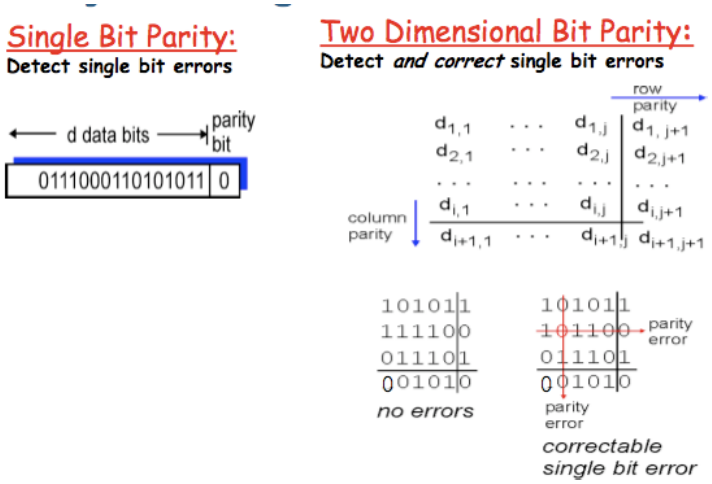
receiving side

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

Error detection and correction techniques

3 detecting techniques

- Parity checks (basic idead behind error detection and correction)



- Checksumming methods (more typically employed in tranport layer)

Goal: detect "errors" (e.g., flipped bits) in transmitted packet (note: used at transport layer *only*)

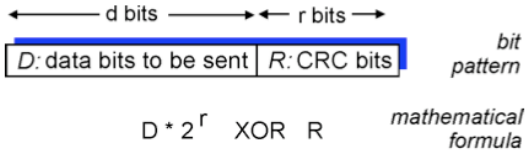
Sender:
- treat segment contents as sequence of 16-bit integers
- checksum: addition (1's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver:
- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?*

- Cyclic redundancy checks (more typically employed in the link layer in the adapters)

view data bits, D, as a binary number
choose r+1 bit pattern (generator), G
goal: choose r CRC bits, R, such that
- <D,R> exactly divisible by G (modulo 2)
- receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
- can detect all burst errors less than r+1 bits

widely used in practice (802.11 WiFi, ATM)



$D * 2^r$ XOR R — *mathematical formula*

Multiple access protocols

- 802.11 wireless LAN
- upstream HFC
- old-fashioned Ethernet

broadcast (shared wire or medium)

- point-to-point link between Ethernet switch and host
- PPP for dial-up access

point-to-point

A point to point link consist a single sender at one end of the link and a single receiver at the other end of the link,
A broadcast link can have multiple sending and receiving nodes all connected to the same, single, shared broadcast channel. (Ethernet and LANs are examples on broadcast link layer technologies)

- single shared broadcast channel
- two or more simultaneous transmissions by nodes: interference
  - collision if node receives two or more signals at the same time

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

Multiple access protocols, are protocols which regulate their transmission onto the share broadcast channel.

- Channel partitioning protocols
- Random access protocols
- Taking-turns protocols

MAC Address

- Mac address on each adapter in the whole world with in all 2^{24} has different addresses.
- Each node on a LAN has an IP address and each node's adapter has a MAC address.

Channel Partitioning

- divide channel into smaller "pieces" (time slots, frequency, code)
- 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

Random Access Protocol

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)

Examples of random access MAC protocols:

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

LAN Technologies

Data link layer so far:

- services, error detection/correction, multiple access

Next: LAN technologies

- addressing
- Ethernet
- switches
- PPP

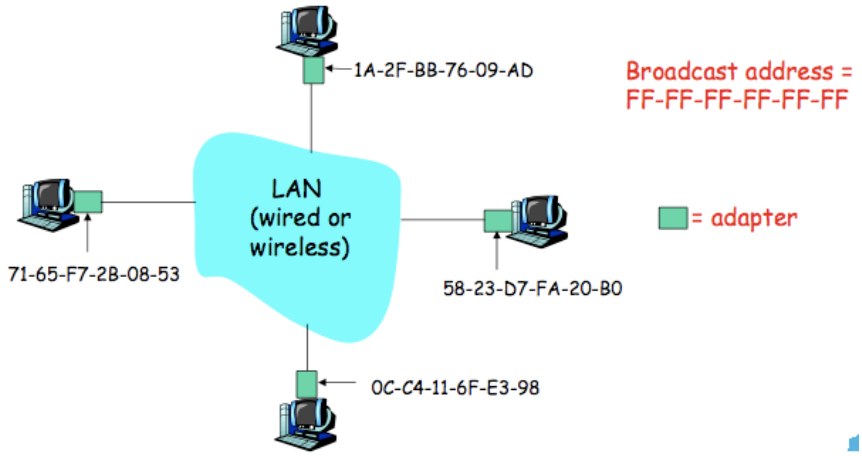
Link layer addressing

32-bit IP address:

- *network-layer* address
- used to get datagram to destination IP subnet

MAC (or LAN or physical or Ethernet) address:

- function: *get frame from one interface to another physically-connected interface (same network)*
- 48 bit MAC address (for most LANs)
  - burned in NIC ROM, also sometimes software settable



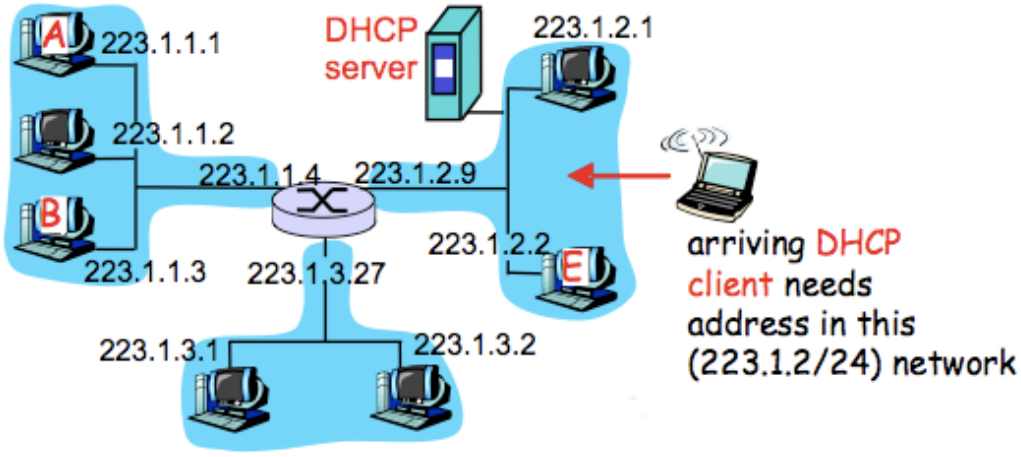
DHCP

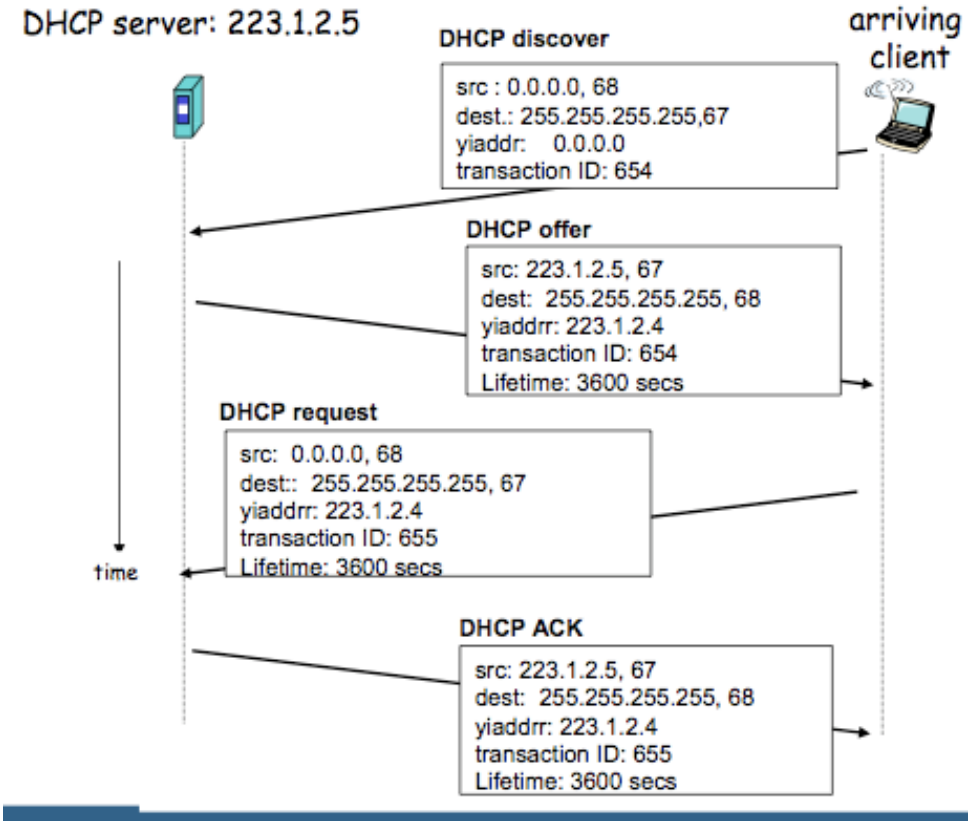
Goal: allow host to *dynamically* obtain its IP address from network server when joining network

- support for mobile users joining network
- host holds address only while connected and "on" (allowing address reuse)
- renew address already in use

➢ DHCP overview:

- 1. host broadcasts "DHCP discover" msg
- 2. DHCP server responds with "DHCP offer" msg
- 3. host requests IP address: "DHCP request" msg
- 4. DHCP server sends address: "DHCP ack" msg

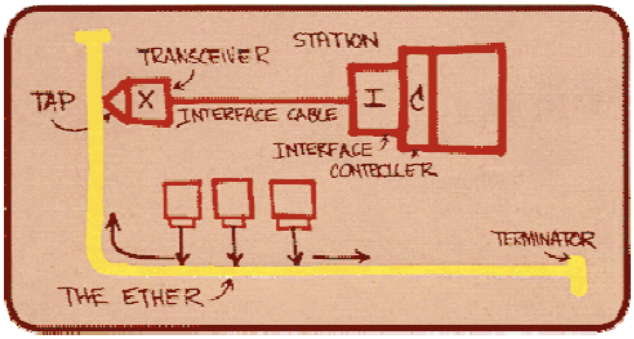

A
223.1.1.1
DHCP
server
223.1.2.1
223.1.1.2
223.1.1.4
223.1.2.9
B
223.1.2.2
E
223.1.1.3
223.1.3.27
223.1.3.1
223.1.3.2
arriving DHCP
client needs
address in this
(223.1.2/24) network


DHCP server: 223.1.2.5
arriving
client
DHCP discover
src : 0.0.0.0, 68
dest.: 255.255.255.255,67
yiaddr: 0.0.0.0
transaction ID: 654
DHCP offer
src: 223.1.2.5, 67
dest: 255.255.255.255, 68
yiaddrr: 223.1.2.4
transaction ID: 654
Lifetime: 3600 secs
DHCP request
src: 0.0.0.0, 68
dest:: 255.255.255.255, 67
yiaddrr: 223.1.2.4
transaction ID: 655
Lifetime: 3600 secs
time
DHCP ACK
src: 223.1.2.5, 67
dest: 255.255.255.255, 68
yiaddrr: 223.1.2.4
transaction ID: 655
Lifetime: 3600 secs

14. Link Layer II

Ethernet

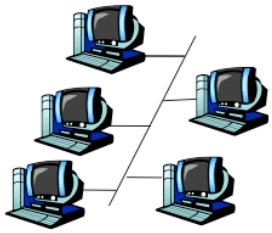
"dominant" wired LAN technology:
- cheap $20 for NIC
- first widely used LAN technology
- simpler, cheaper than token LANs and ATM
- kept up with speed race: 10 Mbps – 10 Gbps



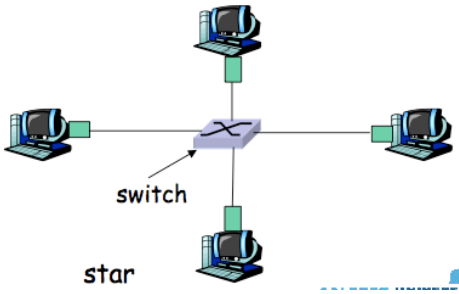
Metcalfe's Ethernet sketch

bus topology popular through mid 90s
- all nodes in same collision domain (can collide with each other)

today: star topology prevails
- active *switch* in center
- each "spoke" runs a (separate) Ethernet protocol (nodes do not collide with each other)

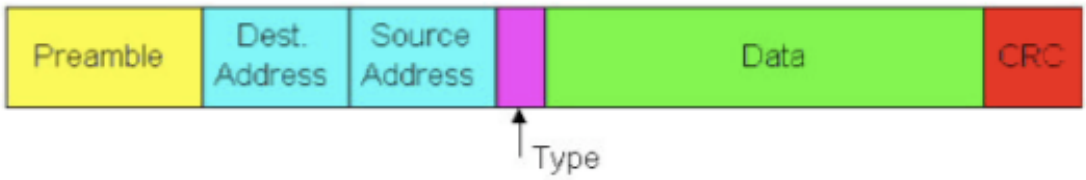


bus: coaxial cable



star

Frame structure



It has a preamble, Addresses, Type, CRC and of course the data.

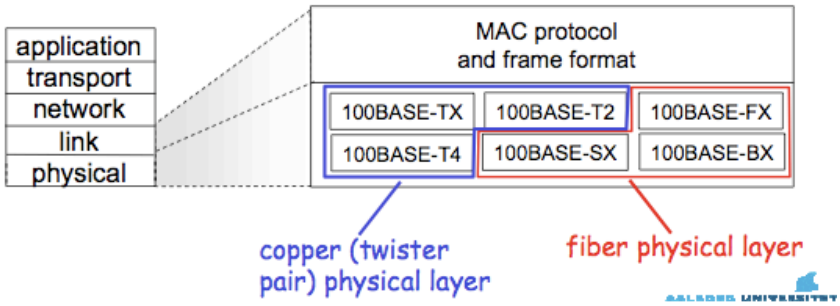
Ethernet CSMA/CD algorithm

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 and sends jam signal
5. After aborting, NIC enters **exponential backoff**: after *m*th collision, NIC chooses *K* at random from {0,1,2,…,2^m-1}. NIC waits K·512 bit times, returns to Step 2

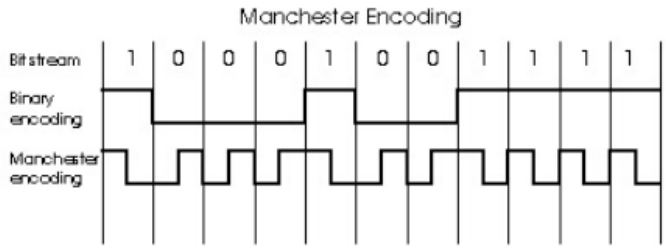
802.3 Ethernet Standards

many different Ethernet standards

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



Manchester Encoding



used in 10BaseT
each bit has a transition
allows clocks in sending and receiving nodes to synchronize to each other

- no need for a centralized, global clock among nodes!

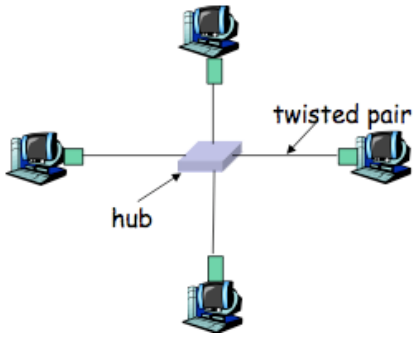
Hey, this is physical-layer stuff!

Link layer Switches

Hubs

physical-layer ("dumb") repeaters:

- bits coming in one link go out *all* other links at same rate
- all nodes connected to hub can collide with one another
- no frame buffering
- no CSMA/CD at hub: host NICs detect collisions



Switch

link-layer device: smarter than hubs, take *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 on segment, uses CSMA/CD to access segment

transparent

- hosts are unaware of presence of switches

plug-and-play, self-learning

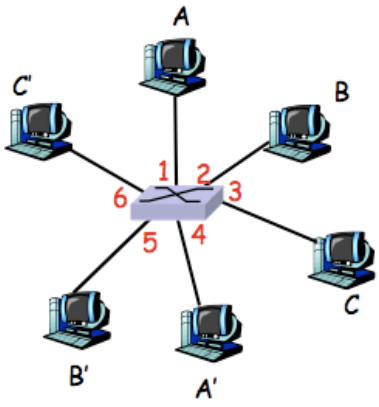
- switches do not need to be configured

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' simultaneously, without collisions

- not possible with dumb hub



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

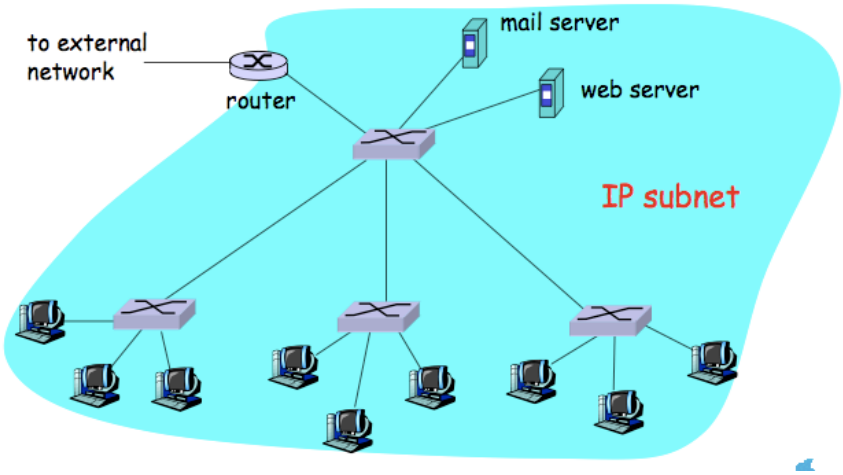
Switch frame : filtering / forwarding

When frame received:

1. record link associated with sending host
2. index switch table using MAC dest address
3. **if** entry found for destination
   then {
   if dest on segment from which frame arrived
   then drop the frame
   else forward the frame on interface indicated
   }
   else flood

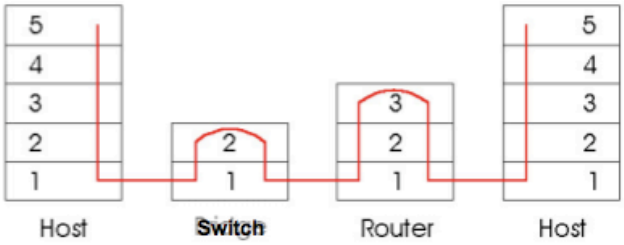
forward on all but the interface on which the frame arrived

Institutional network



Switches vs. Routers

- both store-and-forward devices
  - routers: network layer devices (examine network layer headers)
  - switches are link layer devices
- routers maintain routing tables, implement routing algorithms
- switches maintain switch tables, implement filtering, learning algorithms



AALBORG UNIVERSITET

Summary comparison

	hubs	routers	switches
traffic isolation	no	yes	yes
plug & play	yes	no	yes
optimal routing	no	yes	no
cut through	yes	no	yes

AALBORG UNIVERSITET

PPP

Point to point data link control

one sender, one receiver, one link: easier than broadcast link:

- no Media Access Control
- no need for explicit MAC addressing
- e.g., dialup link, ISDN line

popular point-to-point DLC protocols:

- PPP (point-to-point protocol)
- HDLC: High level data link control (Data link used to be considered "high layer" in protocol stack!

PPP Design requirements

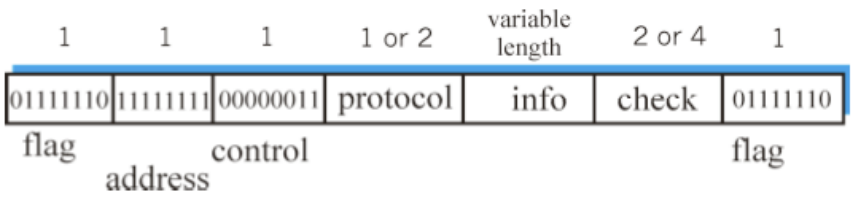
- Packet framing
- Bit transparency
- Error detection
- Connection liveness
- Network layer address negotiation

Error recovery , flow control, data re-ordering, all relegated to higher layers.

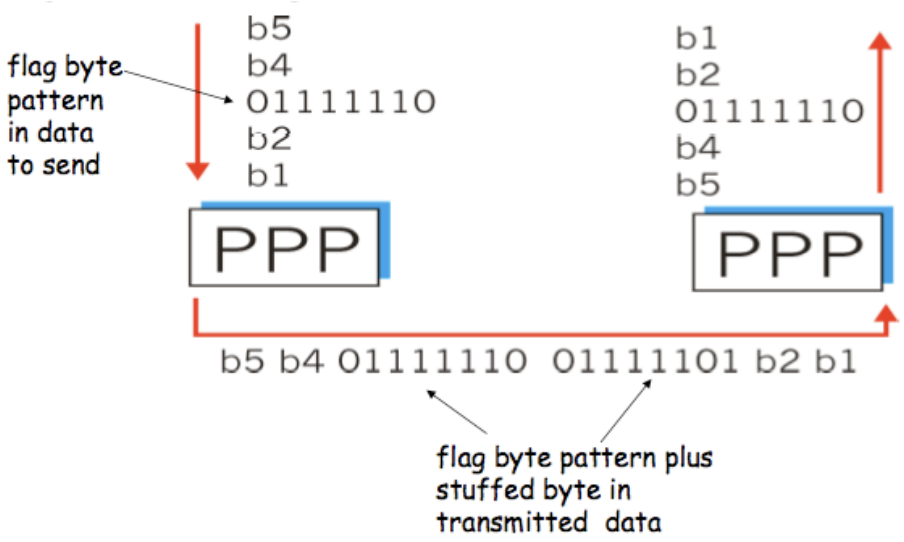
PPP Data frame

Flag: delimiter (framing)
Address: does nothing (only one option)
Control: does nothing; in the future possible multiple control fields
Protocol: upper layer protocol to which frame delivered (eg, PPP-LCP, IP, IPCP, etc)

info: upper layer data being carried
check: cyclic redundancy check for error detection



Byte Stuffing



Link virtualization – ATM and MPLS

Virtual of networks

Virtualization of resources: powerful abstraction in systems engineering:

- computing examples: virtual memory, virtual devices
  - Virtual machines: e.g., java
  - IBM VM os from 1960's/70's
- layering of abstractions: don't sweat the details of the lower layer, only deal with lower layers abstractly

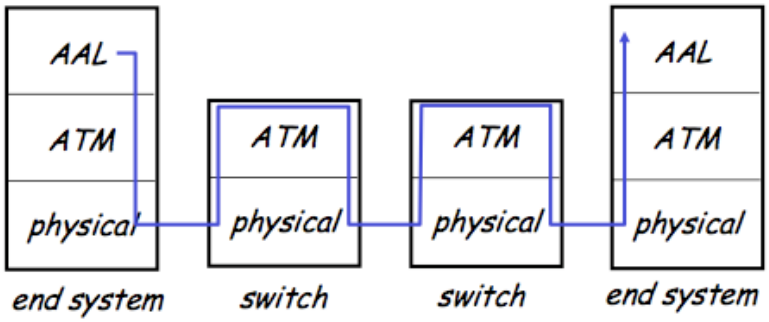
ATM and MPLS

ATM, MPLS separate networks in their own right
- different service models, addressing, routing from Internet

viewed by Internet as logical link connecting IP routers
- just like dialup link is really part of separate network (telephone network)

ATM, MPLS: of technical interest in their own right

ATM layers



- Adaption layer

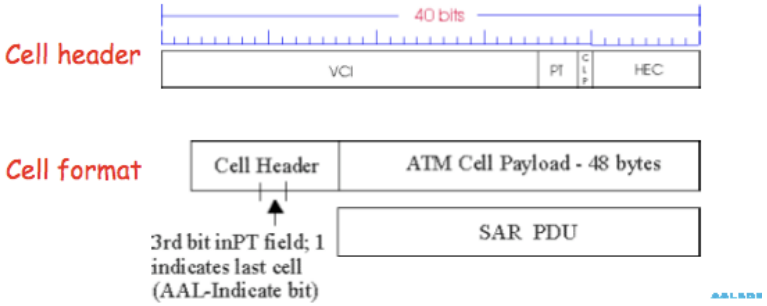
  ATM **Adaptation Layer** (AAL): "adapts" upper layers (IP or native ATM applications) to ATM layer below
  AAL present **only in end systems**, not in switches
  AAL layer segment (header/trailer fields, data) fragmented across multiple ATM cells
  - analogy: TCP segment in many IP packets

- ATM layer

  Service: transport cells across ATM network
  - ➢ analogous to IP network layer
  - ➢ very different services than IP network layer

Network Architecture	Service Model	Guarantees ?				Congestion feedback
		Bandwidth	Loss	Order	Timing	
Internet	best effort	none	no	no	no	no (inferred via loss)
ATM	CBR	constant rate	yes	yes	yes	no congestion
ATM	VBR	guaranteed rate	yes	yes	yes	no congestion
ATM	ABR	guaranteed minimum	no	yes	no	yes
ATM	UBR	none	no	yes	no	no

- 5-byte ATM cell header
- 48-byte payload
  - Why?: small payload -> short cell-creation delay for digitized voice
  - halfway between 32 and 64 (compromise!)



- Physical layer

Physical Medium Dependent (PMD) sublayer

- **SONET/SDH**: transmission frame structure (like a container carrying bits);
  - bit synchronization;
  - bandwidth partitions (TDM);
  - several speeds: OC3 = 155.52 Mbps; OC12 = 622.08 Mbps; OC48 = 2.45 Gbps, OC192 = 9.6 Gbps
- **TI/T3**: transmission frame structure (old telephone hierarchy): 1.5 Mbps/ 45 Mbps
- **unstructured**: just cells (busy/idle)

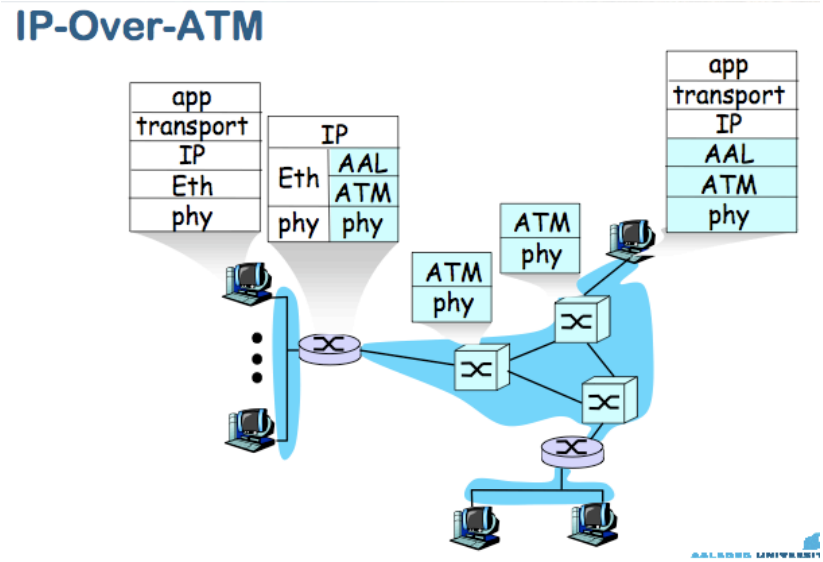
Two pieces (sublayers) of physical layer:

- Transmission Convergence Sublayer (TCS): adapts ATM layer above to PMD sublayer below
- Physical Medium Dependent: depends on physical medium being used

TCS Functions:

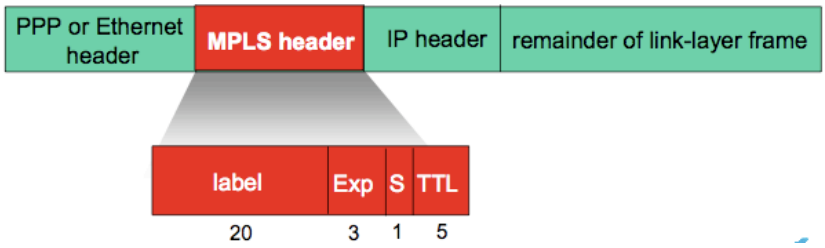
- Header **checksum** generation: 8 bits CRC
- Cell **delineation**
- With "unstructured" PMD sublayer, transmission of **idle cells** when no data cells to send

IP over ATM

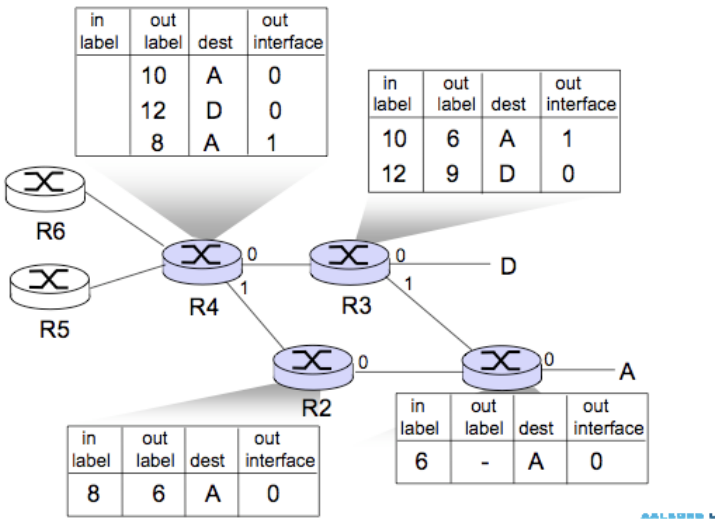


MPLS

- initial goal: speed up IP forwarding by using fixed length label (instead of IP address) to do forwarding
  - borrowing ideas from Virtual Circuit (VC) approach
  - but IP datagram still keeps IP address!

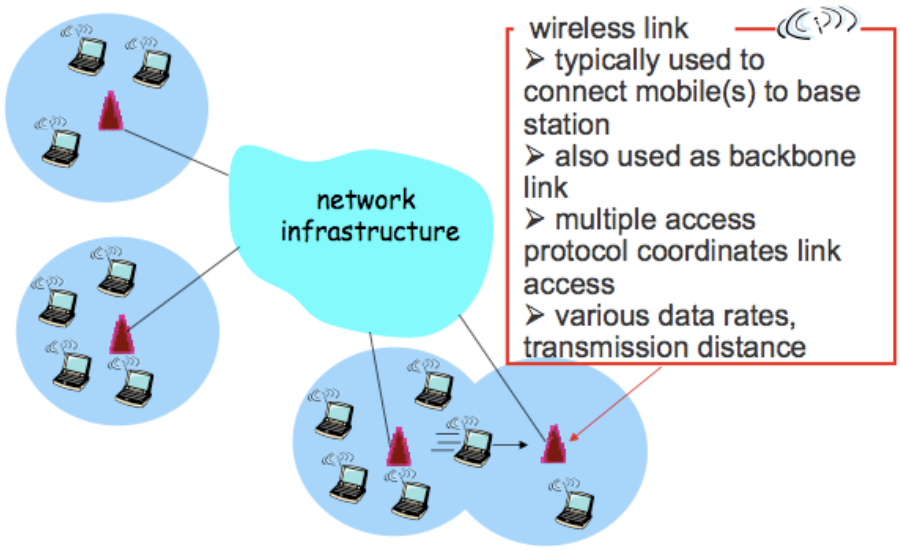


MPLS forwarding tables

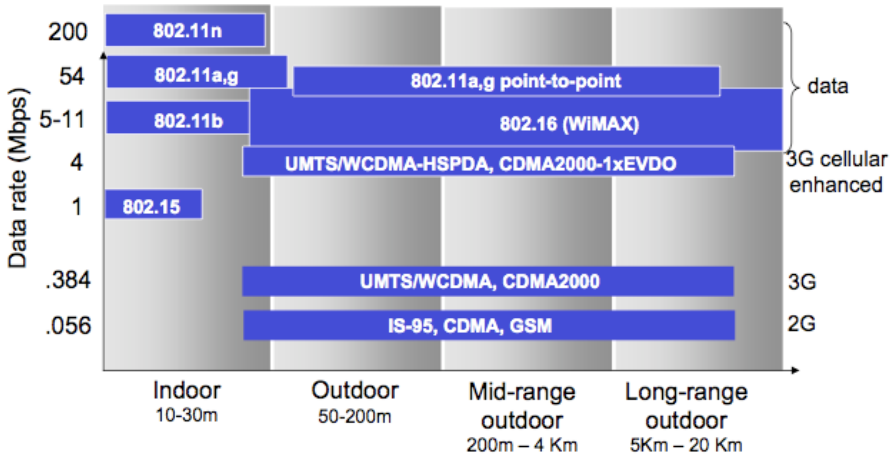


15. Wireless & Mobile Networks

Introduction



Wireless link standards



Networks taxanomy

	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 (802.11 Wireless LAN's (Wi-fi))

The different standards in Wireless

802.11b
- 2.4-5 GHz unlicensed spectrum
- up to 11 Mbps
- direct sequence spread spectrum (DSSS) in physical layer
  - all hosts use same chipping code

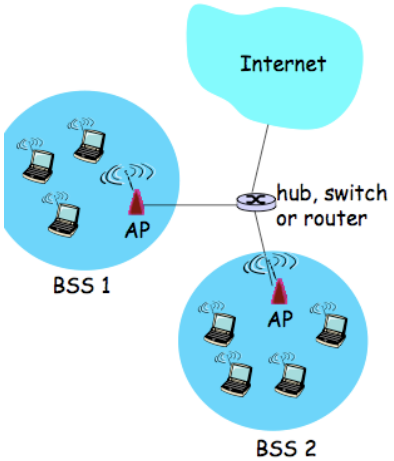
➢ 802.11a
- 5-6 GHz range
- up to 54 Mbps

➢ 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

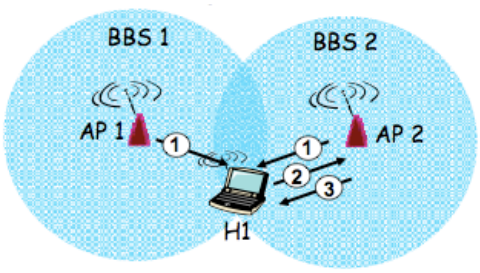
Architecture of 802.11 LAN

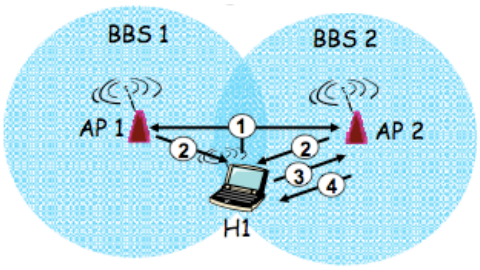


➢ wireless host communicates with base station
- base station = access point (AP)

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

Passive/Active





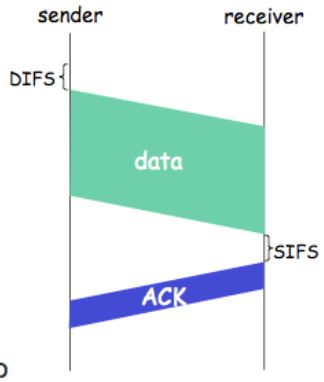
Passive Scanning:
(1) beacon frames sent from APs
(2) association Request frame sent: H1 to selected AP
(3) association Response frame sent: H1 to selected AP

Active Scanning:
(1) Probe Request frame broadcast from H1
(2) Probes response frame sent from APs
(3) Association Request frame sent: H1 to selected AP
(4) Association Response frame sent: H1 to selected AP

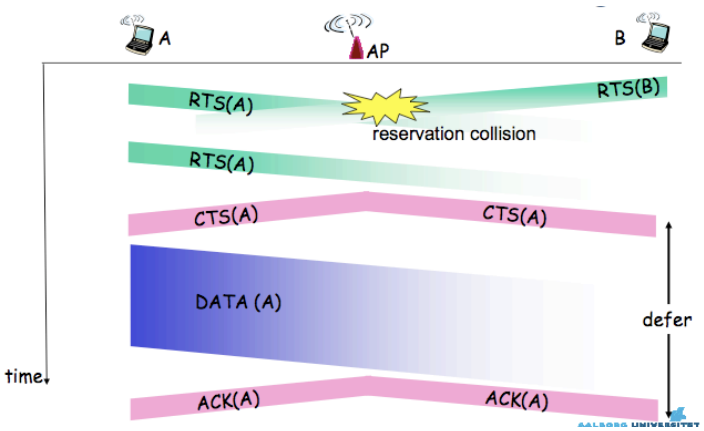
AALBORG UNIVERSITET

802.11 MAC Protocol CSMA/CA

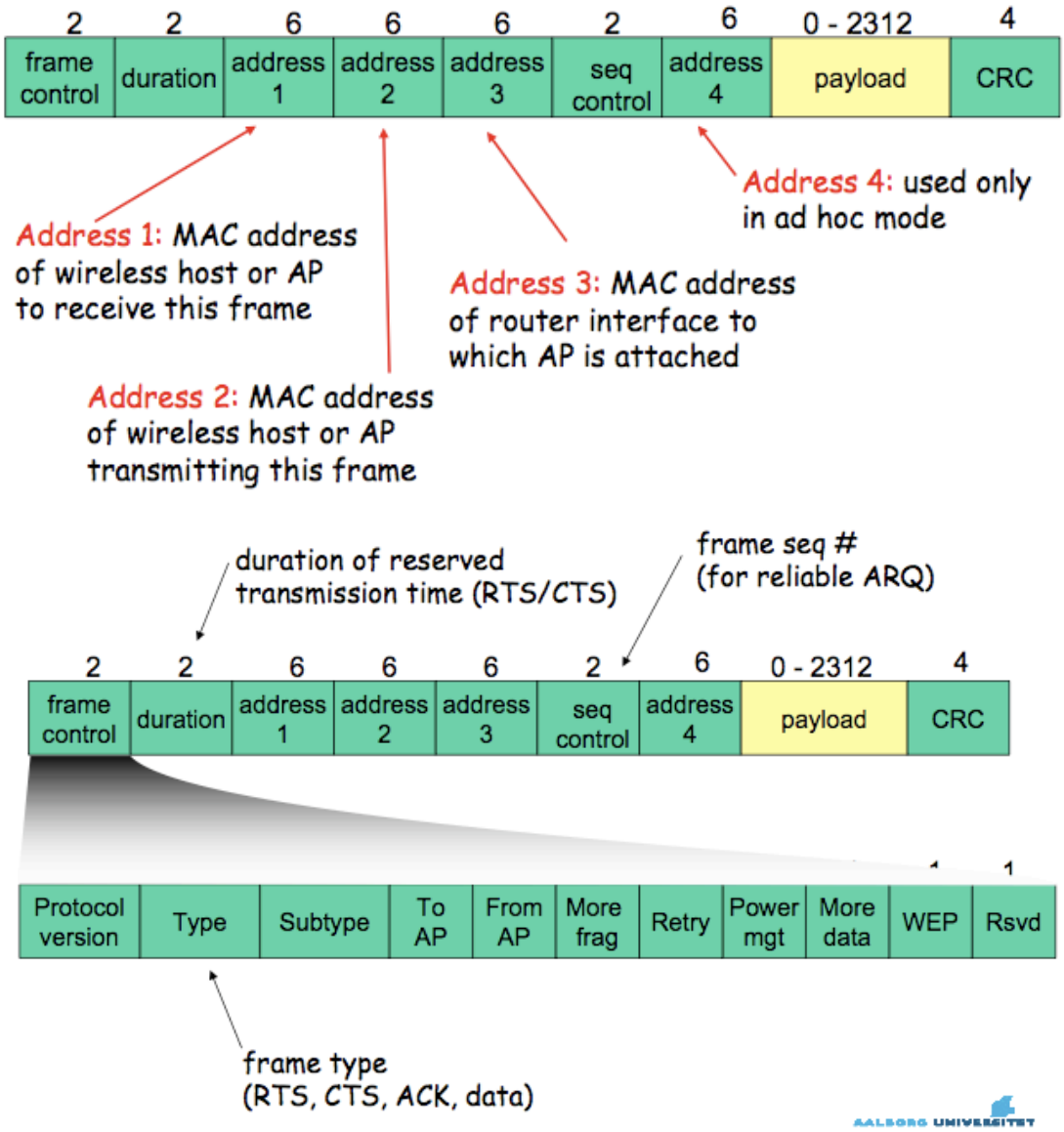
802.11 sender
1 if sense channel idle for **DIFS** then
transmit entire frame (no CD)
2 if sense channel busy then
start random backoff time
timer counts down while channel idle
transmit when timer expires
if no ACK, increase random backoff interval, repeat 2
802.11 receiver
- if frame received OK
return ACK after **SIFS** (ACK needed due to hidden terminal problem)

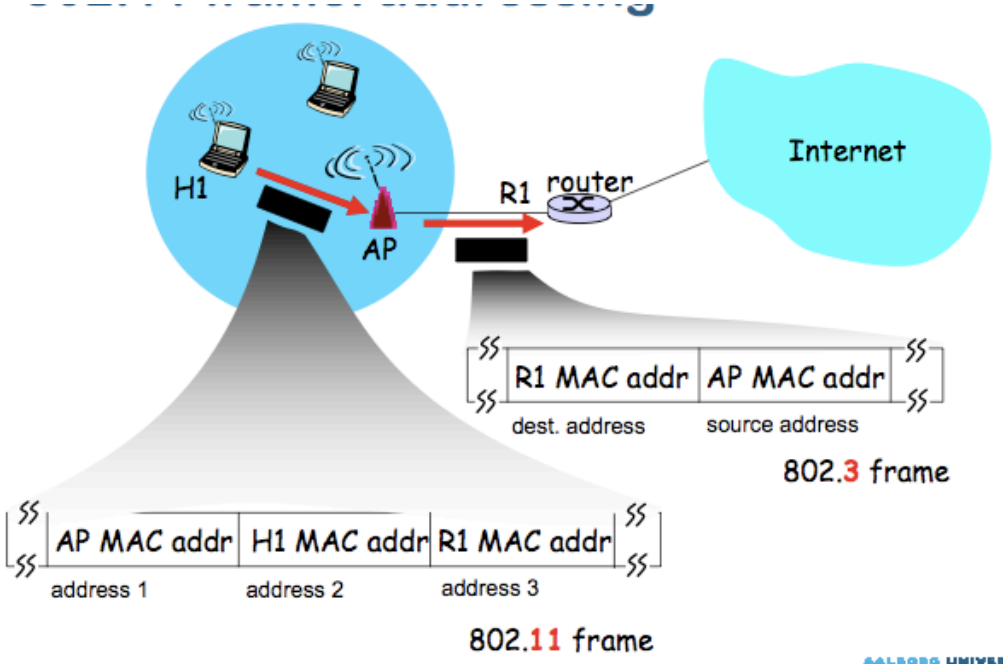


Collision avoidance



Frame of 802.11



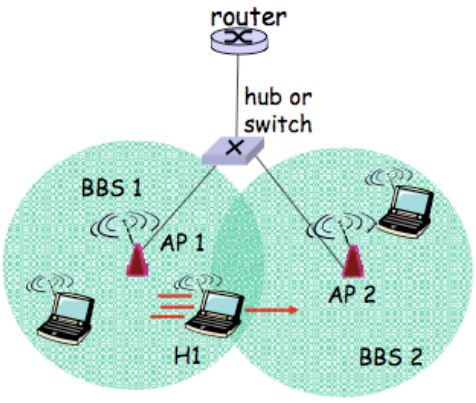


Mobility within the same subnet

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

switch: which AP is associated with H1?

- self-learning (Ch. 5): switch will see frame from H1 and "remember" which switch port can be used to reach H1



WIMAX

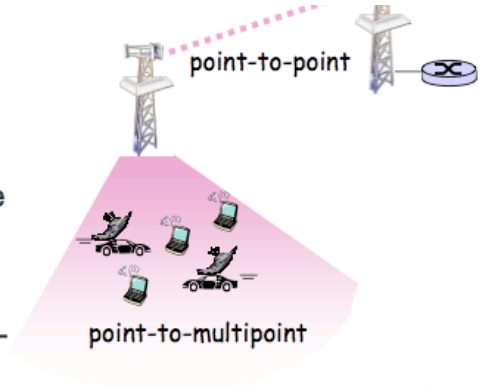
802.16: WiMAX

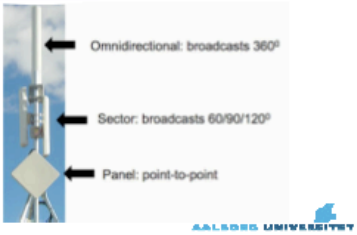
like 802.11 & cellular: base station model

- transmissions to/from base station by hosts with omnidirectional antenna
- base station-to-base station backhaul with point-to-point antenna

unlike 802.11:

- range ~ 6 miles ("city rather than coffee shop")
- ~14 Mbps

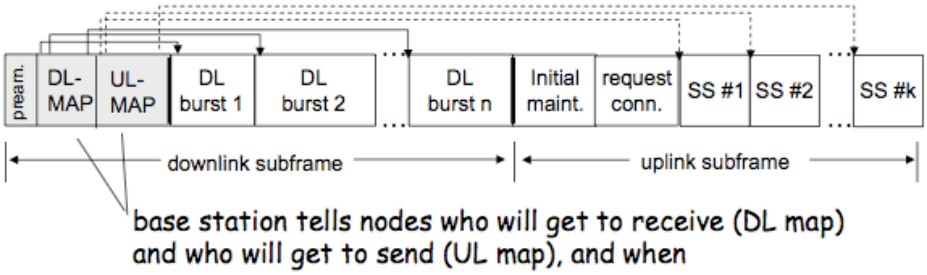




WIMAX – downlink – uplink

transmission frame

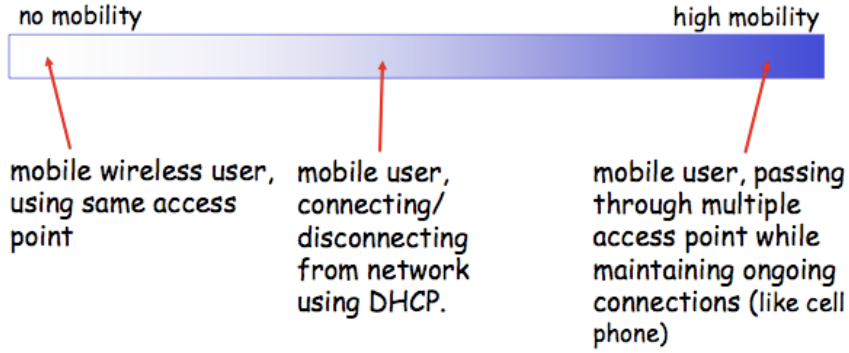
- down-link subframe: base station to node
- uplink subframe: node to base station



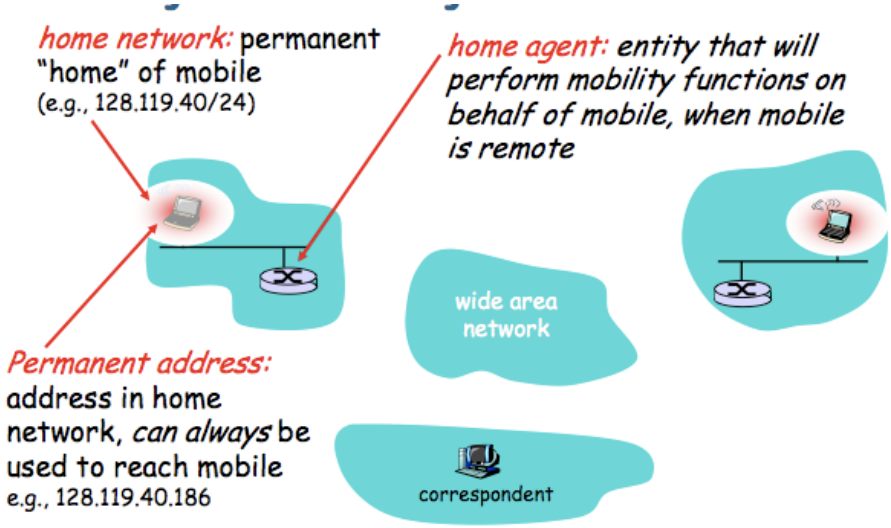
WiMAX standard provide mechanism for scheduling, but not scheduling algorithm

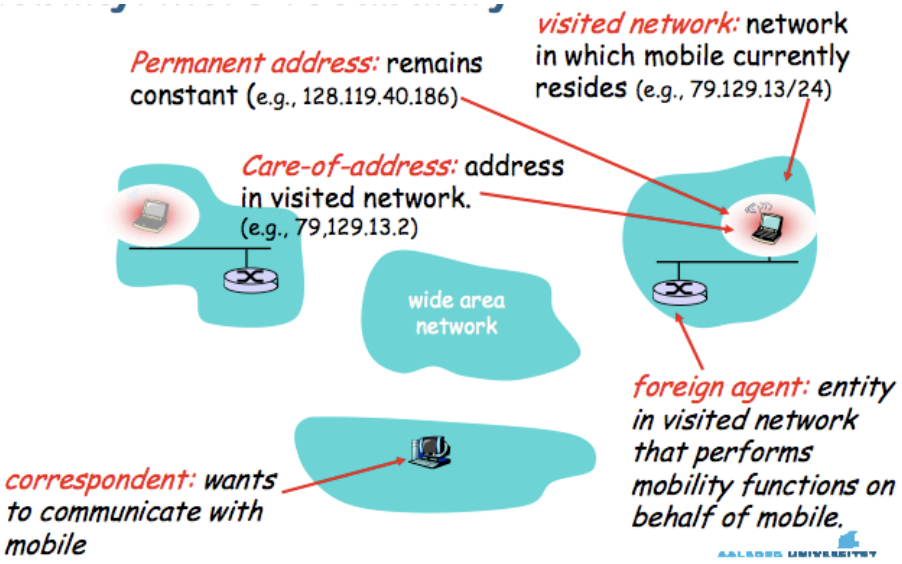
Mobility

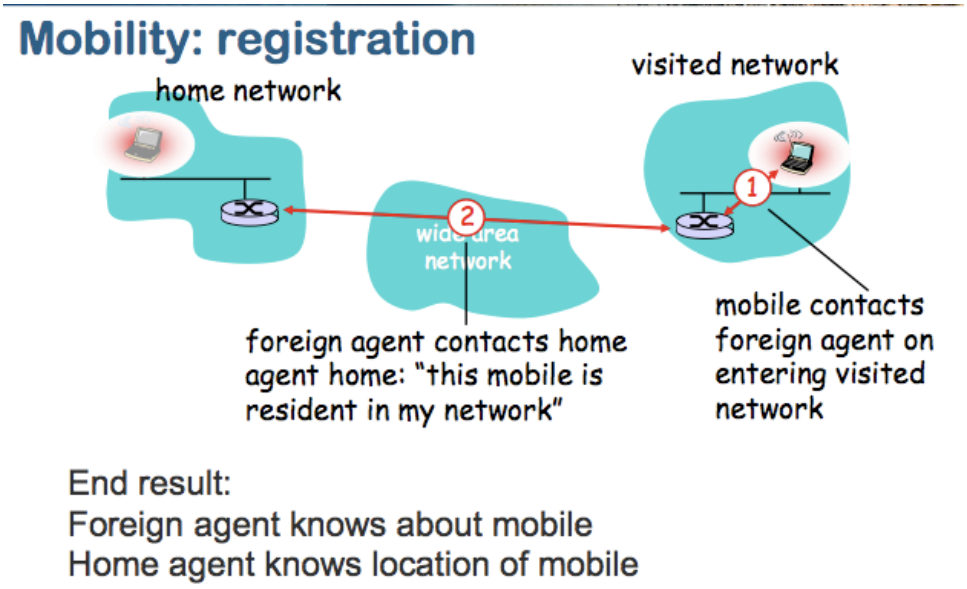
spectrum of mobility, from the *network* perspective:



Vocabulary

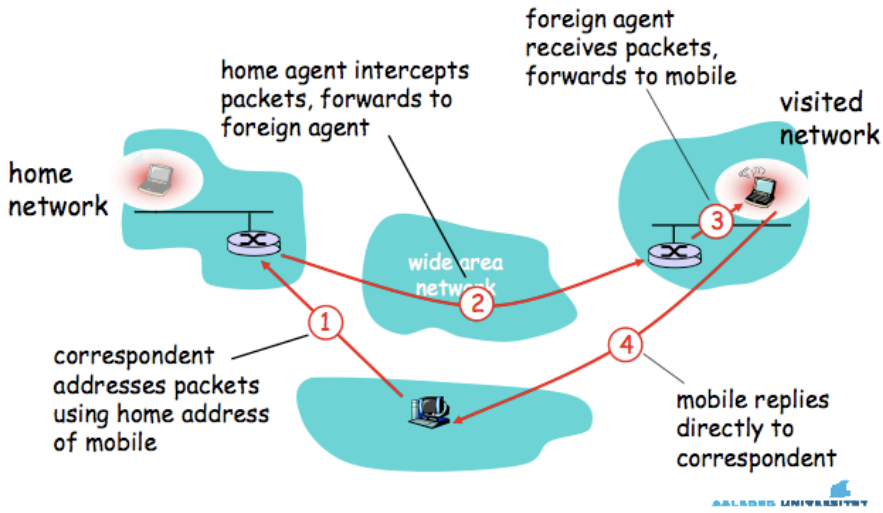




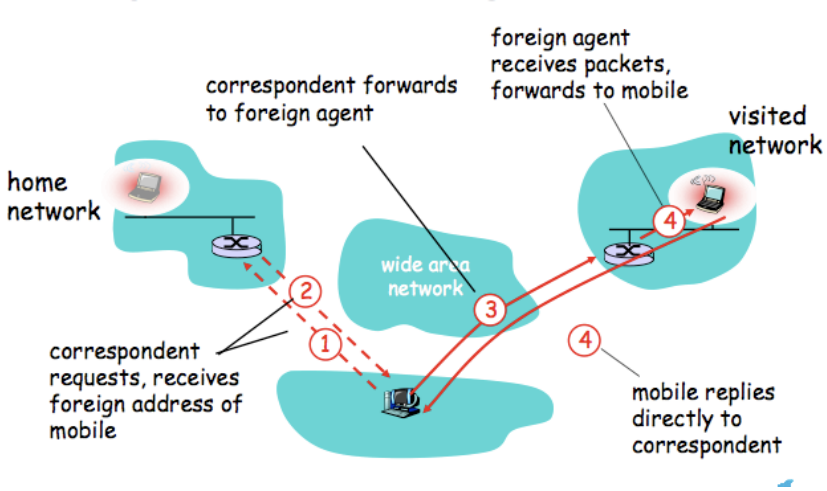


End result:
Foreign agent knows about mobile
Home agent knows location of mobile

Mobility indirect routing

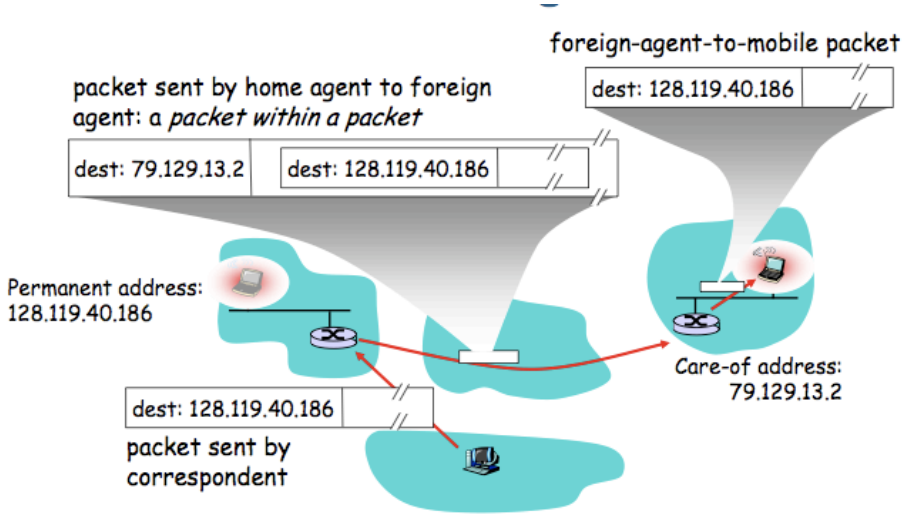


Mobility direct routing



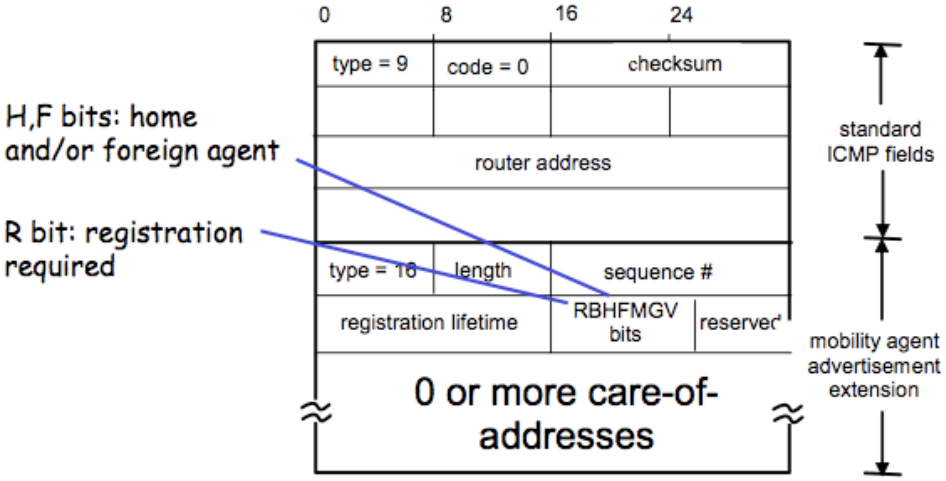
Mobile IP

Indirect routing



Agent Discovery

agent advertisement: foreign/home agents advertise service by broadcasting ICMP messages (typefield = 9)



Registration