计算机网络原理

Author: 经12-计18 张诗颖 2021011056

Supplement: Jim Kurose, Keith Ross: 《Computer Networking: A Top-Down Approach (8th edition)》

期末考点:

1. Network Overview
   network edge, network core (packet switching, circuity switching)
   packet delay, packet loss, throughput
2. Application Layer
   HTTP (HyperText Transfer Protocol): request & reponse types
   stateful protocol: cookies
   SMTP (Simple Mail Transfer Protocol), **IMAP** (Internet Mail Access Protocol)
   DNS (Domain Name system): iterated, recursive
   others: P2P, DASH (Dynamix Adaptive Streaming, video streaming), CDN (content distribution networks)

   multiplexing / demultiplexing (transport layer)
   application, process, port, socket
3. Transport Layer
   checksum calculation
   multiplexing / demultiplexing (transport layer): application, process, port, socket
   - **RDT** (reliable data transfer)
     paradigm: Stop-and-Wait, Go-back N, Selective Repeat
     application: tcp_rdt, fast restransmit (3 unACKed)
   - congestion control / 拥塞控制
     paradigm: End-End, Network-Assisted
     (cwnd) **AIMD** (Additive Increase Multiplicative Decrease) (Reno, Tahoe), **CUBID**, delay-based (measured throughput vs. uncongested throughput); **ECN** (Explicit Congestion Notification) (also network-assisted)
   - 区分：flow control (rwnd)
4. Network Layer - routing
   - Data Plane:
     DHCP allocates host-part IP address, **ICANN** allocates subnet-part IP address; **NAT** (Network Address Translation)

     routers: destination-based forwarding (longest prefix matching), generalized forwarding (orchestrated table)
     input, switching (memory, bus, interconnection), queuing (buffering management), pkt shceduling (FIFO, priority, RR / Round Robin, WFQ / weighted fair queuing)

     SDN (software defined network): match + action, **OpenFlow** protocol
   - Control Plane:
     algorithms: Dijkstra's Link State (con: oscillation), Bellman-Ford's Distance Vector (con: count-to-infinity, black-holing)

     intra-domain: **RIP**, **OSPF** (hierarchical OSPF); inter-domain: **BGP** (Border Gateway Protocol) (iBGP, eBGP, hot-potato routing -priority)

ICMP (Internet control message protocol): carried in IP datagrams

5. Link Layer - switching

   EDC (error detection and correction): two-dimensional parity check, **CRC** (cyclic redundency check)

   MAC (Multiple Access Control): **FDMA** (Frequency Division Mutiple Access), **TDMA** (Time Division Mutiple Access), slotted **ALOHA** (synchronization, 37%), pure ALOHA (18%), **CSMA** (Carrier Sense Multiple Access), **CSMA/CD** (with Collision Detection, decentralized, more efficient than ALOHA), taking turns

   ARP (address resolution protocol): broadcasting + self-learning
   Ethernet: (MAC) unslotted CSMA/CD with binary backoff; connectionless, unreliable; switches functions

   VLAN (Virtual LAN): **MPLS** (Multiprotocol Label Switching), link layer protocol (between IP routing layer and MAC Ethernet layer), use label for fast lookup rather than IP routing

   Wireless LAN
   challenges: decreased signal strength (**SNR** (Signal-to-Noise Ratio), **BER** (Bit Error Rate)), inference from other sources (hidden terminal problem, exposed terminal problem)

   BSS (Basic Service Set): **BS** (Base Station) / **AP** (Access Point), infrastructure mode and **ad hoc** mode

   CDMA (Code Division Multiple Access)

   CSMA/CS (Carrier Sense Multiple Access with Collision Avoidance): DIFS and SIFS, **RTS** (Request-To-Send) - **CTS** (Clear-To-Send)

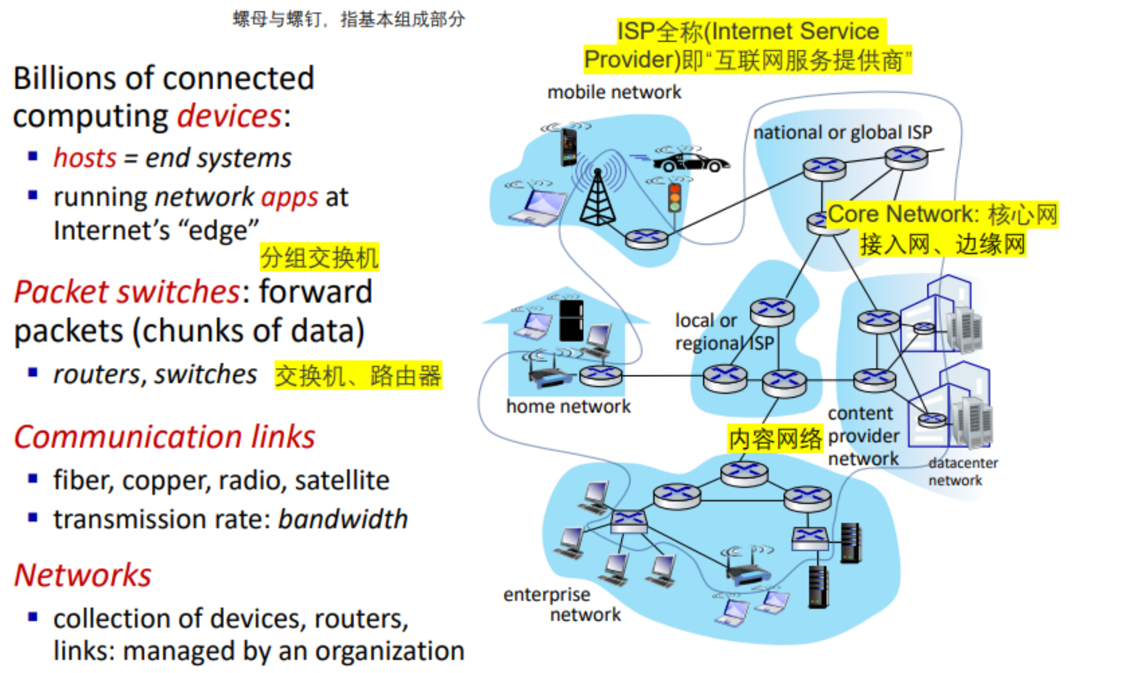
   4G and 5G
   LTE (Long-Term Evolution) standard

6. 得过图灵奖的计算机网络科学家
   www: Tim Berners-Lee
   Ethernet: Bob Metcalfe
   TCP/IP: Vinton Cerf

10选择 + 6判断

Chapter1: Network Overview

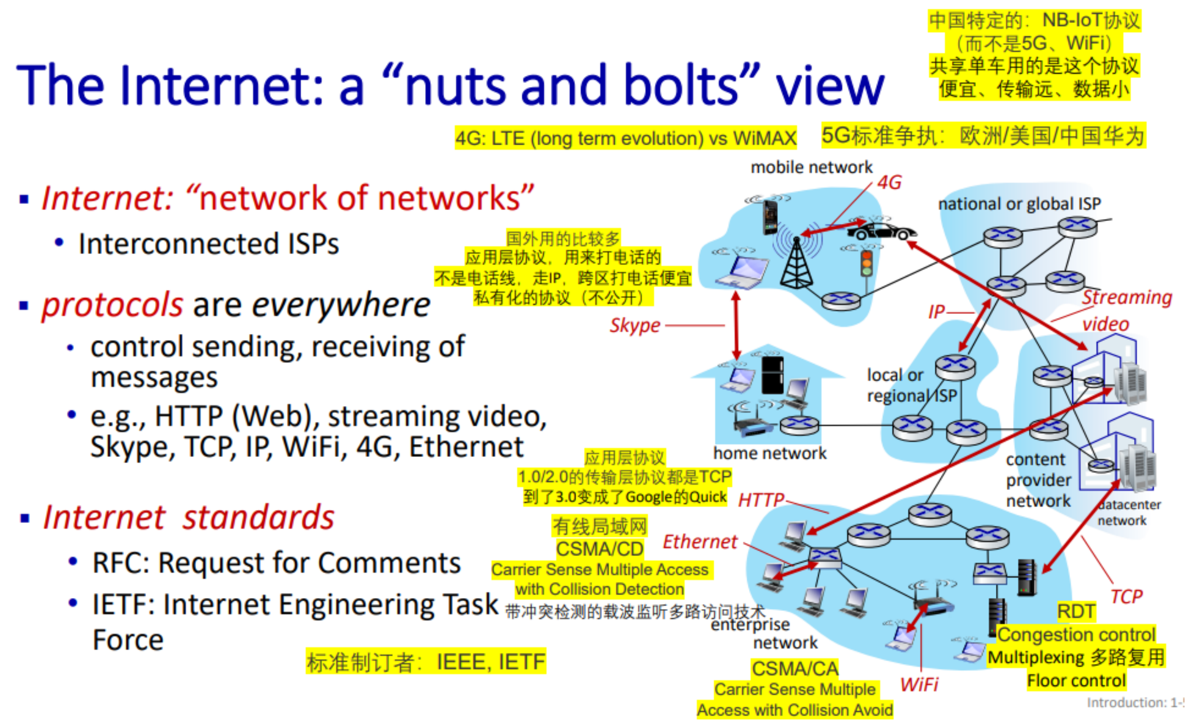
1. a "nuts and bolts" view:



<u>devices/apps, hosts/end systems</u>: Network edge
<u>routers / switches</u>: Packet switches as communication links (bandwidth)
<u>local / regional ISP</u>: access to Core Network

=> **Internet**: network of networks (interconnected ISPs)

2. protocols and standards overview:

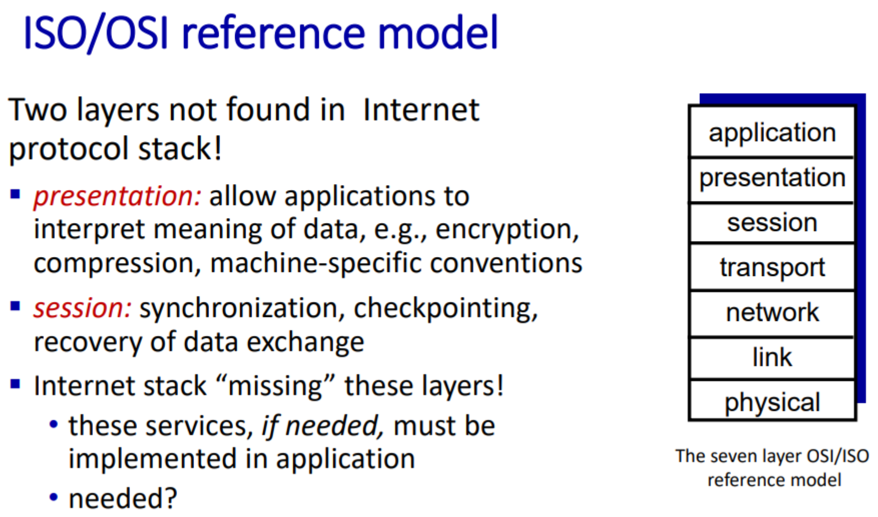


The Internet is a **decentralized** network without a single governing body or organization that has complete control over it. Key organizations and standards bodies playing important roles are:

1. Internet Corporation for Assigned Names and Numbers / ICANN: managing DNS and allocation of IP addresses
2. Internet Engineering Task Force / IETF: community-driven, open standards organization that develops and promotes Internet standards
3. Internet Society / ISOC: promote the open development of the Internet

4. Regional Internet Registries / RIRs: manage IP address blocks within respective regions. There are **5** RIRs globally.
5. Request for Comments / RFC: docs describing protocols, developed by IETF

- **Protocols**

> Using **OSI (Open Systems Interconnection) model** to introduce these protocols:
>
> 
>

  - **Application-Layer Protocols:**
    HTTP (HyperText Transfer Protocol): web
    Skype: inter-household, private protocol
    IMAP (Internet Message Access Protocol): email protocol, email access
    SMTP (Simple Mail Transfer Protocol): email protocol, email delivery
    DNS (Domain Name System): domain names to IP address translation
  - **Transport-Layer Protocols**:
    TCP (Transmission Control Protocol): server-client
    UDP (Unreliable Datagram Protocol): unreliable but faster
  - **Network-Layer Protocols**:
    IP (Internet Protocol): rout packest across networks
  - **Link-layer / Physical Protocols**:
    WiFi (Wireless Fidelity): router-host wireless communication technology
        link-layer protocol: IEEE802.11 (defines wireless LANs)
        physical-layer protocol: IEEE802.11
    Ethernet: local area network (LAN) by physical cable transmission
        link-layer protocol: IEEE802.3 (called Ethernet protocol)
        physical-layer protocol: IEEE802.3 (cable type and length...)
    PPP (Point-to-Point Protocol): computer ISP using telephone lines
  - 5G: defines physical, data-link, network, and transport layer protocols
- **Internet Standards**
  - RFC (Request for Comments): documents maintained by organizations
  - IETF (Internet Engineering Task Force): primary RFC designers

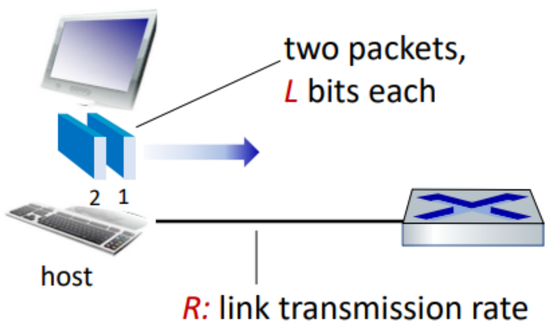
1.1 Internet Structure

1.1.1 Network Edge

> **Components**: hosts / end-devices / clients, servers (data center)
> End systems are connected to edge routers.

- **Access Network**: wired or wireless communication links, connecting end devices to the network infrastructure
    - Cable-based Access：实现插电上网（powerline communication）
      HFC: hybrid fiber coax
      CMTS: cable modem termination system
      FDM: frequency division multiplexing
    - DSL / Digital Subsriber Line
      use existing telephone lines to transmit data
    - Home Networks
      WiFi wireless access point + Wired Ethernet + router, firewall, NAT + cable or DSL modem（三合一 / 交换机、WiFi基站）
      via a base station (aka. access point): wireless local area networks (WLANs)（目前使用的是WiFi5~6，WiFi7正在生产，WiFi8正在制定标准），Wide-area cellular access networks（一个4G基站能覆盖500m~2km半径）
    - Data Center Networks
- **Customer Network**: under control of the end users (e.g., home / enterprise network)

1. Host: sends packets of data



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

现实里一个包/package大约1K
一个图像（100K）：100个包传输
一个bit的物理传输距离：200m

$$Packet\ Transmission\ Delay = \frac{L(bits)}{R(bits/sec)}$$

R: link transmission rate, link capacity, link bandwidth
L: length of a packet

2. Links: physical media

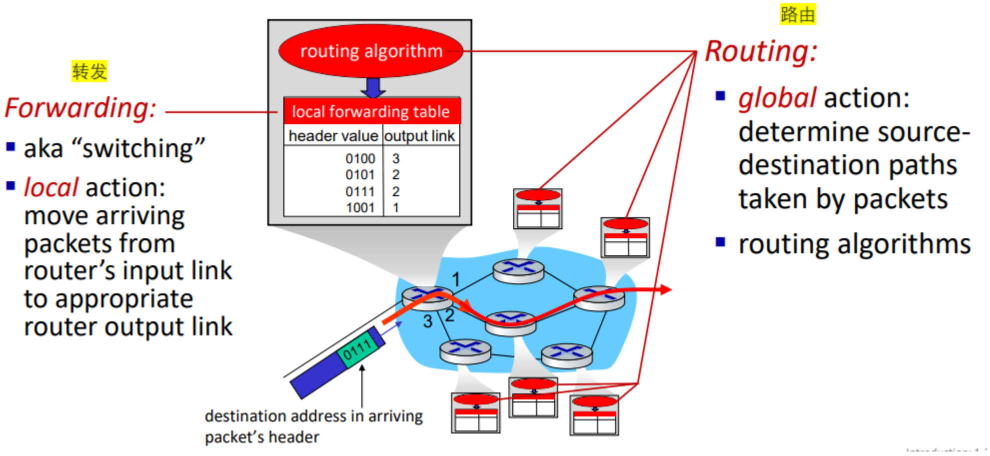
bit, physical link, guided media, unguided media, TP (Twisted Pair)
Coaxial cable, Fiber optic cable, Wireless radio, Radio link types...

1.1.2 Network Core

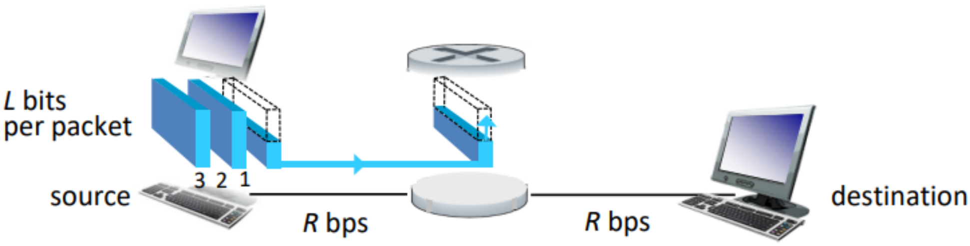
> **Components**: interconnected routers, network of networks

Key Functions:

- **Forwarding / 转发**: aka., switching, local action via **local forwarding table** (which is a "header value - output link" table)
- **Routing / 路由**: global action via **routing algorithms**



1. Packet Switching

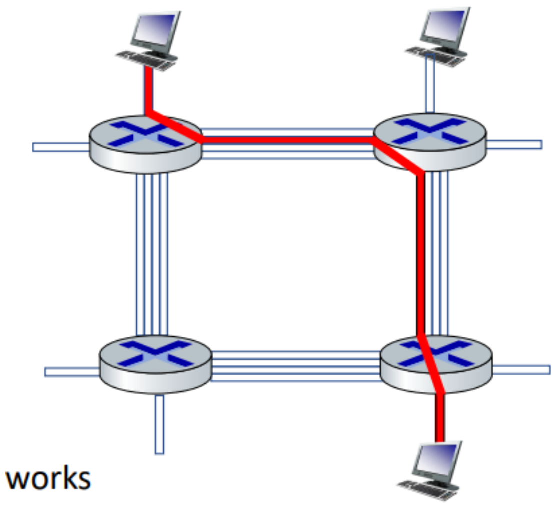
> **store-and-forward**: entire packet must arrive at router before it can be transmitted to next link
>
> 
>

`pps`: packet per second
`bps`: bit per second

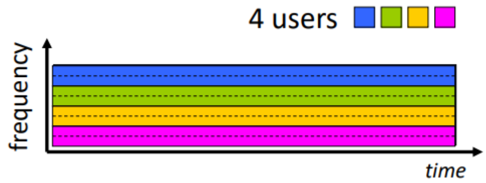
characteristic: on-demand allocation (packetized data, connectionless / no dedicated path before data transmission, variable delay, shared resources)

2. Circuit Switching

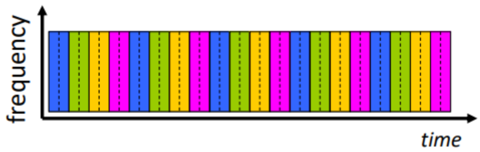
> end-end resources allocated to, reserved for “call” between source and destination



- FDM / Frequency Division Multiplexing



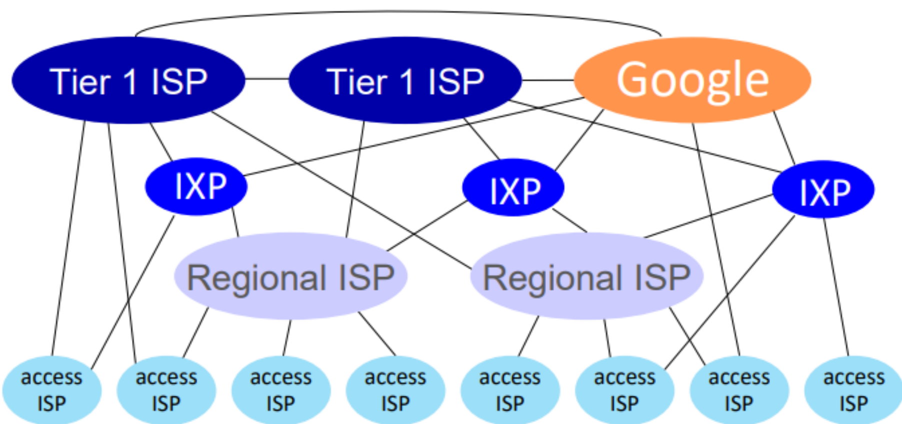
- TDM / Time Division Multiplexing



characteristic: reserved resources (dedicated path, connection-oriented, predictable delay)

3. Internet Structure

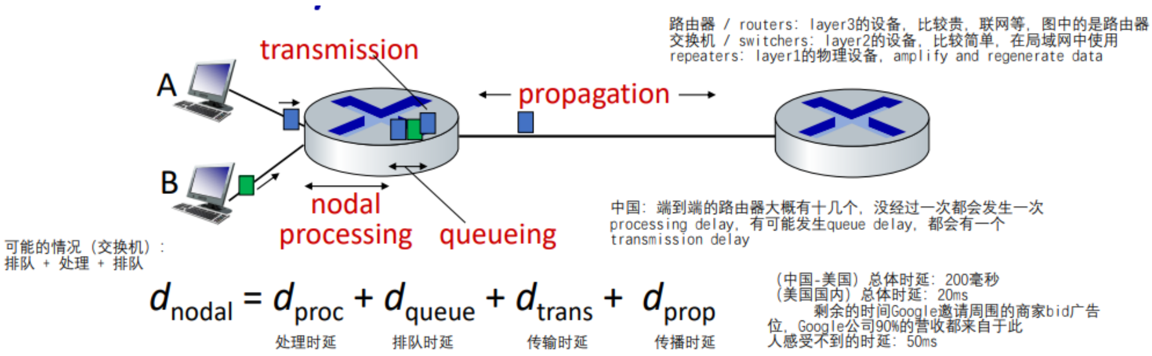
- **Tier-1 Commercial ISPs**: national and international coverage
  ISPs and regional ISPs connected to each other via **IXP** (Internet exchange points) and **peering link**, finally connected to **access ISP**
- **Content Provider Network**: private networks that: data centers => internet
  e.g., Google



1.2 Packet Delay and Loss

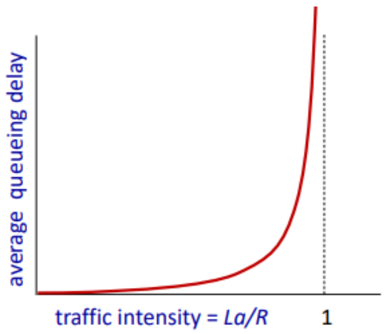
> 一个包一般1KB大小

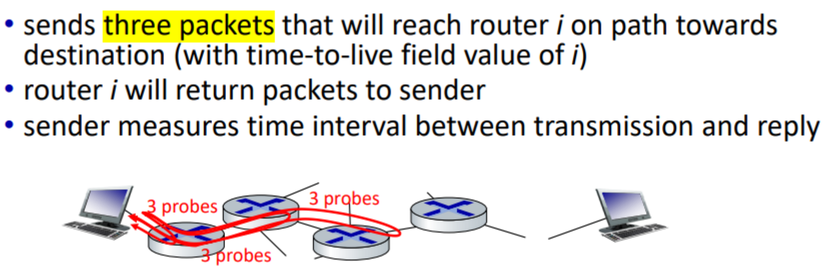
1.2.1 Packet Delay



- **d_{proc}: nodal processing** / 处理时延: < ms
  check bit errors, determine output link
  交换机：查5000万行的路由表，处理300万个包/秒
- **d_{queue}: queueing delay** / 排队时延: 20ms (domestic), 200ms (international)
  time waiting for transmission, depends on the congestion level of router
  <u>通常是国际通讯中最大的时延因素</u>

$$traffic\ intensity = \frac{L \cdot a}{R} = \frac{arrival\ rate\ of\ bits}{service\ rate\ of\ bits}$$



> Visualization: `tracert cmu.edu`
> This **traceroute** program provides delay measurement from source to router along end-end Internet path towards destination.
>
> 
>

- **d_{trans}: transmission delay** / 传输时延 $= \frac{L}{R}$
  happens when 通过网卡发出包的时候
- **d_{prop}: propagation delay** / 传播时延 $= \frac{d}{s}$
  d: length of physical link
  s: propagation speed (~2*10^8 m/sec)
  物理上传播比特信息的时间
  <u>通常是国内通讯中最大的时延因素</u>

1.2.2 Packet Loss

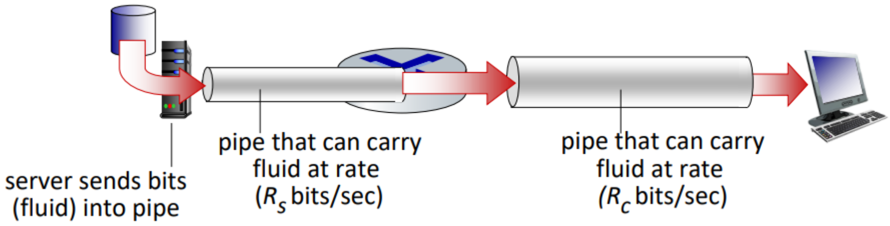
- **Buffer Overflow**: 0%~100%
- **Hardware Error**: reversed bytes, cannot pass CRC verification, 0.04%

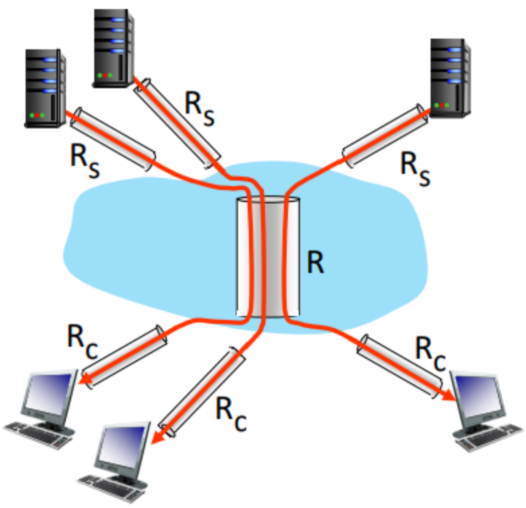
> 将包逐个"异或"（bitwise exclusive or）起来得到一个check packet，因此如果丢了一个包，无论丢了哪个，都可以及时复原那个包的内容
> finite field：可以实现任意丢掉两个包的自动纠错（包括n个包）

- **WiFi air interface**: 15% (which is severe!)
- **on purpose**: early congestion control, send warnings when buffer is to reach its limit

1.2.3 Throughput

> Throughput / 吞吐量: rate (bits/time unit) at which bits are being sent from sender to receiver





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

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

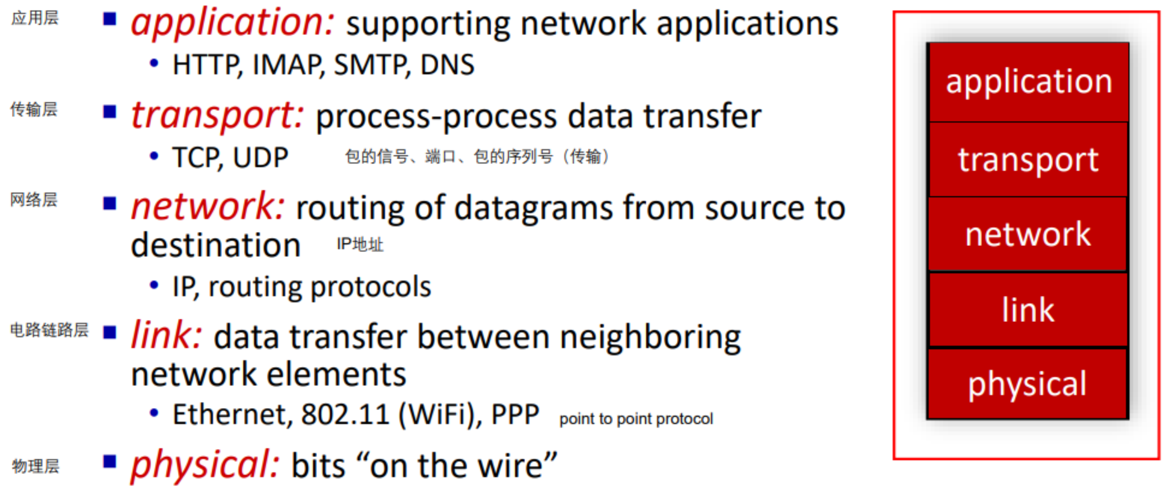
$$Throughput = min\{R_c(client), R_s(server)\}$$

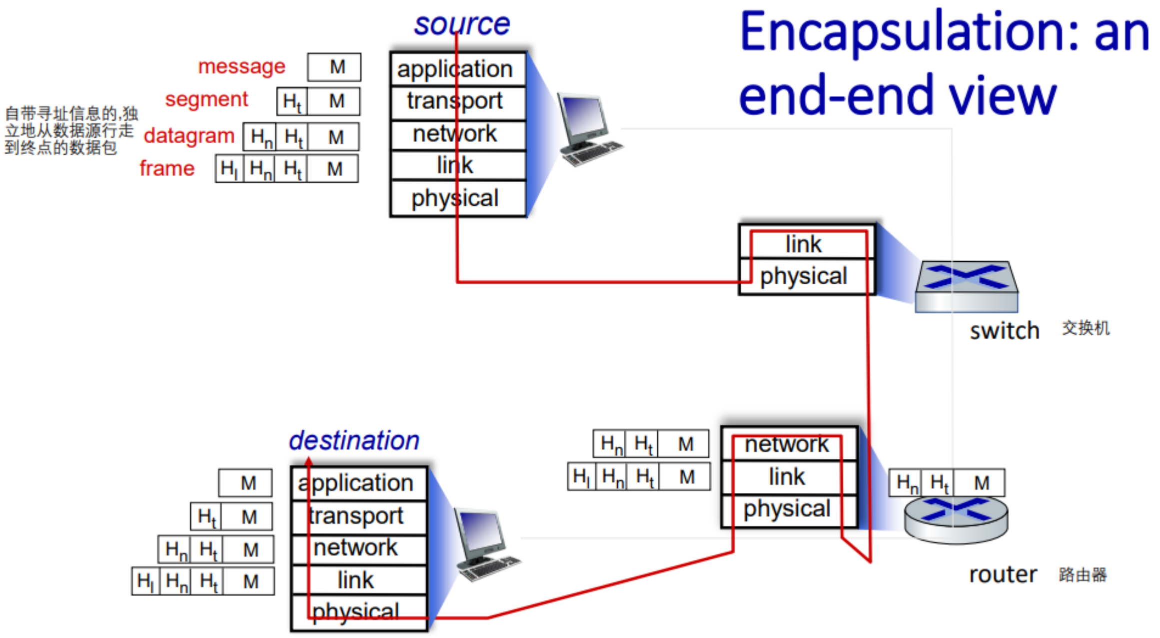
1.3 Network Security

> 1. 钓鱼攻击：如钓鱼邮件
> 2. 鱼叉攻击（APT / Advanced Persistent Threat）：长期的、有预谋、有组织的攻击；单点突破，木马植入，持续监听（密码和口令）
> 3. 水坑攻击：在你可能的常用网站上埋伏密码
> 4. 供应链攻击：提前预测你要下载xx软件，提前监听
>
> 防御：防火墙；蜜罐（引诱攻击程序暴露信息）

- **Attacks**
  Packet "Sniffing": 窃取数据
  IP Spoofing: 伪造身份
  Denial of Service (DoS): 阻断服务攻击（DDoS / Distributed Denial of Service）
  Others: playback attack (Alice - Attacker - Bob), Digital Signatures
- **Defense**
  CIA standard:
  Confidentiality / 加密
  Integrity checks / 完整
  Access restrcitions / 访问控制（不只是基于用户名和密码）
  Methods: DES (Data Encryption Standard, or 3DES), RSA（模运算 modular algorithm（大素数乘积素因数分解）对称加密生成session keys）

1.4 Internet Layers





Chapter2: Application Layer

2.1 Paradigms

> IP地址指的是逻辑地址，并不唯一，可以根据实际情况进行更改；
> MAC地址指的是硬件地址，具有全球唯一性，并不可以进行更改（软件仿冒并不属于更改）
>
> 主机：一般有一个wifi网网卡卡，一个以太网网卡
> 路由器：有多个网卡，多个ip address
>
> 动态分配ip address——分为静态和动态ip地址

2.1.1 Client-Server Paradigm

> classical protocol: Dynamic Host Configuration Protocol (DHCP)

- **server**
  always-on host
  permanent IP address
  often in data centers, for scaling
- **clients**
  contact, communicate with server
  may be intermittently connected
  may have dynamic IP address
  do NOT communicate directly with each other
  <u>Examples</u>: HTTP, IMAP, FTP (File Transfer Protocol, 很古老)

2.1.2 Peer-Peer Architecture

- **characteristics**
  no always-on server
  arbitrary end systems directly communicate
  <u>self scalability</u>: new peers bring new service capacity, as well as new service demands
  <u>complex management</u>: peers are intermittently connected and change IP addresses
  <u>Examples</u>: P2P file sharing
- **last-coupon problem** (not really mentioned on the internet)

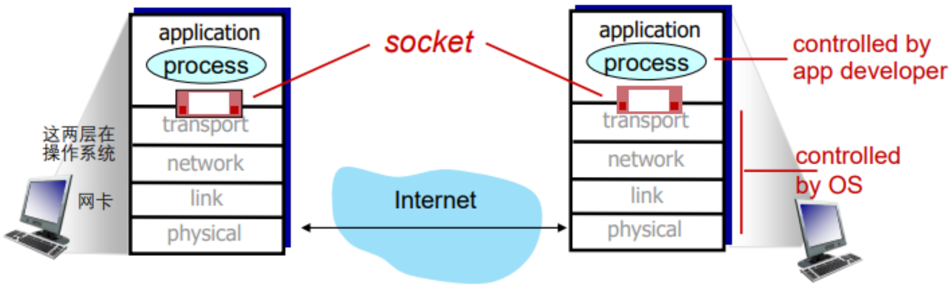
2.1.3 Processes Communicating

> <u>Process</u>: program running within a host

- **Clients and Servers**
  client process: process that initiates communication
  server process: process that waits to be contacted
- **Methods**
  <u>within same host</u>, two processes communicate using **inter-process communication** (defined by **OS**)
  <u>processes in different hosts</u>: communicate by **exchanging messages**

1. Sockets

> process sends/receives messages to/from its socket



- **identifier**: unique to each process to receive messages
  identifier includes **IP address** and **port numbers** associated with process on host.
  Example port numbers: HTTP server - `80`; mail server - `25`

2. General Principles

- **content**:
  types of messages exchanges: request, response...
  message syntax && semantics && rules
  open protocols / proprietary protocols (e.g., Skype, Zoom)
- **purpose**:
  data integrity, timing, throughput, security

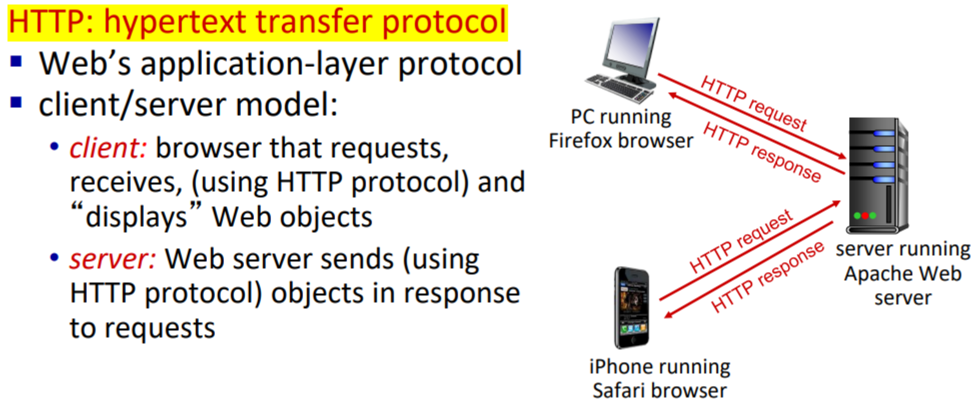
application	data loss	throughput	time sensitive?
file transfer/download	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kbps-1Mbps video:10Kbps-5Mbps	yes, 10's msec
streaming audio/video	loss-tolerant	same as above	yes, few secs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

<u>underlying transport protocols</u>:

application	application layer protocol	transport protocol
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Web documents	HTTP 1.1 [RFC 7320]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
streaming audio/video	HTTP [RFC 7320], DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

2.2 Specific Analysis

2.2.1 Web and HTTP

> **Web**: web page consists of **objects**, each of which can be stored on different Web servers. Also, web page consists of **base HTML-file** which includes several referenced objects, each addressable by a **URL**, e.g., `eee.someschool.edu` (hostname) `/someDept/pic.gif` (path name)
>
> **HTTP** is Web's application-layer protocol
>
> Paradigm: client-server model
>
> 
>

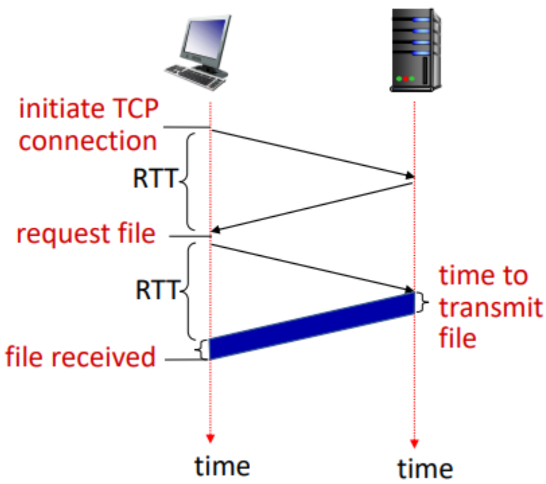
1. HTTP overviews

- **uses TCP**
  client initiates TCP connection (creates socket) to server, port `80`
  server accepts TCP connection from client
  HTTP messages exchanged between browser (client) and Web server (server)
  TCP connection closed
  3.0是基于Quick的（传输层协议）
- **stateless**
  server maintains no information about past client requests
- **classification: non-persistent vs persistent**

  > Definition:
  > **RTT / Round Trip Time**: time for a small packet to travel from client to server and back

  Non-persistent: downloading multiple objects required multiple connections
  每一个server object都需要单独建立TCP连接 overhead还是比较大的

$$response\ time = 2RTT + file\ transmission\ time$$



Persistent: multiple objects can be sent over single TCP connection between client and that server
一次建立连接可以传输多个objects
HTTP1.1：平均下来每个referenced objects只需要1个RTT时延

- **Socket Programming with TCP**

  Python version:

```
from socket import *
serverName = 'servername'
serverPort = 12000

# client
clientSocket = socket(AF_INET, SOCK_STREAM) # using IPv4 (different hosts), type = TCP, OS would create the port number for the client socket
clientSocket.connect((serverName, serverPort))
sentence = input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print('From Server: ', modifiedSentence.decode())
clientSocket.close()

# server
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1) # maximum number of queued connections = 1
print('The server is ready to receive')
while True:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.encode())
    connectionSocket.close()
```

  Comparison with UDP:

```
from socket import *
serverName = 'servername'
serverPort = 12000

# client
clientSocket = socket(AF_INET, SOCK_DGRAM) # type = UDP
message = input('Input lowercase sentence:')
clientSocket.sendto(message.encode(),(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage.decode())
clientSocket.close()

# server
serverSocket = socket(AF_INET, SOCK_DGRAM) # type = UDP
serverSocket.bind(('', serverPort))
print("The server is ready to receive")
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

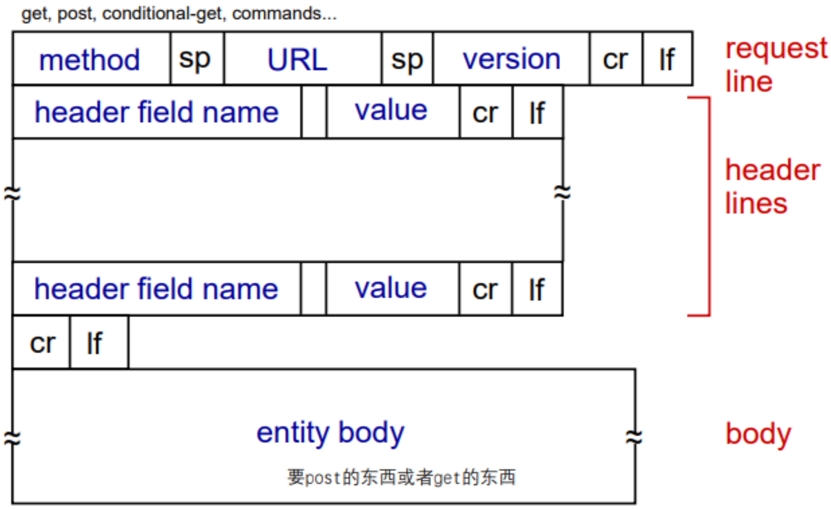
C++ version:

```
#include <arpa/inet.h>
#include <pthread.h>

#define PORT 8888
#define CLIENT_IP "183.173.19.162"
#define BUFFER_SIZE 1024

#define SERVER_PORT 8888
#define MAX_QUEUE 200
#define MY_IP "183.173.19.162"

// client
int main() {
    char buffer[BUFFER_SIZE];
    struct sockaddr_in serv_addr;
    int sd; // client socket descriptor
    sd = socket(AF_INET, SOCK_STREAM, 0); // protocol chosen automatically
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT); // host byter order => network byte
order
    inet_pton(AF_INET, CLIENT_IP, &serv_addr.sin_addr); // presentation
format (CLIENT_IP) => network byte order (&serv_addr.sin_addr)
    connect(sd, (struct sockaddr*)&serv_addr, sizeof(serv_addr)); // ret=0
on success
    send(sd, buffer, BUFFER_SIZE, 0); // flag = 0
    read(sd, buffer, BUFFER_SIZE);
}

//server
int main() {
    int sd; // server socket descriptor
    int opt = 1;
    int new_socket;
    struct sockaddr_in address;
    sd = socket(AF_INET, SOCK_STREAM, 0);
    setsockopt(sd, SOL_SOCKET, SO_REUSEADDR & SO_REUSEPORT, (char*)&opt,
sizeof(opt)); // set REUSEADDR and REUSEPORT to opt=1
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = inet_addr(MY_IP);
    address.sin_port = htons(SERVER_PORT);
    bind(sd, (struct sockaddr*)&address, sizeof(address));
    listen(sd, MAX_QUEUE);

    pthread_t thread_id; // compile: -pthread
    while ( (new_socket = accept(sd, (struct sockaddr*)&address,
(socklen_t*)&addrlen)) ) {
        pthread_create(&thread_id, NULL, connection_handler,
(void*)&new_socket); // pass arg=new_socket to connection_handler
        pthread_join(thread_id, NULL); // exit status of thread == NULL
    }
}

void* connection_handler(void* socket) {
    int sd = *(int*)socket;
    int number;
```

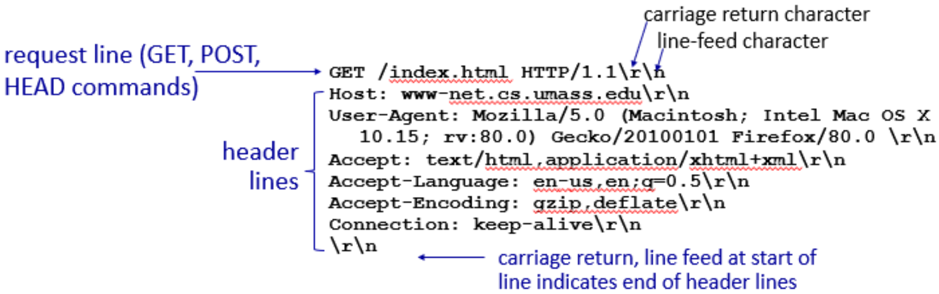
```
    char* message;
    char buffer[BUFFER_SIZE];
    char response[RESPONSE_SIZE];
    int read_size = recv(sd, buffer, BUFFER_SIZE, 0);
    // do something
    send(sd, response, strlen(response), 0);
    close(sd);
}
```

2. Two Types: Request && Response

- **HTTP request message**
  request line: (GET / 请求, HEAD commands / 描述数据, POST / 上传, PUT / 更新)



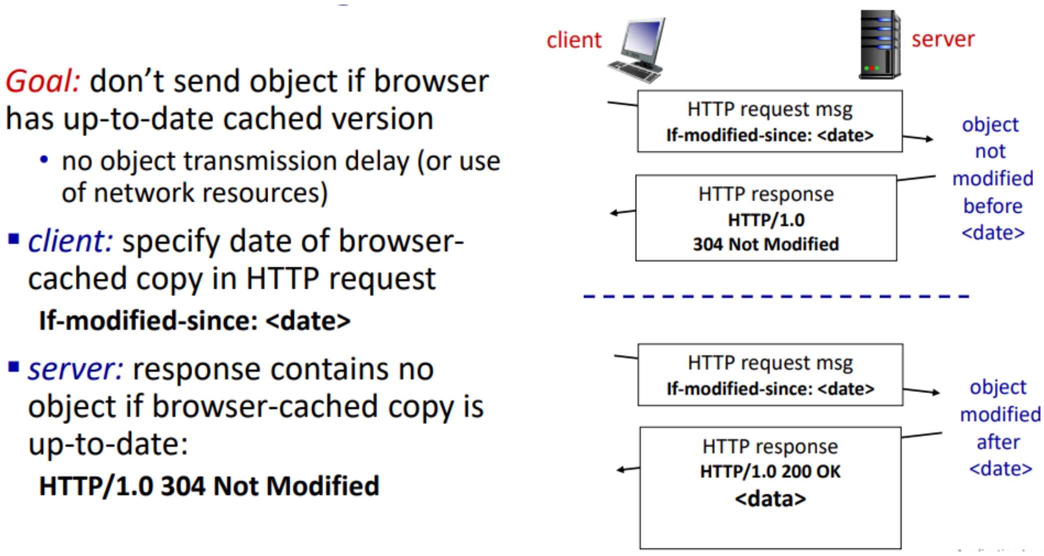

```
GET /index.html HTTP/1.1
Host: www.example.com
If-Modified-Since: Sat, 20 May 2023 12:00:00 GMT # conditional GET
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36
Accept: text/html,application/xhtml+xml
Accept-Language: en-US,en;q=0.9
Accept-Encoding: gzip, deflate
Connection: keep-alive
```



> 其中：`cr`表示`carriage return`，`lf`表示`line fitting`

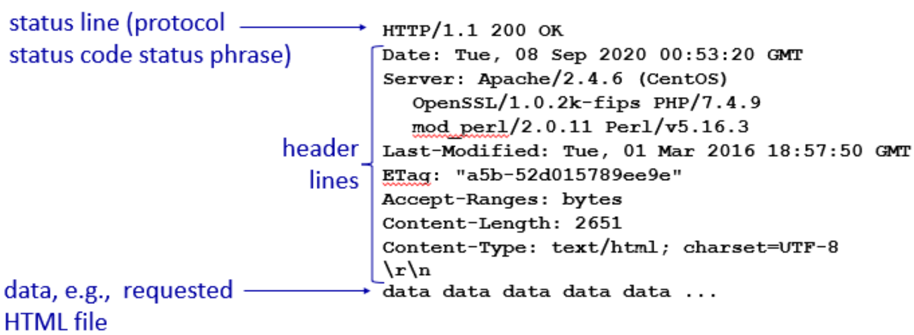
  - **GET method**: include user data in URL field of HTTP GET request message (following a '?'), e.g., `www.somesite.com/animalsearch?monkeys`

- **HEAD method**: requests headers (only) that would be returned if specified URLs were requested with an HTTP GET method.
- **POST method**: user input sent from client to server via entity body
- **PUT method**: uploads new file (object) or completely replaces old files (stated in URL section) to server via entity body
- (with cache) **Conditional GET**: don't send object if browser has up-to-date cached version by sending an `if-modified-since` message



- **HTTP response message**

```
HTTP/1.1 304 Not Modified
Date: Sun, 21 May 2023 15:30:00 GMT
Server: Apache/2.4.29 (Unix)
Connection: keep-alive
ETag: "abc123"
```



- **Status Codes**

  `200 OK`: request succeeded, requested object later in this message

  `300 Moved Permanently`: requested object moved, new location specified later in this message (in Location: field)

  `400 Bad Request`: request msg not understood by server, probably due to error format

  `404 Not Found`: requested document was not found on this server

  `505 HTTP Version Not Supported`
- **wsl codes**

`nc -v gaia.cs.umass.edu 80`: opens TCP connection to port 80 (default HTTP server port) at gaia.cs.umass.edu, anything typed in will be sent

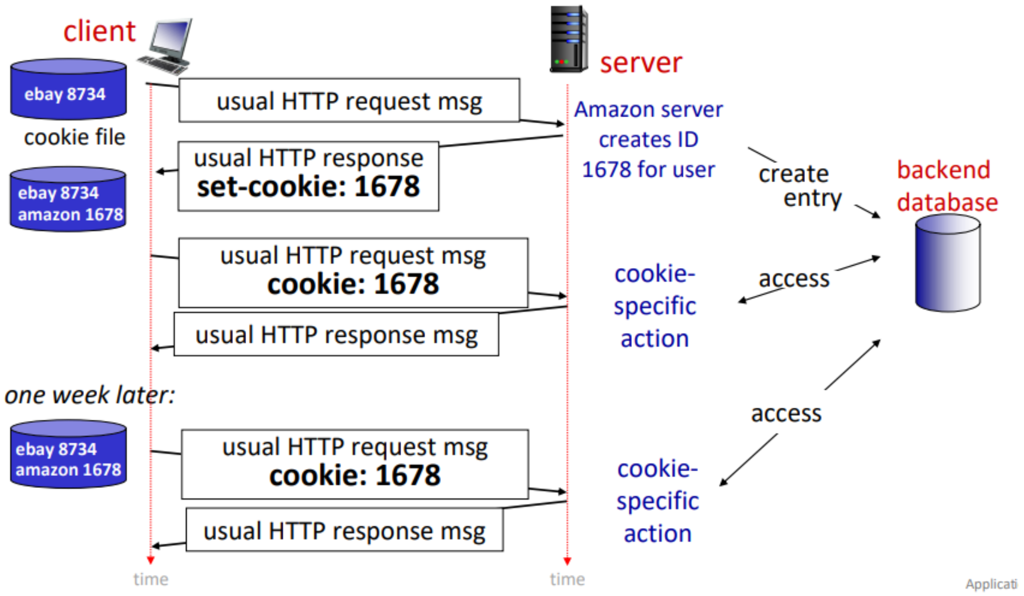
`GET /kurose_ross/interactive/index.php HTTP/1.1`

`Host: gaia.cs.umass.edu`

: by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. Stateful Protocol: cookies

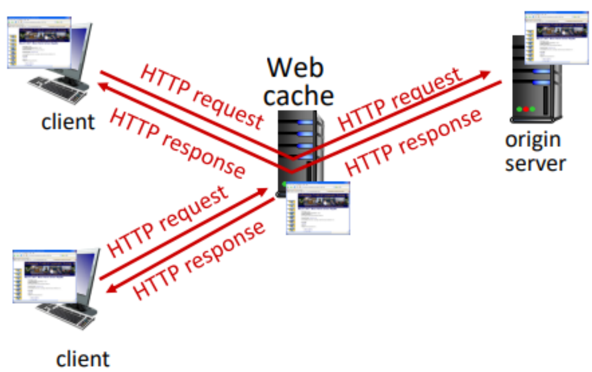
- **Four components**
  1 cookies header line of HTTP response message
  2 cookie header line in next HTTP request message
  3 cookie files kept on user's host, managed by user's browser
  4 backend databasese at Web site



- **tracking users' behavior**
  first-party cookies: track user behavior on a given website
  third party cookies: track user behavior across multiple websites (can be disabled by browsers), created by a domain other than the specific website

  GDPR (EU General Data Protection Regulation): when cookies can identify an individual, cookies are considered personal data, subject to GDPR personal data regulations

- **alternative: Web caches**
  Goal: satisfy client requests without involving origin server



Web cache / proxy servers: user can configure browser to point to a local Web cache server tells cache object's allowable caching in response header:

```
Cache-Control: max-age=<seconds>
```

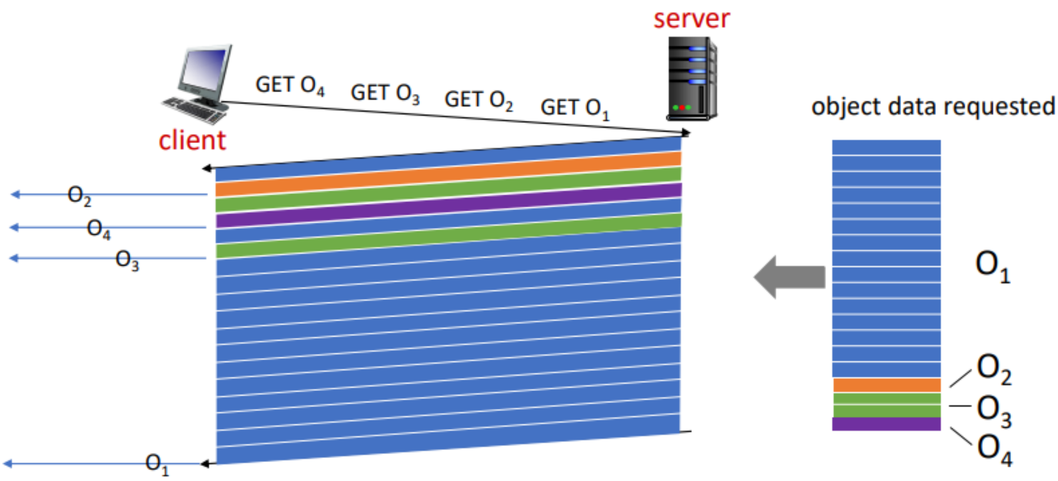
```
Cache-Control: no-cache
```

compared with "buying a faster access link": cheaper

公司短信支付模式：一个时间段里面最高频率pic来收费，因此公司需要使得网络流量尽可能的平稳

4. Version of HTTP

- **HTTP/1**: introduced multiple, pipelined GETs over single TCP connection
  server responds in-order (FCFS: first-come-first-served scheduling) to GET requests => "head-of-line / HOL blocking" small objects by large objects
- **HTTP/2**: decreased delay in multi-object HTTP requests
  allow clients to customize order of requested objects' transmission
  divide objects into frames, schedule frames to mitigate HOL blocking
  push unrequested objects to clients

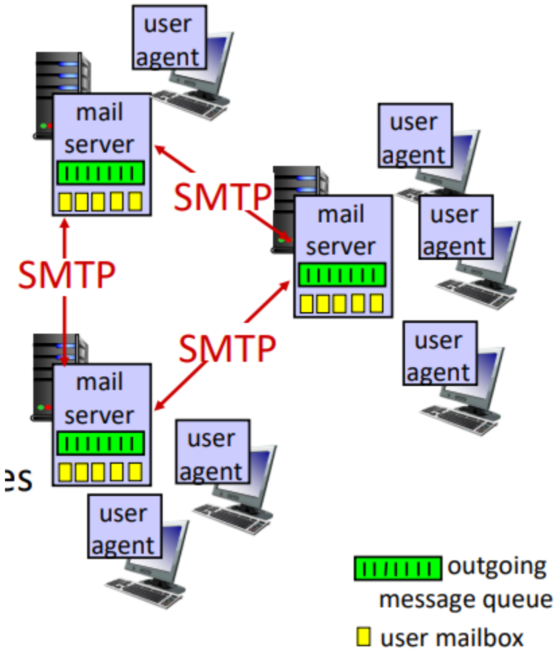


O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

- **HTTP/3**: adds security, per object error- and congestion-control (more pipelining) over UDP

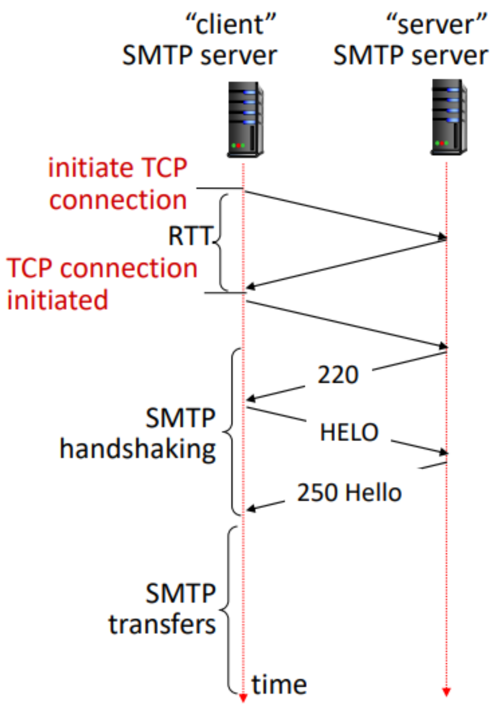
2.2.2 E-mail, SMTP, IMAP

1. SMTP: Three Major Components



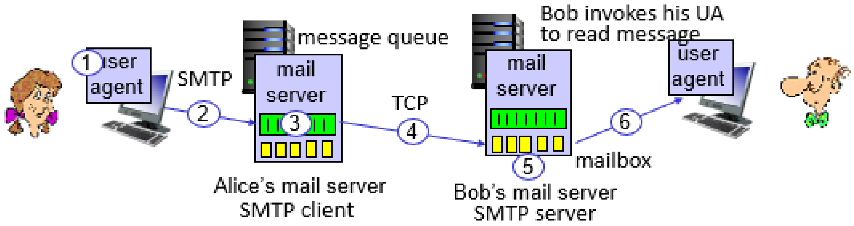
- **user agents**: aka., mail reader, e.g., outlook
- **mail servers**: mailbox (contains incoming messages for user) + message queue (to-be-sent mail messages)
- **SMTP** / Simple Mail Transfer Protocol: introduce SMTP RFC(5321) here
  uses TCP to reliably transfer email messages (mail server initiating connection) to server, port `25`

  <u>three phases of transfer</u>:
  SMTP handshaking (greeting) => SMTP transfer of messages => SMTP closure

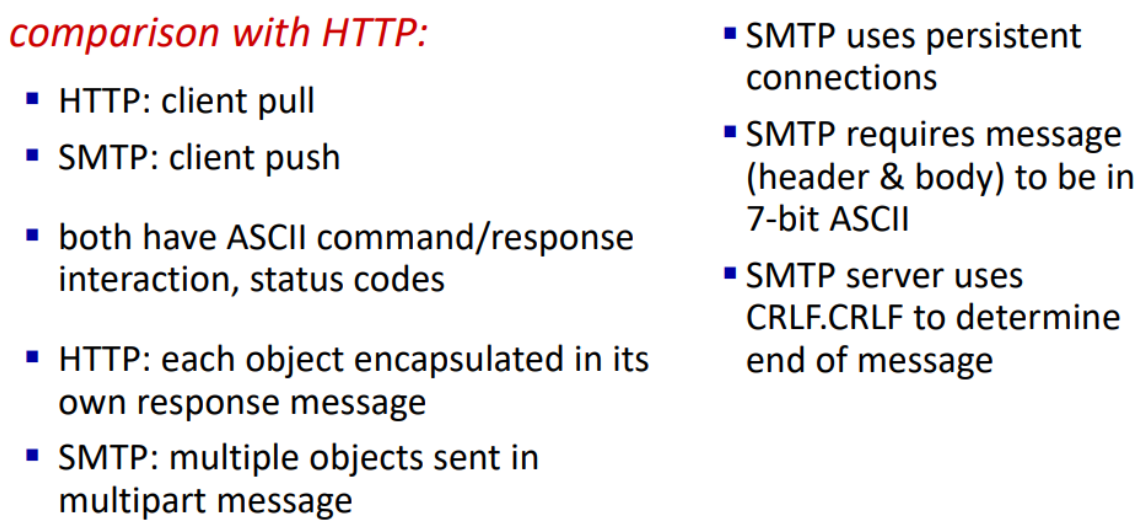
  

  `220` *here indicates the application-layer protocol used is FTP*

2. SMTP: Typical Scenario



3. SMTP: Characteristics

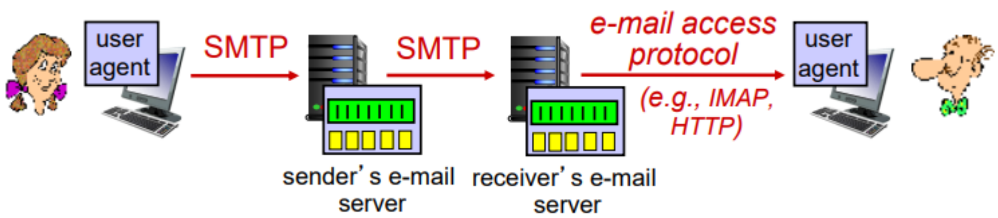


HTTP encapsulates each different type of objects (HTML, JavaScript files, CSS files) in different resposne messages, while SMTP uses MIME (Multipurpose Internet Mail Extensions) standard to send multiple objects within a single message

> HTTP - client **pull**: client requests data from server
> SMTP - client **push**: client sends data to server

4. IMAP: mail access protocols

> **IMAP / Internet Mail Access Protocol**: messages stored on server, IMAP provides retrieval, deletion, folders of stored messages on server
>
> **HTTP**: Gmail, Hotmail, Yahoo!Mail, etc. provides web-based interface on top of STMP (to send), IMAP (or POP) to retrieve e-mail messages



2.2.3 DNS: Domain Name System

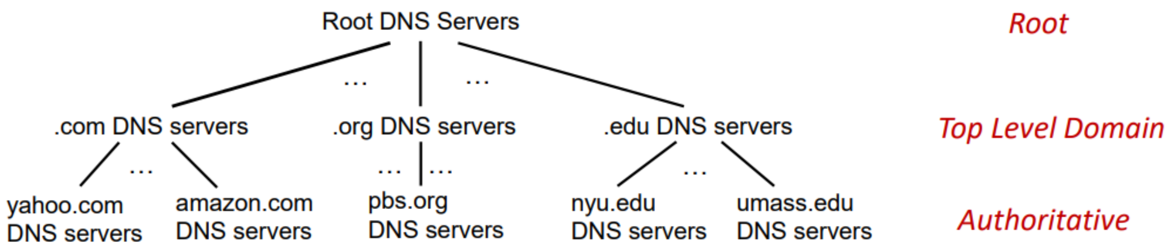
<u>Definition</u>:

distributed database implemented in **hierarchy** of many **name servers**
application-layer protocol: host and DNS servers communicate to resolve names
millions of different organizations responsible for their records

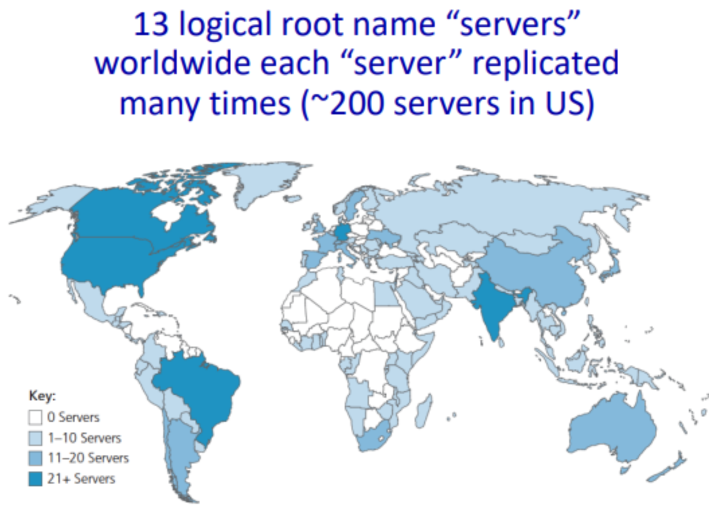
<u>Functions</u>:

hostname-to-IP-address translation

host / mail server aliasing 别名使用, load distribution: many IP addresses correspond to one name



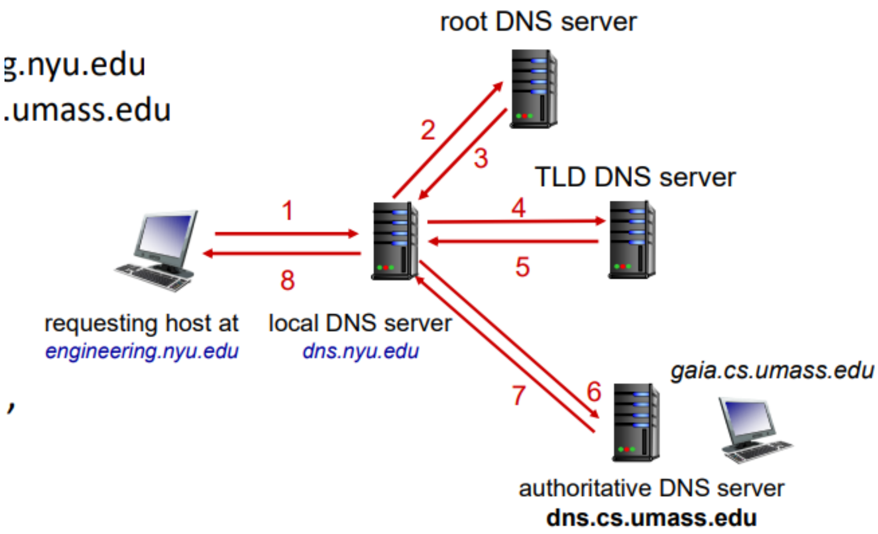
- **Root name servers**: official, contact-of-last-resort by name servers that can not resolve name



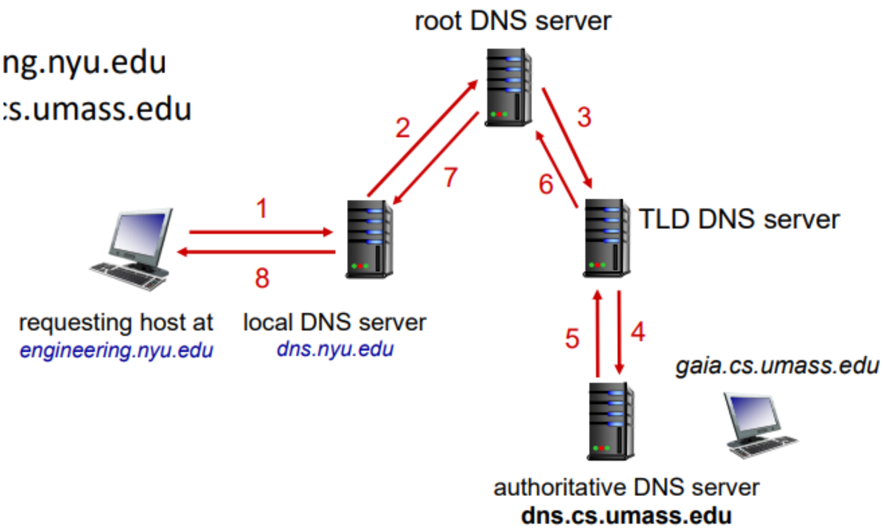
- **Top Level Domain and Authoritative Servers**
  <u>TLD / Top-Level Domain servers</u>: responsible for `.com`, `.org`, `.net`, `.edu`, `.aero`, `.jobs`, `.museums`, and all top-level country domains, e.g., `.cn`, `.uk`, `.fr`, `.ca`, `.jp`.
  <u>Authoritative DNS servers</u>: organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts; can be maintained by organization or service provider
- **Local DNS name servers**: local cache of recent name-to-address translation pairs (or forwarding request into DNS hierarchy for resolution)
  each ISP has local DNS name server: `ipconfig /all`
  cache entries timeout after some time (TTL) and could be out-of-date

1. iterated and recursive query

- Iterated query



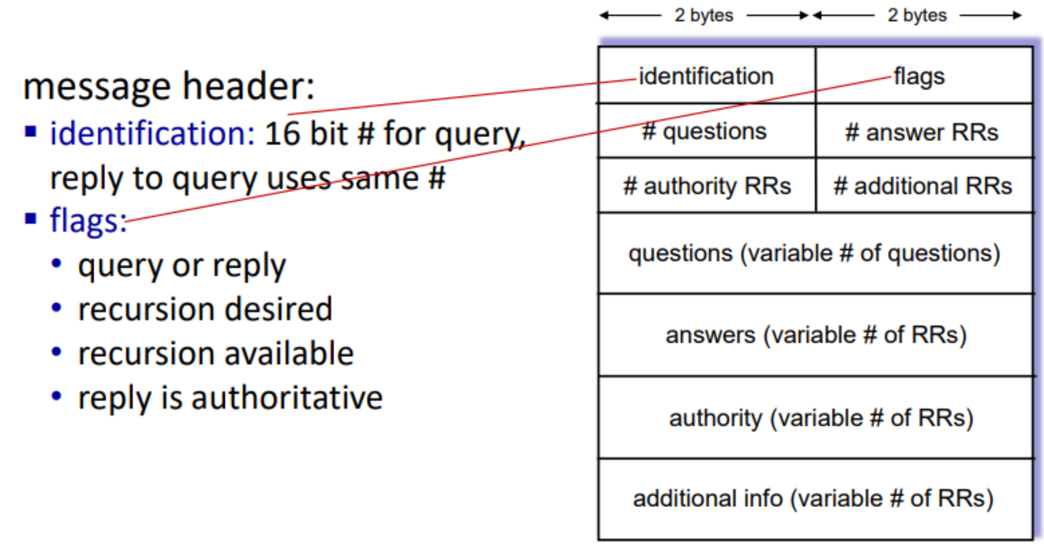
- Recursive query

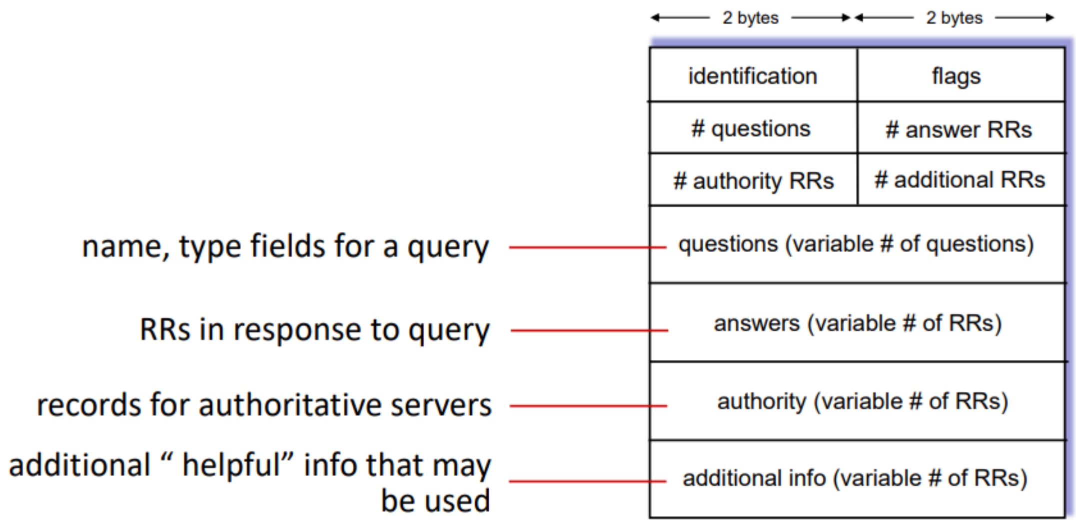


2. DNS records and protocols

> DNS: distributed database storing resource records **(RR)**
>
> $$RR\ format: (name, value, type, ttl)$$

- `type`
  A: `name`=hostname, `value`=IP address
  NS: `name`=domain (e.g., foo.com), `value`=hostname of authoritative name server for this domain
  CNAME: `name`=alias name for some “canonical” (the real) name, `value`=canonical name
  MX: `value`=canonical name of SMTP mail server associated with alias hostname `name`
- **protocols**
  DNS query and reply messages have same format:

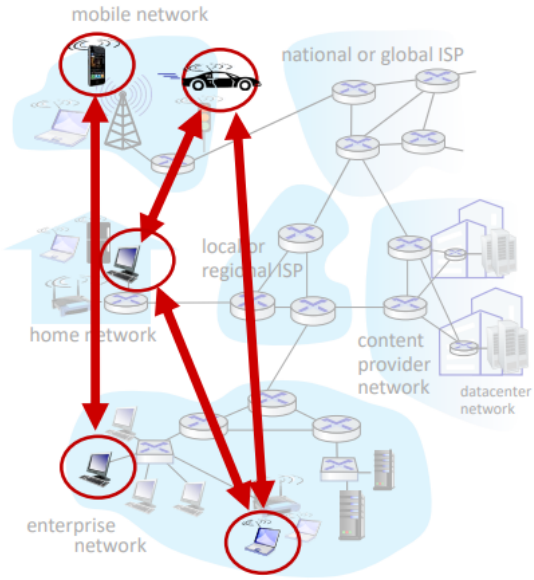




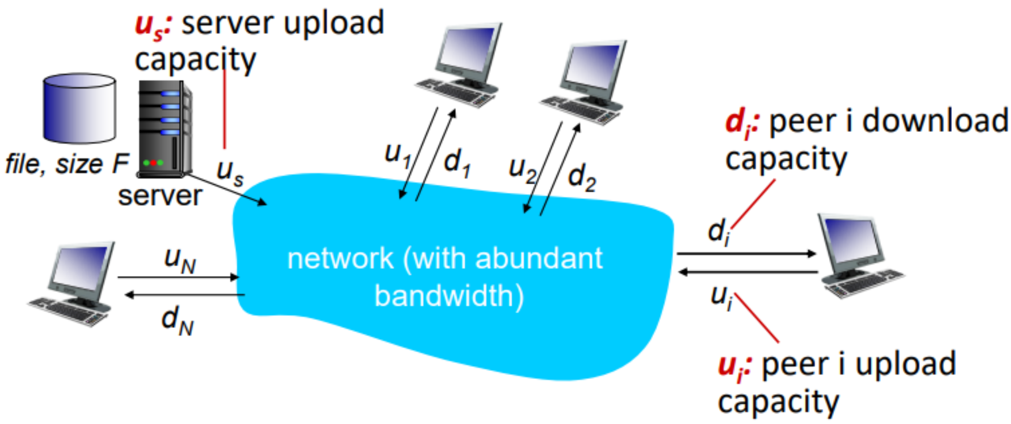
2.3 Other Applications

2.3.1 P2P Applications

NO always-on server
arbitrary end systems directly communicate
Example: P2P file sharing (BitTorrent), streaming (KanKan), VoIP (Skype)



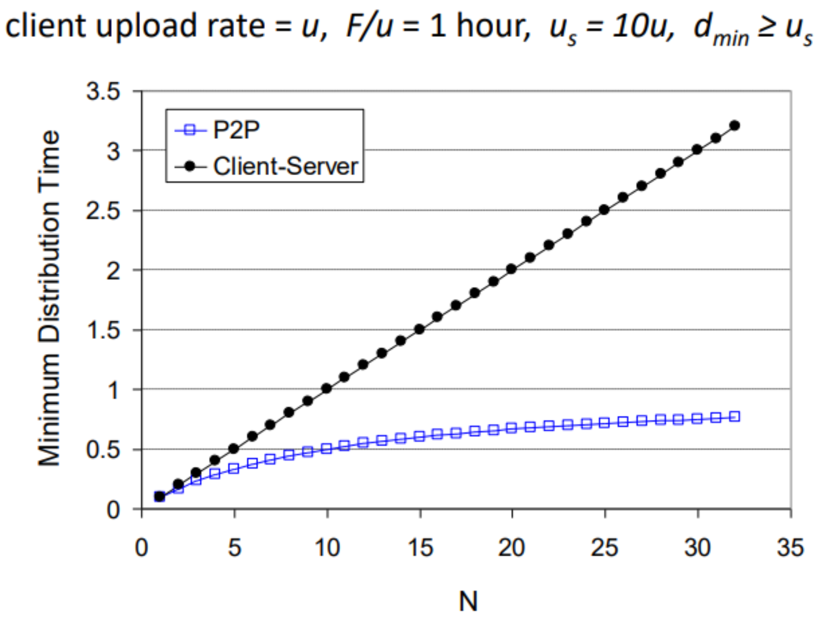
- **client-server file distribution time**
  Notation: N file copies, d_{min} = min client download rate
  the time needs increase linearly in N



$$D_{c-s} \geq max\{\frac{NF}{u_s}, \frac{F}{d_{min}}\}$$

- **P2P file distribution time**

$$F_{P2P} \geq max\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum u_i}\}$$

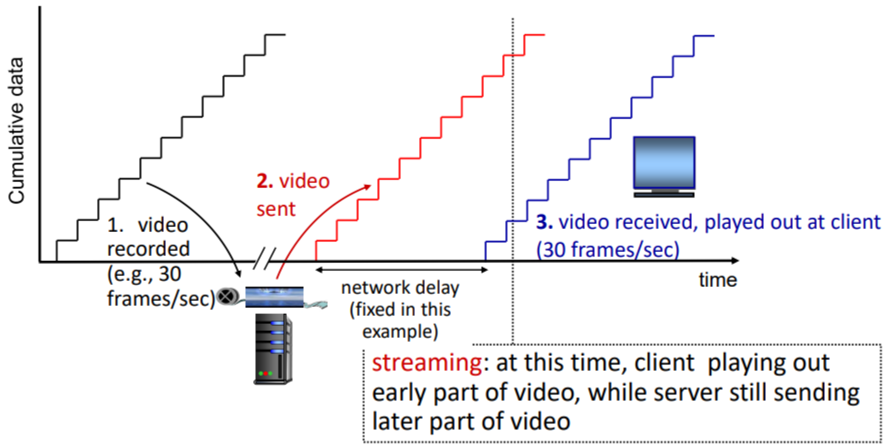


2.3.2 video streaming && content distribution networks

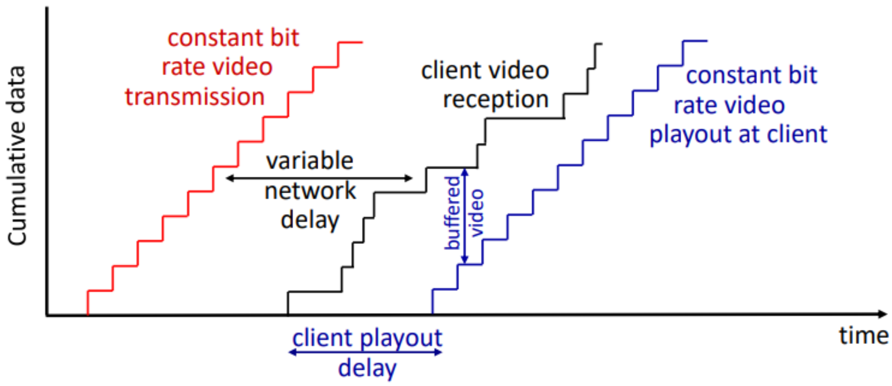
1. **Multimedia: video**
   coding: use redundancy within and between images to decrease bits used to encode image
      spatial (within image)
      temporal (from one image to next)
   measure: CBR (constant bit rate), VBR (variable bit rate)
2. **Streaming stored video**

> Streaming video = encoding + DASH + playout buffering

streaming: at this time, client playing out early part of video, while server still sending later part of video



playout buffering: compensate for network-added delay, delay jitter



DASH / Dynamix Adaptive Streaming over HTTP: clients determine things

3. **Content distribution networks / CDNs**
   store/serve multiple copies of videos at multiple geographically distributed sites: enable streaming content to numerous users simultaneously

   - *enter deep:* push CDN servers deep into many access networks
     - close to users
     - Akamai: 240,000 servers deployed in > 120 countries (2015)
   - *bring home:* smaller number (10's) of larger clusters in POPs near access nets
     - used by Limelight

Akamai

Limelight NETWORKS

2.4 Socket programming

1. Python UDP

- UDP client

Python UDPClient

include Python's socket library → `from socket import *`

```
serverName = 'hostname'
serverPort = 12000
```

create UDP socket for server → `clientSocket = socket(AF_INET, SOCK_DGRAM)`

get user keyboard input → `message = raw_input('Input lowercase sentence:')`

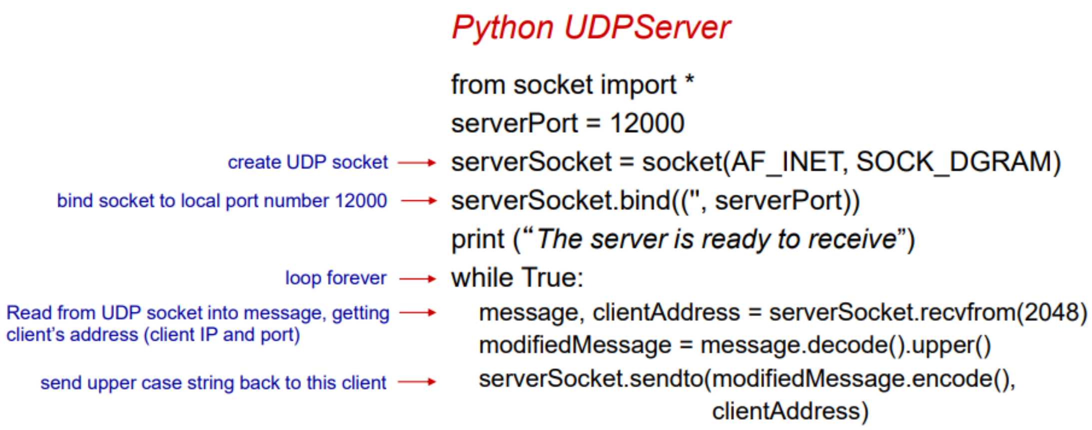
attach server name, port to message; send into socket → `clientSocket.sendto(message.encode(), (serverName, serverPort))`

read reply characters from socket into string → `modifiedMessage, serverAddress = clientSocket.recvfrom(2048)`

print out received string and close socket → `print modifiedMessage.decode()`

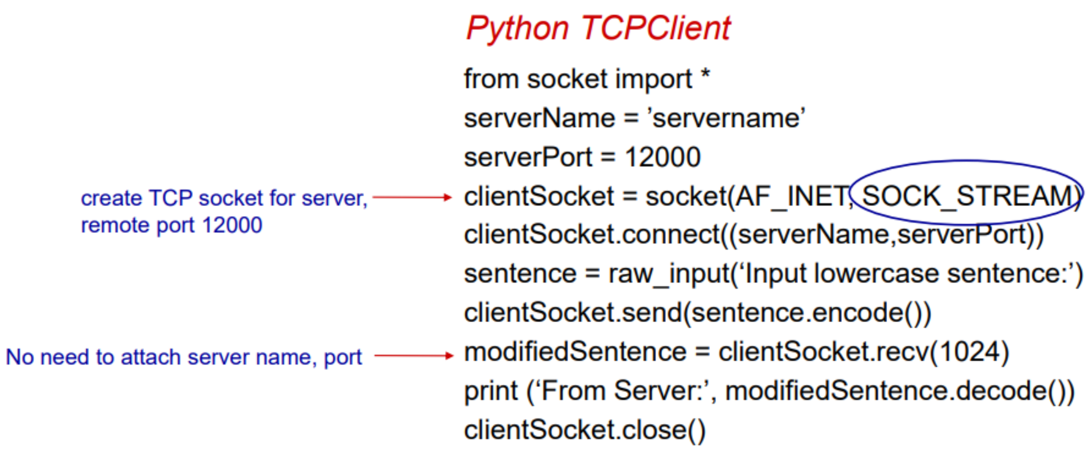
```
clientSocket.close()
```

- UDP server

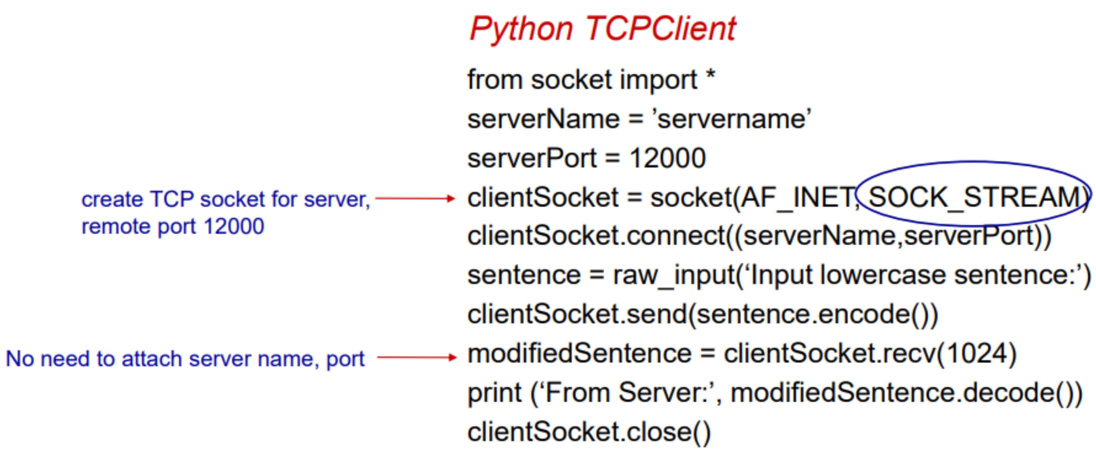


2. Python TCP

- TCP client



- TCP server



Chapter 3: Transport Layer

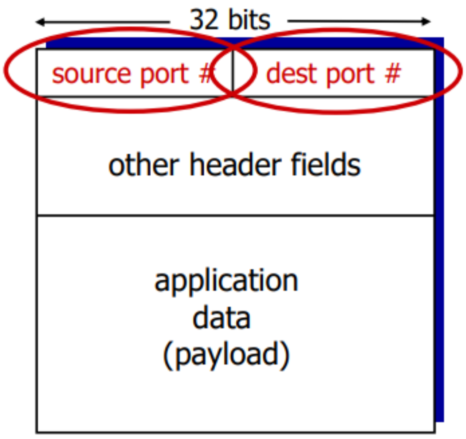
<u>Key Functions</u>: 把Network Layer提供的IP to IP和host to host的网络连接转换成Application Layer的process to process间的管道pipe, so transport layer is about **communication between processes** while network layer is about communication between hosts

<u>Key Components</u>: sender IP, sender Port, destination IP, destination Port

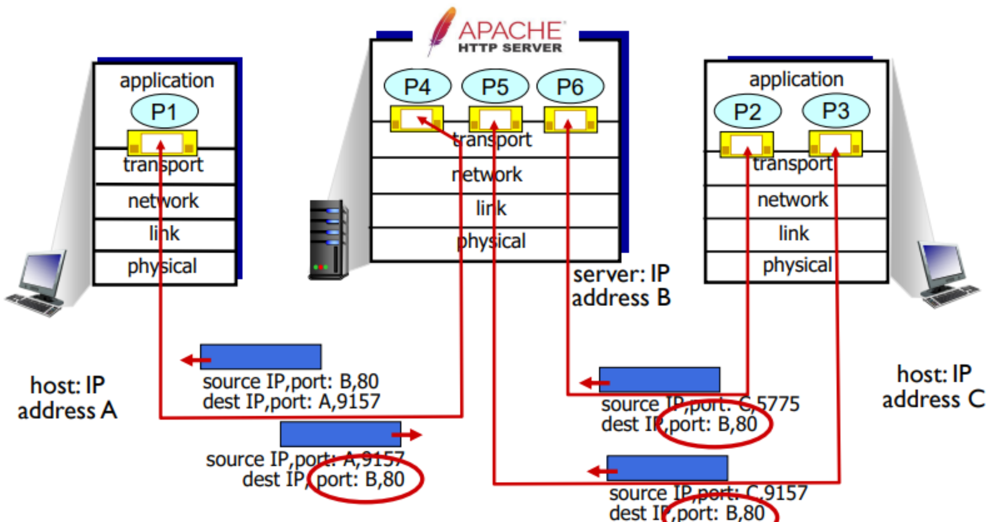
<u>Principal Protocols</u>: TCP / Transmission Control Protocol, UDP / User Datagram Protocol

3.1 Basic Functions and UDP

3.1.1 Multiplexing and Demultiplexing

> Multiplexing: (sender) handle data from multiple sockets, add transport header (later used for demultiplexing)
> Demultiplexing: (receiver) use header info to deliver received segments to correct socket （将网卡接到的packet分给不同的processes, transport layer's duty)
>
> 
>
>
> TCP/UDP segment format
> TCP/UDP是操作系统的一部分
> data link是网卡的一部分

- **IP => Processes / Ports => Sockets**
  one process can be demultiplexed to **different** sockets (ports)



Three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets

一个端口只能被一个进程监听。服务器进程监听`80`端口（default HTTP server port）建立连接的请求，建立连接时新建一个socket，然后分配一个worker process去通过socket和客户端交互，这里APACHE server的`P4`，`P5`，`P6`就代表3个worker process。客户端给服务端发送packet的时候dest port都是`80`，服务端的tansport layer将他们demultiplexed给不同的sockets（进程）。

注意，这里的`80`是固定端口号，只在需要监听的时候有用，只适用于服务端。

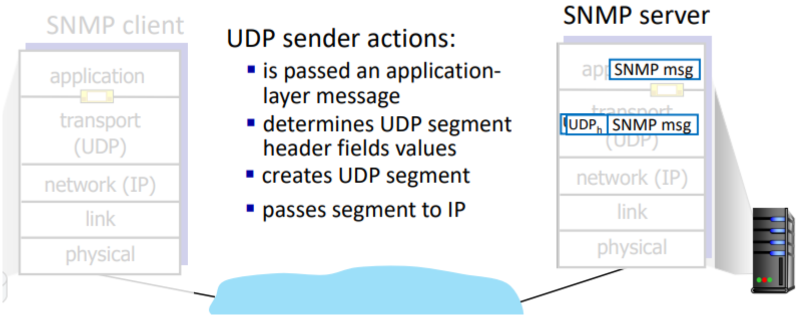
- **TCP vs UDP in demultiplexing**
  TCP: using 4-tuple: source and destination IP addresses + port numbers
  UDP: using destination port number (only)

3.1.2 UDP: connectionless transport

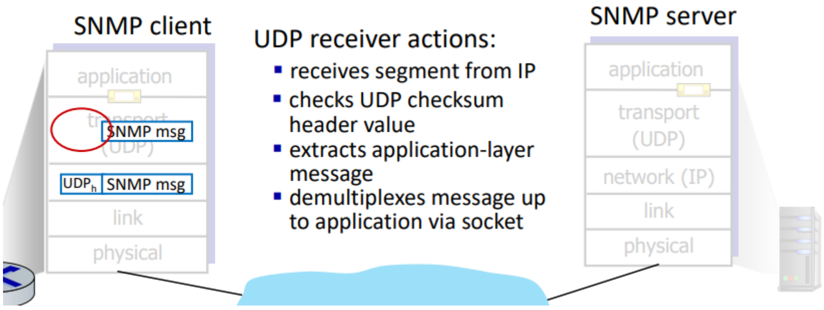
> Connectionless: no handshaking between UDP sender and receiver; each UDP segment handled independently of others
>
> Strength of UDP:
>
> no connection establishment => decrease an RTT delay
> simple: no connection state at sender and receiver
> small header size
> no congestion control => fast!
> **can add needed congestion control and reliability at application layer (HTTP/3)**
>
> Application: streaming multimedia apps, DNS, SNMP (Simple Network Management Protocol), HTTP/3

1. Procedure

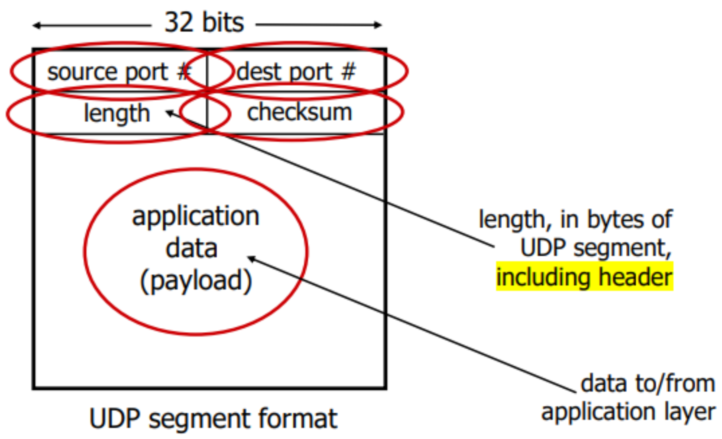
- **UDP sender actions**
  is passed an application
  determines UDP segment header fields values
  creates UDP segment
  passes segment to IP



- **UDP receriver actions**
  receives segment from IP
  checks UDP checksum header value
  extracts application-layer message
  demultiplexes message up to application via socket



2. UDP segement header



- **UDP checksum**
  Goal: detect errors (i.e., flipped bits) in transmitted segment
  Formula: addition (one's complement sum) of segment content (including UDP header fields and IP addresses, as sequence of 16-bit integers)

example: add two 16-bit integers

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

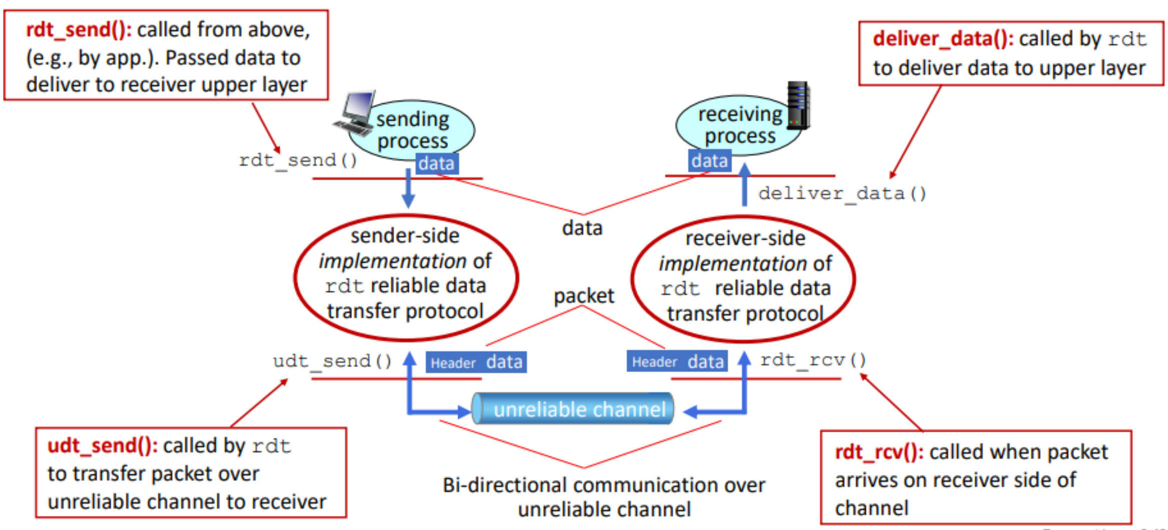
Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

one's complement sum: invert all the bits

3.2 Advanced Functions and TCP

3.2.1 Reliable Data Transfer

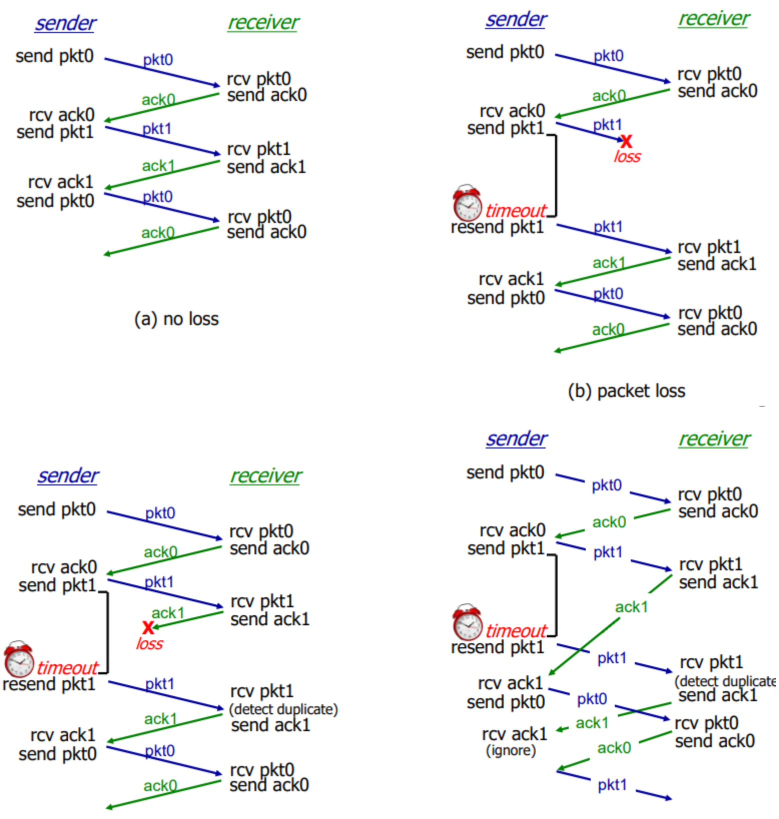
1. interfaces



2. develop rdt protocol

- **rdt1.0**: reliable transfer over a reliable channel
  underlying channel perfectly reliable: no bit errors + no loss of packets

- **rdt2.0**: channel with bit errors
  use checksum to detect bit errors
  - **stop-and-wait FSM / Finite State Machine**
    `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
  - **rdt2.1**: add **sequence number** to each pkt: two seq. #s (0, 1) would suffice
  - **rdt2.2** (NAK-free): same functionality as rdt2.1, using ACKs **only**
- **rdt3.0**: channel with errors and loss
  Approach: sender waits reasonable amount of time for ACK



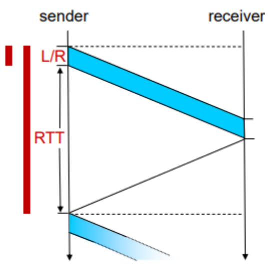
(a) no loss
(b) packet loss
(c) ACK loss
(d) premature timeout/ delayed ACK

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

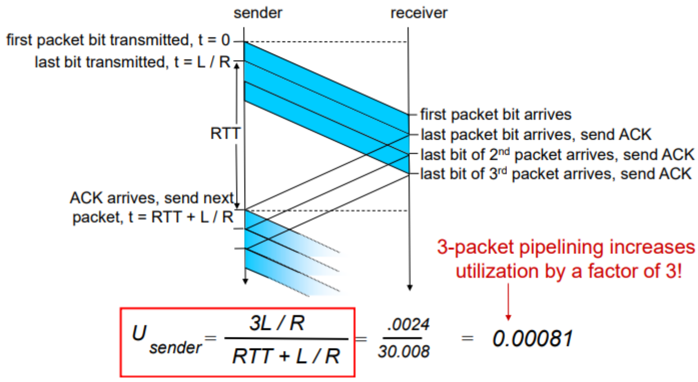
$$\boxed{U_{sender} = \frac{L / R}{RTT + L / R}}$$

$$= \frac{.008}{30.008}$$

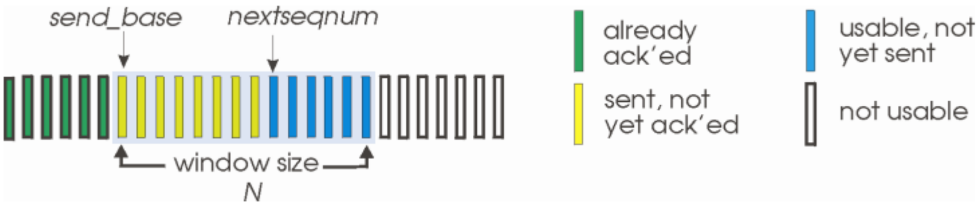
$$= 0.00027$$



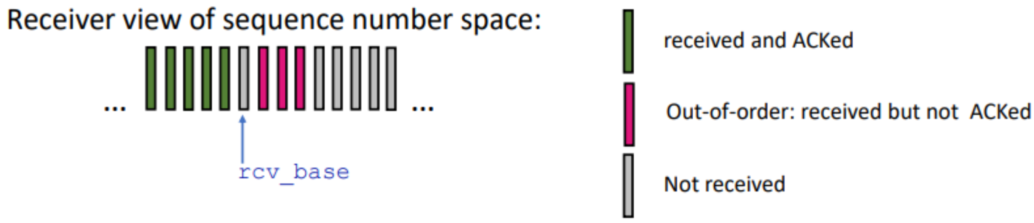
——**pipelining**: increase utilization



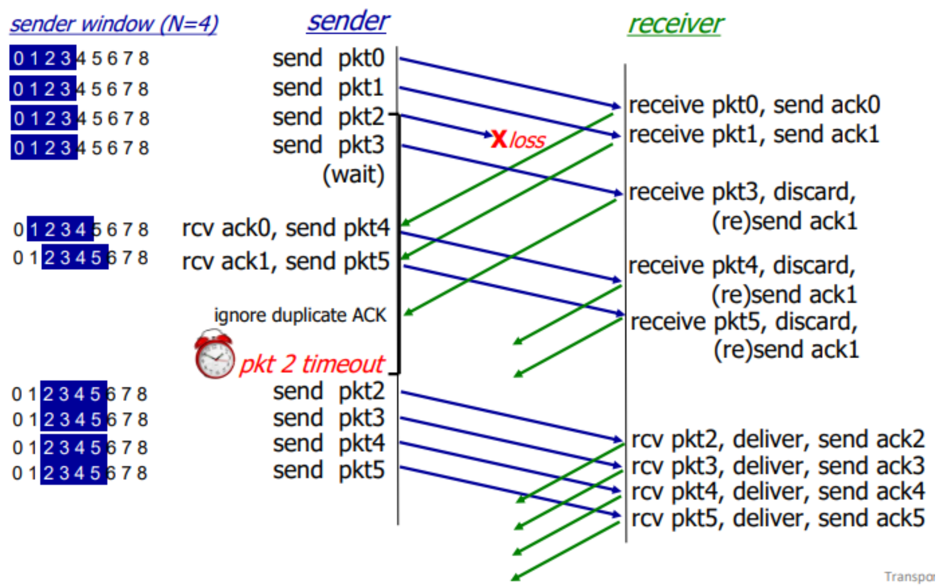
- **Go-Back-N**: TCP sliding window, 有包被丢掉的时候效率很低, alternative: ARMD
  sender: "window" of up to N
  `cumulative ACK` - ACK(n): ACKs all pkt up to, including seq # n
  - k-bit seq # in pkt header



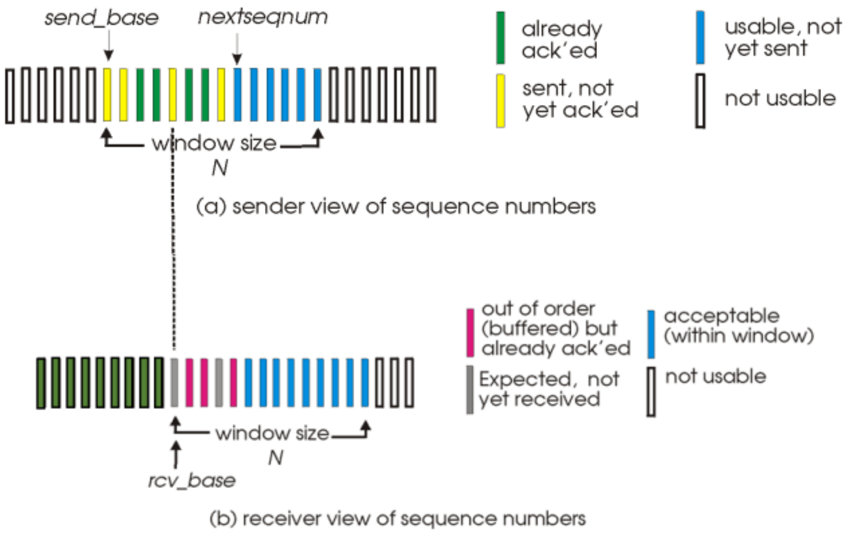
receiver: always send ACK for correctly-received pkt so far, with highest in-order seq #



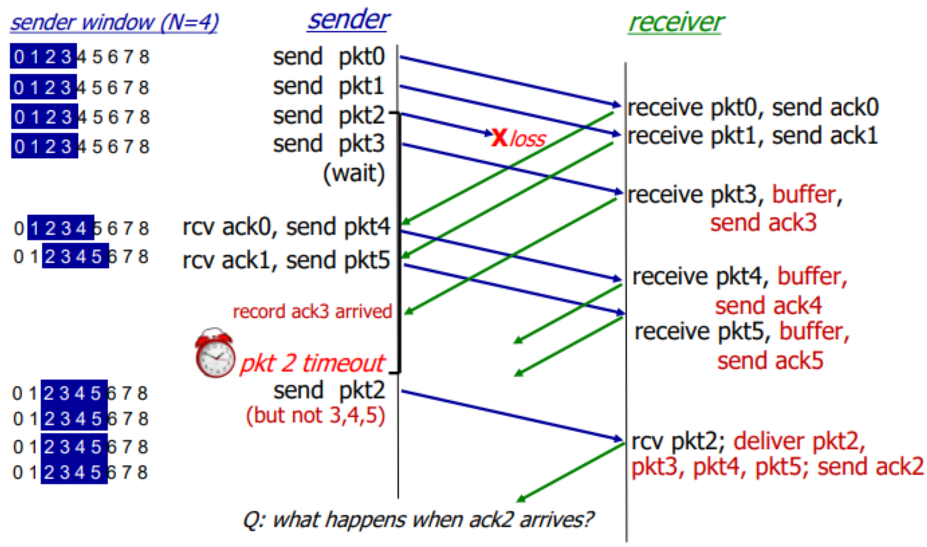
in action:



- **Selective Repeat**: 并不是任何情况下性能都比Go-Back-N方法更好，比如只丢一个ack
  sender and receiver:

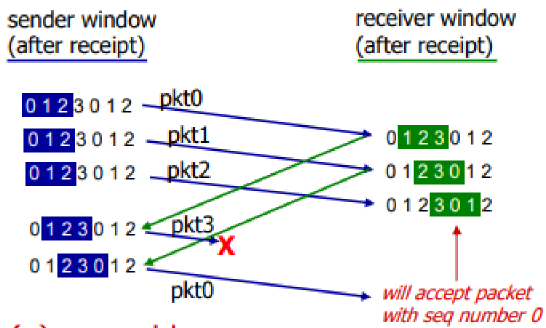


in action:

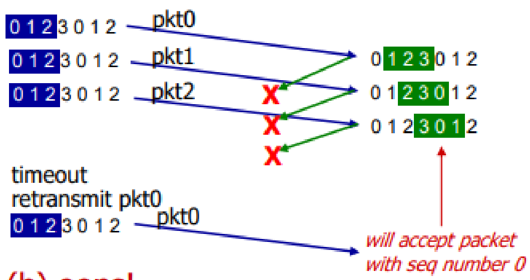


problem: relationship between **sequence # size** and **window size**

Example: seq #s: 0, 1, 2, 3 (base 4 counting) —— identical cases from receiver's perspective



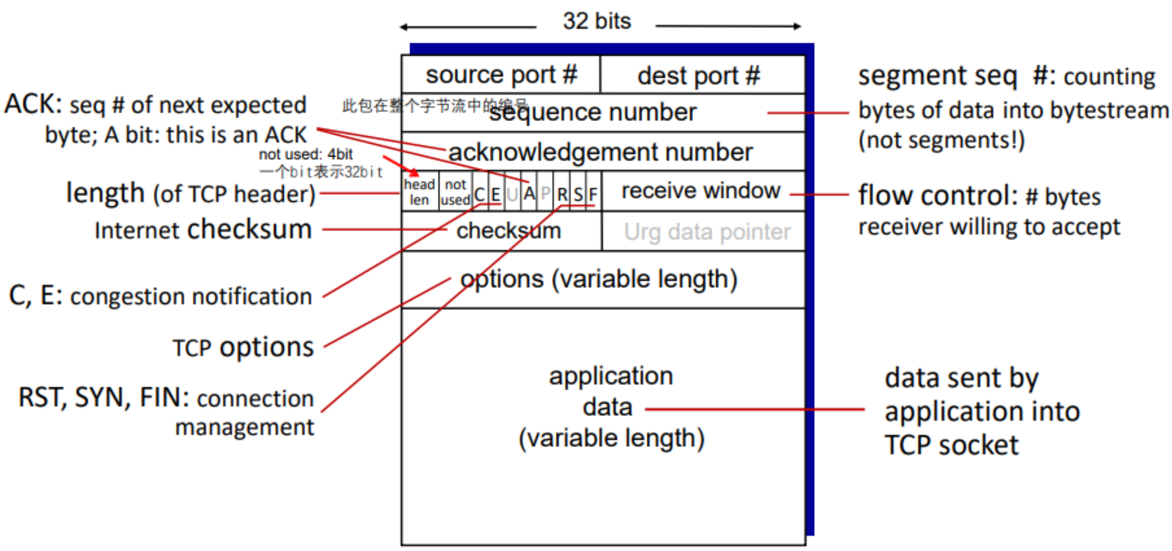
(a) no problem



(b) oops!

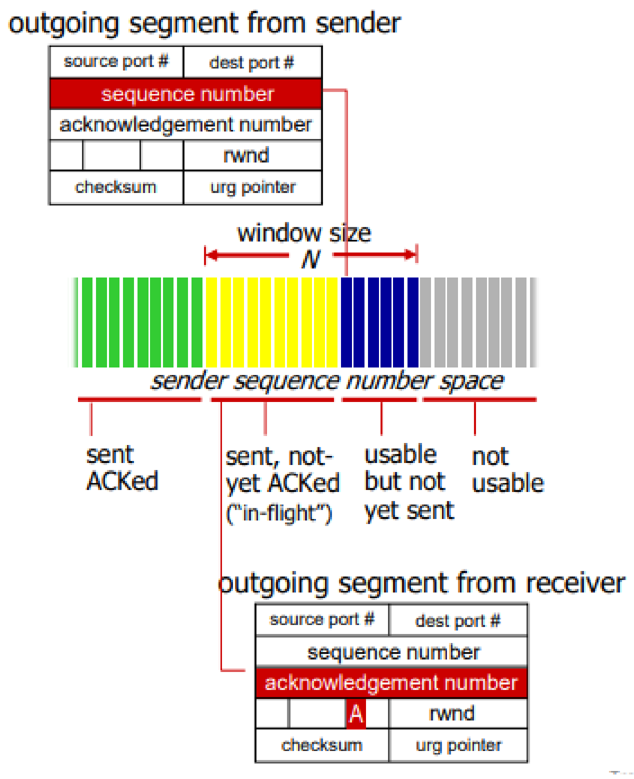
3.2.2 TCP: connection-oriented

TCP segement structure

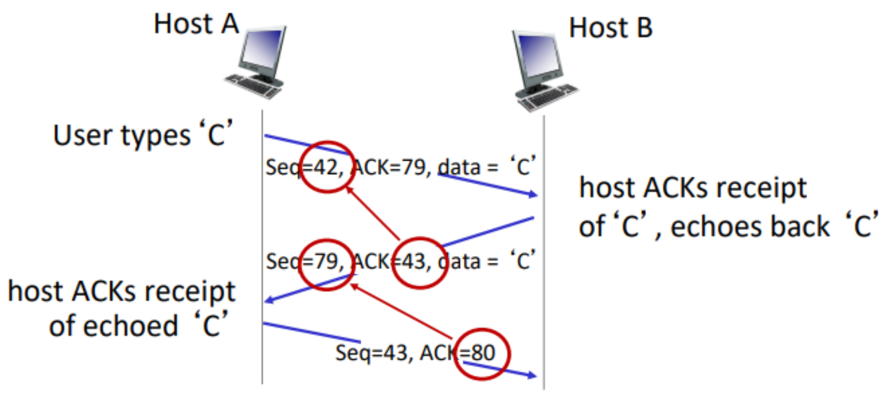


1. TCP reliable data transfer

- **sequence numbers and ACKs**
  sequence numbers: byte stream "number" of first byte in segment's data
  acknowledgements: seq # of next byte expected from other side; cumulative ACK



- simple telnet scenario:



- **RTT and timeout**

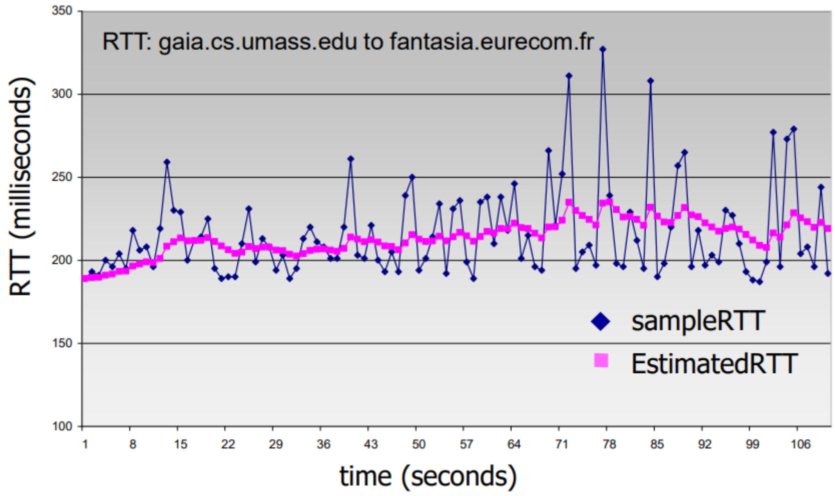
  at least, `timeout` should be longer than RTT, but shouldn't be too long (slow reaction)

  - EstimatedRTT

    $SampleRTT$: average several recent RTTs measurements

    $$EstimatedRTT = (1-\alpha) \times EstimatedRTT_{-1} + \alpha \times SampleRTT$$
    $$(typically\ \alpha = 0.125)$$

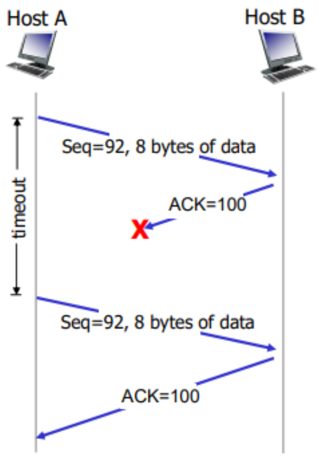
    ***EWMA** / Exponential Weighted Moving Average: influence of past sample decreases exponentially fast*



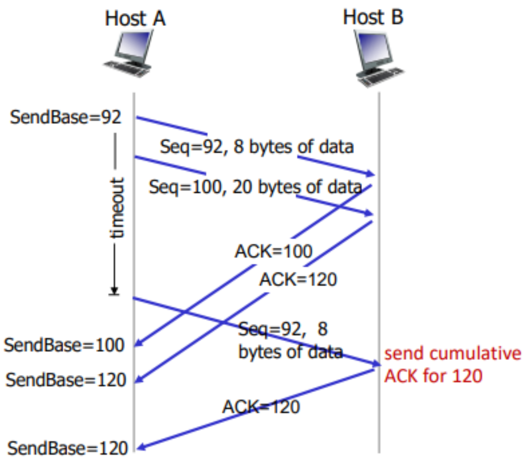
- TimeoutInterval

$$TimeoutInterval = EstimatedRTT + 4 \times DevRTT \ (safety\ margin)$$
$$, where\ DevRTT = (1-\beta) \times DevRTT + \beta \times |SampleRTT - EstimatedRTT|$$
$$(typically\ \beta = 0.25)$$

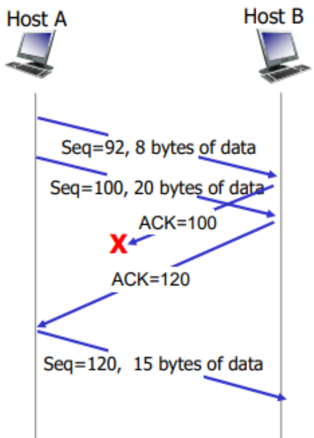
- **in action**



lost ACK scenario

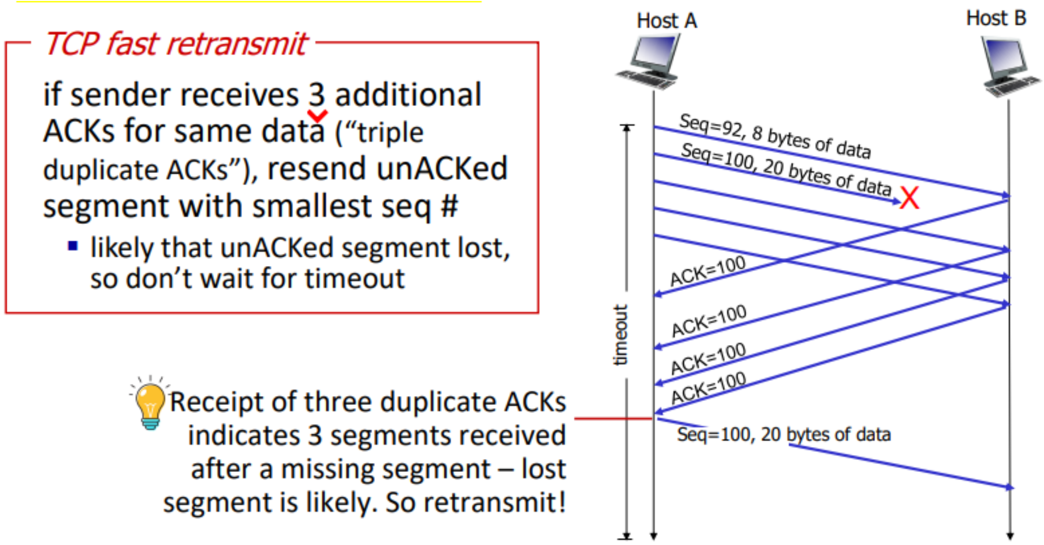


premature timeout

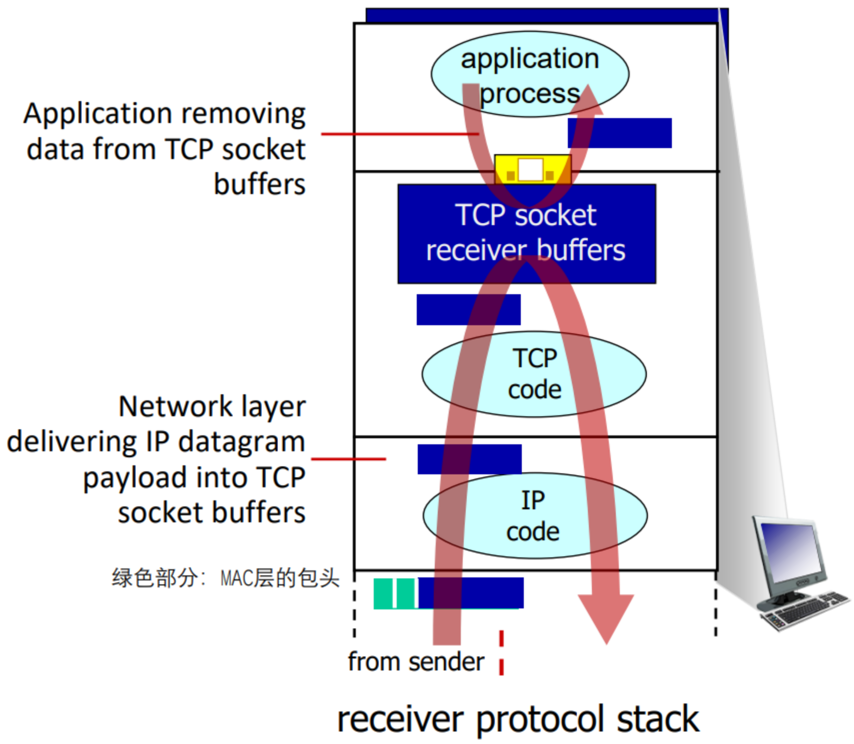


cumulative ACK covers
for earlier lost ACK

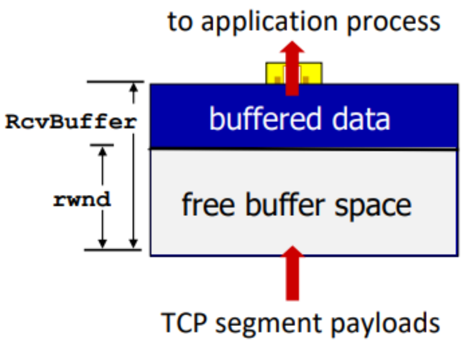
TCP fast retransmit:
If sender receives 3 additional ACKs for same data (“triple duplicate ACKs”), resend unACKed segment with smallest seq #
—— likely that unACKed segment lost, so don’t wait for timeout



2. TCP flow control

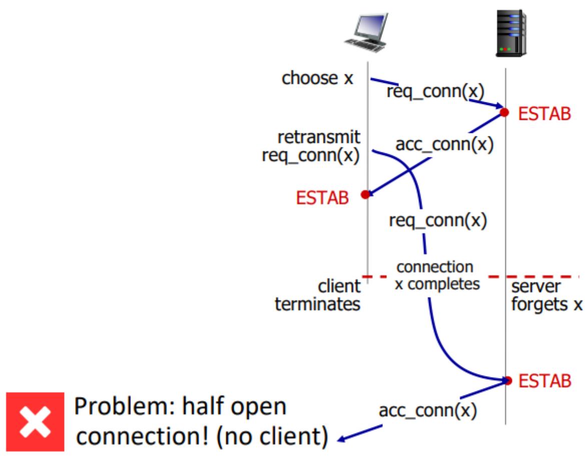
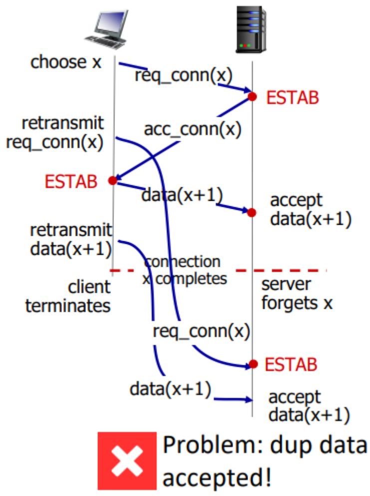
> **Problem**: What happens if network layer delivers data faster than application layer removes data from socket buffers?
>
> 
>
>
> Flow Control: receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

- **rwnd field**
  TCP receiver "advertises" free buffer space in **rwnd** field in TCP header "receive window"
  RcvBuffer size set via socket options (typical default is 4096 bytes)
  sender limites amount of unACKed ("in-flight") data to received **rwnd**

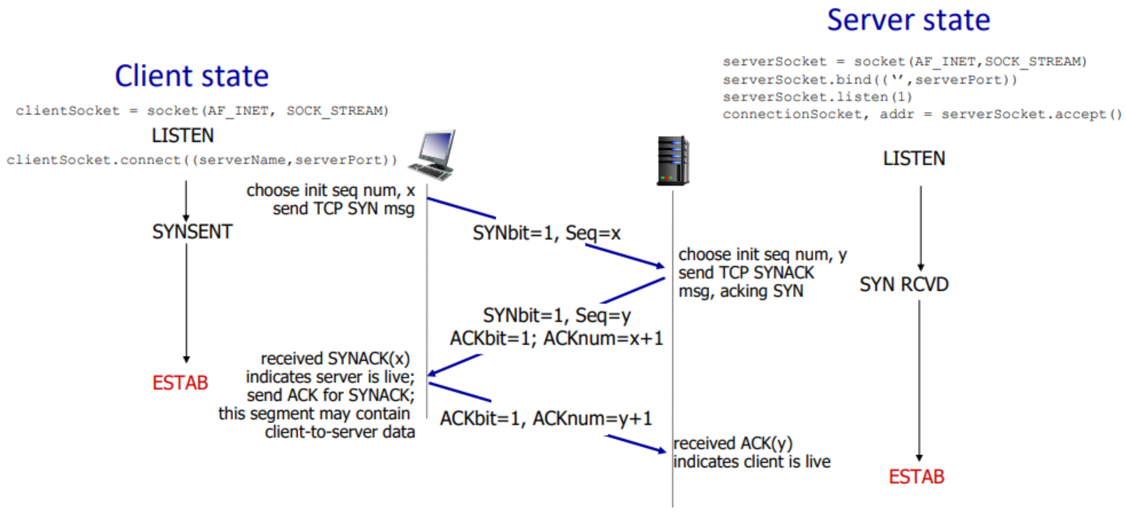


TCP receiver-side buffering

3. TCP 3-way handshake

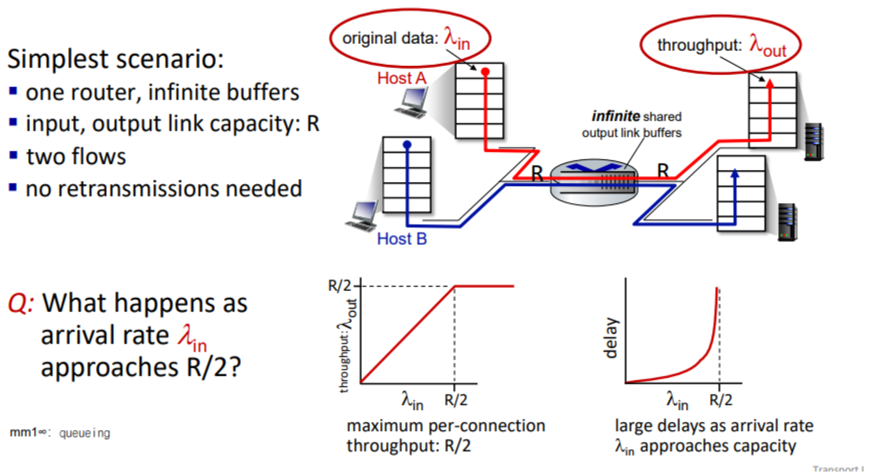
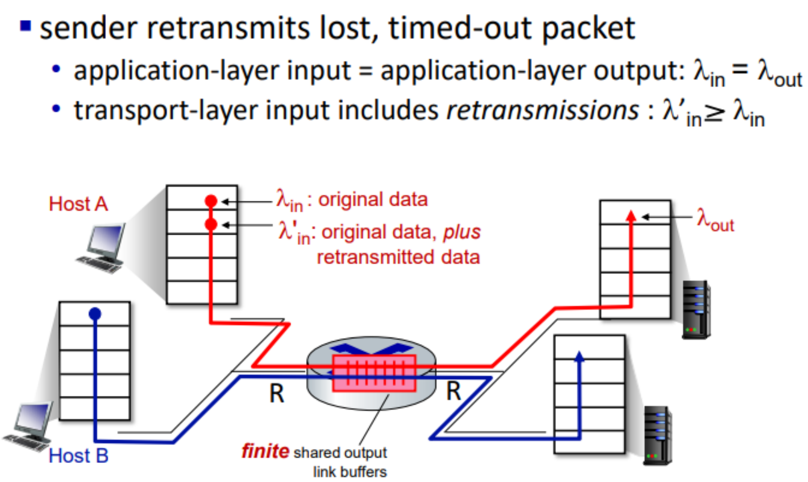
> problem with 2-way handshake:
>
> 
>
>
> 
>

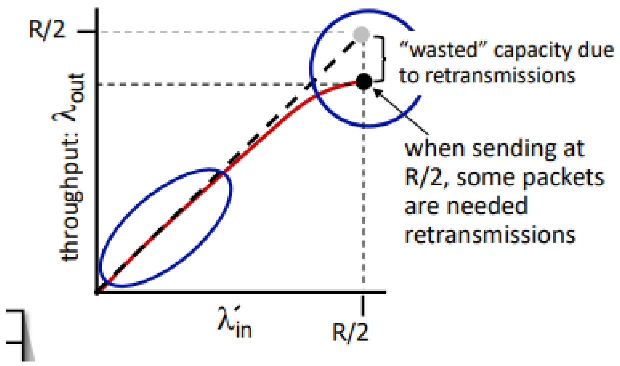
- **TCP connection management: initialize a request**
  client: send a connection request via `SYNbit=1`, `Seq=x`
  server: ACK client's request via `SYNbit=1`, `ACKnum=x+1`, `ACKbit=1`, `Seq=y`
  client: ACK server's ACK and may contain data via `ACKnum=y+1`, `ACKbit=1`



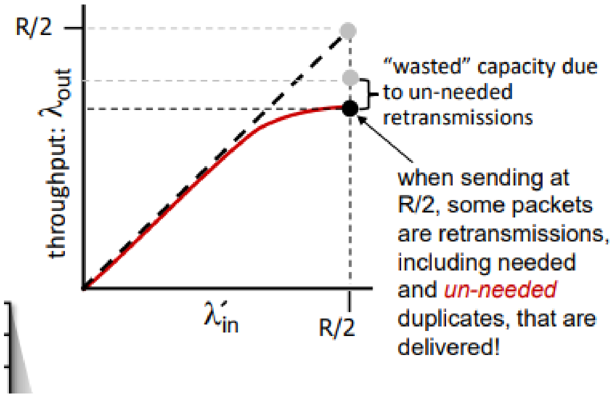
- **TCP connection management: close a connection**
  Side A: send TCP segment with `FINbit=1`
  Side B: ACK A's request via `FINbit=1`, `ACKnum=x+1`, `ACKbit=1`
  Side A: ACK B's ACK via `ACKnum=y+1`, `ACKbit=1`

4. TCP congestion control

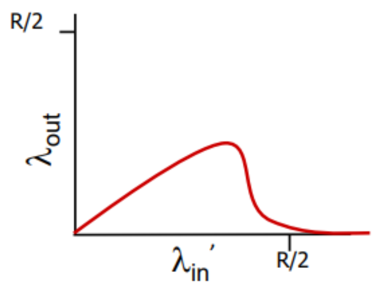
> Different from flow control:
>
> flow control is about one sender sending too fast to one receiver
> congestion control is about too many senders sending too fast
>
> Scenario 1: infinit router buffers
>
> 
>
>
> Scenario 2: finite router buffers
>
> 
>
>
> - lost packets => retransmission

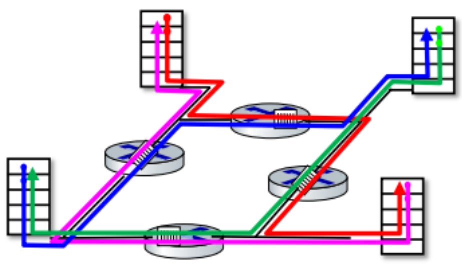


- suggested lost packets => un-needed duplicates

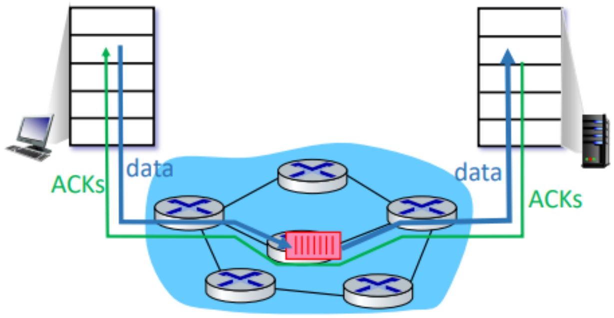


Scenario 3: four senders, multi-hop paths, timeout/retransmit
when packet dropped, any upstream transmission capacity and buffering used for that packet was wasted

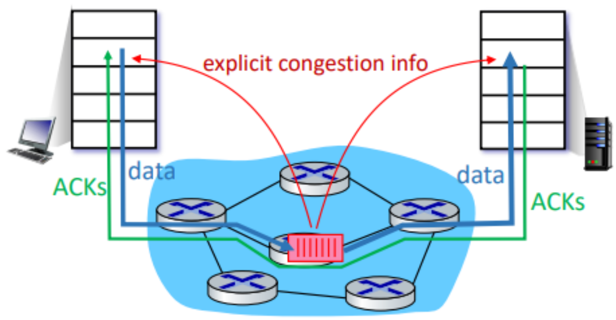




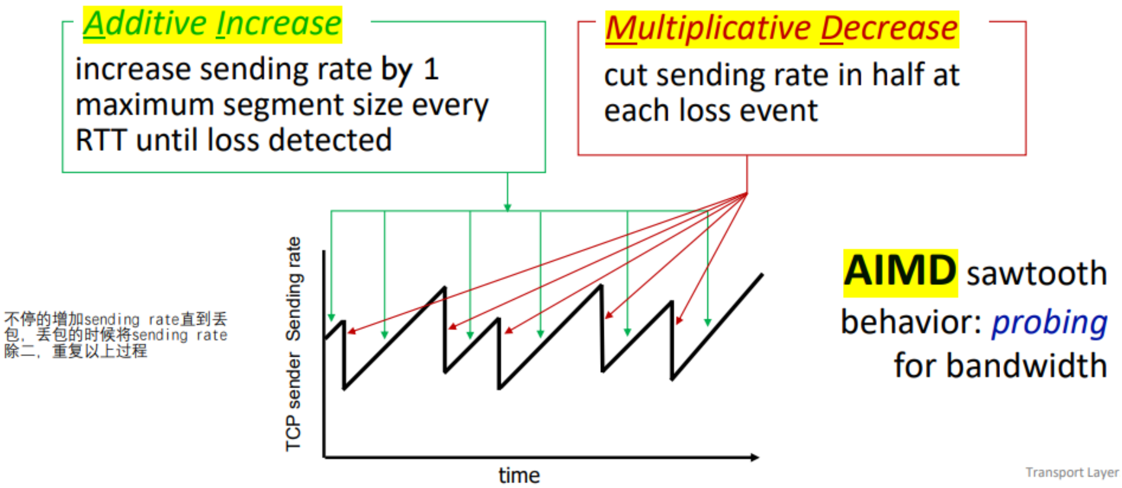
- **congestion problems**
  throughput can never exceed capacity
  delay increases as capacity approached
  loss/retransmission decreases effective throughput
  un-needed duplicates further decreases effective throughput
  upstream transmission capacity / buffering wasted for packets lost downstream
- **Approach 1: End-End congestion control**
  no explicit feedback from network
  congestion inferred from observed loss, delay
  approach taken by TCP



- **Approach 2: Network-assisted congestion control**
  routers provide direct feedback to sending/receiving hosts with flows passing through congested router
  may indicate congestion level or explicitly set sending rate
  TCP ECN, ATM, DECbit protocols

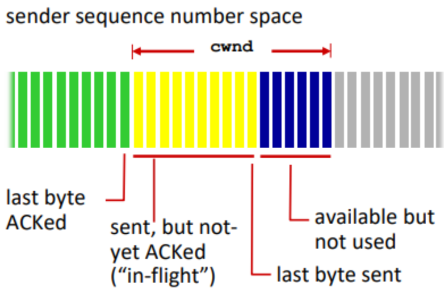


1. **TCP congestion control: AIMD**
   AIMD: Additive Increase Multiplicative Decrease
   Approach: senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event



Cut in half on loss detected by triple duplicate ACK **(TCP Reno)**（新的）
Cut to 1 MSS (maximum segment size) when loss detected by timeout **(TCP Tahoe)**（老的）

- **cwnd**: congestion window (number of bytes that a sender is allowed to transmit before it must receive an ACK from the receiver)



TCP sending behavior:

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

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

- TCP sender limits transmission: `LastByteSent- LastByteAcked ≤ cwnd`
- `cwnd` is dynamically adjusted in response to observed network congestion (implementing TCP congestion control)

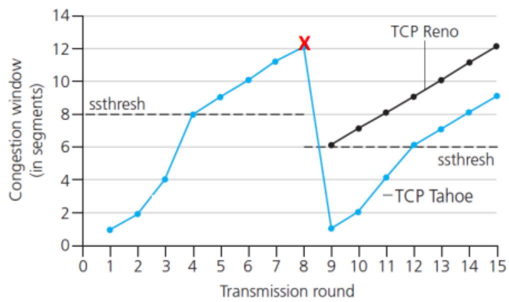
  - **TCP Reno**: quick start and fast recovery
    TCP Tahoe (original version): slow start but rate ramps up exponentially fast

Q: when should the exponential increase switch to linear?

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

Implementation:

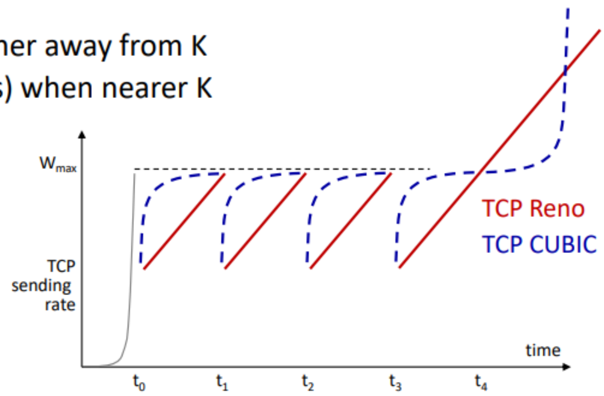
- variable **ssthresh**
- on loss event, **ssthresh** is set to 1/2 of **cwnd** just before loss event



2. **TCP congestion control: CUBIC**
   increase W as a function of the **cube** of the distance between current time and K (point in time when TCP window size will reach W_{max})

   - larger increases when further away from K
   - smaller increases (cautious) when nearer K

   - TCP CUBIC default in Linux, most popular TCP for popular Web servers



   <u>CUBIC</u>: TCP (classic, CUBIC) increase TCP's sending rate until packet loss occurs at some router's output: **the bottleneck link**

3. **TCP congestion control: delay-based**

$$measured\ througput = \frac{\#\ bytes\ sent\ in\ last\ RTT\ interval}{RTT_{measured}}$$

$$uncongested\ throughput = \frac{cwnd}{RTT_{min}}$$

Delay-based approach:

- RTT_{min} - minimum observed RTT (uncongested path)
- uncongested throughput with congestion window `cwnd` is $cwnd/RTT_{min}$

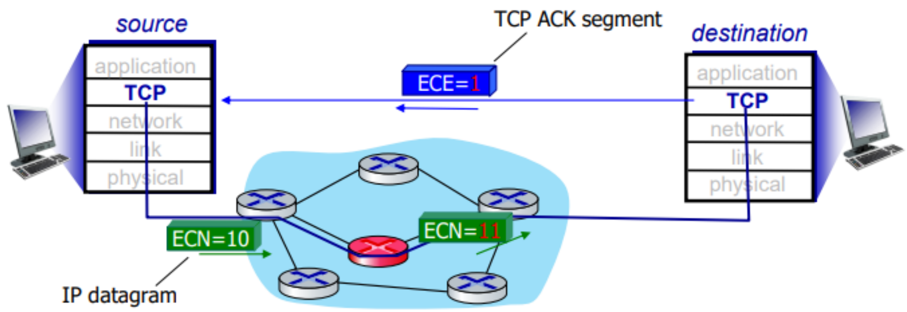
```
if measured throughput "very close" to uncongested throughput
    increase cwnd linearly          /* since path not congested */
else if measured throughput "far below" uncongested throughout
    decrease cwnd linearly          /* since path is congested */
```

4. **TCP congestion control: ECN**

   ECN: Explicit Congestion Notification

   1 two bits in IP header (ToS field) marked by network router to indicate congestion, and this congestion indication carried to destination

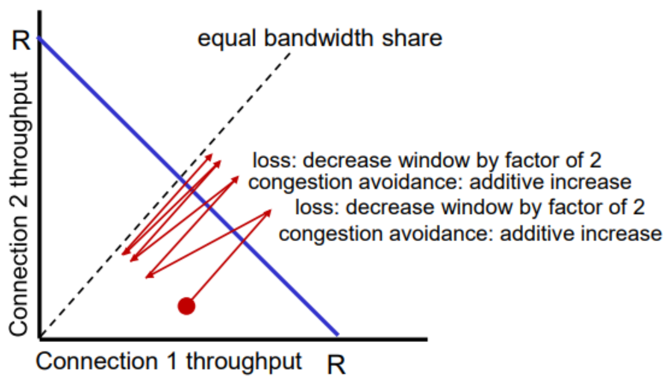
   2 destination sets `ECN` bit on ACK segment to notify sender of congestion, involving both IP (IP header `ECN` bit marking) and TCP (TCP header `C`, `E` bit marking)



5. **Discussion: is TCP fair?**

   TCP fairness: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of $\frac{R}{K}$

   Answer: YES, under assumptions that ① same RTT and ② fixed number of sessions only in congestion avoidance (or new TCP sessions would crowd out available bandwidth thus owning a larger share)



   Other solution: multiple parallel TCP connections between hosts

   TCP favors short slow (shorter RTT)

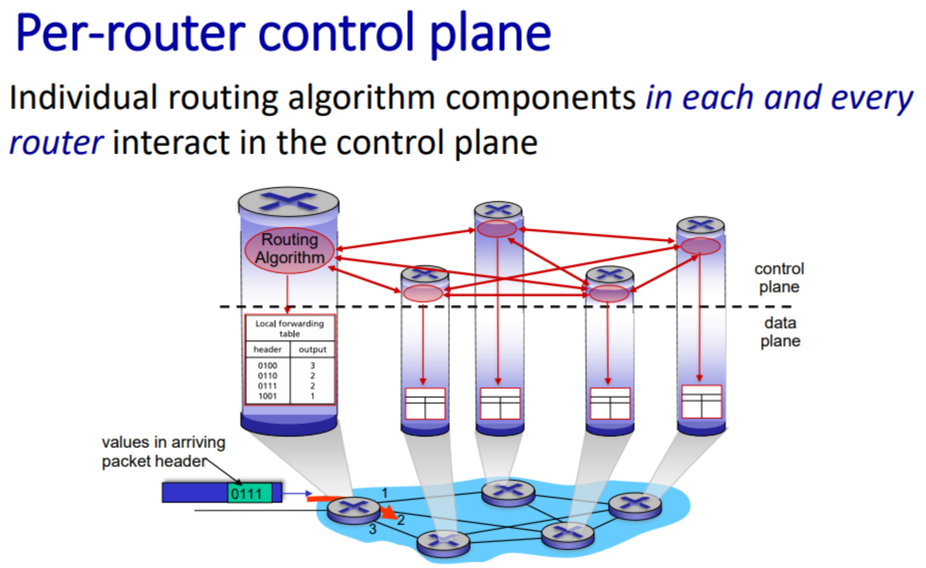
3.2.3 Evolution: QUIC

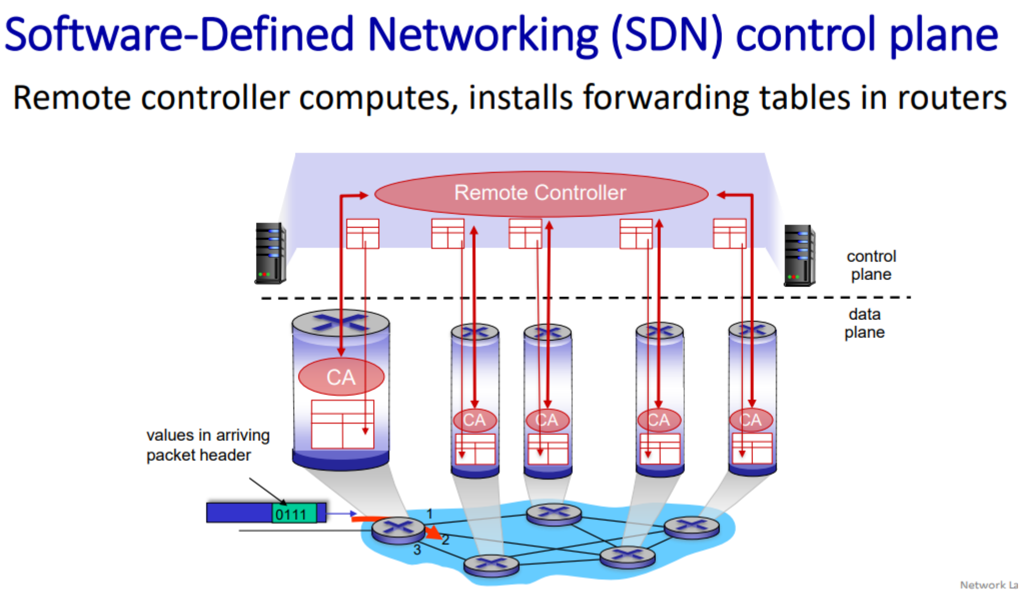
> HTTP/3: moving transport–layer functions to application layer, on top of UDP

Chapter 4: Network Layer: Data Plane

4.1 Overview

- Two key network-layer functions: **forwarding** and **routing**
- **Data Plane and Control Plane**
    - Data Plane: local, per-router function, determining how datagram arriving on router input port is forwarded to router output port
    - Control Plane: network-wide logic, determining how datagram is routed among routers along end-end path from source host to destination host
      traditional routing algorithms: implemented in routers
      software-defined networking (SDN): implemented in (remote) servers





> The **Controller Agent (CA)** acts as an intermediary between the controller and the network devices. It is responsible for translating the policies and commands defined by the controller into configurations that can be executed by the network devices.

- **Network Service Model**
  example services for individual datagrams: rdt, delivery within 40 msec delay
  example services for a flow of datagrams: in-order, minimum bandwidth guarantted, restictions on changes in inter-packet spacing

Internet "best effort" service model: most fundamental service model (no loss, order, timing guarantees)

other service model:

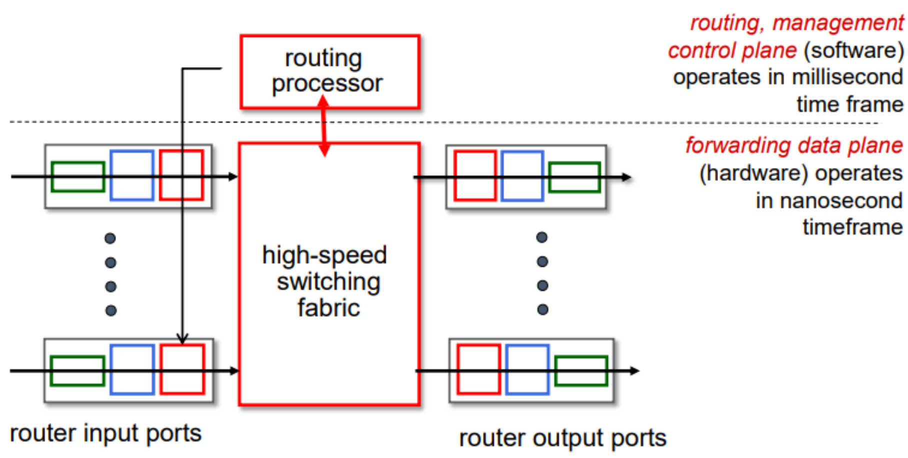
Network Architecture	Service Model	Quality of Service (QoS) Guarantees ?			
		Bandwidth	Loss	Order	Timing
Internet	best effort	none	no	no	no
ATM	Constant Bit Rate	Constant rate	yes	yes	yes
ATM	Available Bit Rate	Guaranteed min	no	yes	no
Internet	Intserv Guaranteed (RFC 1633)	yes	yes	yes	yes
Internet	Diffserv (RFC 2475)	possible	possibly	possibly	no

4.2 Router

> input ports, switching, output ports
> buffer management, scheduling

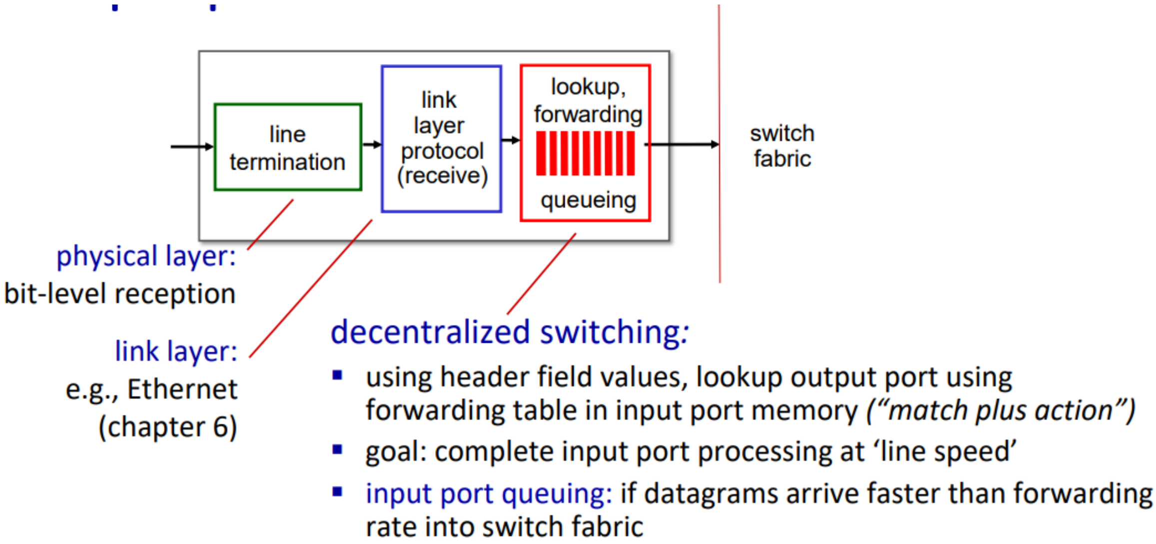
4.2.1 Router architecture overview

high-level view of generic router architecture:



4.2.2 Router architecture specifics

1. input port



- **destination-based forwarding**: forward based on destination IP address (traditional)

forwarding table

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

  Longest prefix match: when looking for forwarding table entry for given destination address, use longest address prefix that matches destination address.

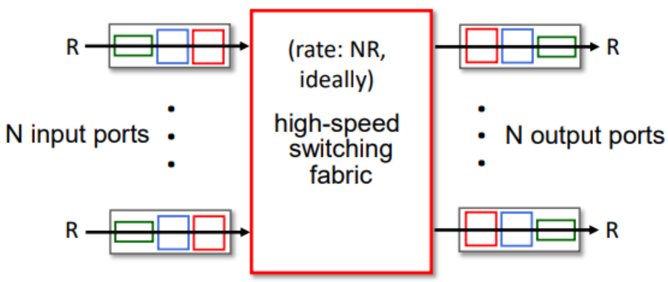
  TCAM / Ternary Content Addressable Memories: a type of high-speed memory used for performing the longest prefix matching algorithm. TCAMs allow the router to compare the incoming packet's destination address to multiple routing table entries simultaneously, enabling very fast routing table lookups.

  - **content addressable**: present address to TCAM and retrieve address in one clock cycle, regardless of table size
  - **Cisco Catalyst** (a brand of network swtiches): about 1M routing table entries in TCAM
- **generalized forwarding**: forward based on any set of header field values

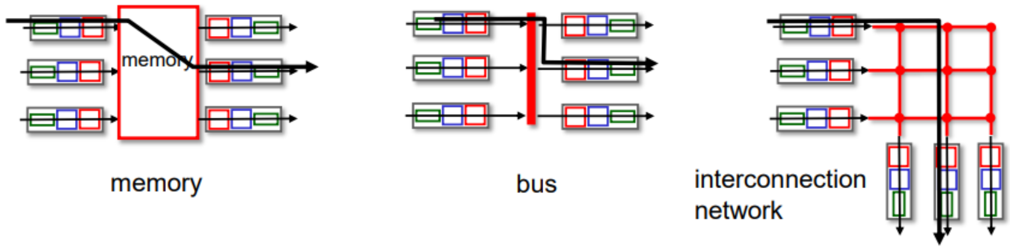
2. switching fabrics

- **switching rate**: rate at which packets can be transfer from inputs to outputs

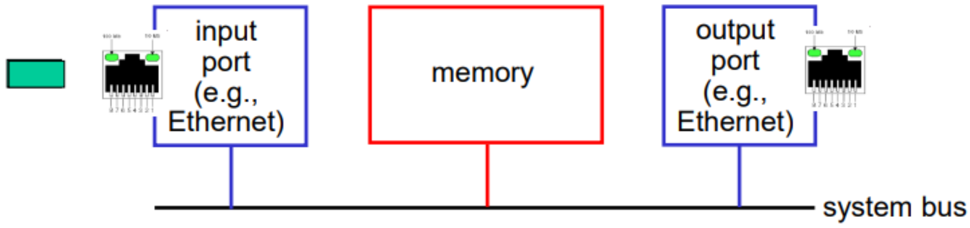
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable



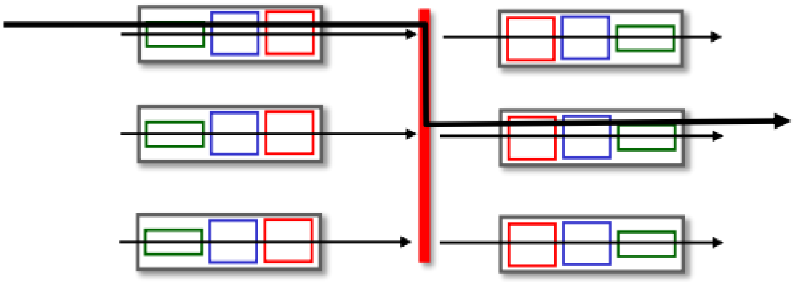
- **3 types of switching fabrics**: memory, bus, interconnection network



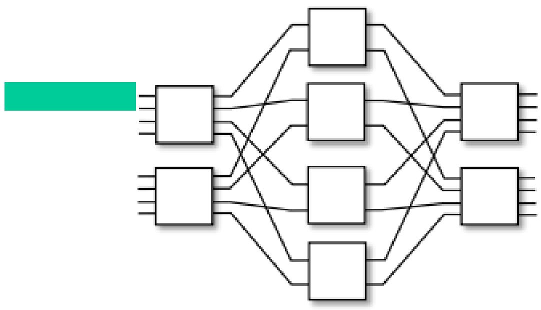
  - switching via memory
    first generation routers:
        switching under direct control of CPU
        packet copied to system's memory
        speed limited by memory bandwidth (2 bus crossings per datagram)



  - switching via a bus
    bus contention: switching speed limited by bus bandwidth

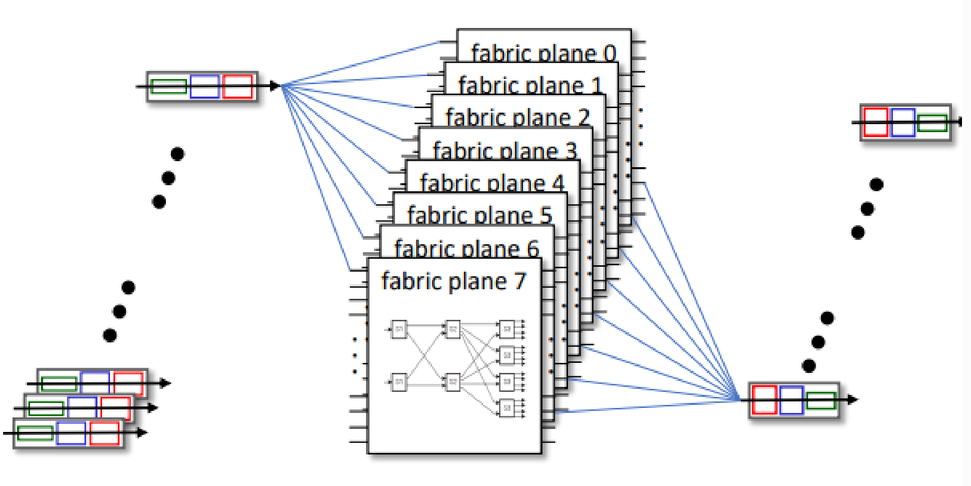


  - switching via interconnection network
    multistage switch: nxn switch from 3x3 crossbar multiple stages of smaller switches
    exploiting parallelism: fragment datagram into fixed length cells on entry; switch cells through the fabric, reassemble datagram at exit



8x8 multistage switch
built from smaller-sized switches

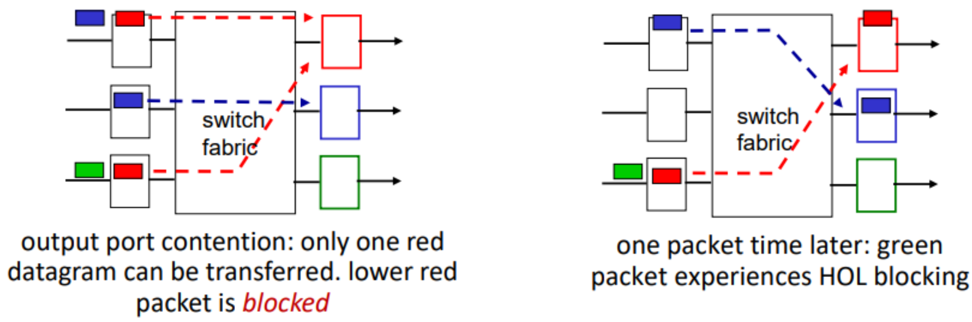
Cisco CRS router: basic unit (8 switching planes), each plane (3-stage interconnection network), up to 100's Tbps switching capacity



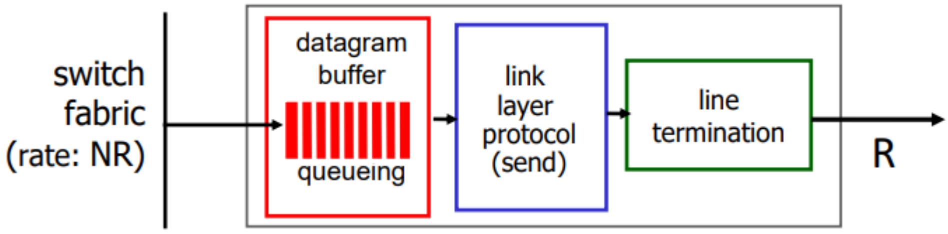
3. queuing

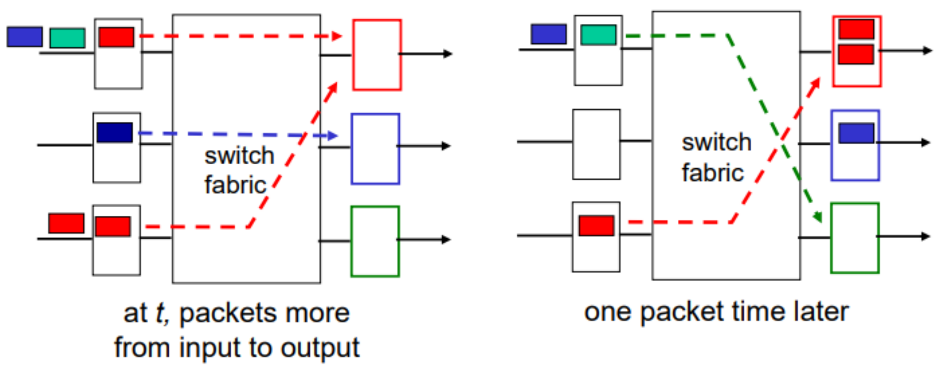
- **input port queuing**
  if switch fabric slower than input ports combined => queueing may occur at input queues, queuing delay and loss due to input buffer overflow

  Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward



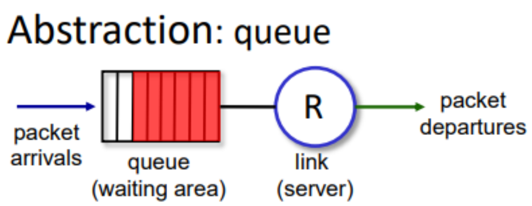
- **output port queuing**
  Buffering required when datagrams arrive from fabric faster than link transmission rate.
  Drop policy may applies due to congestion, lack of buffers.
  Scheduling discipline chooses among queued datagrams for transmission. One possible execution may be priority scheduling.



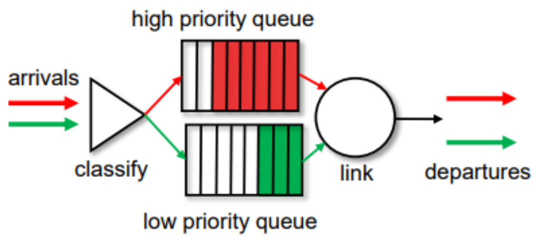


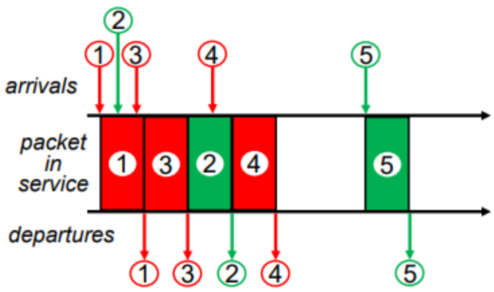
- **buffering size**
  RFC 3439 rule of thumb: avergae buffering equal to typical RTT (say 250 msec) times link capacity C (e.g., C = 10 Gbps link: 2.5 Gbit buffer)
  more recent recommendation: with N flows, $buffering = \frac{RTT \cdot C}{\sqrt{N}}$
- **buffer management**
  drop: decide which packet to add, drop when buffers are full (e.g., tail drop, priority)
  marking: which packets to mark to signal congestion (ECN, RED)

4. packet scheduling

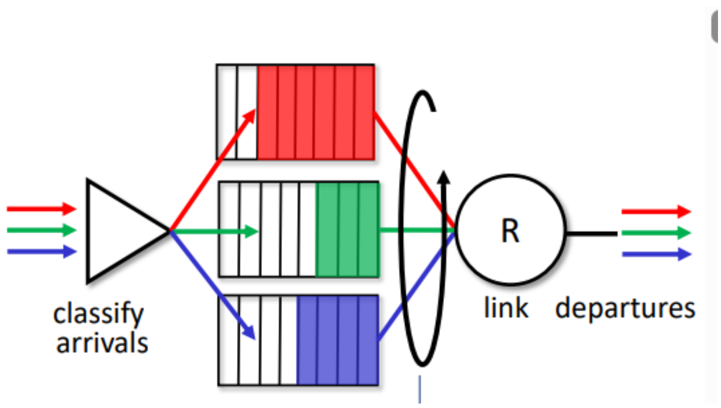
> packet scheduling: decide which packet to send next on link
>
> 
>

- **FCFS** / First come, first served (also known as FIFO / First-in-first-out): packets transmitted in order of arrival to output port
- **priority**
  1 arriving traffic classified, queued by class (any header fields can be used for classification)
  2 send packet from highest priority queue that has buffered packets (FCFS within priority class)

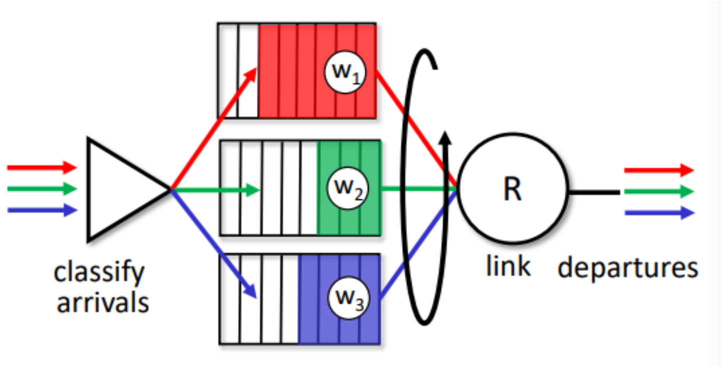




- **RR** / round robin: arriving traffic classified, queued by class, server cyclically, repeatedly scans class queues, sending one complete packet from each class (if available) in turn

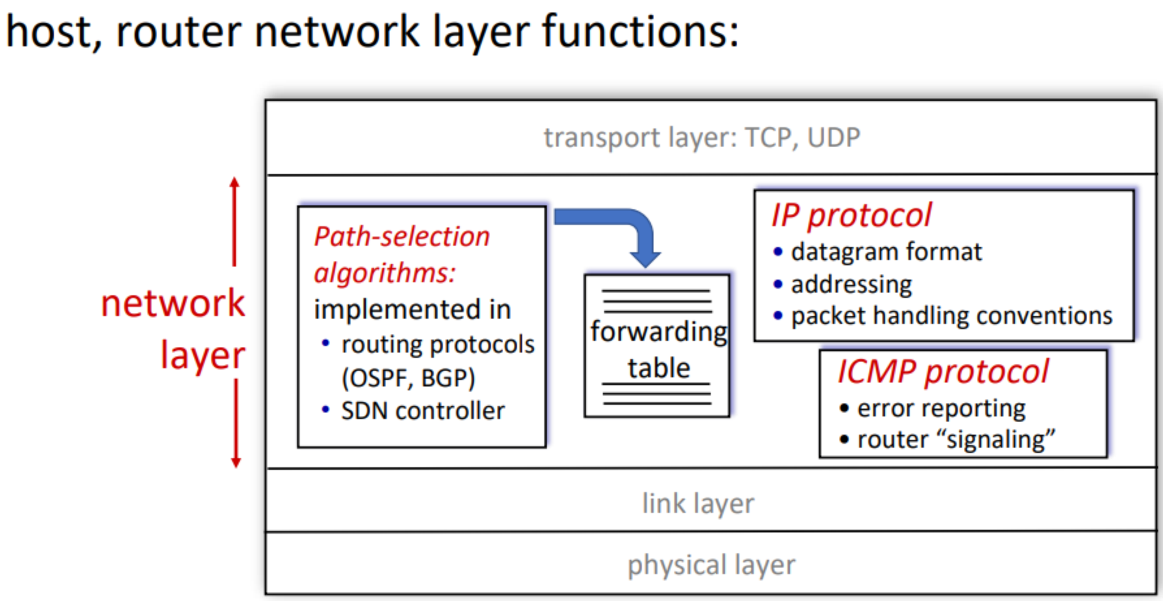


- **WFQ** / Weighted Fair Queuing: generalized round robin; each class, i, has weight w_i, and gets weighted amount of service in each cycle: $\frac{w_i}{\sum_j w_j}$; minimum bandwidth guaranteed (per-traffic-class)



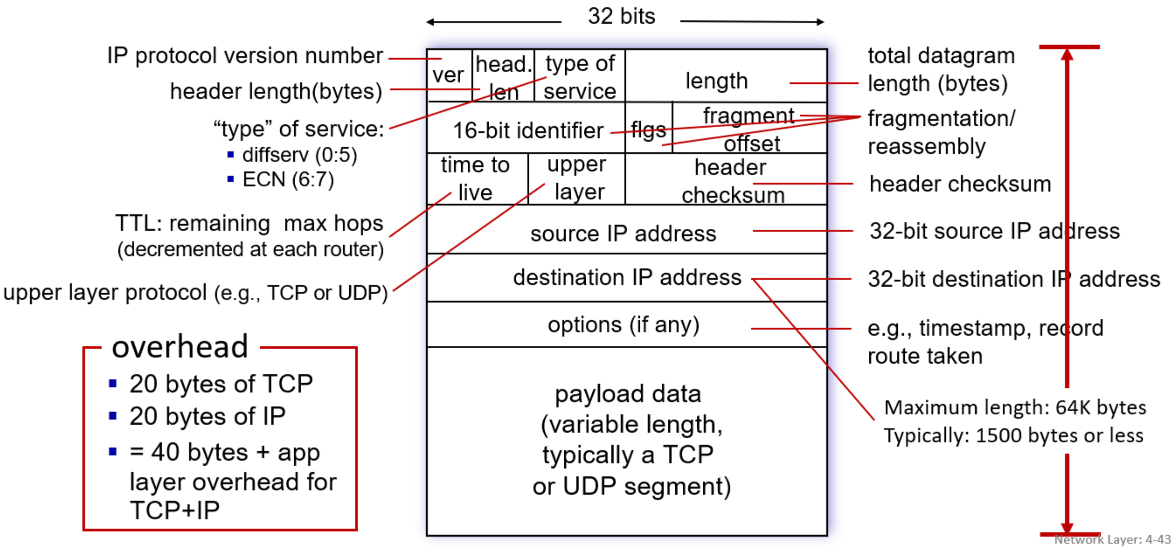
4.3 IP: Internet Protocol

Network Layer Overview



> OSPF / Open Shortest Path First：开放式最短路径优先
> BGP / Border Gateway Protocol：边界网关协议

4.3.1 IPv4 Datagram format



4.3.2 IP addressing

1. Terms

interface / 接口: connection between host / router and physical link

routers typically have multiple interfaces

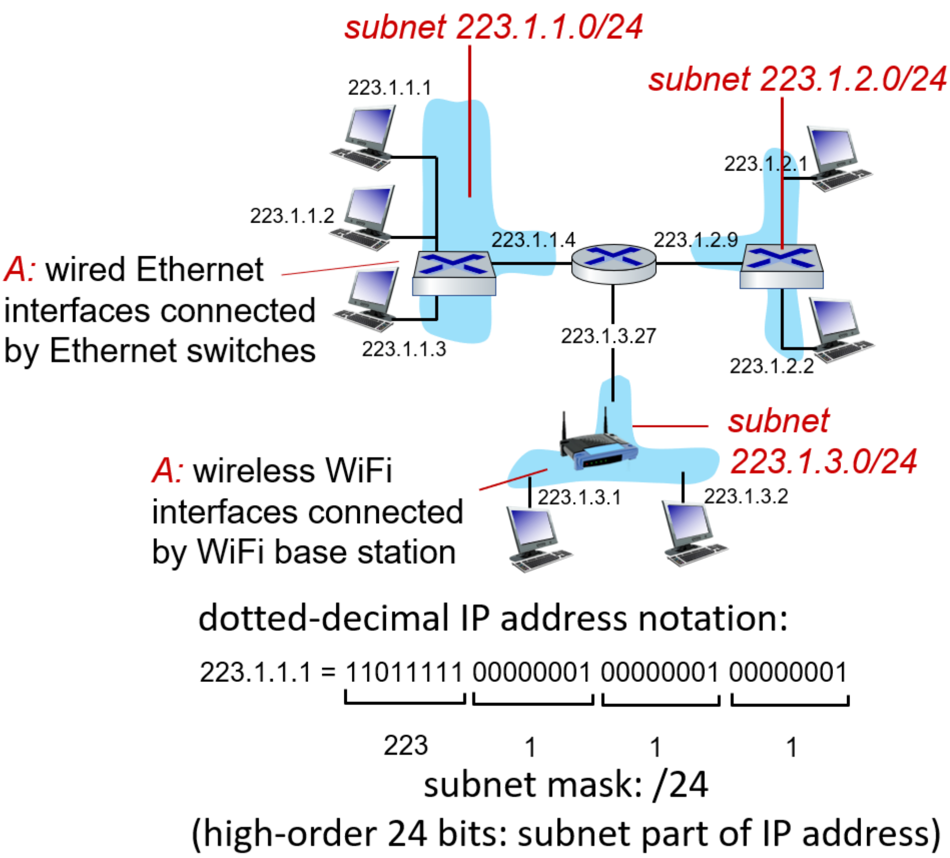
host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)

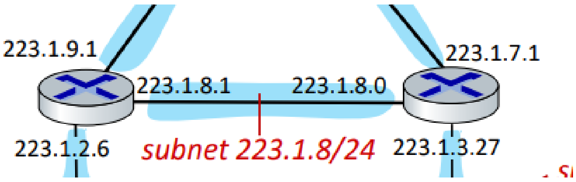
IP address: 32-bit identifier associated with each host or router interface

subnet: device interfaces that can physically reach each other without passing through an intervening router

subnet part: devices in same subnet have common high order bits

host part: remaining low order bits



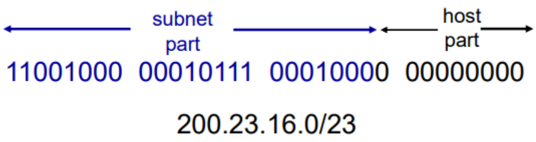


subnets can exist without hosts and only have routers connected to it, used for routing and interconnecting different parts of the network.

<u>CIDR / Classless InterDomain Routing</u>:

subnet portion of address of arbitray length

address format: `a.b.c.d/x`, where x is # bits in subnet portion of address



2. IP allocation: DHCP + ICANN

> Two questions:
> How does a **host** get IP address within its network?
> How does a **network** get IP address for itself?

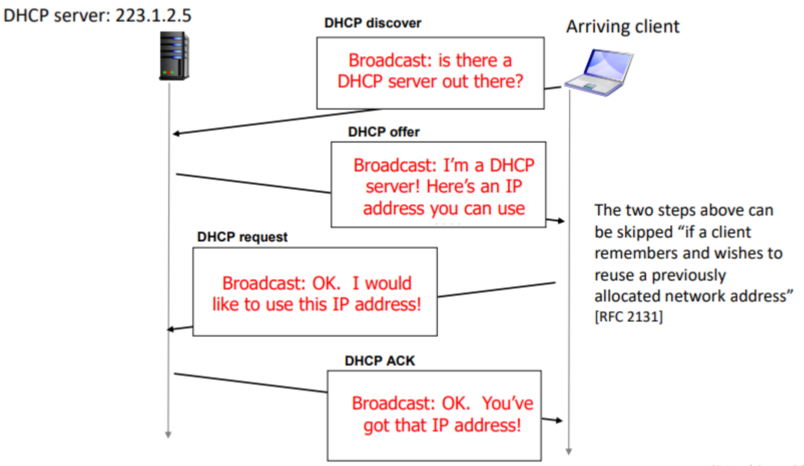
- **DHCP / Dynamic Host Configuration Protocol**: dynamically get address from a server

  hard-coded by sysadmin in config file (e.g., `/etc/rc.config` in UNIX)

  <u>goal</u>: host dynamically obtain IP address from network server when it joins network

  <u>workflow</u>: (DHCP server generally locates in router)

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

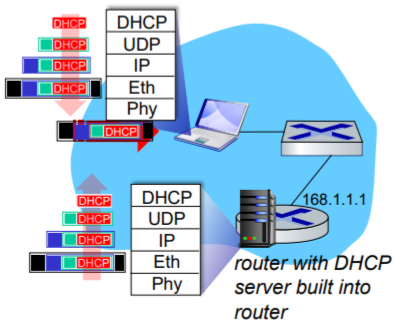
  

  <u>other functions</u>:

  address of first-hop router for client

  name and IP address of DNS server

  network mask (indicating network versus host portion of address)



> When a device wants to send a packet to another device on the same local area network (LAN), it needs to first determine the MAC address of the destination device. To do this, it sends out an Ethernet frame with a broadcast destination MAC address of `FF-FF-FF-FF-FF-FF`.
>
> When an Ethernet frame with a broadcast destination MAC address is sent on a LAN, all devices on that LAN receive the frame. However, only the device with the matching MAC address in the frame's destination field will process the frame and respond.
>
> The DHCP server's response will be unicast, meaning it will be sent directly to the MAC address of the requesting client, rather than being broadcast to all devices on the network. This unicast response will have the MAC address of the DHCP server as the source address and the MAC address of the requesting client as the destination address.
>
> DCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server

- **ISP's address space**
  network get subnet part of IP address from its provider ISP's address space

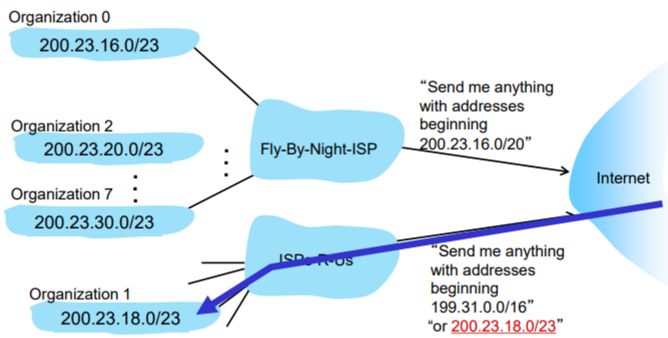
ISP's block	11001000 00010111 00010000 00000000	200.23.16.0/20

ISP can then allocate out its address space in 8 blocks:

Organization 0	11001000 00010111 00010000 00000000	200.23.16.0/23
Organization 1	11001000 00010111 00010010 00000000	200.23.18.0/23
Organization 2	11001000 00010111 00010100 00000000	200.23.20.0/23
...	....	....
Organization 7	11001000 00010111 00011110 00000000	200.23.30.0/23

Hierarchical addressing: more specific routes

- Organization 1 moves from Fly-By-Night-ISP to ISPs-R-Us
- ISPs-R-Us now advertises a more specific route to Organization 1



hierarchical addressing: route aggregation

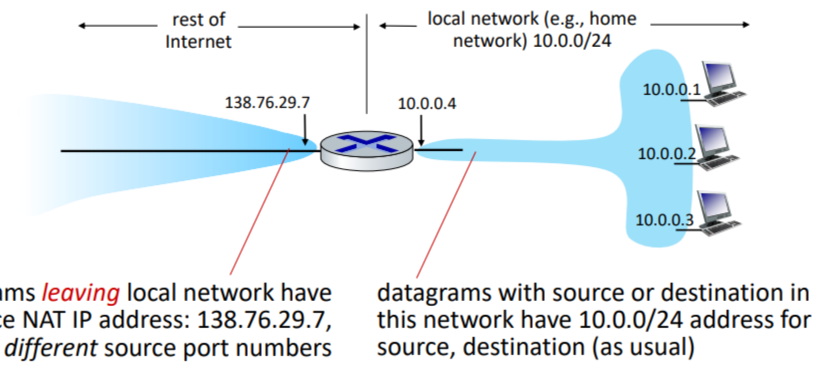
- **ICANN** / Internet Corporation for Assigned Names and Numbers: from whom ISP get block of addresses

  allocates IP addresses through 5 regional registries (RRs), which are responsible for administering and distributing IP address blocks to ISPs and other organizations within their respective regions

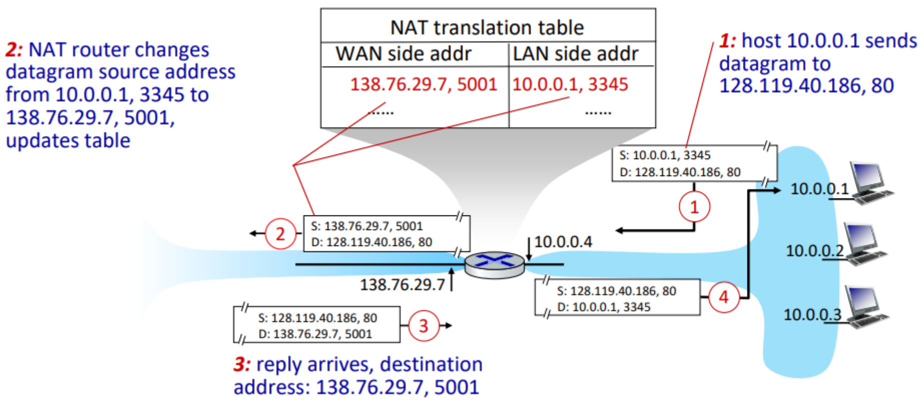
  manages DNS root zone, including delegation of individual TLD (.com, .edu , ...) management

  > For example, ICANN has delegated the management of the .com TLD to Verisign, who is responsible for maintaining the registry of all .com domain names and the associated IP addresses. Similarly, the management of the .edu TLD is delegated to Educause, a nonprofit association of universities and colleges in the United States.

4.3.3 NAT / network address translation

> 
>
>
> all devices in local network have 32-bit addresses in a "private" IP address space (10/8, 172.16/12, 192.168/16 prefixes) that can only be used in local network

- **the whole process**



- **pros and cons**

  pros: extensively used in home and institutional nets, 4G / 5G cellular nets

  cons (controversial):

  routers should only process up to layer 3

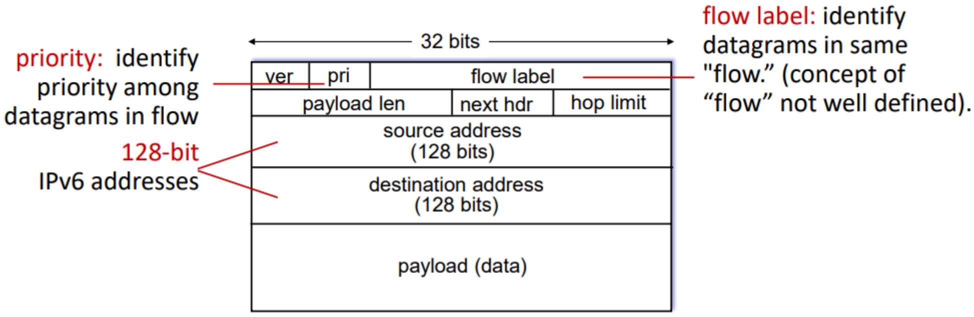
  address shortage should be solved by IPv6

  violates end-to-end argument (port # manipulation by network-layer device)

  NAT traversal: what if client wants to connect to server behind NAT?

4.3.4 IPv6

1. IPv6 Datagram format



What's missing (compared with IPv4):
- no checksum (to speed processing at routers)
- no fragmentation/reassembly
- no options (available as upper-layer, next-header protocol at router)

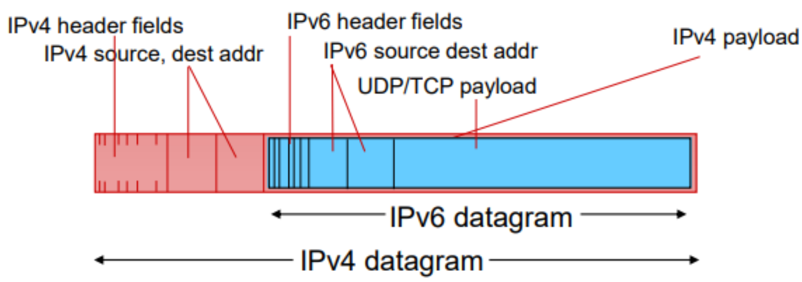
```
overall layout of a typical IPv6 packet:
0               8               16              24              32
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Traffic Class |           Flow Label                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Payload Length        |  Next Header  |   Hop Limit   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                      Source IPv6 Address                      |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                   Destination IPv6 Address                    |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

> 40-byte fixed length header
> `hop limit` == `TTL`，国内一般设置为32 / 64（国内一般经过10个路由器，国外20个）

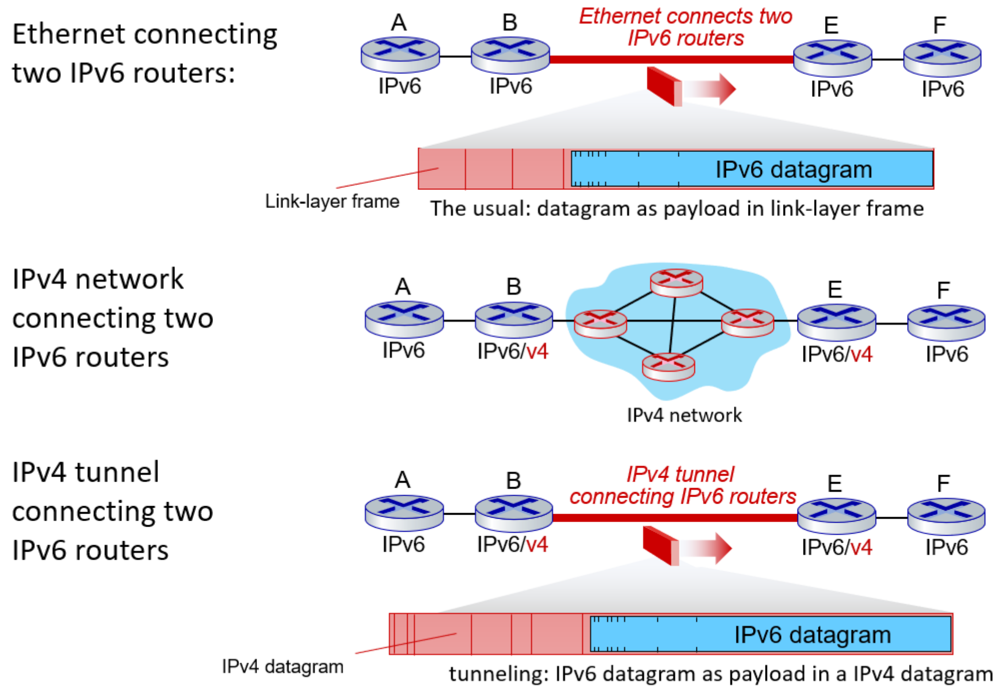
2. tunneling and encapsulation

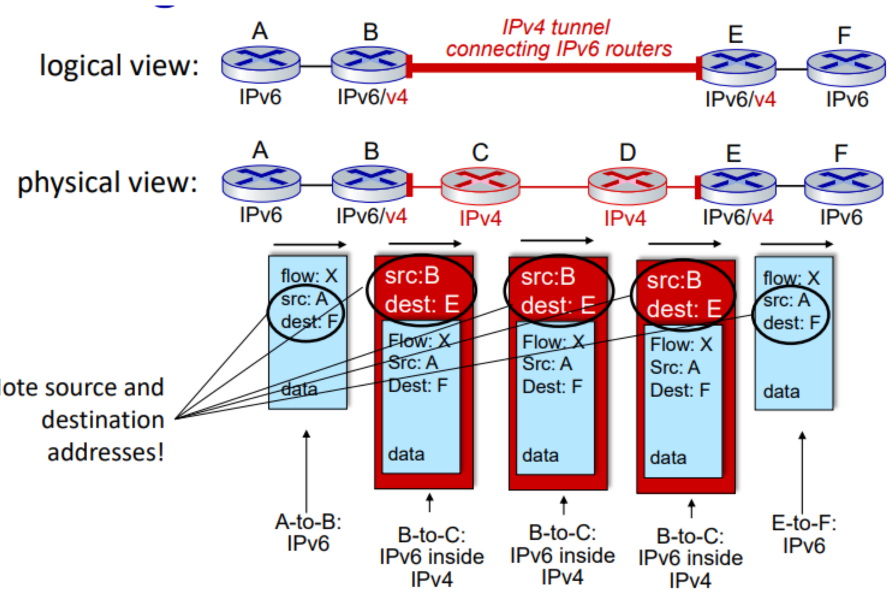
> **transition from IPv4 to IPv6**

tunneling: IPv6 datagram carried as payload in IPv4 datagram among IPv4 routers ("packet within a packet")



tunneling and encapsulation:





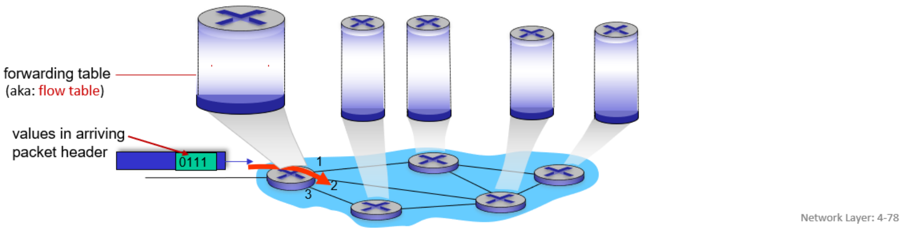
- **IPv6 adoption**
  Google: ~ 30% of clients access services via IPv6
  NIST: 1/3 of all US government domains are IPv6 capable

4.4 Generalized Forwarding (SDN) and Middleboxes

4.4.1 Match + action

Review: each router contains a forwarding table (aka: flow table)

- "match plus action" abstraction: match bits in arriving packet, take action
  - *destination-based forwarding:* forward based on dest. IP address
  - *generalized forwarding*:
    - many header fields can determine action
    - many action possible: drop/copy/modify/log packet



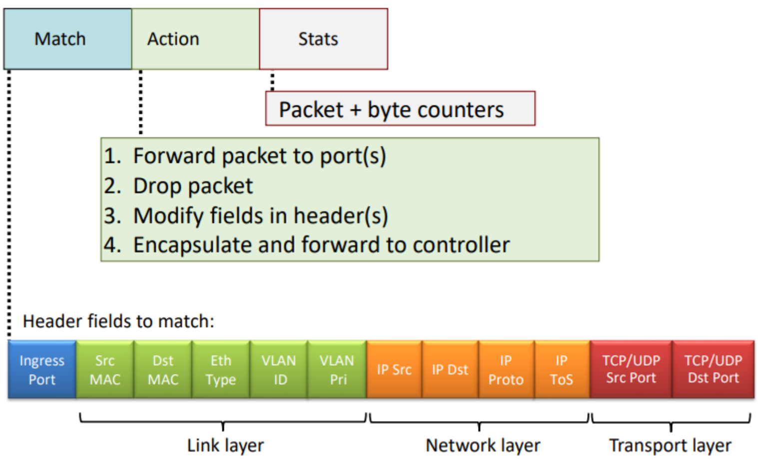
1. Flow table abstraction

flow: defined by header field values (in link-, network-, transport-layer fields)
generalized forwarding: simple packet-handling rules
match: pattern values in packet header fields
actions: for matched packet: drop, forward, modify, matched packet or send matched packet to controller
priority: disambiguate overlapping patterns
counters: #bytes and #packets

2. OpenFlow Protocol

> OpenFlow is a protocol that enables the communication between the control plane and the data plane of a Software-Defined Networking (SDN) architecture.
>
> It allows the control plane to dynamically program the forwarding behavior of network devices, such as switches and routers, by installing and updating flow rules in their flow tables.

- **flow table entries**



- **examples**

Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	51.6.0.8	*	*	*	*	port6

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

Firewall:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	*	*	*	*	22	drop

Block (do not forward) all datagrams destined to TCP port 22 (ssh port #)

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	128.119.1.1	*	*	*	*	*	drop

Block (do not forward) all datagrams sent by host 128.119.1.1

Layer 2 destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	22:A7:23:11:E1:02	*	*	*	*	*	*	*	*	*	port3

layer 2 frames with destination MAC address 22:A7:23:11:E1:02 should be forwarded to output port 3

- **match + action**: abstraction unifies different kinds of devices

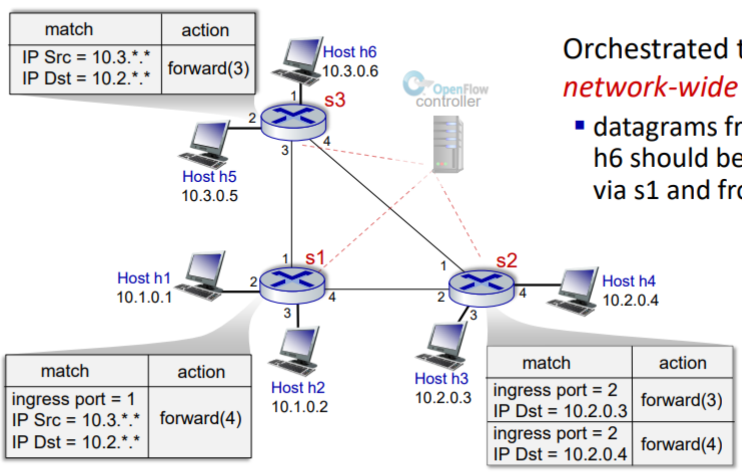
Router
- *match:* longest destination IP prefix
- *action:* forward out a link

Switch
- *match:* destination MAC address
- *action:* forward or flood

Firewall
- *match*: IP addresses and TCP/UDP port numbers
- *action:* permit or deny

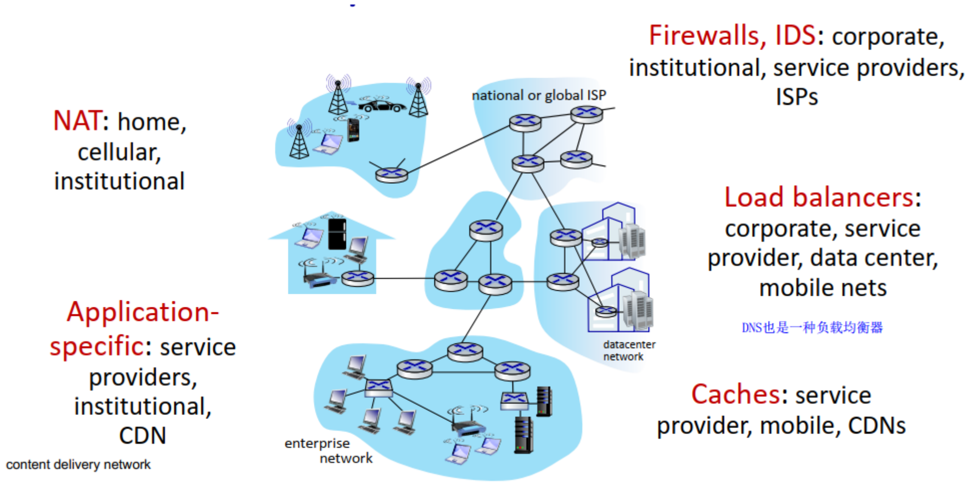
NAT
- *match:* IP address and port
- *action:* rewrite address and port

- **orchestrated tables** can create network-wide behavior



Orchestrated tables can create *network-wide* behavior, e.g.,:
- datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

4.4.2 Middleboxes



Initially: proprietary (closed) hardware solutions
move towards: “whitebox” hardware implementing open API

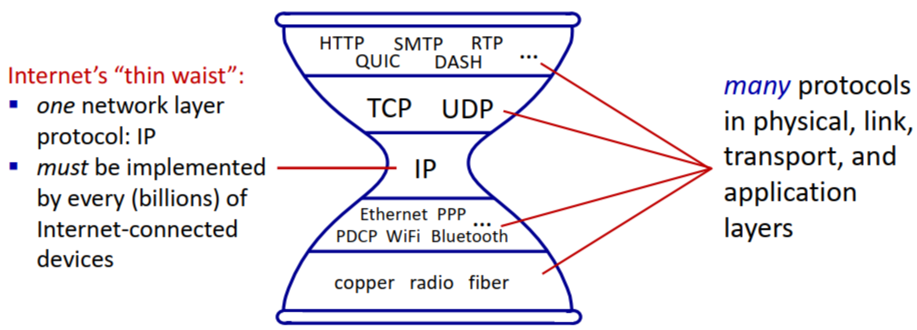
move away from proprietary hardware solutions

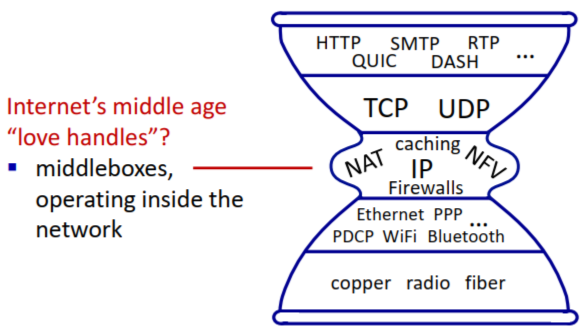
programmable local actions via match+action

move towards innovation/differentiation in software

SDN: (logically) centralize control and configuration management often in private/public cloud
network functions virtualization (NFV): programmable services over white box networking, computation, storage

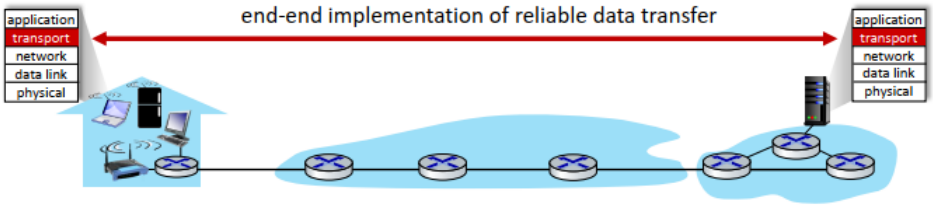
1. IP protocol: that narrow waist

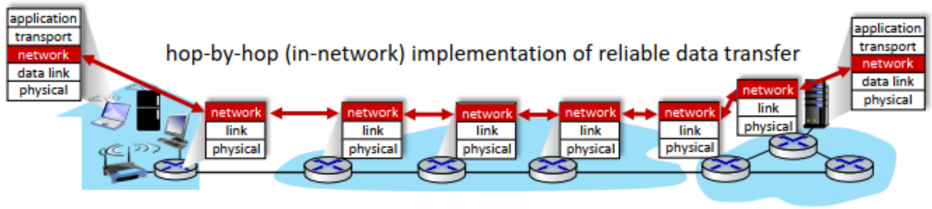




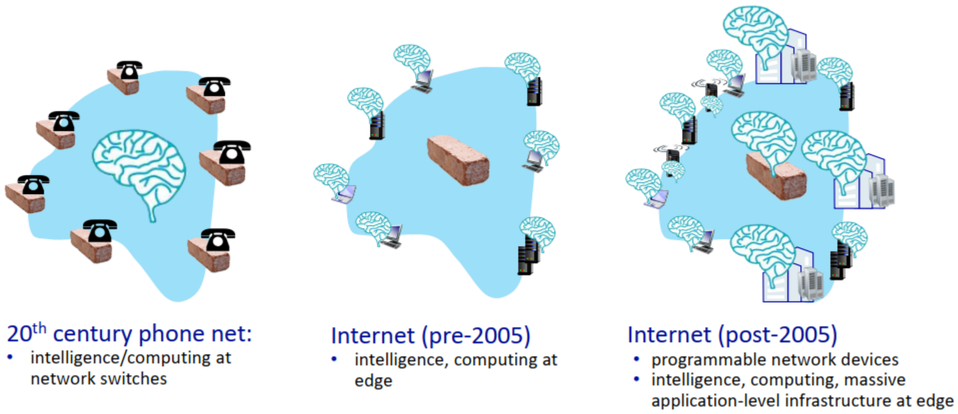
2. simple connectivity: end-end argument

the first graph is preferred in the following picture





3. intelligence, complexity at network edge

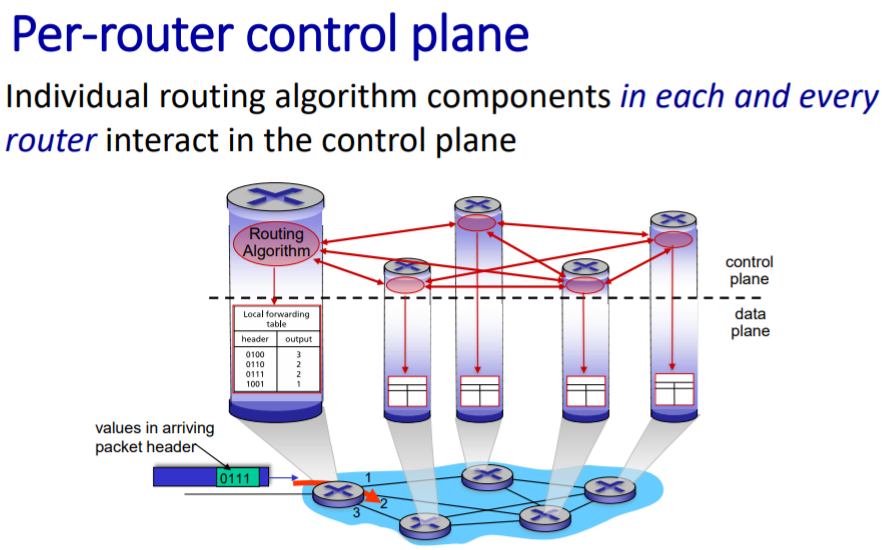


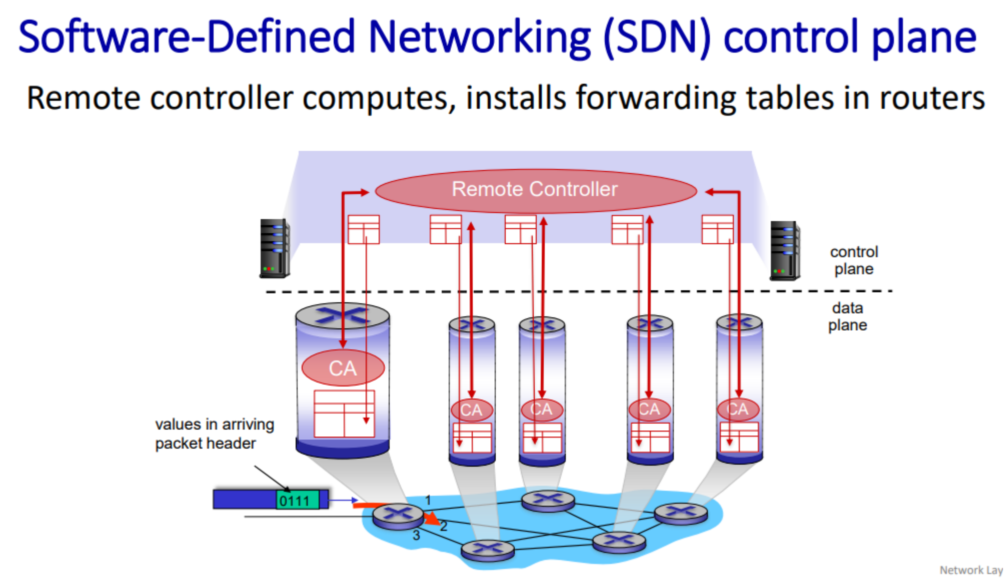
Chapter 5: Network Layer: Control Plane

Two approaches to structuring network control plane:

per-router control (traditional)

logically centralized control (software define networking)

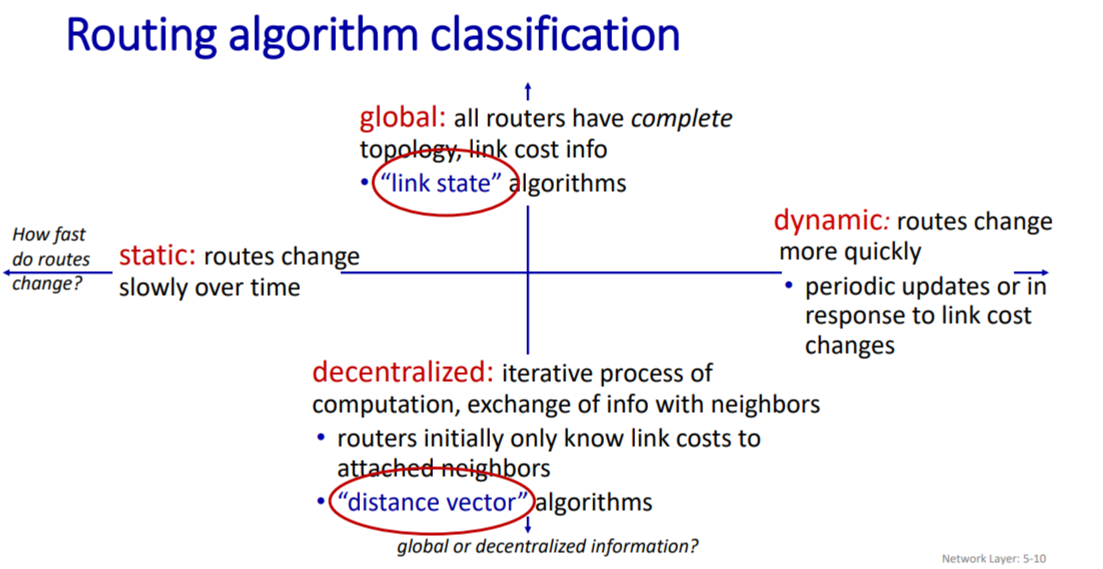




CA: Controller Agent

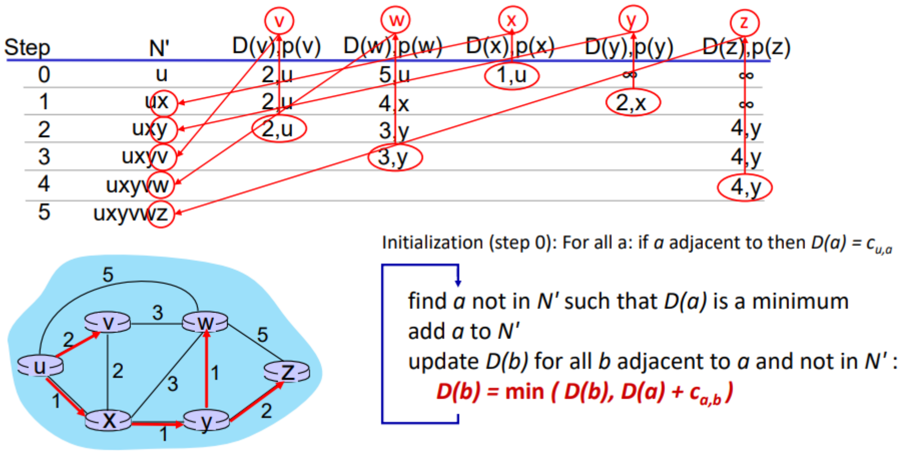
5.1 routing protocols

goal: determine good (least cost, fastest, least congested) paths from sending hosts to receiving host through network of routers



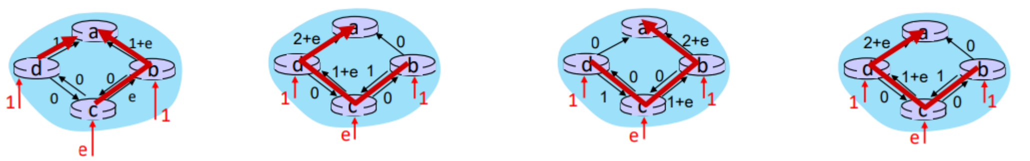
5.1.1 Dijkstra's link state

> 求单源、无负权的最短路
>
> 使用邻接表，时间复杂度为$O(n^2)$
>
> 使用fibonacci堆，时间复杂度为$O(nlogn)$
>
> iterative: after k iterations, know least cost path to k destinations
>
> **Notation**:
>
> $c_{x,y}$: direct link cost from node x to y $= \infty$ if not direct neighbors
>
> $D(v)$: current estimate of cost of least-cost-path from source to destination v
>
> $p(v)$: predecessor node along path from source to v
>
> N': set of nodes whose least-cost-path definitively known
>
> **Example**:



Dijkstra生成的shortest path tree不一定是minimal spanning tree

- **Messages Complexity**
  each router must broadcast its link state information to other n routers
  efficient (and interesting!) broadcast algorithms: $O(n)$ link crossings to disseminate a broadcast message from one source
  each router's message crosses $O(n)$ links: overall message complexity: $O(n^2)$
- **Oscillation Possible**
  sample scenario:
    routing to destination a, traffic entering at d, c, b with rates 1, e (<1), 1
    link costs are directional, and volume-dependent



initially | given these costs, find new routing…. resulting in new costs | given these costs, find new routing…. resulting in new costs | given these costs, find new routing…. resulting in new costs

5.1.2 Bellman-Ford's distance vector - RIP

求单源最短路，可以判断有无负权回路

Based on *Bellman-Ford* (BF) equation (dynamic programming):

Bellman-Ford equation

Let $D_x(y)$: cost of least-cost path from x to y.
Then:

$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

v's estimated least-cost-path cost to *y*

min taken over all neighbors *v* of *x*

direct cost of link from *x* to *v*

Naturally, the estimate $D_x(y)$ converge to the actual least cost $d_x(y)$

- **algorithm analysis**
  <u>iterative, asynchronous</u>: each local iteration caused by:
    local link cost change

DV update message from neighbor

distributed, self-stopping: each node notifies neighbors only when its DV changes

neighbors then notify their neighbors – only if necessary

no notification received, no actions taken!

Comparison of LS and DV algorithms

message complexity

LS: n routers, $O(n^2)$ messages sent

DV: exchange between neighbors; convergence time varies

speed of convergence

LS: $O(n^2)$ algorithm, $O(n^2)$ messages

- may have oscillations

DV: convergence time varies

- may have routing loops
- count-to-infinity problem

robustness: what happens if router malfunctions, or is compromised?

LS:

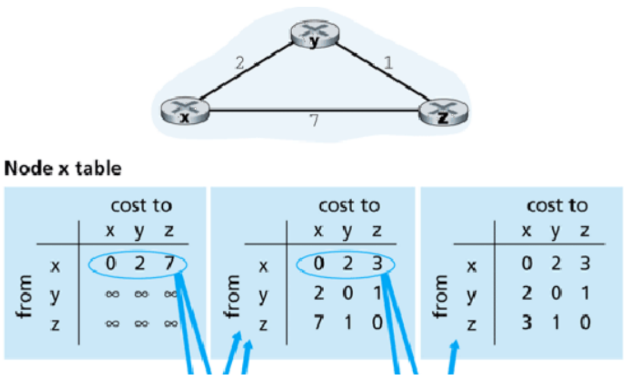
- router can advertise incorrect *link* cost
- each router computes only its *own* table

DV:

- DV router can advertise incorrect *path* cost ("I have a *really* low cost path to everywhere"): black-holing
- each router's table used by others: error propagate thru network

problem: count-to-infinity problem

potential solution: split horizon with poison reverse technique (but not completely)



X: Bellman-Ford's Distance Vector (RIP)

Y: count to infinity

Z: split horizon with poison reverse technique

the "x-y(2)" and "x-z(1)"

for example, x-y changes from 2 to 50

5.2 scalable routing

aggregate routers into regions known as **"autonomous systems" (AS)** (a.k.a. "domains")

- **intra-AS** (a.k.a. intra-domain): routing among within same AS ("network")

  all routers in AS must run same intra-domain protocol

  routers in different AS can run different intra-domain routing protocols

  gateway router: at "edge" of its own AS, has link(s) to router(s) in other AS'es

  Most common intra-AS routing protocols:

  - **RIP**: Routing Information Protocol

    classic DV: DVs exchanged every 30 secs

    no longer widely used
  - **EIGRP**: Enhanced Interior Gateway Routing Protocol

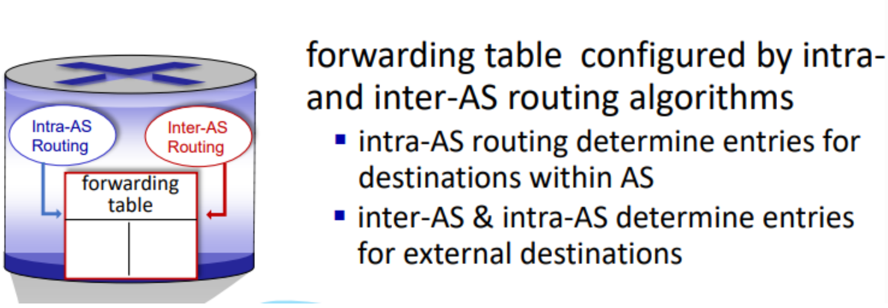
    DV based

    formerly Cisco-proprietary for decades
  - **OSPF**: Open Shortest Path First

    link-state routing

IS-IS protocol (ISO standard, not RFC standard) essentially same as OSPF

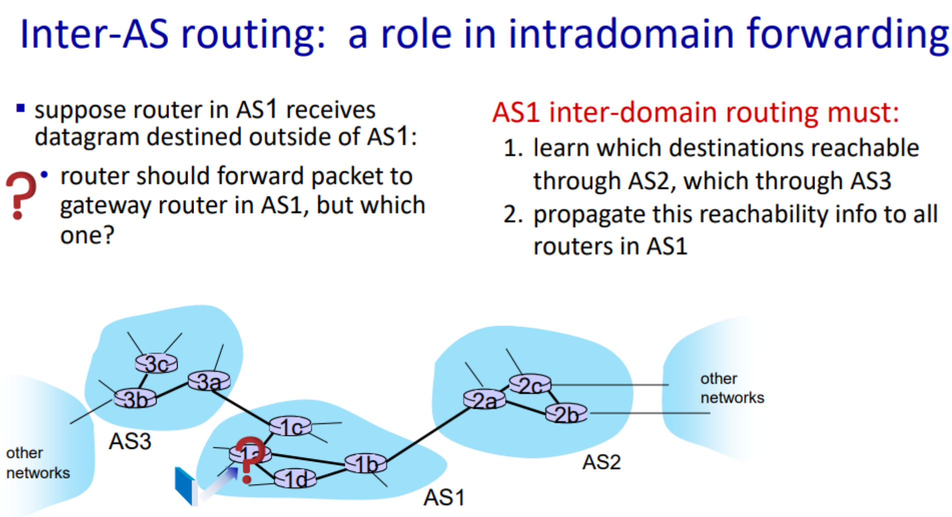
- **inter-AS** (a.k.a. inter-domain): routing among AS'es, use **BGP** algorithm
  gateways perform inter-domain routing (as well as intra-domain routing)



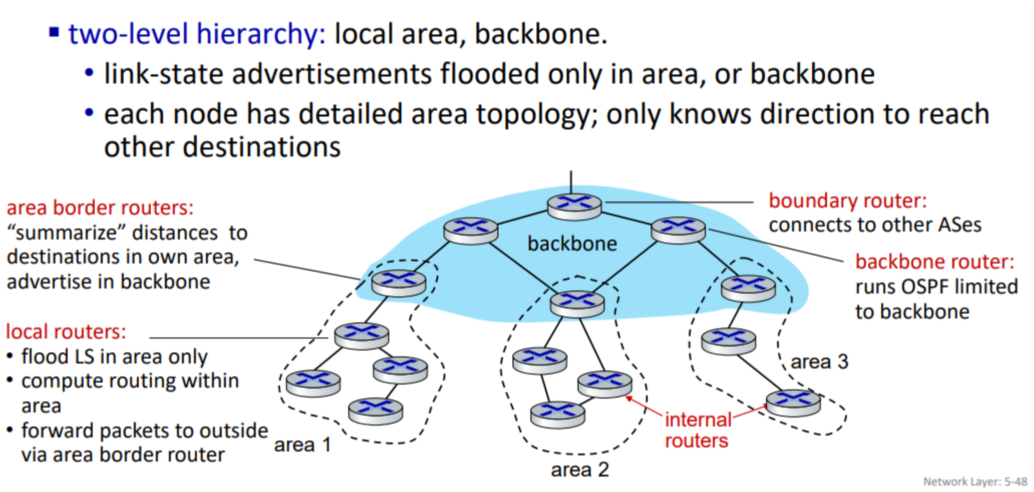
5.2.1 intra-AS: OSPF

`IGP` *(intra-AS routing): 最开始用RIP，之后用了OSPF、ISIS*

Open: publicly available
Classic link-state:
　　each router floods OSPF link-state advertisements (directly over IP rather than using TCP/UDP) to all other routers in entire AS
　　multiple link costs metrics possible: bandwidth, delay
　　each router has full topology, uses Dijkstra’s algorithm to compute forwarding table
security: all OSPF messages authenticated (to prevent malicious intrusion)



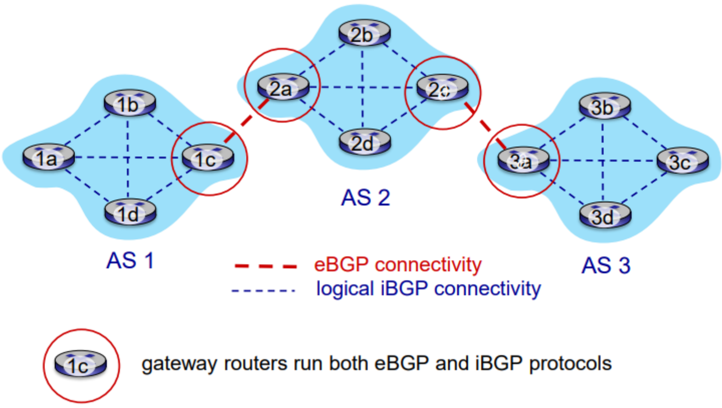
Hierarchical OSPF



5.2.2 inter-AS: BGP

BGP / Border Gateway Protocol: the de facto inter-domain routing protocol

BGP provides each AS a means to:
  `eBGP` (path vector): obtain subnet reachability information from neighboring ASes
  `iBGP`: propagate reachability information to all AS-internal routers.
  determine "good" routes to other networks based on reachability information and policy



1. BGP session

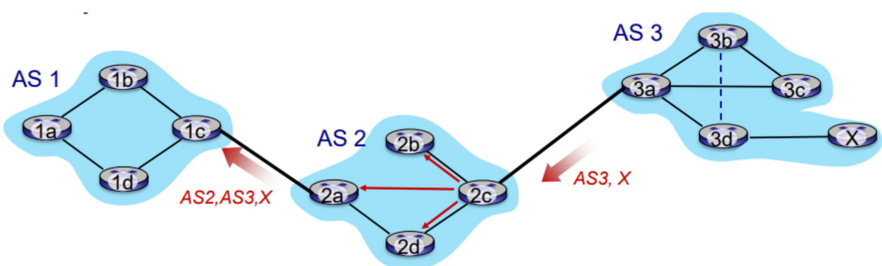
> Two BGP routers ("peers") exchange BGP messages over semi-permanent TCP connection: advertising paths to different destination network prefixes (BGP is a "path vector" protocol)

when AS3 gateway 3a advertises path AS3,X to AS2 gateway 2c:
  AS3 promises to AS2 it will forward datagrams towards X

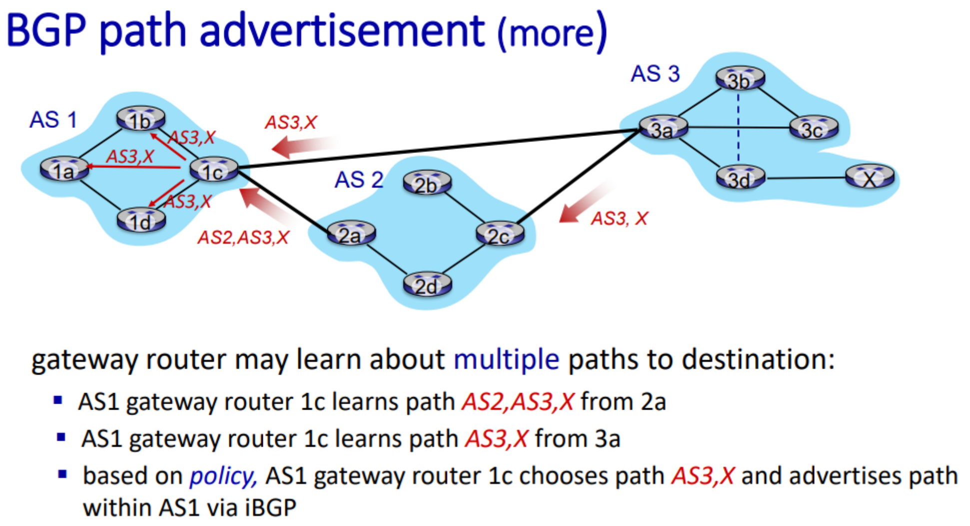
2. Path attributes

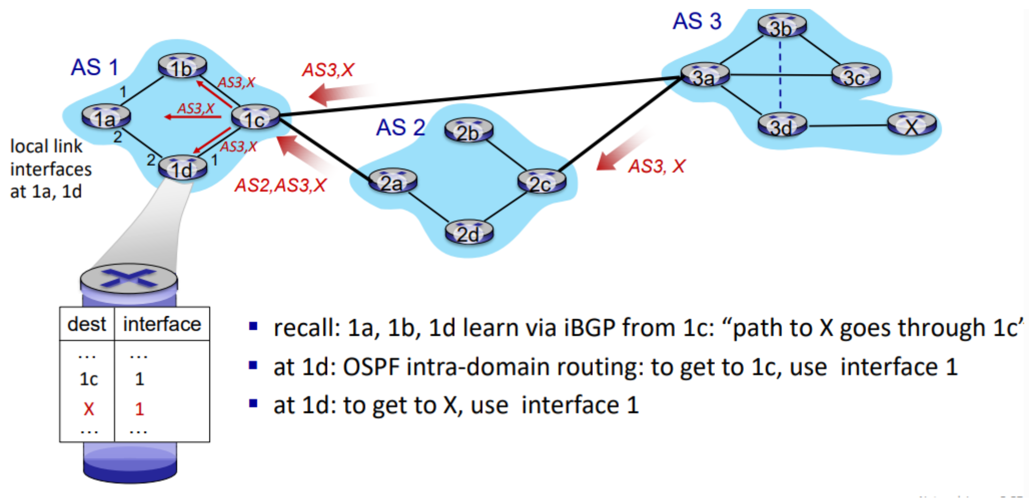
BGP advertised route: prefix + attributes
  prefix: destination being advertised
  two important attributes:
    `AS-PATH`: list of ASes through which prefix advertisement has passed
    `NEXT-HOP`: indicates specific internal-AS router to next-hop AS

policy-based routing:
  gateway receiving route advertisement uses import policy to accept/decline path (e.g., never route through AS Y).
  AS policy also determines whether to advertise path to other neighboring ASes

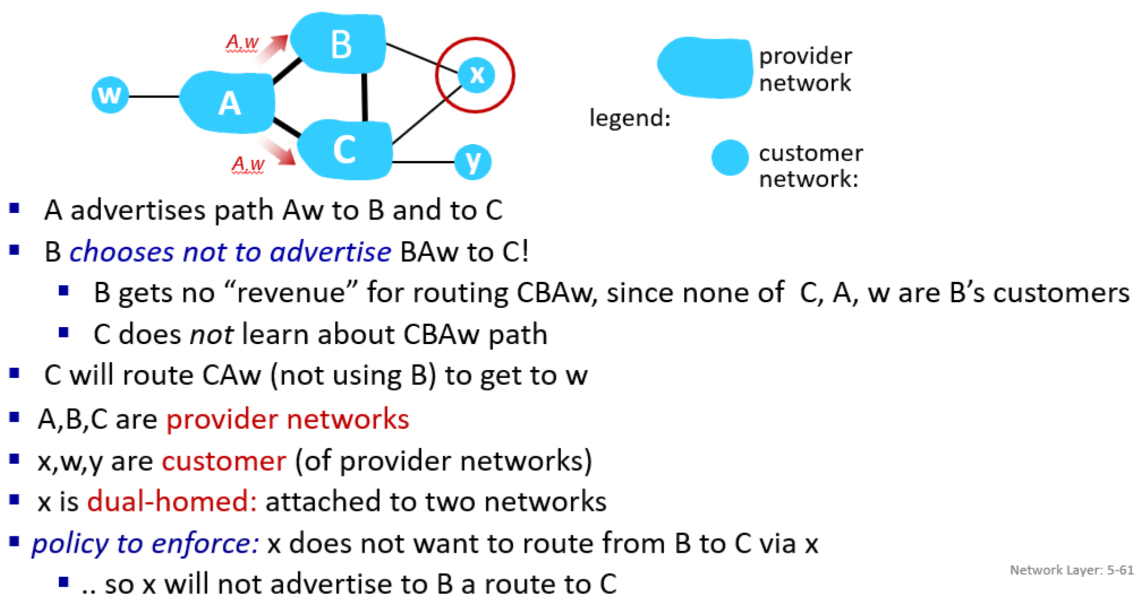


- AS2 router 2c receives path advertisement AS3,X (via eBGP) from AS3 router 3a
- based on AS2 policy, AS2 router 2c accepts path AS3,X, propagates (via iBGP) to all AS2 routers
- based on AS2 policy, AS2 router 2a advertises (via eBGP) path AS2, AS3, X to AS1 router 1c





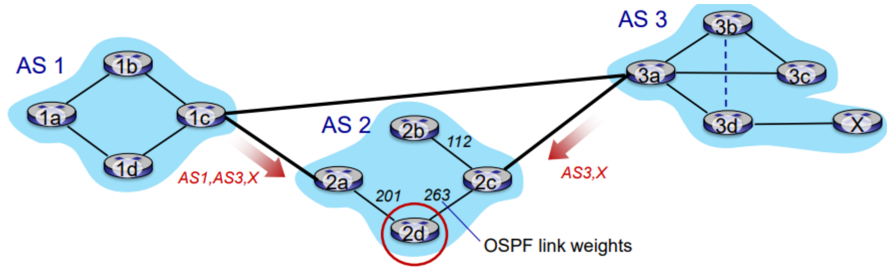
ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs – a typical "real world" policy)



3. Hot potato routing

——closest NEXT-HOP router

hot potato routing: choose local gateway that has least intra-domain cost (e.g., 2d chooses 2a, even though more AS hops to X): don't worry about inter-domain cost!



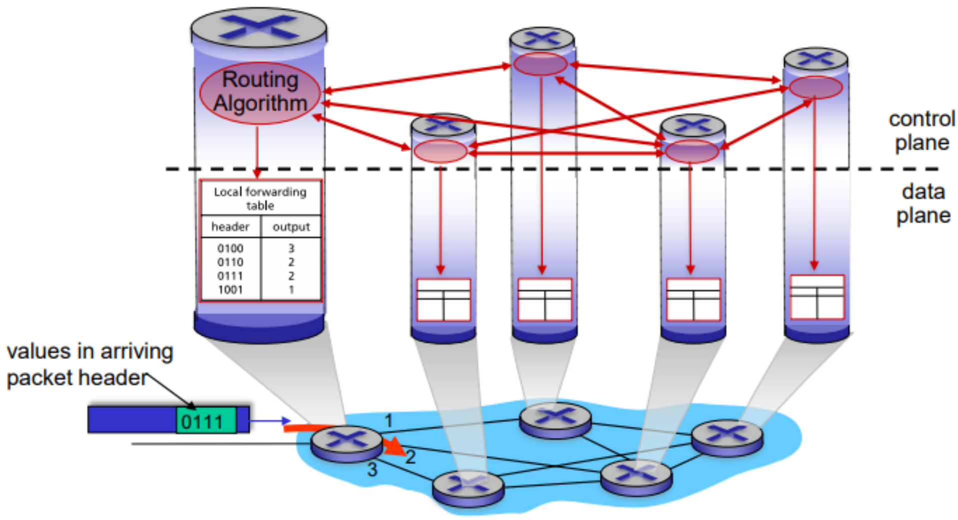
4. priority for BGP

router may learn about more than one route to destination AS, selects route based on:

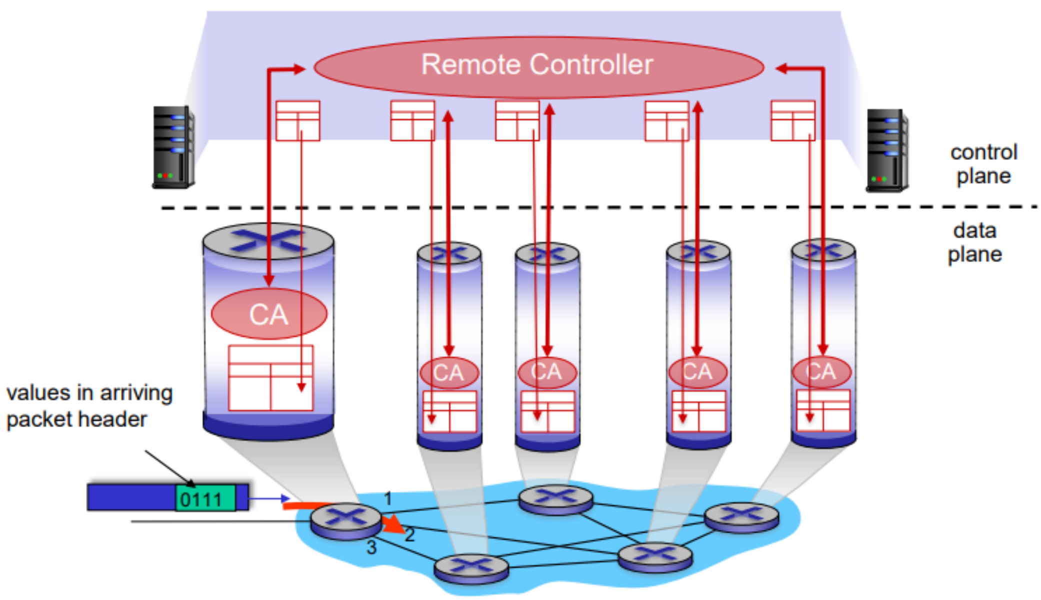
1. local preference value attribute: policy decision
2. shortest AS-PATH
3. closest NEXT-HOP router: hot potato routing
4. additional criteria

5.3 SDN control plane

- **Per-router control plane**: Individual routing algorithm components in each and every router interact in the control plane to computer forwarding tables



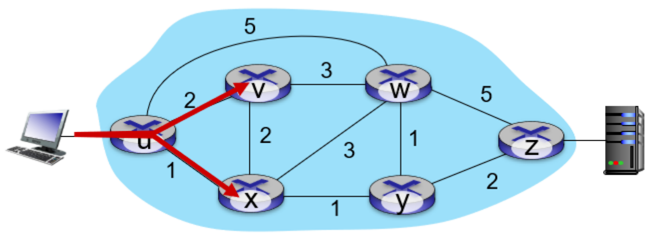
- **SDN control plane**: Remote controller computes, installs forwarding tables in routers



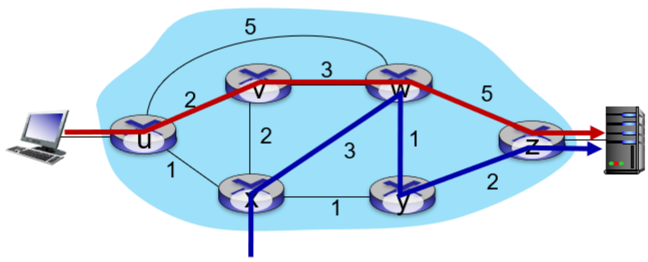
- **easier network management**: avoid router misconfigurations, greater flexibility of traffic flows
- table-based forwarding (recall OpenFlow API) **allows "programming" routers**: centralized "programming" easier: compute tables centrally and distribute distributed "programming" more difficult: compute tables as result of distributed algorithm (protocol) implemented in each-and-every router
- **open (non-proprietary) implementation** of control plane: foster innovation: let 1000 flowers bloom

5.3.1 traffic engineering

1. difficulty with traditional routing



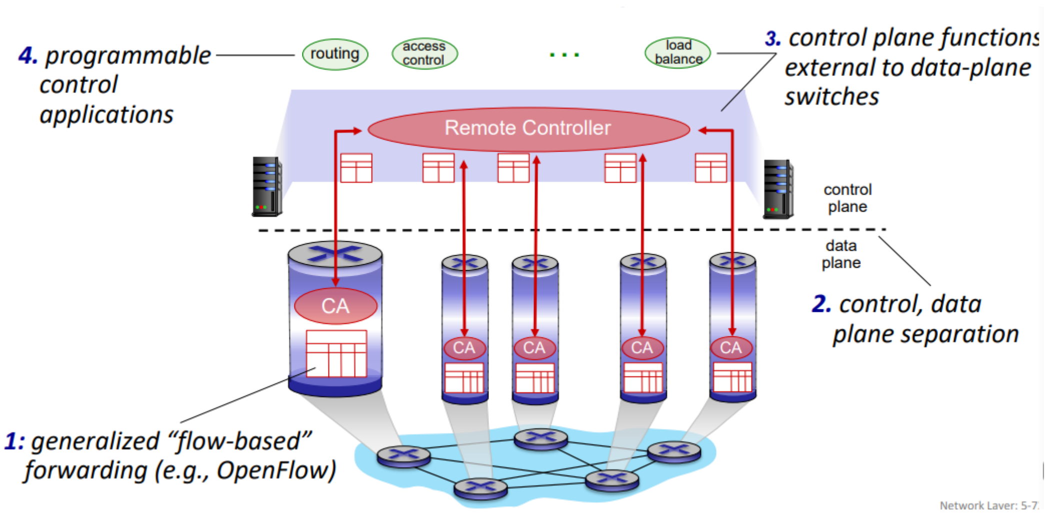
Q: what if network operator wants to split u-to-z traffic along uvwz *and* uxyz (load balancing)?
A: can't do it (or need a new routing algorithm)



Q: what if w wants to route blue and red traffic differently from w to z?

A: can't do it (with destination-based forwarding, and LS, DV routing)

2. SDN / Software Defined Networking

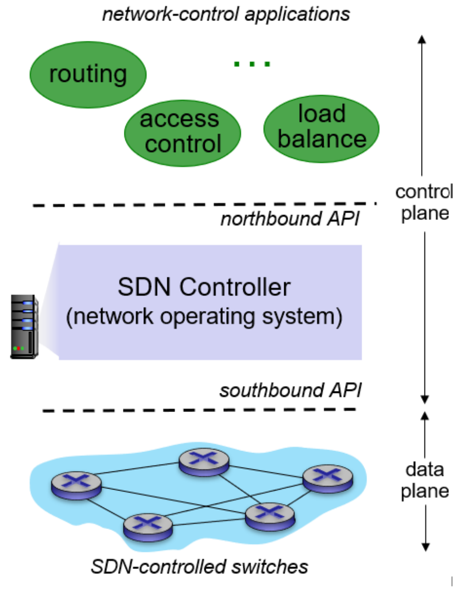


- Network-control apps, SDN controller, Data-plane switches:

network-control apps:
- "brains" of control: implement control functions using lower-level services, API provided by SDN controller
- *unbundled:* can be provided by 3rd party: distinct from routing vendor, or SDN controller

SDN controller (network OS):
- maintain network state information
- interacts with network control applications "above" via northbound API
- interacts with network switches "below" via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness

Data-plane switches:
- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- flow (forwarding) table computed, installed under controller supervision
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable, what is not
- protocol for communicating with controller (e.g., OpenFlow)

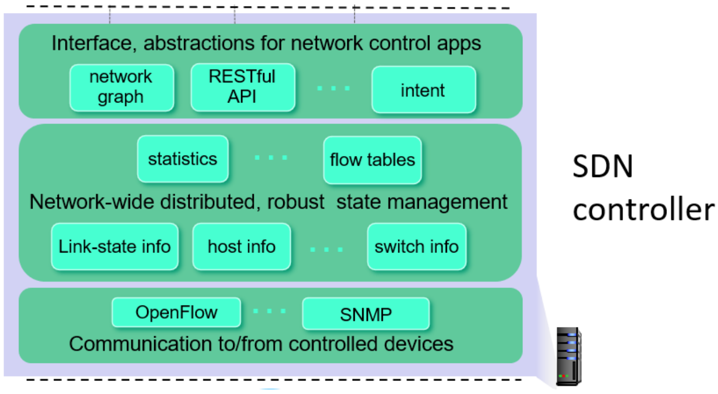


- Components of SDN controller:

interface layer to network control apps: abstractions API

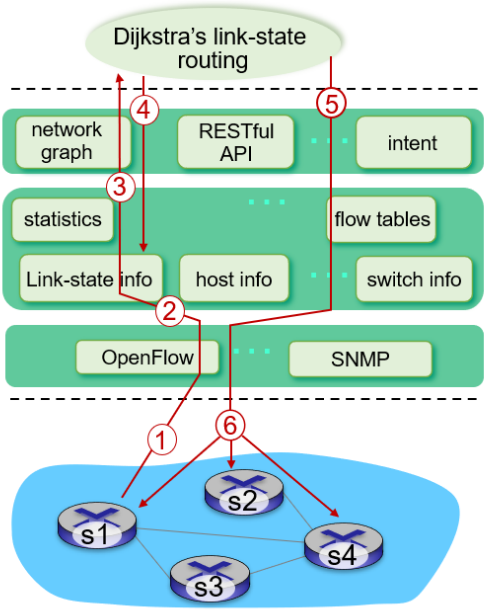
network-wide state management : state of networks links, switches, services: a *distributed database*

communication: communicate between SDN controller and controlled switches



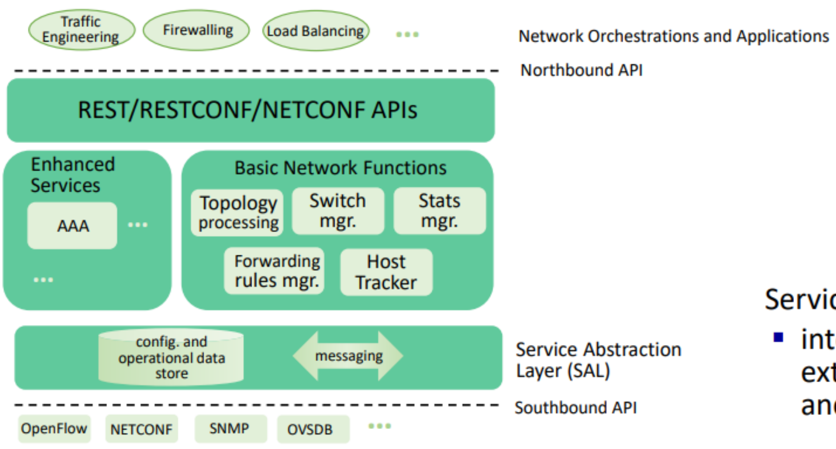
5.3.2 OpenFlow, OpenDaylight, ONOS

- **OpenFlow**



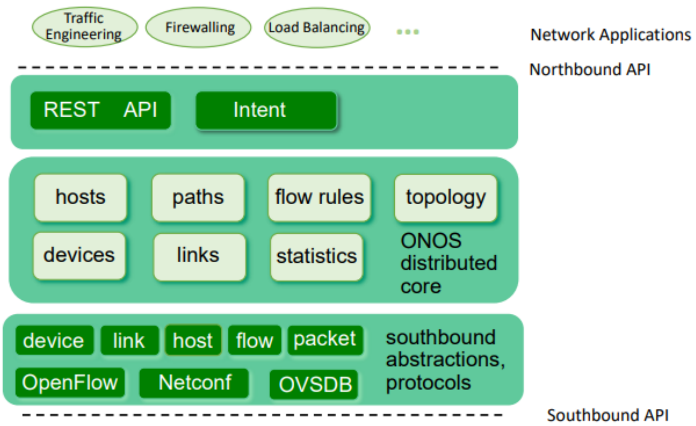
1. S1, experiencing link failure uses OpenFlow port status message to notify controller
2. SDN controller receives OpenFlow message, updates link status info
3. Dijkstra's routing algorithm application has previously registered to be called when ever link status changes. It is called.
4. Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes
5. link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed
6. controller uses OpenFlow to install new tables in switches that need updating

- **OpenDaylight**



Service Abstraction Layer:
- interconnects internal, external applications and services

- **ONOS controller**



- control apps separate from controller
- intent framework: high-level specification of service: what rather than how
- considerable emphasis on distributed core: service reliability, replication performance scaling

5.4 ICMP: Internet Control Message Protocol

used by hosts and routers to communicate network-level information

error reporting: unreachable host, network, port, protocol

echo request/reply (used by ping)

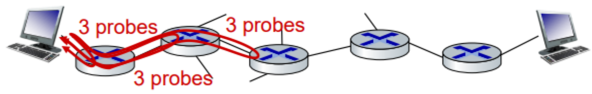
ICMP is a network-layer protocol

network-layer “above” IP: ICMP messages carried in IP datagrams

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

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

Traceroute and ICMP



- source sends sets of UDP segments to destination
  - 1st set has TTL =1, 2nd set has TTL=2, etc.
- datagram in *n*th set arrives to nth router:
  - router discards datagram and sends source ICMP message (type 11, code 0)
  - ICMP message possibly includes name of router & IP address
- when ICMP message arrives at source: record RTTs

stopping criteria:

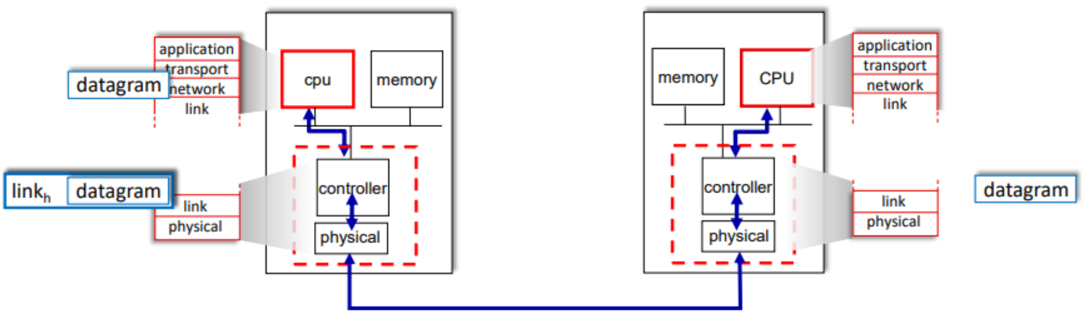
- UDP segment eventually arrives at destination host
- destination returns ICMP "port unreachable" message (type 3, code 3)
- source stops

Chapter 6: Link Layer and LANs

> **Terminology**
>
> nodes: hosts and routers
>
> links: communication channels tha connect adjacent nodes, classified as wired, wireless and LANs
>
> frame: layer-2 packet, encapsulates datagram
>
> *link layer has responsibility of transferring datagram from one node to physically adjacent node over a link*

- **link layer services**: framing / link access, rdt between adjacent nodes, flow control, error detection, error correction, half-duplex and full-duplex
  use MAC addresses in frame headers to identify source and destination
- **implementation**: implemented in host, NIC (network interface card) / chip, attaches into host's system buses, combination of hardware\software\firmware...

Interfaces communicating



sending side:

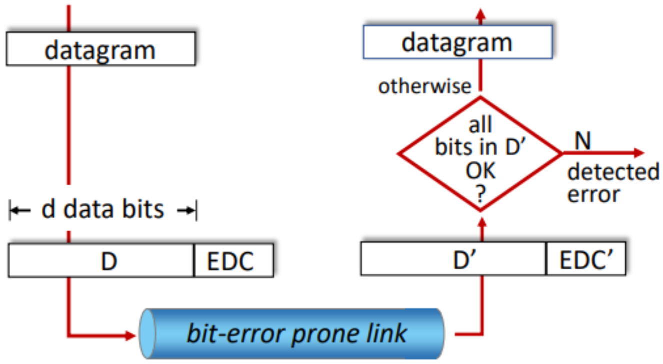
- encapsulates datagram in frame
- adds error checking bits, reliable data transfer, flow control, etc.

receiving side:

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

6.1 Error Detection and Correction

<u>**EDC**</u>: error detection and correction bits
<u>D</u>: data protected by error checking, may include header fields

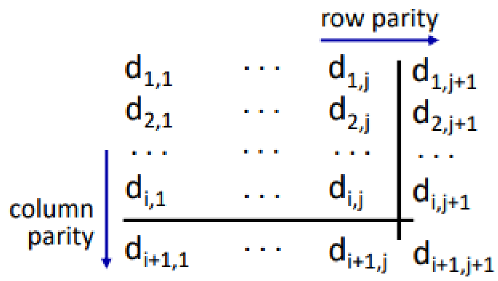


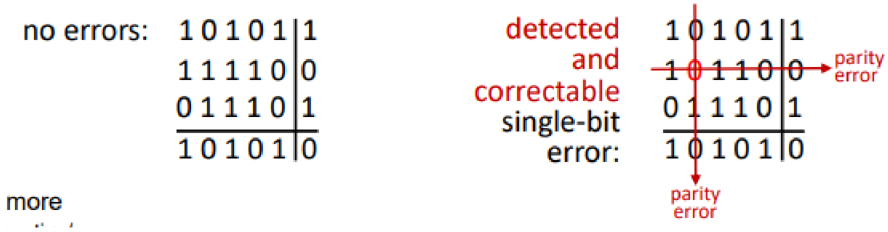
Error detection not 100% reliable!

- protocol may miss some errors, but rarely
- larger EDC field yields better detection and correction

1. Parity checking

- **single bit parity**: detect single bit errors
  (set parity bit so there is an even number of 1's)
- **two-dimensional bit parity**: detect and <u>correct</u> single bit errors
  (use column parity + row parity)

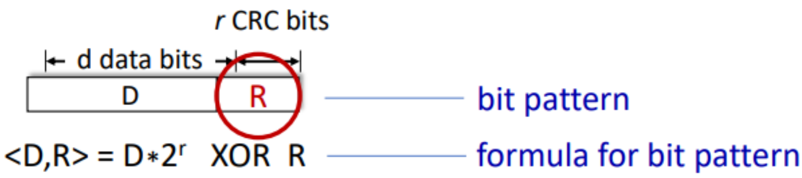




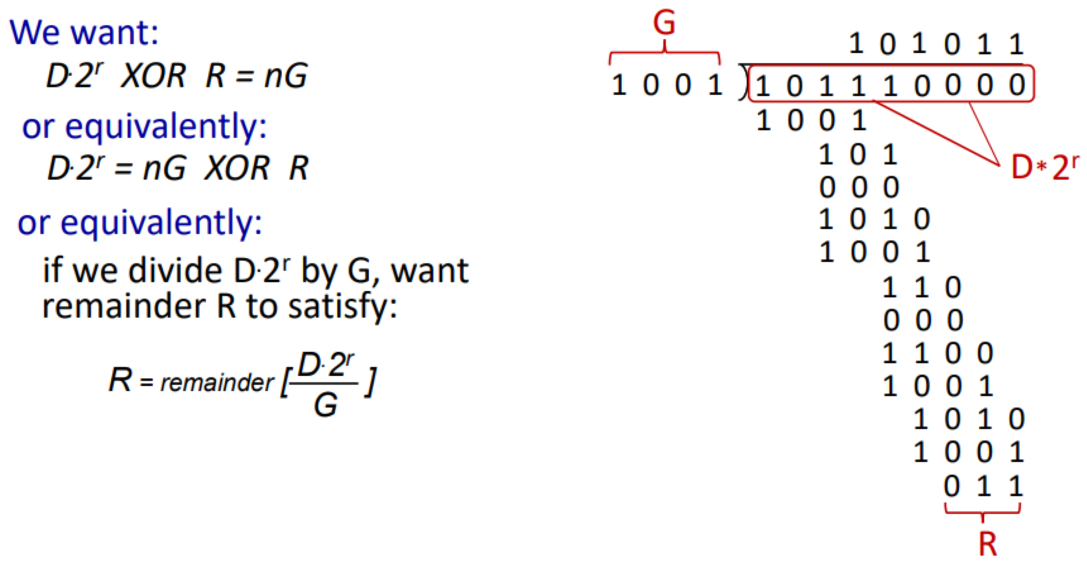
2. Cyclic Redundancy Check (CRC)

<u>**D**</u>: data bits (given, think of these as a binary number)
<u>**G**</u>: bit pattern (generator), of r+1 bits (given)

<u>**goal**</u>: choose r CRC bits, R, such that <D, R> exactly divisible by G (mod 2)
receiver knows G and divided <D, R> by G. if non-zero remainder: error detected!
can detect all burst errors less than r+1 bits
widely used in practice (Ethernet, 802.11 WiFi)



> <u>Example</u>:

> D = 101110, G = 1001 ($r = 3$)
>
> 
>
>
> Thus the result shall be: R = 011

6.2 Multiple Access Protocols

6.2.1 two types of "links"

- **point-to-point**: point-to-point link between Ethernet switch and host
- **broadcast (shared wire or medium)**: old-fashioned Ethernet, upstream HFC in cable-based access network (main focus of this chapter)

6.2.2 Multiple Access Protocols

- **usage scenario**: single shared broadcast channel
  a single communication channel that is shared by multiple nodes
  This channel is typically a broadcast channel (any transmission on it is received by all nodes)
- **targeted problem**: Interference from simultaneous transmissions
  collision if node receives two or more signals at the same time
- **multiple access protocol**
  - distributed algorithm for channel sharing
    determines how the nodes share the channel and when each node can transmit its data
  - Communication about channel sharing using the channel itself
    no separate "out-of-band" channel available for coordination
    the same channel used for transmitting data also used for exchanging control information and coordinating access among the nodes

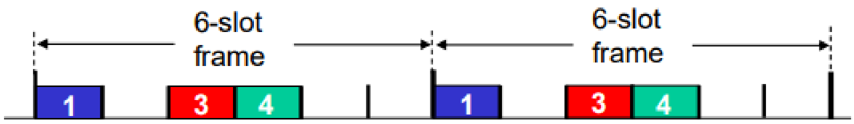
An ideal multiple access protocol:

> **given**: multiple access channel (MAC) of rate R bps
>
> **desiderata**:
>
> 1. when one node wants to transmit, it can send at rate R.
> 2. when M nodes want to transmit, each can send at average rate $\frac{R}{M}$
> 3. fully decentralized
>   no special node to coordinate transmissions
>   no synchronization of clocks, slots
> 4. simple

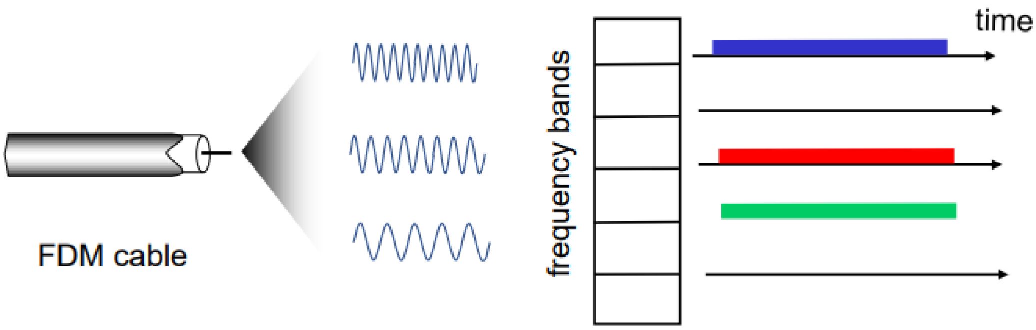
1. channel partitioning

> divide channel into smaller "pieces" (time slots, frequency, code)
> allocate piece to node for exclusive use

- **TDMA**: time division multiple access
  access to channel in "rounds"
  each station gets fixed length slot (length = packet transmission time) in each round
  unused slots go idle
  example: 6-station LAN, 1,3,4 have packets to send, slots 2,5,6 idle



- **FDMA**: frequency division multiple access
  channel spectrum divided into frequency bands
  each station assigned fixed frequency band
  unused transmission time in frequency bands go idle



2. random access

> channel not divided, allow collisions
> "recover" from collisions

- when node has packet to send, transmit at full channel data rate R (no a priori coordination among nodes)
- problem to be solved
  how to detect collisions + **how to recover from collisions**
- example of random access MAC protocols:
  ALOHA, slotted ALOHA
  CSMA, CSMA/CD, CSMA/CA

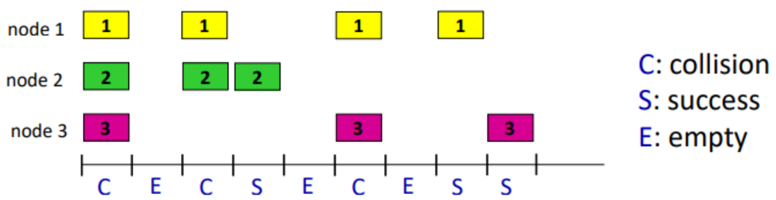
1. **ALOHA**

- **Slotted ALOHA**

assumptions:
- all frames same size
- time divided into equal size slots (time to transmit 1 frame)
- nodes start to transmit only slot beginning
- nodes are synchronized
- if 2 or more nodes transmit in slot, all nodes detect collision

operation:
- when node obtains fresh frame, transmits in next slot
  - *if no collision:* node can send new frame in next slot
  - *if collision:* node retransmits frame in each subsequent slot with probability *p* until success

randomization – *why*?



Pros:
- single active node can continuously transmit at full rate of channel
- highly decentralized: only slots in nodes need to be in sync
- simple

Cons:
- collisions, wasting slots
- idle slots
- nodes may be able to detect collision in less than time to transmit packet
- clock synchronization

Efficiency proof: (more efficient that pure ALOHA at the expense of node synchronization)

- *suppose:* *N* nodes with many frames to send, each transmits in slot with probability *p*
  - prob that given node has success in a slot $= p(1-p)^{N-1}$
  - prob that *any* node has a success = $Np(1-p)^{N-1}$
  - max efficiency: find p^* that maximizes $Np(1-p)^{N-1}$
  - for many nodes, take limit of $Np^*(1-p^*)^{N-1}$ as *N* goes to infinity, gives:

  max efficiency = 1/e = .37
- *at best:* channel used for useful transmissions 37% of time!

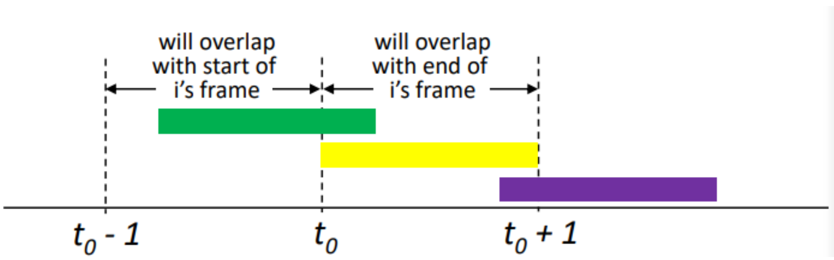
- **Pure ALOHA**

unslotted Aloha: simpler, no synchronization

  when frame first arrives: transmit immediately

collision probability increases with no synchronization

  frame sent at t_0 collides with other frames sent in $[t0-1, t0+1]$



Efficiency = 18%

2. **CSMA / Carrier Sense Multiple Access**
   - **simple CSMA**: listen before transmit
     if channel sensed idle: transmit entire frame
     if channel sensed busy: defer transmission
   - **CSMA/CD**: with collision detection
     collisions detected within short time
     colliding transmissions aborted, reducing channel wastage
     collision detection easy in wired, difficult with wireless

     After aborting, NIC (Network Interface Card, 网卡) enters binary (exponential) backoff:
     after m^{th} collision, NIC chooses K at random from $\{0, 1, 2, \ldots, 2m - 1\}$. NIC waits $K \cdot 512$ bit times, returns to Step 2
     more collisions: longer backoff interval

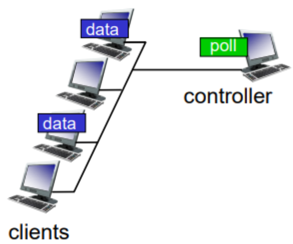
     - T_{prop} = max prop delay between 2 nodes in LAN
     - t_{trans} = time to transmit max-size frame

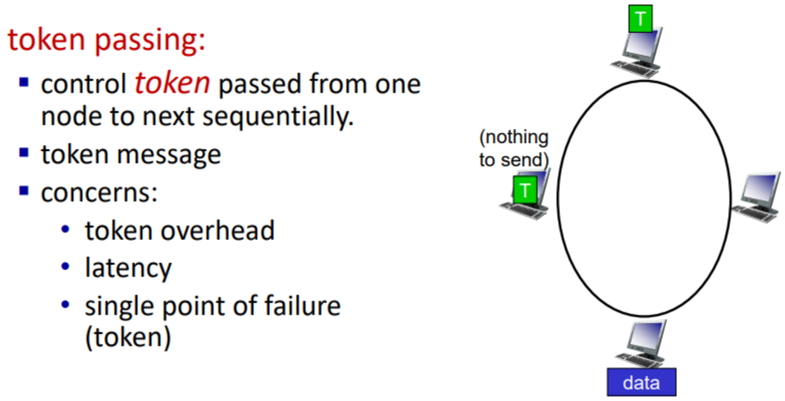
     $$efficiency = \frac{1}{1 + 5t_{prop}/t_{trans}}$$

     - efficiency goes to 1
       - as t_{prop} goes to 0
       - as t_{trans} goes to infinity
     - better performance than ALOHA: and simple, cheap, decentralized!
   - **CSMA/CA**: with collision avoid
3. **"Taking turns" MAX protocols**
   - **channel partitioning MAC protocols**
     share channel efficiently and fairly at high load
     inefficient at low load: delay in channel access, 1/N bandwidth allocated even if only 1 active node!
   - **random access MAC protocols**
     efficient at low load: single node can fully utilize channel
     high load: collision overhead
   - **taking-turns**
     polling from central site, token passing

polling:

- controller node “invites” other nodes (clients)to transmit in turn
- typically used with “dumb” devices
- concerns:
  - polling overhead
  - latency
  - single point of failure (controller)





6.3 LANs

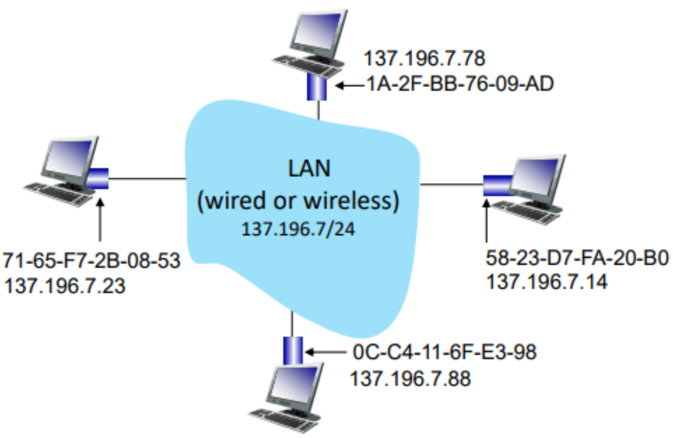
6.3.1 addressing, ARP

1. IP and MAC

- **32-bit IP address**
  network-layer address for interface
  used for layer 3 (network layer) forwarding
  e.g.: 128.119.40.136
- **MAC (or LAN or physical or Ethernet) address**

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

  allocated by IEEE
  manufacturer buys portion of MAC address space

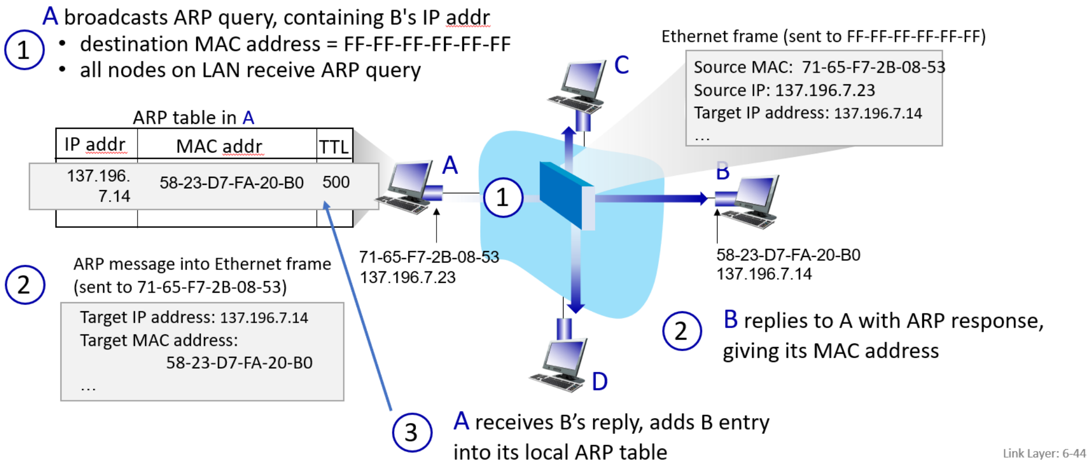


2. ARP: address resolution protocol

> determine interfaces' MAC address using IP address

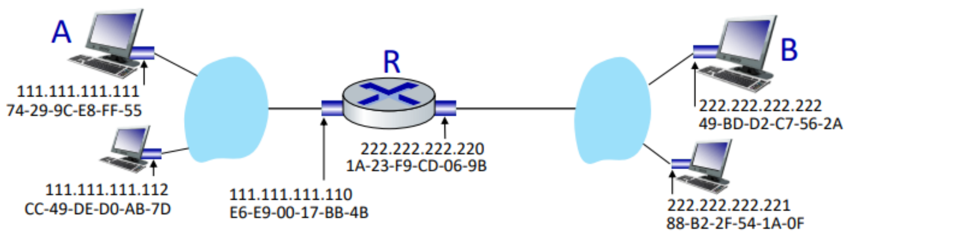
- **ARP table**: each IP node (host, router) on LAN has table
  IP/MAC address mappings for some LAN nodes: `< IP address; MAC address; TTL>`
  TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)
- **same LAN in action**

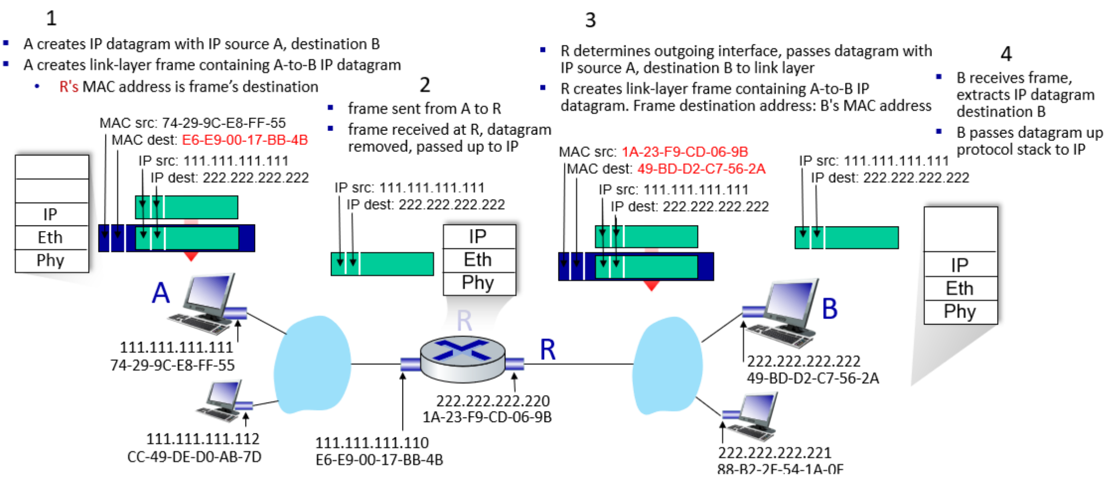
example: A wants to send datagram to B
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



- **routing to another subnet**

walkthrough: sending a datagram from *A* to *B* via *R*
- focus on addressing – at IP (datagram) and MAC layer (frame) levels
- assume that:
  - A knows B's IP address
  - A knows IP address of first hop router, R (how?)
  - A knows R's MAC address (how?)





6.3.2 Etherent

<u>brief introduction</u>: dominant wired LAN technology

first widely used LAN technology

simpler, cheap

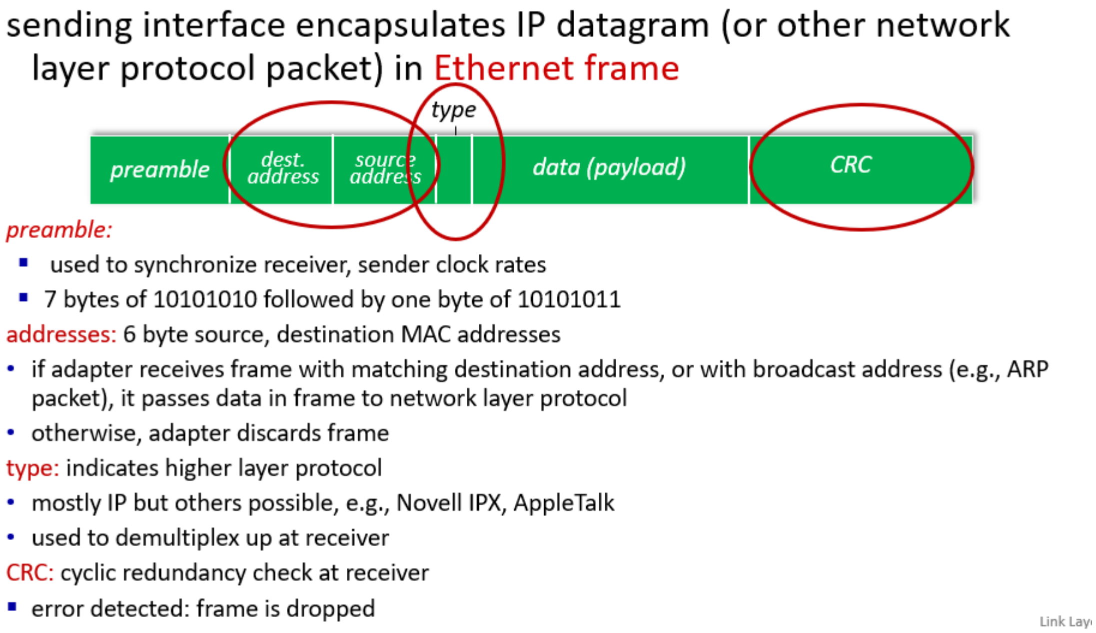
kept up with speed race: 10 Mbps – 400 Gbps

single chip, multiple speeds (e.g., Broadcom BCM5761)

1. physical topology

bus (coaxial cable): popular through mid 90s
    all nodes in same collision domain (can collide with each other)
switched: prevails today
    active link-layer 2 switch in center
    each “spoke” runs a (separate) Ethernet protocol (nodes do not collide with each other)

2. frame structure



SFD (Start Frame Delimiter): one byte of 10101011 which marks the end of the preamble and indicates the start of the frame; this is also called the unique synchronization byte

Type: 2 bytes long

CRC: 4 bytes long

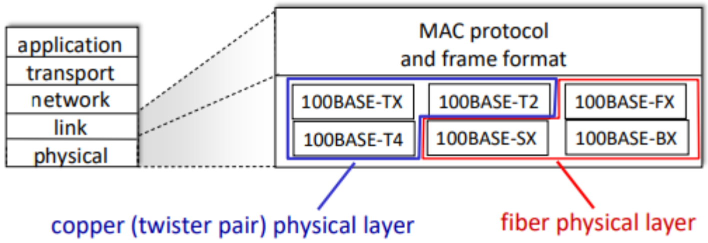
3. Ethernet characteristics

connectionless: no handshaking between sending and receiving NICs
unreliable: receiving NIC doesn’t send ACKs or NAKs to sending NIC
    data in dropped frames recovered only if initial sender uses higher layer rdt (e.g., TCP), otherwise dropped data lost

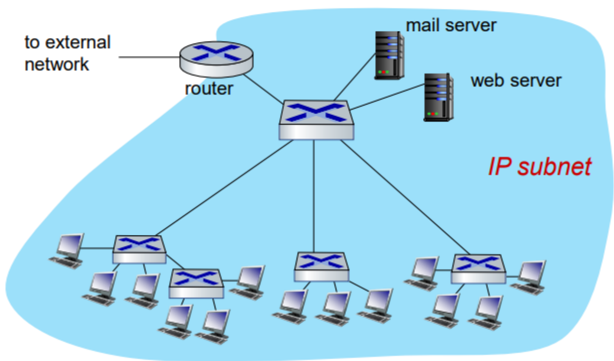
Ethernet’s MAC protocol: unslotted **CSMA/CD with binary backoff**

> The binary backoff algorithm specifies that the waiting time is chosen from a range of exponentially increasing values.

802.3 Ethernet standards: link & physical layers:
many different Ethernet standards:
common MAC protocol and frame format
different speeds: 2 Mbps, 10 Mbps, 100 Mbps, 1Gbps, 10 Gbps, 40 Gbps
different physical layer media: fiber, cable



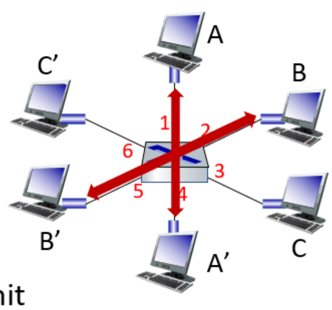
6.3.3 switches

> 为什么routers要运行那么复杂的路由算法，但是switches却什么都不需要?
> ——路由器连接成了一个复杂的网络（graph），switches连接的是tree，路径唯一，所以不需要复杂的路径
>
> 
>

switch: a link-layer device
store, forward Ethernet frames
examine incoming frame's MAC address and selectively forward them
use CSMA/CD to access segment
hosts **unaware** of presence of switches
do not need to be configured

1. multiple simultaneous transmission

- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, so:
  - no collisions; full duplex
  - each link is its own collision domain
- switching: A-to-A' and B-to-B' can transmit simultaneously, without collisions
  - but A-to-A' and C to A' can *not* happen simultaneously



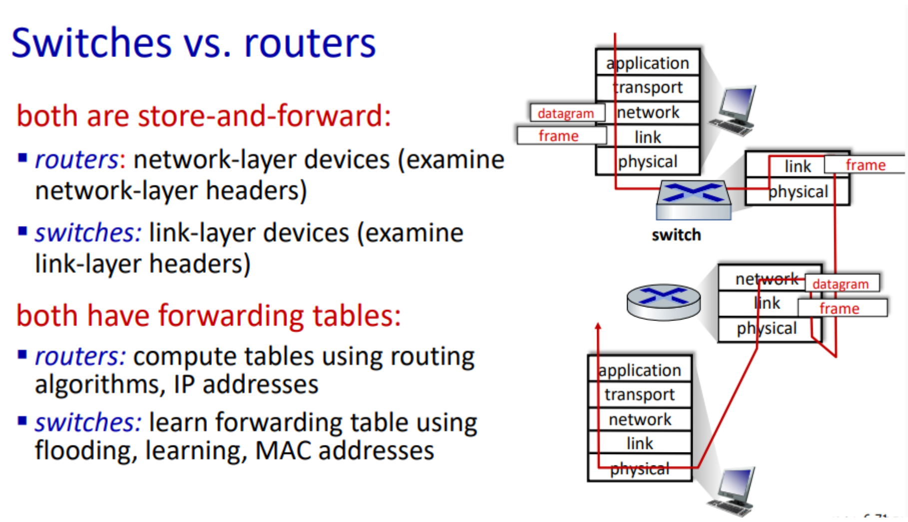
switch with six interfaces (1,2,3,4,5,6)

- each switch has a **switch table**, each entry is like: (MAC address of host, interface to reach host, time stamp)
- fill the switch table through **self-learning**

$$
\begin{aligned}
&when\ frame\ received\ at\ switch:\\
&1.\ record\ switch\ table\ via\ MAC\ dest\ address\\
&2.\ \textbf{if}\ entry\ found\ for\ destination\ \textbf{then}\ \{\\
&\qquad \textbf{if}\ destination\ on\ same\ segment\ as\ sender\\
&\qquad\quad \textbf{then}\ drop\ frame\\
&\qquad \textbf{else}\ forward\ frame\ on\ interface\ indicated\ by\ entry\\
&\quad \textbf{else}\ flood\ /*\ forward\ on\ all\ interfaces\ except\ the\ sender\ */
\end{aligned}
$$

(ARP helps hosts communicate to each other within the same segment)

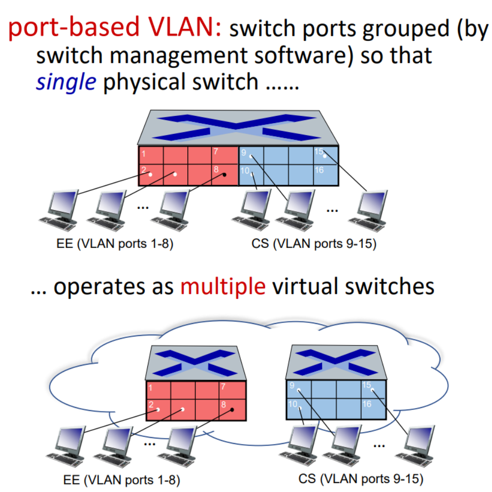
2. switches vs. routers



6.3.4 VLANs

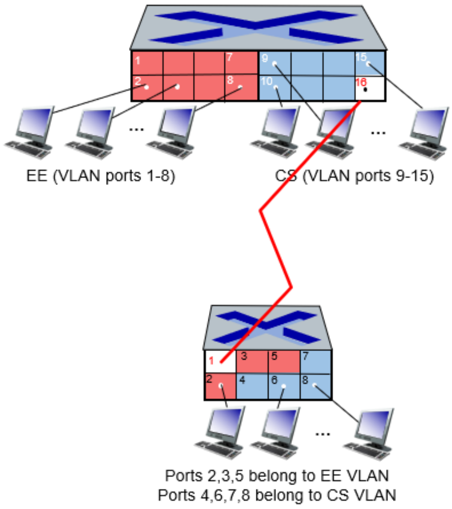
VLANs: Virtual LANs, or Virtual Local Area Network, switch(es) supporting VLAN capabilities can be configured to define multiple virtual LANS over single physical LAN infrastructure.

1. Port-based VLANs

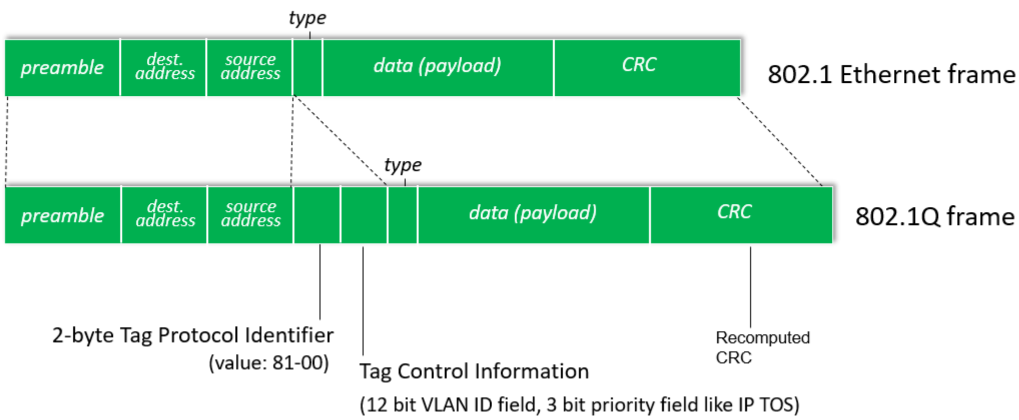


- traffic isolation: frames to/from ports 1-8 can *only* reach ports 1-8
    - can also define VLAN based on MAC addresses of endpoints, rather than switch port
- dynamic membership: ports can be dynamically assigned among VLANs
- forwarding between VLANS: done via routing (just as with separate switches)
    - in practice vendors sell combined switches plus routers

trunk port: carries frames between VLANS defined over multiple physical switches
- frames forwarded within VLAN between switches can't be vanilla 802.1 frames (must carry VLAN ID info)
- 802.1q protocol adds/removed additional header fields for frames forwarded between trunk ports



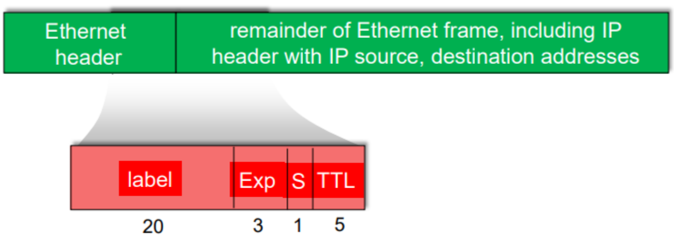
2. 802.1Q VLAN frame format



6.3.5 Link Virtualization: MPLS

MPLS: Multiprotocol Label Switching, operates between the IP routing layer and the MAC Ethernet link layer (classified as link layer), header has 32 bits

goal: use fixed length label (instead of shortest prefix matching of IP) for faster lookup within MPLS network, with the help of **MPLS capable routers** (a.k.a. label-switched router)
Also, MPLS allows the creation of VPNs by segregating traffic using different labels. This enables secure and isolated communication between different sites in a network.

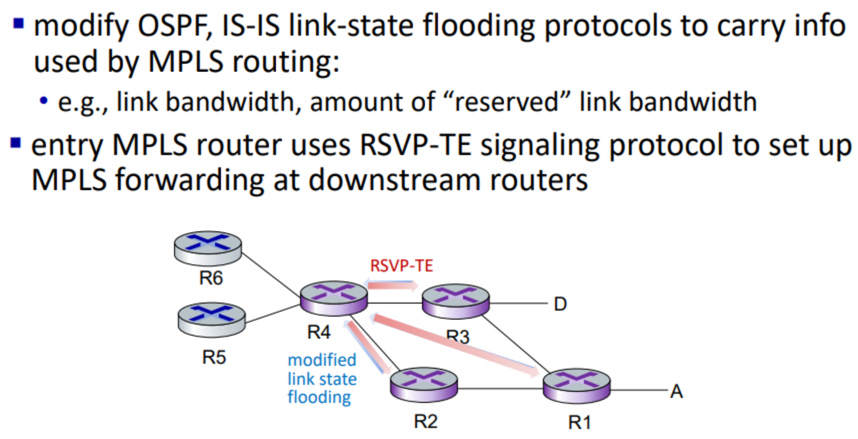


MPLS capable routers: forward packets to outgoing interface based only on **label** value (don't inspect IP address, thus its forwarding decision could **differ** from IP)

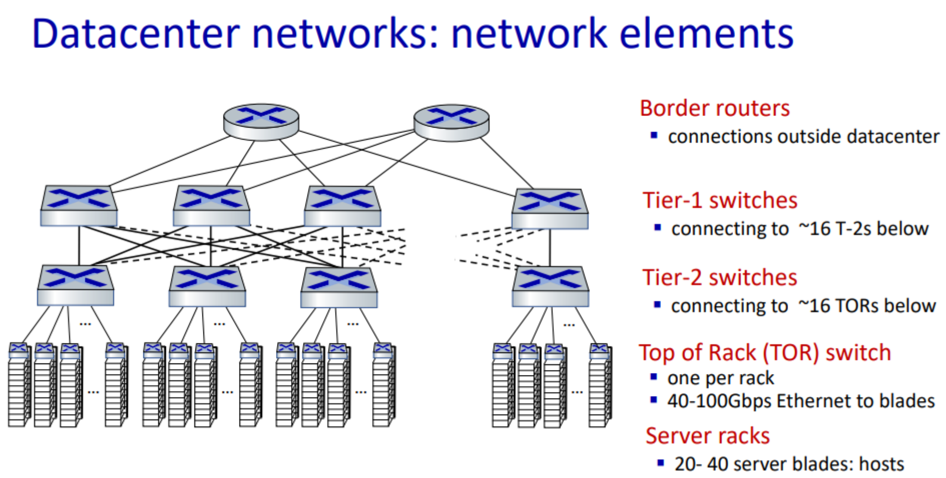
1. routing comparision with IP

IP routing: path to destination determined by destination address **alone**
MPLS routing: path to destination can be based on **source and** destination address
    flavor of generalized forwarding (MPLS 10 years earlier)
    fast reroute: precompute backup routes in case of link failure

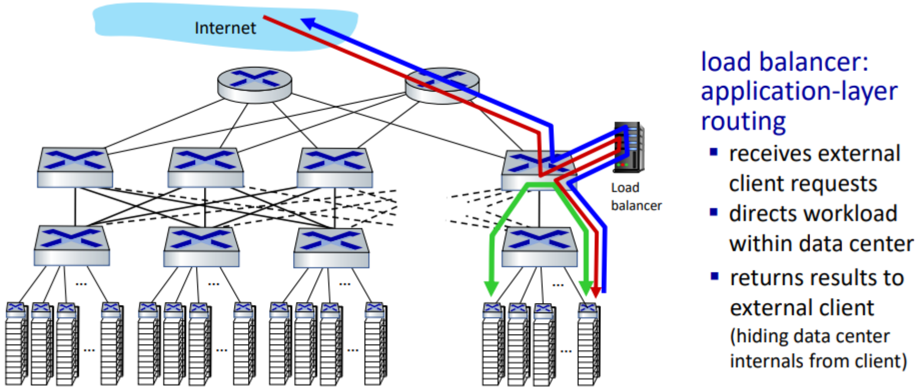
2. MPLS signaling



6.4 Data Center Networking



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

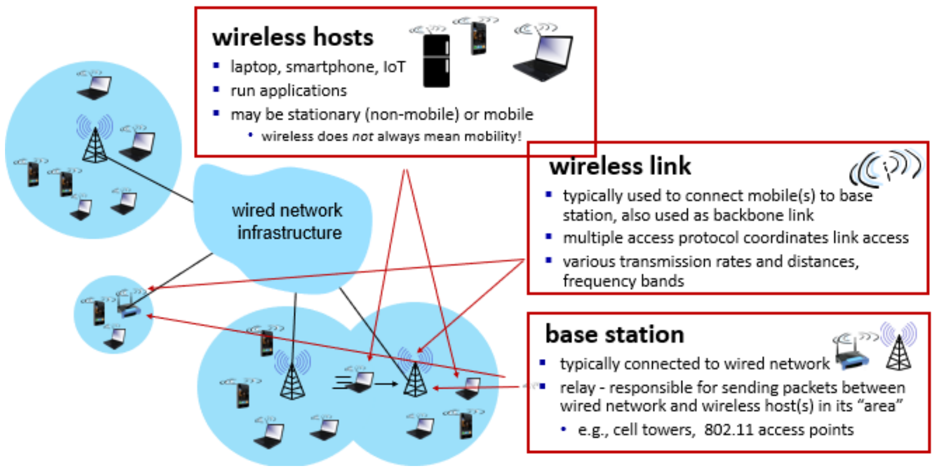


- **protocol innovations**
  link layer: RoCE: remote DMA (RDMA) over Converged Ethernet
  transport layer:
  ECN (explicit congestion notification) used in transport-layer congestion control (DCTCP, DCQCN)
  experimentation with hop-by-hop (backpressure) congestion control
  routing, management:
  SDN widely used within/among organizations' datacenters
  place related services, data as close as possible (e.g., in same rack or nearby rack) to minimize tier-2, tier-1 communication

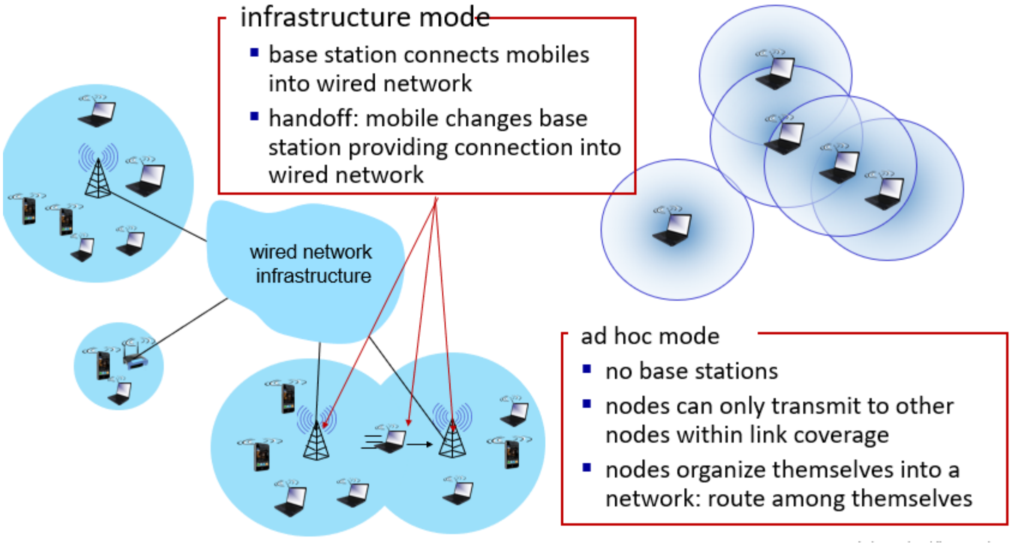
Chapter 7: Wireless and Mobile Networks

Two important challenges of wireless link: 1. wireless, 2. mobility (change point of attachment)

- **elements of a wireless network**: wireless hosts, wireless link, base station

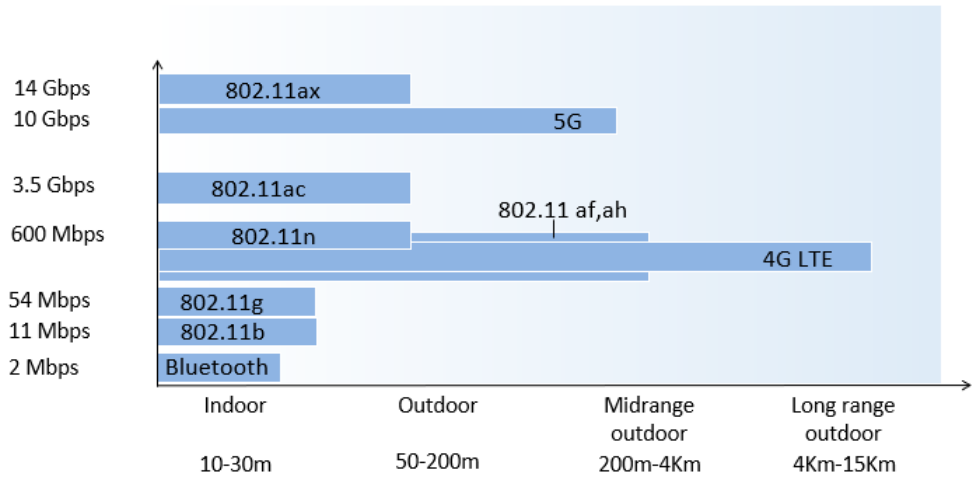


Two modes: infrastructure mode (base station), ad hoc mode (no base stations)



	single hop	multiple hops
infrastructure (e.g., APs)	host connects to base station (WiFi, 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

- **characteristics of selected wireless links**



7.1 Wireless

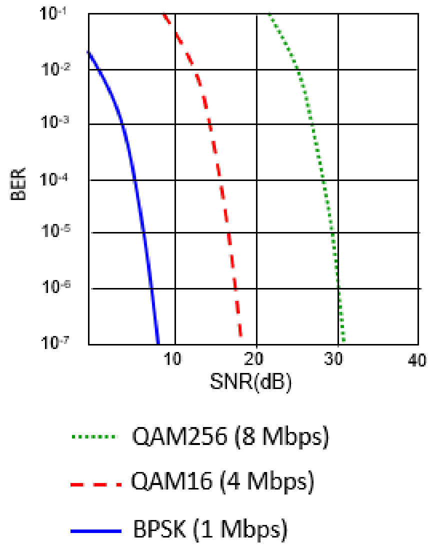
7.1.1 wireless links and network characteristics

- **Characteristics**:
  - **decreased signal strength**: radio signal attenuates as it propagates through matter (path loss)

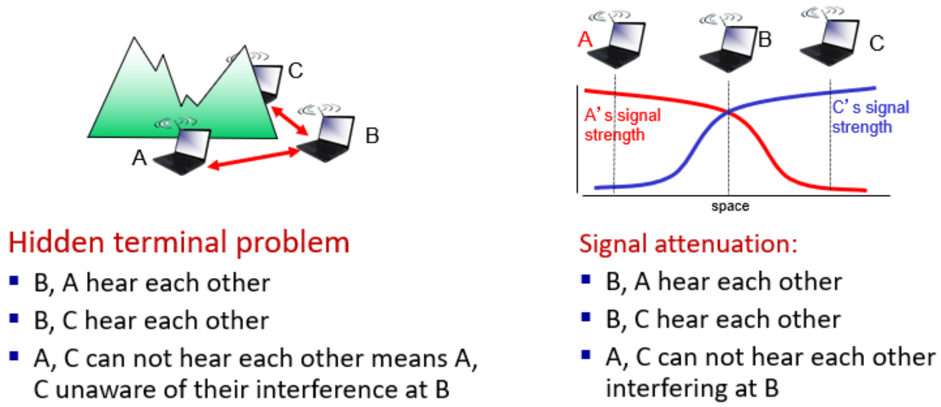
    SNR (Signal-to-Noise Ratio): larger SNR => easier to extract signal from noise
    BER (Bit Error Rate)

    - <u>given physical layer</u>: increase power => increase SNR => decrease BER
    - <u>given SNR</u>: choose physical layer that meets BER requirement, giving highest throughput

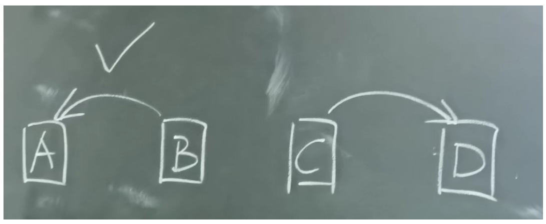
SNR may change with mobility: dynamically adapt physical layer (modulation technique, rate)



- **interference from other sources**: wireless network frequencies (e.g., 2.4 GHz) shared by many devices (e.g., WiFi, cellular, motors)
  - Hidden terminal problem 隐藏站问题 + Signal attenuation



  - Exposed terminal problem 暴露站问题
    同一个AP下因为“怕错传”而导致的效率低下



> B正在给A传文件，则C没法给D传文件

WiFi的CSMA/CA的协议解决了隐藏站问题和暴露站问题吗？解决了Hidden terminal problem，但是没有解决Exposed terminal problem

- **multipath propagation**: radio signal reflects off objects ground, arriving at destination at slightly different times

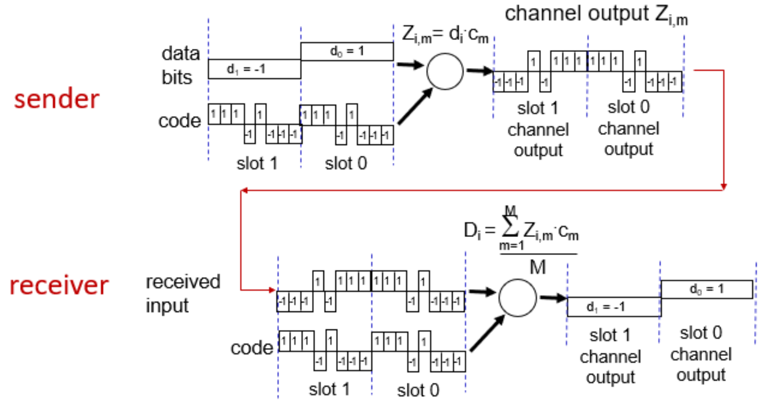
1. **CDMA / Code Division Multiple Access**

- **unique "code"** assigned to each user, i.e., code set partitioning
  all users share same frequency, but each user has own “chipping” sequence (i.e., code) to encode data
  allows multiple users to “coexist” and transmit simultaneously with minimal interference

(if codes are “orthogonal”)

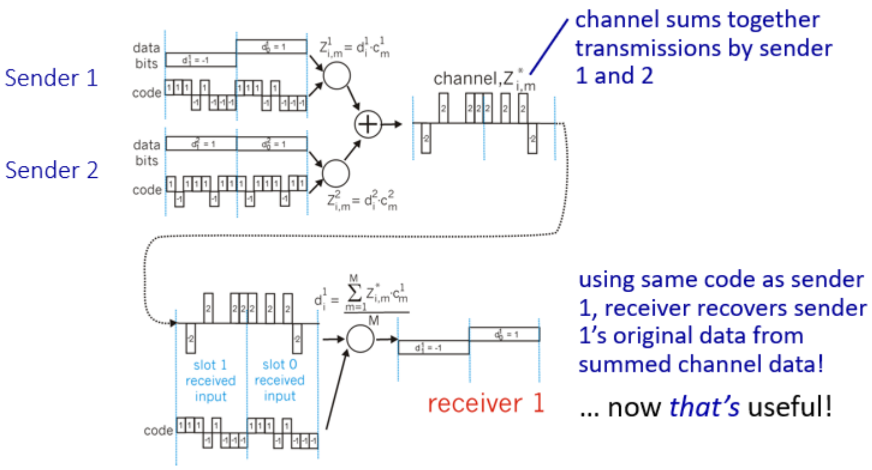
encoding: inner product: (original data) X (chipping sequence)
decoding: summed inner-product: (encoded data) X (chipping sequence)

CDMA encode/decode



... but this isn’t really useful yet!

CDMA: two-sender interference



The codes for the two sender are ORTHOGONOL

2. CDMA其它

同步CDMA、异步CDMA

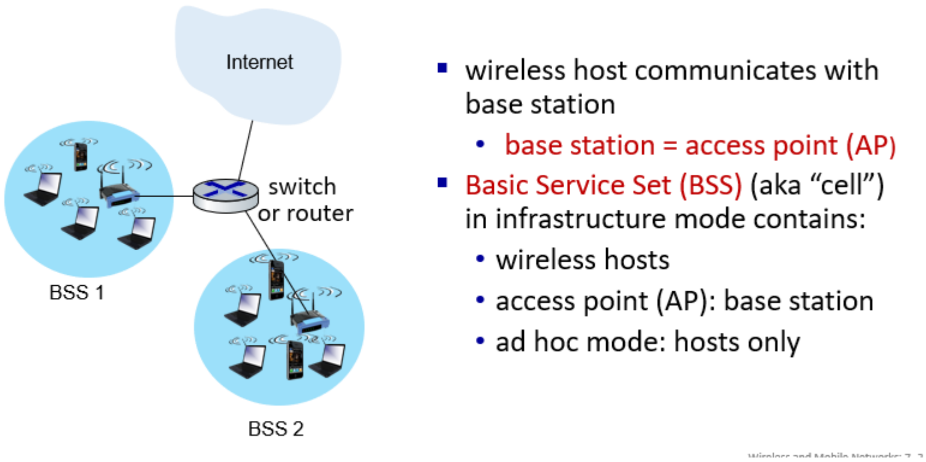
7.1.2 WiFi: 802.11 wireless LANs

IEEE 802.11 standard	Year	Max data rate	Range	Frequency
802.11b	1999	11 Mbps	30 m	2.4 Ghz
802.11g	2003	54 Mbps	30m	2.4 Ghz
802.11n (WiFi 4)	2009	600	70m	2.4, 5 Ghz
802.11ac (WiFi 5)	2013	3.47Gpbs	70m	5 Ghz
802.11ax (WiFi 6)	2020 (exp.)	14 Gbps	70m	2.4, 5 Ghz
802.11af	2014	35 – 560 Mbps	1 Km	unused TV bands (54-790 MHz)
802.11ah	2017	347Mbps	1 Km	900 Mhz

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

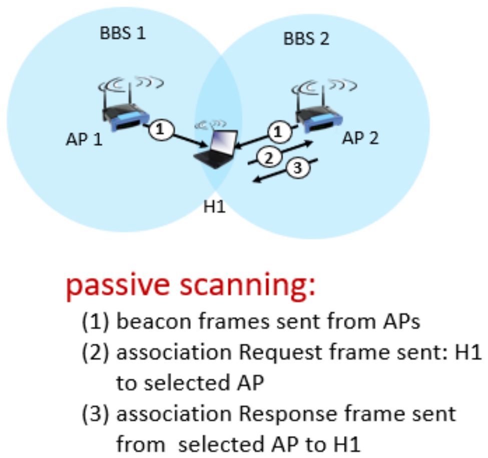
1. 802.11 LAN architecture

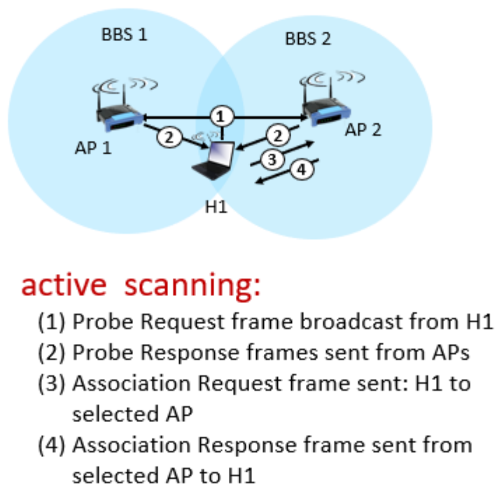
Basic Service Set / BSS: contains wireless hosts, access points (AP, or base station) (if ad hoc mode: hosts only)



2. channel

- spectrum divided into channels at different frequencies
  AP admin chooses frequency for AP
  interference possible: channel can be same as that chosen by neighboring AP!
- arriving host: must **associate** with an AP
  scans channels, listening for *beacon frames* containing AP's name (SSID) and MAC address, selects AP to associate with
  then may perform authentication
  then typically run DHCP to get IP address in AP's subnet
- **passive / active scanning**





3. MAC (Medium Access Control) Protocol: CSMA/CA

WiFi没有Collision detection，只有Collision avoidance

> avoid collisions (2+ nodes transmitting at same time):
> 802.11: CSMA - sense before transmitting
> don't collide with detected ongoing transmission by another node
> 802.11: no collision detection
> difficult to sense collisions: high transmitting signal, weak received signal due to fading

> can't sense all collisions in any case: hidden terminal, fading
> goal: avoid collisions: **CSMA / Collision Avoidance**

802.11 sender:

1. if sense channel idle for **DIFS** (Distributed Inter-Frame Space), then transmit entire frame (no CD)

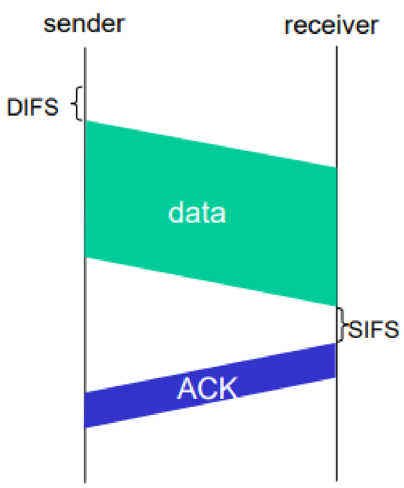
   > DIFS is a time interval that needs to elapse before a device can transmit a frame if it senses the channel as idle. It is a form of a waiting period that helps prevent collisions and ensures fair access to the channel.
   >
   > When a device has data to transmit and senses the channel as idle for at least the duration of DIFS, it can proceed to transmit the entire frame **without** using Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) with collision detection (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:

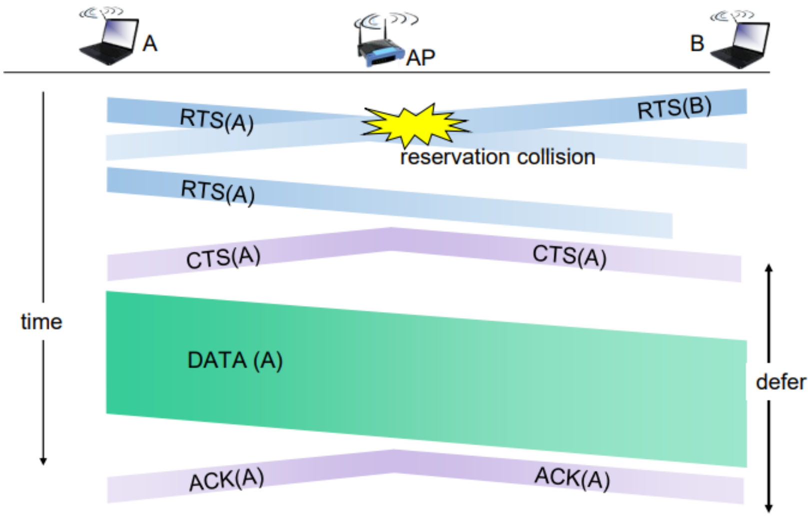
1. if frame received OK, then return ACK after **SIFS** (Short Inter-Frame Space) (ACK needed due to hidden terminal problem)

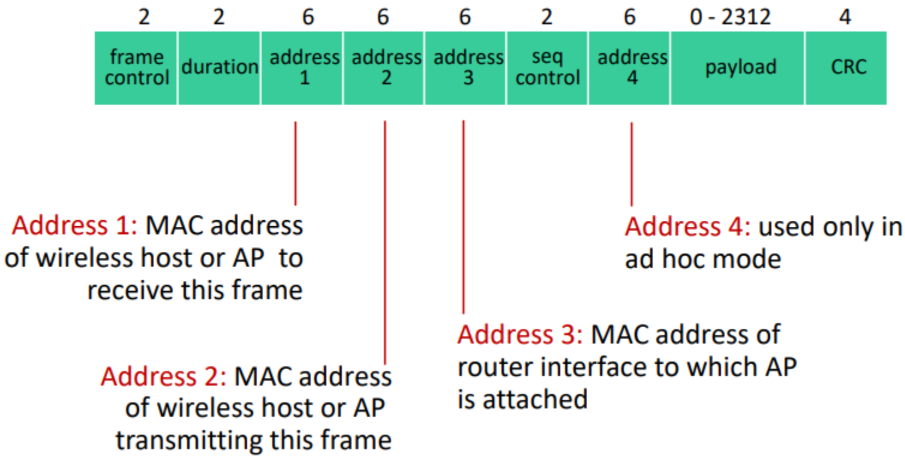
   > When a device receives a frame successfully and needs to send an ACK, it waits for a period equal to SIFS before transmitting the ACK frame. This helps reduce delays and ensures prompt acknowledgment of received frames, addressing issues like the hidden terminal problem.



- addition collision avoidance: **small RTS / request-to-send packet**

  sender first transmits small request-to-send (RTS) packet to BS using CSMA
  BS broadcasts clear-to-send **CTS** in response to RTS
  CTS heard by all nodes: sender transmits data frame + other stations defer transmissions

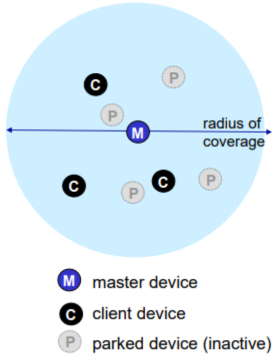




4. advanced capabilities

- **mobility**
  how does switch know which AP is associated with specific host?
  ——self-learning: switch will see frame from H1 and "remember" which switch port can be used to reach H1
- **rate adaptation**
  base station, mobile dynamically change **transmission rate** (physical layer modulation technique) as mobile moves, SNR varies
  1 SNR decreases, BER increase as node moves away from base station
  2 When BER becomes too high, switch to lower transmission rate but with lower BER
- **power management**
  beacon frame: contains list of mobiles with AP-to-mobile frames waiting to be sent
  node will stay awake if AP-to-mobile frames to be sent; otherwise sleep again until next beacon frame
- **personal area networks: Bluetooth**
  less than 10 m diameter
  replacement for cables (mouse, keyboard, headphones)
  ad hoc: no infrastructure
  master controller / clients devices: master polls clients, grants requests for client transmissions

- TDM, 625 μsec sec. slot
- FDM: sender uses 79 frequency channels in known, pseudo-random order slot-to-slot (spread spectrum)
  - other devices/equipment not in piconet only interfere in some slots
- parked mode: clients can "go to sleep" (park) and later wakeup (to preserve battery)
- bootstrapping: nodes self-assemble (plug and play) into piconet



7.1.3 Cellular networks: 4G and 5G

the solution for wide-area mobile Internet
technical standards: 3rd Generation Partnership Project (3GPP)
4G: Long-Term Evolution (LTE) standard

Key differences from wired internet:
different wireless link layer
mobility as a 1st class service
user "identity" (via SIM card)
business model: users subscribe to a cellular provider
  strong notion of "home network" versus roaming on visited nets
  global access, with authentication infrastructure, and inter-carrier settlements

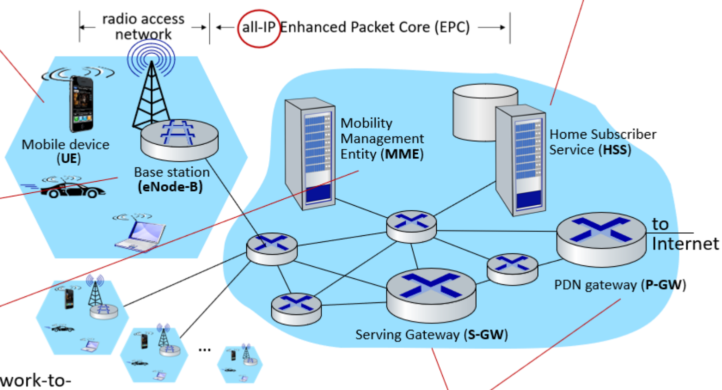
1. 4G LTE architecture

Mobile device:
- smartphone, tablet, laptop, IoT, ... with 4G LTE radio
- 64-bit International Mobile Subscriber Identity (IMSI), stored on SIM (Subscriber Identity Module) card
- LTE jargon: User Equipment (UE)

Base station:
- at "edge" of carrier's network
- manages wireless radio resources, mobile devices in its coverage area ("cell")
- coordinates device authentication with other elements
- similar to WiFi AP but:
  - active role in user mobility
  - coordinates with nearly base stations to optimize radio use
- LTE jargon: eNode-B

Mobility Management Entity
- device authentication (device-to-network, network-to-device) coordinated with mobile home network HSS
- mobile device management:
  - device handover between cells
  - tracking/paging device location
- path (tunneling) setup from mobile device to P-GW

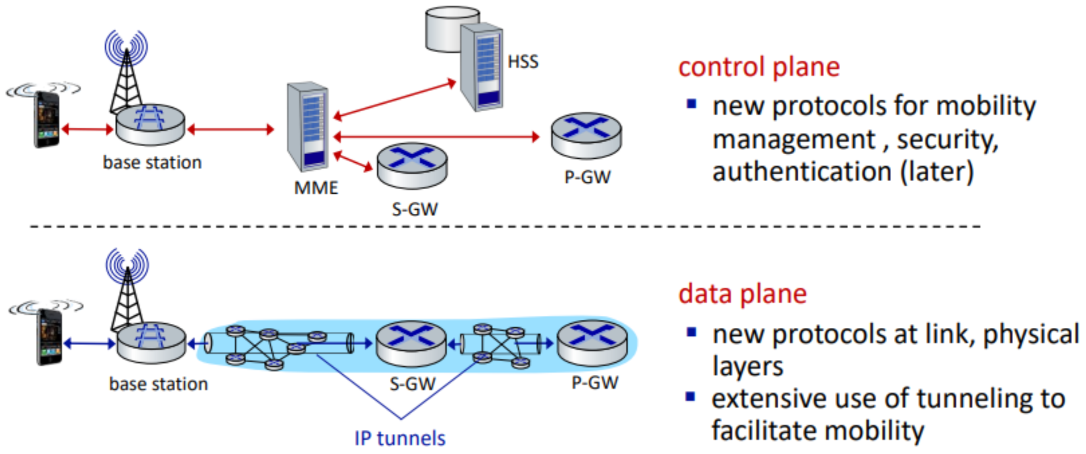
Home Subscriber Service
- stores info about mobile devices for which the HSS's network is their "home network"
- works with MME in device authentication



Serving Gateway (S-GW), PDN Gateway (P-GW)
- lie on data path from mobile to/from Internet
- P-GW: 1. gateway to mobile cellular network 2. Looks like nay other internet gateway router 3. provides NAT services
- other routers: extensive use of tunneling

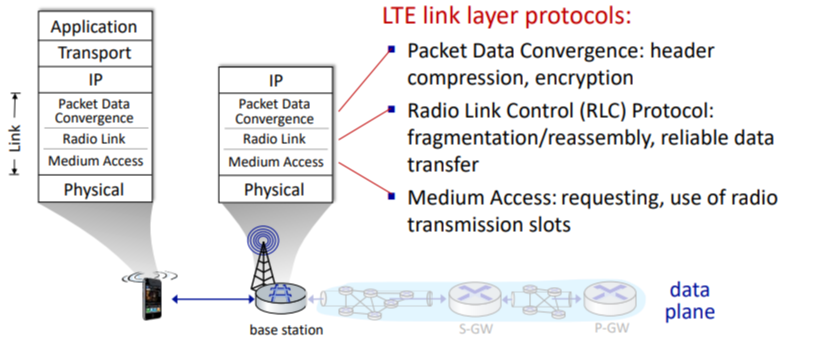
Wireless and Mobile Networks: 7- 39

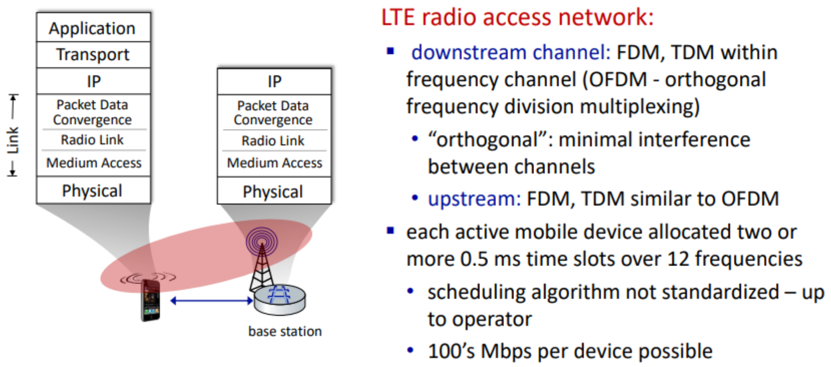
2. LTE: data plane & control plane separation



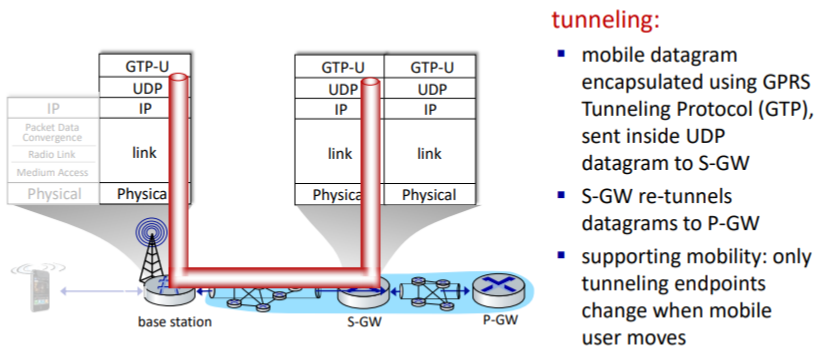
- **LTE data plane protocol stack**:

  First hop:

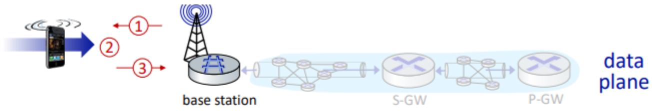




  Packet core:

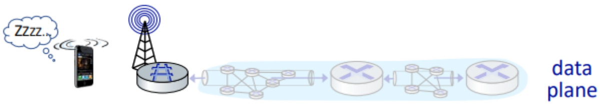


  association with a BS:



1. BS broadcasts primary synch signal every 5 ms on all frequencies
   - BSs from multiple carriers may be broadcasting synch signals
2. mobile finds a primary synch signal, then locates 2nd synch signal on this freq.
   - mobile then finds info broadcast by BS: channel bandwidth, configurations; BS's cellular carrier info
   - mobile may get info from multiple base stations, multiple cellular networks
3. mobile selects which BS to associate with (*e.g.*, preference for home carrier)
4. more steps still needed to authenticate, establish state, set up data plane

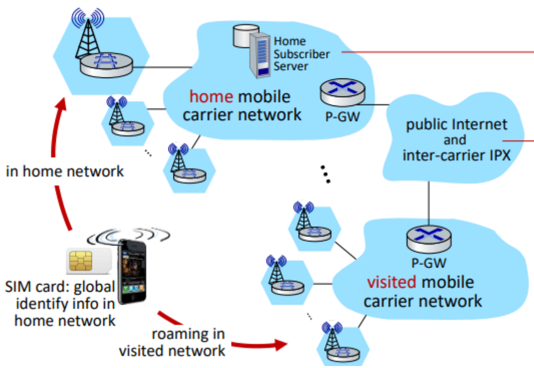
sleep mode:



as in WiFi, Bluetooth: LTE mobile may put radio to "sleep" to conserve battery:

- light sleep: after 100's msec of inactivity
  - wake up periodically (100's msec) to check for downstream transmissions
- deep sleep: after 5-10 secs of inactivity
  - mobile may change cells while deep sleeping – need to re-establish association

- **global cellular network**: a network of IP networks



home network HSS:

- identify & services info, while in home network and roaming

all IP:

- carriers interconnect with each other, and public internet at exchange points
- legacy 2G, 3G: not all IP, handled otherwise

(*) 4G/5G: TDMI + FDMI

7.2 Mobility