CSEE4119 Computer Networks

Chapter 1 - Computer Network and the Internet

Introduction

Networking is ubiquitous. We use daily for example

- Internet
- WWW
- Wi-Fi
- LTE Network

Also, the challanges that remain:

- Security
- Reliable service in dynamic environment
- Rich near-real time interactive content
- Critical services

Networking challange

- How to support diverse and ever-changing use cases?
- How to provide near-real-time interactive content?

> 即如何克服用户的多元化，即多种使用情景；以及最大程度降低网络延迟

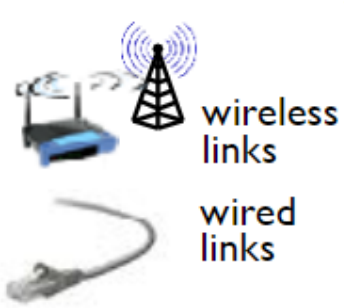
What will study

- Application Layer 应用层
  - How to build applications that span multiple computers and use internet to communicate
  - How common Internet applications work (web, video, DNS)
- Transport Layer 传输层
  - How to deliver message accross Internet, including:
    - Establishing communication between computers 计算机之间建立联系
    - Reliably delivering messages 高保障地传送讯息
    - Avoiding congestion despite many uncorrdinated senders 在有过多发送者时避免拥堵
- Network Layer 网络层
  - How to steer data from source to destination 如何将数据从源头引导到目的地
- Link Layer 链接层
  - How to transfer data between direct neighbors, wired and wireless

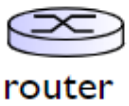
What is Internet

Nuts and bolts view 实质内容

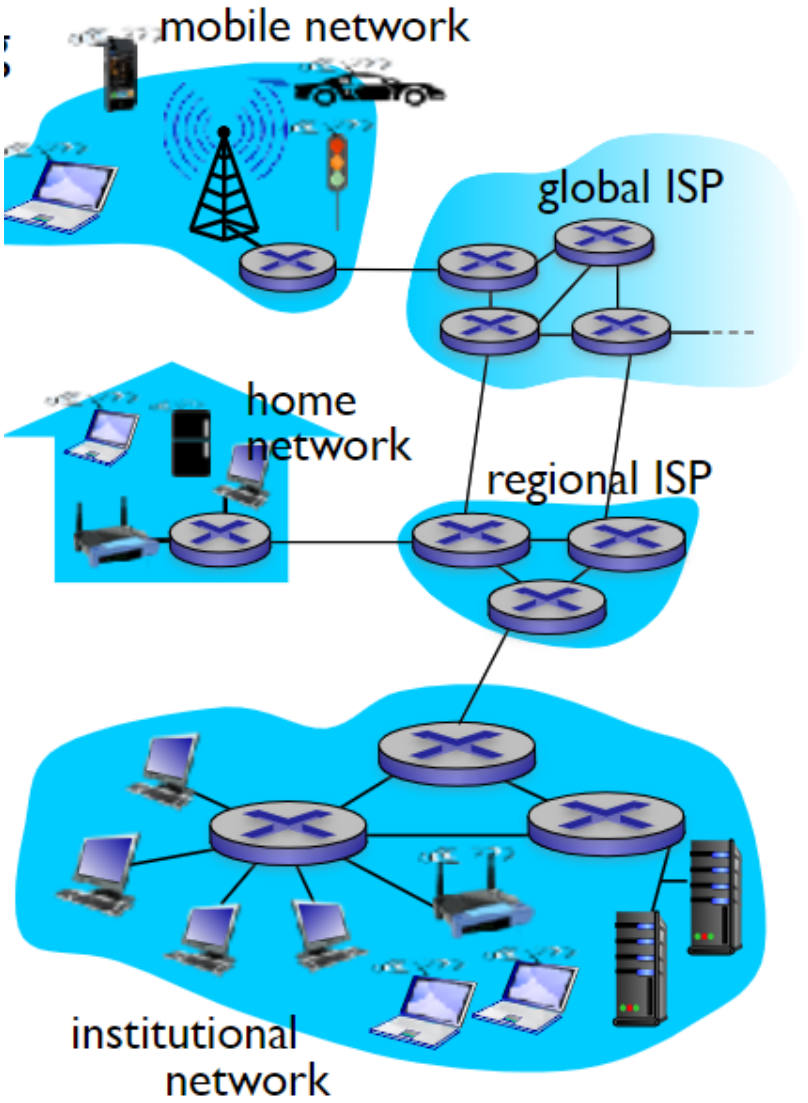
- Billions of connected computing devices (PC, laptop, phones)
  - Hosts = end system 终端
  - running network applications



- Communication links
  - via fiber 光纤, copper 铜线, radio 广播讯号. satellite 卫星讯号
  - transmission rate: bandwidth



- Packet Switches
  - Forward packets, which is chunks of data by two ways:
    - network layer routers (not the router in home)
    - link-layer switches



- ISPs: Internet Service Provider. 互联网服务供应商
  - The interconnect ISPs is the network of the Internet
- Protocols control sending, receiving of messages
  - e.g. TCP, IP, HTTP, 802.11

Service view 服务内容

- Internet is an infrastructure 基础设施 that provides services to applications:
  - Web, VoIP, email, games…
- Internet provides programming interface to applications 为应用程序提供编程接口
  - Hooks that allow sending and receiving applications to connect to Internet

    允许发送和接收应用程序链接到web的钩子

  - provides service options, like postal service:
    - Reliable data delivery from source to destination
    - Unreliable “best effort” data delivery

Nuts and bolts view

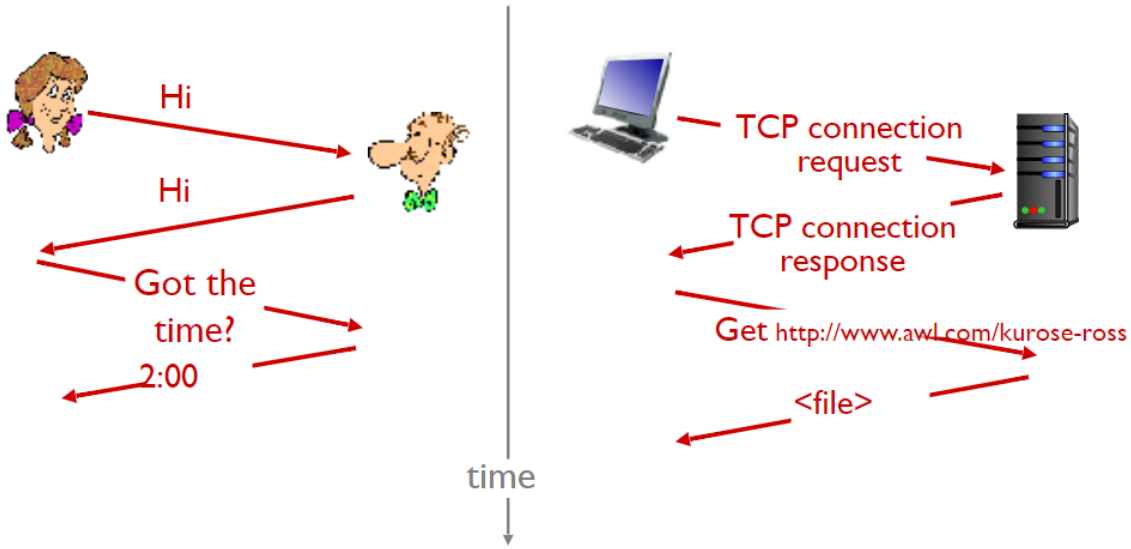
Internet: "network of networks"

Protocols control sending, receiving messages

What is protocol

- Protocols define **format**, **order of messages sent and received** among network entities, and **actions taken** on message transmission, receipt
- All communication activity in Internet governed by protocols

> 协议规定了网络实体之间发送和接收信息的格式和顺序，以及在信息传输、接收时采取的行动。



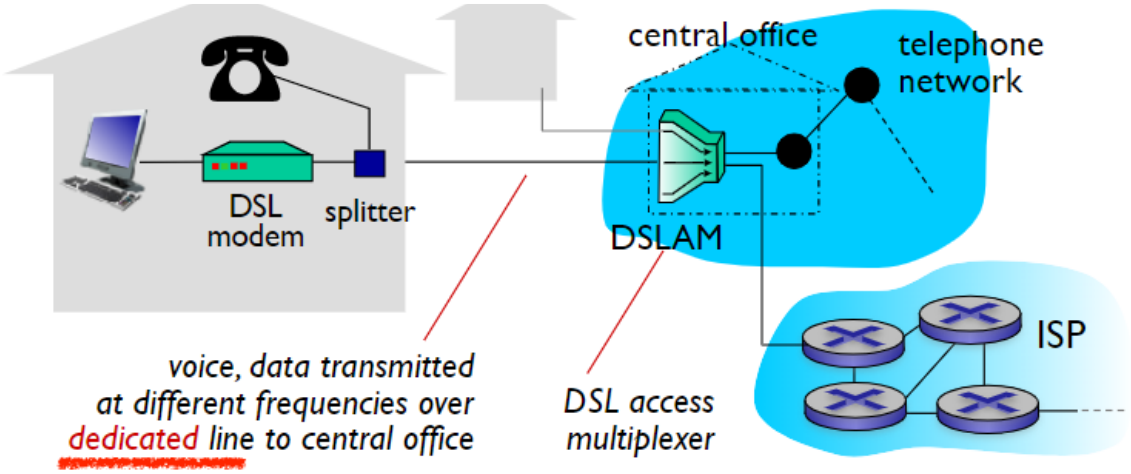
💡 就像人之间的对话一样，协议的目的是规范请求和相应之间的正确性

Network Edge

- Network edge:
  - Hosts: clients and servers
  - servers often in data centers
- Access networks, physical media:
  - wired, wireless communication links
  - How to connect end system (mobile, computer) to edge router?
    - residential access nets
    - institutional access network (schools, company)
    - mobile access network (基站)
- Network core:
  - Interconnected routers
  - network of network (ISP)

> How to connect end system (client) to edge router?
>
> - Residential access nets
> - institutional access networks (school, company)
> - mobile access networks

Access Network: Digital subscriber line (DSL)

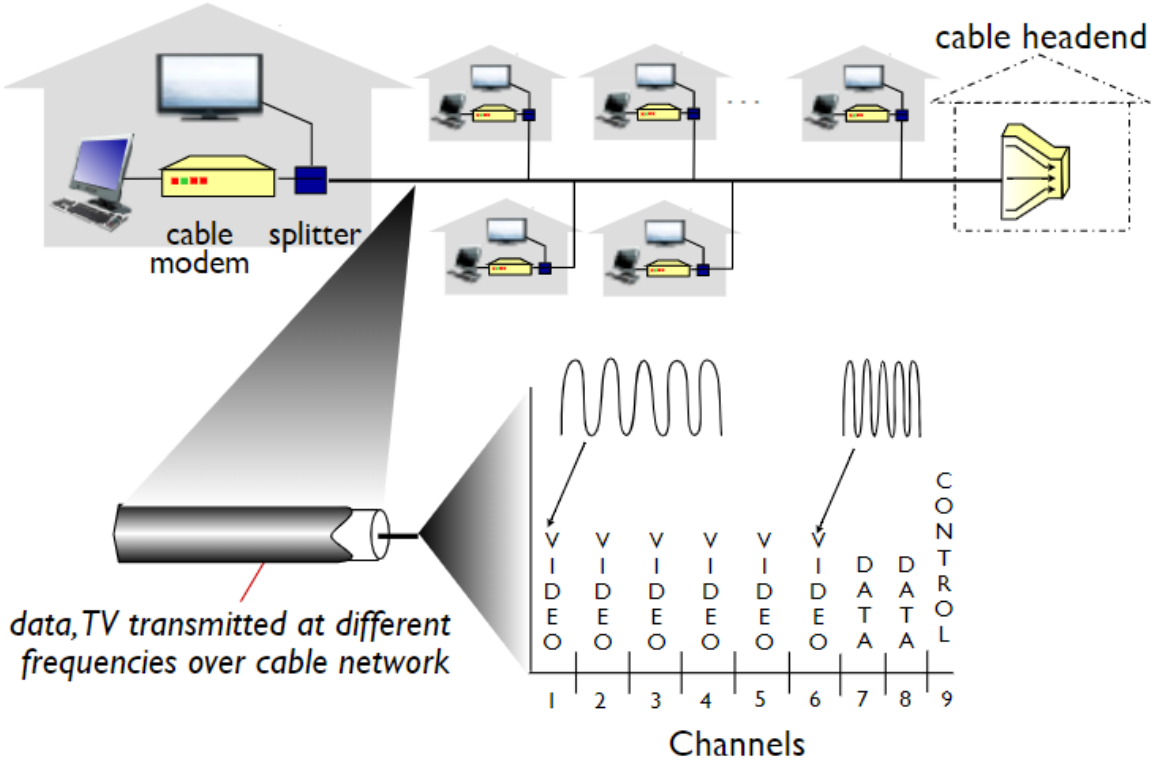


💡 Key network challange: How to support new applications on deployed technologies? 如何在已经部署的技术上支持最新的应用?

- Use existing telephone line to central office DSLAM
  - data over DSL phone line goes to Internet
  - voice over DSL phone line goes to telephone net
- < 16 Mbps upstream transmission rate (typically < 1Mbps)
- < 52 Mbps downstream transmission rate (typically < 10Mbps)
- Also a two-way telephone channel
- The upload/download/telephone channels use different frequencies

国内大概十几年前的电话拨号线路。上网方式和打电话类似，计算机拨打特定的电话号码上网。由于是和固定电话共用线路，所以需要在上网的时候把电话线从固定电话上拔下来。同时在上网时也不能打电话。但是在之后有既可以打电话又可以上网的存在，即通过splitter或者部署两条电话线完成。

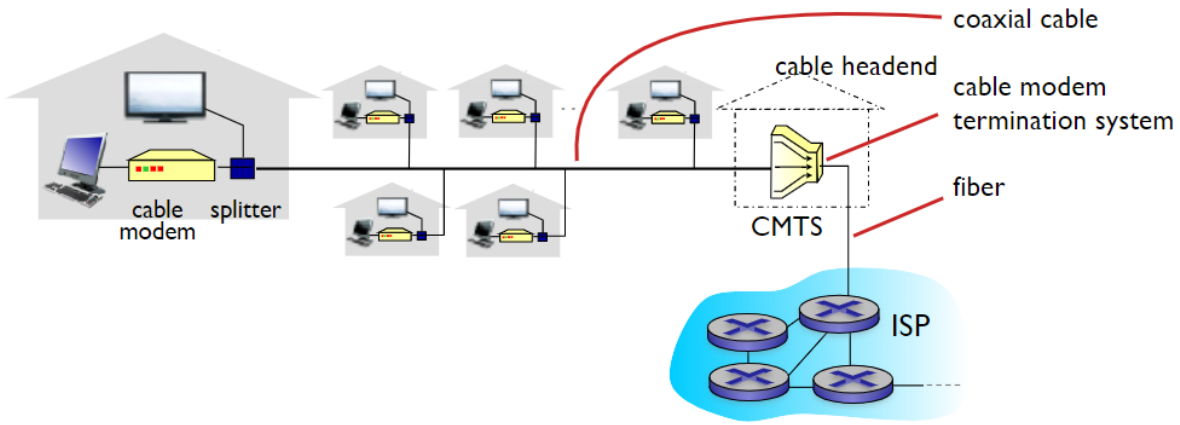
Access Network: Cable network



- Using frequency division multiplexing 频分复用

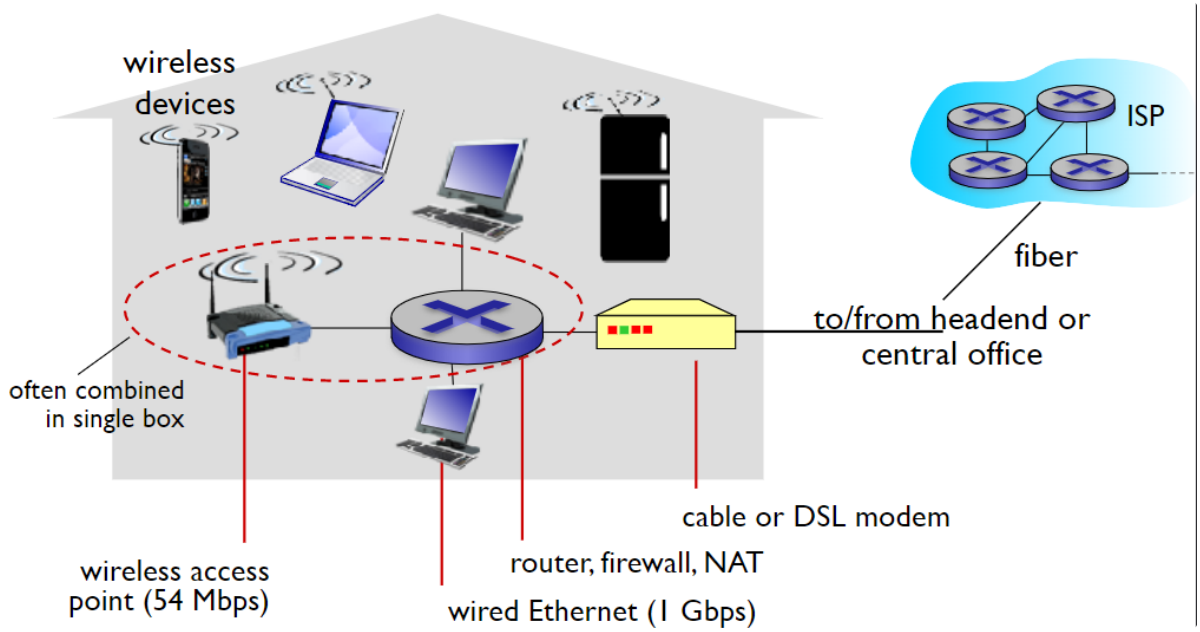
    - different channels transmitted in different frequency bands
- Difference between Cable and DSL:
    - Cable network shared its line, but DSL is dedicated

💡 Key networking challange: How to efficiently share a common medium? 如何高效的共享媒介?



- Network of cable, fiber attaches home to ISP router
    - homes share access network to cable headend
        - unlike DSL, which has dedicated access to central office
- Hybrid fibber coax (HFC) 光纤同轴电缆
    - CMTS controls access of cable modems to shared coax cable segment

Access network: home network



💡 Key networking challange: How to connect different types of networks? 如何连接不同类型的网络?

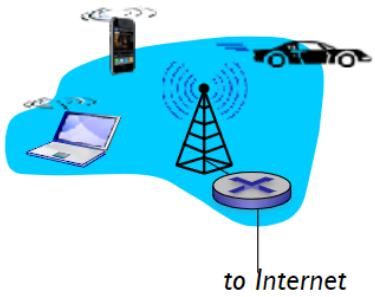
Wireless access network

wireless LANs:

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

wide-area wireless access

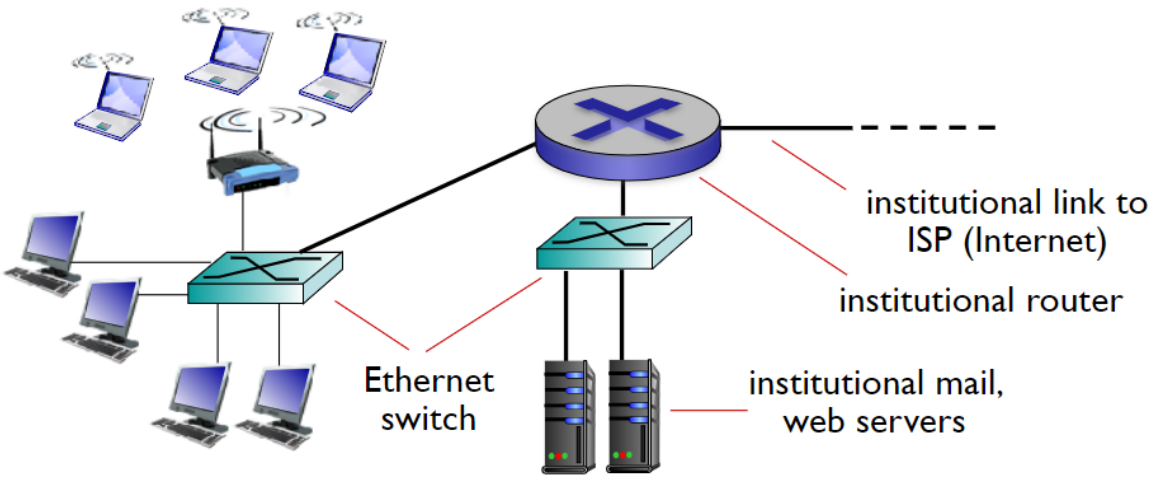
- provided by telco (cellular) operator, 10's km
- up to 100s of Mbps
- 3G, 4G: LTE, 5G



- shared wireless access network connects end system to router via base station, also called "access point"

无线接入包含了路由器的WiFi和广域范围内的无线信号基站。二者都可以被称为无线基站接入。

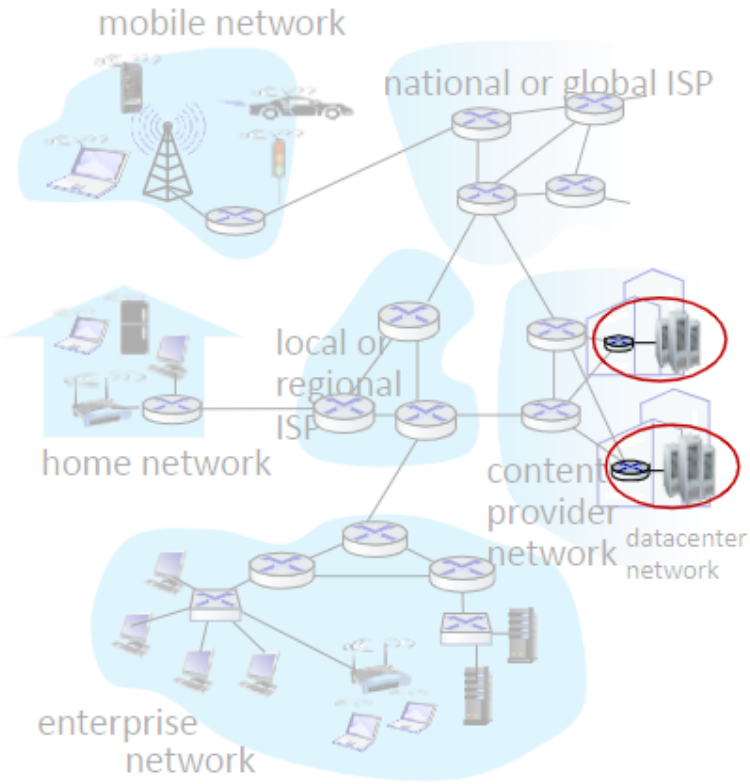
Enterprise access network



- Used in companies, universities, etc
- mix of wired, wireless link technologies, connecting a mix of switches and routers

💡 由于企业级网络接入需要覆盖大范围的室内面积，所以不能使用讯号基站。而是使用多个路由器和交换器，将无线网络拓展到每栋楼的每一层。交换器主要提供有线网络接入和连接路由器。路由器提供无线网络接入。同时交换器也将公司或大学的专用服务器接入。

Access networks: data center networks



- High-bandwidth links (10s to 100s Gbps) connect hundreds to thousands of servers together, and to internet

Hosts: send packets of data

Host sending function:

- takes applications message
- breaks into smaller chunks, known as packets, of length L bits
- transmits packet into access network at transmission rate R, also known as link capacity or link bandwidth

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

Physical Media

- bit: propagates between transmitter/receiver pairs 比特是在接收器和发射器之间传输的
- physical link: what lies between transmitter & receiver 物理链路是连接接收器和发射器的
- guided media: signals propagare in solid media: copper, fiber, coax 导向介质是物理链路的材料

Physical Media: twisted pair, coax, fiber

- Twisted pair (TP) 双绞线
  - two insulated copper wires
    - Category 5: 100 Mbps
    - Category 6: 10 Gbps

- Coaxial cable 同轴电缆
  - two concentric copper conductors 两个同心的铜导体
  - bidirectional
  - broadband:
    - multiple frequency channels on cable
    - HFC

- Fiber optic cable
  - glass fiber carrying light pulses, each pulse a bit
  - high-speed operation:
    - high speed point to point transmission
  - low error rate
    - repeater spaced far apart
    - immune to electromagnetic noise

Physical media: ratio

- Signal carried in electromagnetic spectrum
- No physical “wire”
  - no installation
  - penetrate walls
  - support mobility
- Bi-directional
- Propagation environment effects
  - reflection
  - obstruction by objects
  - interference

Radio link types

- Wireless LAN (WiFi)
  - 100-100's Mbps: 10's of meters
- Wide-area (4G cellular)
  - 10's Mbps over ~10km
- Bluetooth: cable replacement
  - short distance, limited rate
- Terrestrial microwate
  - Point-to-point, 45 Mbps cahnnels
- Satellite
  - Up to 45 Mbps channel
  - 270 msec end-end delay
  - geostationary versus low altitude

> Key networking challenge:
>
> How to reliably transmit data across unreliable network?

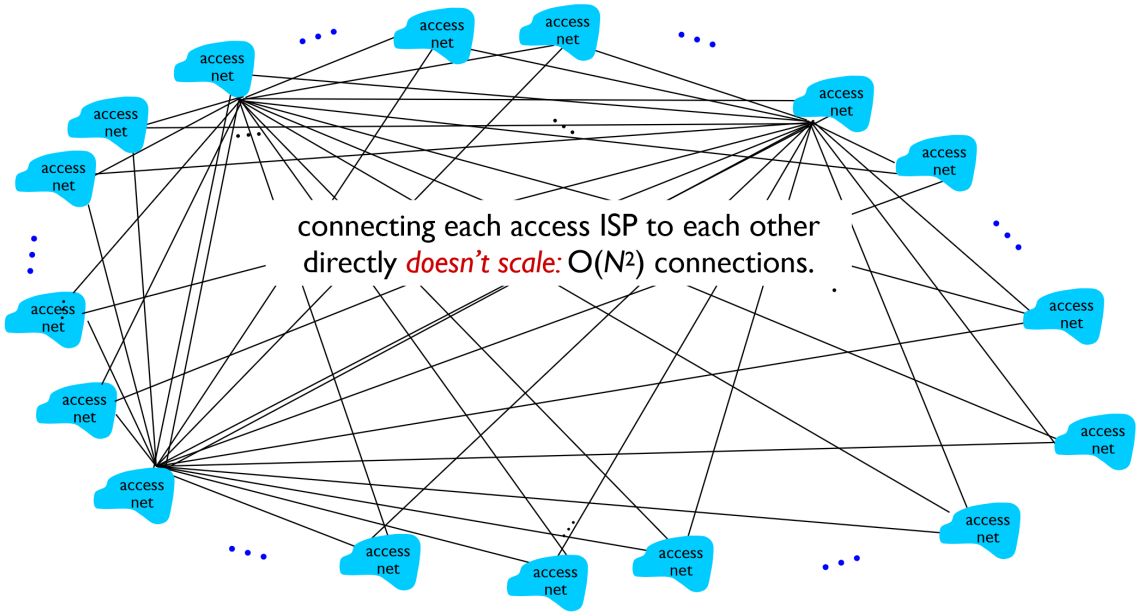
Network Core

Internet structure: network of networks - ISP

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

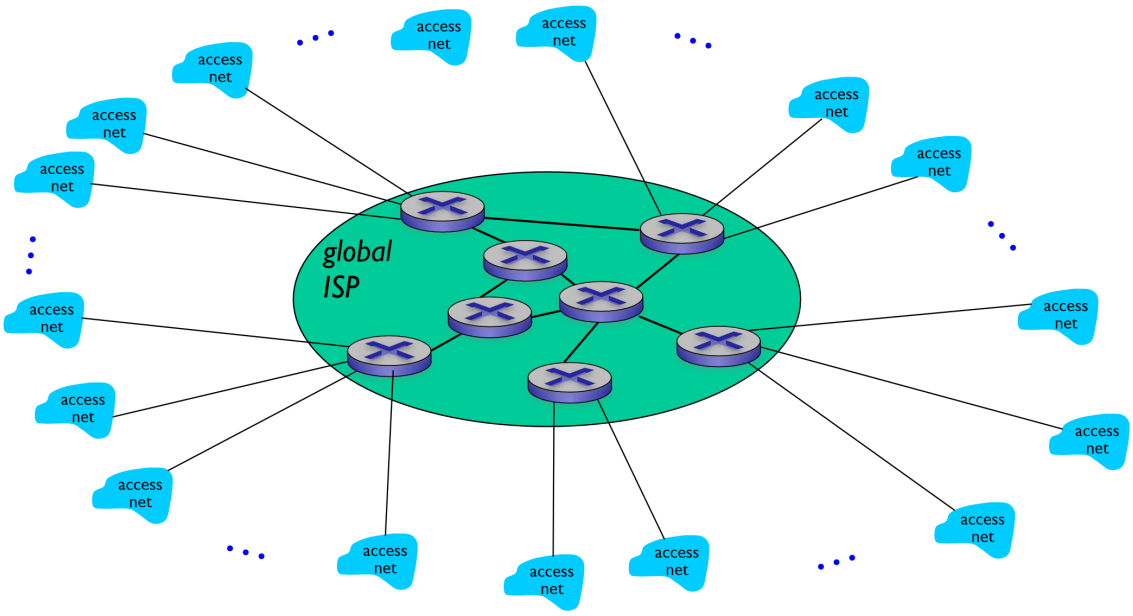
Two options to make ISPs interconnect:

1. connect each access ISP to every other access ISP



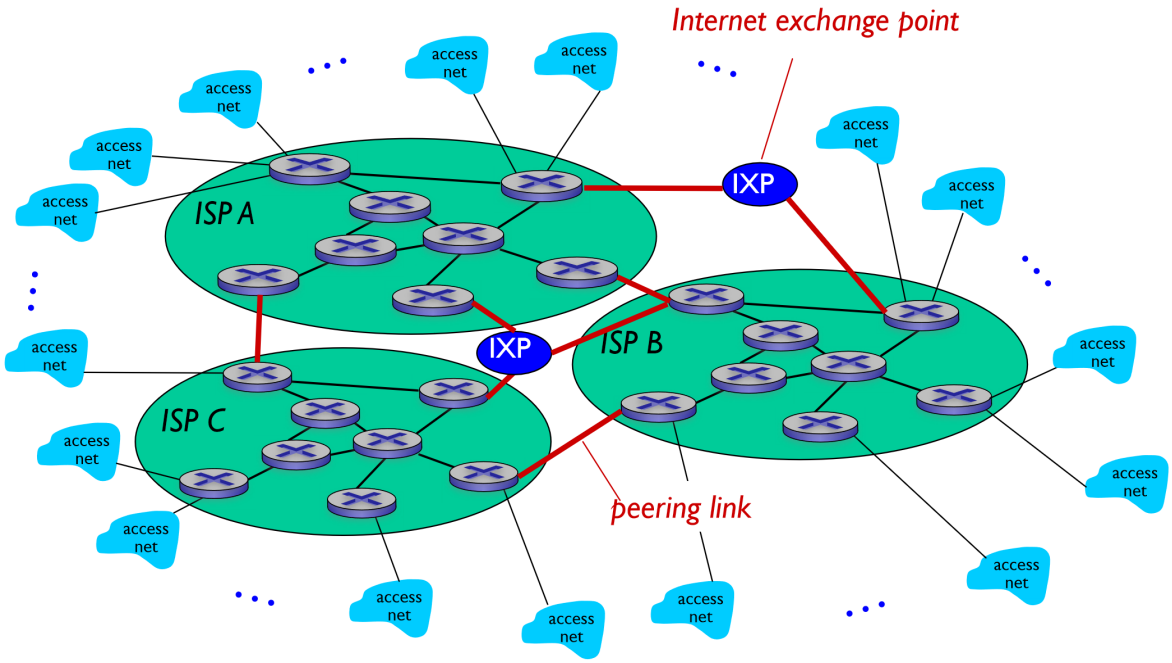
2. connect each access ISP to one global transit ISP

Customer and provider ISPs have economic agreement.

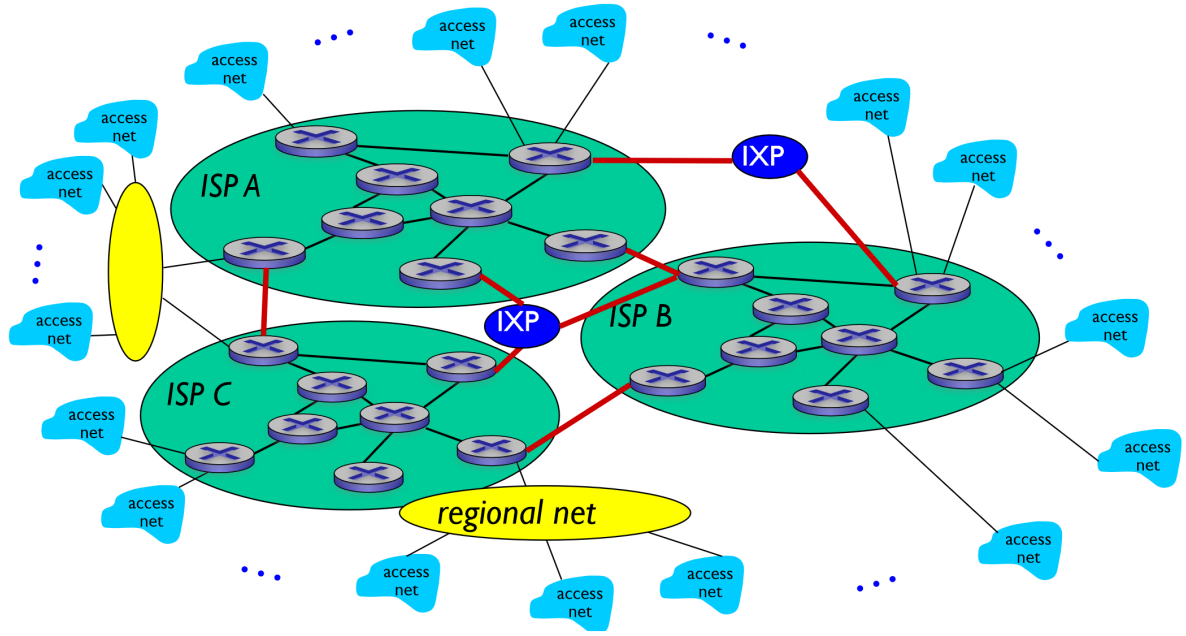


就像美国的T1 ISP Sprint

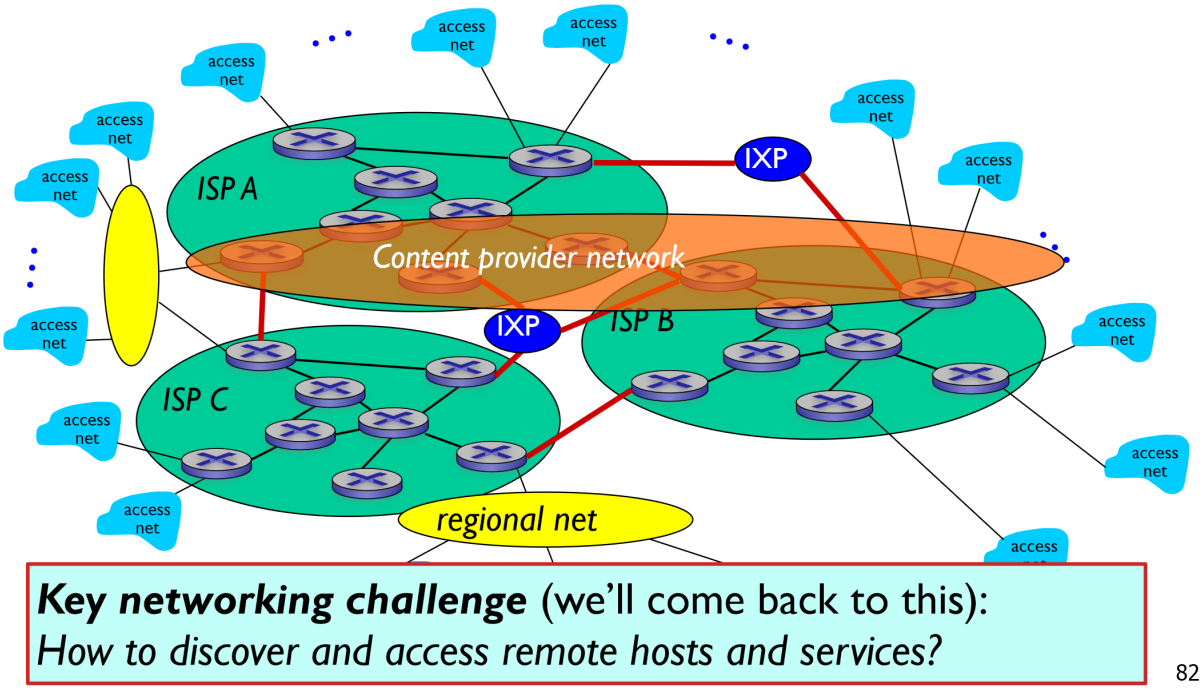
- But one global ISP will have competitors….



- And for the big country, global ISPs are not enough, so there should be regional networks to connect access net to ISPs

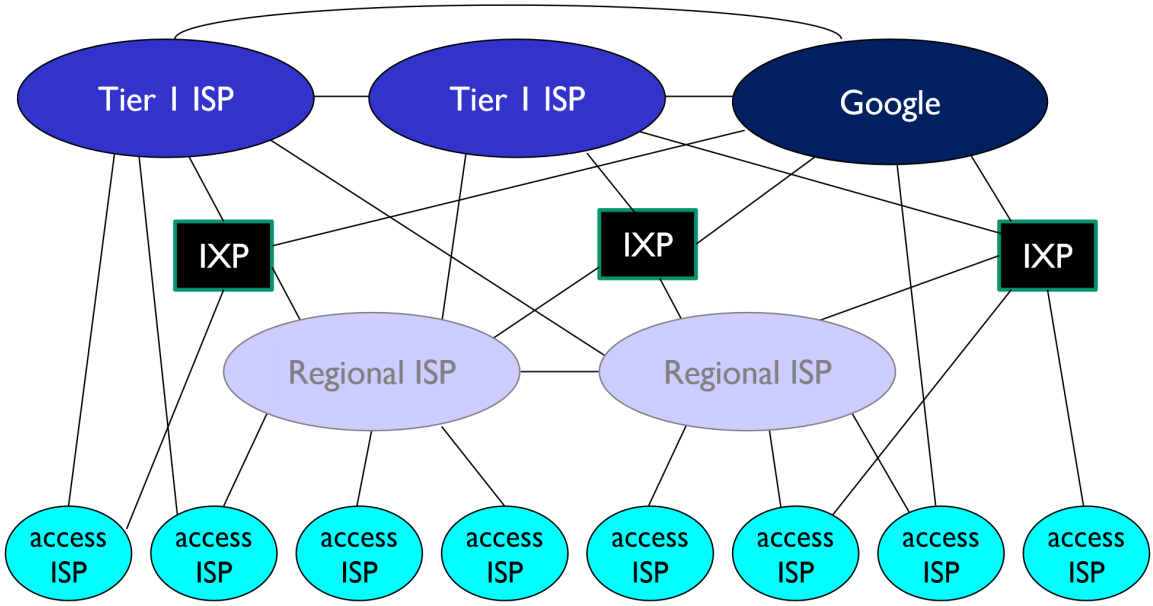


- Also, for some content provider networks (for example Google, Microsoft, Akamai) may run their own network, to bring services, content close to end users



82

- Simplier diagram

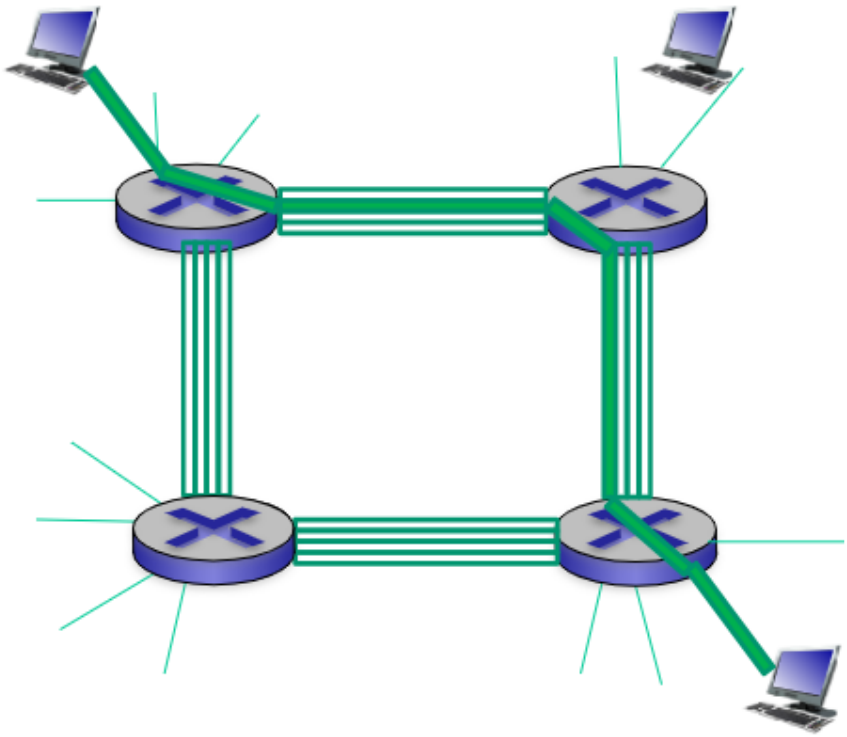


- At the centre: small number of well-connected large networks
  - Tire-I commercial ISPs (e.g. Level 3, Sprint, AT&T, NTT), with national & international coverage
  - content provider network (e.g. Google): private networks that connects its servers to the Internet, often bypassing tier-I, regional ISPs 通常绕过一级及区域级ISP

Circuit Switching

Circuit Switching (Usually used in traditional telephone network)

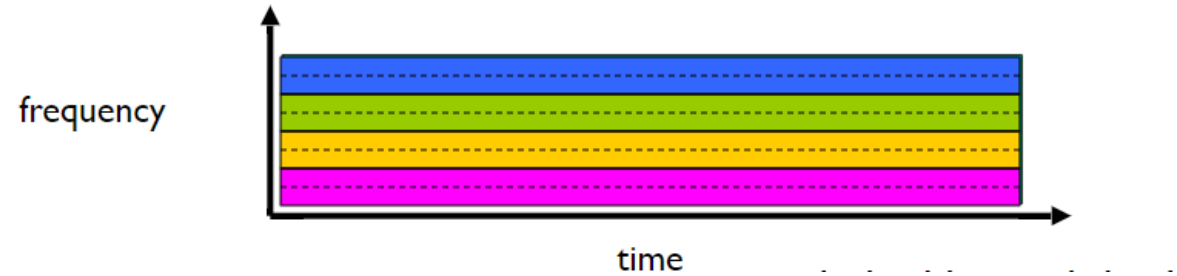
- End to end resources allocated to, reserved for “call” between source & destination
  - In diagram, each link has four circuits
    - call gets 2nd circuit in top link and 1st circuit in right link.
  - It is dedicated resources: no sharing
    - Circuit like guaranteed performance
  - Circuit segment idle while reserved but not used by call 在没有电话的时候空闲，即不和其他人共享线路。



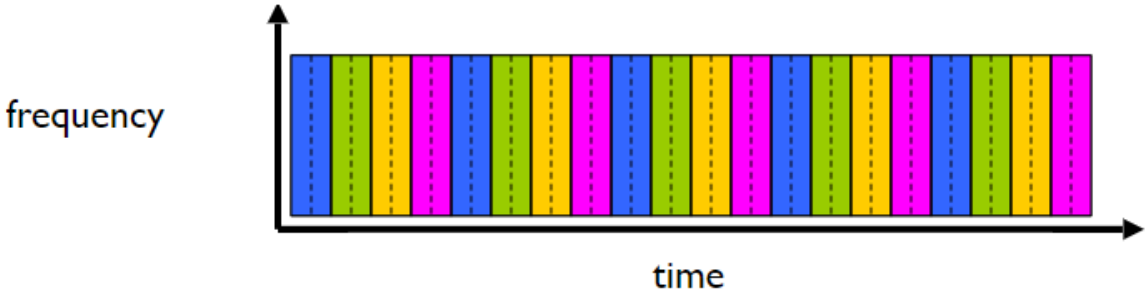
💡 How can we use circuit switching but still support more simultaneous users than we have physical circuits? 在使用线路切换的同时，我们如何实现同时使用的用户人数大于物理线路的个数?

Circuit Switching: FDM versus TDM

- FDM (Frequency Division Multiplexing) 频分复用

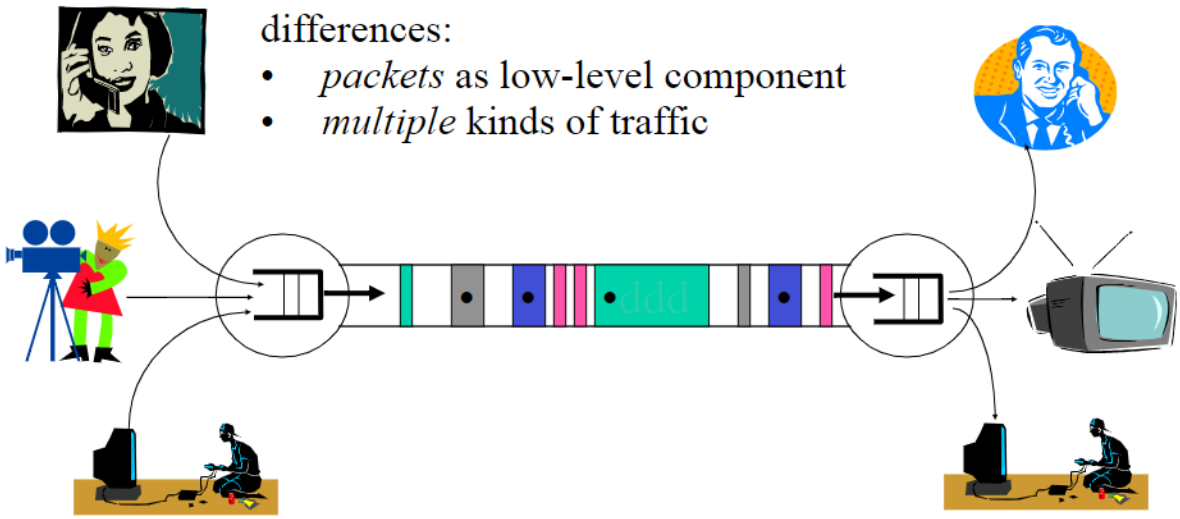


- TDM (Time Division Multiplexing) 时分复用 (时间切片)



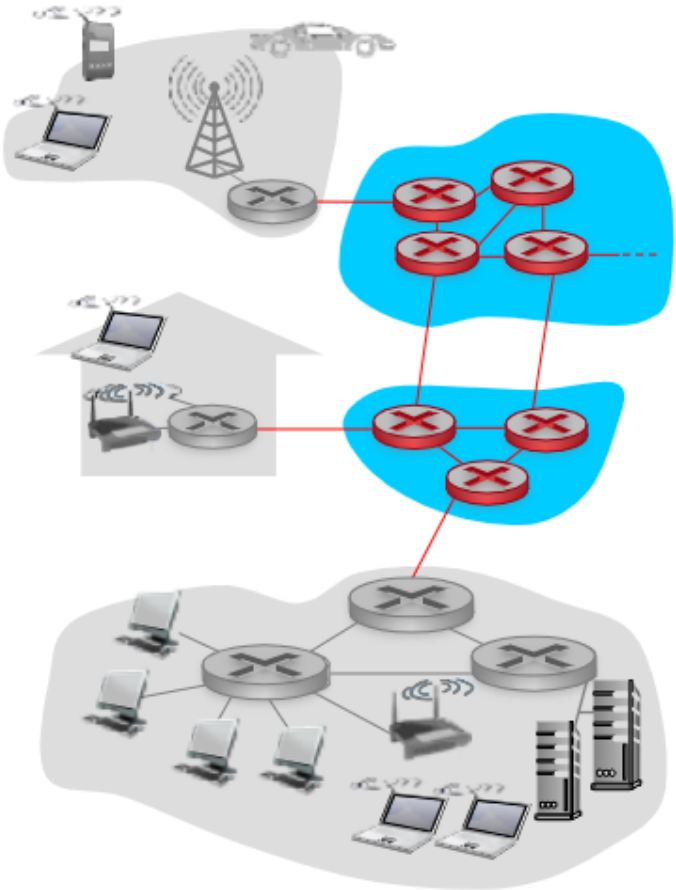
💡 Dashed line: subdivide for upstream and downstream Different colours indicates different users

Packet Switching



The network Core

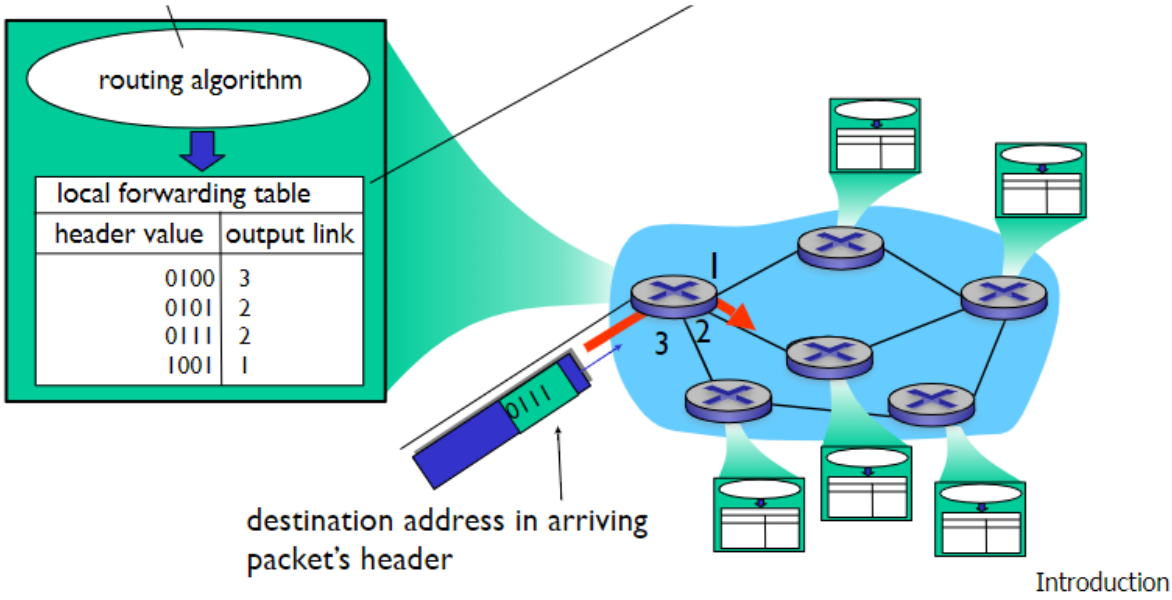
- Mesh of interconnected routers
- Packet-switching: hosts break application layer messages into several packets
  - Forward packets from one router to the next, across links on path from source to destination
  - each packet transmitted at full link capacity
    - after transmission, link is available for other packets (including from other sources)



Two key network-core functions

> Key network challenge:
>
> How to learn consistent, quality routes in dynamic, distributed setting?

- Routing:
  - Determines source-destination route taken by packets
    - routing algorithm
- Forwarding
  - Move packets from router's input to appropriate router output



Packet switching

💡 HW1, HW 1, homework1, homework 1 Question 3

Packet switching versus circuit switching

- Packet switching allows more users to use network

💡 1 Mb/s link Each user use 100 kb/s when "active", and each user active 10% of time How many users can circuit switching support?

- With 100% reliabilty, 10 users can support
- While the company will not only accept 10 users, which is cost-taking

💡 Under Packet Switching With 35 users, what is the probability for active user is greater than 10?

$$\sum_{i=0}^{i=T} \binom{N}{i} p^i (1-p)^{N-i}$$

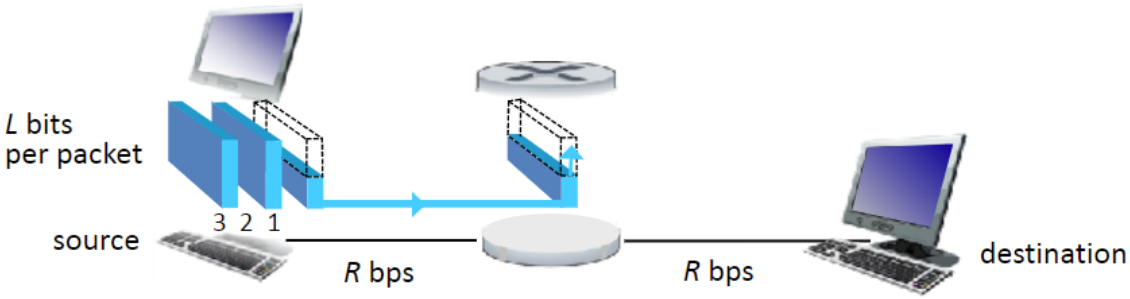
$$= \sum_{i=0}^{i=10} \binom{35}{i} 0.1^i 0.9^{35-i}$$

$$= 0.9^{35} + \binom{35}{1} 0.1 * 0.9^{34} + \binom{35}{2} 0.1^2 0.9^{33} + \ldots + \binom{35}{10} 0.1^{10} 0.9^{25}$$

$$= 0.9995757024045502$$

$$1 - 0.9995757024045502 = 0.0004$$

Packet-switching: store-and-forward

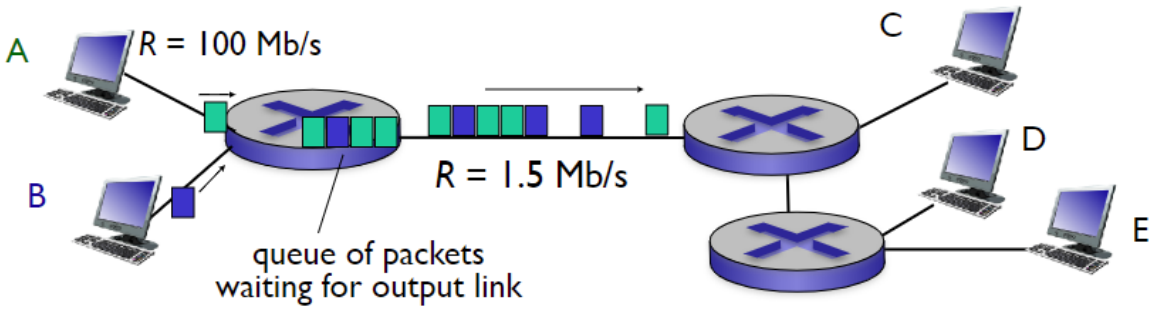


- Takes L/R seconds to transmit L-bit packet into link at R bps
- Store and forward: entire packet must arrive at router before router can transmit it onward
  - buffers the packet until ready to send

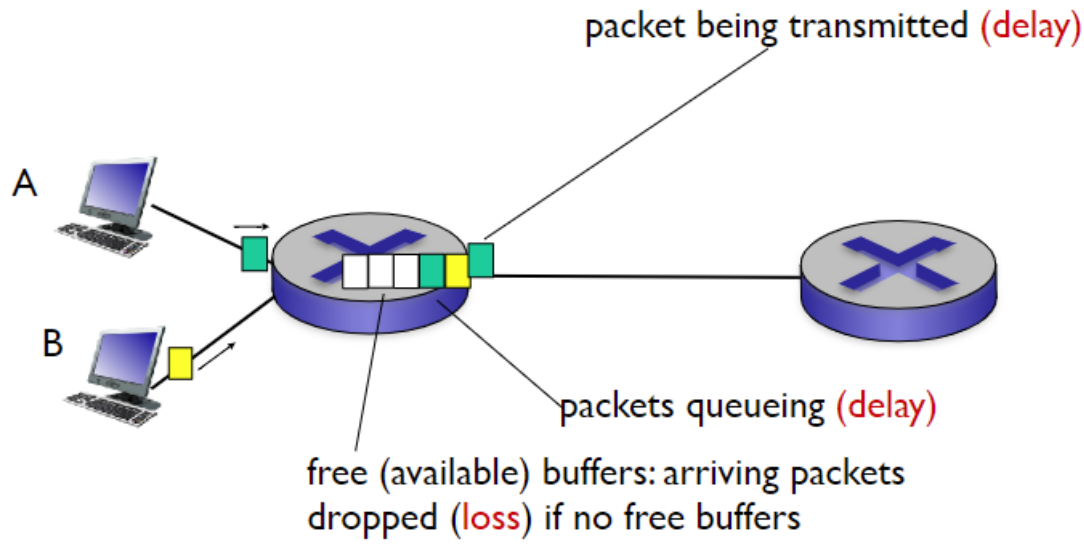
> 整个数据包必须完整达到路由之后，路由才可以继续传输

💡 **End-to-end delay of 1 packet: $2L/R$ Delay for destination to receive all 3 packets:** $4L/R$ ($2L/R$ is for 3 packets from source to router, last L/R is for the entire packet send to the destination)

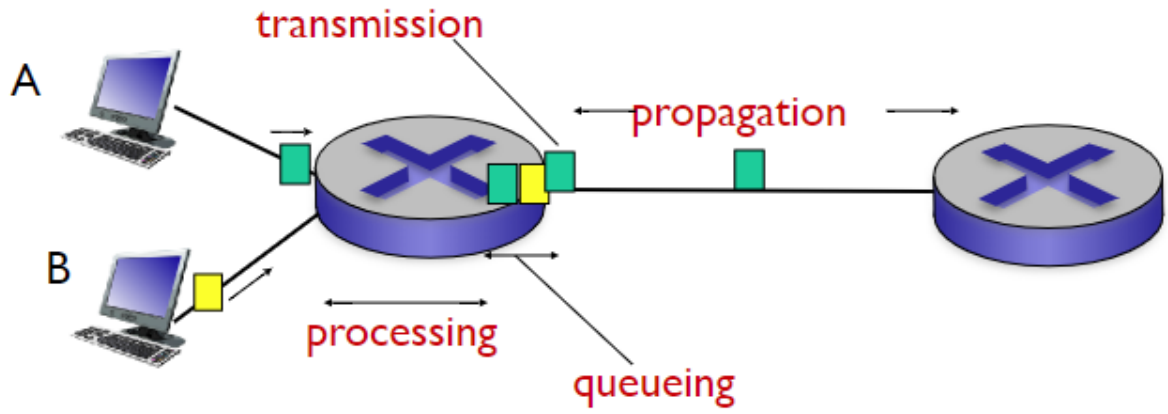
Packet-switching: queuing delay, loss



- If arrival rate (in bit) to link exceeds transmission rate of link for a period of time:
  - **packets will queue in a buffer**
  - **packets can be dropped (lost) if the buffer is full**



4 sources of nodal (per node) delay 四个节点源的延迟原因



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

- d_{proc}: **processing delay**
  - router will check bit error
  - will determine output link
  - the delay typically samller than 1 ms
- d_{queue}: **queuing delay**
  - Time waiting at output link for transmission
  - depends on congestion level of router
- d_{trans}: **transmission delay**
  - $d_{trans} = L/R$
  - $L = packet\ \ length\ \ (bits)$
  - $R = link\ \ bandwidth\ \ (bps)$
- d_{prop}: **propagation delay**
  - $d_{prop} = d/s$
  - $d = length\ \ of\ \ physical\ \ link$
  - $s = propagation\ \ speed\ \ (2 * 10^8 m/sec)$

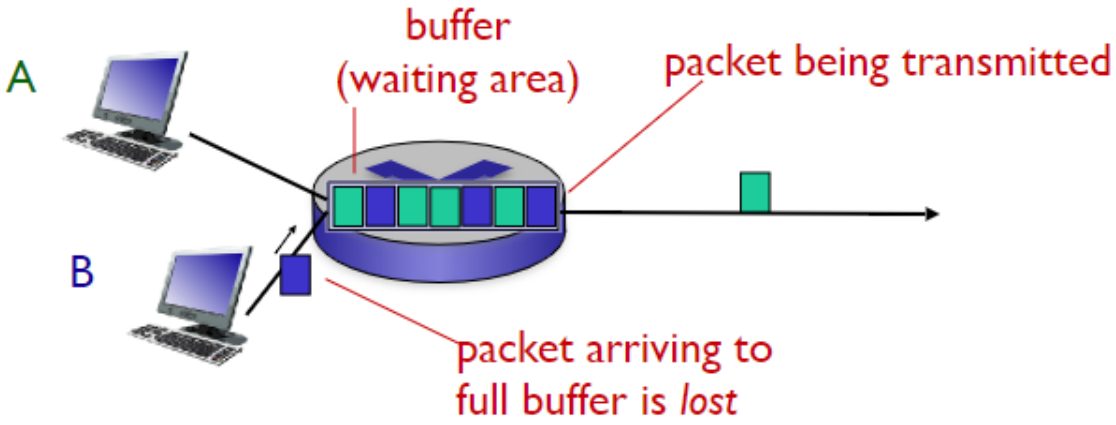
Queueing delay

- **Traffic intensity:**
  - $Traffic\ \ Intensity = La/R$
  - $L = packet\ \ length\ \ (bits)$
  - $R = link\ \ bandwidth\ \ (bps)$
  - $a = average\ \ packet\ \ arrival\ \ rate\ \ (packets\ \ per\ \ second)$

💡 La/R ~ 0: avg. queueing delay small La/R ~ 1: avg. queueing delay large La/R > 1: the average delay infinite, more packets arriving than can be served

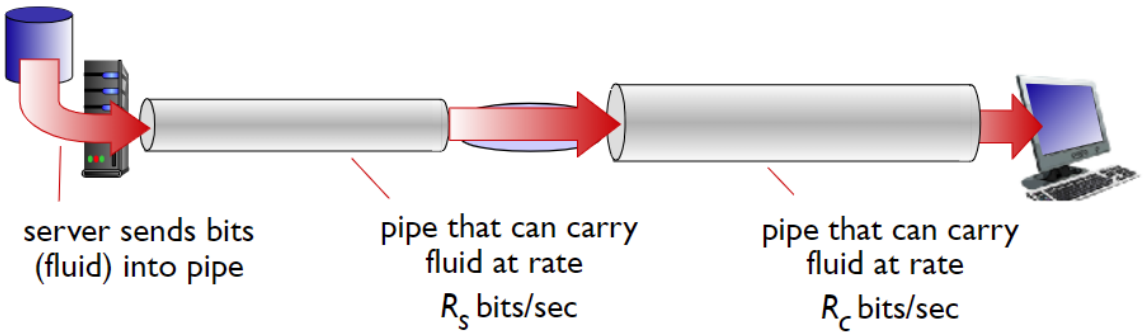
Packet Loss

- queue (buffer) preceding link in buffer has finite capacity
- packet arriving to a full queue will be dropped (lost)
- lost packet may be re-transmitted by previous node, by source end system, but may not

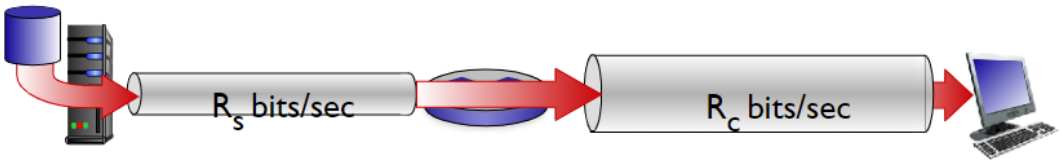


Throughput

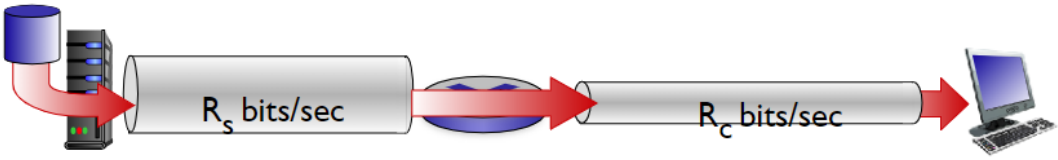
- Throughput rate (bits/time unit) at which bits transferred between sender/receiver
  - instantaneous: rate at given point in time
  - average: rate over longer period of time



- $R_s < R_c$ What is average end-end throughput?



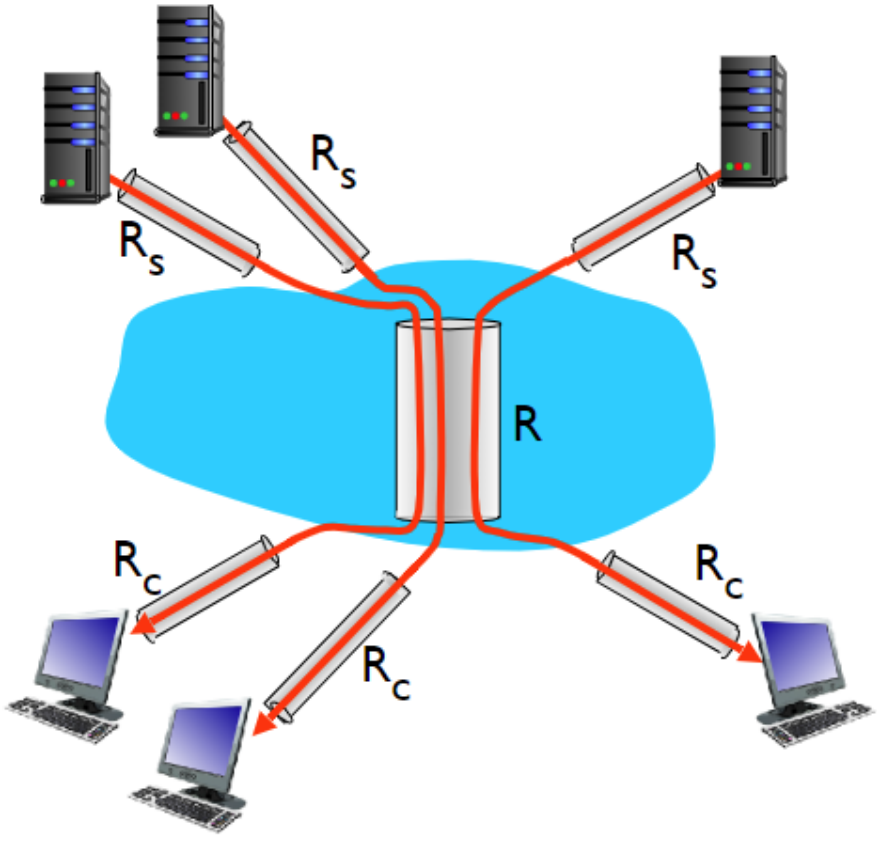
- $R_s > R_c$ What is average end-end throughput?



💡 木桶效应，平均吞吐量取决于这条线上平均吞吐量最小的管道

- per connection end-end thoughput:

$$min(R_c, R_s, R/10)$$



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

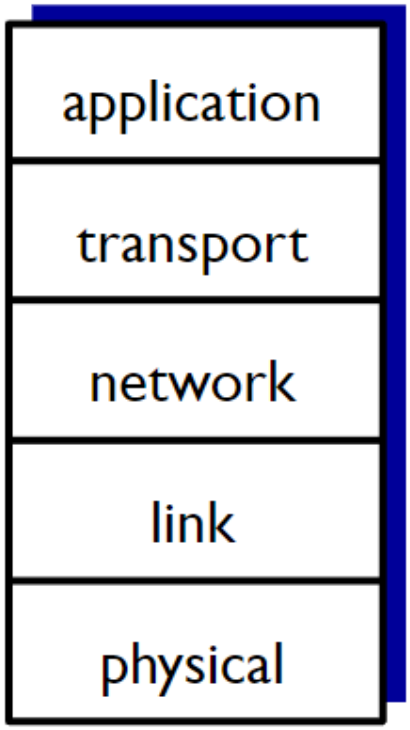
Protocol layers

- Layers: each layer implements a service
  - Via its own internal-layer actions
  - relting on services provided by layer below.

Why layering? (Advantage)

- Dealing with complex systems:
  - explicit structure allows identification, relationship of complex system's pieces
  - modulatization eases maintenance, updating, reuse of system.
    - change of implementation of lay's service transparent to rest of system
    - e.g. change in gate procedure doesn't affect rest of system

Internet protocol stack (5 layers)

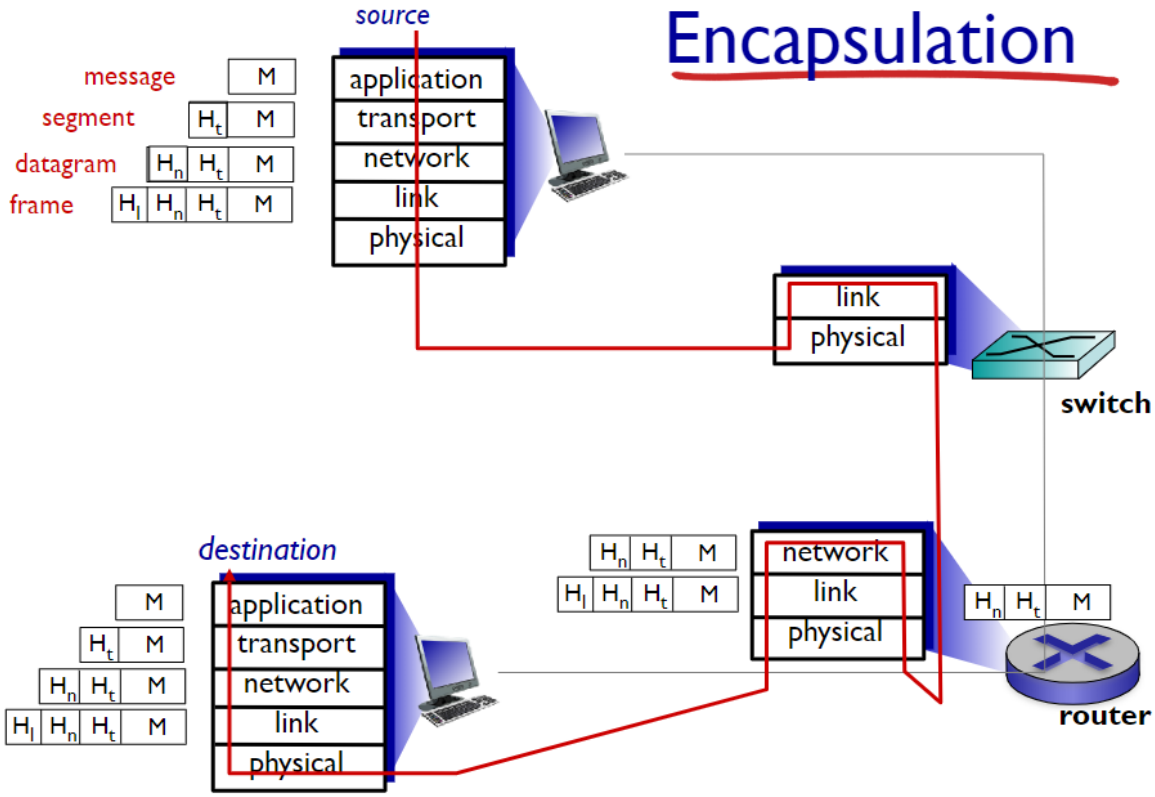


- Application layer:
  - Supporting network applications
    - FTP, SMTP, HTTP…
- Transport layer:
  - process-process data transfer
    - TCP, UDP
- Network layer:
  - Routing of datagrams from source to destination
    - IP, routing portocol
- Link layer:
  - Data transfer between neighboring network elements
    - Ethernet, 802.111 (Wi-Fi), PPP
- Physical layer:
  - bits "on the wire"

ISO/OSI reference model (7 layers)

- Presentation layer:
  - Allow applications to interpret meaning of data
    - Like encryption, compression. Machine specific conventions
- Session layer:
  - Synchronization, checkpointing, recovery of data exchange

Encapsulation - How does one packet travel in the Internet



Why NOT layering? (Disadvantage)

- Dealing with complex systems:
  - Layering considered harmful
    - upper layer may duplicate functionality of lower
    - upper may not need functionality of lower
    - one layer might want info only visible at another layer

Networks under attack: security

Network security

- Field of network security
  - How bad guys can attack computer networks
  - How we can defend networks against attacks
  - How to design architecture that are immune to attacks
- Internet not originally designed with much security in mind

Malware

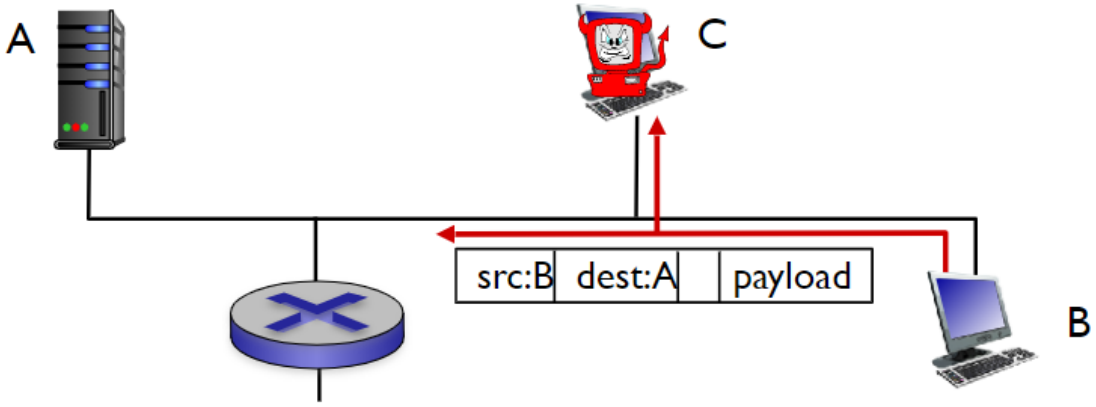
- Malware can get in host from
  - virus: self-replicating infection by receiving/executing objects
  - worm: self-replicating infection by passively receiving object that get itself executed

Attack server, network infrastructure

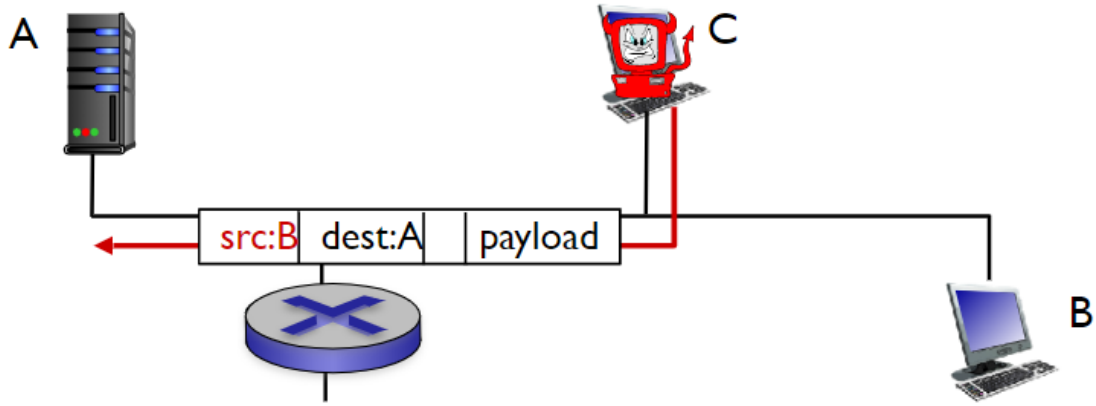
Denial of Service (DoS): attackers make resources (server, bandwidth) unavailable to legitimate traffic by overwhelming resource with bogus traffic.

Packet "sniffing" 数据包嗅探:

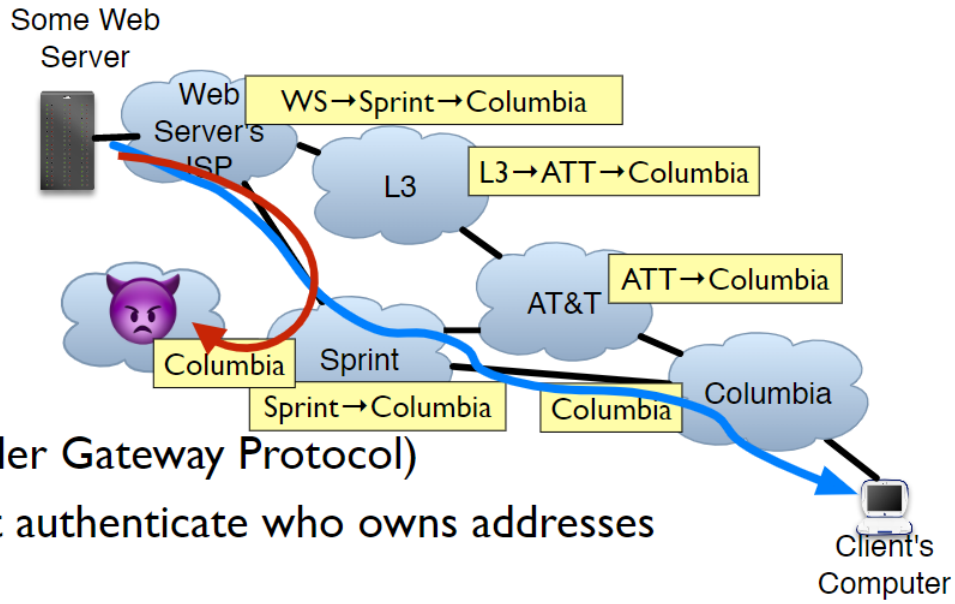
- broadcast media (shared Ethernet, wireless)
- promiscuous network interface reads/records all packets passing by



IP spoofing IP地址嗅探: send packet with false source address



Claim control address:

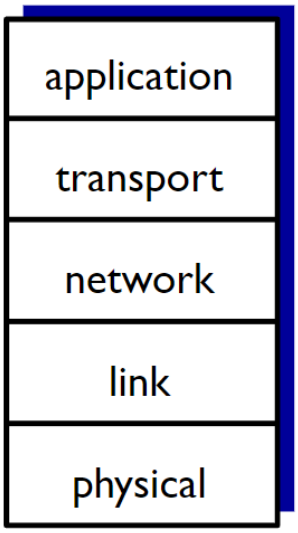


BGP (Border Gateway Protocol)

- does not authenticate who owns addresses

Chapter 2 - Application Layer

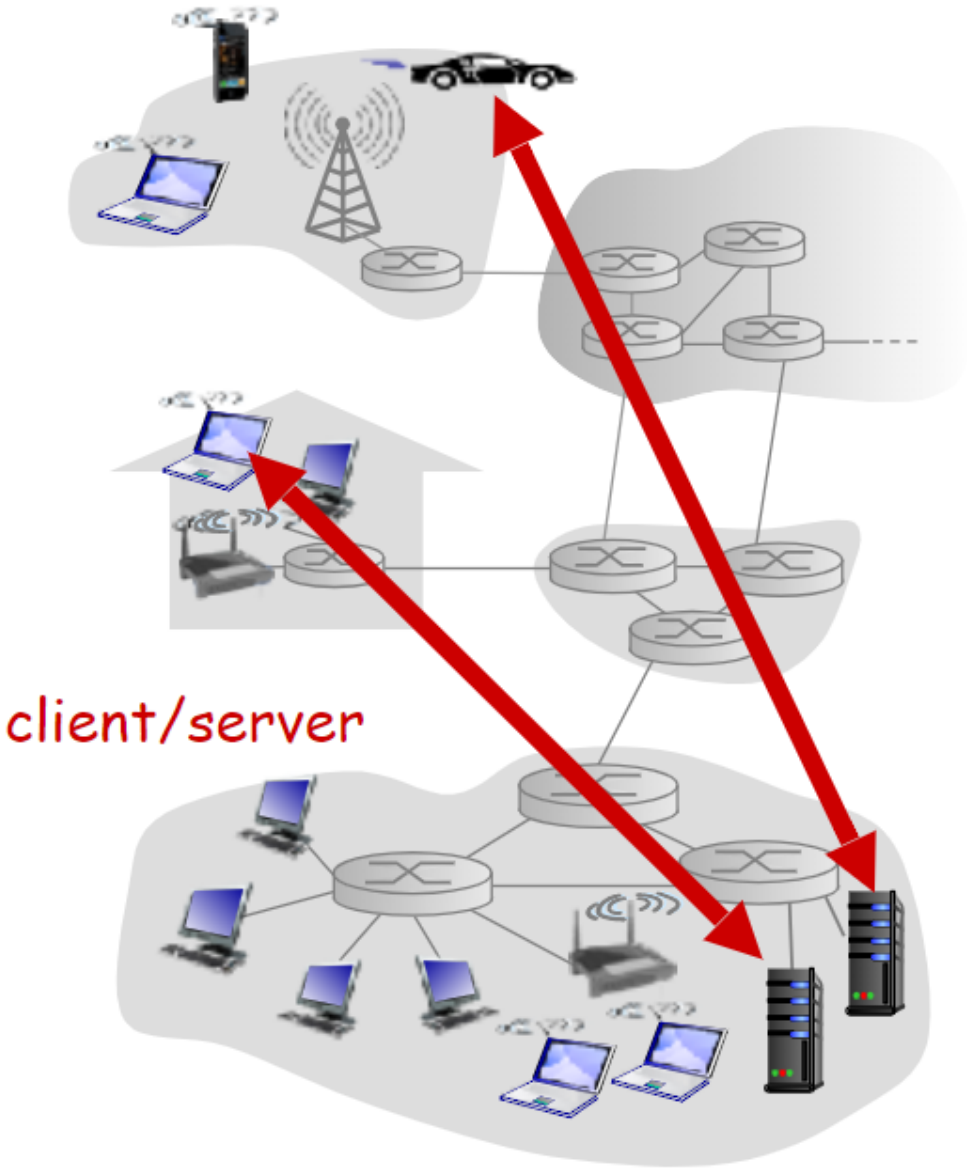
- *application:* supporting network applications
  - FTP, SMTP, HTTP *This chapter*
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet,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



Principles of network application

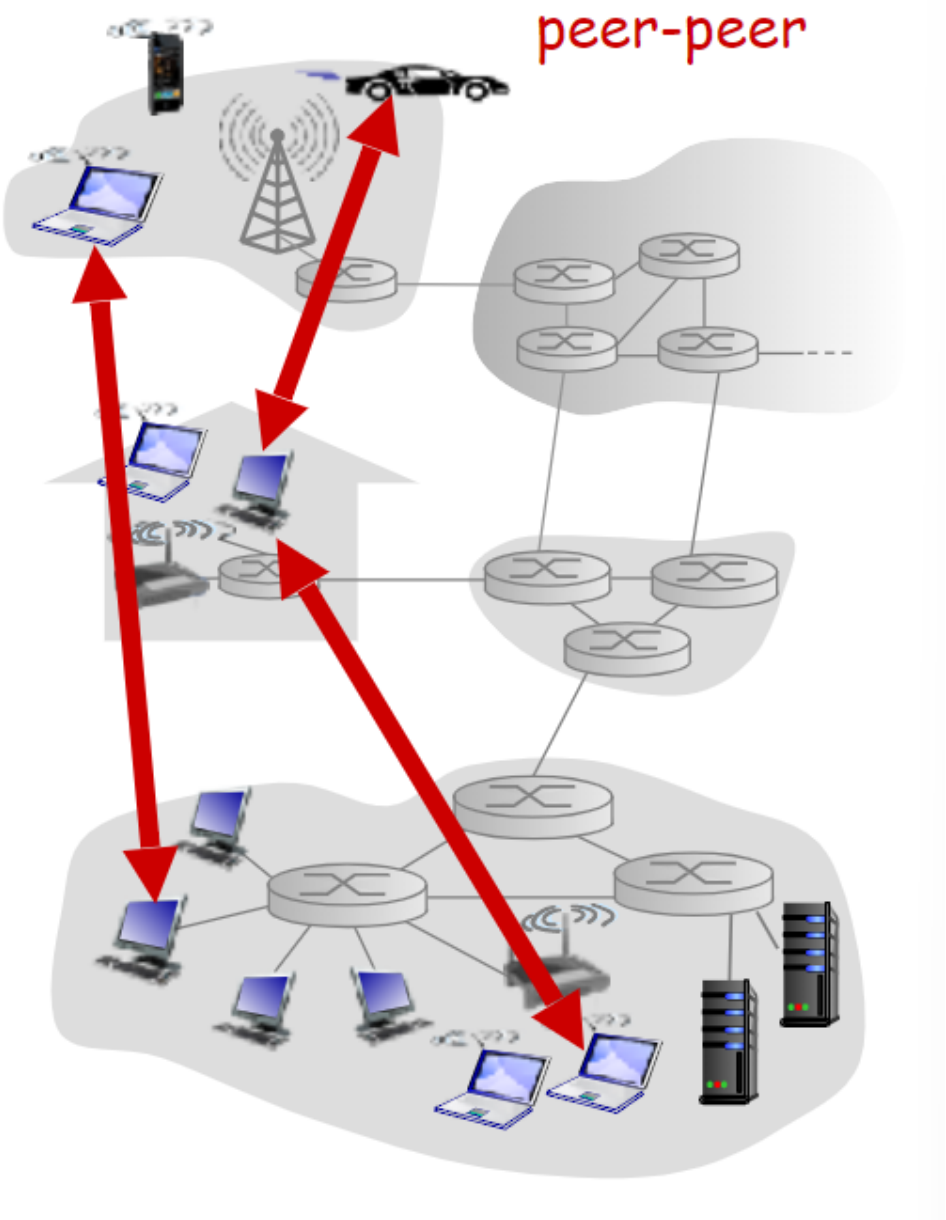
- **Possible structure of applications**
  - Client-server
  - Peer-to-peer (P2P)

Client-server architecture



- Server:
  - always-on host
  - permanent address
  - data centers for scaling
- Clients:
  - Communicate with server
  - May be intermittently connected 间歇性连接
  - May have dynamic IP address
  - do not communicate directly with each other

P2P Architecture



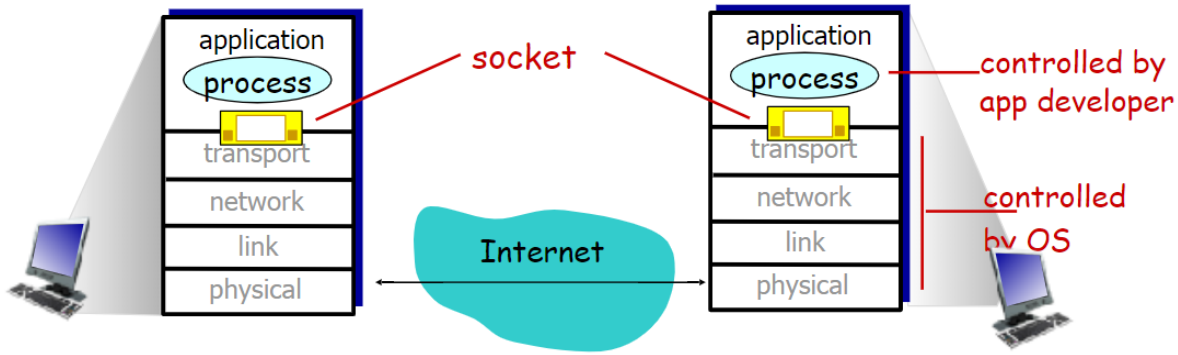
- No always-on server
- arbitrary end systems directly communcate
- peers request service from other peers, provide service in return to other peers
  - **self scalability - new peers bring new service capacity, as well as new service demands**
- peers are intermittently connected and change IP address
  - complex management

Process communicating 进程交流

- Within same host, two processes communicate using **inter-process communication**.
- Processes in different hosts communicate by exchanging **messages**
- **Client process**: process that initiates communication
- **Server process**: process that waits to be contacted

Sockets

- Process sends/ receives messages to/from its socket



Addressing processes

- To receive messages, process must have identifier
- Host device has unique 32-bit IP address (IPv4)
- Note that the IP address is not sufficient to identify the process. because there may have several processes on the same host.
- Identifier includes both IP address and port numbers associated with process on host
  - HTTP server: 80 in default
  - Mail server: 25 in default
- Example, to send HTTP message to gaia.cs.umass.edu:
  - 128.119.245.12:80

Application layer protocol defines

- Types of message exchanged
  - Request, response
- Message syntax
  - What fields in messages & how fields are delineated
- Message semantics
  - Meaning of information in fields
- Rules
  - For when and how processes send & respond to messages
- Open protocols
  - Defined in RFCs
  - Allows for interoperability
  - HTTP, SMTP
- Proprietary protocols
  - Skype

Apps need for transport service

- Data integrity 数据完整性
  - Some app require 100% reliable data transfer
    - Downloading, file transferring
  - Other apps do not require (VOIP)
- Timing 延迟

    - Some app require low delay
        - VOIP
- Throughput 吞吐量
    - Require minimum amount of throughput to be effective
        - Media
- Security

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

Internet transport protocols services

TCP service:

- Provides:
    - Reliable transport between sending and receiving process
    - Flow control: Sender will not overwhelm receiver 传输控制，控制速度以匹配发送者和接收者
    - Congestion control: throttle sender when network overloaded 堵塞控制，当有其他大流量时限制速度
- Do not provide:
    - Timing
    - Minimum throughput guarantee
    - Security

> 适合大流量，文件传输

UDP service:

- Provides:
    - Unreliable data transfer (?)
- Do not provide:
    - reliable
    - flow control
    - congestion control
    - Throughput guarntee
    - security
    - connection setup

> 适合VOIP

Internet apps: application, transport protocols

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

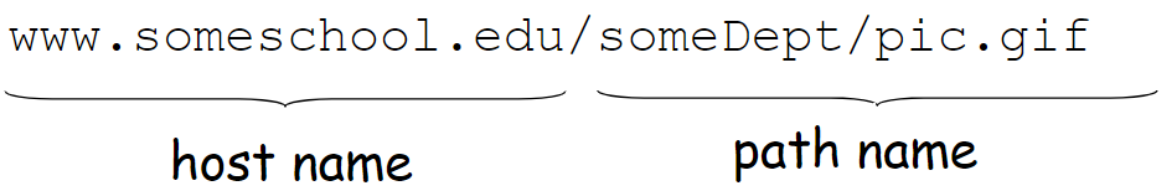
Securing TCP

- TCP and UDP has no encryption
- SSL/TLS provides encrypted TCP connection, data integrity, end-point authentication
- SSL is older, TLS is at application layer
  - App use SSL libraries talk to TCP
  - Client or server negotiate use of TLS

Web and HTTP

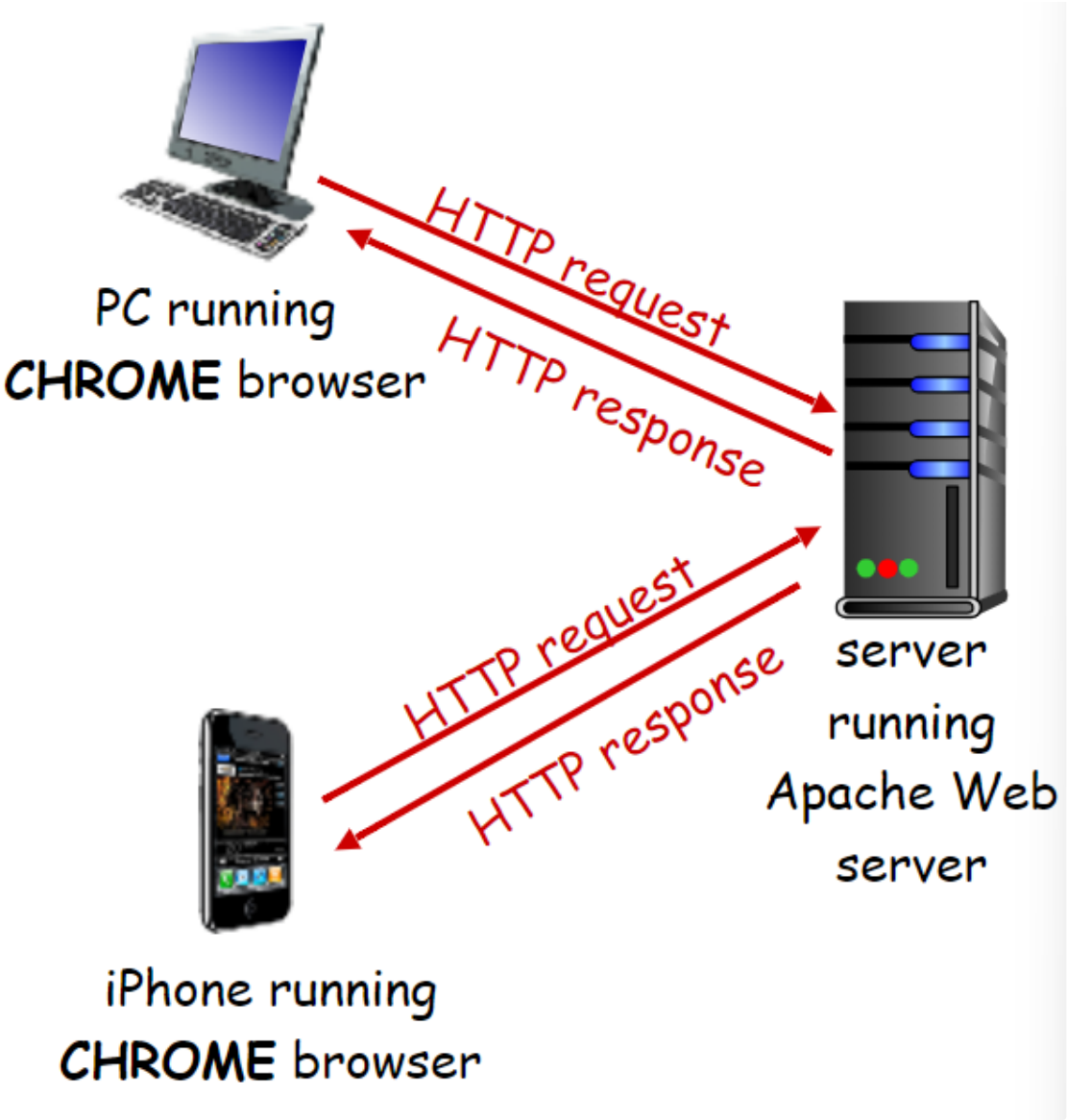
Web Overview

- Web page consists of objects
- object can be HTML file, JPEG file, Java applet, audio file….
- web page need to consists a base HTML-file which includes several referenced objects
- each object is addressable by a URL



HTTP Overview

- HTTP: Hypertext Transfer Protocol超文本传输协议
- Client/server model
  - Client: broswer that requests, receives, (using HTTP protocol) and “displays” Web objects
  - Server: Web server sends (using HTTP protocol) objects in response to requests

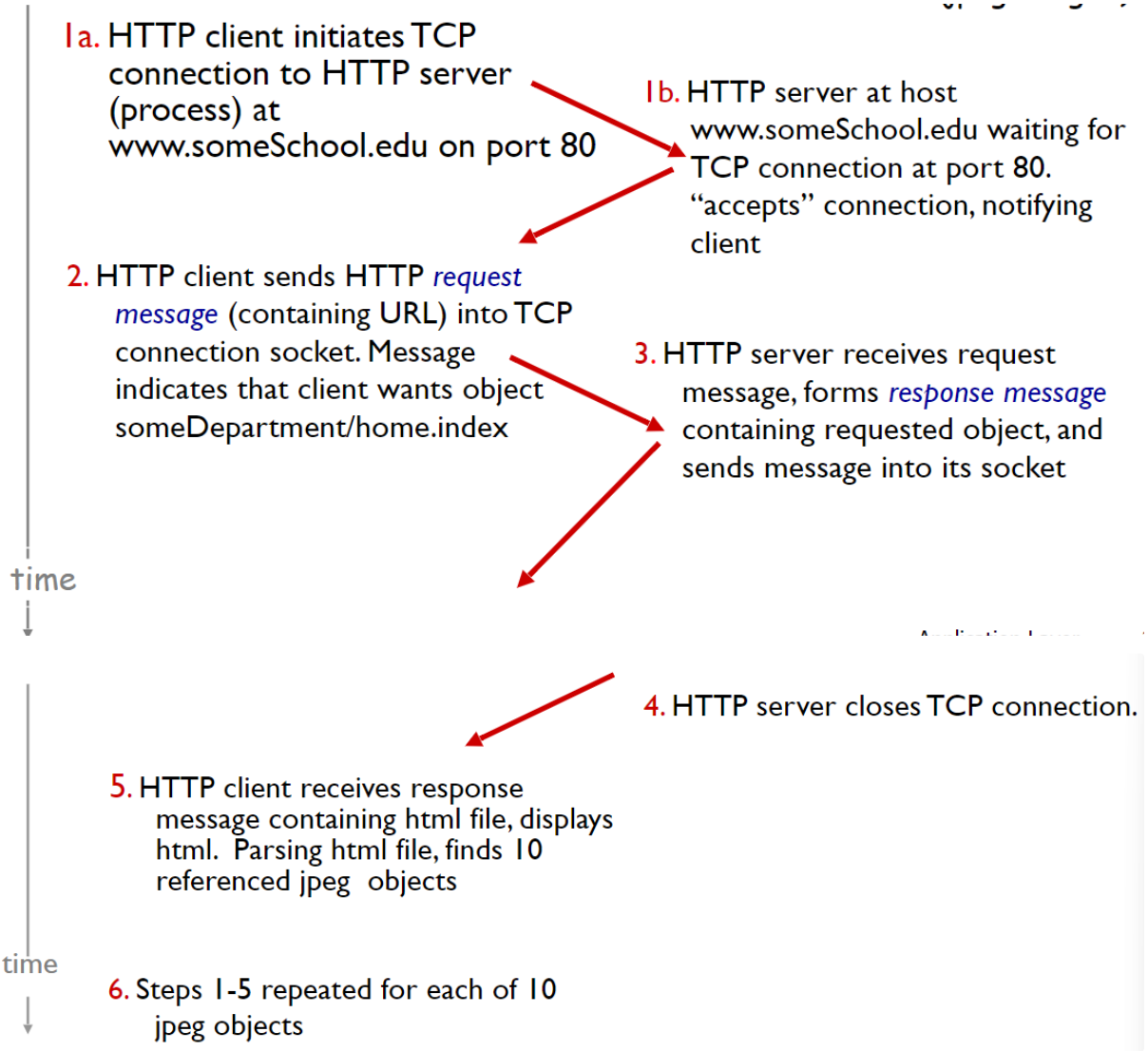


- Using TCP:
  1. Client initiates TCP connection, usually port 80
  2. server accepts TCP connection from client
  3. HTTP messages (application-layer protocol messages exchanged between broswer (HTTP client) and Web server (HTTP server)
  4. TCP connection closed
- HTTP is stateless
  - From perspective of protocol, server maintains no information about past client requests

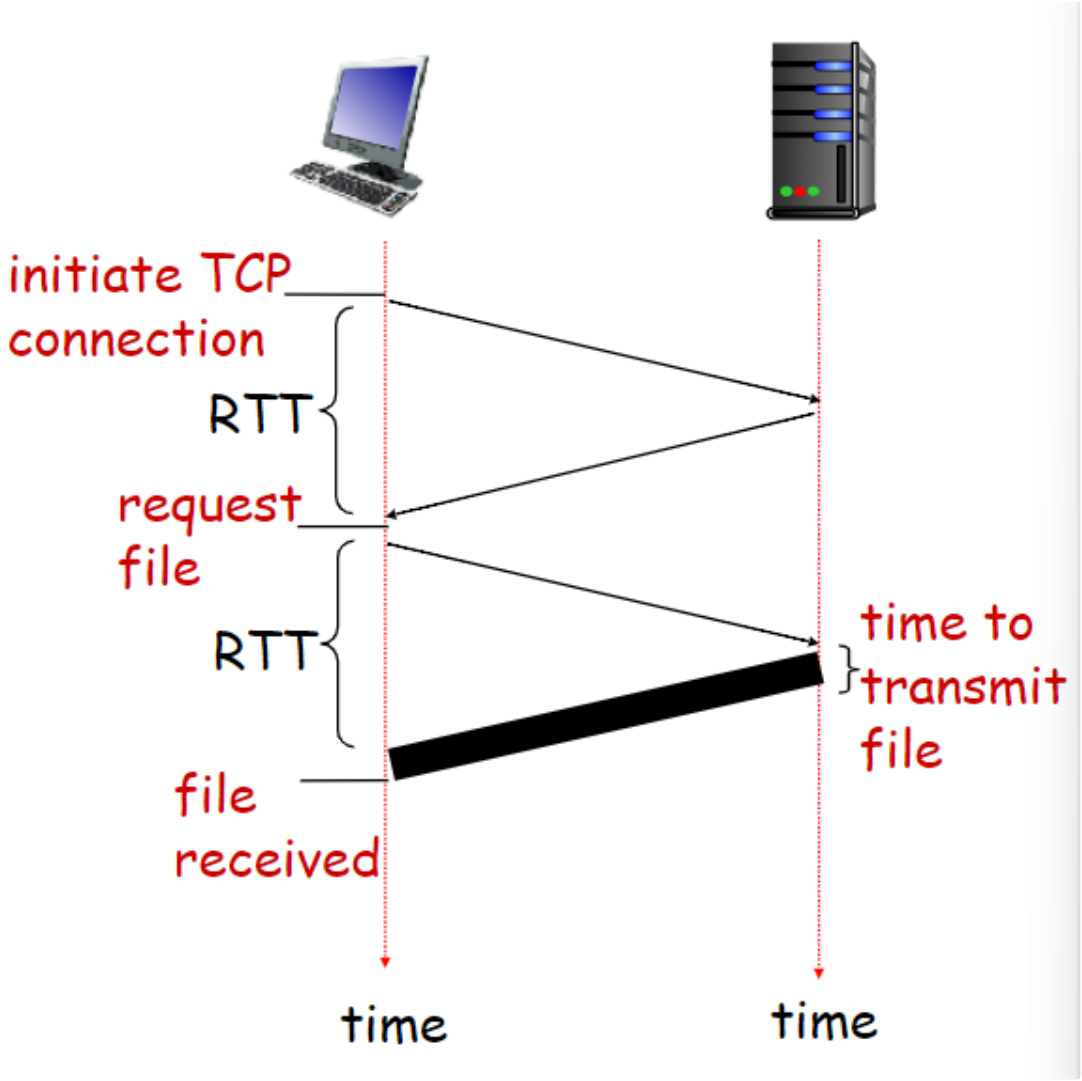
HTTP connections

- Non-persistent HTTP 非持久性HTTP
  - at most one object sent over TCP connection
  - connection then closed
  - downloading multiple objects required multiple connections
- Persistent HTTP 持久性HTTP
  - multiple objects can be sent over single TCP connection between client and server

Non-persistent HTTP

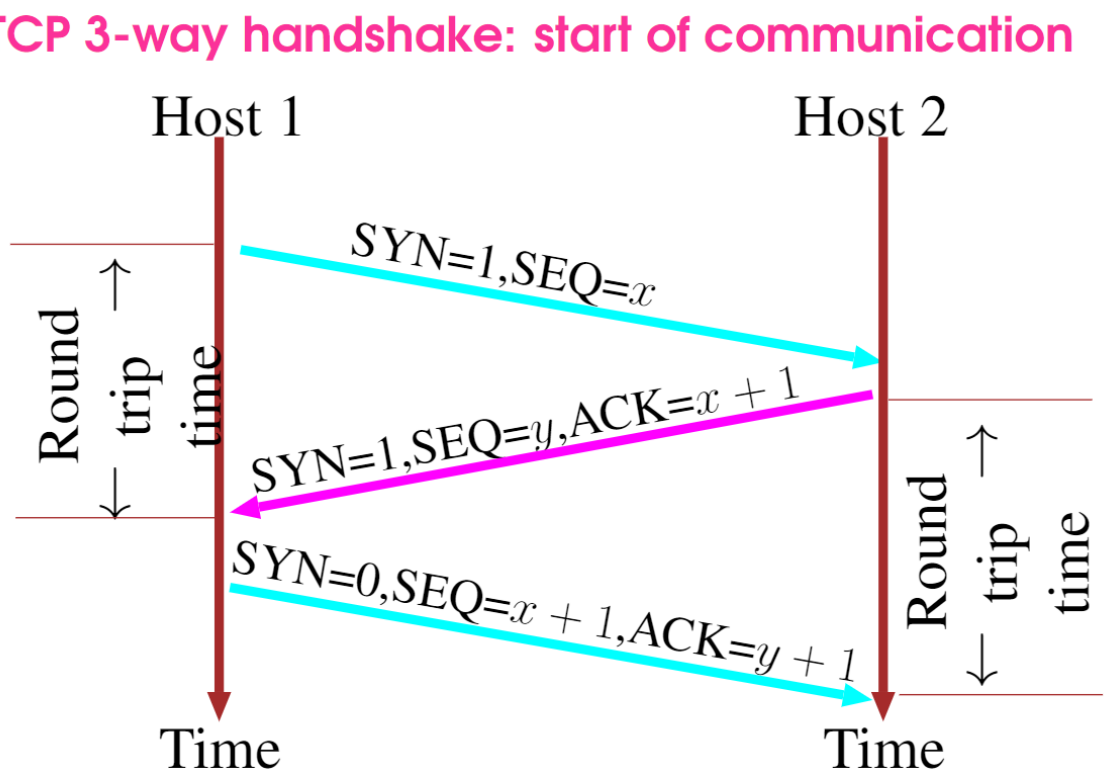


Non-persistent HTTP: response time



- Round Trip Time (RTT) definition:
  - Time for a small packet to travel client to server and back
- From ELEN30024 Data Networking:

TCP Handshake



  - SEQ means the sequence number
  - ACK means acknowledge signal
  - Round trip time = RTT
  1. Host 1 send SYN=1 to determine that Host 1is available and ask to Host 2 whether it is available, and send the SEQ=x
  2. Host 2 send positive signal with SYN=1, with it's SEQ=y. ACK=x+1 means that Host 2 got Host 1's segment x. And, Host 2 expect Host 1 send ACK=y+1 to acknowledge
  3. Host 1 send SEQ=x+1 and ACK=y+1 to determine that Host 1 got Host 2's segment y.
- HTTP response time:
  - one RTT to initiate TCP connection
  - one RTT for HTTP request and first few bytes of HTTP response to return
  - file transmission time
- Non-persistent HTTP response time = 2RTT+file transmission time

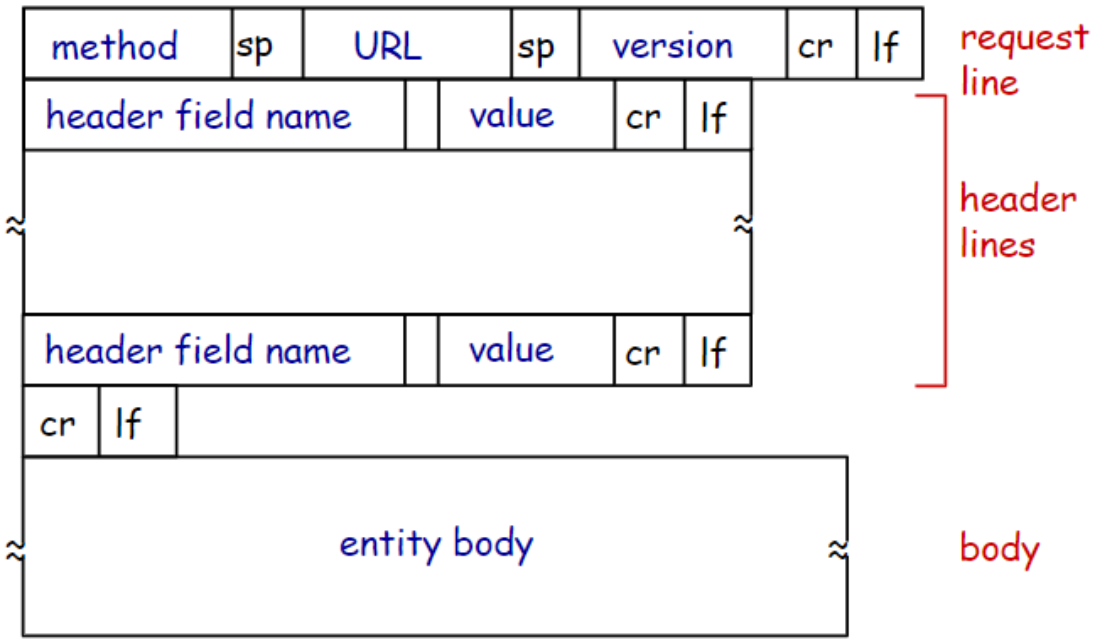
Persistent HTTP and Non-persistent HTTP

- Non-persistent HTTP issues:
  - requires 2RTT per object
  - OS overhead for each TCP connection 每个TCP链接都会消耗OS的资源
  - broswer often open parallel TCP connection to fetch referenced objects
- Persistent HTTP
  - server leaves connection open after sending response
  - subsequent HTTP messages between same client/server sent over connection
  - client sends requests as soon as it encounters a referenced object
  - as little as one RTT for all the reference objects

HTTP request message

- Two types of HTTP message
  - request
  - response
- Request message:
  - ASCII (human-readable format)

HTTP request message: general format



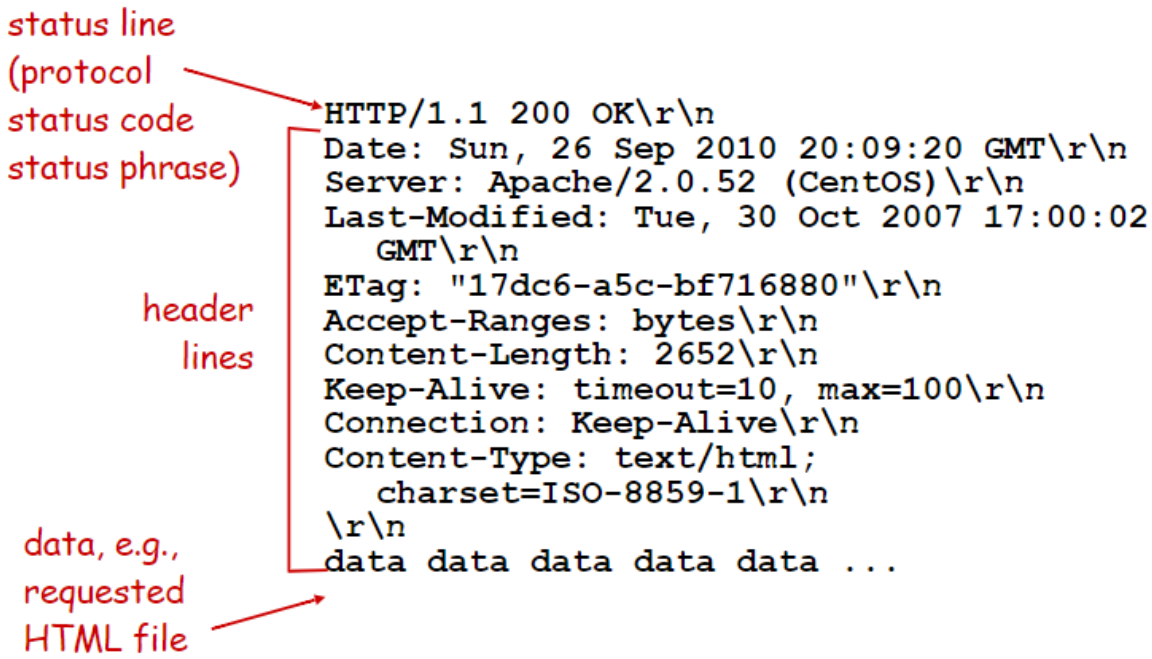
HTTP request message: uploading form input

- POST method
  - web page often includes input
  - input is uploaded to server in entity body
- URL method
  - uses GET method
  - input is uploaded in URL field

Method types

- HTTP 1.0
  - GET
  - POST
  - HEAD
    - Ask server to leave requested object out of response
- HTTP 1.1
  - GET, POST, HEAD
  - PUT
    - upload file in entity body to path specified in URL field
  - DELETE
    - delete file specified in the URL field

HTTP response message



HTTP response status code

200 → OK

request succeeded, requested object later in this msg

301 → Moved permanently

requested object moved, new location specified later in this message (Location:)

400 → Bad request

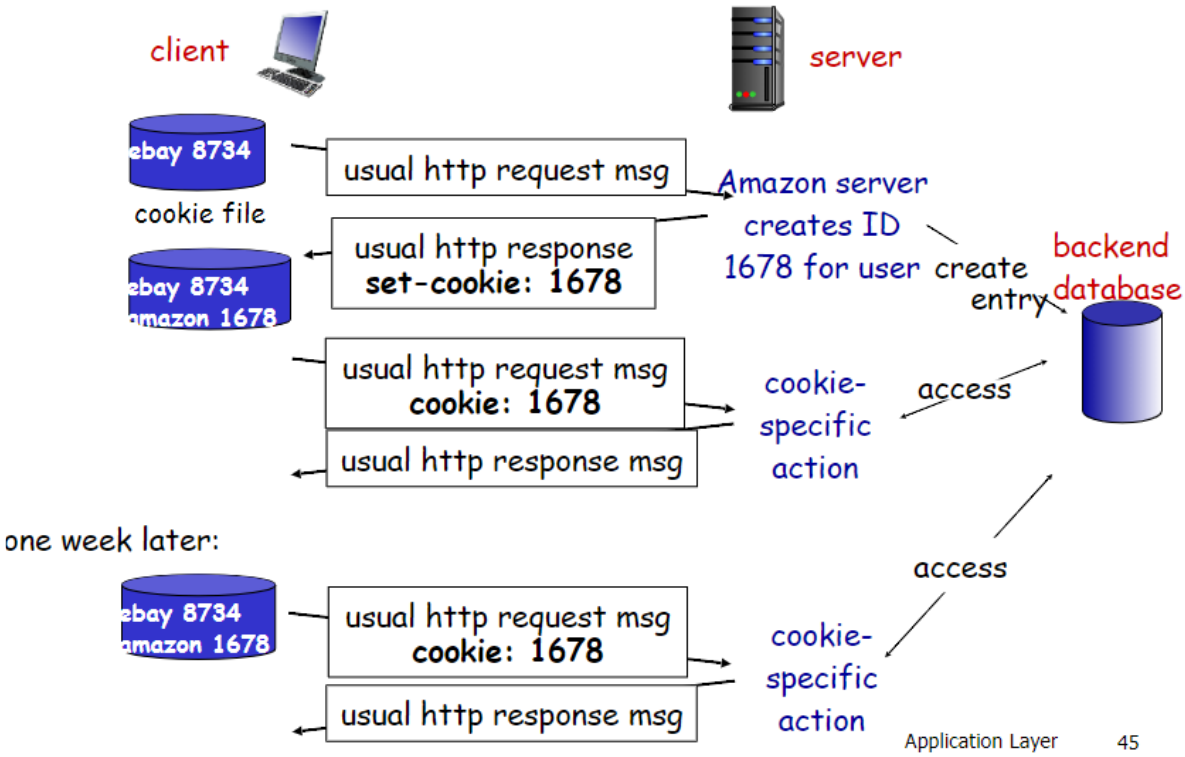
Requested msg not understood by server

404 → Not found

requested document not found on this server

505 → HTTP version not supported

Cookies: keeping state



User-server state: cookies

Many Web sites use cookies:

Four components:

1. Cookie header line of HTTP response message
2. cookie header line in next HTTP request message
3. cookie file kept on user's host, managed by browser
4. back-end database at web site

What cookies can be used for:

- Authorization
- Shopping carts
- Recommendations
- User session state

How to keep state:

- Protocol endpoints: maintain state at sender/receiver over multiple transcation
- cookies: http messages carry state

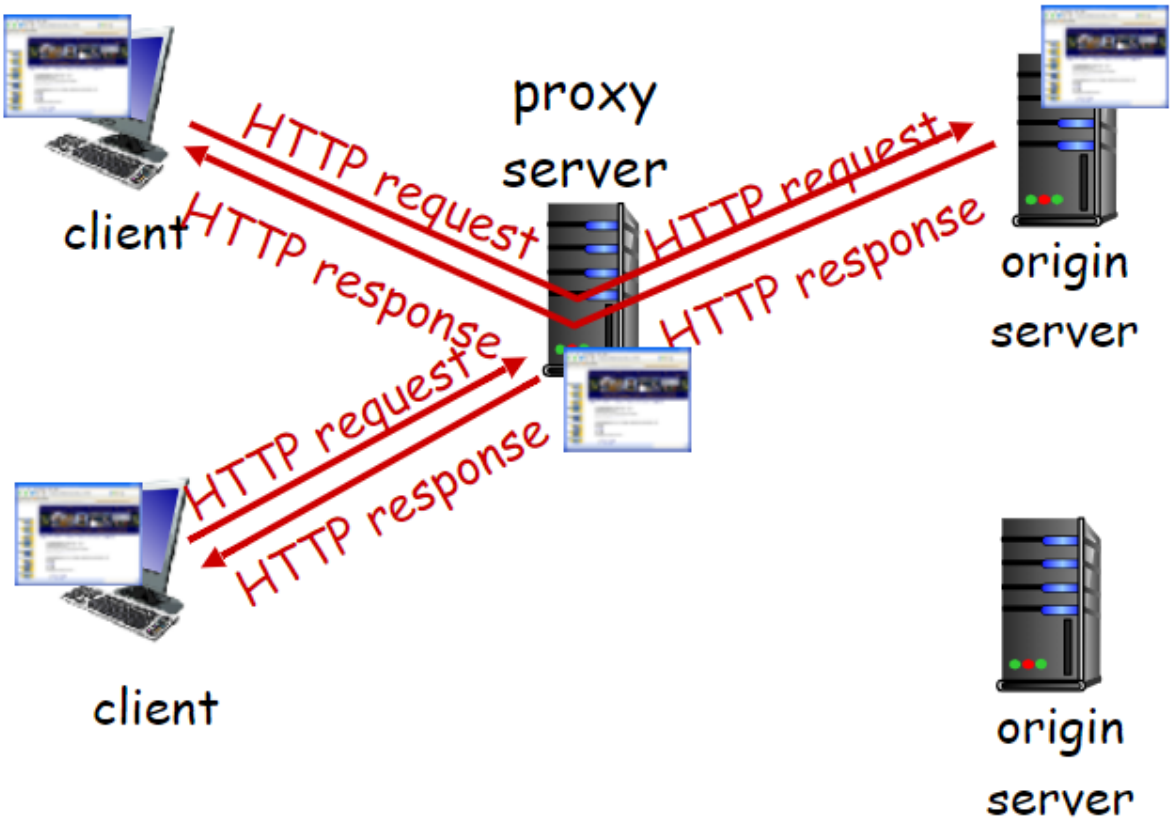
Cookies and privacy

- Cookies permit sites to learn a lot about you
- you may supply name and e-mail address to sites

Web caches (proxy server)

💡 Goal: satisfy request without involving origin server

- User set browser: Web accesses via cache
- Browser sends all HTTP requests to cache
  - If object in cache: cache returns object
  - Else, cache requests object from origin server, then returns object to client



- Cache acts as both clients and server:
  - server for original requesting client
  - client for the original server
- Typically, cache is installed by ISP
  - also used by content delivery networks (CDN)
- **Why web caching?**
  - Reduce response time for client request
  - reduce traffic on an institution's access link
  - Internet dense with caches: enables “poor” content provider to effectively deliver content

Conditional GET

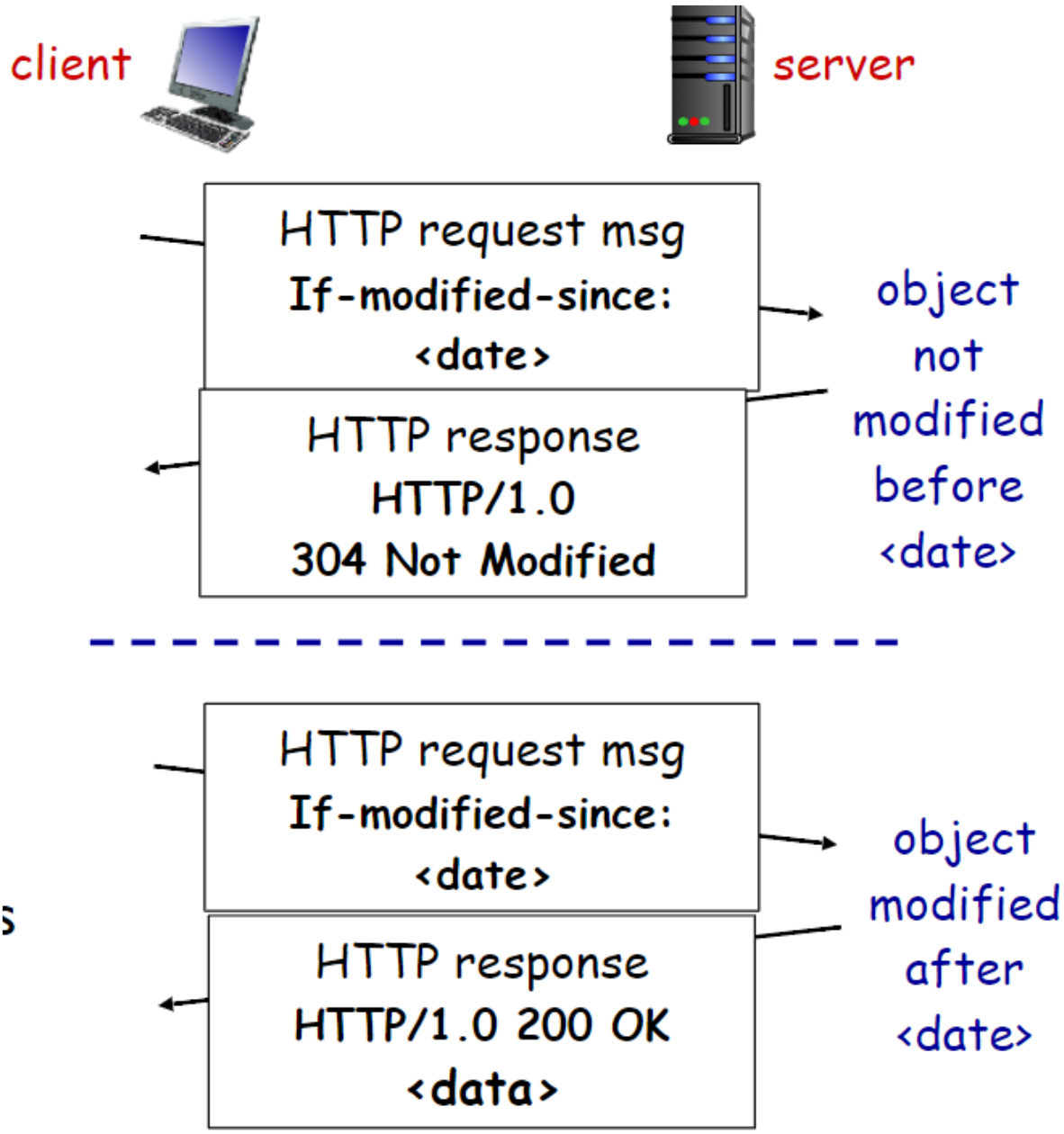
> Goal: do not send object if cache has up-to-date cached version
>
> - no object transmission delay
> - lower link utilisation

- Cache: specify date of cached copy in HTTP request

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

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

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



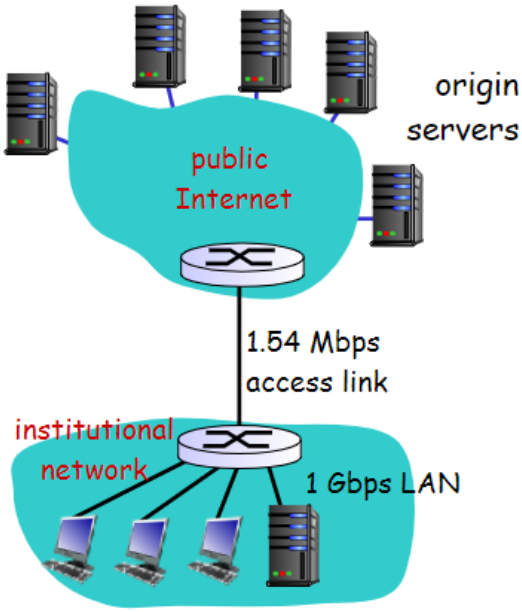
Caching example:

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:

- LAN utilization: 0.15%
- access link utilization = 97%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + **minutes** + usecs



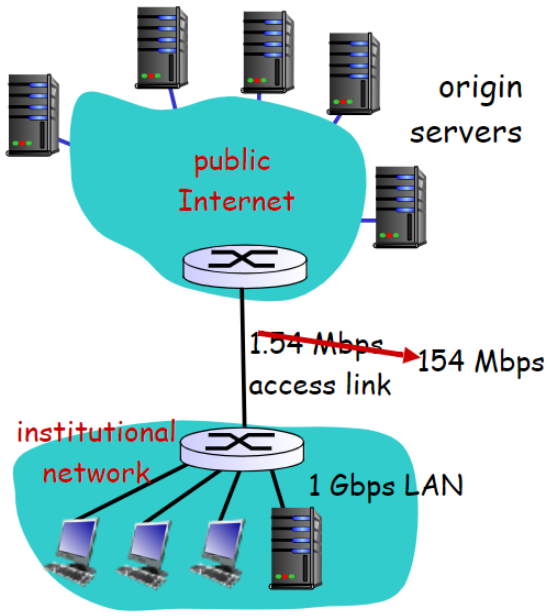
Caching example: fatter access link

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps → 154 Mbps

consequences:

- LAN utilization: 0.15%
- access link utilization = 97% → 0.97%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + minutes + usecs → msecs



Cost: increased access link speed (not cheap!)

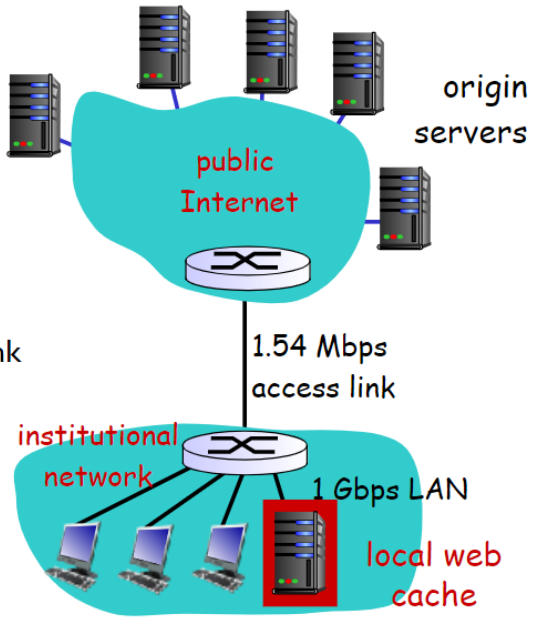
Caching example: install local cache

Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin

- access link utilization:
  - 60% of requests use access link
- data rate to browsers over access link
  = 0.6*1.50 Mbps = .9 Mbps
  - utilization = 0.9/1.54 = .58

- total delay
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs
  - less than with 154 Mbps link (and cheaper too!)



HTTP/1.1

HTTP1.1 introduced multiple, pipelined GETs over single TCP connection

- server responds in-order (FCFS: First Come First Served scheduling) to GET requests
- with FCFS, small object may have to wait for transmission (**head-of-line (HOL) blocking**) behind large objects
- loss recovery (retransmitting lost TCP segments) stalls object transmission
- even if server knows client is likely to request certain resources, it has to wait for client to send them

> Head-of-line blocking (HOL blocking): 队头阻塞，其原因是队列的第一个数据包受阻而导致整列数据包受阻

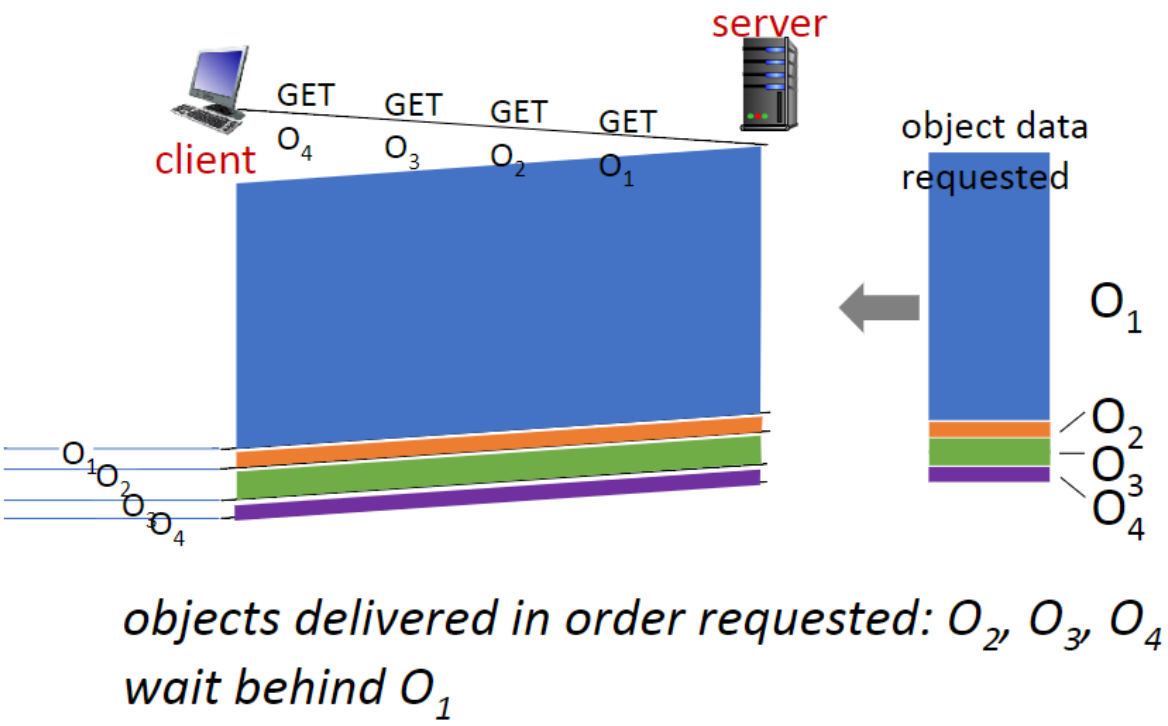
HTTP/2

> Key goal: decreased delay in multi-object HTTP requests

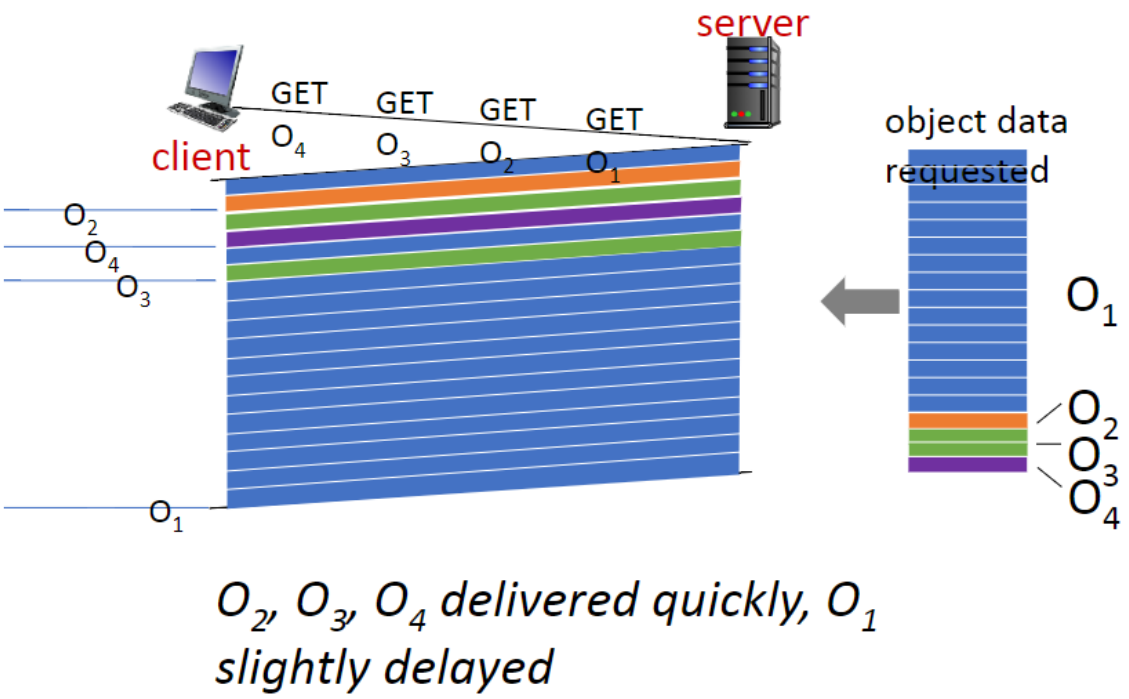
- Increased flexibility at server in sending objects to client.
- Methods, status codes, most header fields unchanged from HTTP/1.1
- transmission order of requested objects based on **client-specified object priority** (not necessarily FCFS)
- *push* **unrequested** object to client
- interleave multiple requests/responses within same connection
  - **divide** objects into frames, schedule frames to mitigate HOL blocking

HTTP/2: mitigating HOL blocking

HTTP/1.1: client request 1 large object and 3 smaller objects



HTTP/2: objects divided into frames, frame transmission **interleaved**



HTTP/2 to HTTP/3

HTTP/2 over single TCP connection means:

- recovery from packet loss still stalls all object transmission
  - as in HTTP 1.1, browsers have incentive to open multiple parallel TCP connections to reduce stalling, increase overall throughput
- No security over vanilla (普通的) TCP connection
- **HTTP/3:** adds security, per object error and congestion control (more pipelining) over UDP

Electronic Mail

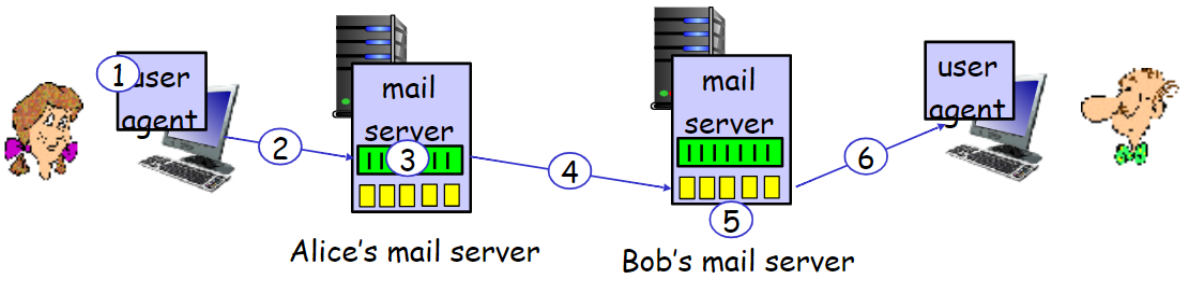
- Three major components:
  - User agents
  - mail servers
  - simple mail transfer protocol (SMTP)
- User agent
  - A mail reader
  - composing, editing, reading mail messages
  - outgoing, incoming messages stored on server
- Mail server
  - mailbox contains incoming messages for each user
  - message queue of outgoing messages
- SMTP (Simple Mail Transfer Protocol
  - Send email messages between mail servers
  - client is the sending mail server
  - “server” is the receiving mail server

SMTP

- SMTP uses TCP to reliably transfer email message from client to server, default in port 25
- direct transfer: sending server to receiving server
- There are three phases to transfer:
  1. handshaking 握手信号
  2. transfer of messages
  3. closure
- command or response interaction (like HTTP)
  - ASCII text as the command
  - status code and phrase as response
- message must be in 7-bit ASCII

Scenario: Alice sends message to bob

1. Alice uses UA (user agent) to compose message to xx@xxxx.com
2. Alice's UA sends message to her mail server via SMTP. Message placed in message queue.
3. client side of SMTP opens TCP connection with receiver (AKA router server or Bob's mail server)
4. SMTP client sends Alice's message over the TCP connection
5. Bob's mail server places the message in mailbox
6. Bob invoke his UA to read message



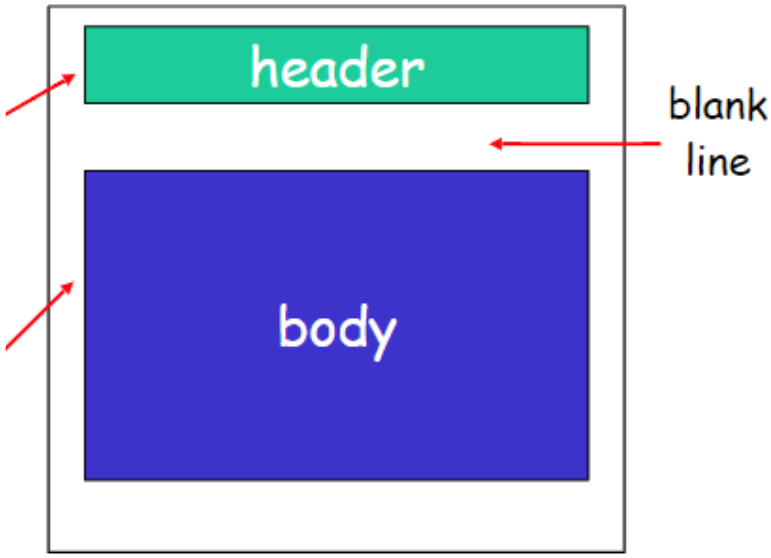
- Sample SMTP interaction

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

SMTP: final words

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses CRLF.CRLF to determine the end of message
- Comparison with HTTP:
  - HTTP: pull
  - SMTP: push
  - both have ASCII command/response interaction, status codes
  - HTTP: each object encapsulated 封装 in its own response message
  - SMTP: multiple objects sent in multipart message, encoded in ASCII

Mail message format



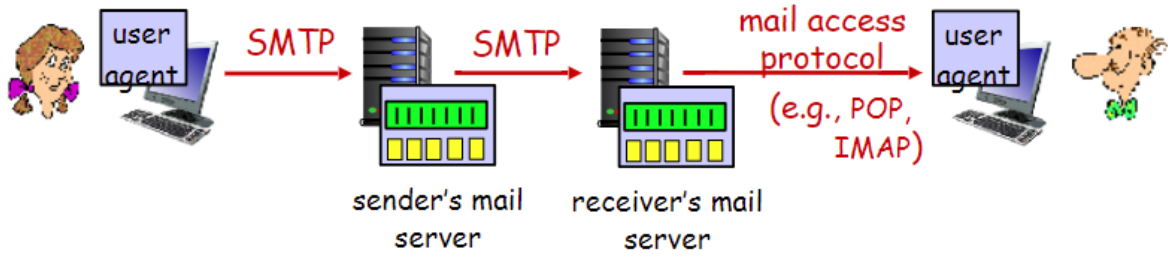
SMTP is the protocol for exchanging email messages

- Header lines:
  - From
  - Rcpt to:

RFC 822 is a standard for text message format:

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

Mail access protocols



- SMTP: delivery and storage to receiver's server
- mail access protocol: retrieval from server:
  - POP (Post Office Protocol): authorization, download
  - IMAP (Internet Mail Access Protocol): More features than POP, like manipulation of stored message s on server
  - HTTP: gmail, Hotmail, Yahoo mail

POP3 Protocol

- Authorization phase

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

  - client commands:
    - User: declare username
    - pass: passwd
  - server responses
    - +OK
    - -ERR
- Transaction phase, client:

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

    - list: list message numbers
    - retr: retrieve message by number
    - dele: delete
    - quit
- Download and delete Mode:
    - cannot re-read the email if the client has changed
- Download and keep mode:
    - copies of messages on different clients
- POP3 is stateless across sessions

Features between POP3, IMAP and web mail

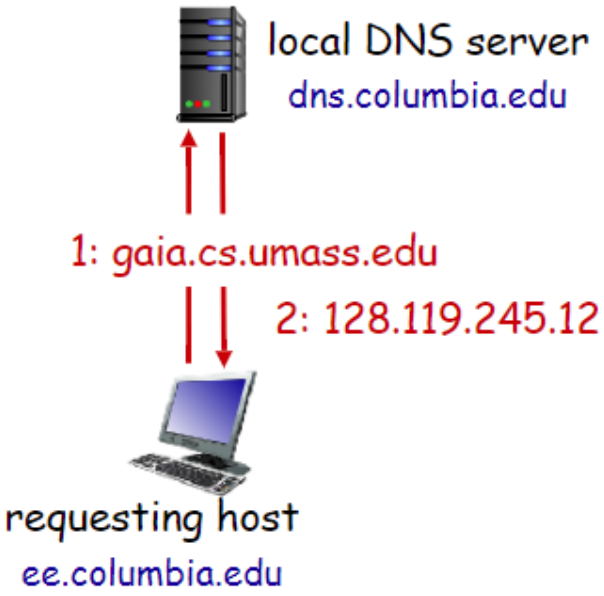
POP3	IMAP	Web mail
Once download from server, cannot re-read on other client under download and delete mode		
Will have download and keep mode to allow to save message on multiple devices	Keep all messages (email) at the server	Send/receive mail via HTTP to mail server
Mail server communicate each other use SMTP		
	Allow user to organize messages in folders	
Stateless across session	Keep user state across sessions	

DNS - Domain Name System

💡 Each object has identifiers: For people, name and passport are identifiers For Internet Hosts and Routers, IP address and domain name are identidiers The question that the DNS need to handle is that, how to map between the IP address and the domain name? DNS存储IP地址和主机万维网（WWW）地址的映射

- Domain Name System (DNS)
    - distributed database
        - implementated in hierarchy of many name servers
    - application-layer protocol
        - hosts, name server communicate to resole names
        - core Internet function implemented as application layer protocol
        - used by many other applications and app-layer protocols
        - complexity at network's "edge"

DNS name resolution: client view

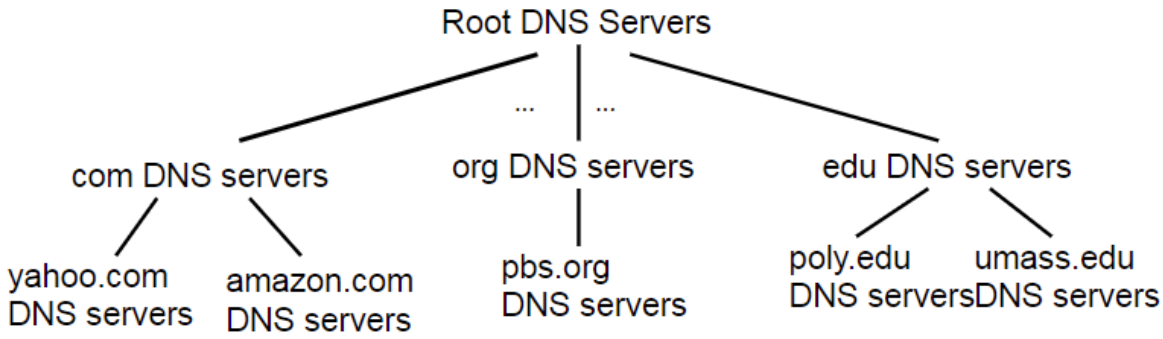


- For example, the host at “ee.columbia.edu” wants the IP address for “gaia.cs.umass.edu”
  - The host of “ee.columbia.edu” asks local DNS server
    - Which is usually hosted at local ISP
    - UDP port 53
  - For client’s perspective, local DNS server responds with answer:
    - rest of DNS system is a black box for client

DNS: services, structure

- DNS services
  - Hostname to IP address translation
  - Host aliasing 主机别名
    - canonical, alias names 典型的，别名的名字
  - mail server aliasing
  - load distribution 负载均衡
    - replicated Web servers: many IP address correspond to one name 将多个IP地址映射到同一个主机名
- Why DNS is distributed not centralized?
  - single point of failure
  - traffic volume
  - distant centralized database
  - maintenance/updates

DNS: a distributed, hierarchical database分布式的、分层的数据库



client wants IP for www.amazon.com; 1st approximation:

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

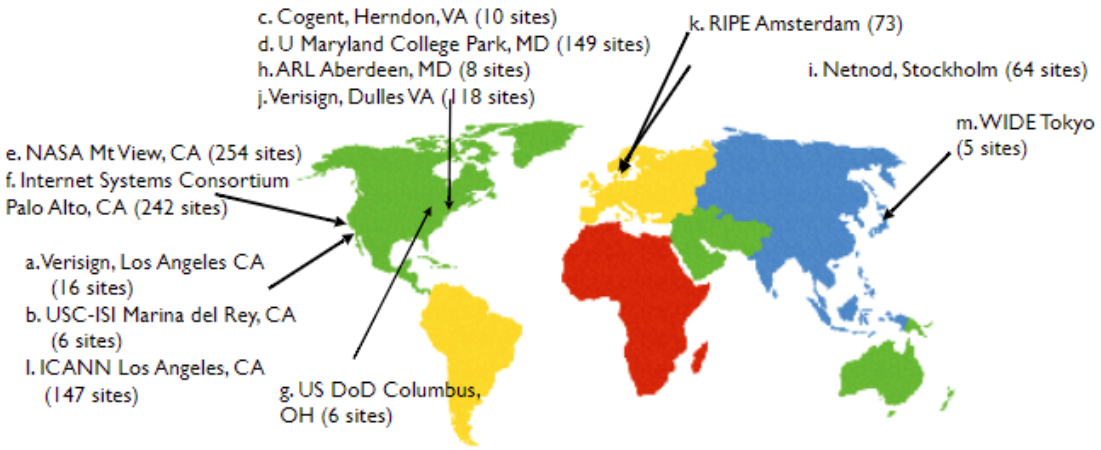
DNS: Root Name Servers

- DNS root name server contacted by local name server that cannot resolve name
  - root name server can contact any, regardless of what query
  - root server IP addresses loaded into ISP's resolver

DNS根服务器由无法解析主机名称的本地DNS服务器联系
只有在本地名称服务器不能解析时，才会和根服务器通信

💡 全世界有13台逻辑DNS根服务器，其中为1台主根服务器和12台辅根服务器。并在多个国家有根服务器的镜像

13 logical root name "servers" worldwide
•each "server" replicated many times
•numbers updated 10/2020



TLD, authoritative servers 顶级域名服务器，权威DNS服务器

- TLD (top-level domain) servers 顶级域名服务器
  - com, org, net, edu and all top-level country domain

顶级域名像com，顶级国家域名像uk，cn

  - VeriSign maintains servers for .com TLD
  - Educause for .edu TLD
- authoritative DNS servers 权威DNS服务器
  - orgnisation's own DNS servers, providing authoritative hostname to IP mappings for organisation's name hosts
  - every organisation with publicly accessible hosts must provide publicly accessible DNS records to map hosts to IP address
  - can be maintained by organisation or service provideer

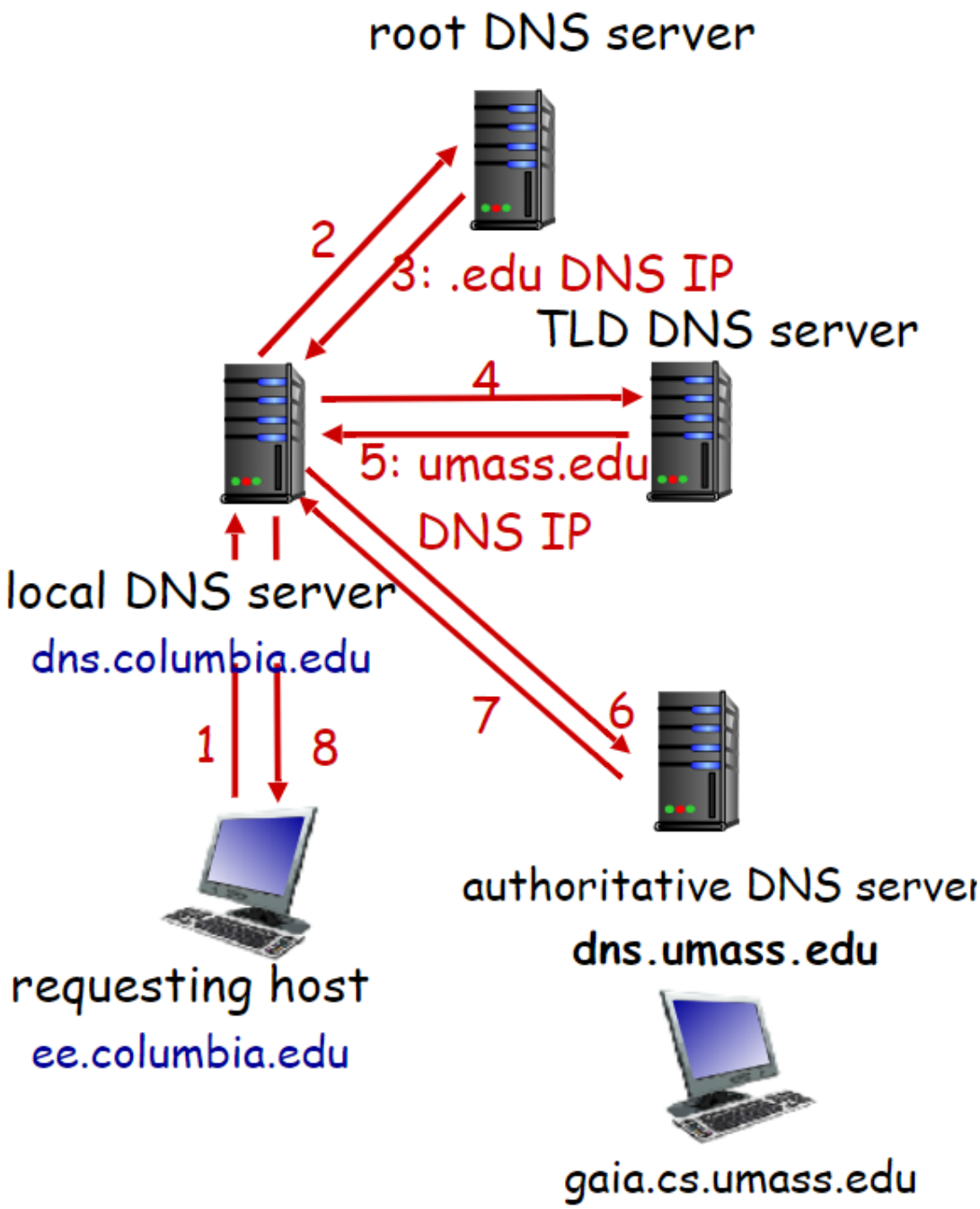
> 权威DNS服务器由组织或服务供应商提供及维护（Cloudflare，DNSpod），必须提供可公开访问的DNS记录以映射IP地址和主机名

Local DNS name server 本地DNS服务器

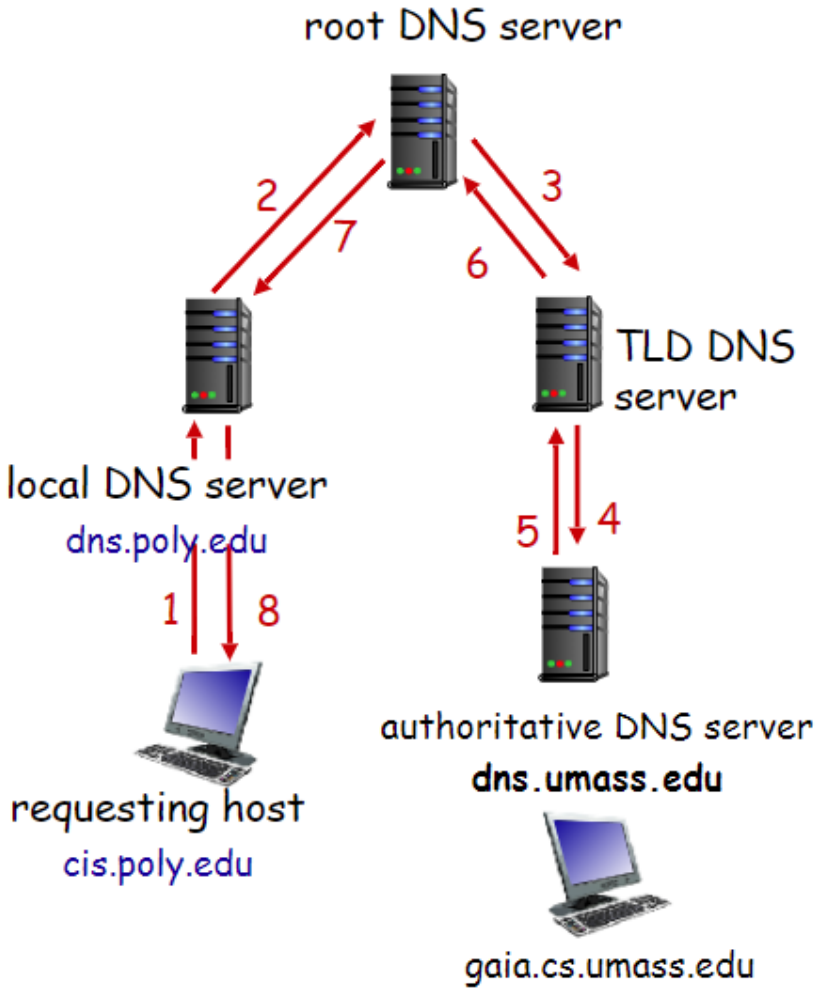
- Does not strictly belong to hierarchy 本地DNS服务器并不属于严格意义上的分层制度中
- Each ISP (residential ISP, company, university) has one
  - also called "default name server"
  - also public DNS server
    - Google public DNS (8.8.8.8), cloudflare DNS (1.1.1.1)
- when host makes a DNS query, query is sent to its local DNS server
  - The local DNS server has local cache of recent name-to-address mapping (may out of date)
  - The local DNS server acts as proxy, forwards query into hierarchy

DNS name resolution example DNS解析举例

Iterative query:



Recursive query:



DNS: caching, updating records

- Once name server learns mapping, it caches mapping
  - cache entries timeout after specific time (TTL)
  - TLD servers typically cached in local name servers
- cached entries may out of date
- update and notify mechanisms proposed IETF standard (RFC 2136)

DNS records

- Distributed database storintg resource records (RR)
  - RR format: (name, value, type, ttl)
  - Type = A (A记录)
    - 主机名和IP地址的映射被称作A记录
    - name is hostname, value is IP address

  - Type = CNAME (CNAME记录)
    - 主机名的别名
    - name is alias name
    - value is canonical name

www.wu.engineer 是 wu.engineer. 的别名并通过 Cloudflare 代理其流量。

类型	名称 （必需）	目标 （必需）	代理状态	TTL
CNAM	www	@	已代理	自动

root 使用 @

- Type = NS
  - 即标明主机使用的命名服务器名称
  - Name is domain
  - value is hostname of authoritative name server for this domain

Cloudflare 名称服务器

要使用 Cloudflare，请确保已更改权威 DNS 服务器或名称服务器。这些服务器是分配的 Cloudflare 名称服务器。

类型	值
NS	kaiser.ns.cloudflare.com
NS	serena.ns.cloudflare.com

- Type = MX
  - value is name of mailserver associated with name

aspmx.l.google.com 为 wu.engineer 处理邮件。

类型	名称 （必需）	邮件服务器 （必需）	TTL	优先级 （必需）
MX	@	aspmx.l.google.com	自动	1

root 使用 @

0 - 65535

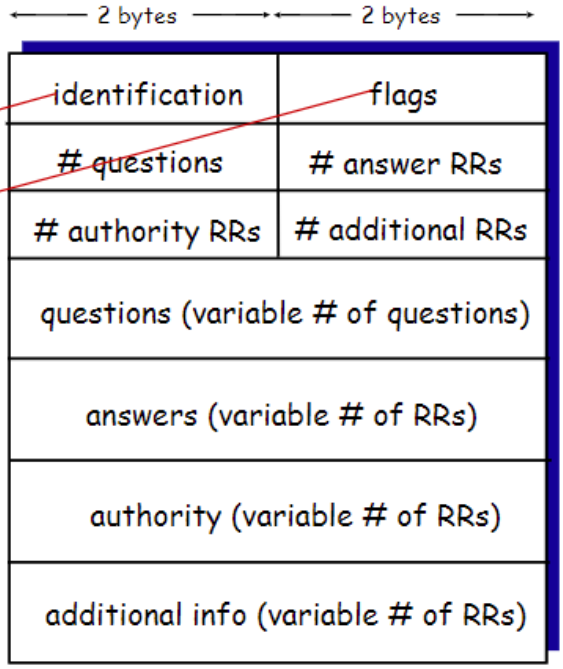
取消 保存

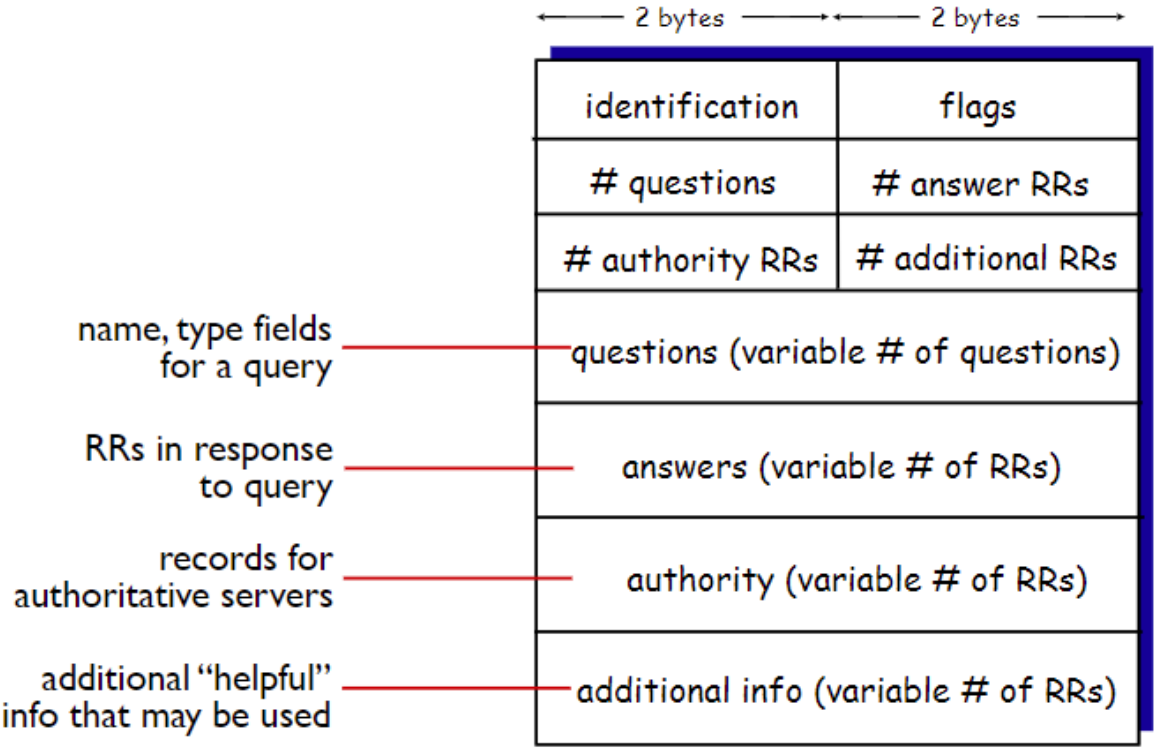
DNS protocol, messages

- query and reply messages, both with same message format

message header

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





Inserting records into DNS

- example: new startup "Network Utopia"
- register name networkutopia.com at DNS registrar
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts two RRs into .com TLD server
- create authoritative server type A record for www.networkuptopia.com; type MX record for networkutopia.com

Attacking DNS

DDoS attacks 分布式拒绝服务

- Bombard root server with traffic
  - traffic filtering
  - local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombard TLD servers
  - potentially more dangerous

Redirect attacks 重定向攻击

- main-in-middle 中间人攻击
  - Intercept queries
- DNS poisoning DNS污染 (xx)
  - Send bogus replies to DNS, which caches

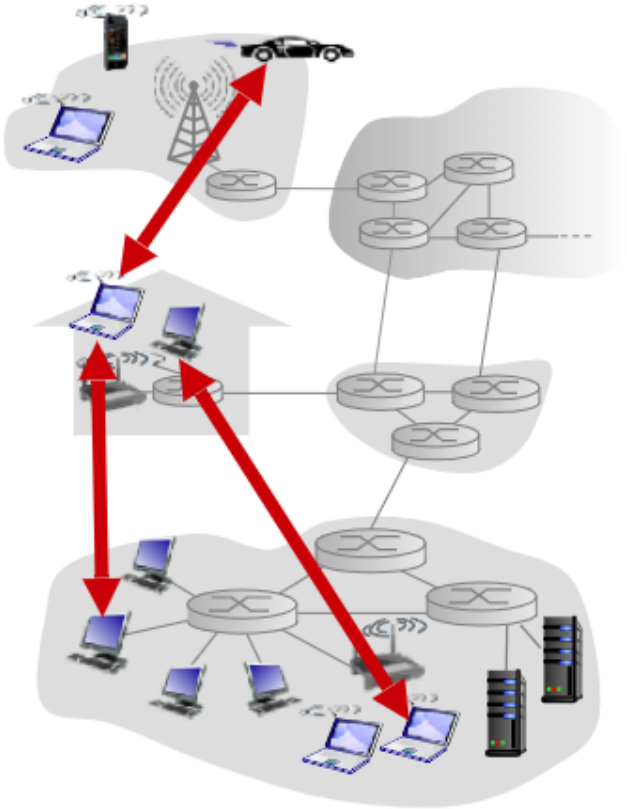
Exploit DNS for DDoS

- Send queries with spoofed source address: target IP
- requires amplification

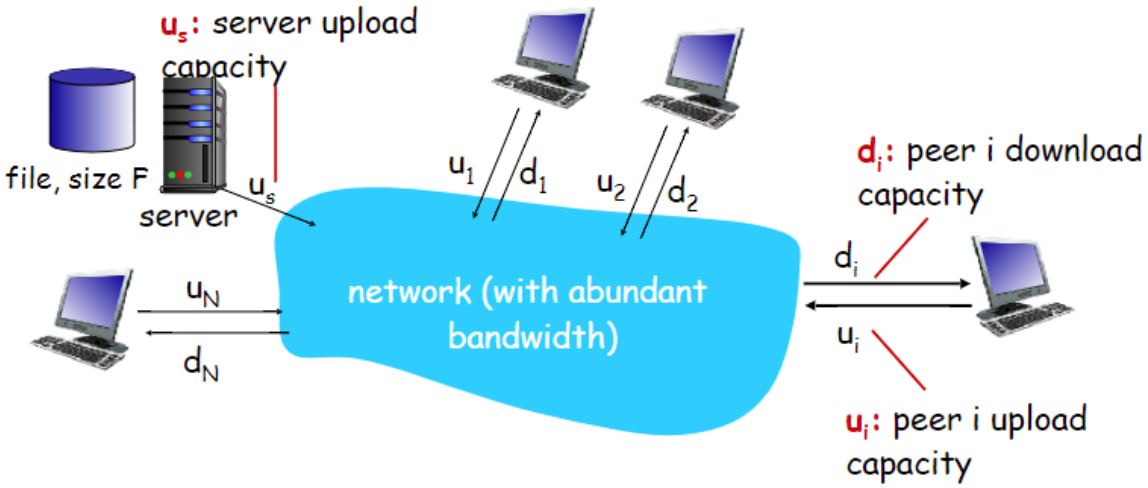
P2P applications

Pure P2P architecture

- no always-on server
- arbitrary end systems directly communicate 终端之间的直接通讯
- peers are intermittently connected and change IP addresses
- For example:
  - File distribution - BitTorrent
  - PPTV (Streaming)
  - VoIP



File distribution: client-server vs P2P



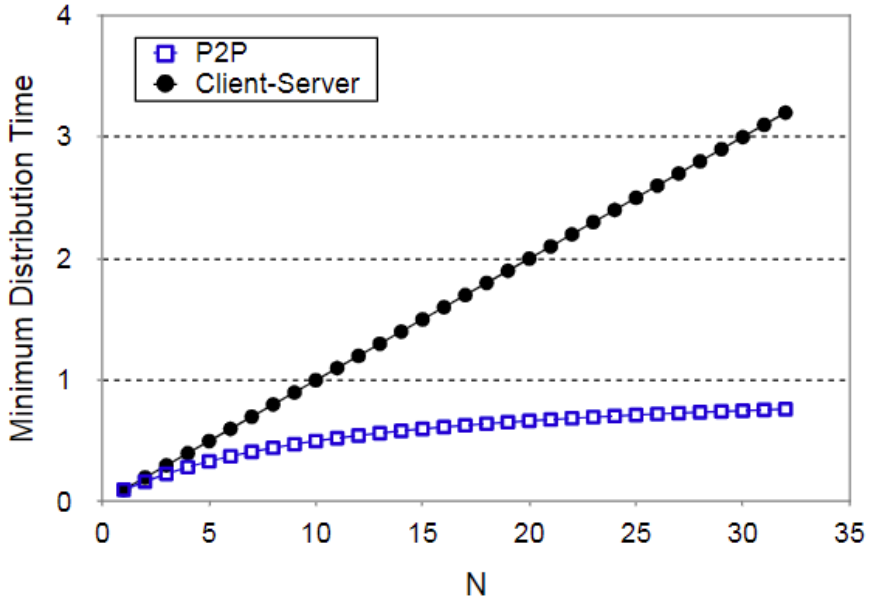
- Server transmission: must sequentially send (upload) N file copies from server

- time to send one copy: F/u_s
- Time to send N copies at one time: FN/u_s

- Client: each client must download file copy
  - d_{min} = min client download rate
  - F/d_{min} = max download time
- Time to distribute a file F to N client using client-server approach:

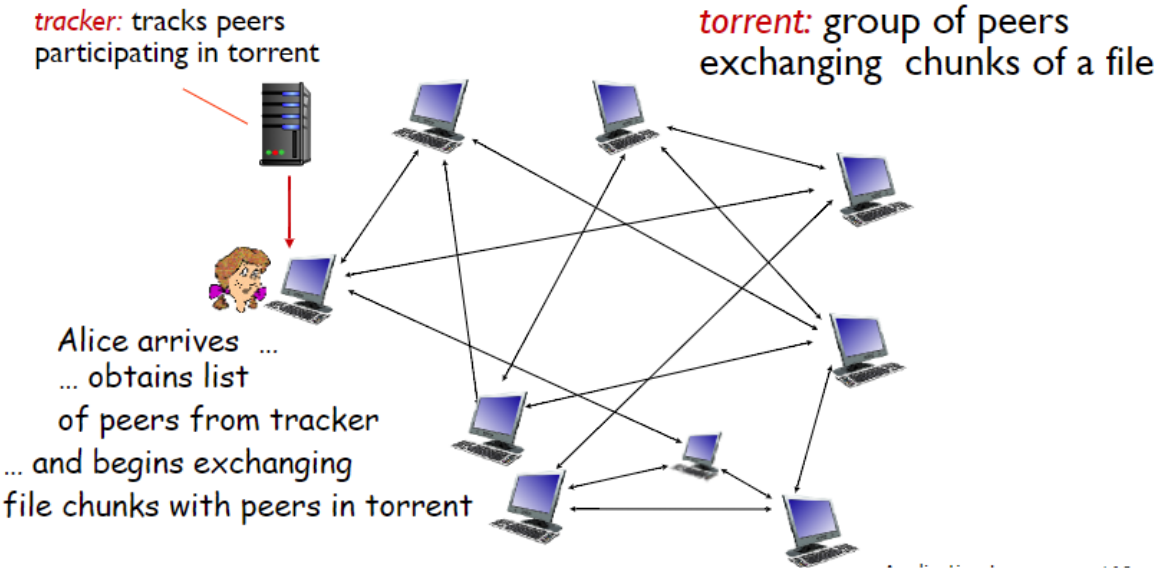
$$D_{c-s} \geq max\{NF/u_s, F/d_{min}\}$$

client upload rate = u, F/u = 1 hour, u_s = 10u, $d_{min} \geq u_s$



P2P file distribution: BitTorrent

- File is divided into chunks, e.g. of 256kb each
- peers in torrent send/receive file chunks



- peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers (download)
  - registers with tracker to get list of peers, connects to subset of peers (neighbors)
- while downloading, peer uploads chucks to other peers

- peer may change peers with whom it exchange chunks
- churn: peers may come and go
- once peer has entire file. it may leave or remain to seeding

BitTorrent: requesting, sending file chunks

- requesting chunks:
  - at any given time, different peers have differetn subsets of file chunks
  - periodically, user asks each peer for list of chunks that they have
  - user requests missing chunks from peers, rarest first
- sending chunks:
  - user sends chunks to those peers currently sending their chunks at highest rate
  - every 30 secs, randomly select another peer, start sending chunks

Video streaming and content distribution networks

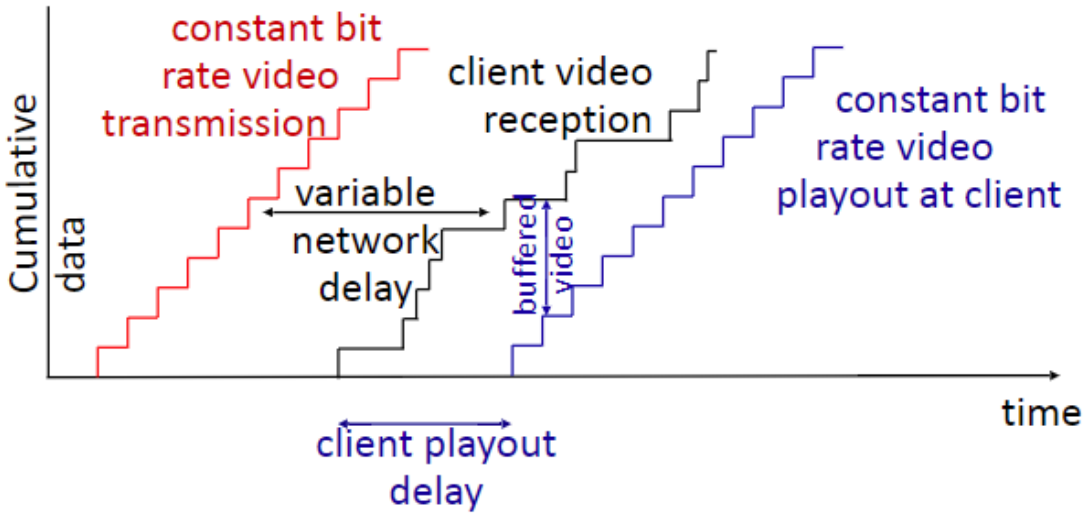
Multimedia: video

- CBR (constant bit rate): video encoding rate fixed
- VBR (variable bit rate): video encoding rate changes as amount of spatial
- examples
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG 2 (DVD) 3-6 Mbps
  - MPEG 4 < 1 Mbps

Streaming stored video

- Main challenges:
  - server-to-client bandwidth will vary over time, with changing network congestion level
  - packet loss, delay due to congestion
  - need to continuous playout

Streaming stored video: playout buffering



- client-side buffering and playout delay

播放时做缓冲

Streaming multimedia: DASH

- DASH: Dynamic, Adaptive Streaming over HTTP
- server side:
  - divides video file into chunks
  - each chunk stored and encoded at different rates
  - manifest files: provide URLs for different chunks
- client side:
  - periodically measures server-to-client bandwidth
  - consulting manifest, requests one chunk at time

码率自适应/自适应流畅度

Content distribution networks (CDN)

Challenge: How to stream content to thousands of simultaneous users?

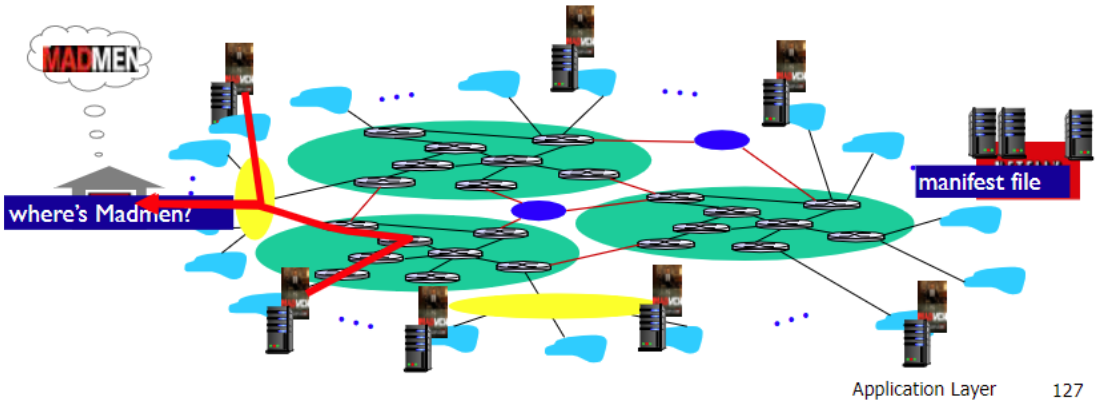
- CDN steers client request to server that is ‘close’ to client or has high bandwidth

Multiple ways for CDN

- enter deep: push CDN servers into many access networks
  - close to users
  - Used by Google, Netflix
- bring home: smaller number of larger clusters in POPs near but not within access network

Principle

- CDN stores copies of content at CDN nodes
- users requests content from CDN
  - directed to the nearby copy, then user retrieves contene
  - may choose different copy if network path congested



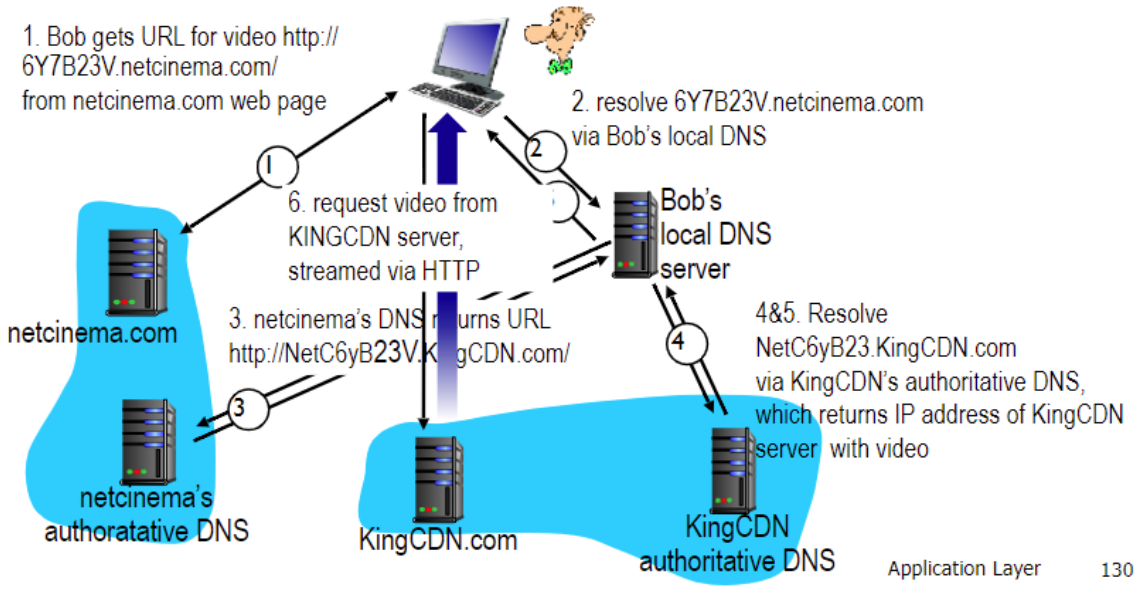
OTT(over the top) challenge: coping with a congested internet

- from which CDN node to retrieve content?
- user behaviour when congestion occurred?
- what content to place in which CDN node?

How to direct user to particular CDN node

Three main approaches:

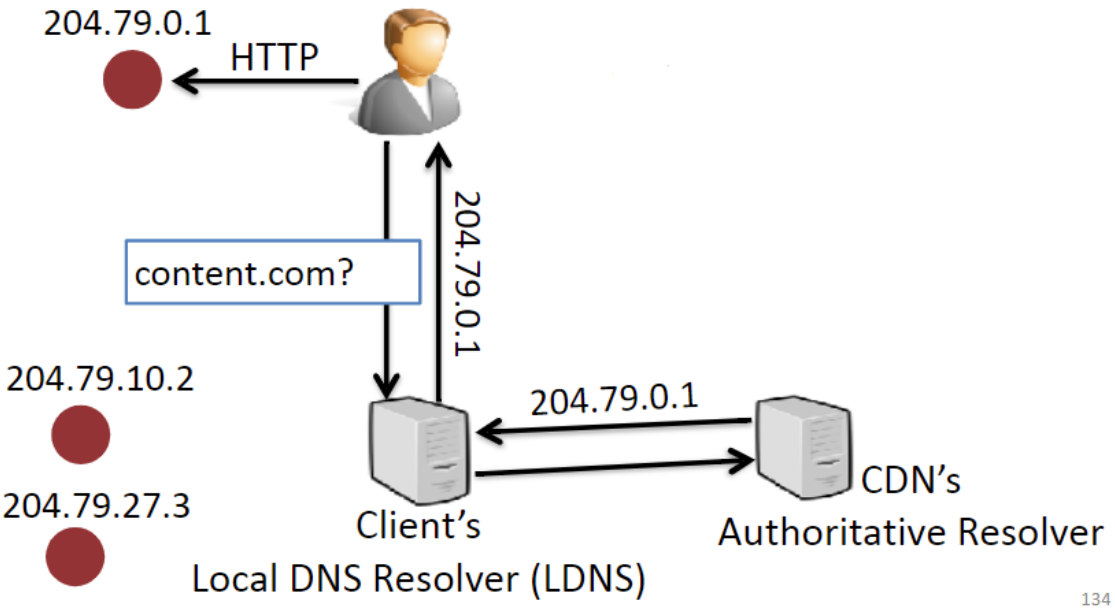
1. custom URL



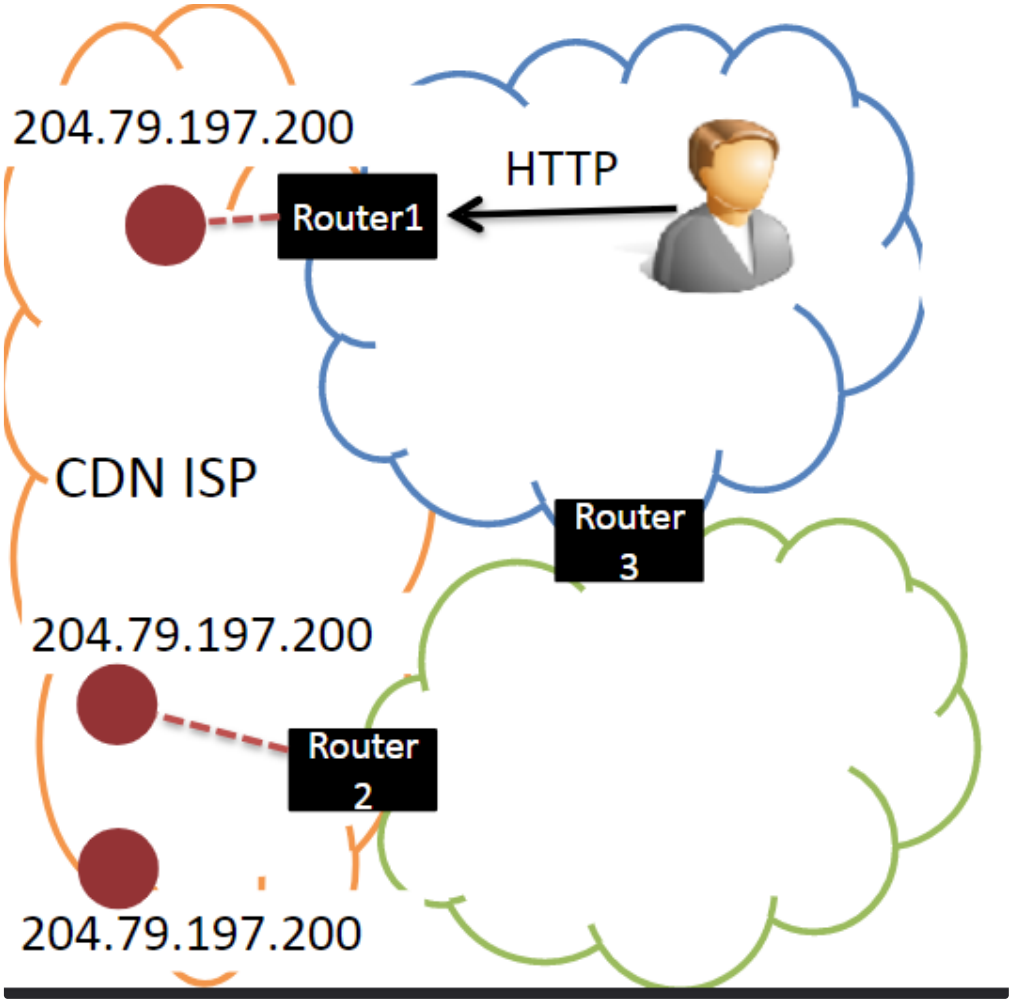
   Pros: precise per-client targeting

   Cons: not relevant on first page, since customization is embedded in URL

2. DNS redirection
   - Which is a traditional approach, used by Akamai
   - Each server has own IP address
   - CDN's client gives client the IP address of a particular physical server



3. Anycast routing
   - Gaining in popularity
   - All servers use the same IP address
   - BGP routing picks server
   - Run HTTP/TCP over anycast



	DNS	Anycast
Operational Complexity	- Complex: requires global traffic manager	+ Simple to deploy and operate + Naturally resilient against DDOS + High availability and fast fail over
Control	+ CDN-controlled + Picks server of choice + React quickly to: • Changes in performance • Changes in load	- BGP-controlled - Subject to vagaries of BGP routing • May not pick nearest server • Hard to control load
Granularity	- Per LDNS can be poor representative of clients	+ Per packet

142

Chapter 3: Transport Layer

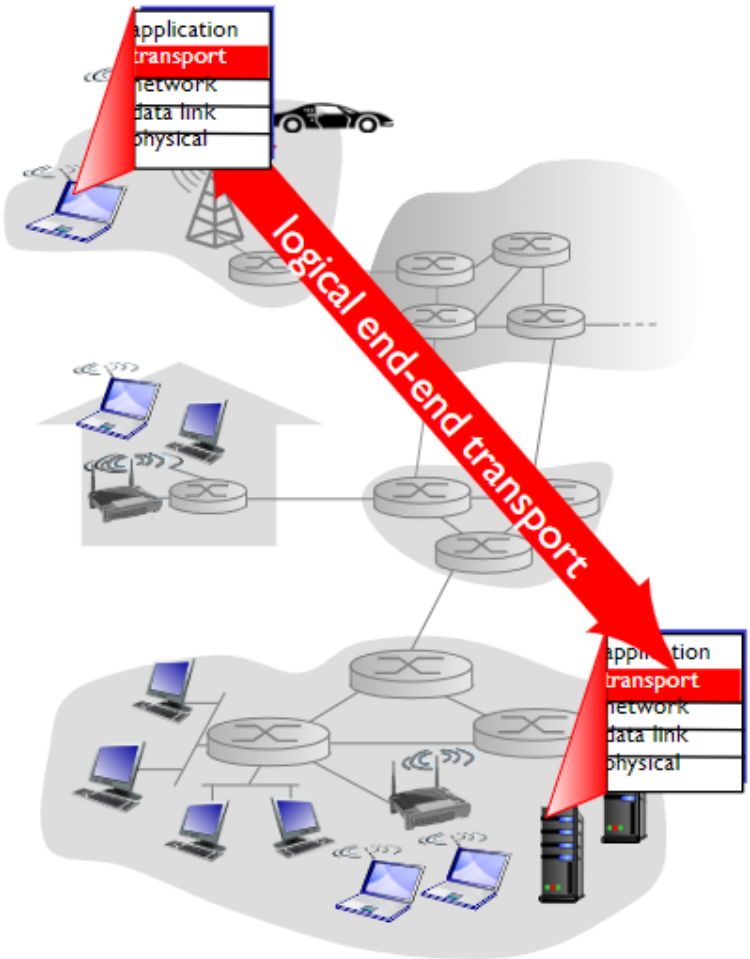
Understand principles behind transport services:

- multiplexing
- demultiplexing
- reliable data transfer
- flow control
- congestion control

Learn about Internet transport layer protocols:

- UDP: connectionless transport
- TCP: connection-oriented reliable transport
- TCP congestion control

3.1 Transport Layer Services



Transport services and protocols

- Provide **logical communication** between app processes running on different hosts
- Transport protocols run in end systems
  - Send side: breaks app message into few **segments**, passes to network layer.
  - Receive side: reassembles segments into messages, passes to app layer.
- More than one transport protocol available to apps:
  - For example the Internet has TCP and UDP.

Transport vs. Network layer

- Network layer: logical communication between **hosts**.
- Transport layer: logical communication between **processes**.
  - relies on and enhances network layer services.

> Network layer 类似于房屋(host)和房屋之间的邮递信件服务，而Transport layer 则类似于房屋内部对信件的分配（即该信件是发送给哪位家庭成员(process)的）

Transport layer actions

- Sender:
  - passed an application layer message
  - determines segment header fields values
  - creates segment
  - passes segments to IP
- Receiver:
  - receives segments from IP
  - checks header values
  - extracts application-layer message
  - demultiplexes message up to application via **socket**

Internet transport-layer protocols

- TCP (Transmission Control Protocol): reliable, in-order delivery
  - Congestion control
  - Flow control
  - Connection setup
  - Bytestream abstraction
- UDP (User Datagram Protocol): unreliable, unordered delivery
  - No-frills extension of 'best-effort' IP
  - Datagram abstraction
- Both services not avaliable:
  - Delay guarantees
  - Bandwidth guarantees

3.2 Multiplexing and Demultiplexing 多路复用和多路分解

Multiplexing/Demultiplexing

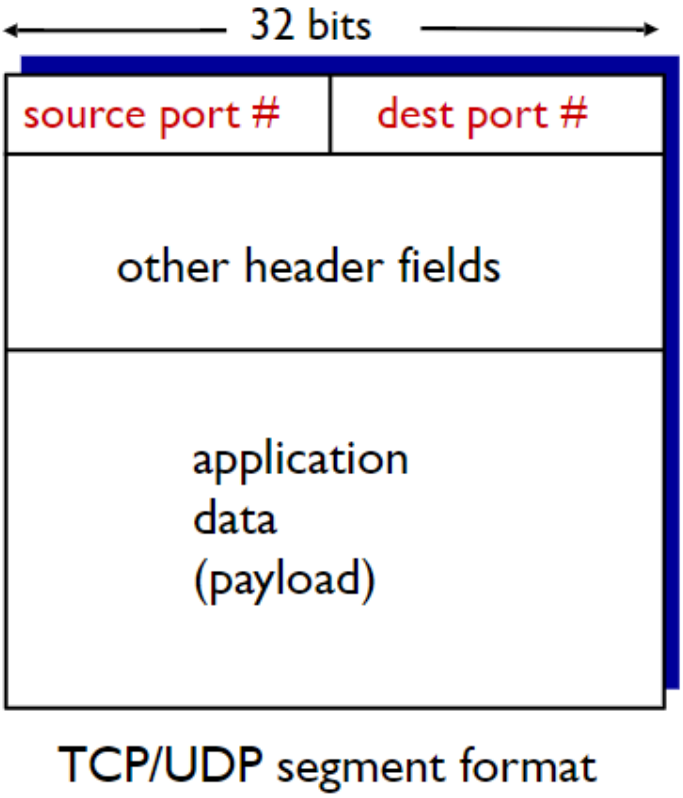
- Multiplexing at sender:
  Handle data from multiple sockets, add transport header
- Demultiplexing at receiver:
  Use header information to deliver received segments to correct socket

> 将主机间交付（网络层）拓展到进程间交付（运输层）被称为"运输层的多路复用和多路分解"
>
> 从不同socket中收集讯息(segments)，并为每个讯息封装header形成datagram，将所有datagram组成报文段，然后将报文段传递到网络层(IP)被称为多路复用(multiplexing)
>
> 将运输层的报文段中的datagram准确交付到对应的socket被称为多路分解（demultiplexing)

How demultiplexing works

- Host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries one transport-layer segment
  - each segment has source, destination port number
- Host uses **IP address & port numbers** to direct segment to appropriate socket.

> header 中包含IP address 和 port number



TCP/UDP segment format

Connectionless demultiplexing 无连接的多路分解(UDP)

- Create socket has host-local port number:

```python
from socket import *

serverName = 'hostname'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = raw_input('input lowercase sentence:')
clientSocket.sendto(message.encode(), (serverName, serverPort))

modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print(modifiedMessage.decode())
clientSocket.close()
```

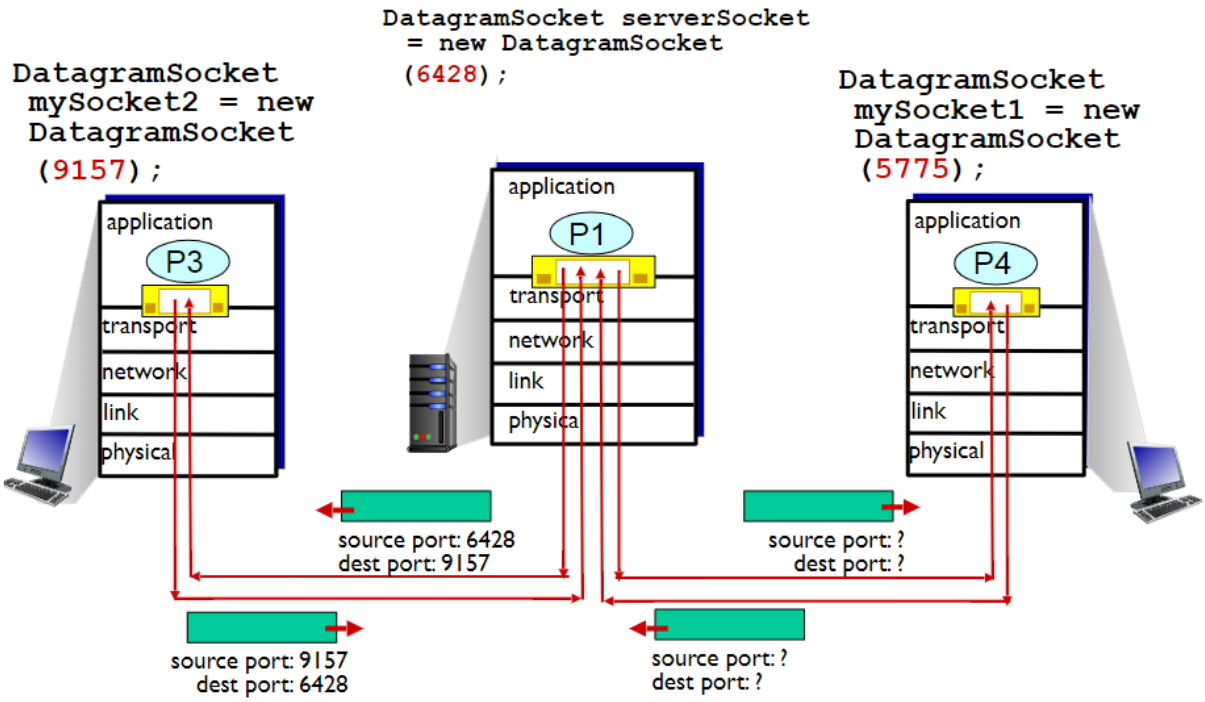
> 此socket为client所创建，所用的端口号12000为client接收讯息时使用的socket的端口号，此端口号被分配给mySocket1的UDP socket

- When creating datagram to send into UDP socket, must specify:
  - destination IP address

  - destination port number
- When host receives UDP segments:
  - checks destination port number in segment
  - direct UDP segment to socket with that port number
- IP datagrams with **same destination port number, but different source IP addr and/or port #** will be directed to **same socket** at destination.

即只要目标IP地址和端口号是不变的，那么到达目标是使用的UDP socket也是使用同一个，无论源IP地址或源端口号

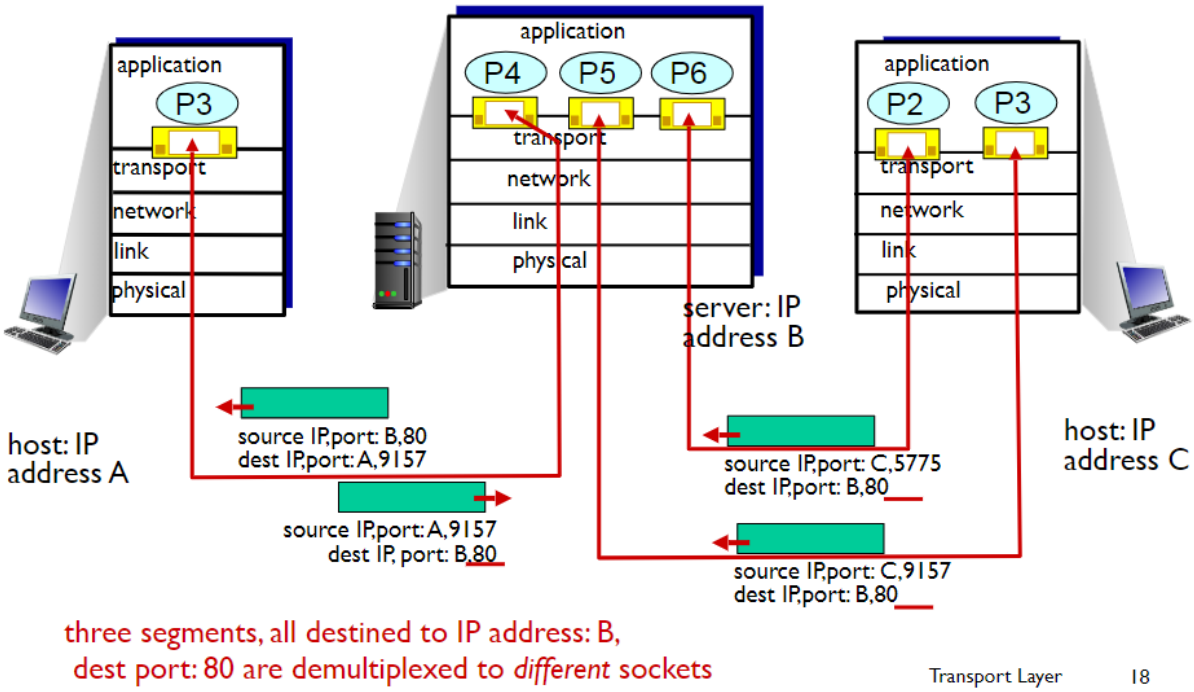
Example



Connection-oriented demultiplexing 有连接的多路分解(TCP)

- TCP socket is identified by **4-tuple**:
  - source IP address
  - source port number
  - destination IP address
  - destination port number
- Demultiplexing (demux): receiver uses **all 4 values** to direct segment to correct socket.
- Server host may support may simultaneous TCP sockets.
- Web servers have different sockets for each connecting client.
  - non-persistent HTTP will have different socket for each request.

Example



Summary

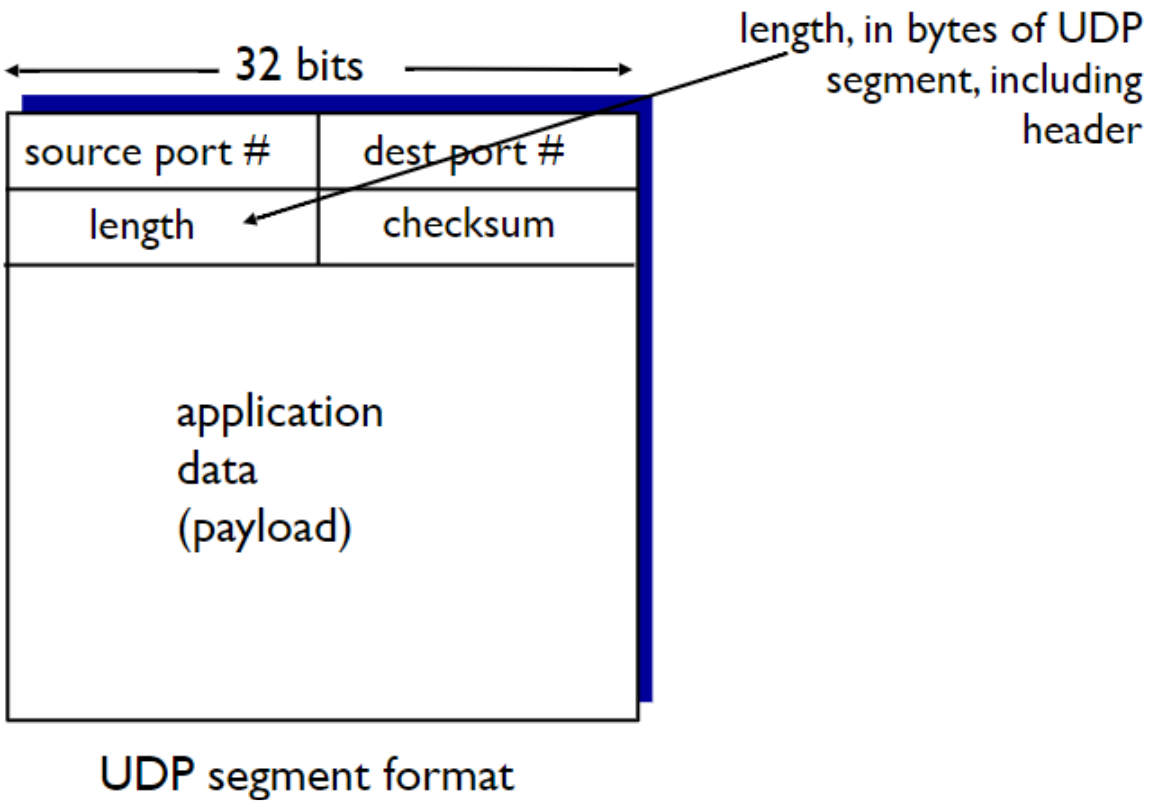
- Multiplexing, demultiplexing: based on segment, datagram header field values.
- UDP: demultiplexing using destination port number only.
- TCP: demultiplexing using 4-tuple: source IP address and port number, destination IP address and port number.
- Multiplexing/demultiplexing happen at *all* layers

3.3 Connectionless transport: UDP

UDP: User Datagram Protocol [RFC 768]

- 'No-frills', 'bare bones' Internet transport protocol 赤裸的网络协议
- 'Best-effort' service, UDP segments may be:
  - lost
  - delivered out-of-order to app
- **Connectionless**:
  - No handshaking between UDP sender and receiver
  - Each UDP segment handled independently of others
- UDP uses include:
  - streaming multimedia apps (allow loss, rate sensitive)
  - DNS
  - SNMP
  - HTTP/3
- Reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery

UDP: segment header



UDP segment format

Why is there a UDP?

- No connection establishment (less delay)
- Simple: no connection rate at sender and receiver
- No congestion control: UDP can blast away as fast as possible

UDP checksum 校验值

Goal: detect “errors” (flipped bits) in transmitted segments

Sender:

- Treat segment contents, including header fields, as sequence of 16-bit integers
- Checksum: addition (one’s complement sum) of segment contents
- Sender puts checksum value into UDP checksum field

Receiver:

- Compute checksum of received segment
- Check if computed checksum equals checksum field value:
  - If NO: error detected
  - If YES: no error detected *or may has error* .

Example

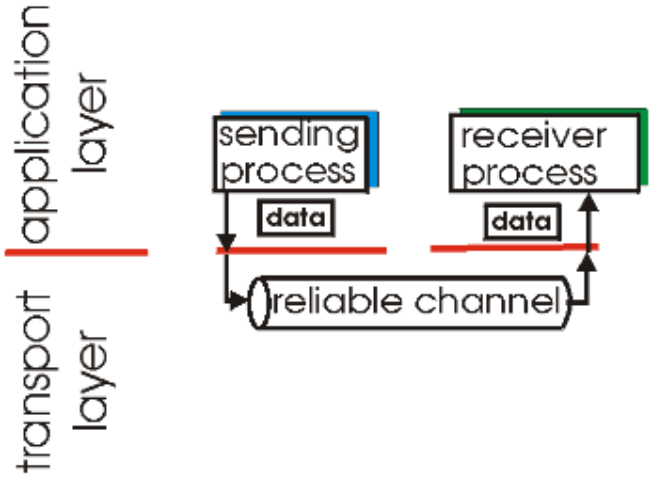
example: add two 16-bit integers

	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

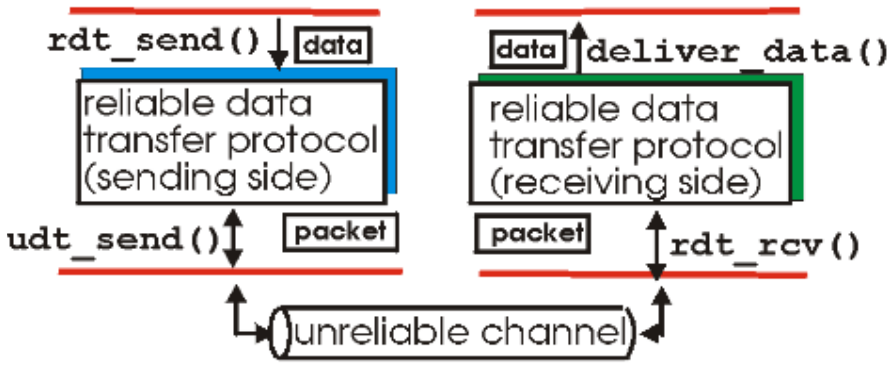
Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

3.4 Principles of reliable data transfer

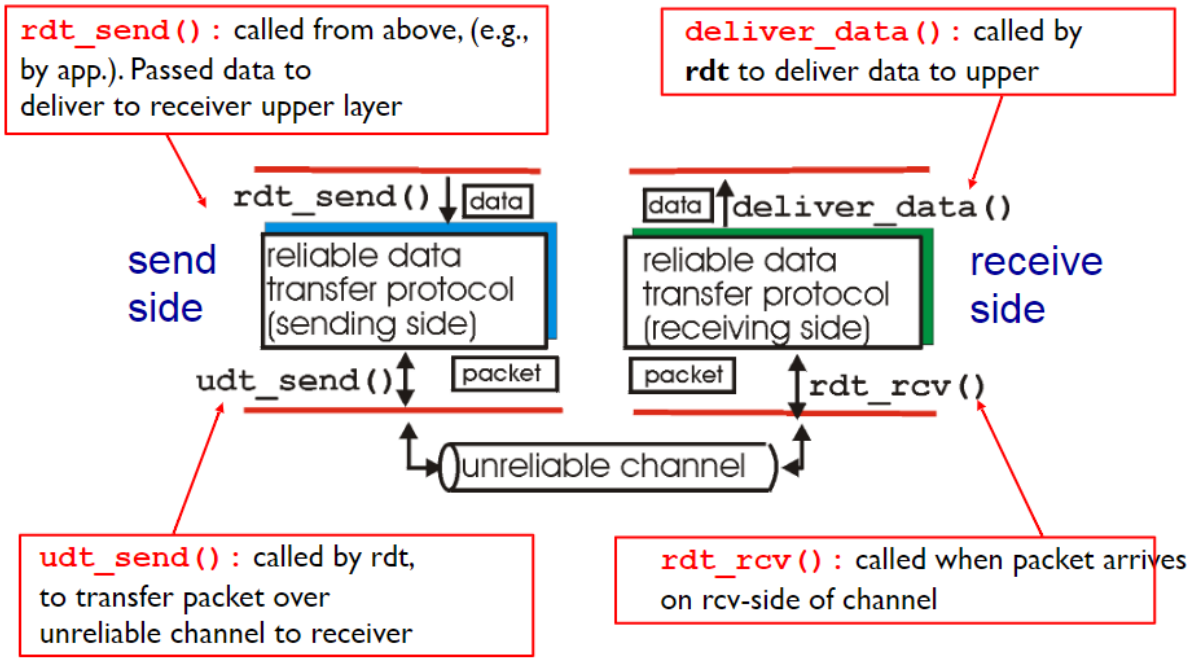
- Important in application, transport and link layers



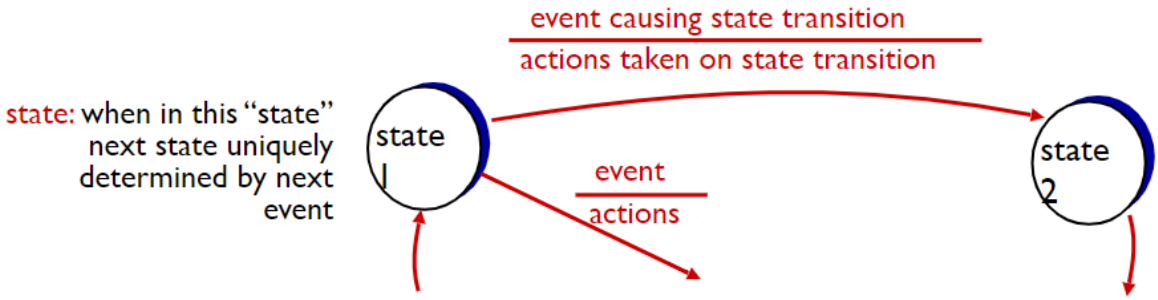
- The reliable channel should be defined as **bit in order, no corruption/loss**.
- Characteristics of unreliable channel will determine complexity of reliable data transfer protocol.



Reliable data transfer: getting started

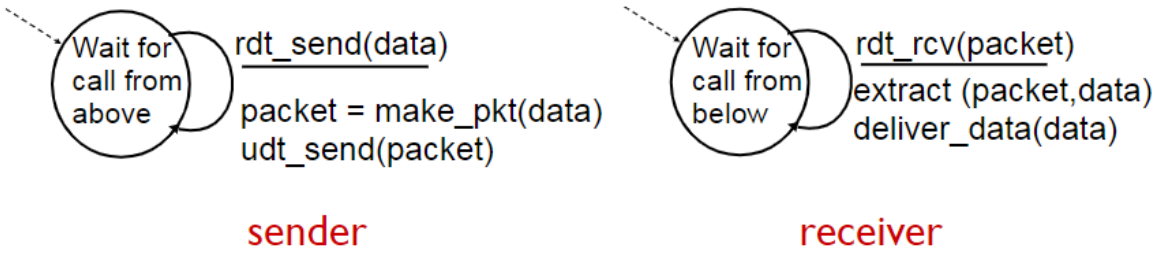


- Incrementally develop sender, receiver sides of Reliable Data Transfer protocol (rdt)
- Consider only unidirectional data transfer
  - but control info will flow on both directions
- Use finite state machines (FSM) to specify sender and receiver



rdt 1.0: reliable transfer over a reliable channel

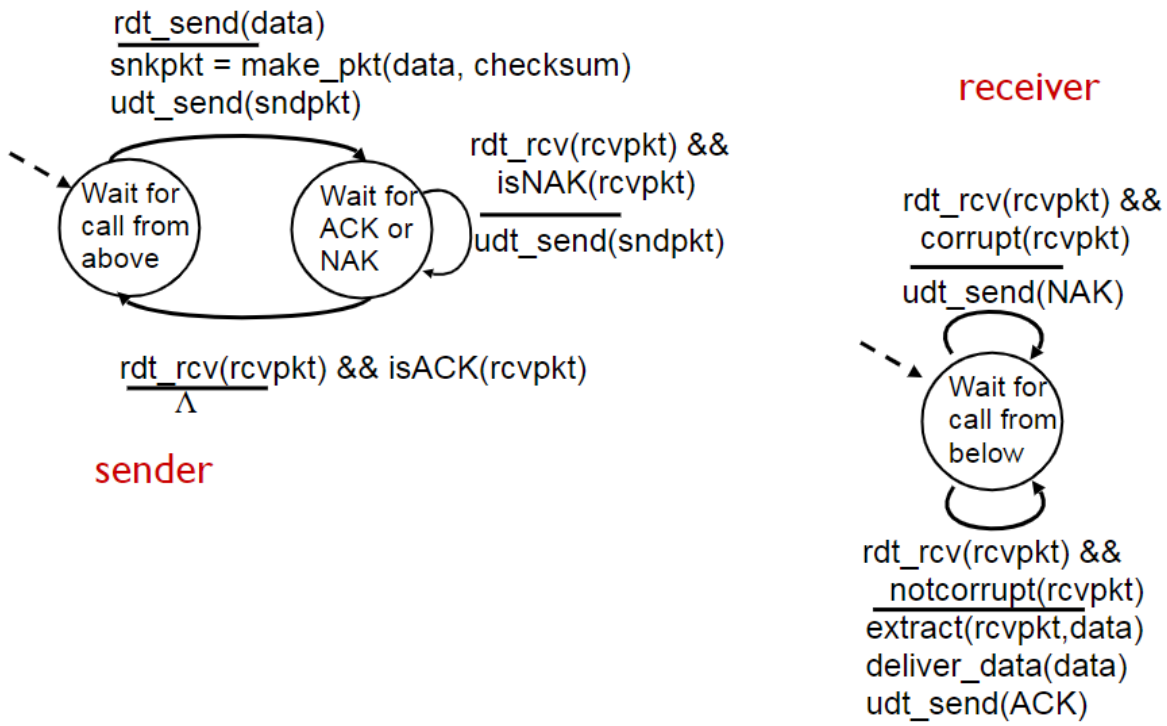
- Underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- separate FSMs for sender and receiver:
  - sender sends data into underlying channel
  - receiver reads data from underlying channel



rdt 2.0: channel with bit errors

- Underlying channel may flip bits in packet, which causes bit error
- Retransmit packets with errors
- How to recover from errors:
  - **checksum** to detect bit errors
  - **acknowledgements (ACKs)**: receiver explicitly tells sender that packet received OK
  - **negative acknowledgements (NAKs)**: receiver explicitly tells sender that packet had error
    - sender retransmits packet on receipt of NAK
- New mechanisms in rdt2.0 (beyond rdt1.0):
  - error detection
  - feedback: control messages (ACK, NAK) from receiver to sender

rdt 2.0: FSM specification

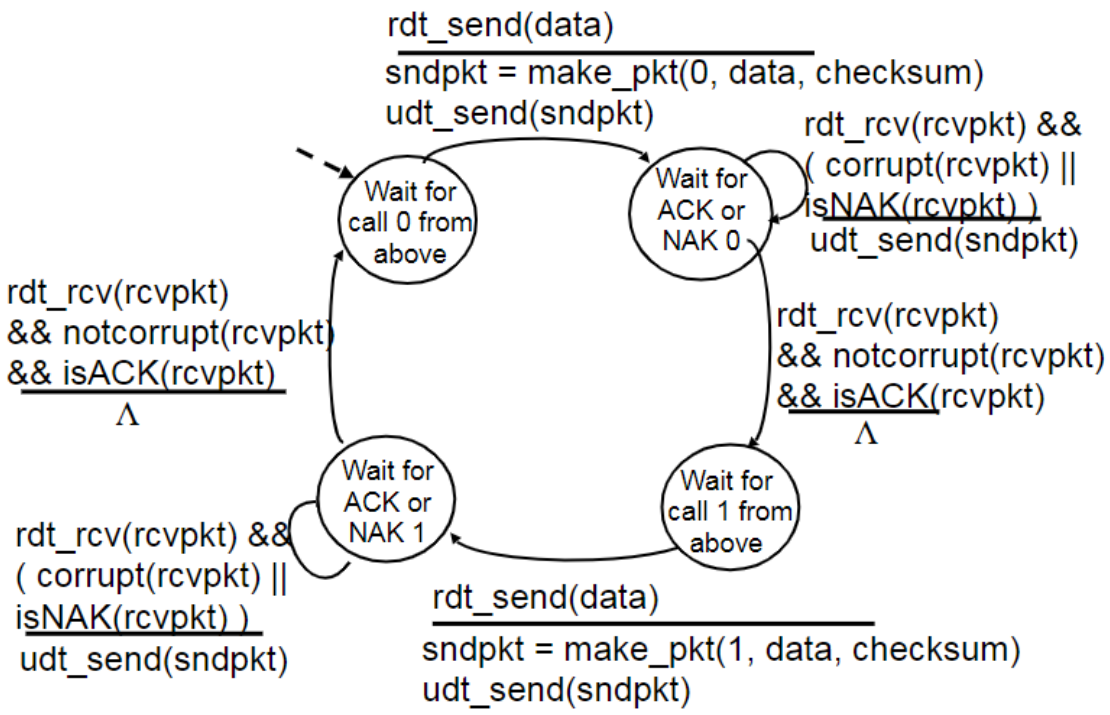


rdt 2.0: fatal flaw 致命缺陷

- What happened if ACK/NAK corrupted?
  - Sender will not know what happened at receiver
  - Sender will not be able to retransmit the corrupted packet since the ACK/NAK of that packet is corrupted, possible duplicate.
- Handling duplicates:
  - Sender retransmits current packet if ACK/NAK corrupted
  - Sender adds *sequence number* to each packet
  - Receiver discards duplicate packet

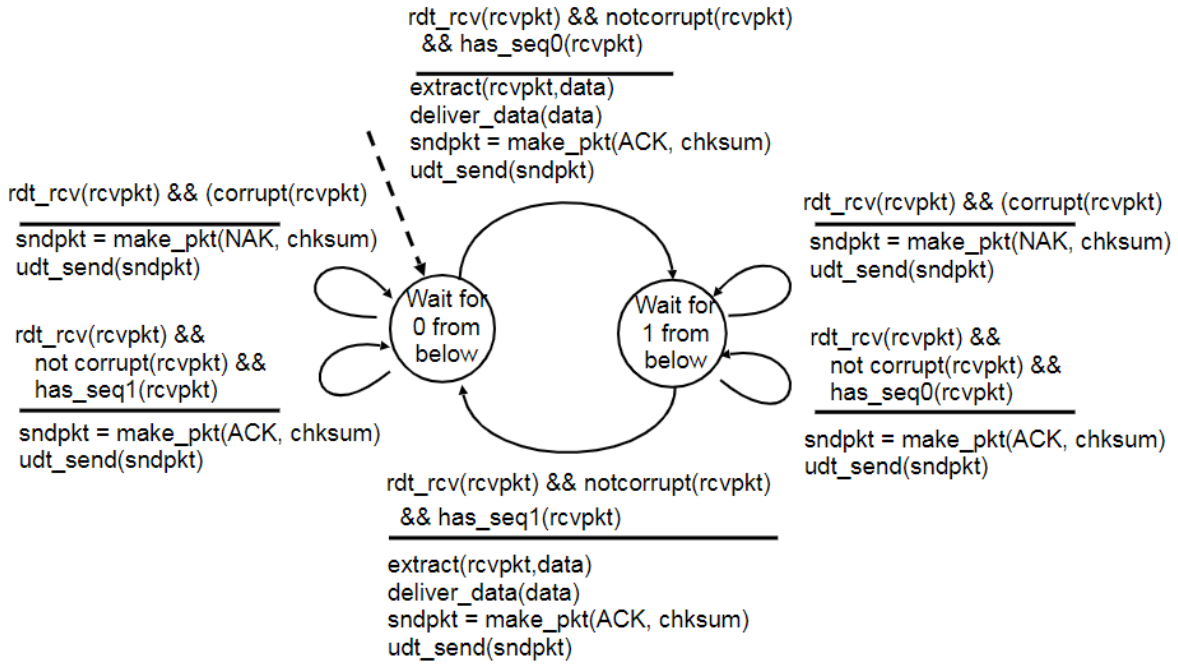
> Stop and wait:
>
> Sender sends one packet, then waits for receiver response.

rdt 2.1: sender, handles corrupted ACK/NAKs



1. Sender send packet that include the sequence (0), data, and checksum to the receiver -> next state
2. Sender wait for **ACK 0 or NAK 0** packet from receiver
    1. if the packet from receiver is corrupted, or received msg is NAK 0-> re-send the packet, remain state
    2. if the packet from receiver is not corrupted, and the packet is ACK 0 -> output call 1, next state
3. Sender send packet that include the sequence (1), data, and checksum to the receiver -> next state
4. Sender wait for **ACK 1 or NAK 1** packet from receiver
    1. if the packet from receiver is corrupted, or received msg is NAK 1-> re-send the packet, remain state
    2. if the packet from receiver is not corrupted, and the packet is ACK 1
        1. Sender send packet that include the sequence (0), data, and checksum to the receiver -> next state (go to state 1)

rdt 2.1: receiver, handles corrupted ACK/NAKs



,

1. Wait for the packet from sender
    1. if the packet is corrupted, then send NAK and the checksum of received packet -> remain state
    2. if the packet is not corrupted, and the sequence is 1, then send **ACK 1** and checksum packet -> remain state
    3. if the packet is not corrupted, and the sequence is 0, **then deliver data to upper layer**, send ACK and checksum packet to sender -> next state
2. Wait for the packet from sender
    1. if the packet is corrupted, then send NAK and the checksum of received packet -> remain state
    2. if the packet is not corrupted, and the sequence is 0, then send **ACK 0** and checksum packet -> remain state
    3. if the packet is not corrupted, and the sequence is 1, **then deliver data to upper layer**, send ACK and checksum packet to sender -> next state

rdt 2.1 Summary

可以从上面两副图看到packet被加上了序号0和1。这样的目的是**避免receiver接收重复的数据包。**

一旦receiver确认收到的序号0数据包是无损的，便会发送ACK 0数据包并将自身状态跳到等待序号1。若此时发送的ACK 0数据包发生比特翻转（混淆），则sender会重复发一个序号0的数据包。此时由于receiver处于等待序号1数据包，接收到的序号0数据包会被忽略。

sender:

- seq # added to packet
- two seq # (0,1) will suffice
- must check if received ACK/NAK corrupted

receiver:

- must check if received packet is duplicate
- note: receiver cannot know if ACK/NAK is corrupted

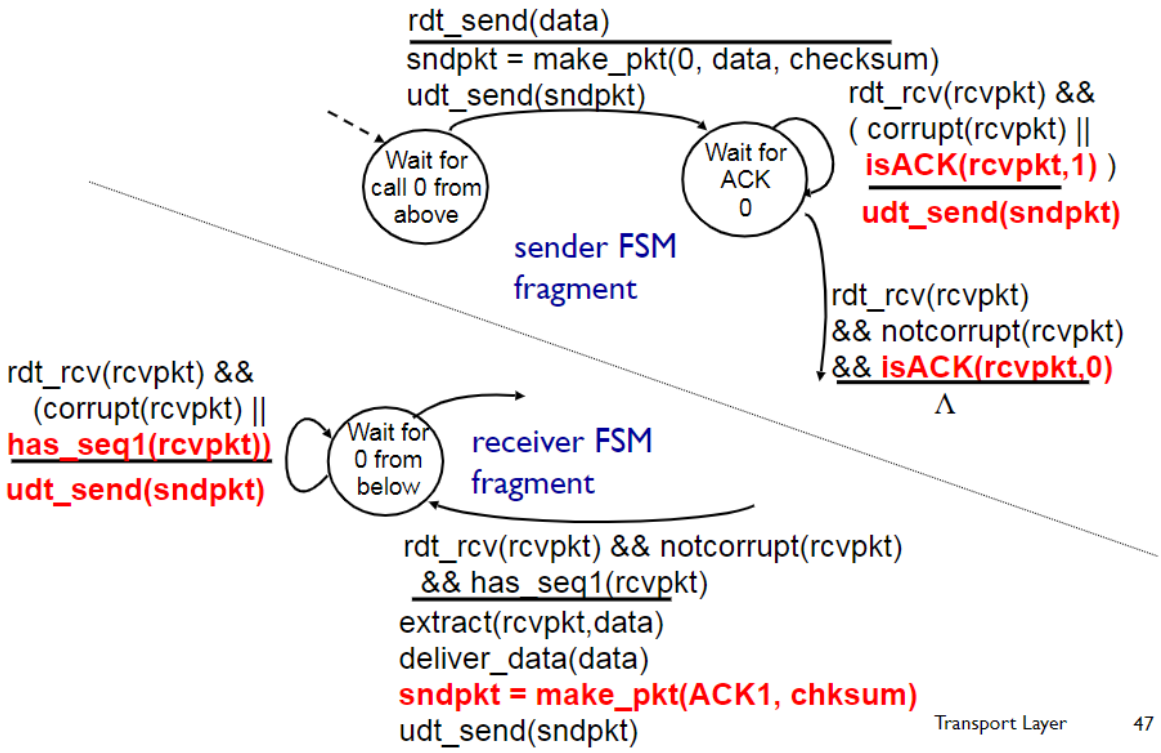
channel with bit errors

- new mechanisms in rdt2.1 (beyond rdt2.0):

- deal with corrupted ACK/NAK by retransmission
- sequence numbers to detect duplicates

rdt 2.2: a NAK-free protocol

- Same functionality as rdt2.1, using ACKs only
- Instead of NAK, receiver sends ACK for last packet received OK
  - receiver must explicitly include sequence number of packet being ACKed
- Duplicate ACK at sender results in same action as NAK: retransmit current packet



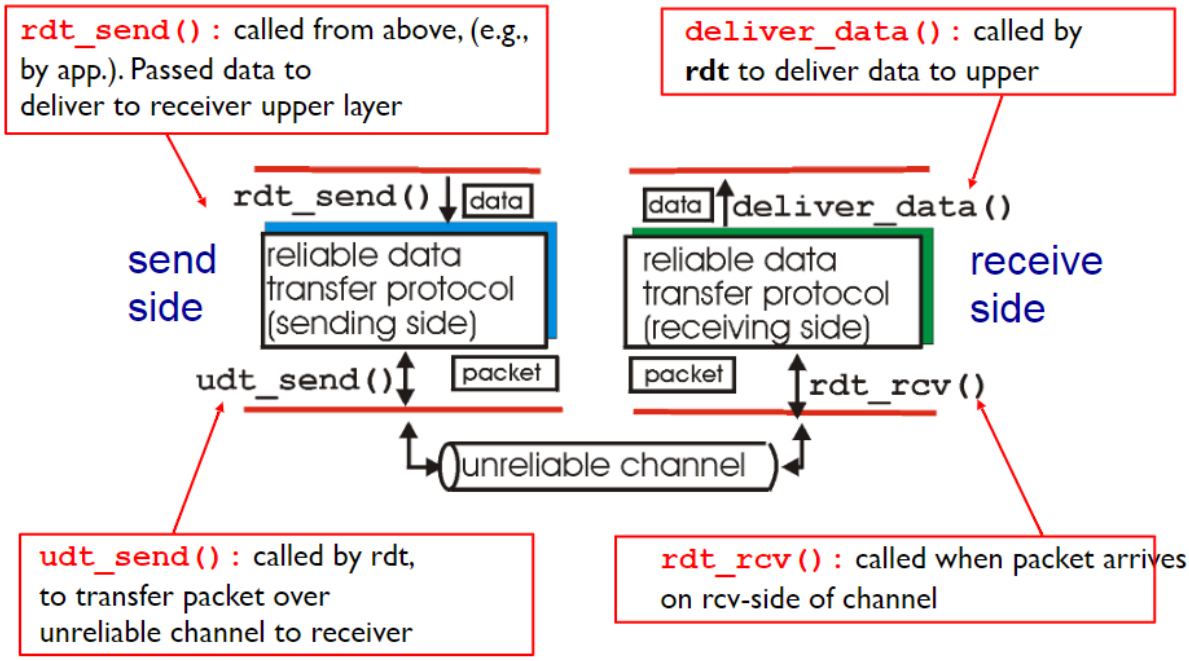
若receiver收到了受损数据，则不发送当前数据的NAK，而是发送上一个无损数据的ACK，以达到同样的效果。

sender收到上一个数据的两次ACK，便能知道当前的数据未发送成功

rdt 3.0: channels with errors and loss

New assumption:

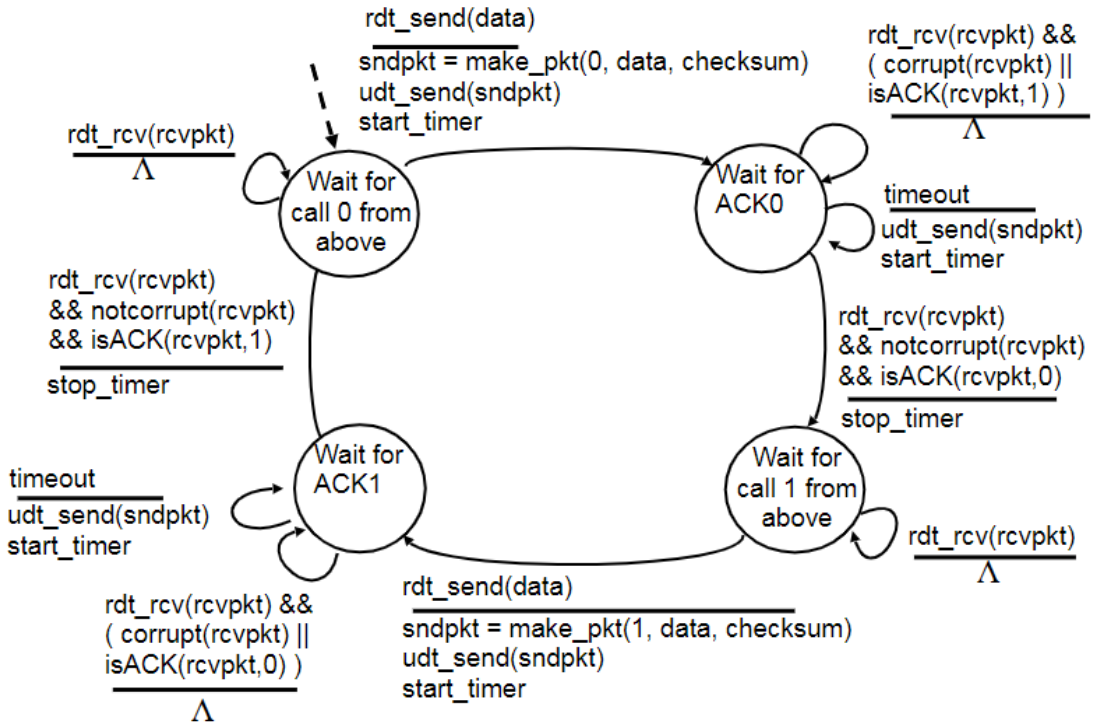
- **underlying channel** can also loss packets (data, ACKs)
  - checksum, sequence number, ACKs, retransmission will help, but not enough

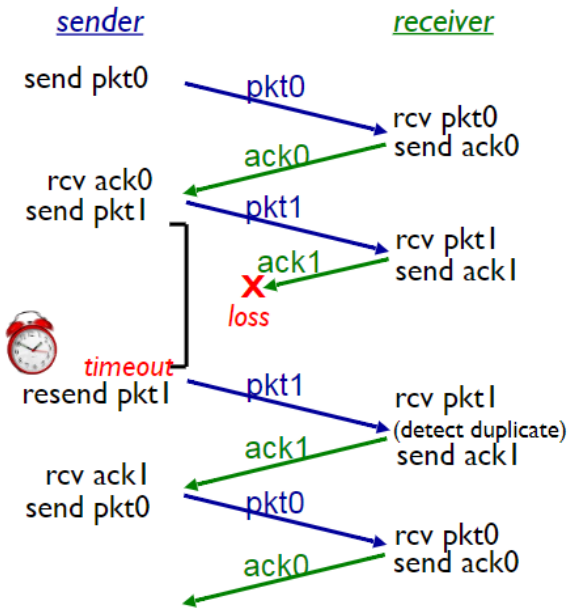


Approach: sender waits 'reasonable' amount of time for ACK

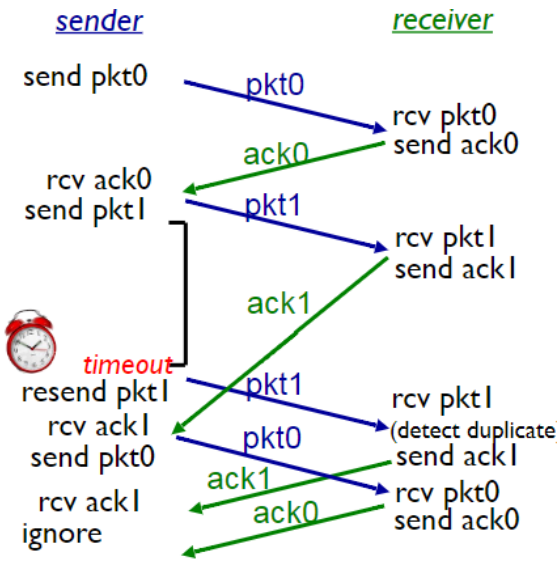
- retransmits if no ACK received in this time
- if packet (or ACK) just delayed (not lost)
  - retransmission will be duplicate, but sequence number is already handled this (in rdt 2.1)
  - receiver must specify sequence number of packet being ACKed
- requires countdown timer

rdt3.0 sender



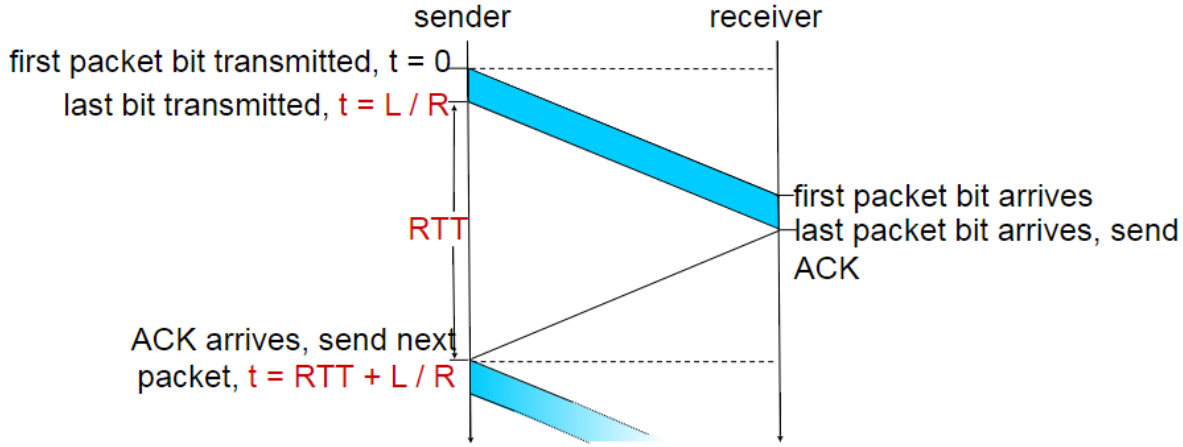


(c) ACK loss



(d) premature timeout/ delayed ACK

rdt3.0: stop-and-wait operation



L = packet size

R = transmission rate

RTT = Round Trip Time

rdt3.0: Performance

- rdt3.0 is correct, but not enough in performance
- For example: for 1 Gbps link, 15 ms propagation delay, sending 8000 bit packet:
  - $D_{trans} = \frac{L}{R} = \frac{8000\ bits}{10^9\ bits/sec} = 0.008\ ms$
  - U_{sender}: utilisation–fraction of time sender busy sending
    $U_{sender} = \frac{L/R}{RTT+L/R} = \frac{0.008}{30.008} = 0.00027$, where RTT should be twice of the propagation delay
  - if RTT = 30 ms, 1 KB packet every 30 ms: 33KB/sec throughput over 1 Gbps link.

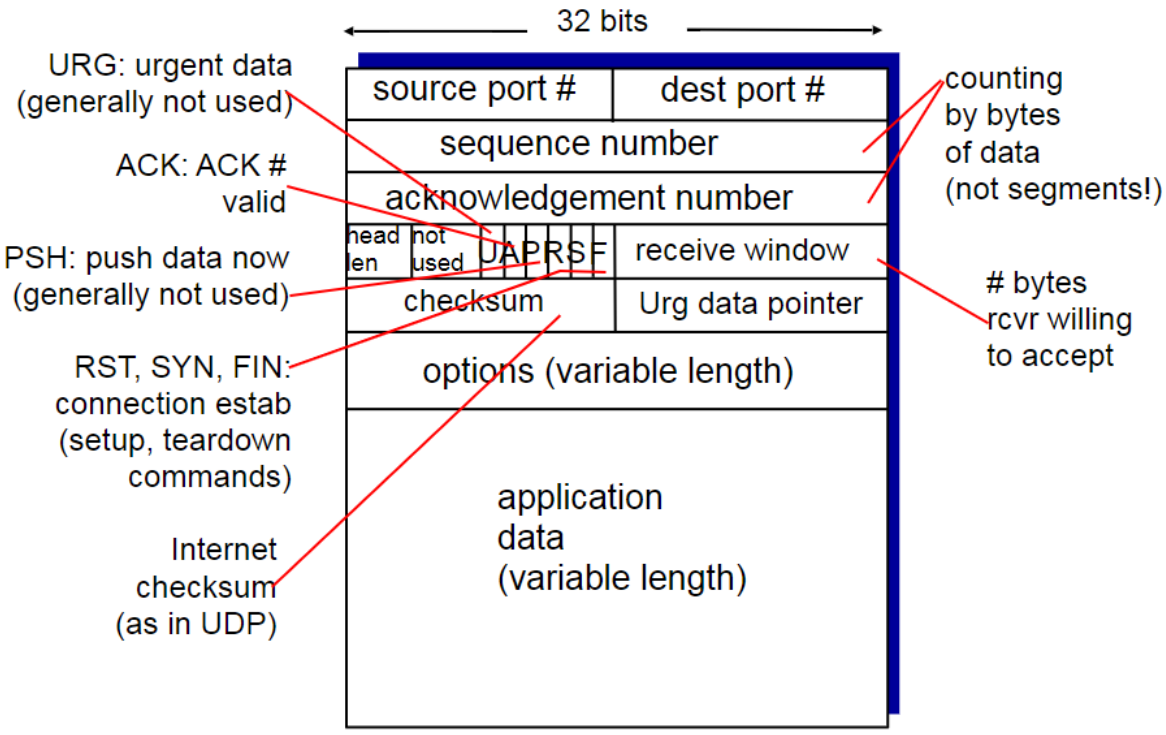
Pipelined protocols

3.5 Connection-oriented transport: TCP

TCP Overview

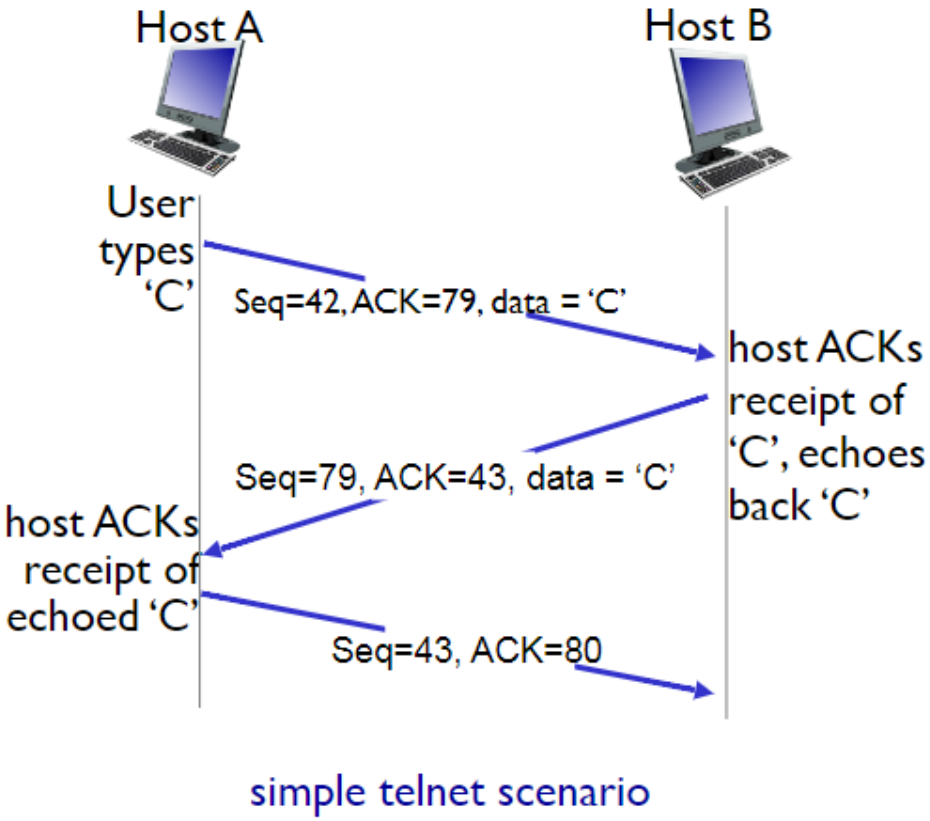
- Point-to-point
  - One sender, one receiver
- Reliable, in-order byte stream
  - no “message boundaries”
  - Sent as segments, delivered to application as byte stream
    - MSS: maximum segment size
- Pipelined
  - TCP congestion and flow control set window size
- Full duplex data (全双工数据)
  - bi-directional data flow in same connection
- Connection-oriented
  - handshaking (exchange of control messages) initialise sender and receiver state before data exchange.
- Flow controlled:
  - sender will not overwhelm (强制) receiver

TCP segment structure



TCP sequence numbers, ACKs

- Sequence numbers:
  - byte stream “number” of first byte in segment’s data
- Acknowledgements (ACK):
  - sequence number of next byte expected from other side
  - cumulative ACK



simple telnet scenario

TCP round trip time (RTT), timeout

- Timeout interval: Estimated RTT plus “safety margin”
  - large variation in estimated RTT cause larger safety margin

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

- Estimated RTT = EWMA (Exponential Weighted Moving Average) of RTT measurements
- Dev RTT = EWMA of deviation from estimated RTT

TCP reliable data transfer

- TCP creates rdt service on top of IP’s unreliable service
  - pipelined segments
  - cumulative ACKs
  - single retransmission timer
- **Retransmission** triggered by:
  - timeout events
  - duplicate ACKs

TCP sender events

event: data received from application above:

- create segment with sequence number
- sequence number is byte-stream number of first data byte in segment
- start timer if not ready running
  - think of timer as for oldest unacked segment
  - expiration interval: `TimeOutInternal`

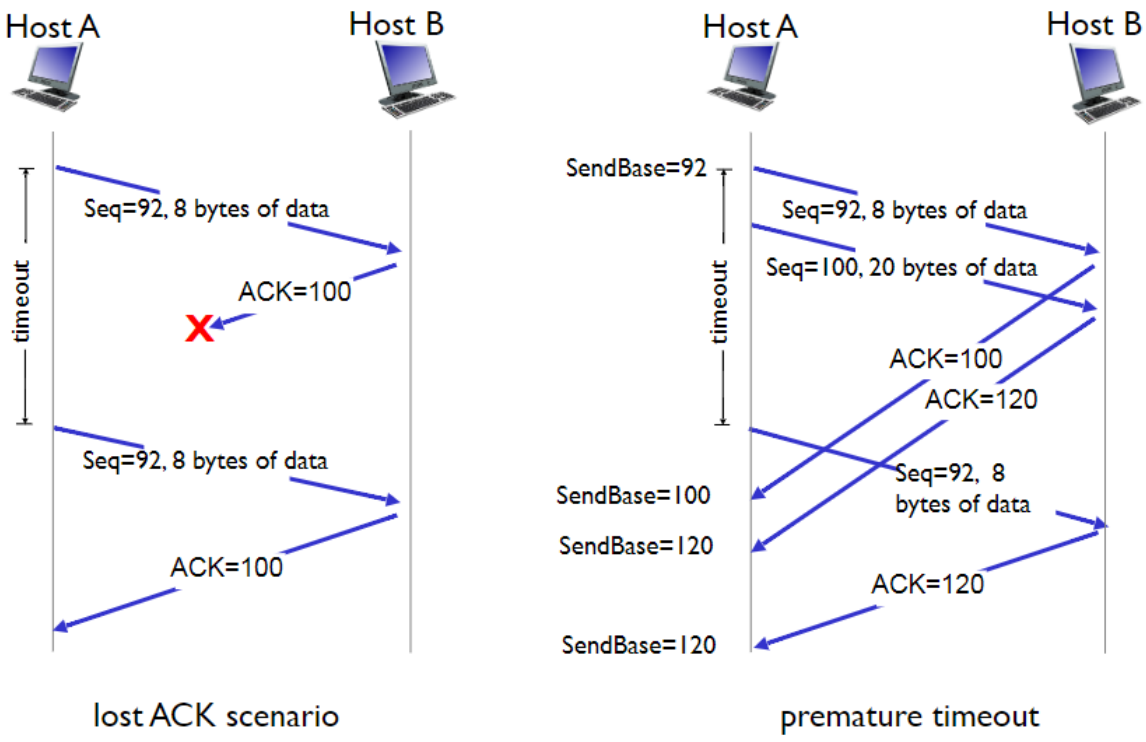
event: timeout:

- retransmit segment that caused timeout
- restart timer

event: ACK received:

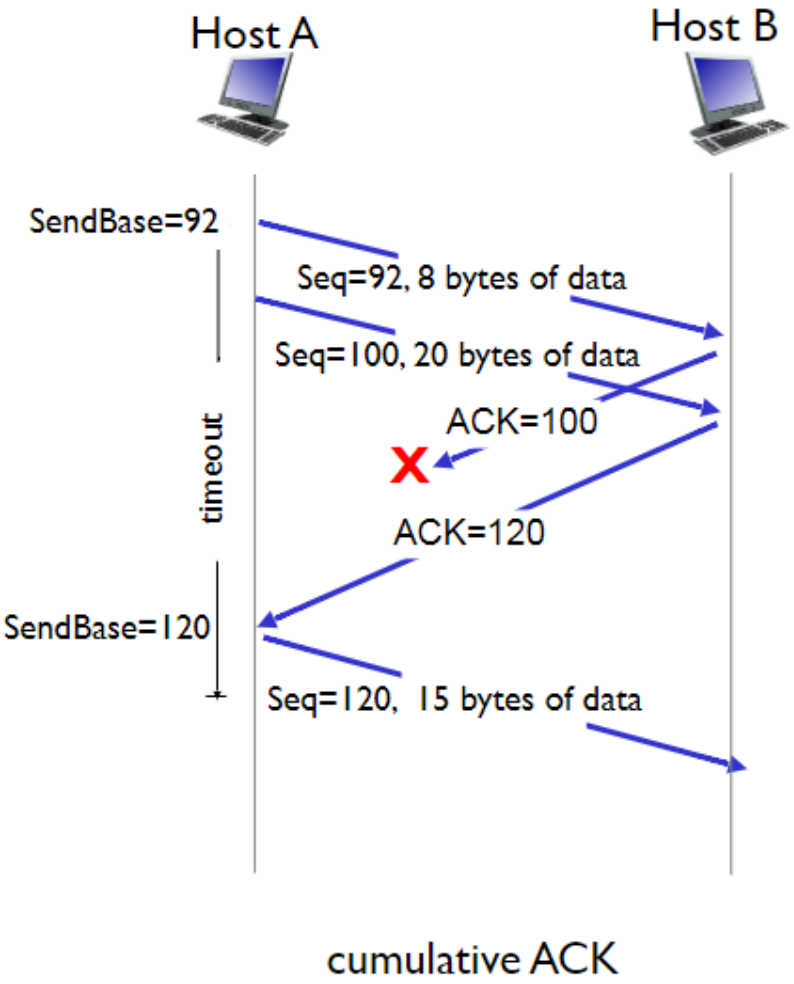
- if ACK acknowledges previously unacked segments
  - update what is know to be ACKed
  - start timer if there are still unacked segments

Retransmission scenarios



左图为packet loss的情况，即A不会收到ACK，那么在timeout之后重新发送seq 92， 若A在timeout内收到Ack 100即视为传送到。B在收到重复的seq 92时重复发送ack 100。注：ACK 的数字可以被看做B已准备好接收从100（例）开始的数据包。seq的数字可以看作是数据包的起始比特位。

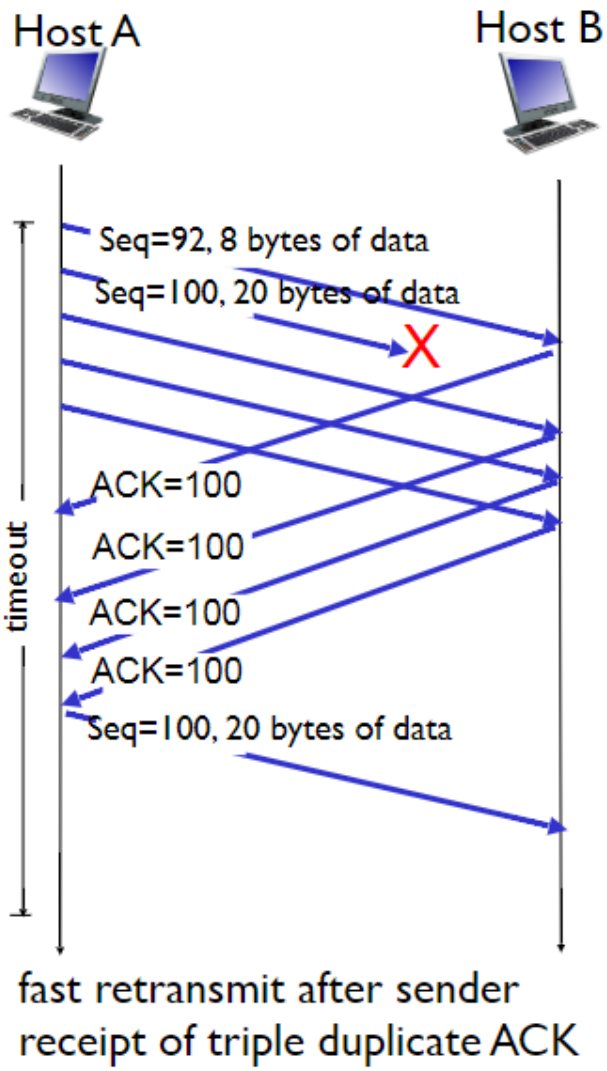
右图为在timeout外接收到ACK，和左图一样A还是会在timeout时重新发送seq 92， 但在接收到ack 100和120时始终会以**收到的ACK最高位**为下一次发送的起始位



对此图同理，若ACK 100丢包但ACK 120被A接收，那么A仍然会将下一个数据包起始位定在120

TCP fast retransmit

- timeout period often relatively long
  - long delay before resending lost packet
- detect lost segments via duplicate ACKs
  - sender often sends many segments back-to-back
  - if segment lost, there will likely be many duplicate ACKs.
- **if sender receives 3 ACKs for same data ("triple duplicate ACKs")**
  - resend unacked segment with **smallest sequence number**
    - likely that unacked segment lost, so do not wait for the timeout

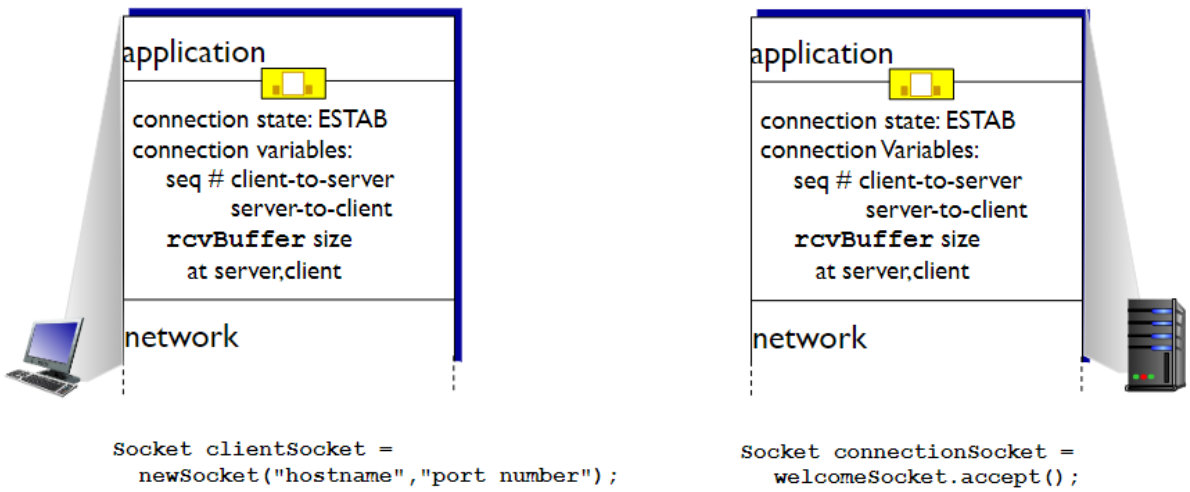


在连续的数据包发送中，若发送方接收到三次对相同seq的重复ACK，即表明该seq起始的数据包丢失，发送方立即重新发送该数据包

Connection management - TCP握手

Before exchanging data, sender or receiver need to “handshake”

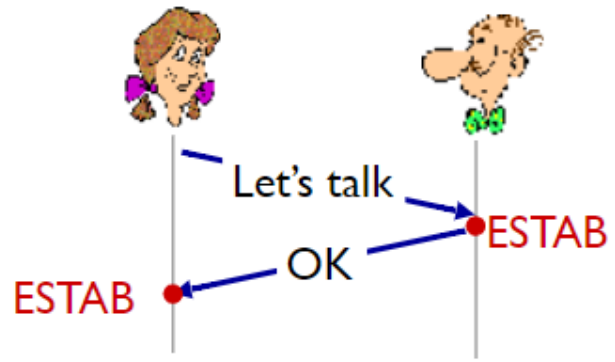
- Agree to establish connection (each knowing the other willing to establish connection)
- Agree on connection parameters

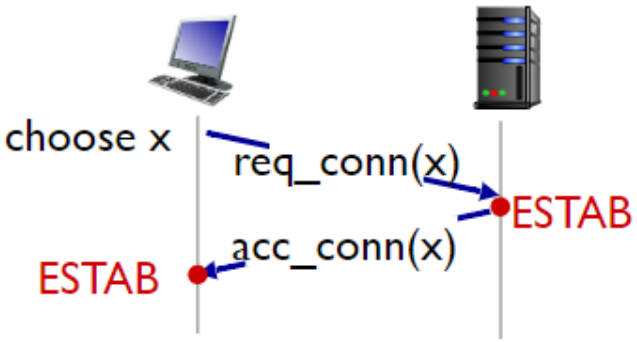


TCP 2-way handshake:

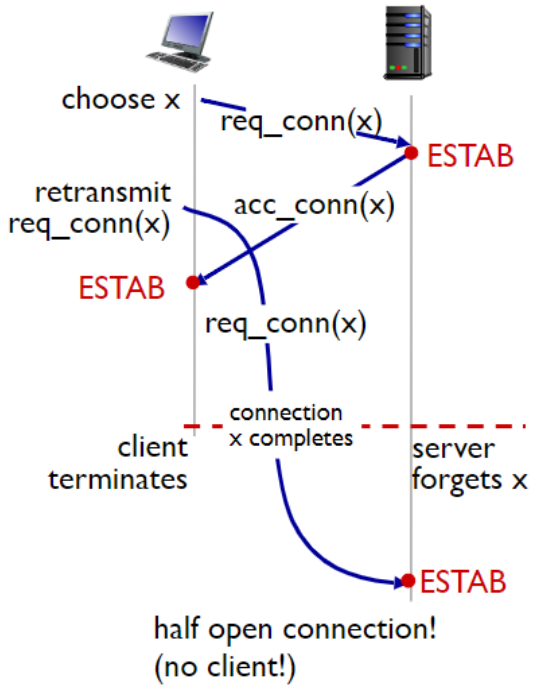
Advantage:

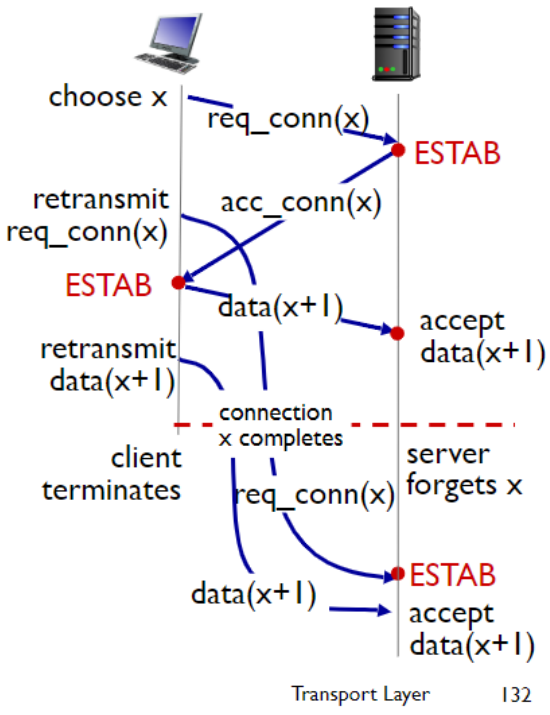
- variable delays
- retransmitted messages (example: req_conn(x)) due to message loss
- message reordering
- can't see other side





Failure scenarios:

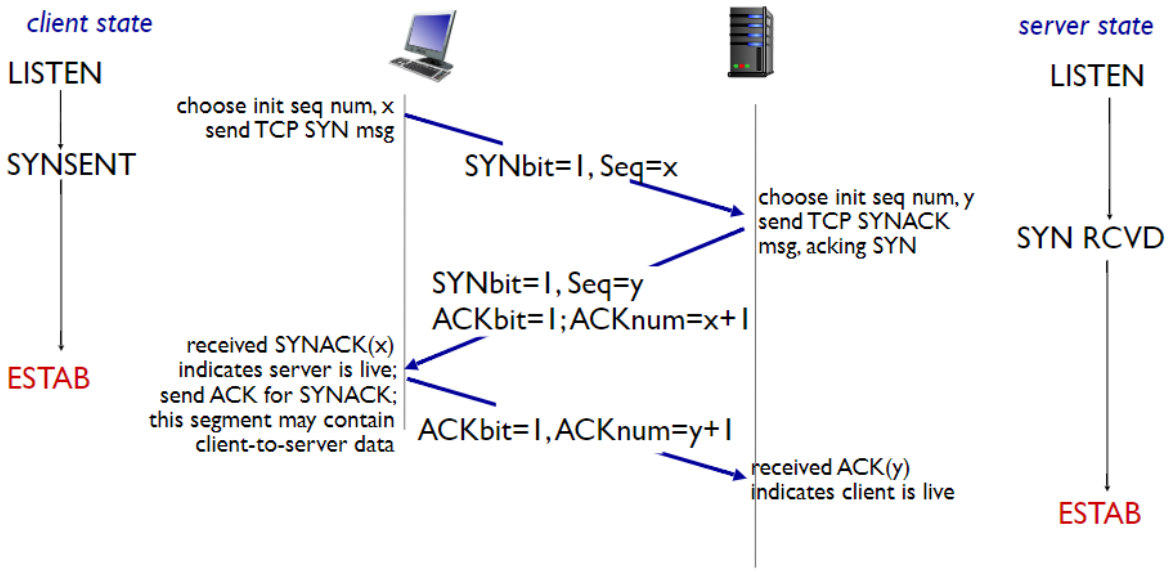




左图为半开放连接，即PC重传的握手数据包在关闭端口之后才送达，造成只有主机开放TCP端口

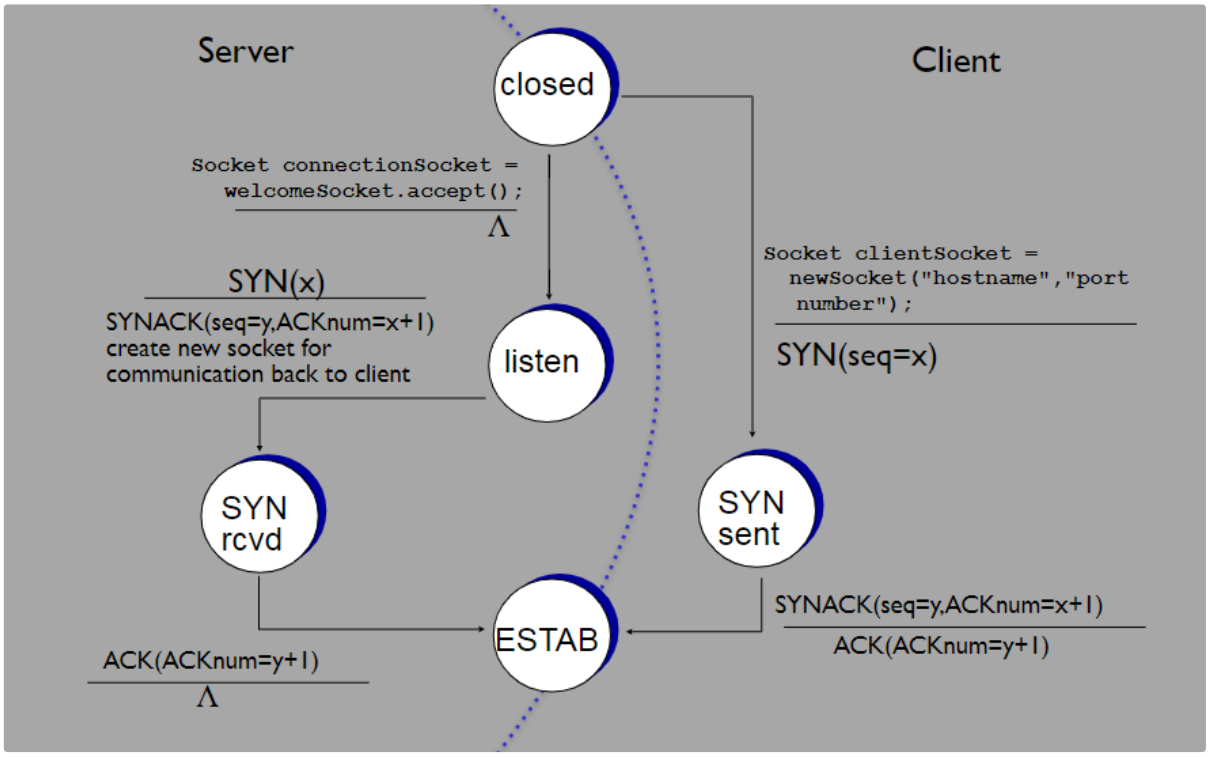
右图为重复传送，即PC重传的握手数据包和重传的data数据包均在关闭端口后送达->服务器：我收两次

TCP 3-way handshake



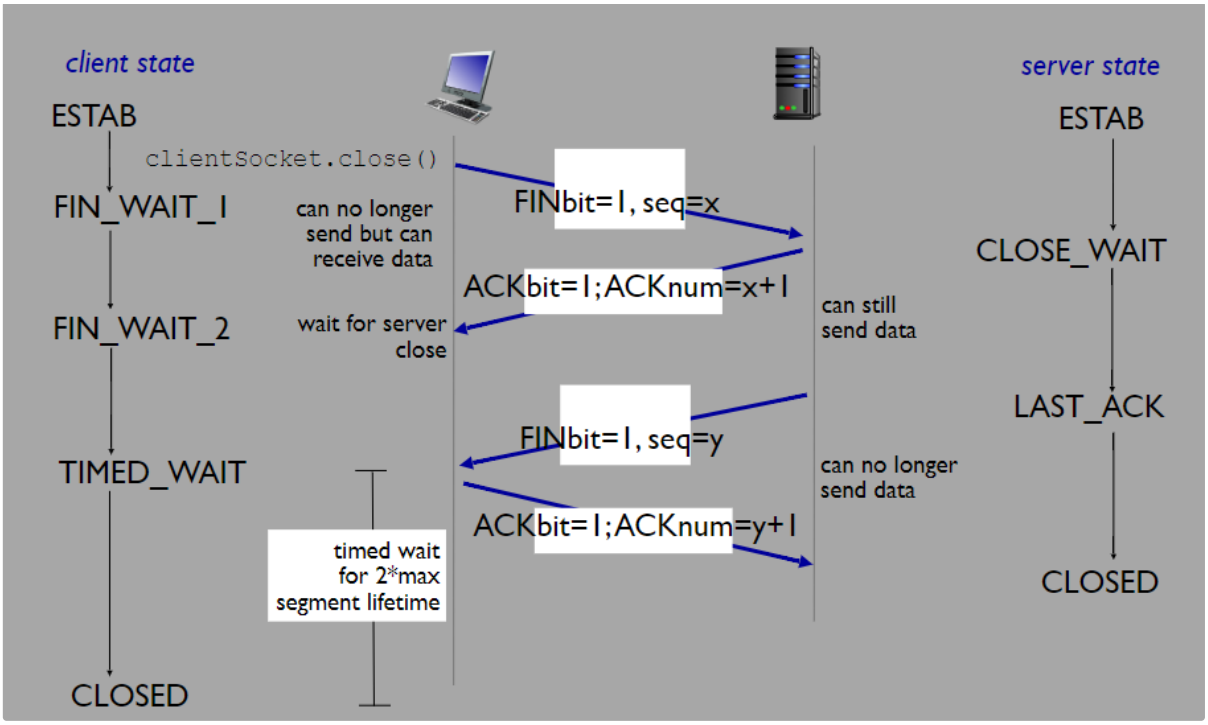
握手信号中增加了seq和ACK。只有在客户端和主机皆接收到了正确的ack之后才建立连接

FSM:



TCP closing a connection

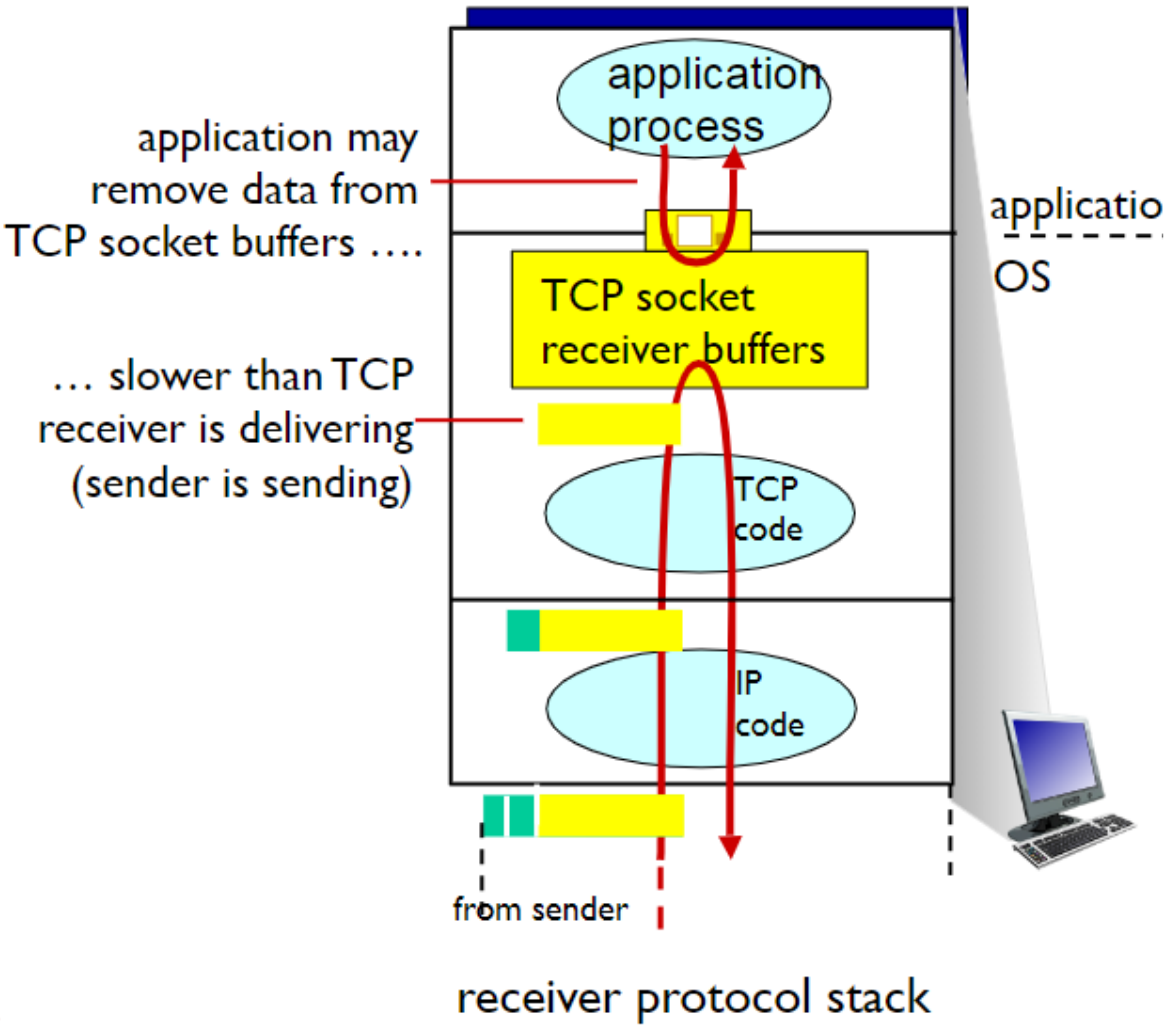
- client, server each close their side of connection
  - send TCP segment with `FINbit=1`
- respond to received `FIN` with `ACK`
  - on receiving `FIN`, `ACK` can be combined with own `FIN`
- simultaneous `FIN` exchanges can be handled



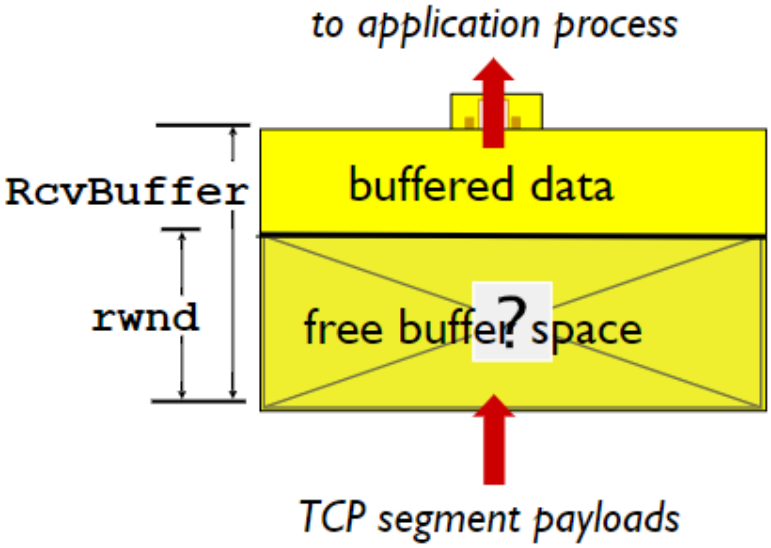
TCP flow control

Flow control:

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



- receiver "advertises" free buffer space by including `rwnd` value in TCP header of receiver-to-sender segments
- sender limits amount of unacked ("in-flight") data to receiver's `rwnd` value
- guarantees receive buffer will not overflow



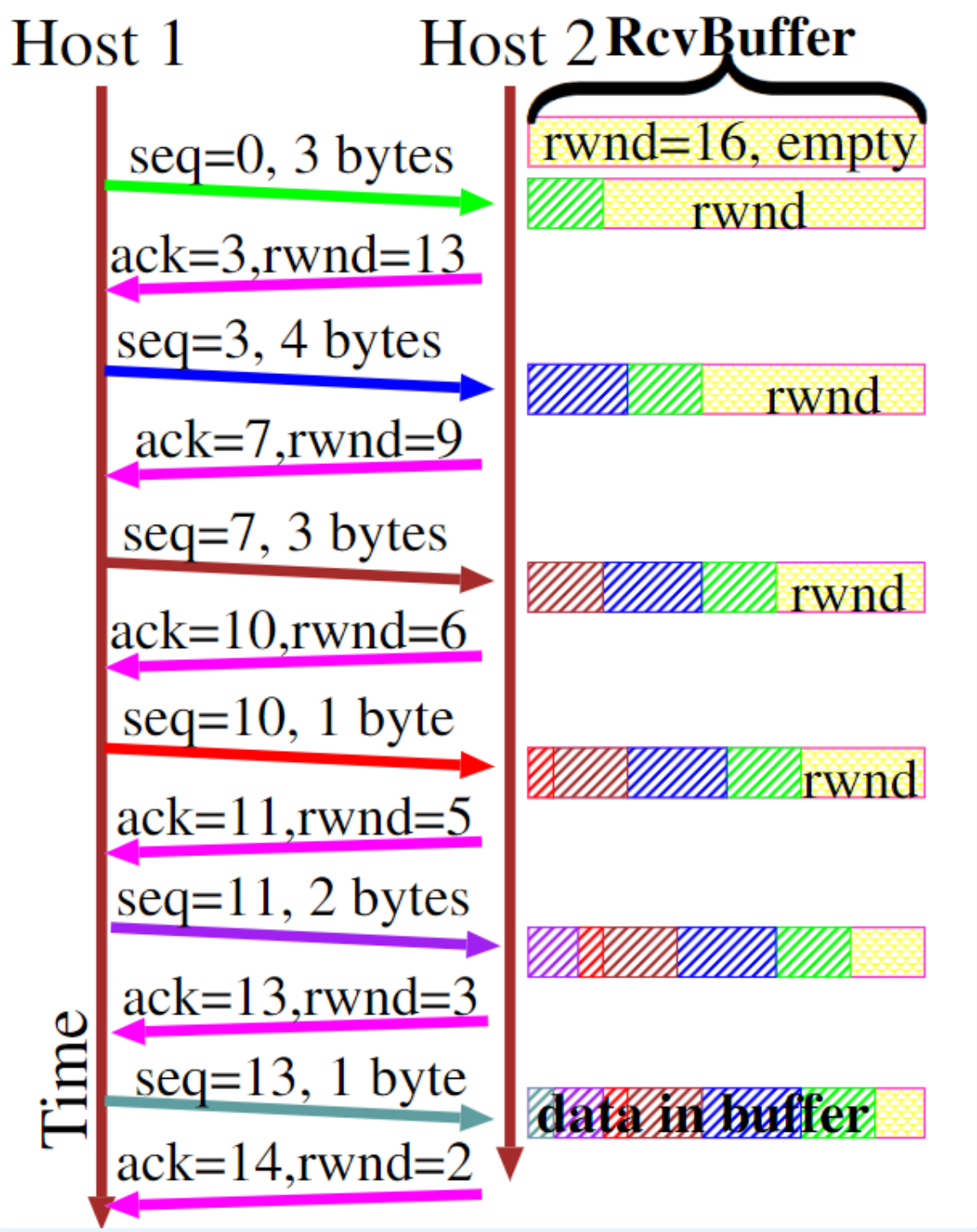
receiver-side buffering

Details below about `rwnd` is referenced from Data_Networking@UoM:

TCP flow control

Two major parameters to control the flow of message:

- TCP receive window "rwnd" to deal with the size of the bucket (The capacity of receiver)
- TCP congestion window "cwnd" to deal with the narrow part of the pipe(Transmission network)

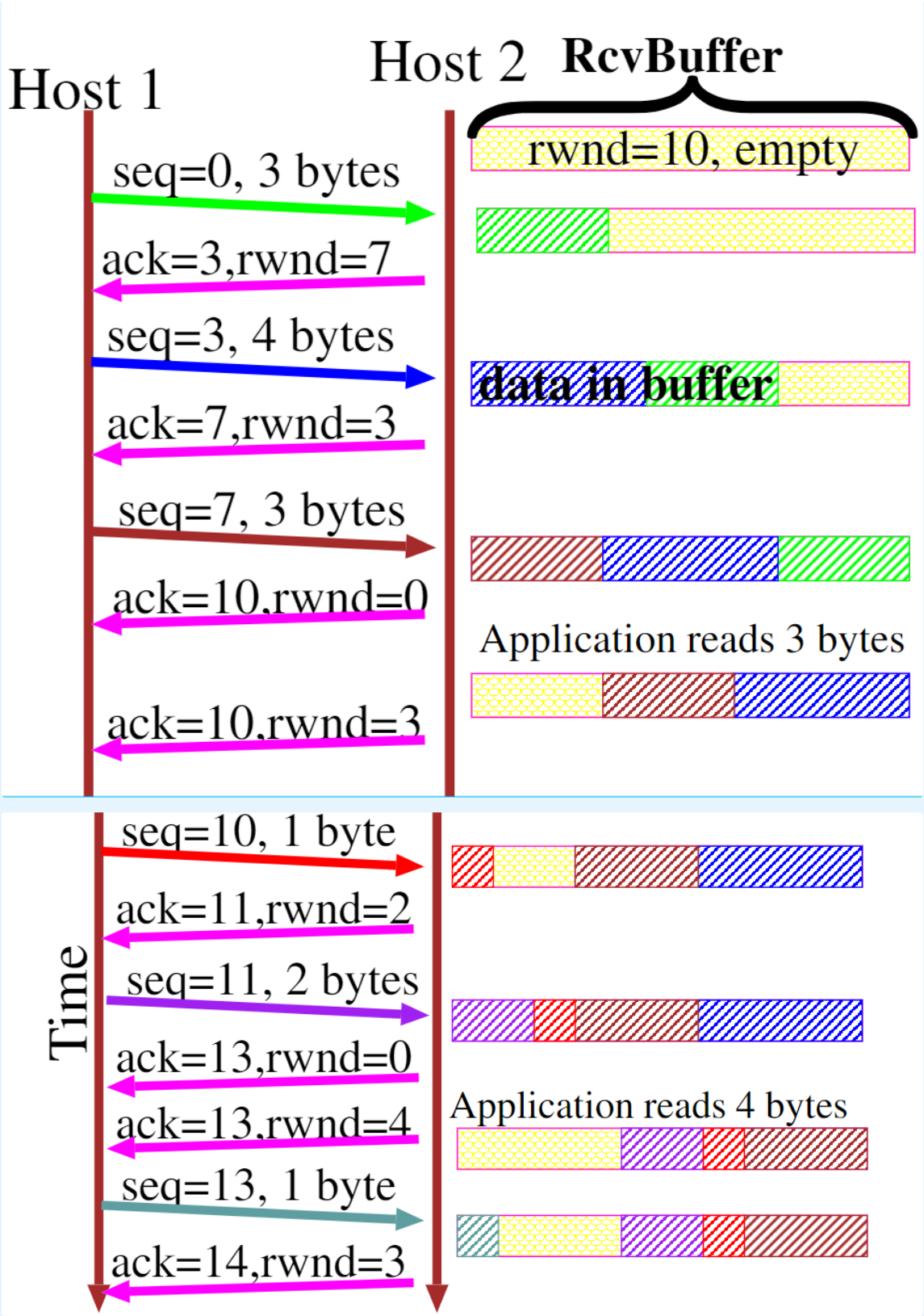


The maximum window size, rwnd, has been set to 16. So the receive buffer, RcvBuffer, is 16 bytes.

1. Host 1 send 3 bytes of data, SEQ=0
2. Host 2 send ACK=3, the receive buffer stores 3 bytes of data, the free space is 13 bytes, so send rwnd=13
3. ......

TCP receive window rwnd with RcvBuffer=10

当Buffer更小，不能承载多的数据时，TCP协议中对Buffer读写的程序



- Host 1 send data to Host 2, Host 2 return ACK and rwnd value to show the data received and free space of RcvBuffer
- When rwnd=0, Host 2 send rwnd=0 to Host 1 to determine that Host 2 cannot receive any more data, while Host 2 send ACK=10 to show that Host 1 should send segment 10 when there is a space to store it.
- When an application read 3 bytes from buffer, Host 2 send rwnd=3 to Host 1
- Host 1 send data

3.6 Principles of congestion control

- Congestion:
  - Informally: “too many sources sending too much data too fast for network to handle”
  - Different from flow control
  - Manifestations 拥塞的具体表现:
    - lost packets (buffer overflows at routers)
    - long delay (queueing in router buffers)

拥塞：**过多**的信息源向主机**过快地**发送了**过多**的数据

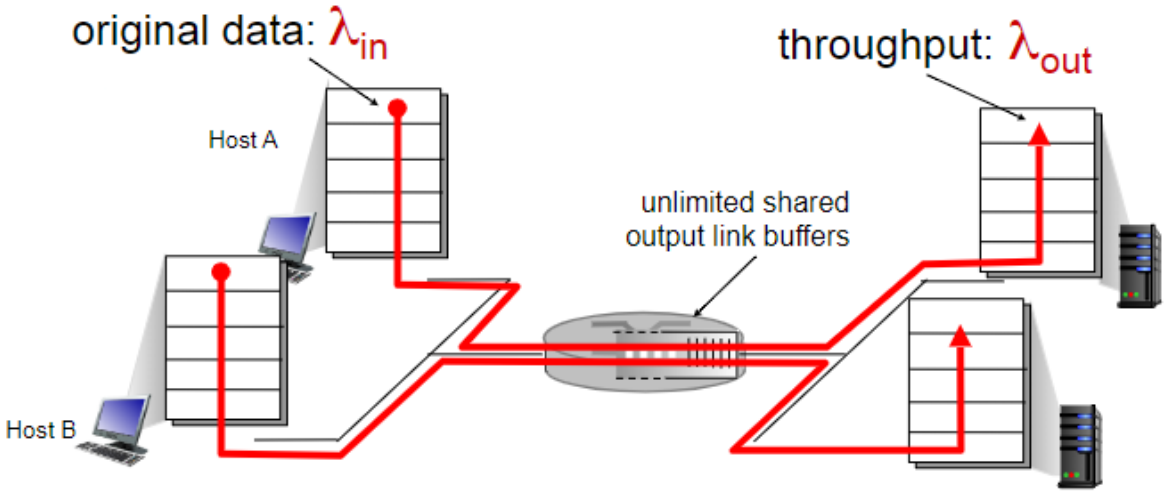
拥塞控制有别于流量控制

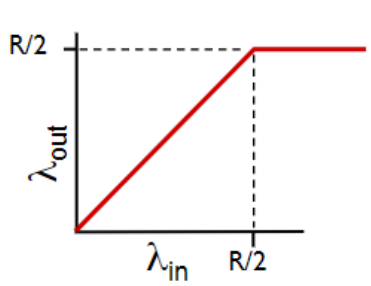
拥塞的具体表现为：

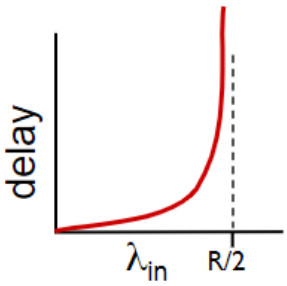
1. 丢包（路由器的缓存满了，丢掉最新的包）
2. 大延迟（数据包被积压在缓存中）

Causes/costs of congestion: scenario 1

- Two senders, two receivers
- One router, infinite buffers
- Output link capacity R
- No retransmission







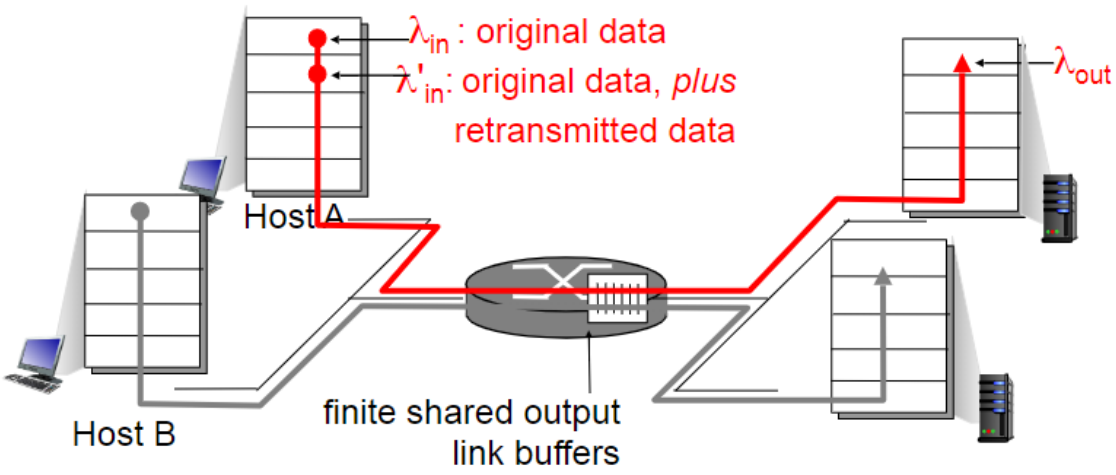
- maximum per-connection throughput: R/2
- large delays as arrival rate, λ_{in}, approaches capacity

在理想情况下，假设有两个主机和两个客户机在一个路由器上通讯。该路由器有带宽R， 不能重传。原始数据带宽被称为λ_{in}，接收数据带宽被称为λ_{out}。此时的两种结果：

1. 每一个客户机最大的分配带宽为：$\frac{1}{2}R$， 因为路由器的带宽上限为R
2. 同理，每一个主机最大的分配带宽也为$\frac{1}{2}R$
3. 延迟随着主机带宽的提高而指数级提高，最终在到达$\frac{1}{2}R$带宽时达到无限的延迟。

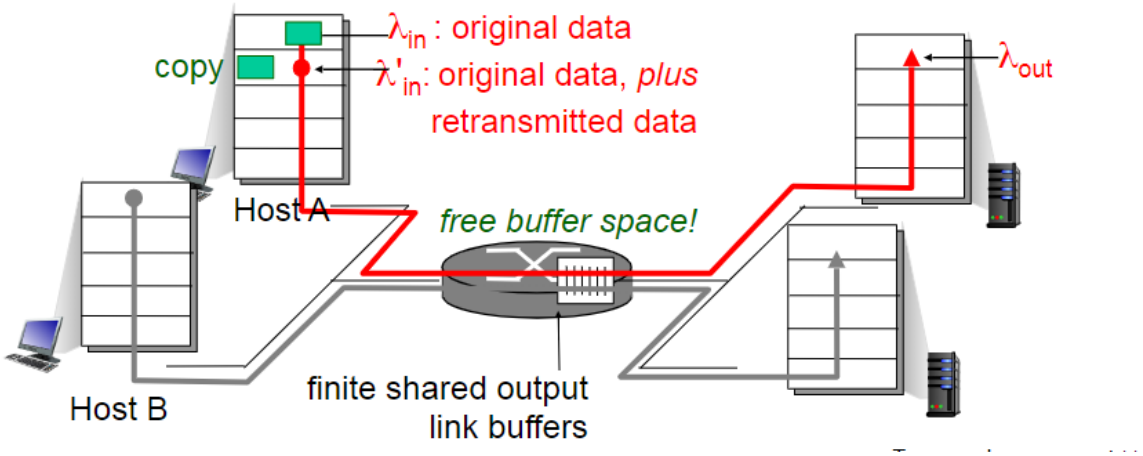
Causes/costs of congestion: scenario 2

- One router, *finite* buffers
- Sender will retransmission of timeout packet
  - Application-layer input = application layer output: $\lambda_{in} = \lambda_{out}$
  - Transport layer input will **BIGGER** than application layer input: $\lambda'_{in} \geq \lambda_{in}$
    Because the transport layer input includes **retransmitted data**



Causes/costs of congestion: scenario 2a

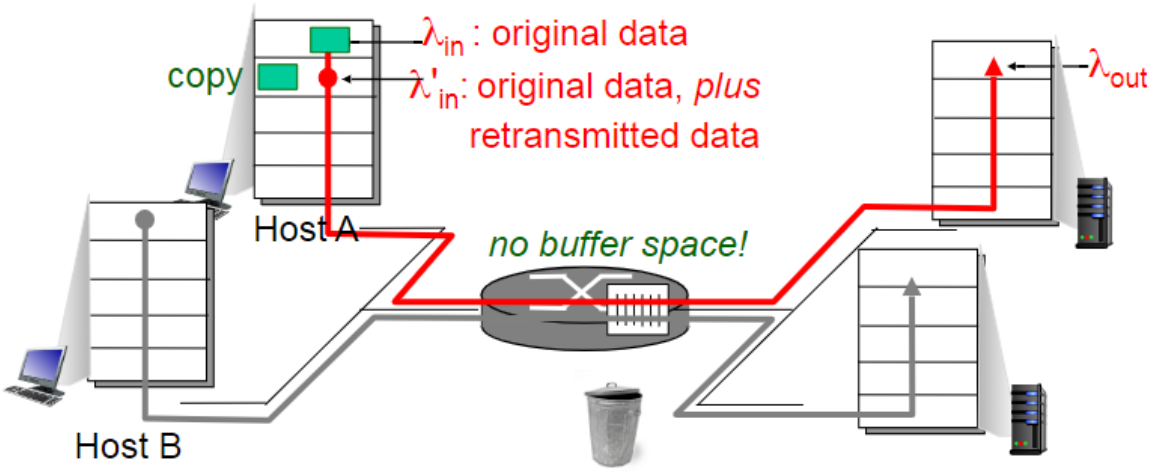
- Idealisation: perfect knowledge
  - sender sends only when router buffers available

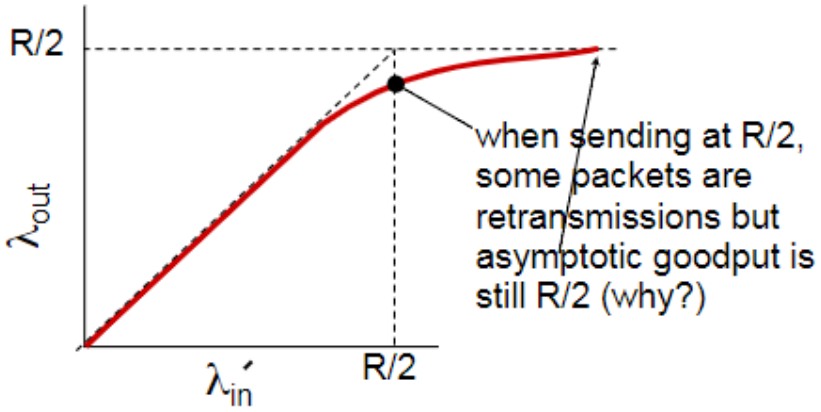


3.7 TCP congestion control

Causes/costs of congestion: scenario 2b

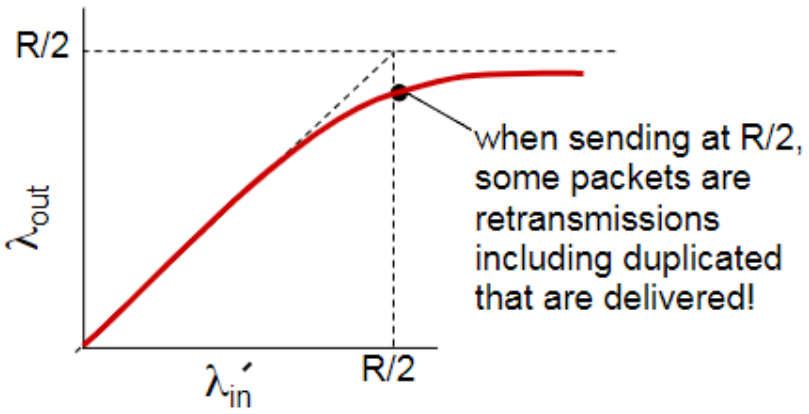
- Idealisation: **known loss**
  - packets can be lost, dropped at router due to full buffers
  - sender only resends if packet known to be lost





Causes/costs of congestion: scenario 2c

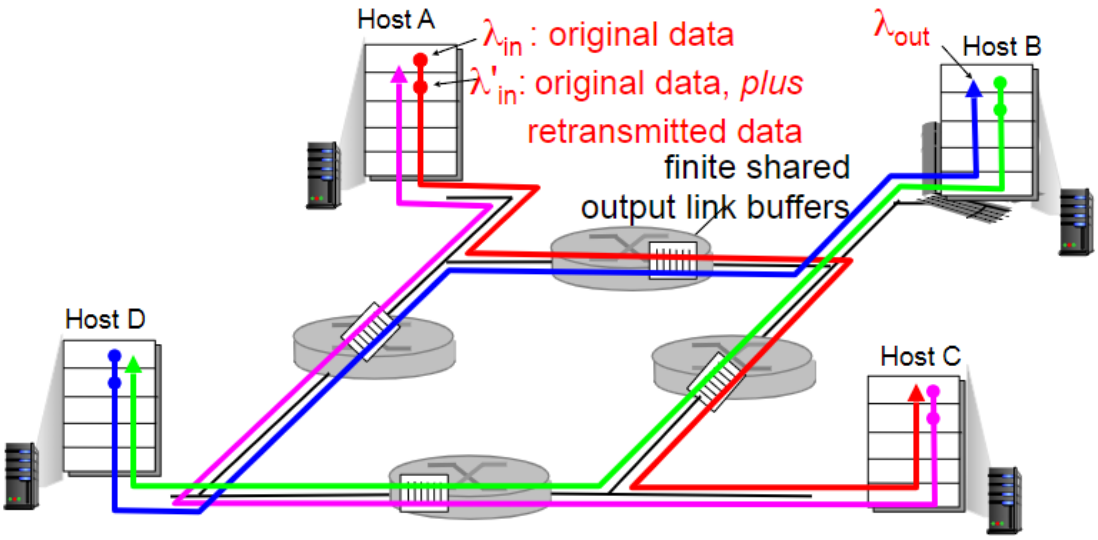
- Realistic: **duplicates**
  - packets can be lost, dropped at router due to the full buffers
  - sender times out prematurely, sending *two* copies, both of which are delivered



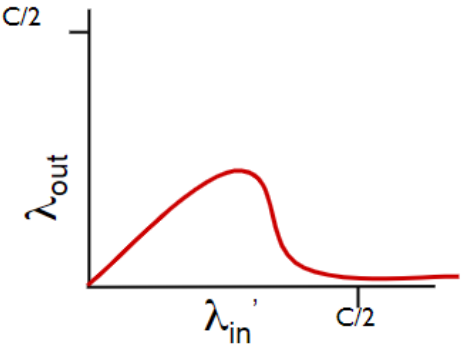
- "costs" of congestion
  - more work (retransmission) for given "goodput"
  - unneeded retransmissions: link carries multiple copies of packet
    - decreasing goodput

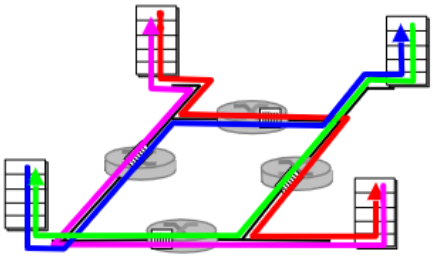
Causes/costs of congestion: scenario 3

- Four senders
- Multi-hop paths
- If timeout then retransmit



- If λ_{in} and λ'_{in} increases, then all arriving blue packets at upper queue are dropped, blue throughput -> 0

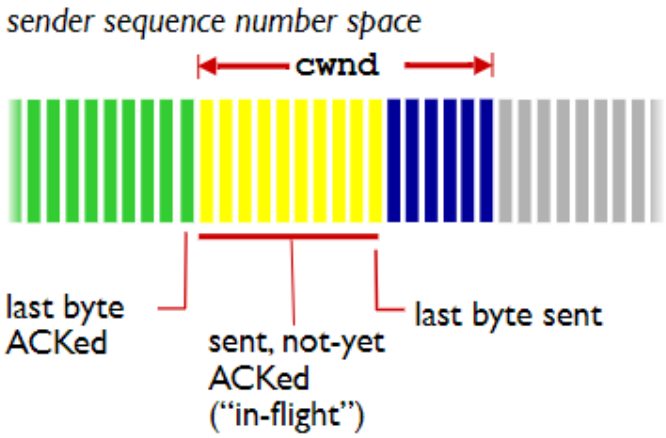




Another “cost” of congestion

- When packet dropped, any “upstream transmission” capacity used for that packet was wasted.

TCP congestion control: details



- Sender limits transmission at:
  $\text{LastByteSent} - \text{LastByteAcked} \leq cwnd$
- `cwnd` is dynamic, function of perceived network congestion

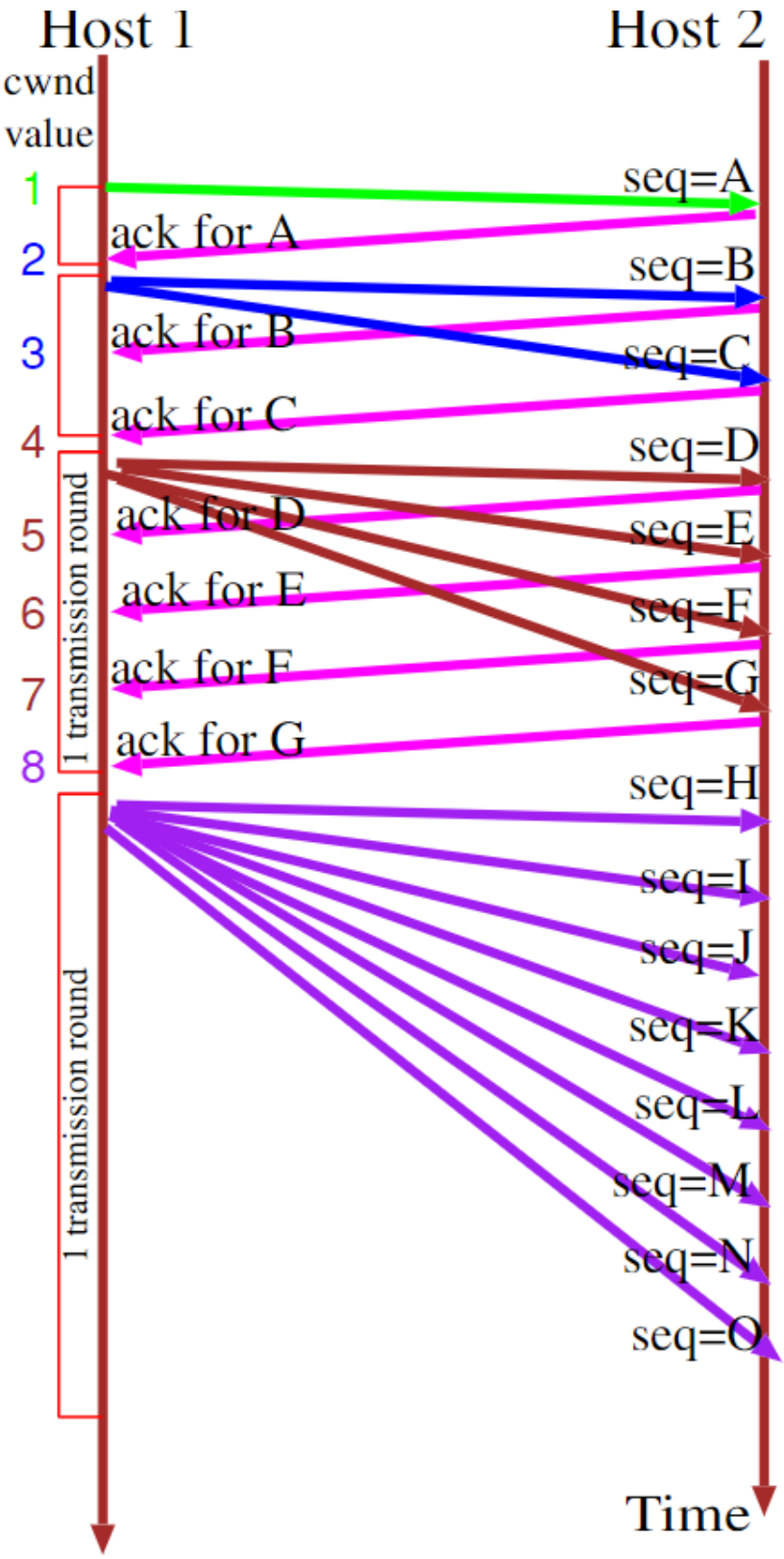
> TCP congestion window `cwnd` to deal with the narrow part of the pipe(Transmission network)

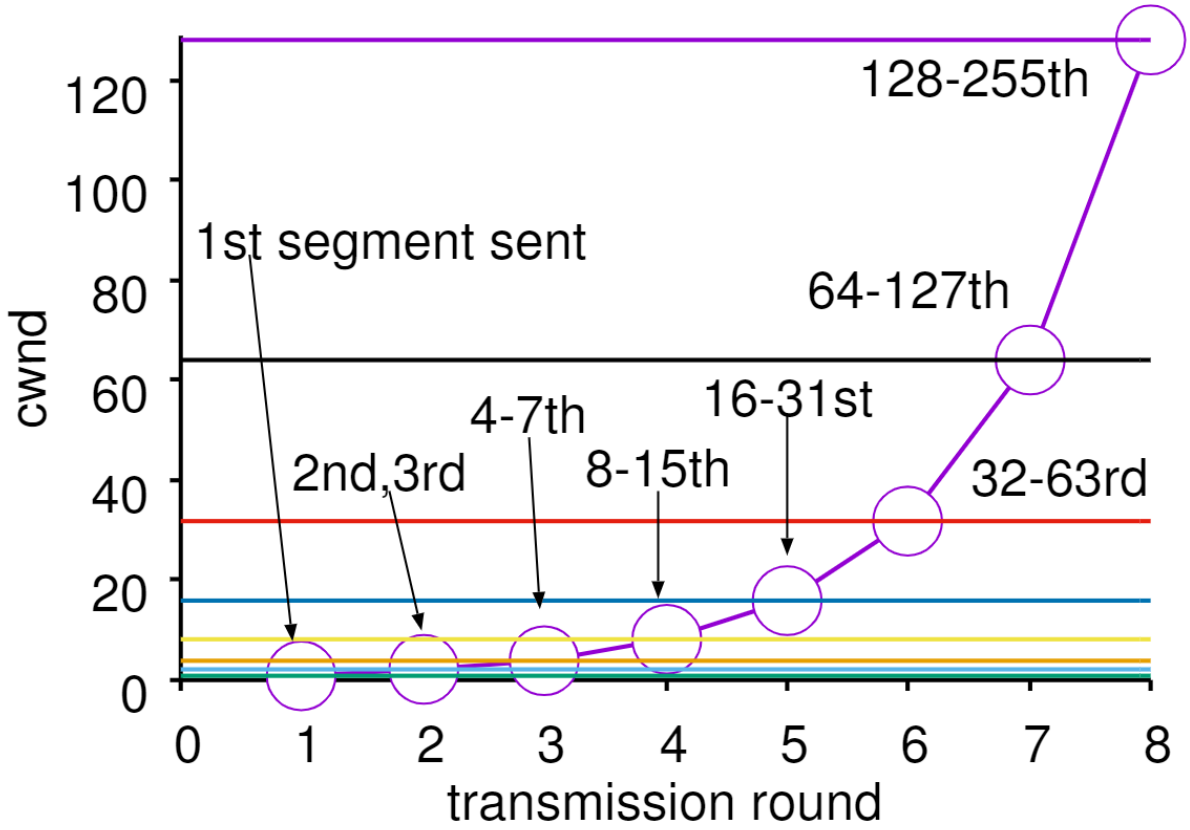
TCP sending rate:

- roughly: send `cwnd` bytes, wait RTT for ACKs, then send more bytes:
  $\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}}\ bytes/sec$

Slow-start: simplest congestion control method

- When connection begins, increase rate exponentially until first loss packet event
  - initially `cwnd = 1MSS`
  - double `cwnd` every RTT
  - done by incrementing `cwnd` for every ACK received


Host 1
Host 2
cwnd
value
1
2
3
4
5
6
7
8
seq=A
ack for A
seq=B
ack for B
seq=C
ack for C
seq=D
ack for D
seq=E
ack for E
seq=F
ack for F
seq=G
ack for G
seq=H
seq=I
seq=J
seq=K
seq=L
seq=M
seq=N
seq=O
1 transmission round
1 transmission round
Time



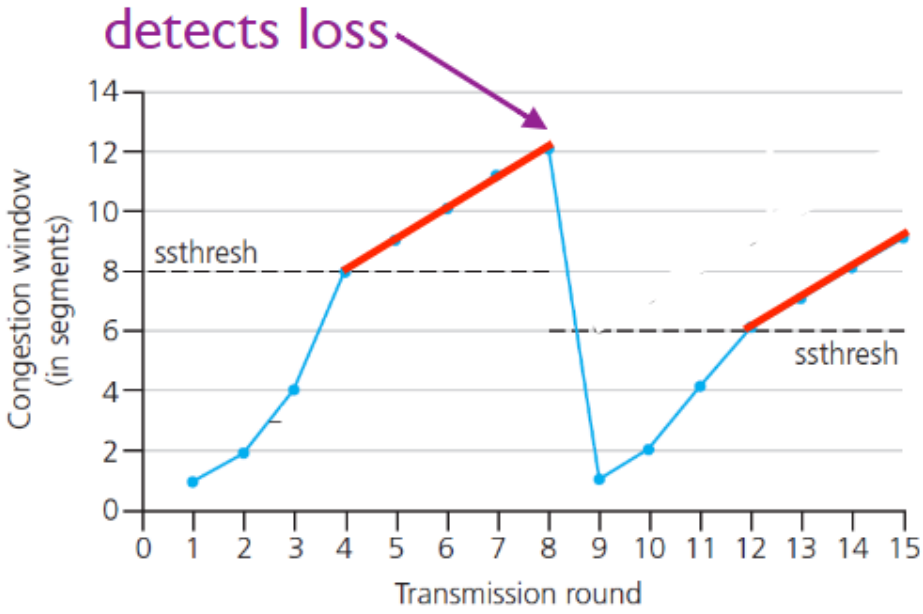
If loss packet:

- `cwnd = 1 MSS`

Congestion Avoidance and packet loss detection

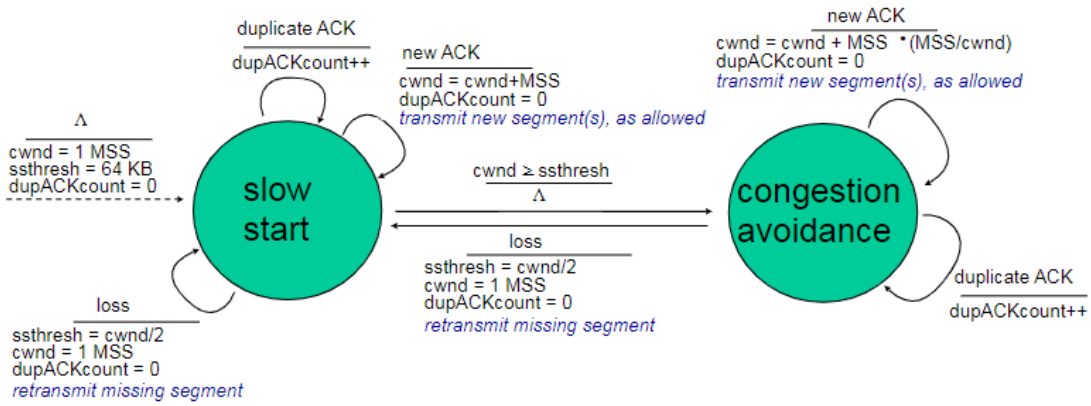
- Congestion avoidance is a linear increasement, and slow-start is exponential increasement
- TCP need to switch from exponential to linear, since
  - exponential will create extra large number when steps deep in

- Create a variable `ssthresh`
- If `cwnd >= ssthresh`
  - Then set mode to congestion avoidance
- Whenever the mode is, if loss packet detect:
  - `ssthresh = cwnd / 2`
  - `cwnd = 1`
  - switch mode to slow-start



Flowchart for switching between slow-start and congestion avoidance

> This is also called TCP Tahoe



Reno's Fast Recovery and detecting duplicate ACKs

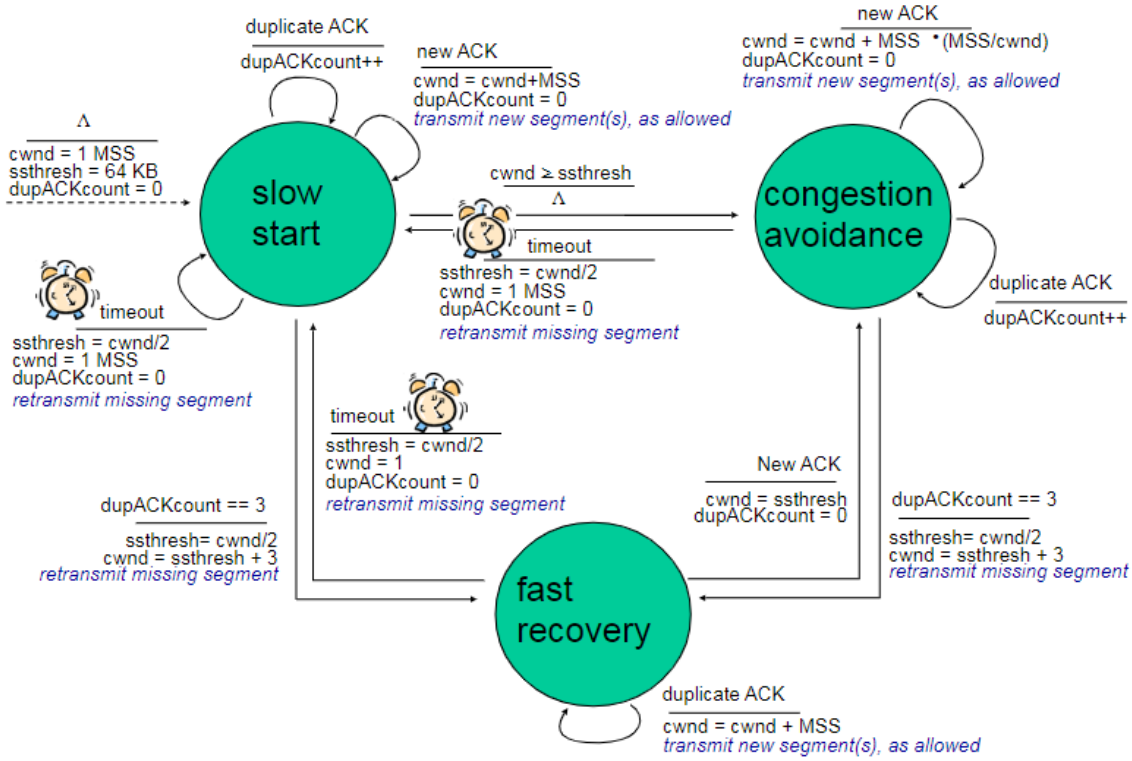
> Duplicate ACK will occur if:
>
> - packet loss, the receiver sends ACK for previous packet after timeout

TCP Reno:

- If **3 duplicate ACKs** detected
  - `ssthresh = cwnd / 2`
  - `cwnd = ssthresh + 3`
  - **REMAIN THE CONGESTION MODE**

> `cwnd` 变为 `ssthresh + 3` 中，3的意义是acknowledge从发送方发出的3个duplicate ACK

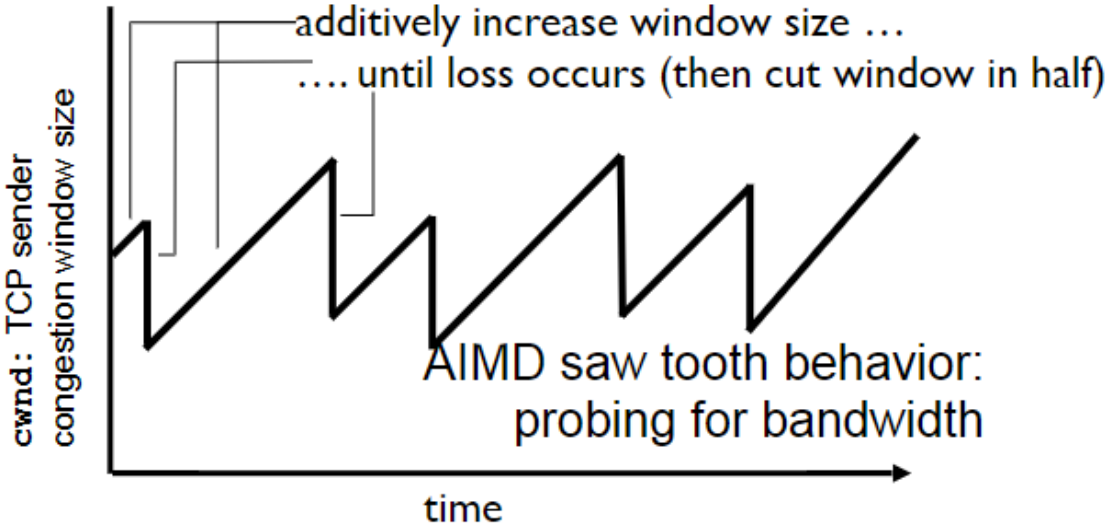
Full Flowchart for TCP Congestion Control



Additive increase and Multiplicative decrease

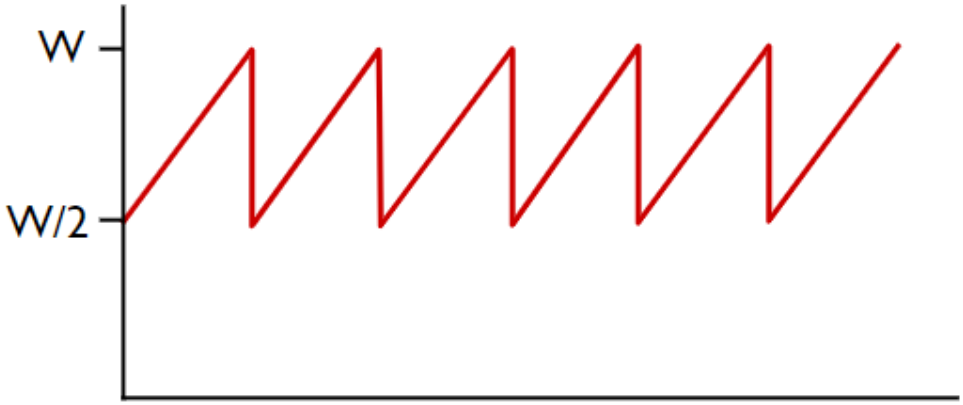
Additive increase: increase `cwnd` by 1 MSS every RTT until loss detected

Multiplicative decrease: cut `cwnd` in half after loss



TCP throughput

- Average TCP throughput as function of window size, RTT
  - ignore slow-start, assume always data to send
- W: window size (measured in bytes) where loss occurs
  - window grows linearly to W, then drops to $\frac{W}{2}$
  - average window size (number of in-flight byte) is $\frac{3}{4}W$
  - average throughput is $\frac{3}{4}W$ per RTT



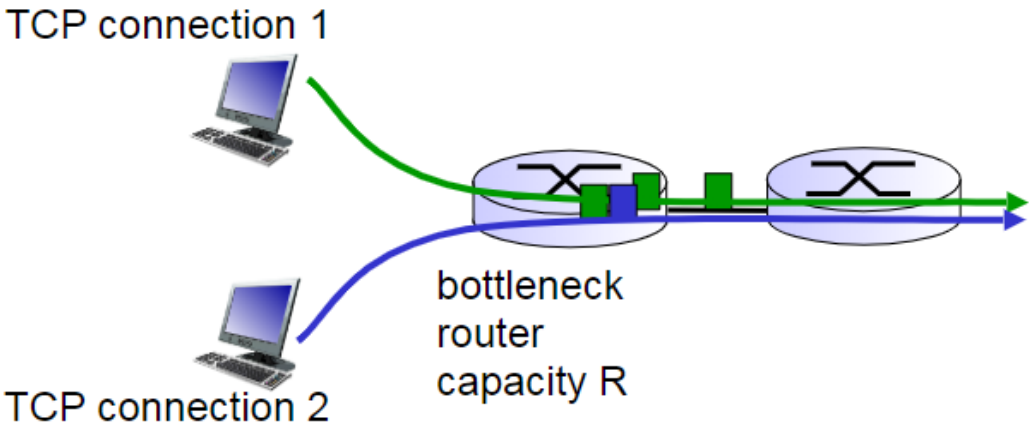
TCP over “long fat pipes” - TCP在长肥管道上

> 带宽延迟积很大的网络叫做长肥网络(LFN, Long Fat Network)，在LFN中建立的TCP信道叫做长肥管道

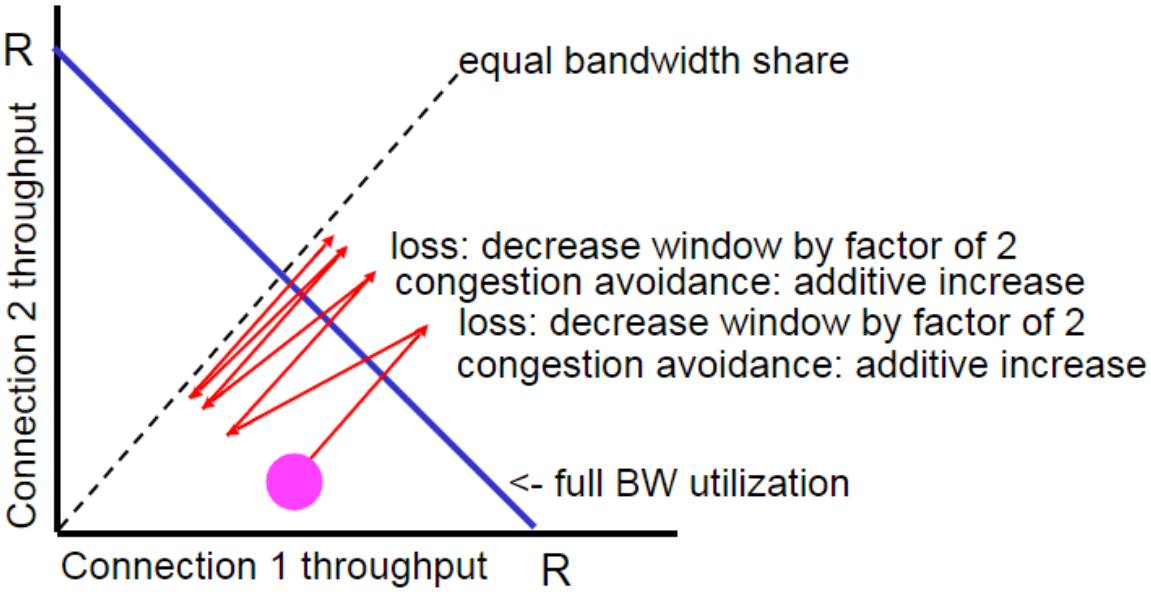
- example:
  - 1500 byte segments, 100ms RTT
  - 10 Gbps requires W=83333 in-flight segments
- throughput in terms of segment loss probability, L
  $\text{TCP throughput} = \frac{1.22 \times MSS}{RTT \times \sqrt{L}}$
  - To achieve 10 Gbps throughput, the loss rate needs $L = 2 \times 10^{10}$

TCP fairness

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

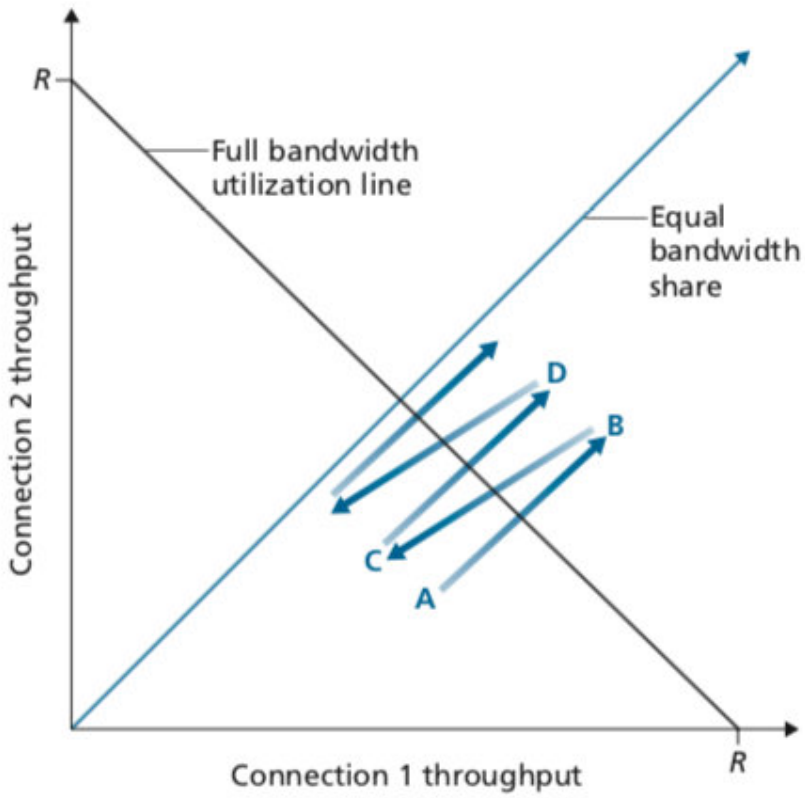


- Two competing sessions:
  - additive increase gives slope of 1, as throughout increases
  - multiplicative decrease decreases throughput proportionally



理想假设：仅TCP连接通过瓶颈链路传输，TCP连接具有相同的RTT，一对收发双方仅有一条TCP连接

理想情况：考虑congestion avoidance模式下，连接1和2的吞吐量呈同等速率的线性增长(equal bandwidth line)，所以是45°角朝右上。设置起点为A，两个连接都持续增大吞吐量，走出AB路径。在B点时总吞吐量大于总带宽，故连接1和2都会发生congestion导致丢包，`cwnd`减半，到C点。然后继续45°角朝上，以此类推。长此以往连接1和2的吞吐量会收敛至相等。



Fairness and UDP

- Multimedia apps often do not use TCP
  - since they do not want link rate throttled by congestion control
- Instead use UDP:
  - send audio/video at constant rate, tolerate packet loss

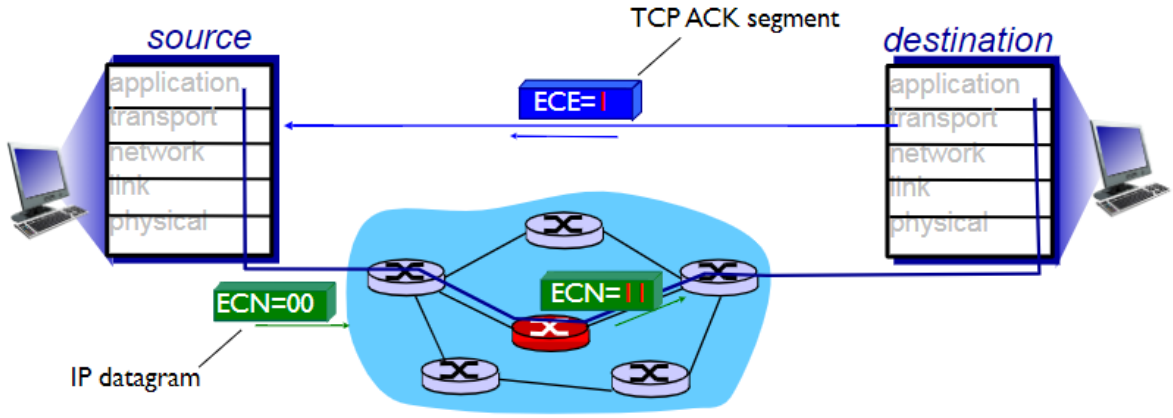
Fairness, parallel TCP connection

- Application can open multiple parallel connections between two hosts
  - For example: web browsers
- Example:
  - link of rate R with 9 existing connections:
    - if a new app asks for 1 TCP gets rate $\frac{1}{10}R$
    - if a new app then asks for 11 TCP, gets rate $\frac{11}{11+10} = \frac{11}{21} \times R \approx \frac{1}{2}R$

Explicit Congestion Notification (ECN) - 显式拥塞提醒

Network-assisted congestion control

- Two bits in IP header (ToS field) marked by *network router* to indicate congestion
- Congestion indication carried to receiving host
- Receiver (seeing congestion indication in IP datagram) sets ECE bit in TCP header of receiver-to-sender ACK segment to notify sender of congestion

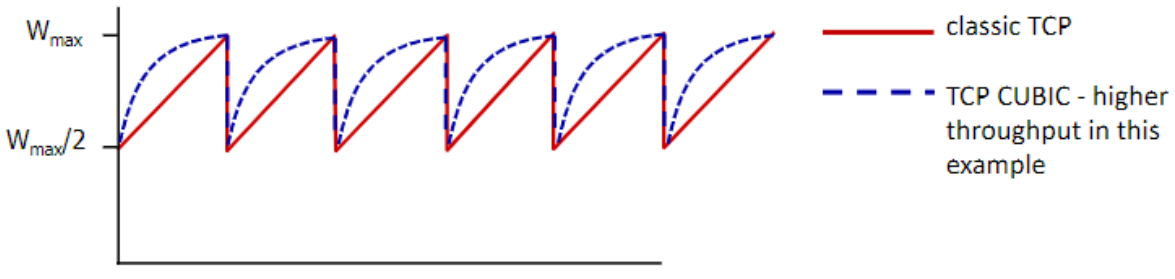


显式拥塞提醒（Explicit Congestion Notification，ECN）【RFC 3168】是一种网络拥塞控制的形式，在网络层的IP数据包包头，有两比特的表示位用于ECN

- 用法一：当路由器处于拥塞状态时，会置位为1，接收方在回复ACK时，会将ECN Echo标志位置为1，发送方收到这种ACK后，会对拥塞窗口进行调整，并在下一个发出的包里面将拥塞窗口已减少的标志位（CWR）置为1
- 用法二：发送方可以通过设置ECN标志位去告知路由器，收发双方支持ECN，故可以执行ECN的相关行为

3.8 BBR and QUIC

TCP CUBIC



- W_{max}: sending rate at which congestion loss was detected
- congestion state of bottleneck link probably hasn't changed much

...

BBR

Network Layer Data Plane

4.1 - Overview of Network Layer

- transport segment from sending to receiving host
  - sender: network layer encapsulates segments into datagrams, passes to link layer
  - Receiver: delivers segments to transport layer
- Network layer protocols in every host and router, but not in switch
- Router:
  - examines header fields in all IP datagrams passing through it
  - moves datagrams from input ports to output ports to transfer datagrams along end-end path

Key network-layer functions

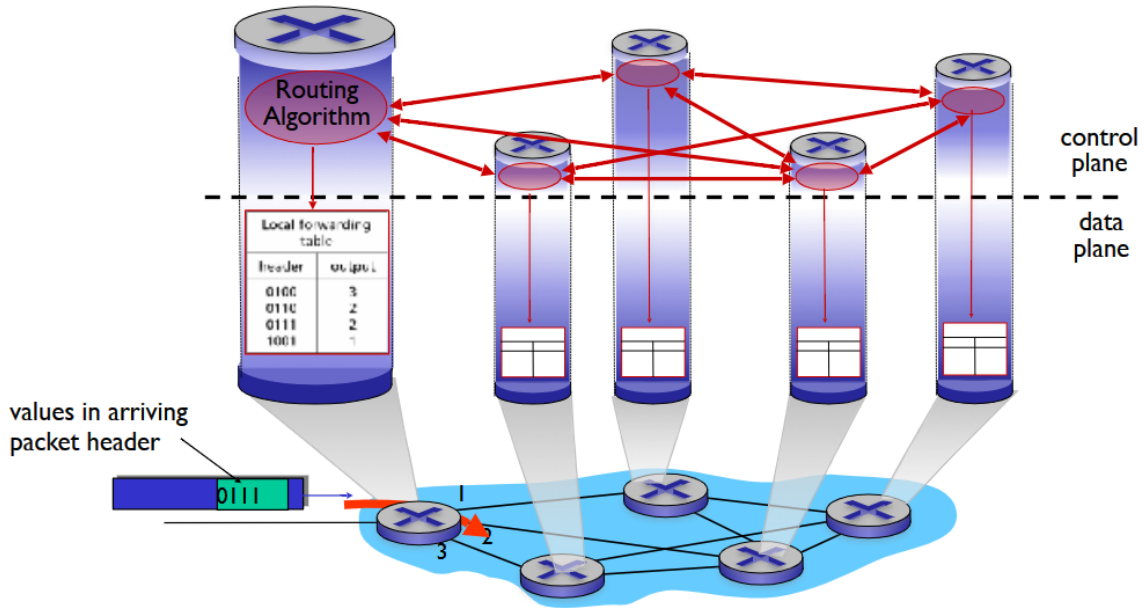
- Forwarding: move packets from router's input to appropriate router output
- Routing: determine route taken by packets from source to destination
  - Routing algorithm

Network layer: data plane, control plane

- Data plane
  - **local**, per-router function
  - determines how datagram arriving on router input port is **forwarded** to router output port.
  - forwarding function
- Control plane
  - **network-wide** logic
  - determines how datagram is **routed** among routers along end-end path from source host to destination host
  - two control-plane approaches:
    - traditional routing algorithm:
      implemented in routers
    - software-defined networking (SDN):
      implemented in (remote) servers

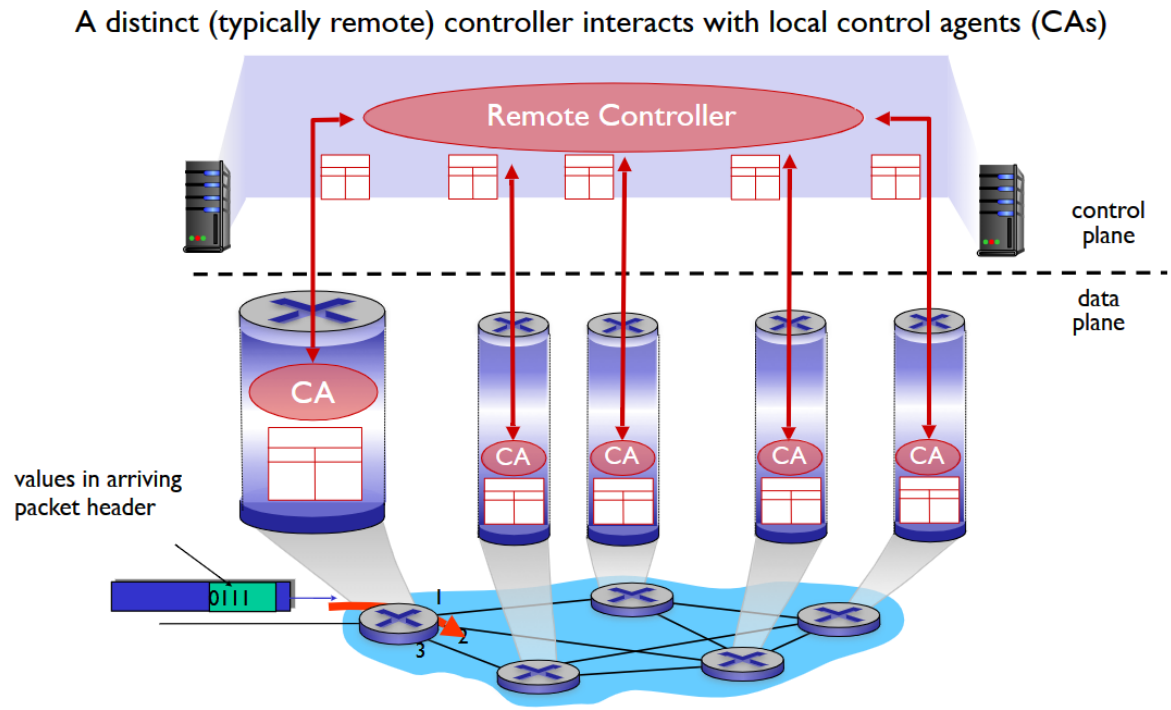
Per-router control plane

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



Software-Defined Networking (SDN) Logically centralised control plane

A distinct (typically remote) controller interacts with local control agents (CAs)



Network service model

> What service model for “channel” transporting datagrams from sender to receiver?

example services for individual datagrams:

- guaranteed delivery
- guaranteed delivery with less than 40ms delay

example services for a flow of datagrams:

- in-order datagram delivery
- guaranteed minimum bandwidth to flow
- restrictions on changes in inter-packet spcaing

Network layer service model

Network Architecture	Service Model	Guarantees ?				Congestion feedback
		Bandwidth	Loss	Order	Timing	
Internet	best effort	none	no	no	no	no (inferred via loss)

Internet "best effort" service model

No guarantees on:

i. successful datagram delivery to destination
ii. timing or order of delivery
iii. bandwidth available to end-end flow

Network Architecture	Service Model	Guarantees ?				Congestion feedback
		Bandwidth	Loss	Order	Timing	
Internet	best effort	none	no	no	no	no (inferred via loss)
ATM	Constant Bit Rate	constant rate	yes	yes	yes	no congestion
ATM	Variable Bit Rate	guaranteed rate	yes	yes	yes	no congestion
ATM	Available Bit Rate	guaranteed minimum	no	yes	no	yes
ATM	Unspecified Bit Rate	none	no	yes	no	no
Internet	Intserv (RFC 1633)	yes	yes	yes	yes	
Internet	Diffserve (RFC 2475)	possible	possibly	possibly	possibly	

Reflections on best-effort service:

- Simplicity of mechanism has allowed Internet to be widely deployed adopted
- Sufficient provisioning of bandwidth allows performance of real-time applications to be "good enough" for "most of the time"
- Replicated, application-layer distributed services (datacenters, content distribution network) connecting close to clients' networks, allow services to be provided from multiple locations.
- Congestion control of "elastic" services helps.

机制的简单性使得互联网被广泛部署采用

足够的带宽供应使实时应用程序的性能在 "大部分时间 "都是 "足够好"。

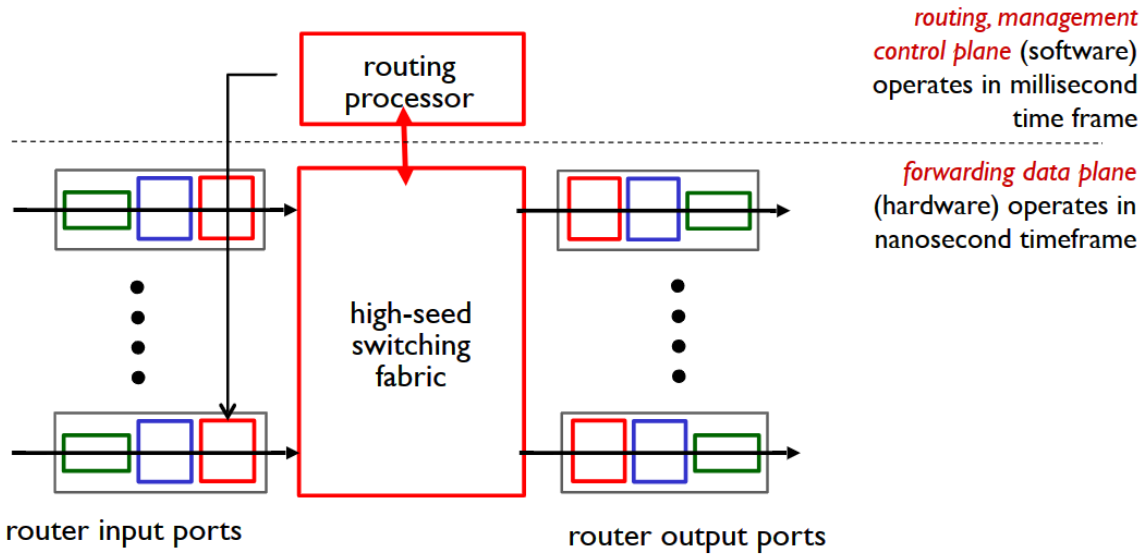
复制的、应用层的分布式服务（数据中心、内容分发网络）连接到客户的网络附近，允许从多个地点提供服务。

弹性"服务"的拥堵控制。

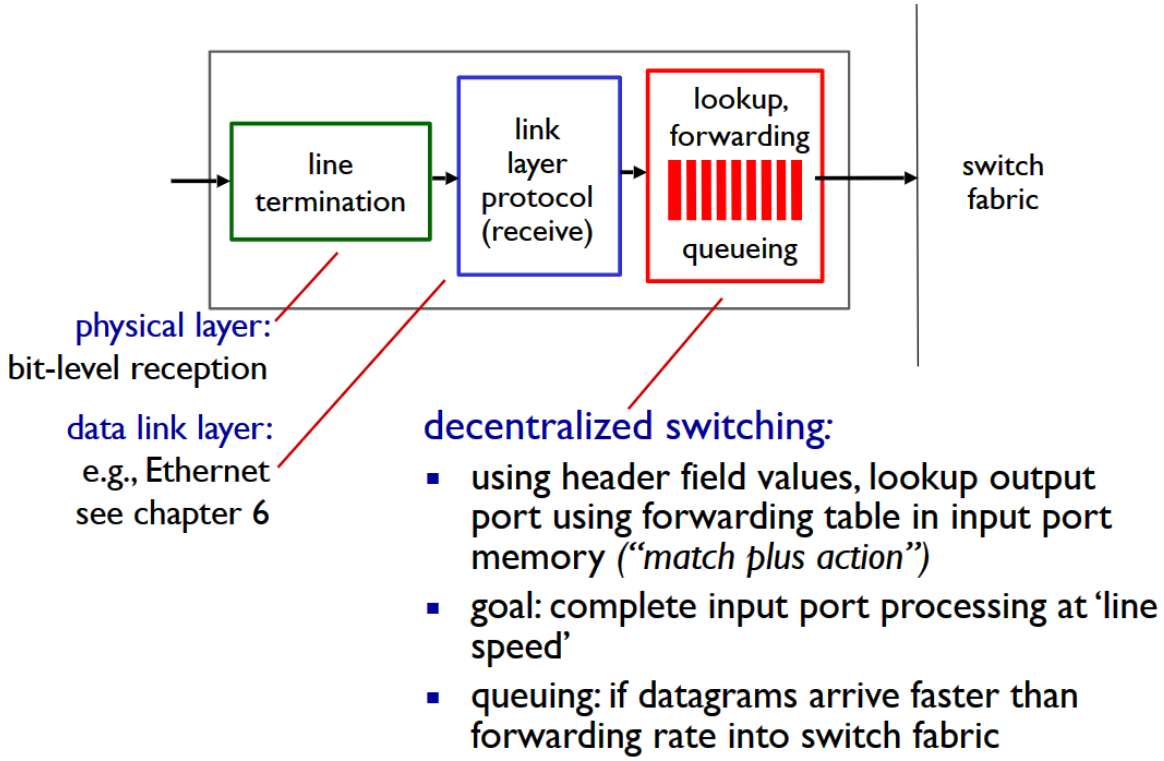
4.2 - What's inside a router

Router architecture overview

- High-level of generic router architecture:



Input port functions



In lookup, forwarding, queueing:

- Destination-based forwarding: forward based only on destination IP address (traditional)
- Generalised forwarding: forward based on any set of header field values

Destination-based forwarding

forwarding table

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

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

Longest prefix matching

When looking for forwarding table entry for given destination address, use **longest** address prefix that matches destination address.

例如

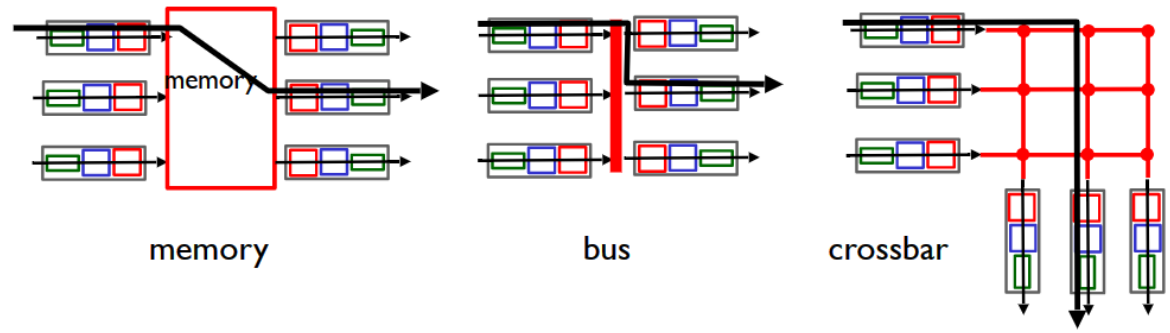
地址“11001000 00010111 00010110 10100001”会被分配到端口0

地址“11001000 00010111 00011000 10101010”会被分配到地址1，根据最长前缀匹配原则

- Longest prefix matching often performed using ternary content addressable memories (TCAMs)
  - Content addressable: present address to TCAM: retrieve address in one clock-cycle, regardless of table size
  - Cisco Catalyst: up ~1M routing table entries in TCAM

Switching fabrics

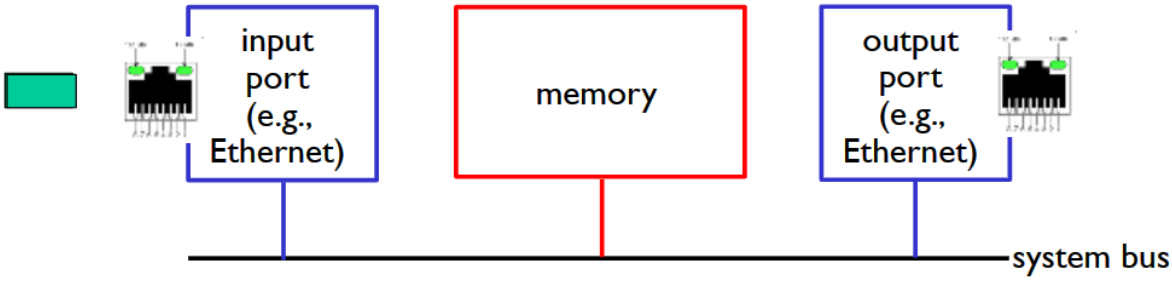
- Transfer packet from input buffer to appropriate output buffer
- Switching rate: rate at which packets can be transfer from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable
- Three types of switching fabrics



Switching via memory

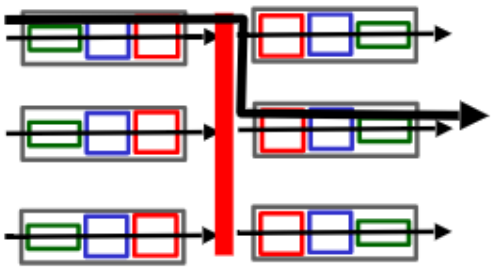
First generation routers:

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



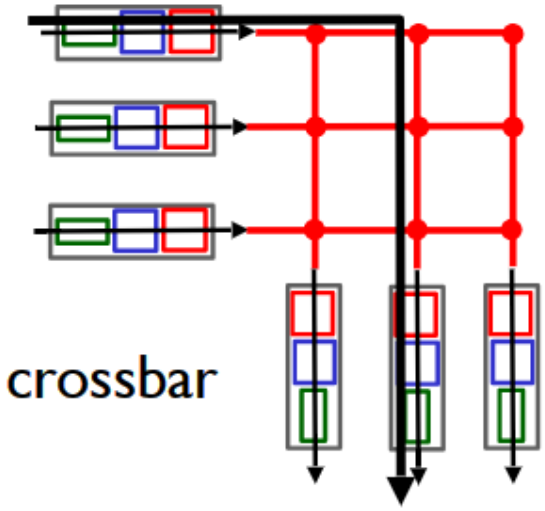
Switching via a bus

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

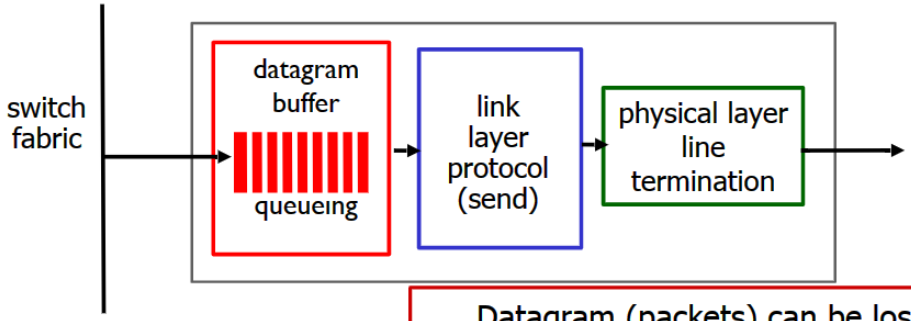


Switching via interconnection network

- Overcome bus bandwidth limitations
- Banyan networks, crossbar, other interconnection nets initially developed to connect processors in multiprocessor
- Advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric



Output ports



Datagram (packets) can be lost due to congestion, lack of buffers

- *buffering* required when datagrams arrive from fabric faster than the transmission rate
- *scheduling discipline* chooses among queued datagrams for transmission

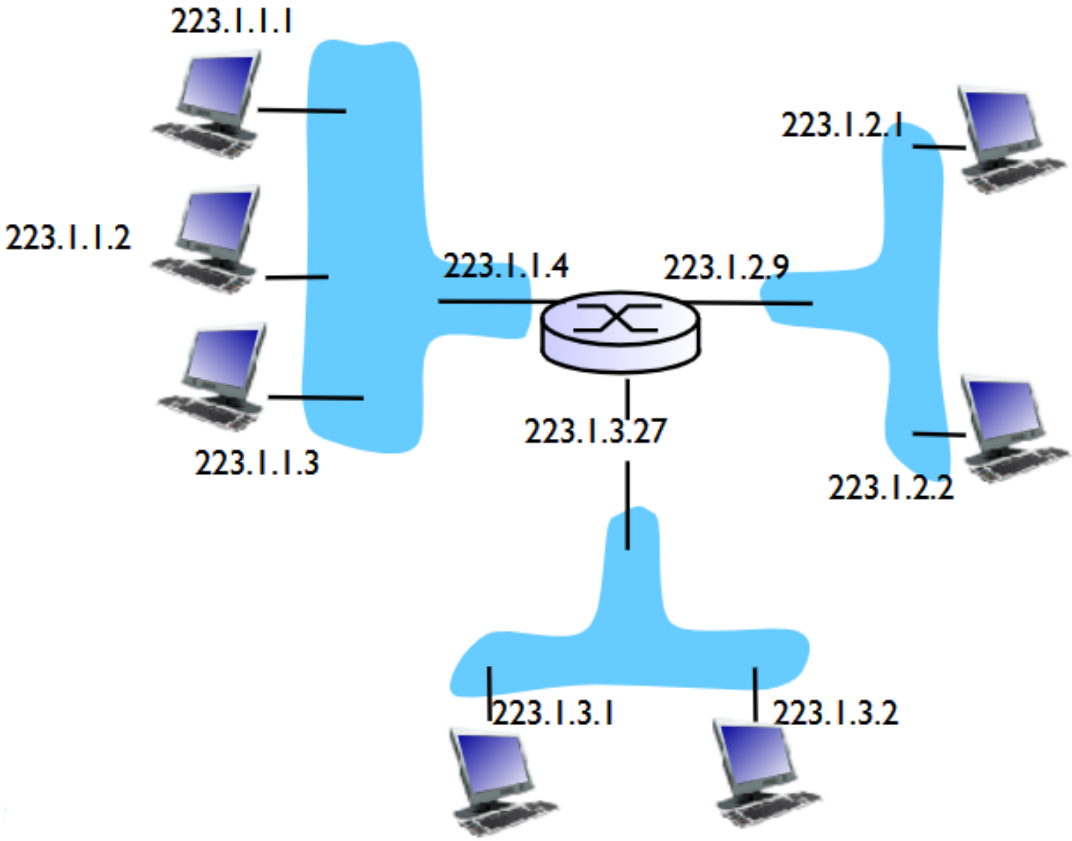
Priority scheduling – who gets best performance, network neutrality

4.3 - IP: Internet Protocol

IPv4 addressing

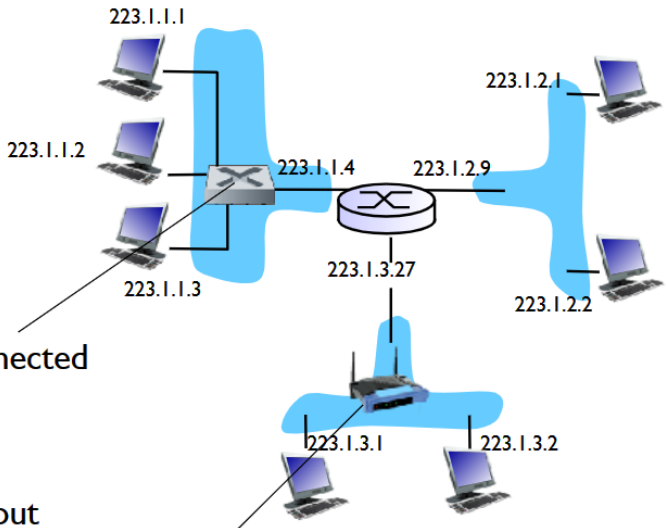
Introduction

- IP address: 32-bit identifier for host, router interface
- Interface: connection between host/router and physical link
  - router's typically have multiple interfaces
  - host typically has one or two interfaces (Ethernet, wireless 802.11)
- **IP addresses associated with each interface**



223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1



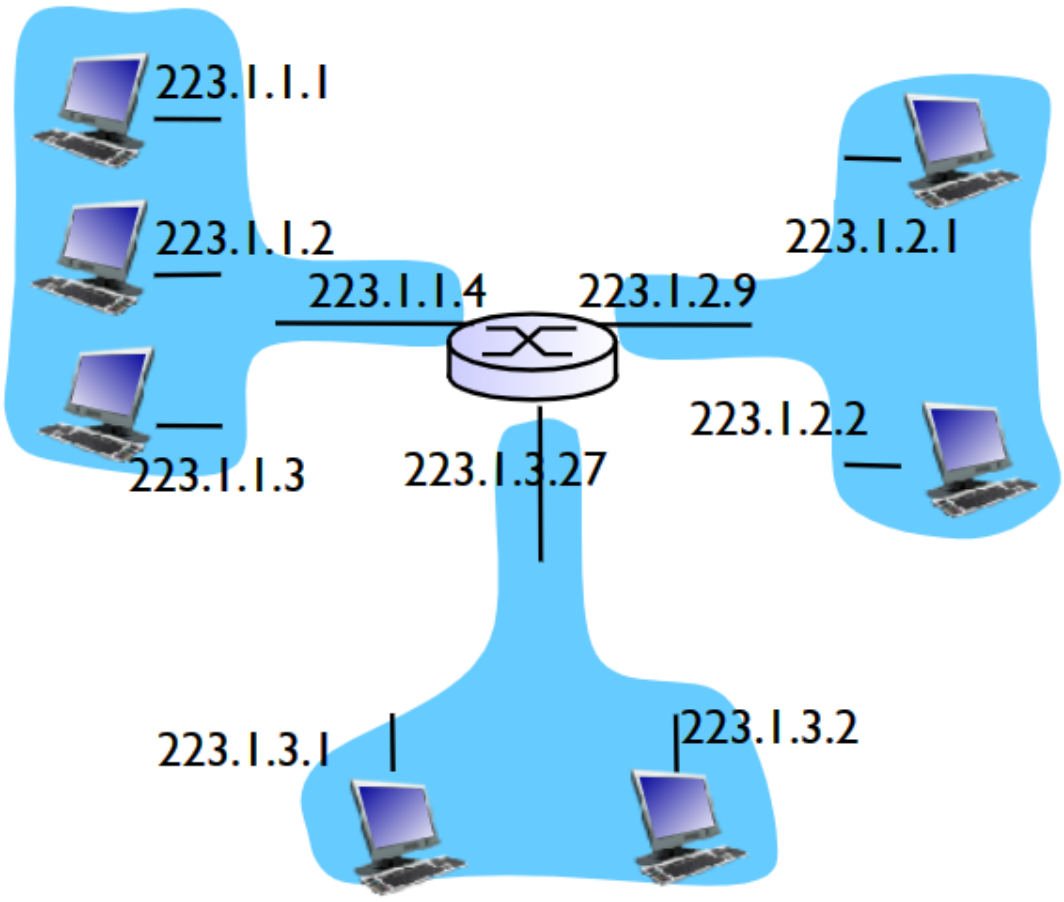
A: wired Ethernet interfaces connected by Ethernet switches

For now: don't need to worry about how one interface is connected to another (with no intervening router)

A: wireless WiFi interfaces connected by WiFi base station

Subnets

- Recipe
  - to determine the subnets, detach each interface from its host or router, creating islands of isolated networks.
  - isolated network terminates at each interface
  - each isolated network is called subnet



Classful Addressing Trade-offs

Classful Routing

(Internet Classic—changed in 1993)

- take a 32-bit IP address, get 4 classes

High Order Bits	Format	Class
0	7 bits of net, 24 bits of host	a
10	14 bits of net, 16 bits of host	b
110	21 bits of net, 8 bits of host	c
111	escape to extended addressing mode	

- A: 128 networks @ 16M hosts each
- B: 16k networks @ 64k hosts each
- C: 2M networks @ 256 hosts each

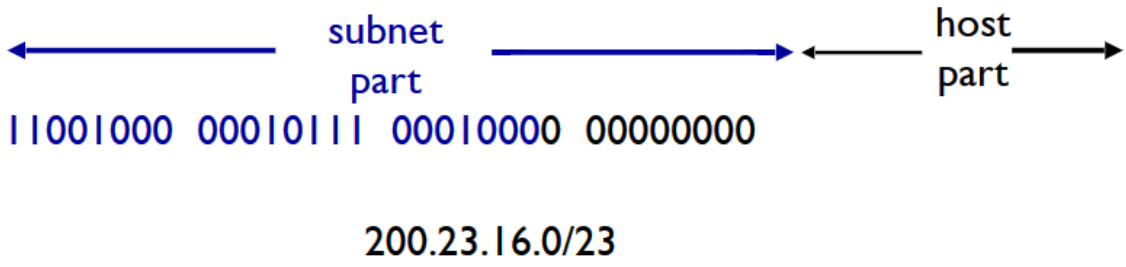
Pros	Cons
Easy to identify what the network is just by looking at the address	Can waste address space: big gaps in size from A to B to C. Not flexible, uneven allocation.
Simple to build fast routers	2M class C -> large routing tables, lots of memory
Can support needs of different size networks	Too few mid-sized networks

- Problems
  - But fixed classes are a poor match for the growing Internet
    - Many groups needed > 256 but <<64K hosts
    - AND 128+16k+2M ~= 2M networks >> router routing table capacities (~200k)
- Solutions
  - Short-term: Classless Internet Domain Routing, CIDR
    - Make better use of existing space
    - divide addresses on any bit boundary
  - Long-term: IPv6
    - increase address space (to 128-bits)

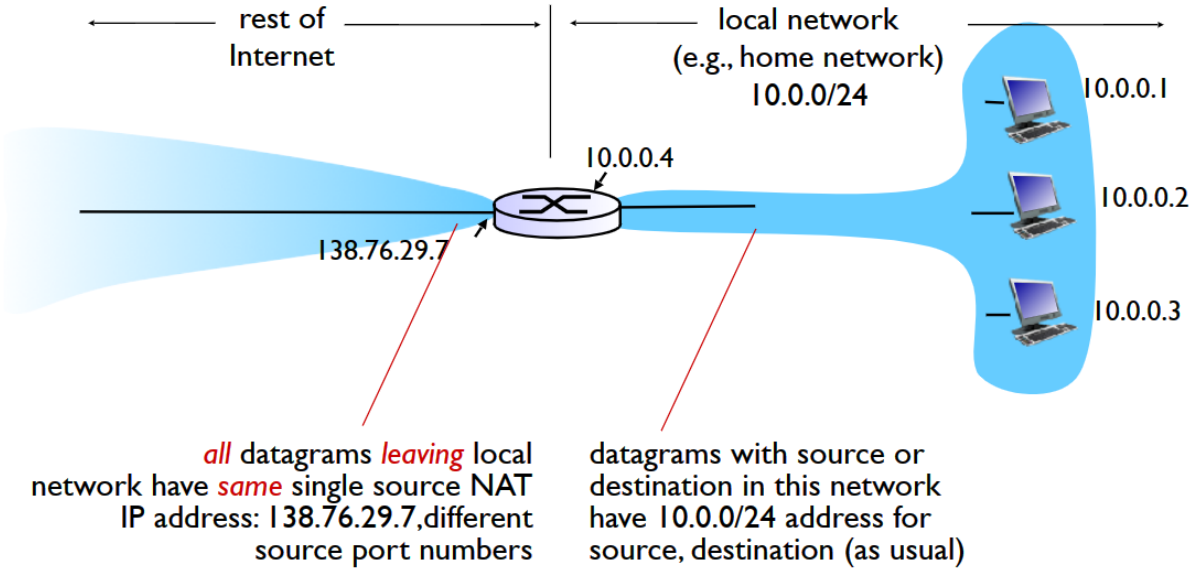
IP addressing: CIDR

CIDR: Classless Internet Domain Routing

- Subnet portion of address of arbitrary length
- address format: **a.b.c.d/x**, where x is number of bits in subnet portion of address



Network Address Translation (NAT)



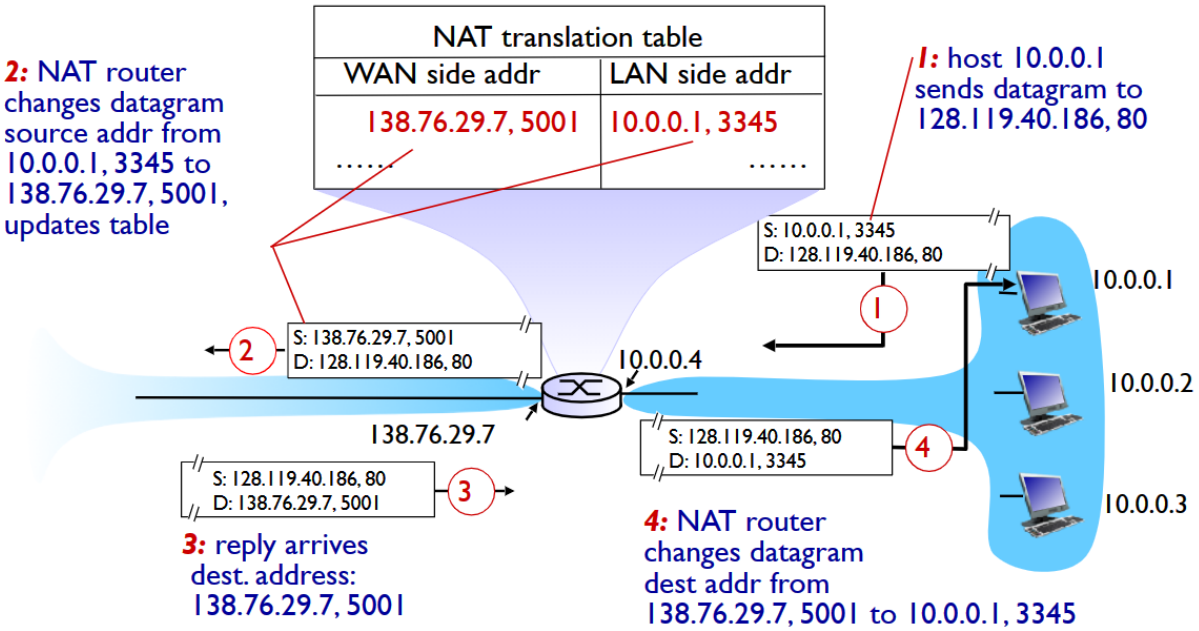
motivation: local network uses just one IP address as far as outside world is concerned:

- Range of addresses not needed from ISP: just one IP address for all devices
- Helps sidestep IPv4 exhaustion
- Can change addresses and/or devices in local network without notifying outside world
- Security: Devices inside local net not explicitly addressable, visible by outside world

网络地址转换： 在IP数据包通过路由器或防火墙时**重写来源IP地址或目标IP地址**的技术。普遍使用在有多台主机但只通过一个公有IP地址访问互联网的私有网络中。

implementation: NAT router must:

- outgoing datagrams: replace [source IP address, port #] of every outgoing datagram to [NAT IP address, new port #]
- remember (in NAT translation table) every [source IP address, port #] to [NAT IP address, new port #] translation pair
- incoming datagrams: replace [NAT IP address, new port #] in dest fields of every incoming datagram with corresponding [source IP address, port #] stored in NAT table



NAT Implications to the Stack

- To the Internet architecture:
  - Breaks global connectivity: not everyone can address everyone
- To transport-layer protocols (TCP, UDP):
  - NAT designed for TCP: TCP setup sets up NAT mapping
  - UDP: make implicit connection setup
- To applications
  - App have to be flexible about port numbers
- To users
  - Easy to add hosts to home network
  - Harder to run applications, P2P

- 16-bit port-number field:
  - 60,000 simultaneous connections with a single LAN-side addres
- often use ***private*** reserved addresses behind NAT
  - RFC1918: 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16
  - only have meaning within local address, not globally
- NAT is controversial:
  - routers "should" only process up to layer 3
  - address shortage "should" be solved by IPv6
  - violates end-to-end argument
    - NAT possibility must be taken into account by app designers, e.g., P2P applications
  - NAT traversal: what if client wants to connect to server behind NAT?
- but NAT is here to stay:
  - extensively used in home/institutional nets, 4G/5G networks

IPv6

Motivation

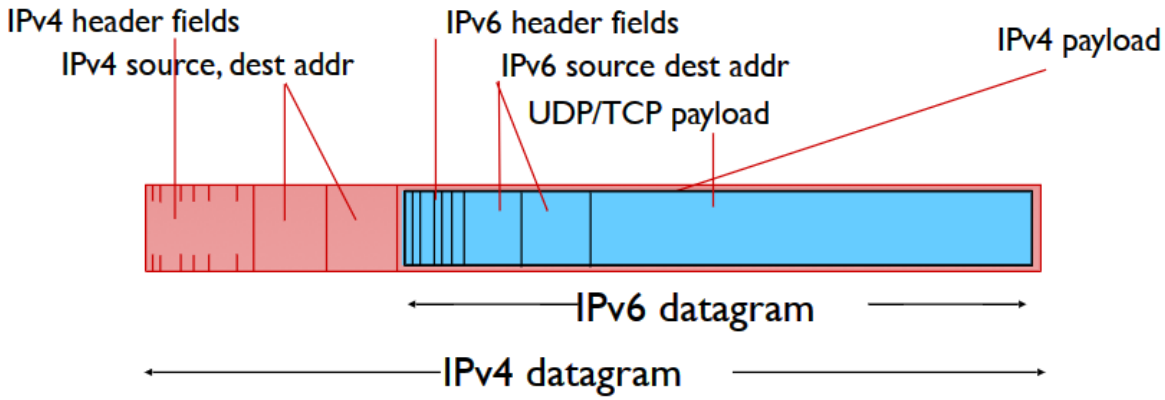
- 32-bit address space soon to be completely allocated
- additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS
  - enable different network-layer treatment of “flows”

IPv6 datagram format

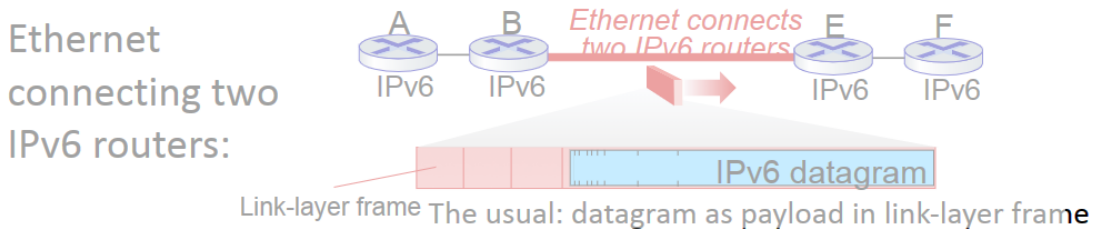
- fixed-length 40 byte header
- **no fragmentation allowed**

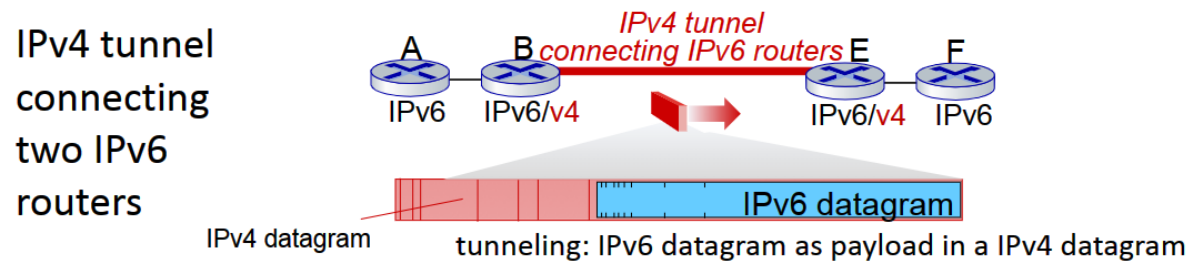
Transition from IPv4 to IPv6

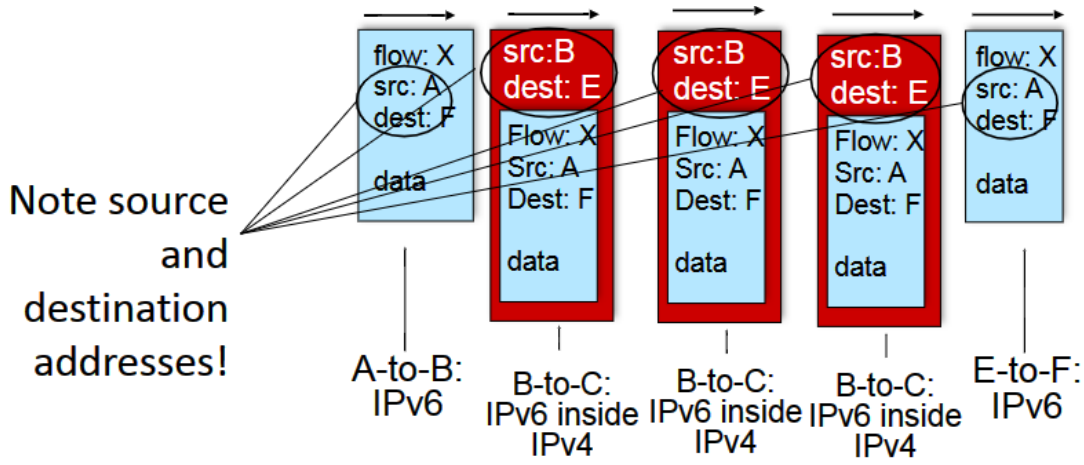
- Not all routers can be upgraded simultaneously
  - no “flag day”
  - how will network operate with mixed IPv4 and IPv6 routers?
- **Tunnelling**: IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers



Tunnelling and encapsulation 隧道和封装







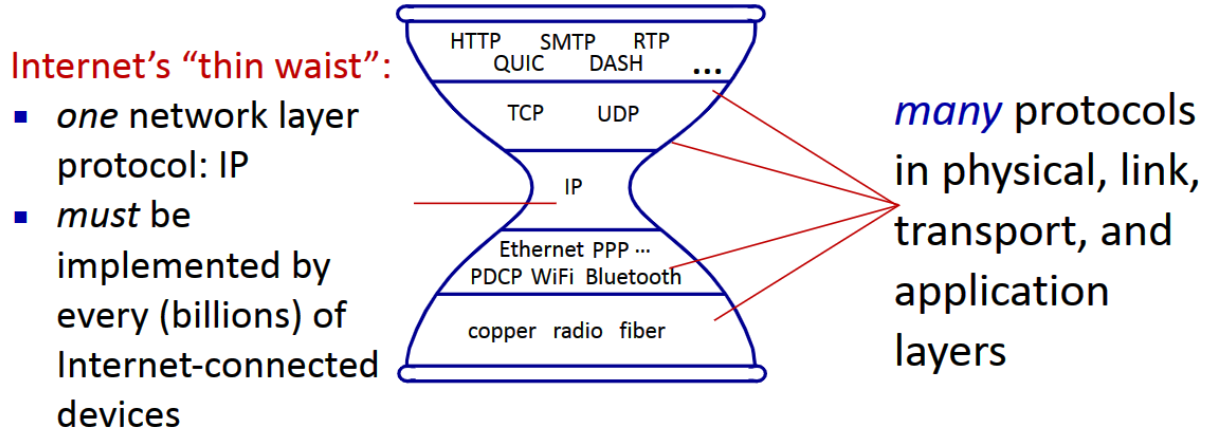
Net Neutrality 网络中立性

Net neutrality:

- Technical: how an ISP should share/allocation its resources
  - packet scheduling, buffer management are the mechanisms
- Social, economic principles
  - protecting free speech
  - encouraging innovation, competition
- Enforced legal rules and policies

> 2015 US FCC Order on Protecting and Promoting an Open Internet: three “clear, bright line” rules:
> ▪no blocking ... “shall not block lawful content, applications, services, or non-harmful devices, subject to reasonable network management.”
> ▪no throttling ... “shall not impair or degrade lawful Internet traffic on the basis of Internet content, application, or service, or use of a non-harmful device, subject to reasonable network management.”
> ▪no paid prioritization. ... “shall not engage in paid prioritization”

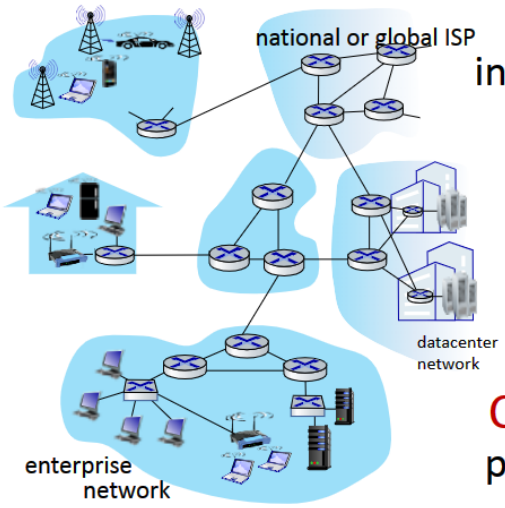
Middleboxes 中间人



Middleboxes are everywhere:

NAT: home, cellular, institutional

Application-specific: service providers, institutional, CDN



Firewalls, IDS: corporate, institutional, service providers, ISPs

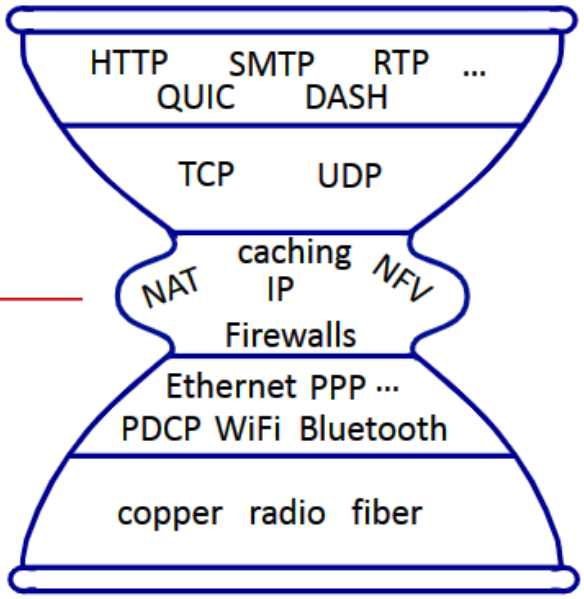
Load balancers: corporate, service provider, data center, mobile nets

Caches: service provider, mobile, CDNs

- 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) centralised control and configuration management often in private/public cloud
- Network functions virtualisation (NFV)
  - programmable services over white box networking, computation, storage

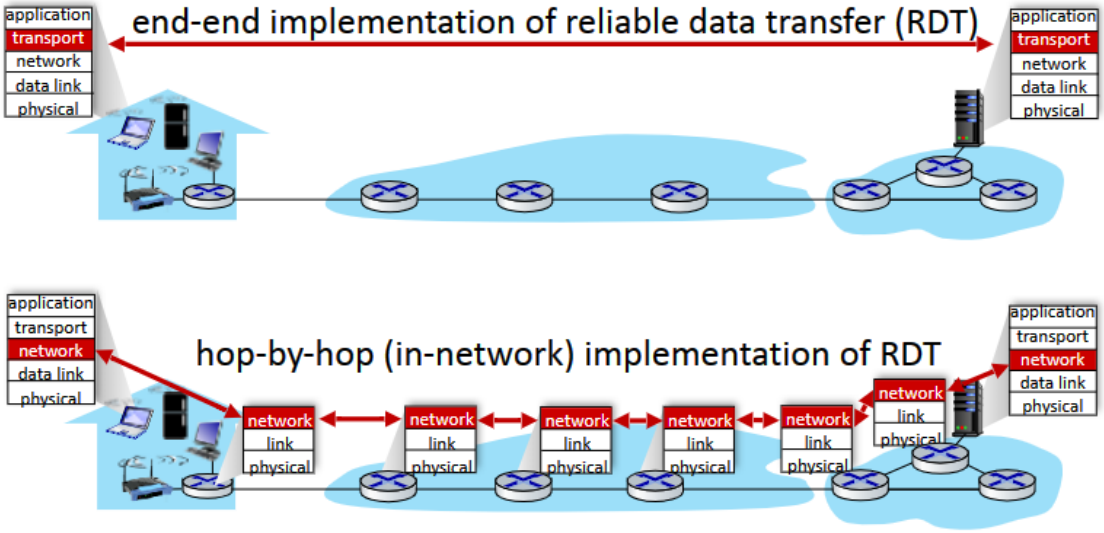
Internet’s middle age “love handles”?

- middleboxes, operating inside the network



The end-to-end argument

Some functionality can be implemented in network or/and at network edge



> “The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system.” “Functions placed at low levels of a system may be redundant or of little value when compared with the cost of providing them at that low level.” “(Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)” We call this line of reasoning against low-level function implementation the “end-to-end argument.”

Network Layer Control Plane

BGP Routing Policy

###

Chapter 6 - Link Layer and LANs

6.1 Introduction, services

Introduction