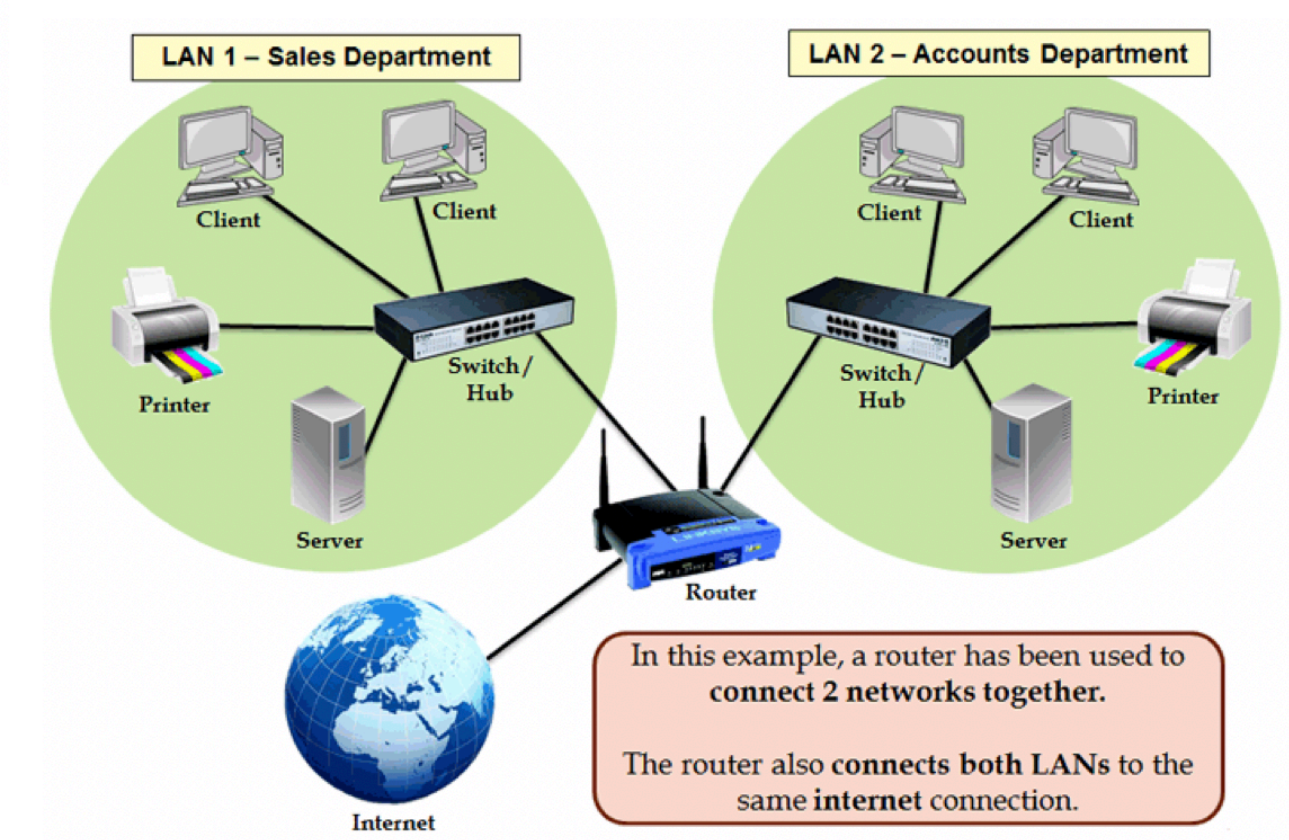
CS 3201 Note

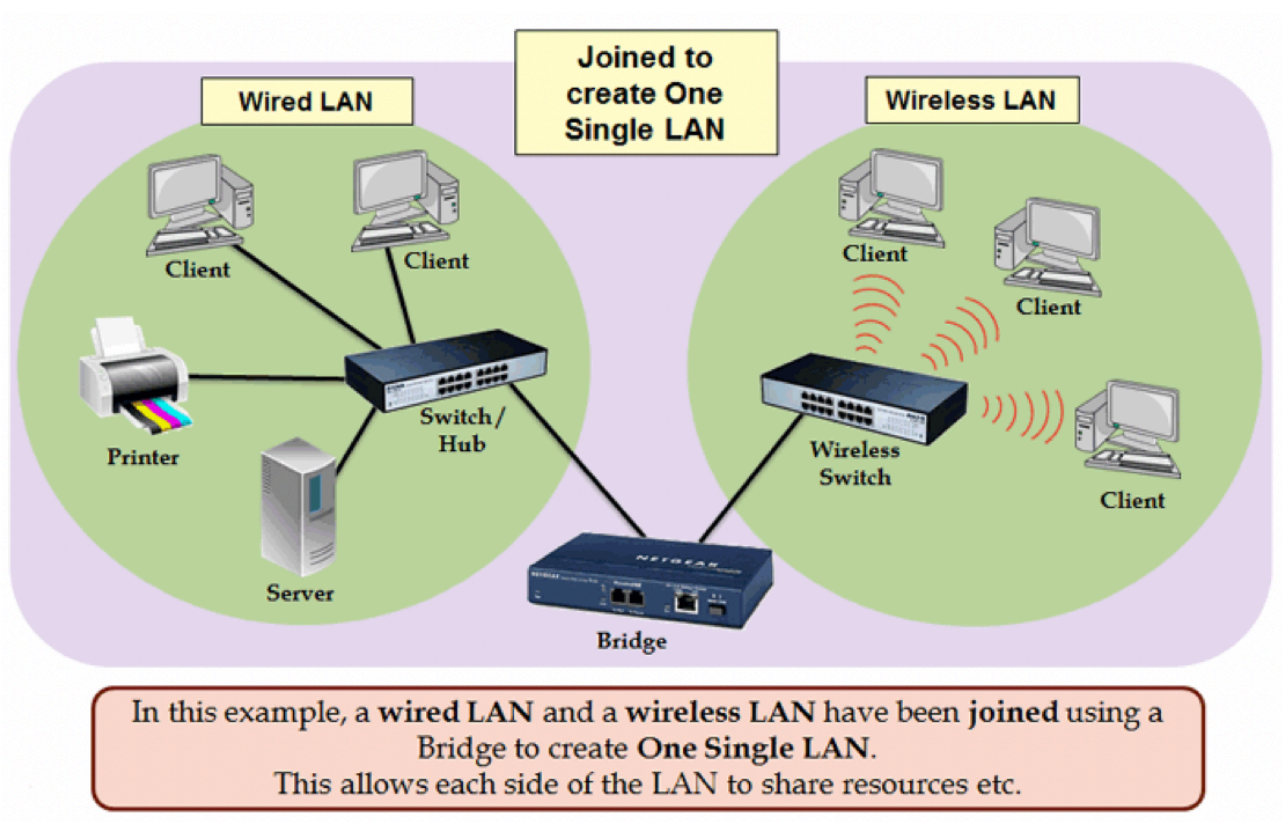
Tutorial 1

1. Some devices

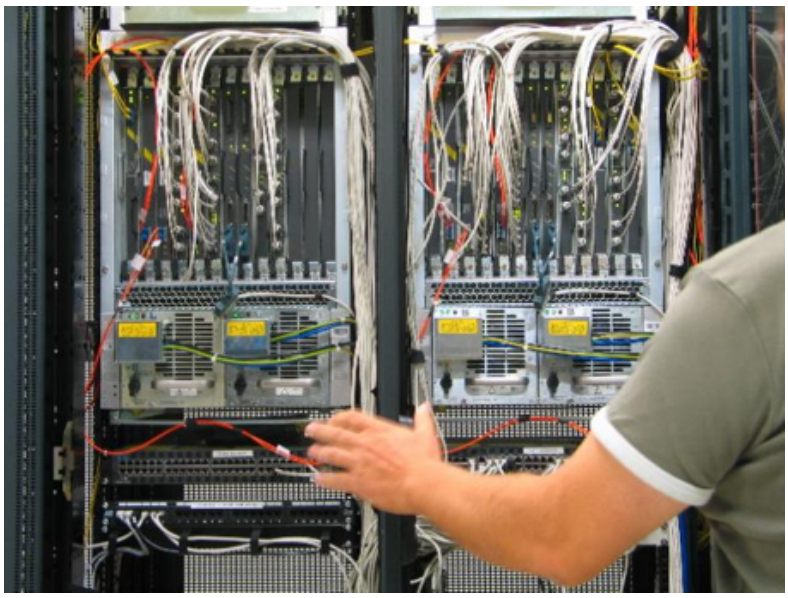
- Network Interface card : Connects computers to a network using cables. Any device needs a network interface card to connect to the internet
- Network cables : Network cables have connectors on each end that plug into network interface cards
- Cable modern() : connets your home computer or wireless router to your *Internet Service Provider* (ISP 中国移动)
- Bluetooth headsets : When Bluetooth is switched on, you device will search for other Bluetooth device to connect to main ebabler if (PAN)
- WIFI router : allow share of internet connection main enabler of (LAN)
- Fare for each device connected to it
    - Hubs : allow devices to connect to each other by plugging network cables into their ports
    - Switch : works similar to hubs, but it can recongonize data (not anyone can access it)
- Router : Allows computers on a LAN to share the same Internet connection



- Bridge: Used to connect multiple LANs together



- Cellular Network Base station: e.g. a 4G station
- A router in the *core network* or wide area network(WAN)

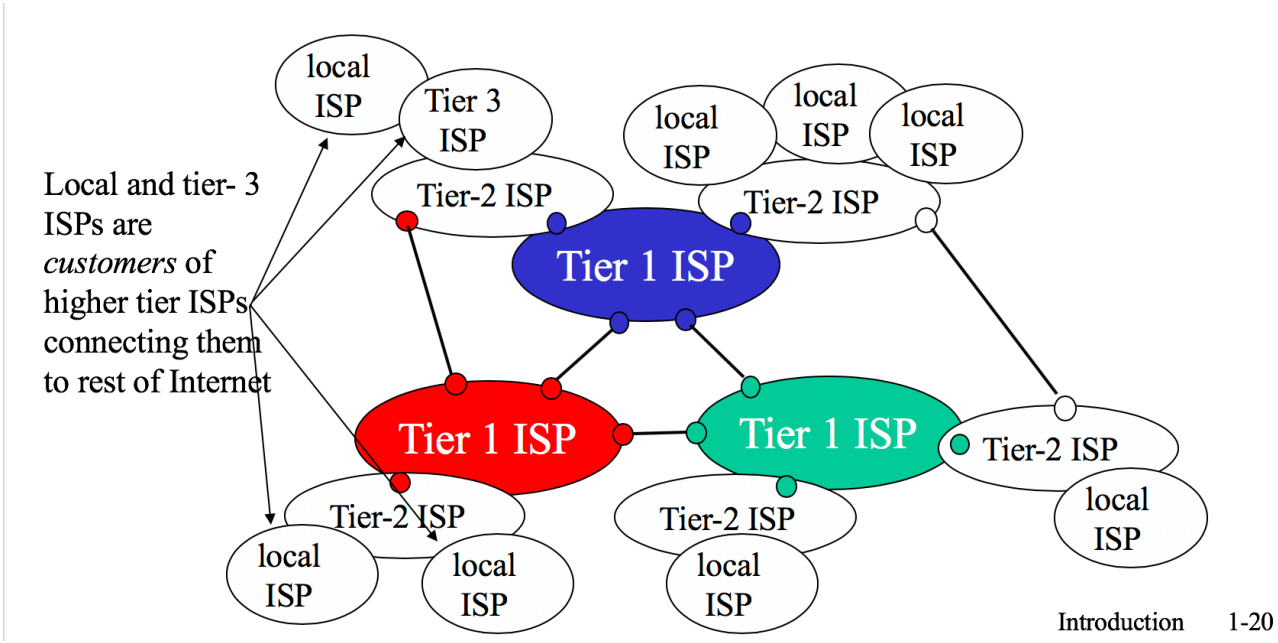


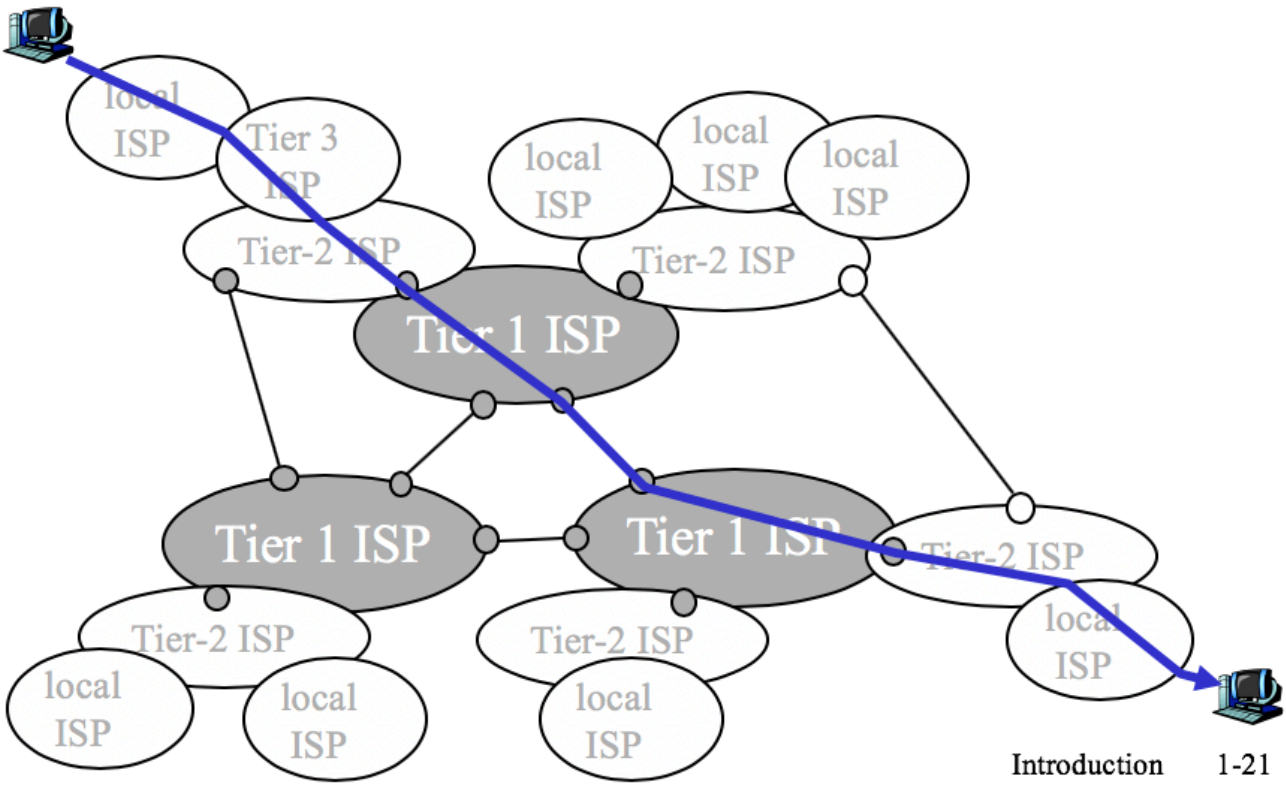
Chapter1 General View

1. 1 What is Internet

1.1.1 A Nuts-and-Bolts Description

- What is Internet
  - millions of connected computing devices: *hosts* = *end systems*
    - running *network apps*
  - communication links
    - Fiber, copper, radio, satellite (physical media)
    - transmission **rate** = bandwidth (bits/seconds)
  - Routers : forward packets
- Packets = header(add by the sender) + segment of file
  - packet switch: receive packets and forward packets toward their ultimate destinations
    - Link-layer switches : used in access networks(Ethernet)
    - Routers: forward packets (chunks of data) (packets: divide big file into small units called packets) used in network cores
- Internet: "network of networks"
  - loosely hierarchical (树) => all connected together
  - public Internet versus private intranet
- ISPs : Internet service provider => provide internet-access service to end systems
  - Tier1: national (also interconnect(peer) privately), treat each other as equals
  - Tier2: regional => connect to 1 or more Tier1 to acess other networks(also peer privately with each other)
  - Tier3: ISPs and local ISPs => closest to end systems





- Protocols controls sending, receiving of msgs
  - e.g. TCP, IP, HTTP, Skype, Ethernet
  - Protocols define **format, order** of msgs sent and received among network entities, and **actions** taken on msg transmission and receiption
  - Internet standards
    - RFC: Request for comments
    - IETF: Internet Engineering Task Force

1.1.2 A Services Description

- Distributed applications: Applications involve multiple end systems that exchange data with each other, and run at the end systems.
- Service
  - communication *infrastructure（基础设施）* (and delivery system i.e. protocols) enables distributed applications
    - Web, VoIP, email, games, e-commerce, file sharing
  - communication services provided to apps:
    - Reliable data delivery from source to destination
    - "Best effort" (unreliable) data delivery => some times we prefer loss of bits than speed of transformation(online game/movie)

1.2 The network edge

1.2.1 Access Networks

- network edge: applications and hosts
- Hosts / end systems
  - client
  - server
- access networks: the network that physically connects an end system to the **first** router(edge router)

- e.g.
    - DSL (digital subscriber line) : A residence typically obtains DSL Internet access from the same local telephone company (telco) that provides its wired local phone access.
    - Cable, FTTH, Dial-Up and Satellite
- Properties
    - **slower than core**
    - dedicated(私有)

1.3 The Network Core

- What is network core :
    - mesh of interconnected routers
    - network of networks
- How data transferred through net?
    - Packet switching
    - Circuit switching

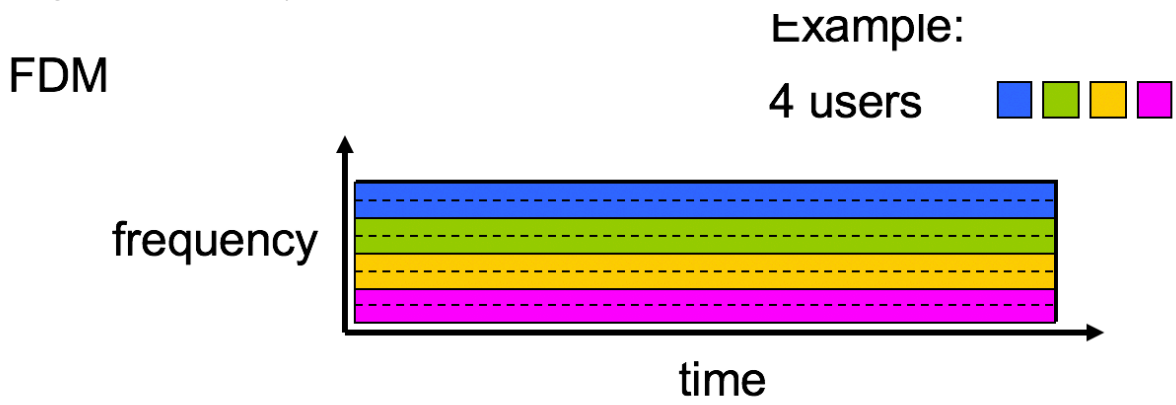
1.3.1 Packet switching

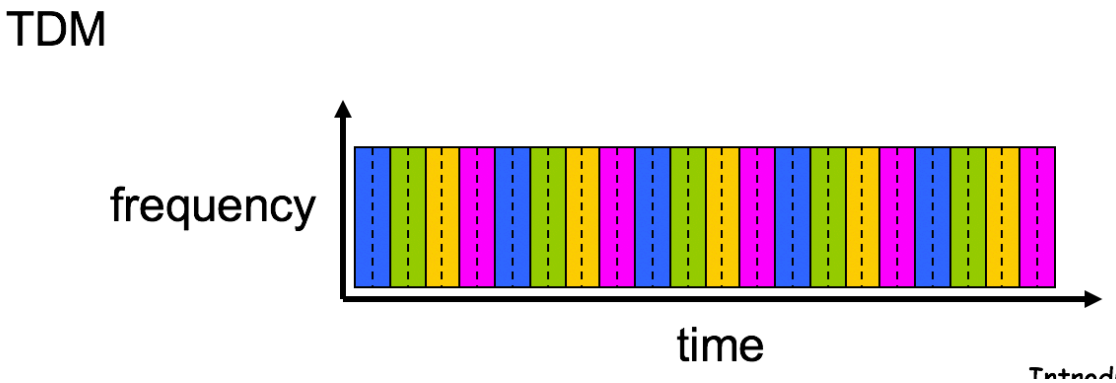
- Method
    - Data stream divided into *packets* , shared by different users
    - each packet uses full link bandwidth
- Cons : resource contention(争夺) :
    - Aggregate(合计) resource demand can exceed amount available
    - Congestion : packets queue, wait for link use
    - store and forward : packets move one hop at a time
        - Node receives complete packet before forwarding
- Statistical multiplexing(统计复用)
    - Sequence of A & B packets does not have fixed pattern, bandwidth shared on demand => statistical multiplexing
    - each host gets same slot in revolving TDM frame
- transmit delay : (num of edges(or links)*size of file)/deliver rate
    - **number of links** means the number of edges consecutively connected together
- Store-and-Forward Transmission :packet switch must receive the entire packet before it can begin to transmit the first bit of the packet onto the outbound link
    - Consider : source has three packets, each consist of L bits
    - First buffer the packet's bits : size of packets is L, rate of buffering is R
    - transmission(for one packet) delay = (L/R)***number of links**
- Queuing Delays and Packet Loss
    - For each link linked to the **output buffer**(output queue)
    - Store the packets to the queue when the output link is busy
    - The queue has finity capacity
    - packet arriving to full queue dropped (aka lost)
    - lost packet may be retransmitted by previous node, by source end system, or not at all
- Forwarding Tables and Routing Protocols
    - IP address of target host is writen in the header of packets by the sender host
    - Forwarding table: Each router has an forward table that maps destination addresses
    - Packets' IP is read by the table to find appropriate outbound link

1.3.2 Circuit Switching

- circuit-switched network : set up a new path between sourse and target
    - dedicated resources: no sharing
    - circuit-like (guaranteed) performance

- call setup required
- dividing link bandwidth into “pieces”





  - frequency-division multiplexing : share channel
    - FM radio stations also use FDM to share the frequency spectrum between 88 MHz and 108 MHz, with each station being allocated a specific frequency band.
  - Time-division multiplexing(TDM): use whole channel but share on time
    - Frame: time is divided into frames of fixed duration
    - Slots: each frame is divided into a fixed number of time slots
    - When the network establishes a connection across a link, the network dedicates one time slot in every frame to this connection
    - Bit rate: total bits per second

$$bit\ rate = bits\ per\ slot * slots\ per\ frame(belongs\ to\ different\ packets) * frame\ per\ sec \tag{1}$$

- transmission rate: for one slot (bit rate/slots per sec), suppose frame per sec is 1
  $bit\ rate = bits\ per\ slots * slots\ per\ frame = bits\ per\ slots\ per\ sec * slots\ per\ sec$

$$dely\ time = time\ to\ set\ up + \frac{packet\ size}{(bits\ per\ slots\ per\ sec)} = time\ to\ set\ up + \frac{packet\ size}{(\frac{bit\ rate}{slots\ per\ sec})}(total\ slots) \tag{2}$$

- Packet Switching Versus Circuit Switching
  - packet switching > circuit switching (avaliable for many users, calculate by probability)
    - Great for **bursty**(Discrete) data
      - resource sharing
      - Simpler, no call setup
    - excessive(过重的) congestion: packet delay and loss
      - protocols needed for reliable data transfer, congestion control
    - How to provide circuit-like behaviour
      - bandwidth guarantees needed for audio/video apps (pay for the service)

1.4 Delay, Loss, and Throughput in Packet-Switched Networks

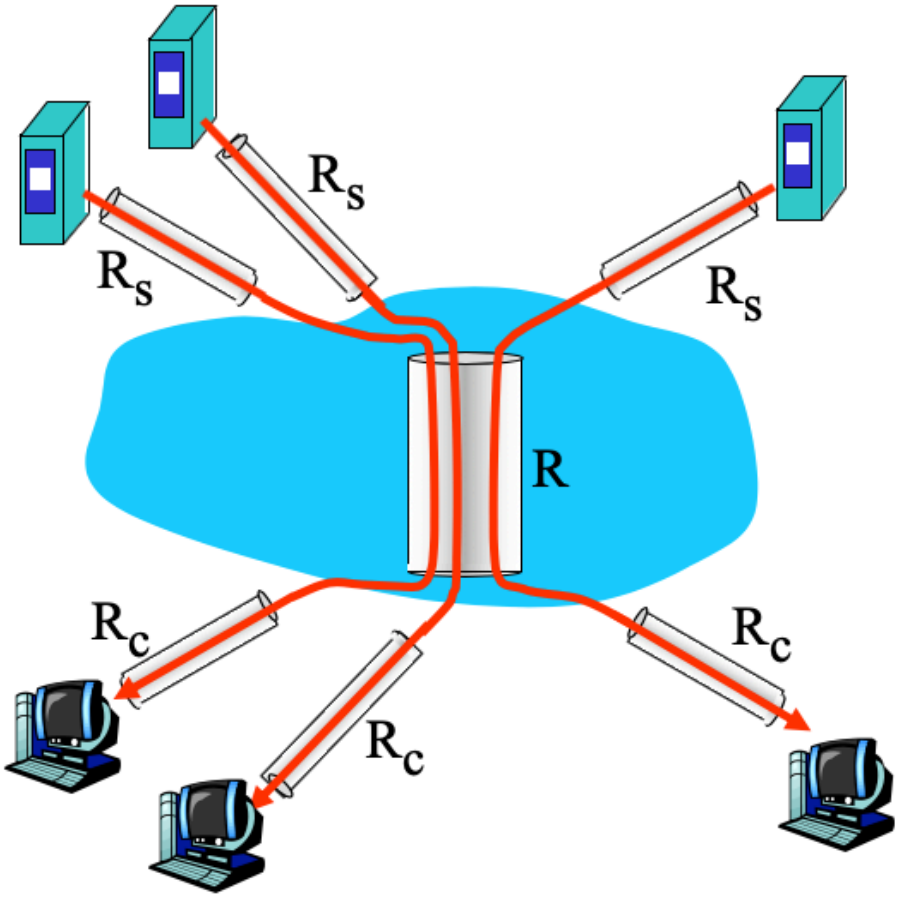
- Four sources of delay
  - Nodal processing : check bit errors and determine output link（negligible 可忽略）
  - Queueing : time waiting at output for transmission and depends on congestion level of router (not negligible)
  - Transmission delay : L(file size)/R(rate) (negligible, but significant for low-speed links)
  - Propagation delay : down vote accepted (negligible)

    Propagation delay is how long it takes *one* bit to *travel* from one end of the "wire" to the other (it's proportional to the length of the wire, crudely).

    Transmission delay is how long it takes to get *all* the bits *into* the wire in the first place (it's packet_length/data_rate).

    $$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{pop} \tag{3}$$

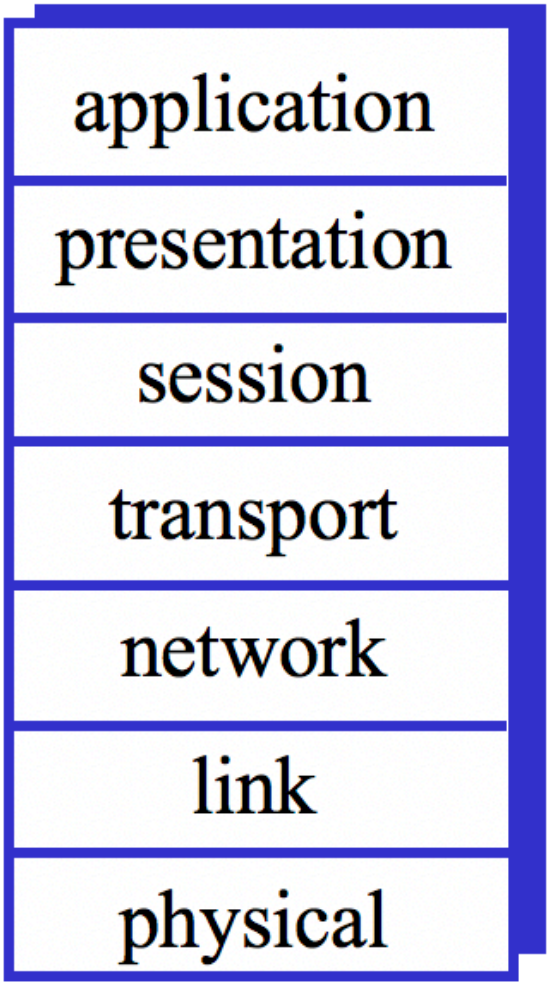
  - Throughput
    - Throughput: rate(bits/time unit) at which bits transfeered between sender/receiver (observe from receiver side)
      - Instantaneous : At any instant of time is the rate(bits/sec) at which Host B is receiving the file
      - Average : 平均速度 F(whole bits)/T(whole time)
      - Consider R_S is the rate of link between server and first router of server, R_c denote the rate of the link between client and first router of client, then the transmission rate is min(R_S,R_c),
        - because the rate for the first routers often are the slowest
      - Per-connection end-end throughput: min(R_S,R_c, $R/10$)
        - where R is the rate of middle routers



      - in practice: R_S and R_c is often bottleneck

1.5 Protocol Layers and Their Service Models

- Layer : each layer implements a service
  - via its own internal-layer actions
  - relying on services provided by layer below (down to top)
  - advantage: different implement of layer(layer's function has not changed) doesn't affect the whole system's function
- Protocol Layering
  - To provide structure to the design of network protocols, network designers organize protocols-and the network harware and software
- Internet protocol stack : Protocols of the various layers are called the protocol stack
  - Application Layer : supporting network applications
    - functions : Such as the translation of human-friendly names for Internet end systems to a 32-bit network address
    - FTP, SMTP, HTTP
    - **Message** : The application in one end system using the protocol to exchange packets of information with the application in another end system, the packet of information **at application layer** is message
  - Transport Layer : processing data and transfer (multiplexing and check error)
    - Transports application-layer messages between application endpoints.
    - May used to
      - deliver the message up to the appropriate application
      - Error-detection bits that allow the receiver to determine whether bits in the message has changed in the route
    - TCP, UDP
      - TCP: provides a connection-oriented service to its applications
        - guaranteed delivery of application-layer messages to the destination and flow control(sender/receiver speed matching)
      - UDP: provides a connectionless service to its applications
    - **Segment**: We will refer to a transport-layer packet as segment
  - Network Layer : routing of datagrams from source to destination (for router decide output link)
    - Responsible for moving network-layer packets known as **datagrams** from one host to another
    - Internet transport-layer protocol(TCP/UDP) in a source host passes a transport-layer segment and a destination address to the network layer
    - The network layer provides the service of delivering the segment to the transport layer in the destination host
    - IP, routing protocols
      - IP : Defines the fields in the datagram as well as how the end systems and routers act on these fields (only one IP protocol, and all Internet component has a network layer must run the IP protocol)
      - Routing protocol : determine the routes that datagrams take between sources and destinations
  - Link Layer: data transfer between neighboring network elements (avoid collision, generate different number)
    - Packets called **Link-layer frame** (has two kinds of fields)
      - header fields
      - **payload field**:Typicaly from layer above
    - To move a packet from one node (host or router) to the next node in the route
    - Service provided depends on the link-layer protocol that is employed over the link
    - PPP, Ethernet
  - Physical : bits "on the wire" (from bits to signal)
- ISO/OSI reference model (2 more layers between application and transport) => do have this in the history



    - presentation : allow applications to interpret meaning of data (encryption, compression, machine-specific conventions)
    - Session : synchronization(同步), checkpointing, recovery of data exchange
    - Now these layers implemented in application layer if needed
- The graph of layers

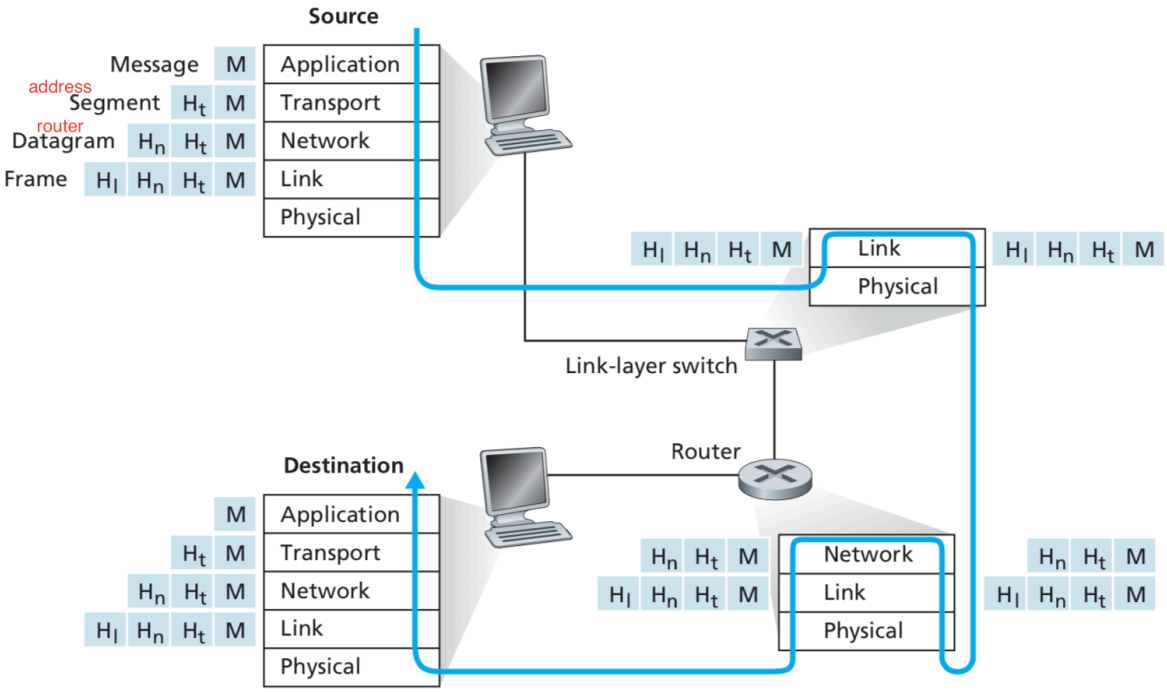


Figure 1.24 ♦ Hosts, routers, and link-layer switches; each contains a different set of layers, reflecting their differences in functionality

- Only destination delete header, only source add header , only this two have 5 layers
- switch only has 2 layers(do not know IP protocol), routers only have 3 layers
- Host(end systems) implement all five layers, (Internet architecture puts much of its complexity at the edge of the network(网络的最边缘， 不是图论中的edge))

- Encapsulation
  - Sending host: An **application-layer message** is passed to the transport layer
  - Transport layer: Take the messages and appends additional information(*Transport-layer header*) that will be used by teh receiver-side transport layer

$$(application-layer\ information) + (transport-layer\ header) = (transport-layer\ segment) \quad (4)$$

  - Network Layer: Then every layer add its header
- Benefit of Layering
  - explicit structure allows identification, relationship of complex system's pieces
    - layered reference model for discussion
  - Modularization eases maintenance, updating of system
    - change of implementation doesn't affect others
    - e.g. change in gate procedure doesn't affect rest of system

---

Tutorial 2

	Circuit Switching	**Packet Switching**
Resource Usage	Reservation	Share the resources
Performance	stable	Unpredictable (congestion may happen) (delay and loss)
Set up	Needed	Not needed
Multiplexing (多路传输)	TDM/FDM	Divide the whole file into packets => statistical multiplexing

- For long-term steady rate transmission Circuit switching is better
- Skype offers a function to have phone call between computers and phones (connect computer network with telephone network) How?

  Skype app ⇐ Packet switching (Internet (packets))⇒ Gateway ⇐ Circuit switching(Telephone net (stream)) ⇒ Phone
- How long does it take a packet of length 2000 bytes to propagate over a link of distance 2000 km, propagation speed $2 * 10^8 m/s$, and transmission rate 2Mbps(it is **bits**)?
  - Transmission delay : time spend for **push data to link**
  - Propogation delay : time spend **on the link**

$$td = \frac{L}{R} = \frac{2000 * 8}{2 * 10^6} = 0.008s \tag{5}$$

$$pd = \frac{2 * 10^6}{2 * 10^8} = 0.01s \tag{6}$$

- Suppose that users share a 10Mbps link, i.e., they all send traffic to a node which has a 10Mbps link to forward the traffic received from the users. Suppose that each user transmits continuously at 5Mbps when transmitting, but each user transmits only 20% of the time.
  - When circuit switching is used, how many users can be supported? => 2
  - Suppose that there are **4** users and packet switching is used. What is the fraction of time that the queue at the node is not empty?

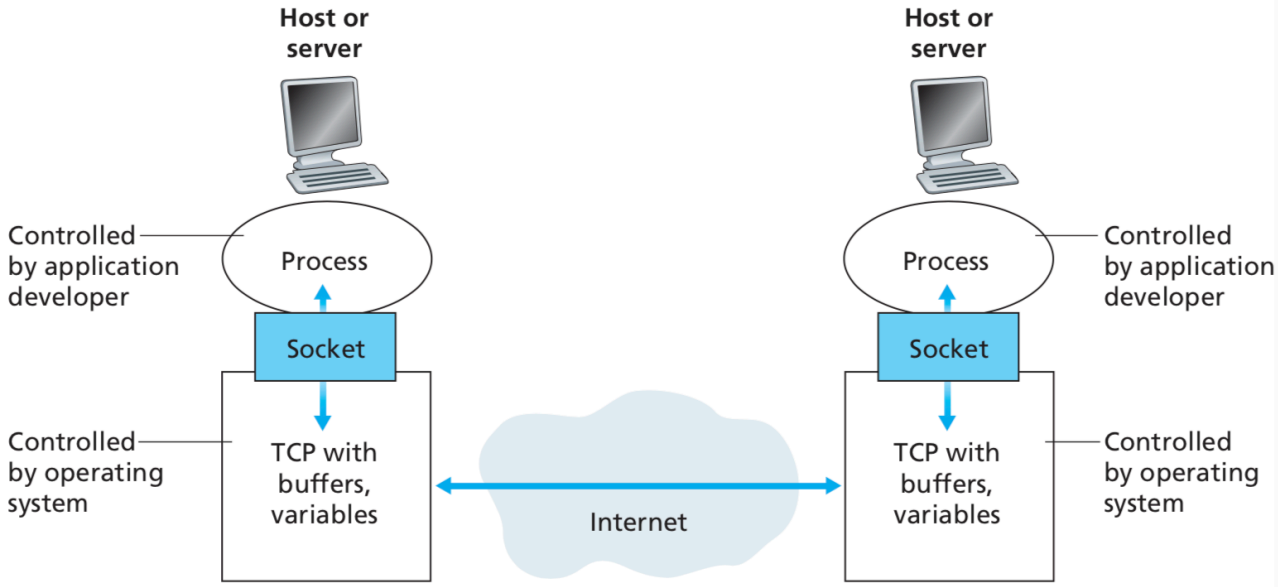
$$p = C_4^3 * 0.2^3 * 0.8 + C_4^4 * 0.2^4 \tag{7}$$

Chapter 2 Application Layer

2.1 Principles of Network Applications

- creating an internet app
  - program on different ending systems(all end systems can use it)
  - no need to write software for network-core devices
- Application architecher
  - p2p (pear to pear)
  - Client - Server
    - Server
      - always on host
      - **permanent IP address** (not local IP address)
      - data centers for scaling
    - Client
      - call communication first(definition of client)
      - may be intermittently(间隔) connected
      - may have dynamic IP addresses
      - do not communicate directly with each other
    - Data center:
      - Single server can't response to all the requests
      - data center housing a large number of hosts, often used to create a powerful virtual server
- P2P
  - no always-on server (everyone service each other)
  - arbitary ending systems directly connected
  - No or minimal reliance on dedicated server
  - peers are intermittently connected and change IP addresses

        - complex management
    - Feature : **self-scalability**(可扩展性)
- Process communicating
    - Process : program running within a host (ending systems)
        - within the same host, two processed using *interprocess communcation* (defined by *OS*)
        - processes in different hosts communicate by **exchanging messages**
        - Include client process(initialize communication) and server process
- **Interface between the process and the computer network**



    - A process sends messages into or receives messages from the network through a software interface called **socket** (also called **API** between applications and network) (multiplication)
    - Control by the application developer
        - (1) the choice of transport protocol
        - (2) perhaps the ability to fix a few transport-layer parameters such as maximum buffer and maximum segment sizes (to be covered in Chapter 3).
    - Address processes : for client to match message returned and process or server to match process and coming message
        - Unique IP address for identifying client
        - *identifiers* includes both **IP address** and **port numbers**
        - for e.g. the port 80 is always for HTTP server (only on server side), 25 for mail server

2.1.3 Transport Services Available to Applications

- On the receiver side, transport layer send the message to the application layer (demultiplexing)
- Transport-layer protocols is determined by the application layer needs
    - Reliable data transfer
        - To transfer data that affected a lot by loss of bits (e-mail, file transfer, remote host access, financial applications)
        - e.g. process-to-process reliable data transfer (otherwise, **loss-tolerant applications**)
    - Throughput
        - Other sessions are sharing bandwidth with current task, the avaliable throughput can fluctuate with time
        - Another natural service : Guaranteed available throughput at some specified rate => **bandwidth-sensitive applications** (otherwise, **elastic applcations**)
    - Timing
    - Security

Application	Data Loss	Throughput	Time-Sensitive
File transfer/download	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	No loss	Elastic (few kbps)	No
Internet telephony/ Video conferencing	Loss-tolerant	Audio: few kbps–1Mbps Video: 10 kbps–5 Mbps	Yes: 100s of msec
Streaming stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps–10 kbps	Yes: 100s of msec
Instant messaging	No loss	Elastic	Yes and no

2.1.4 Transport Services Provided by the Internet

- TCP services
  - Connection-oriented service
    - TCP has the client and server exchange transport-layer control information with each other *before* the application-level messages => hand shaking
    - TCP connection is between the sockets of the two processes can send messages to each other over the connection at the same time
  - reliable data transfer service
    - Deliver all data sent without error and in the proper order
    - When one side of the application passes a stream of bytes into a socket, it can count on TCP to deliver the same stream of bytes to receiving socket
  - Congestion-control mechanism(抗阻塞机制)
    - For the welfare of the whole Internet rather than for the direct benefit of the communicating processes
- UDP services
  - UDP is a no-frills(没多余装饰) lightweight transport protocol => no hand shaking, unreliable transfer, data may out of order, no congestion-control mechanism

2.1.5 Application-Layer Protocols

- Application-layer protocol define how an **application's processes, running on different end systems, pass messages to each other**. It defines:
  - The **type** of messages exchanged, for example, request messages and response messages
  - The **syntax** of the various message types, such as the fields in the message and how the fields are delineated(划定)
  - The **semantics of the fields**, that is, the meaning of the information in the fields
  - Rules for determining when and **how** a process sends messages and **responds** to the message (e.g. HTTP, DNS，Skype)

2.2 The Web and HTTP

- unlike traditional broadcast radio and TV, users receive what they want, when they want it => force user to tune in(收听) when the content provider makes the content available

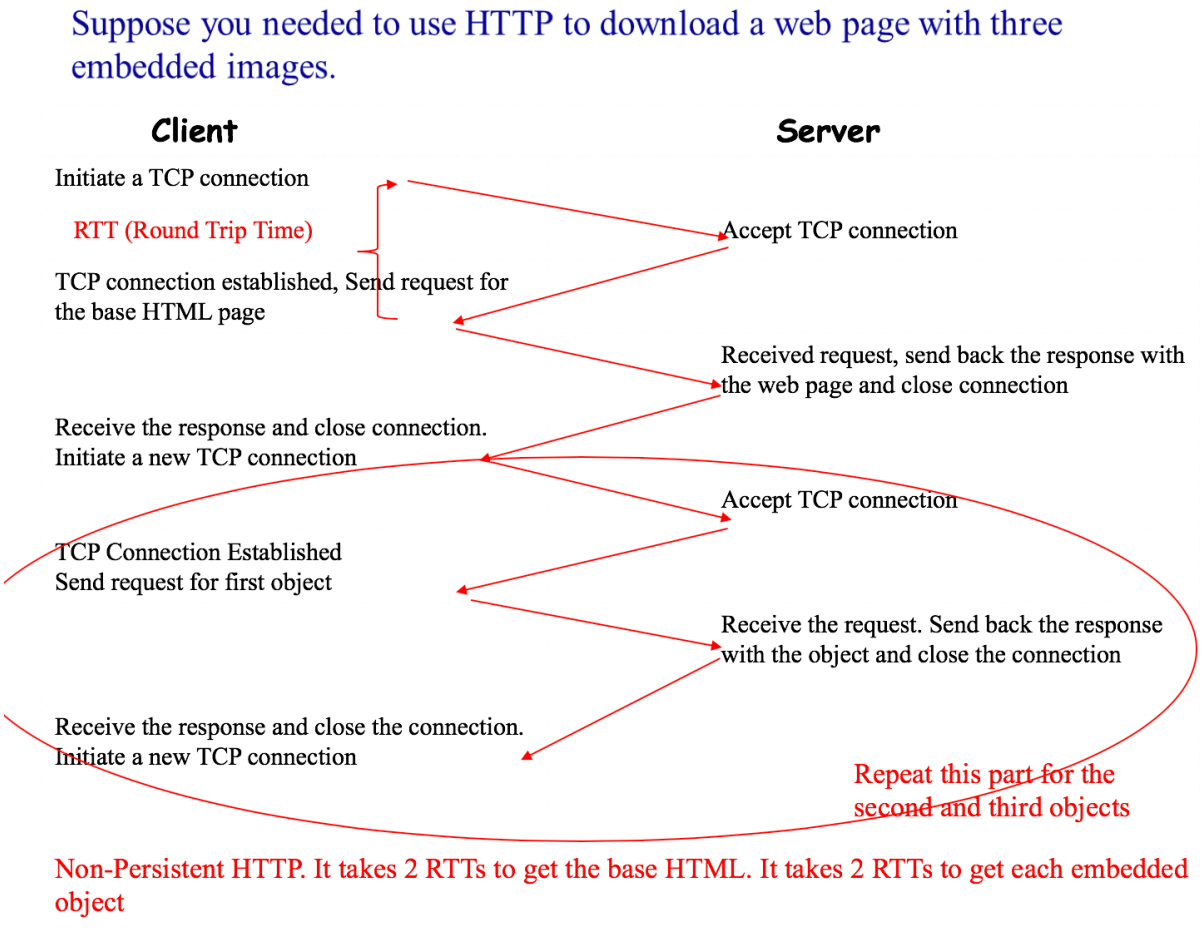
2.2.1 Overview of HTTP

- Web overview
  - Web and HTTP
    - web page consist of referenced objects which are based on *HTML* file
    - Each object is addressed by a URL (Every objects has its URL refered by HTML)

- HTTP: Web's application layer protocol => **HyperText Transfer Protocol** implementing 2 programs
  - Client program : browser (using HTTP protocol and "displays" web objects)
  - server program: Web server sends (using HTTP protocol) objects in response to requests
  - HTTP based on TCP :
    - Client initiates connection
      - client sends HTTP request messages into its socket interface and receives HTTP response messages from its socket interface
      - When message is sent into its socket interface, it is out of hand of HTTP or client's hand, it is in control of TCP => HTTP don't have to warry about how data loss is recovered or bits are reordered, because that is job of TCP(like OOP) => advantage of layered architecture
    - Server accepts TCP connection from client
      - Server sends requested files to clients without storing any state information about the client (don't store what it has done) => if a client ask for the same object twice, server will do it => HTTP is **stateless protocol**
    - HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
    - TCP connection is closed

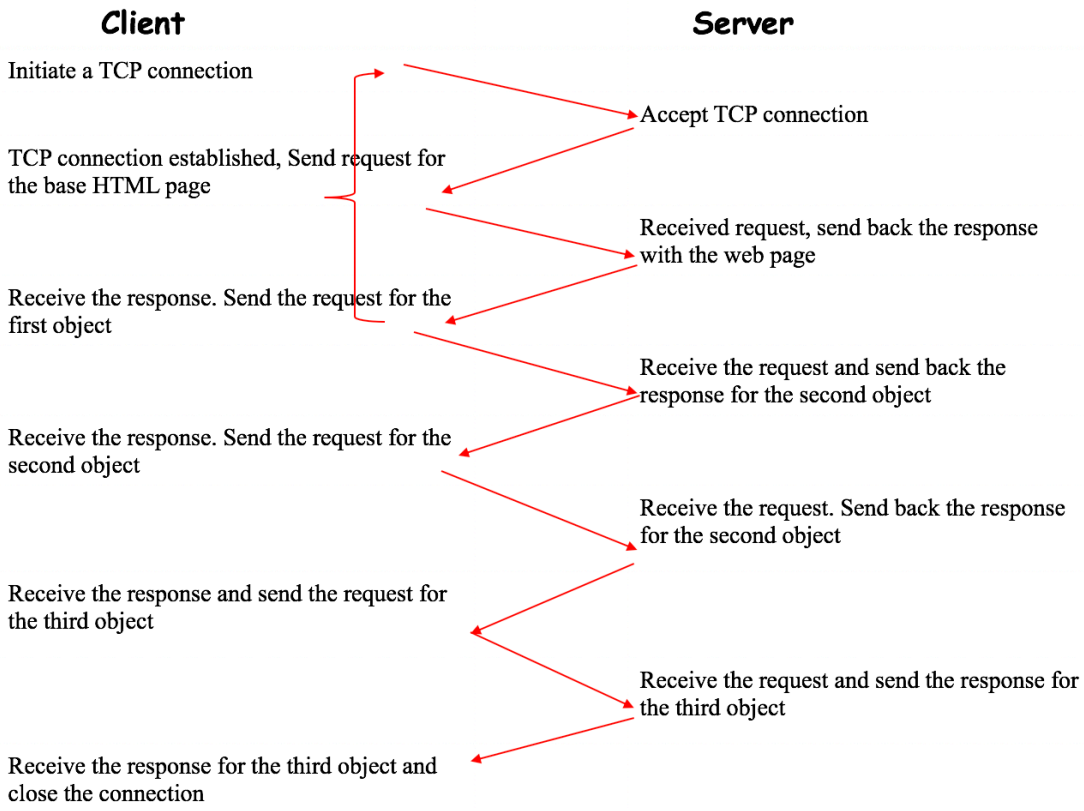
2.2.2 Non-Persistent and Persistent Connections

- HTTP connections
- HTTP : default: persistent connection; runnable : both non-persistent and persistent
  - Non-persistent connection=>each request sent over a *separate* TCP connection



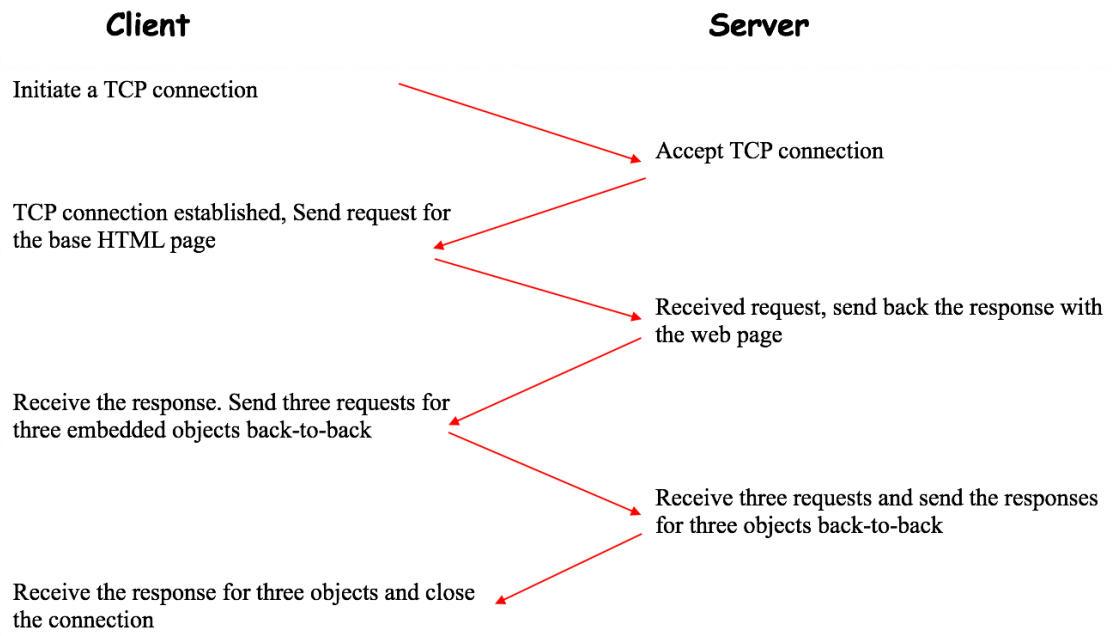
  - steps
    - client initialize connection, 2 sockets(1 client 1 server) established
    - client sends an request message to the server via its socket which includes the path name
    - Server process receives the request message via its socket, retrieves the object, *encapsulates* the object in an HTTP response message, and sent back the message via its socket
    - HTTP server process tells TCP to close the TCP connection (TCP doesn't actually close until it knows client has

received the information)
        - client receive the object (HTML) and ask for reference object
        - repeat the steps above until get all the objects (parallel TCP connections are allowed)
    - at most one object sent over a TCP connection
        - The connection is then closed
    - downloading multiple objects required multiple connections
        - If we do not consider the transmission time, it takes two round trip time (i.e. **from client to server then back to client**) (RTT) to get each object (one for object, one for close TCP link request)
        - HTML : 2 RTT(for connection and request for object) +transmission of the HTML file Object : 2 RTT
        - All : 2*(n+1) at where n is the linked object
- Persist HTTP : Multiple objects can be sent over a single TCP connection between client and server
    - without pipelining



Persistent http without pipeline. It takes 2 RTTs to get the base HTML. One RTT for each embedded object.

        - Client issues new request only when the previous one has been received
        - HTML : 2 RTT(TCP+HTML) Each Object : 1 RTT
        - All : 2 + 1*n(objects)
    - with pipelining



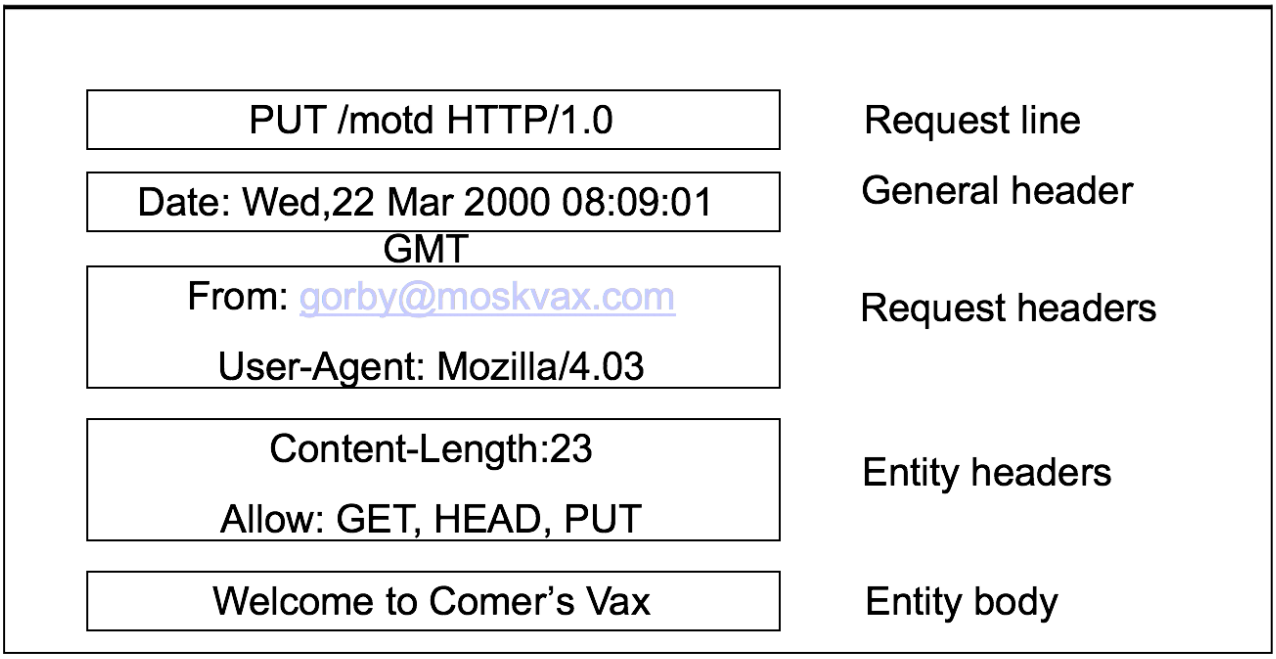
Persistent http with pipeline. It takes 2 RTTs to get the base HTML. One RTT to get ***all*** embedded objects.

- Multiple Web pages residing on the same server can be sent from the server to the same client over a single persistent TCP connection
- Client issues new request as soon as it encounters a reference object
- All : As little as 1 RTT for all the referenced objects (send all objects at the same time) but still 1 RTT for HTML to get the link i.e. **3 RTT in total**

2.2.3 HTTP Message Format

- Two types of message : request messages and response messages
- HTTP Headers
  - General header
  - Request/Response header
  - entity header
- Request message (five lines)

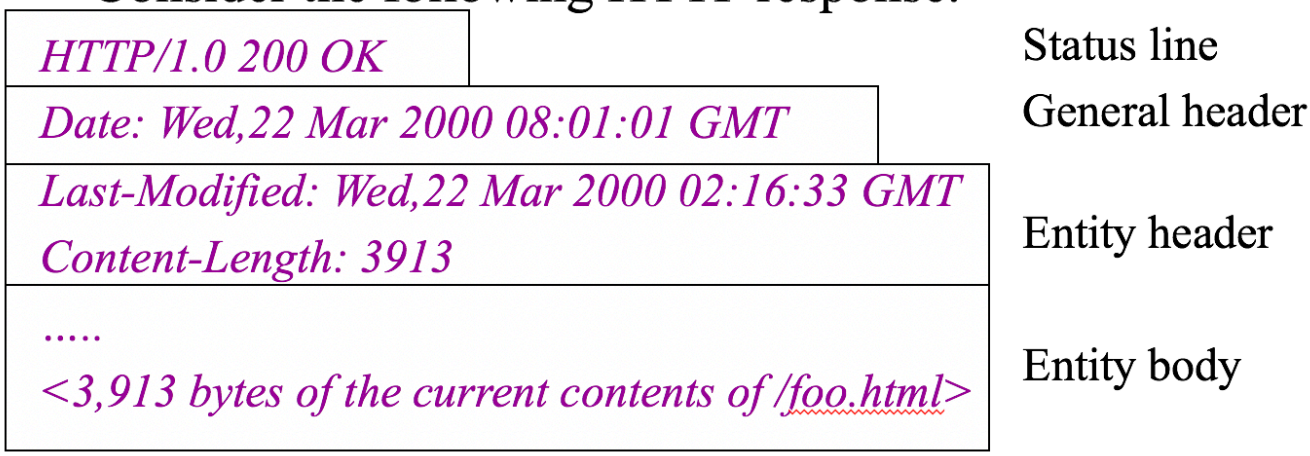
HTTP Request Format



- Request Methods supported
  - POST : Send data to the server (for e.g. use HTML forms)
  - GET : Request data from a specified resource
- HTTP Response message

HTTP Response

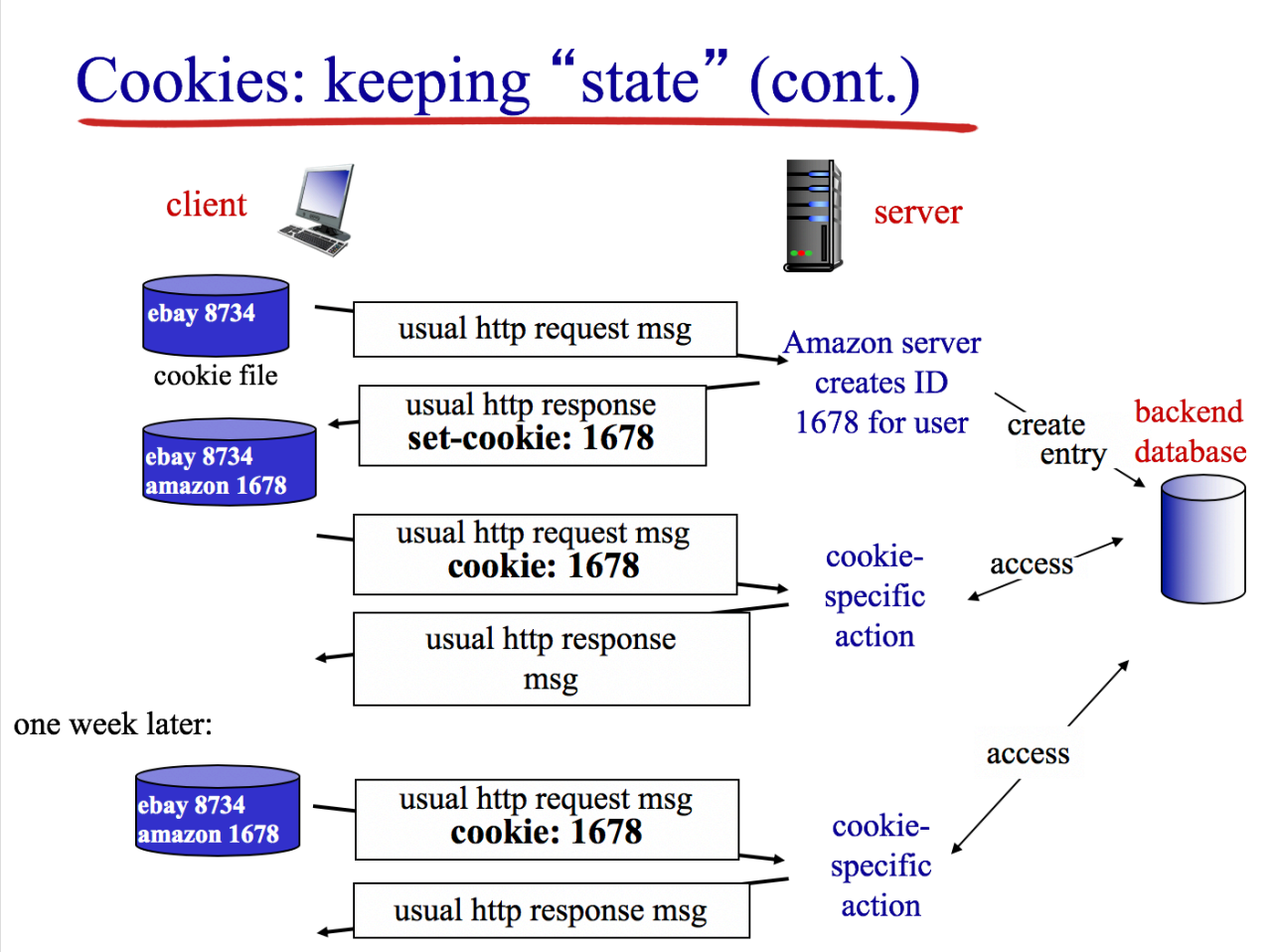
- Consider the following HTTP response:



- Status-line (The first line)
  - HTTP version number
  - Status code (indicating success or failure) (200 OK, 404 Not Found)
  - Status phrase (OK, Not Found)
- General/Response/Entity Header (s)
  - Date
  - Last-Modified
  - Entity headers (propoty of enetity)

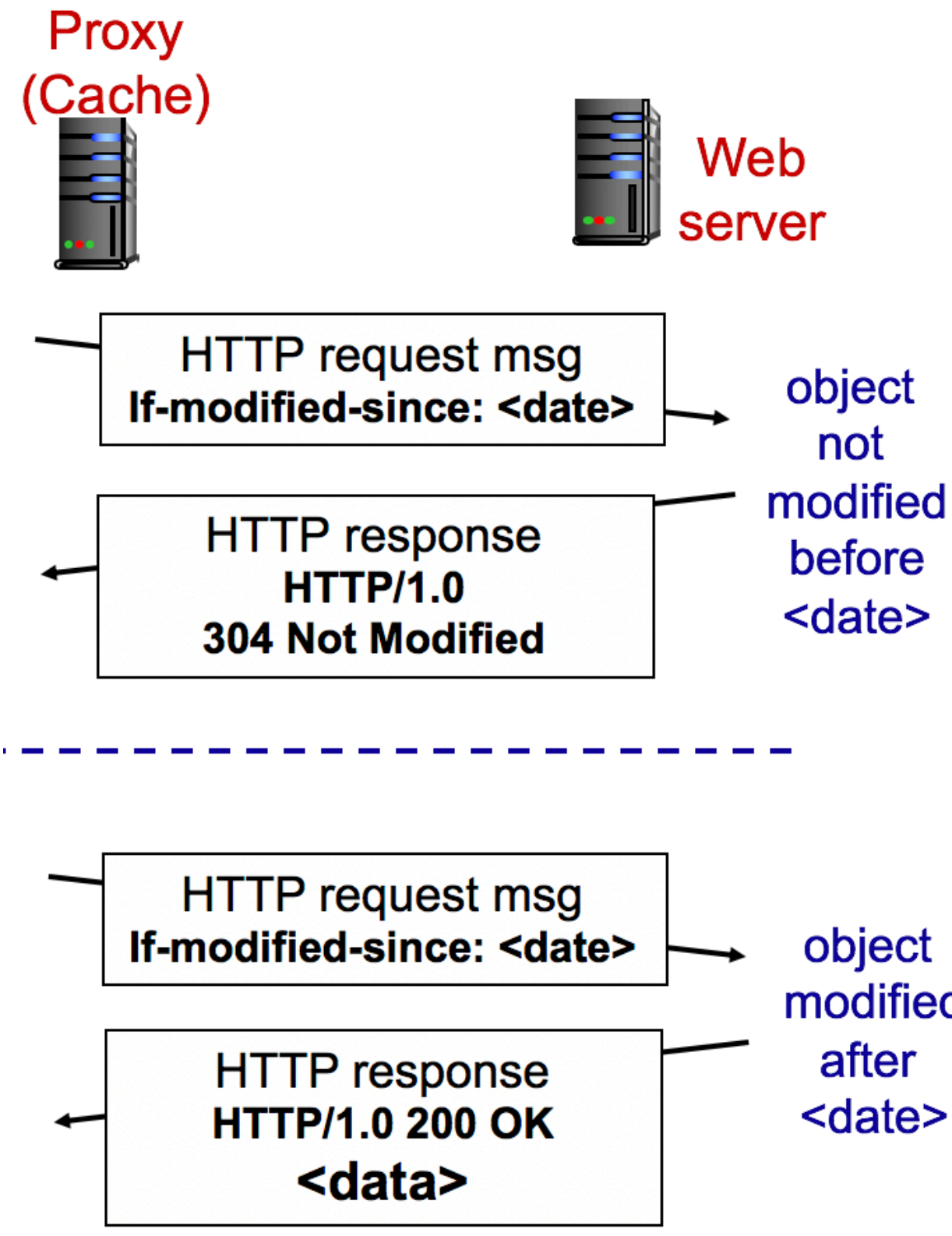
        - Content-Length
        - Content-Type
    - Entity body : Object itself

- Cookies
    - Allow sites to keep track of users
    - small chunk of data generated by Web server and stored on computer's hard disk
    - **Fix problems caused by HTTP's stateless**(server doesn't remember the information of client) protocol
    - Components:
        - a cookie header line in the HTTP response message
        - a cookie header line in the HTTP request message
        - a cookie file **kept on the user's end system** and **managed by the user's browser**
        - there is a **back-end database** at the Web site
    - Can be used:
        - authorization(auto-login)
        - shopping carts
        - recommendations
        - user session state(web e-mail)
    - Steps



        - First time of connect :
            - When request comes in to server, server creates a unique identification number and **creates an entry**(the cookie file) in its back-end database that is indexed by the identification number. Response message includes a `Set-cookie:` header, which contains the number
            - when browser receives the HTTP response message, it sees the `Set-cookie:` header. Then browser appends a line(include the hostname of the server and the identification number in the header) to the special cookie file that it manages
        - After that

        - Every time browser request, it will consults her cookie file, and add the identification number in HTTP request
        - If client has sign up an account on the Web, thus the database has client's information, then everytime it will check the identification number with the database, user don't have to input the account and password again
- Web cache
    - Goal: statisfy client requests without involving the origin server
    - save the data at proxy server (distributed around the world) , client get the data from the closest server
    - User sets browser: Web accesses via cache (i.e. TCP connection to the Web cache)
    - Browser sends all HTTP requests to cache
        - object in the cache: the cache returns object
        - Otherwise the cache requests the object from the origin server, then returns the object to the client and store the object into the Web cache's local storage
    - If **origin updated** (different with object not exist The conditional GET)



- Goal : don't send the object to the cache if cache has up-to-date version
  - no object transmission delay (less than the delays)
  - lower link utilization
- Request message uses the GET method and includes an `If-Modified-Since` : header line
- when proxy server(cache) receive request it will check with origin server(just 版本号)

```
If-modified-since : Wed, 19 Sep 2018 04:09:30 //Date
```

- cache: **specify date of cached copy** in HTTP request

        - server: response contains no object if cached copy is up-to-date: `304 Not Modified`

2.5 DNS - The Internet's Directory Service : belongs to Application layer protocol

- DNS : domain name system
    - IP address (32 bit) - used for addressing datagrams
    - domain name(**hostname**) => used by human
    - DNS : Directory service that **translates** hostnames to IP addresses => **domain name system**
        - property
            - distributed **database**(stores the information) implemented in hierarchy of many **DNS servers** (often **UNIX** machines): to avoid attack(if it is save in a single one, will be attacked) => block chain to avoid attack
            - Application-layer protocol: hosts and name servers communicate to resolve names (address/name translation)

2.5.2 Overview of How DNS Works

- Structure
    - If there is only one DNS server
        - A single point of failure
        - Traffic volume
        - Distant centralized database
        - Maintenance (DNS server has to keep records for all Internet hosts)
    - As a result => use distributed, hierarchical database
        - **Root DNS servers** : In the Internet there are 13 root DNS servers(labeled A through M) in the world
        - **Top-level domain (TLD) servers** : These servers are responsible for top-level domains such as com, org, net, edu, and gov
        - **Authoritative DNS servers** : Every organization with publicly accessible hosts (such as Web servers and mail servers) on the Internet must provide publicly accessible DNS records that map the names pf those hosts to IP addresses (server can be rent or maintain one server yourself)

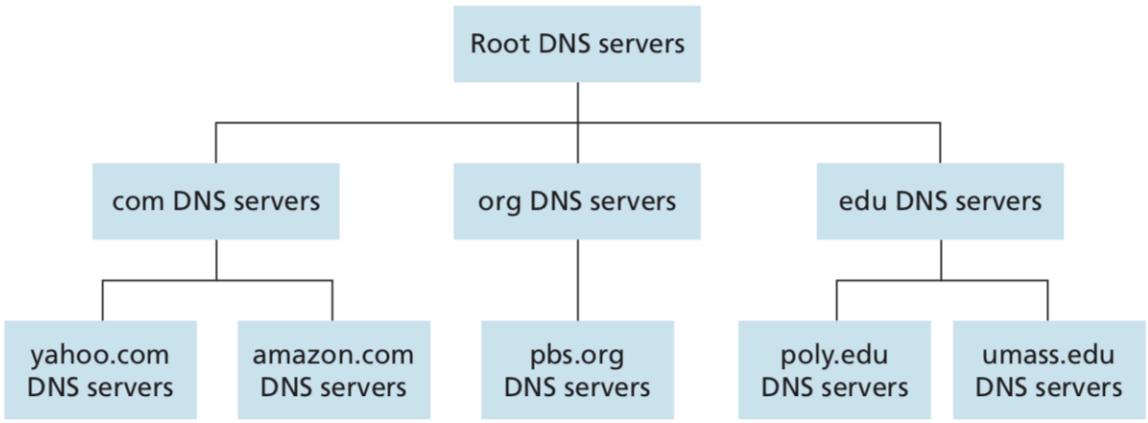
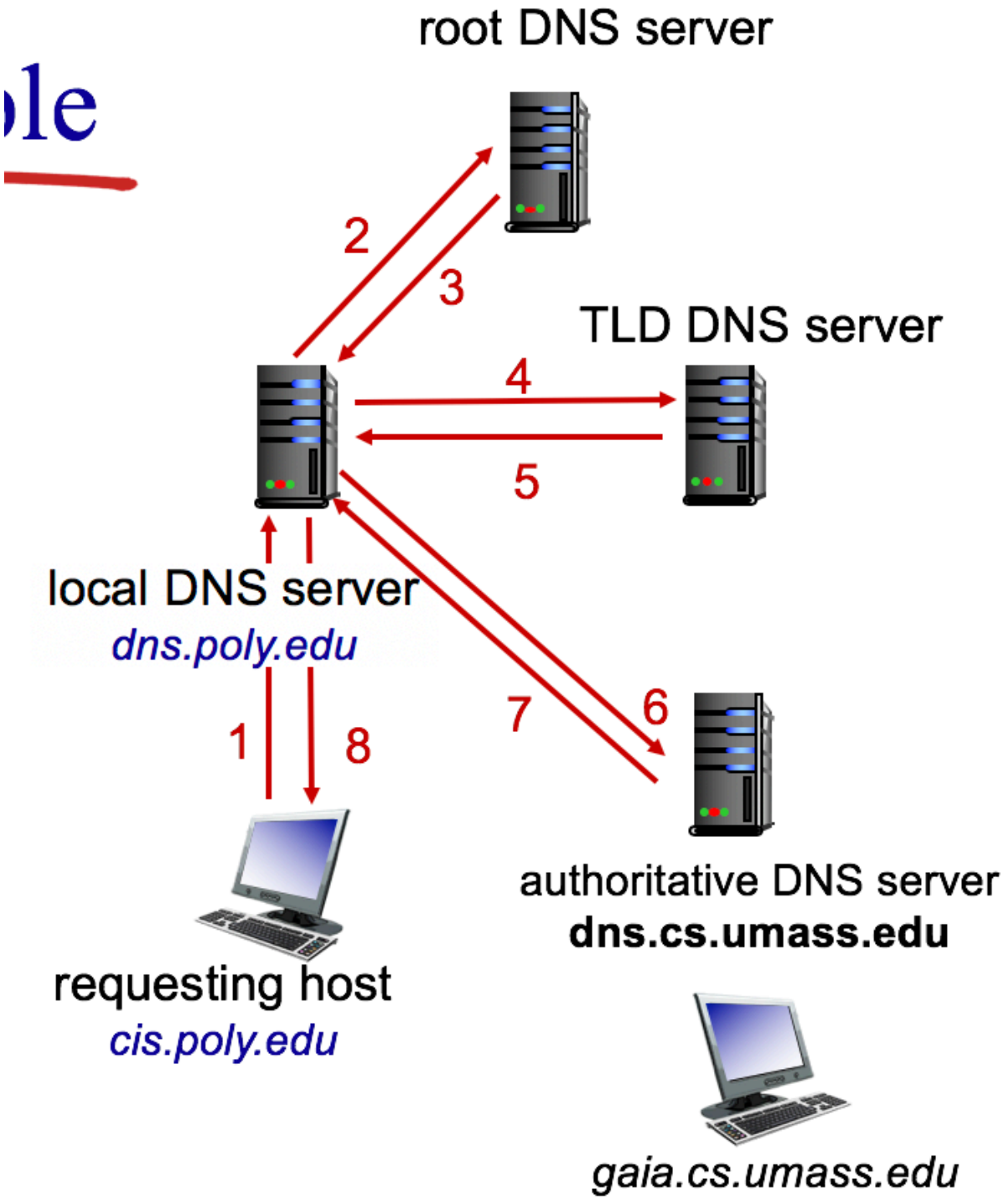


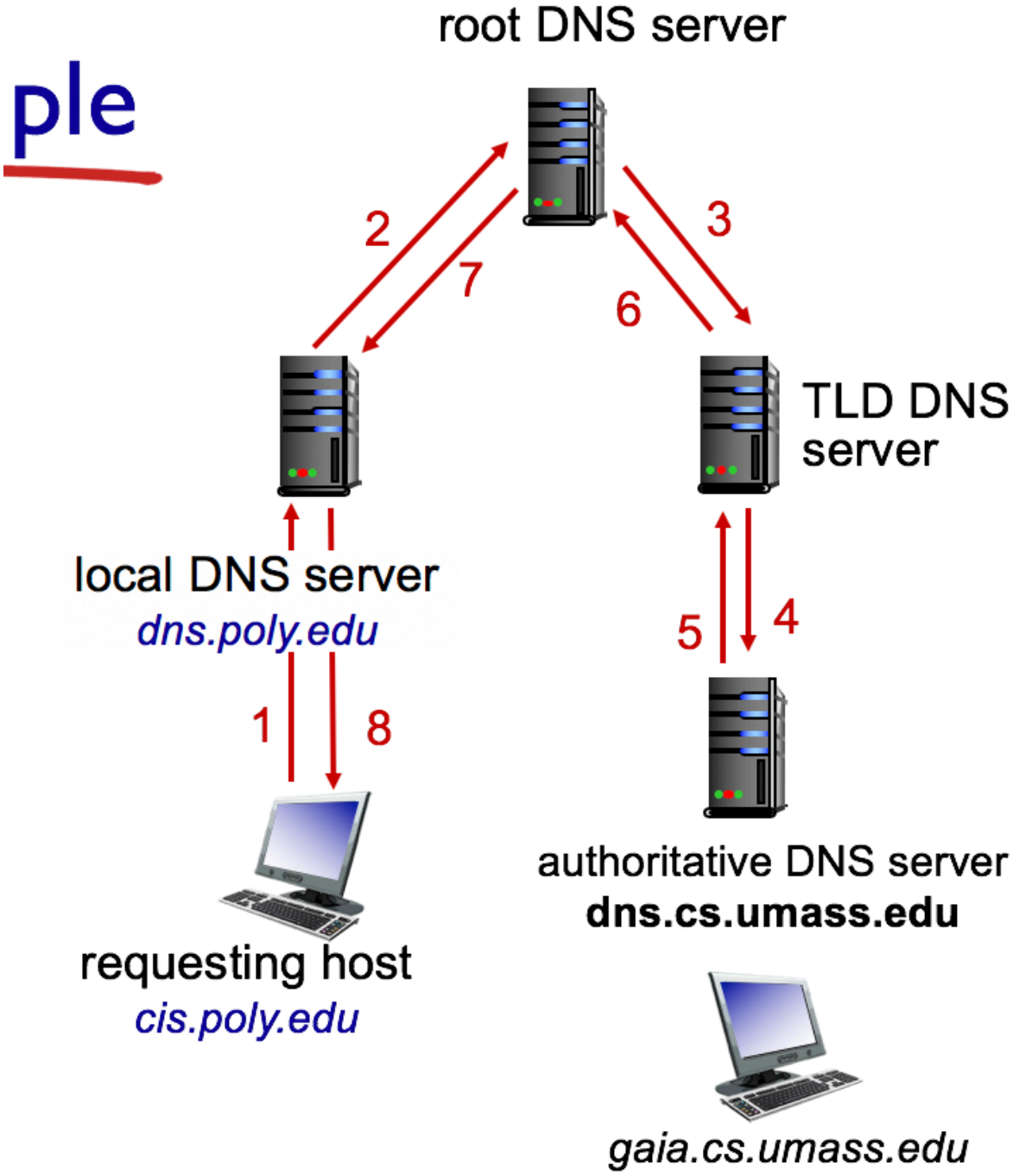
Figure 2.19 ♦ Portion of the hierarchy of DNS servers

- Process : client wants IP for www.amazon.com; (All DNS in the client use UDP transport protocol)
    - DNS client queries root server to find com DNS server
    - client queries .com DNS server to get amazon.com DNS server
    - client queries amazon.com DNS server to get IP address for www.amazon.com (it is a web server www rather than an e-mail server)
- Root DNS servers:
    - contacted by local name server that can not resolve name : nearest
    - root DNS server(向导):

        - contacts authoritative name server if name mapping not known
        - gets mapping
        - returns mapping to local name server
    - Local DNS name server
        - not the hierarchy but it is the one who send the query (beween DNS client and the **DNS Hierarchy**)
        - **each ISP(Internet Service Provider) has one**
        - when host makes DNS query, query is sent to its local DNS server
            - has local cache of recent name-to-address translation pairs (but may be out of date!)
            - acts as proxy(代理人), forwards query into hierarchy



- Iterated query: (the picture above 1&8 is recufsive query, others are Iterated query)
  - contacted server replies with name of server to contact
  - “I don’t know this name, but ask this server”
  - Cons : local DNS server will not benefit for others
- recursive query:
  - deep first search
  - **heavy load at upper levels** of hierarchy



2.6 Peer-to-Peer Applications

2.6.1 P2P File Distribution

- Scalability of P2P Architectures (Not tested)
  - Denotes
    - Denote upload rate of the server's access link by u_s
    - Denote upload rate of ith peer's access link by u_i
    - Denote download rate of the ith peer's access link by d_i
    - Denote the size of the file to be distributed (in bits) by F and the number of peers that want to obtain a copy of the file by N
    - **Distribution time**: time it takes to get a copy of the file to all N peers

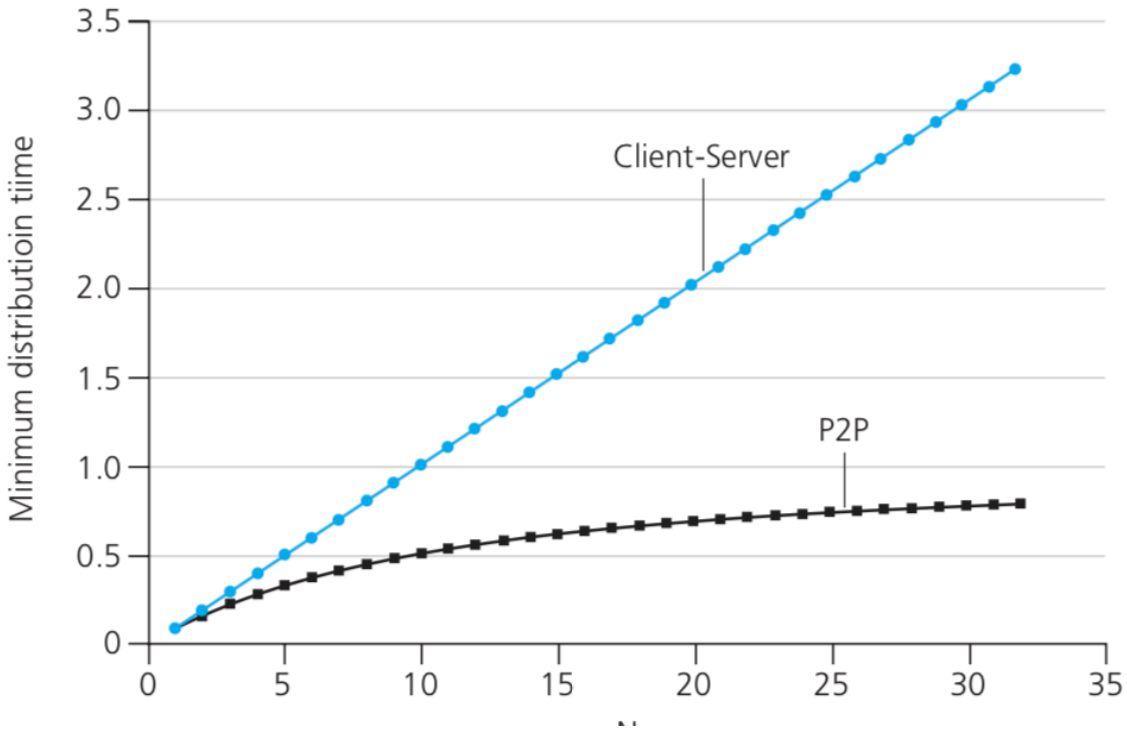
- Assumption
  - Internet core has abundant bandwidth => all of the bottlenecks are in access networks
  - Server and clients are not participating in any other network applications => all bandwidth used to distributing this file
- Distribution time : D_{cs}
  - For Client-Server
    - The server must transmit one copy of the file to each of N peers => total NF bits => rate = $\frac{NF}{u_s}$
    - $d_{min} = min(d_1, d_2, \ldots, d_N)$ The peer obtain all F bits of the file in less than $\frac{F}{d_{min}}$ seconds

$$D_{cs} \geq max\{\frac{NF}{u_s}, \frac{F}{d_{min}}\} \tag{8}$$

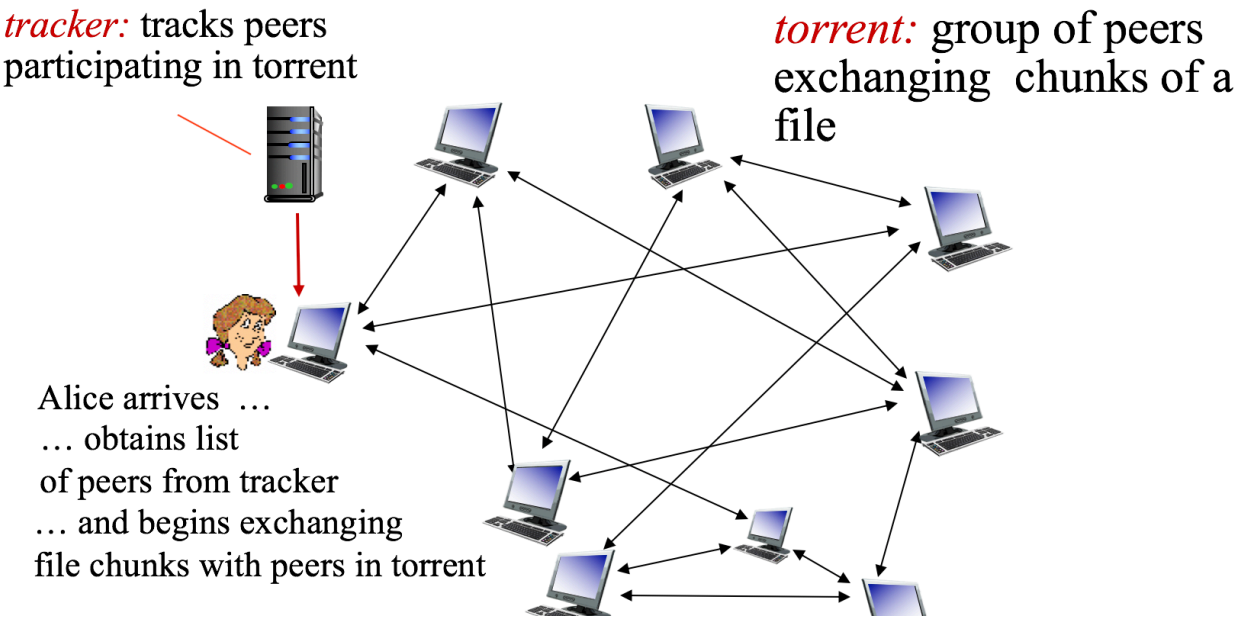
    - We take the lower bound (i.e. the equal case , best case)
    - The formular increasing linearly (with respect to N)
  - For P2P architecture
    - When a peer receives some file data, it can use its own upload capacity to redistribute the data to other peers => distribution time depends on how each peer distributes portions of the file to the other peers
    - Server need to send each bit of the file at least once into its access link (only server has the file at first) => minimum distribution time is at least F/u_s
    - The peer with the lowest download rate cannot obtain all F bits of the file in less than F/d_{min} seconds => minimum $\frac{F}{d_{min}}$
    - Total upload capacity of the system as a whole is equal to the upload rate of the server plus the upload rates of each of the individual peers $u_{total} = u_s + u_1 + \ldots + u_N$ (every one contributes to get the largest rate). And the total file size is NF, thus the minimum time is $\frac{NF}{u_{total}}$.

$$D_{cs} = max\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\} \tag{9}$$

    - In reality, where chunks of the file are redistributed rather than individual bits, the equation is a good approximation
  - Set $F/u = 1\ hour,\ u_s = 10u,\ and\ d_{min} \geq u_s$



- Pure P2P architecture



- file divided into 256Kb chucks
- peers in torrent send/receive file chunks
- server just tell you who has what, and people change with each other
- **tracker** : **tracks peers participating** in torrent (provide a list to client)
- **torrent**: group of peers exchanging chunks of a file
  - each torrent has a infrastructure node called a *tracker*

- P2P file distribution
  - BitTorrent： In BitTorrent lingo(行话), the collection of all peers participating in the distribution of a particular file is called a *torrent* (include the whole group)
    - peers in a torrent download equal-size *chunks*(256KB) of the file from another
    - peer joining torrent
      - has no chunks, but will accumulate them over time from other peers
      - registers with tracker to get list of peers (tracker **randomly** selects a subset of peers, and send IP address to request peer), connects to subset of peers (“neighbors”)
    - requesting chunks
      - Ask neighbors(the subset) for the list of chunks they have (By TCP connections)
      - Alice requests missing chunks from peers, **rarest first**, because no one always online => as a result equalize the copies of the different chunks
    - Sending chunks: tit-for-tat(以牙还牙)
      - sends to those top four peers (**unchoked** 非阻塞) currently send you *at highest rate* (re-evaluate 4 every 10 secs)
      - every 30 secs : randomly select another peer (**optimistically unchoked**), starts sending chunks (for the case of new client who didn't serve others)

2.7 Socket Programming(not tested)

- Initializing the connection
  - First, as in the case of UDP, the TCP server must be running as a process(Listen) before the client attempts to initiate contact.
  - Second, the server program must have a special door—more precisely, a special socket—that welcomes some initial contact from a client process running on an arbitrary host.
- Some java functions and classes

(1)Connecting to a server using java

```
Socekt s = new Socket("ip.address",port#);
s.getInputStream();
InputStream in = s.getInputStream();    // read from socket
OutputStream out = s.getOutputStream();   // write to socket
```

(2) Implementing Server

```
ServerSocket s = new ServerSocket(port#);
Socket = s.accept();

```

Chapter3 Transport layer

Relation between Transport layer and network layer:

- Transport layer **extending the network layer's delivery service** between two end systems to a delivery service between **two application-layer processes** running on the end systems

3.1 Introduction and Transport-Layer Services

- A transport-layer protocol provides for **logical communication**(from an application's perspective, it is as **if the hosts running the processes** were directly connected, it doesn't worry about the physical infrastructure used to carry these messages) between application processes running on different hosts
  - package on transport-layer : **segments** = chunks + transport-layer header
  - Rooter act only on datagram (segments+network-layer header), i.e. it doesn't check the transport-layer header
  - transport layer protocol connect the processes
  - Main tasks:
    - Multiplexing, demultiplexing
    - reliable data transfer
    - flow control
    - congestion control

3.1.1 Relationship Between Transport and Network Layers

- Transport-layer protocol provides logical communication between *processes* running on different hosts, a network-layer protocol provides logical communication between *hosts*.
  - e.g. 2 house, each has a dozen kids, kids from different house to each other, Alice and Bob are person who receive the mail and distribute to other kids in the 2 houses respectively, then
    - application message : mail in envelopes
    - Process : kids
    - Hosts : house
    - transport-layer protocol : Alice and Bob
    - Network-layer protocol : postal service (including mail carriers)
  - So transport-layer protocols only lives in end systems

3.1.2 Overview of the Transport Layer in the Internet

- Two protocol
  - **UDP** : User Datagram Protocol => unreliable
  - **TCP** : Transmission Control Protocol => reliable
    - Congestion&flow control
    - connection setup
- **IP** (Internet protocol)
  - Provides logical communication between hosts
  - a **best-effort delivery service** : makes best effort to deliver segments between 2 communcating hosts, *but makes no guarantees*(order and segment delivery) => **unreliable service**
- Transport protocol
  - sender side: breaks app messages into segments, passes to network layer => **transport-layer multiplexing**
  - receiver side: reassembles segments into messages, passes to app layer => **transport-layer demultiplexing**

3.2 Multiplexing and Demultiplexing

- Transport-Layer protocol deliver data to socket instead of delivering data to application directly (There is more than one socket in the receiving host, more than one socket for a single process, and each socket has a unique identifier) => format of the identifier depends on whether the socket is a UDP or TCP socket
- Multiplexing/demultiplexing
  - Multiplexing at sender : handle data from multiple sockets, break data to segment, add transport header (encapsulation)
  - Demultiplexing at receiver : (Transport layer) use header info to deliver received segments to **correct socket** (UDP use less information to demultiplexing)
    - each datagram contains one segment=>one IP address and one port number
  - Transport-layer in the middle host in the picture must **demultiplex segments arriving from the network layer** below to either process P1 or P2 above (done by directing the arriving segment's data to the corresponding process's socket), and must also **gather outgoing data from these sockets**
  - when a **single protocol** at one layer (transport layer or other layers) is used by **multiple protocols** at the next **higher** layer (底层一个对应上层多个), it is called multiplexing
- Port number
  - multiplexing requires:
    - sockets have unique identifiers
    - each segment have special fields that indicate the socket to which the segment is to delivered
  - Source port number field and destination port number field
  - Well-known port numbers : 0-1023 (used by well-known application protocols such as HTTP , FTP)
  - Typically, the client side of the application lets the transport layer automatically (and transparently) assign the port number, whereas the server side of the application assigns a specific port number
- Connectionless Multiplexing and Demultiplexing (UDP)
  - Use 2 number to generate socket at server side(destination IP address and destination port number) => maybe two different client ask for the same socket at the server side (connectionless)
  - Source port number : used as a return address
  - if segement from **2 different client IP addresses** have the **same dest IP and port number**, such segments will be directed to the **same process**
- Connection-Oriented Multiplexing and Demultiplexing (TCP)
  - four numbers to spercify the process (source IP, source port, dest IP, dest port) (connection oriented)
  - two arriving TCP segments with different source IP addresses or source port numbers will (with the exception(除此以外) of a TCP segment carrying the original **connection-establishment** request) be directed to 2 different sockets
  - Further insight
    - There is a "welcome socket" waits for connection-establishment requests from TCP clients on port number 12000

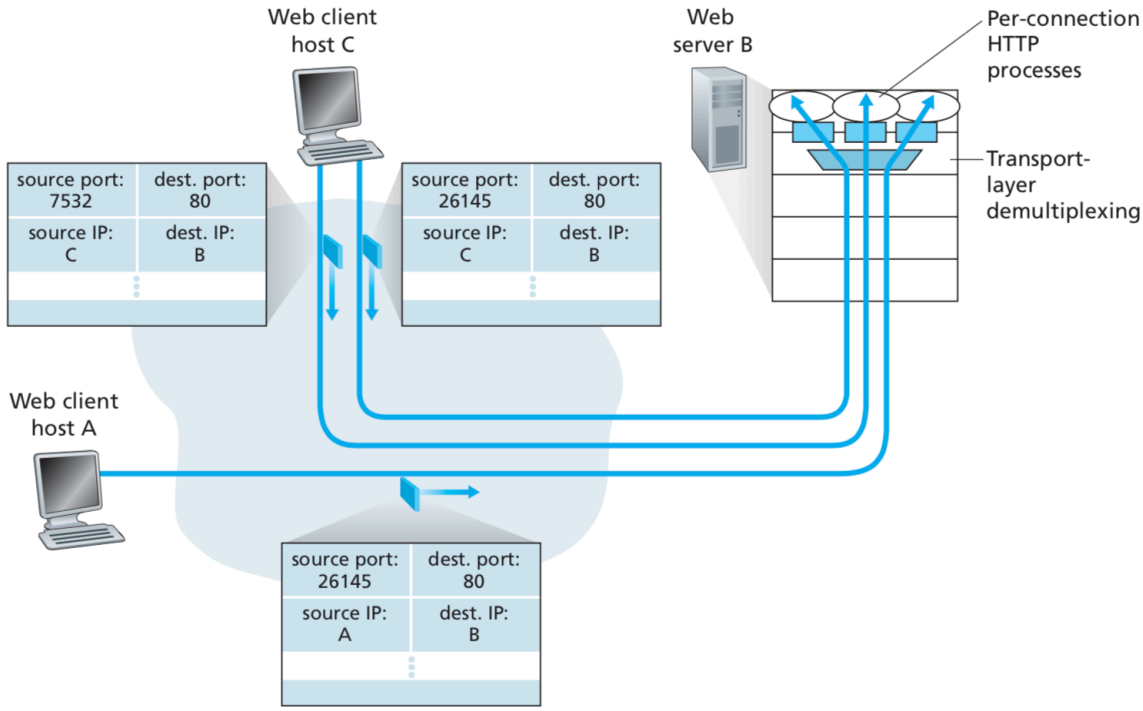
- The TCP client creates a socket and sends a connection establishment request segment

```
clientSocket = socket(AF_INET,SOCK_STREAM)
clientSocket.connect((serverName,12000))
```

- the request consist of :
  - destination port number 12000
  - a **special connection-establishment bit**(syn) set in the TCP header
  - source port number chosen by client
- The server receives the incoming connection-request segment with destination port 12000, and creates a new socket

```
connectionSocket, addr = serverSocket.accept();
```

- Also transport layer at the server side notes the four values (1) source port (2) source IP (3) destination port (4) its own IP address, the newly created socket is identified by these four (newly comed segment has to match the 4 values to go in the socket)



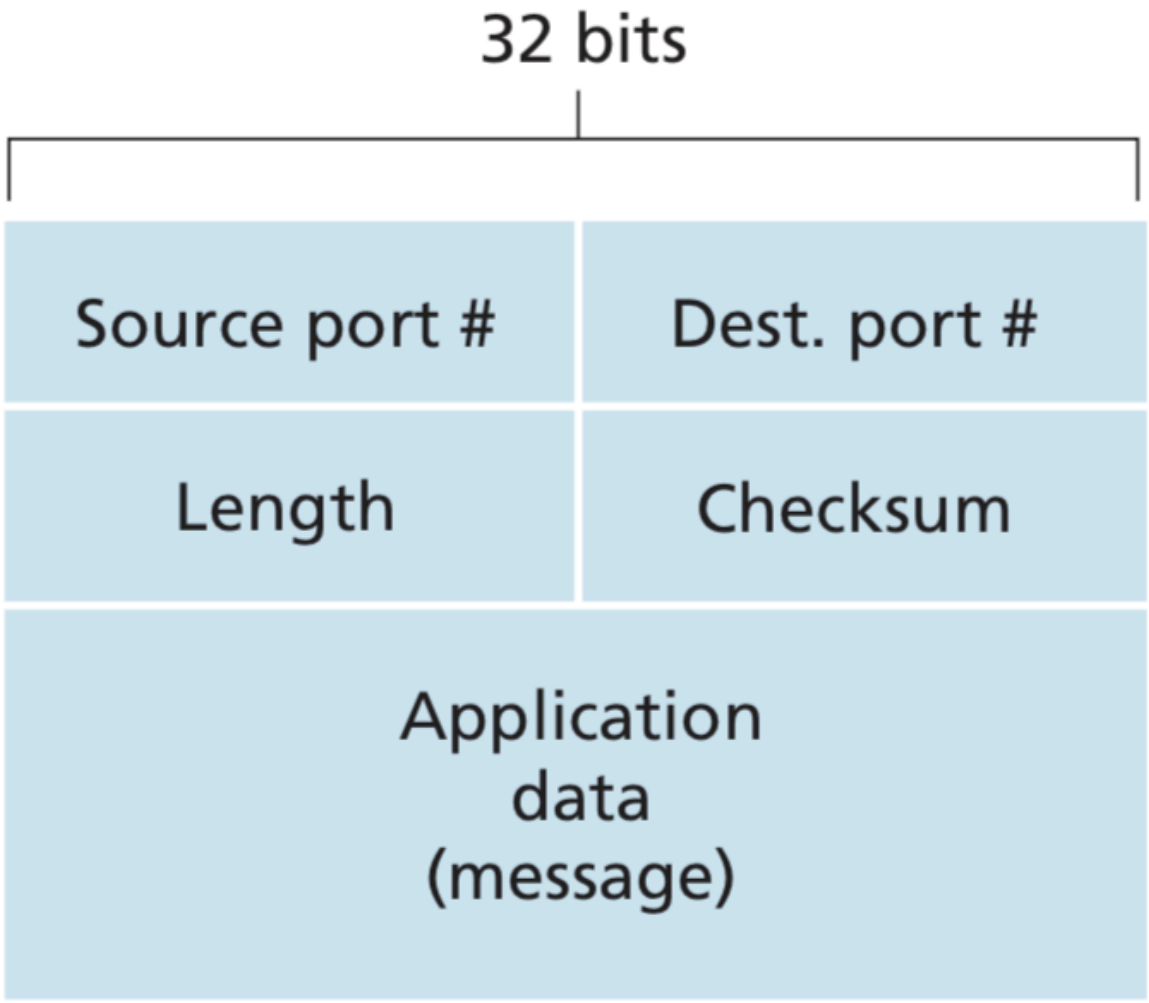
- Web Servers and TCP
  - There is not always a one-to-one correspondence between connection sockets and processes
  - High performance server often use only one process, and create a new thread with a new connection socket for each new client connection (i.e. there may be many sockets(identified by 4 numbers) attached to the same process)
  - Threaded(线程) server : in the **same port**, generate child API to deal with request of different client (main process generate new socket generate new thread process and combine them)
  - recall the HTTP
    - Persistent connection : client and server exchange HTTP messages via the same server socket
    - Not-persistent connection : different socket

3.3 UDP : Does as little as transport protocol can do (almost directly from application to IP)

- No handshaking : so called *connectionless*
- Used in DNS and streaming multimedia(ask for the hierarchy server) why? => fast
- UDP's advantage
  - Finer application-level control over what data is sent, and when (i.e. **Fast**)
  - No connection establishment
  - No connection state
  - **Small packet header overhead**
  - lost and out-of-order

3.3.1 UDP Segment Structure

- Data field: data provided application layer process
- Header field : Four fields each has 2 bytes
  - Source port and Destination port : for recongnizing process (no IP address because it is network layer header)
  - Length field : specifies the number of bytes in the UDP segment (both header and data i.e. 4*2 bytes+sizeof(data)) => needed since the size of the data field may differ from one UDP segment to the next
  - Checksum field : used by the receiving host to check whether errors have been introduced into the segment
    - In truth, also calculated over a few of fields in IP header

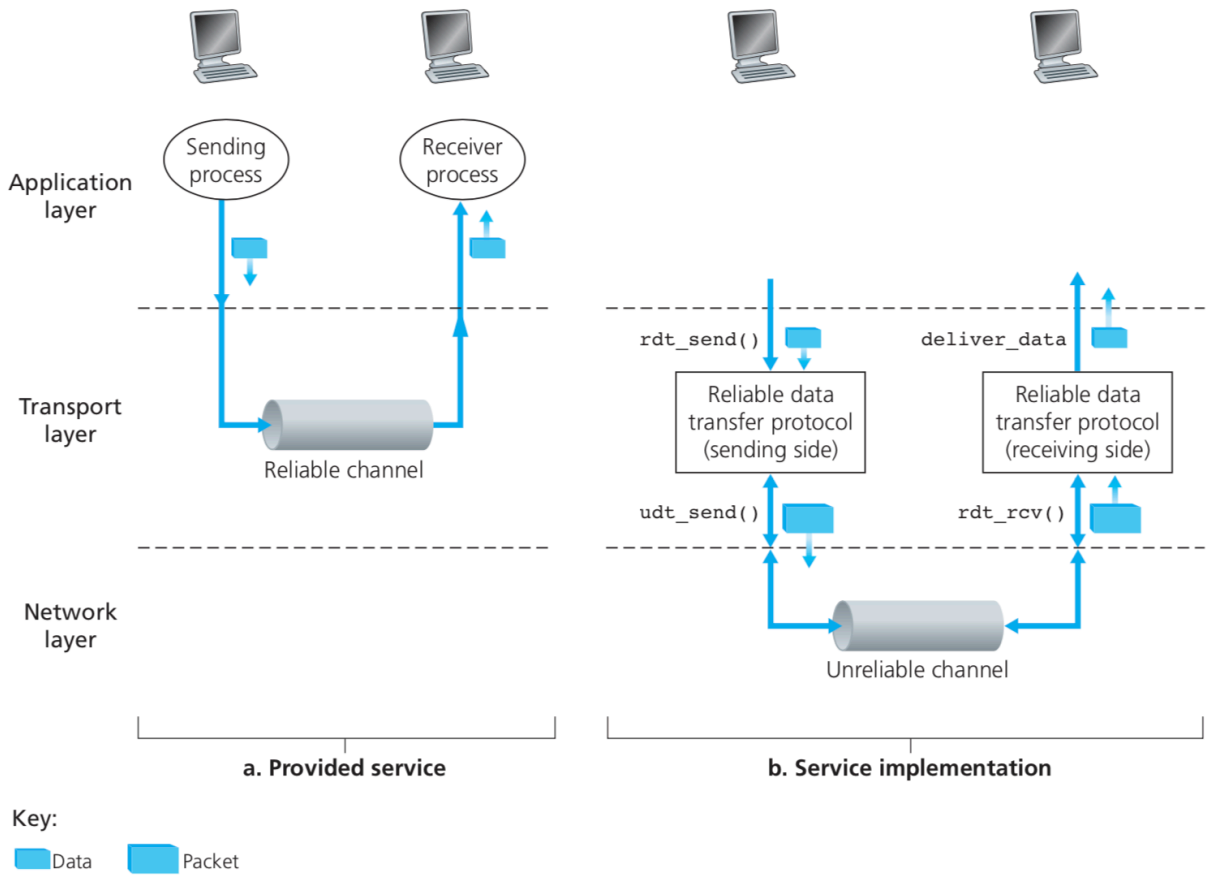


3.3.2 *UDP Checksum*

- At sender :
  - add all the 2 bytes fields in the segment together
  - If there is an overflow, add the overflow digit to end
  - perform the 1s complement (flip the number '1' to '0' and '0' to '1')
  - get the Checksum header
- At receiver:
  - Add all the fields including checksum
  - if the answer is not all '1' (because flip the number, there exist and only exist one '1' in each digits), there is error
  - if it is all '1' , maybe error, maybe not(because maybe when transmit the data the number in the same digits change together, answer will still be 1)
- Reason to provide the service
  - As Link-layer protocol also provide check error service but there is no guarantee that all the links between source and destination provide error checking
  - Also possible that bit errors could be introduced when a segment is **stored in a router**
  - **end-end principle** : end-end data transfer service is to provide error detection
- Do with error
  - discard the damaged segment
  - pass the damaged segment to the application with a warning

3.4 Principles of Reliable Data transfer

- Reason of difficult to implement reliable : The layer below may not be reliable
- One assumption : packets will be delivered in the order in which they were sent, but some packets may lost
- Terminology : use packet instead of segment, because the theory apply for all the layers
- Data transfer : only consider **unidirectional data transfer** (directly between sender and receiver rather than full-duplex)

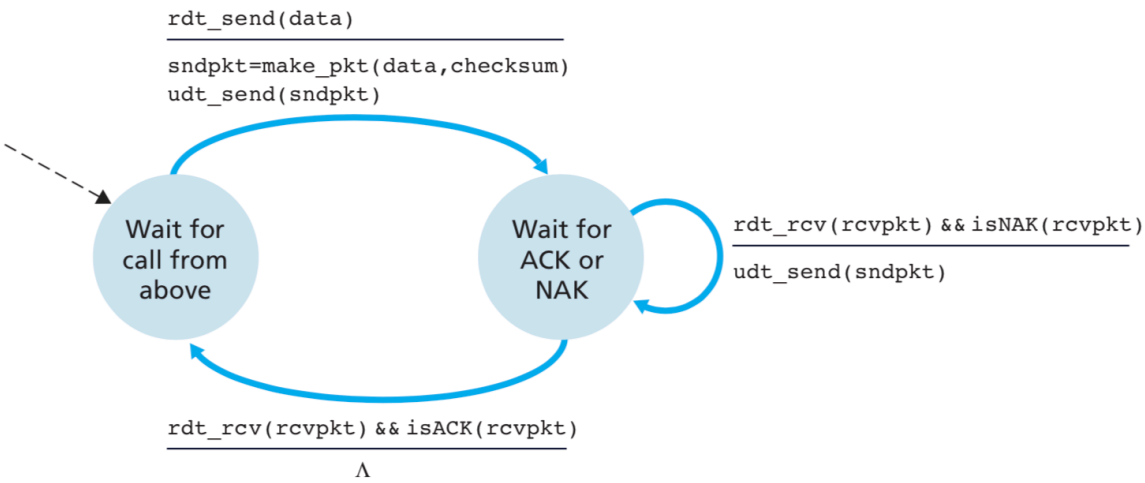


3.4.1 Building a Reliable Data Transfer Protocol

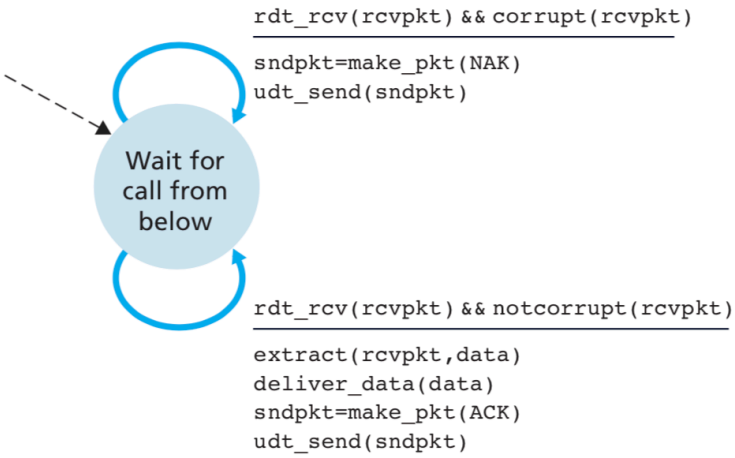
- First case : When the underlying channel (底层) is completely reliable `rdt1.0`
  - sender side : just wait for data from above (layer)
  - Receiver side : just wait for data from below
  - Both sender and receiver has 1 state , that is wait for data
  - Property
    - In this case, there is no difference between a unit of data and a packet, no need to provide feedback from receiver to sender
    - We also assume that the receiver is able to receive data as fast as the sender happens to send data => no need for receiver to ask for slow down
- Second case : Reliable Data Transfer over a Channel with Bit Errors `rdt2.0`
  - Situation : There might be some error in the packet
    - **positive acknowledgments** : OK
    - **negative acknowledgments**: Please repeat that
    - retransmission protocol : ARQ protocols
  - Fundamental additional functions
    - *Error detection* : require extra bits be sent from the sender to the receiver
    - *Receiver feedback* : Since the sender and receiver may be separated by thousands of miles, it is neccessary to let the sender know that whether data transfer is successful
      - Positive : ACK packets sent by receiver
      - Negative : NAK packets sent by receiver
    - *Retransmission* : A packet that is received in error at the receiver will retransmitted by the sender
  - Sender :
    - wait for data above
    - wait for ACK(or NAK) from receiver, cannot get data from upper layer in this state

      ```
      while(!isACK(rcvpkt)){
      if(rdt_rcv(rcvpkt)&&isACK(rcvpkt))
        changeState();
      else if(!isACK(rcvpkt))
        retransmit(theLastPacket);
      }
      ```

  - Receiver : just judge whether it receive the correct data

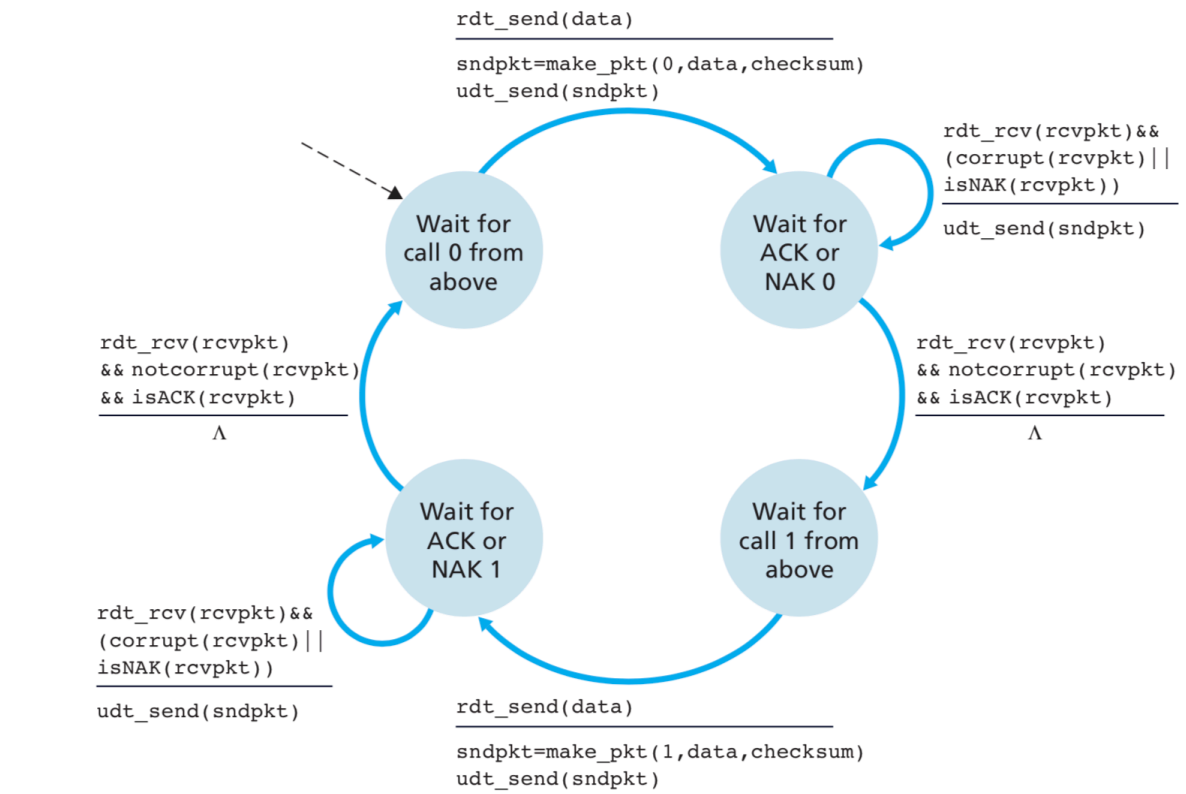


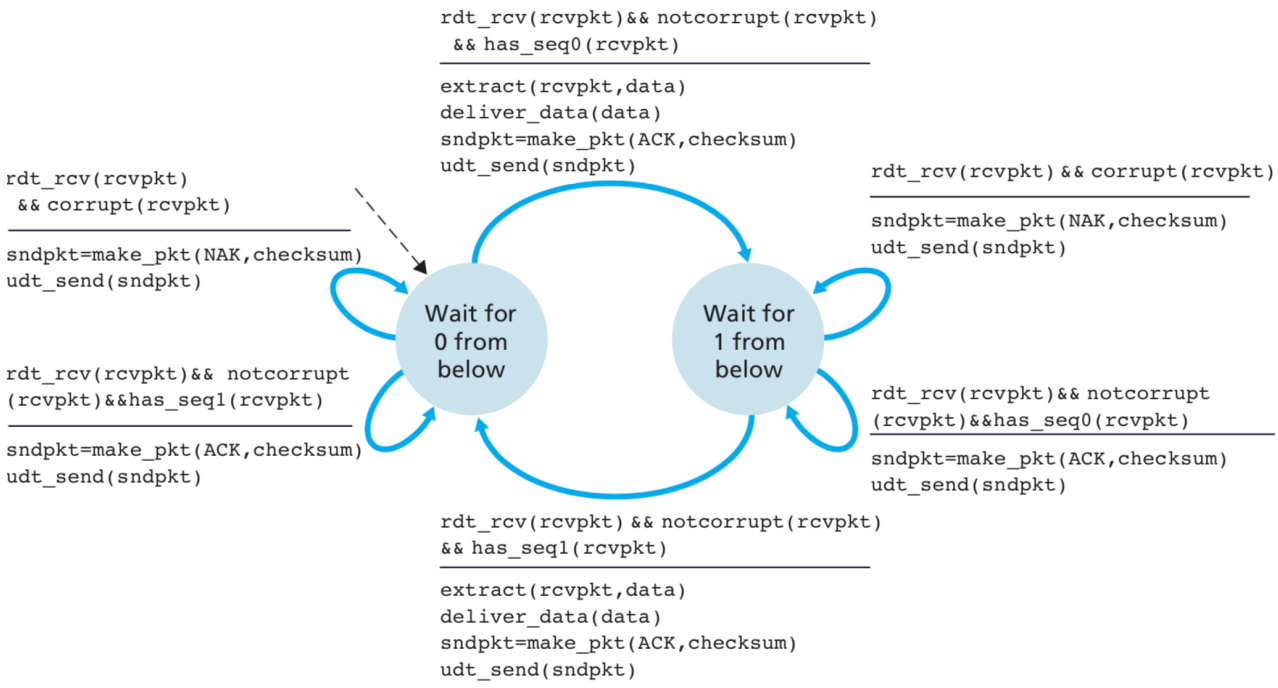
a. rdt2.0: sending side

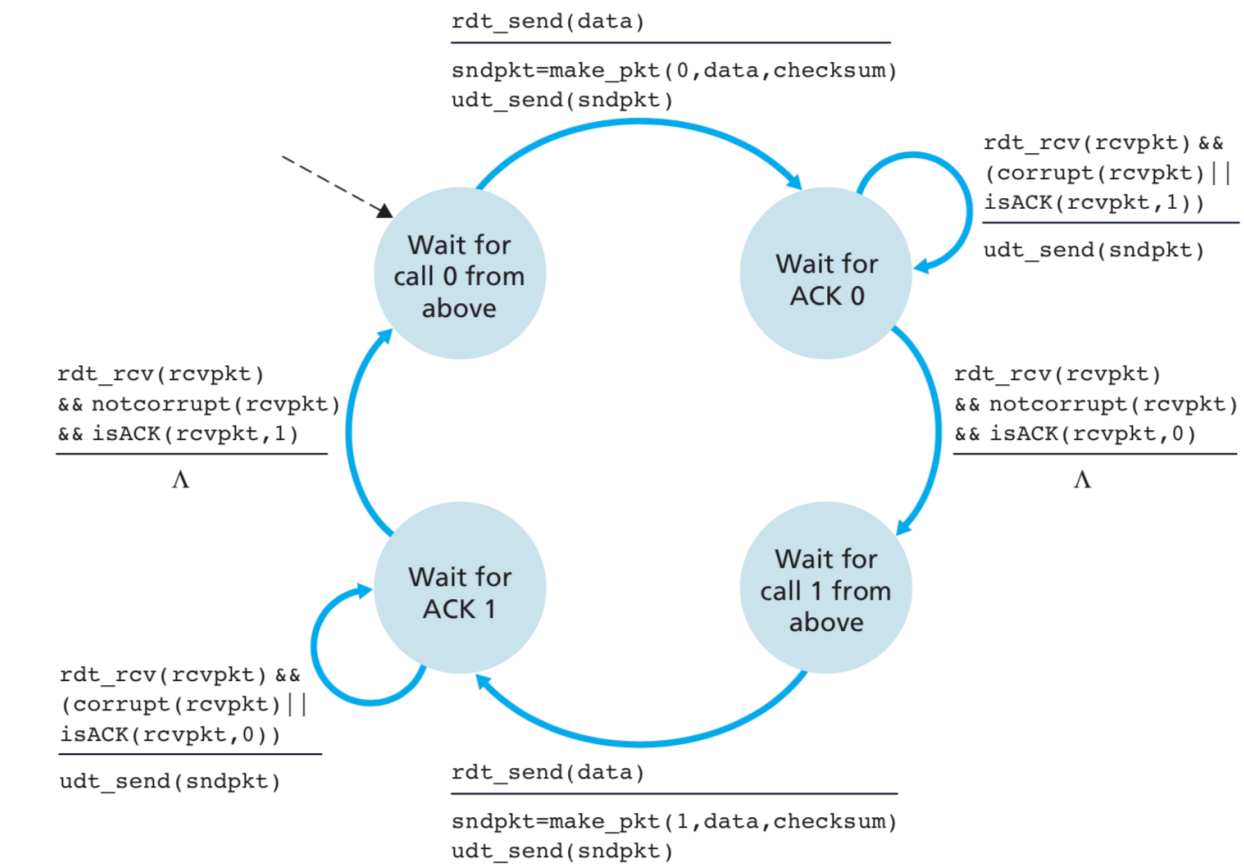


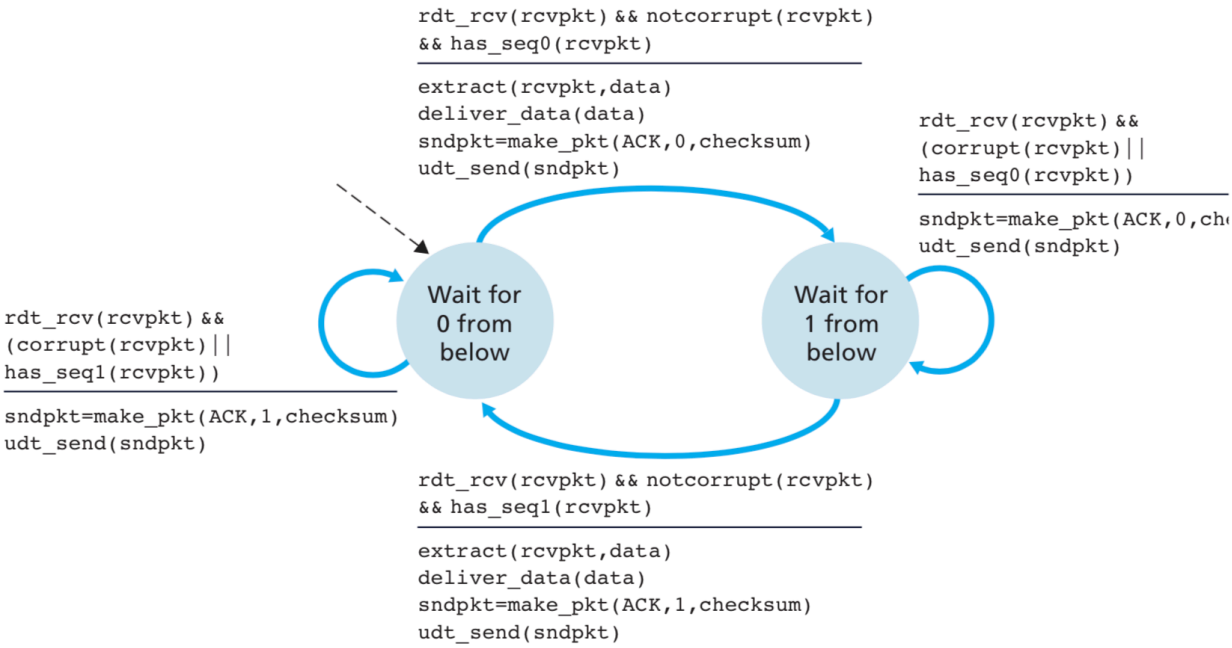
b. rdt2.0: receiving side

- Thus known as **stop-and-wait** protocols
- BUT : How to recover from errors in ACK and NAK packets => ACK and NAK packets is corrupted when transferred
  - To solve : sender simply to *resend the current data packet* (treat bad acknowledgment as NAK) when receive garbled ACK or NAK packet (**duplicate packet**) => cause new problem that receiver doesn't know the coming data is new data or resent data
  - **sequence number** (给packet编号): putting a sequence number into this field, to make receiver identify the data (0 for even number, 1 for odd number)
- `rdt2.1` : 防止收到packet后，ACK error导致duplicate packet
  - 2 new state to identify whether the packet currently being sent(by the sender) or expected(by the receiver) should have a sequence number '0' or '1'(differen state of odd seq# and even seq#)
  - Use ACK for the same ACK for the last correctly received packet(**duplicate ACKs**) instead of NAK (sender receive 2 same ACKs => the packet after the ACK is corrupted => retransmit) just add argument in `isNAK(rcvpkt,0)`


rdt_send(data)
sndpkt=make_pkt(0,data,checksum)
udt_send(sndpkt)
Wait for call 0 from above
Wait for ACK or NAK 0
rdt_rcv(rcvpkt)&& (corrupt(rcvpkt)|| isNAK(rcvpkt))
udt_send(sndpkt)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt)
Λ
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt)
Λ
Wait for ACK or NAK 1
Wait for call 1 from above
rdt_rcv(rcvpkt)&& (corrupt(rcvpkt)|| isNAK(rcvpkt))
udt_send(sndpkt)
rdt_send(data)
sndpkt=make_pkt(1,data,checksum)
udt_send(sndpkt)

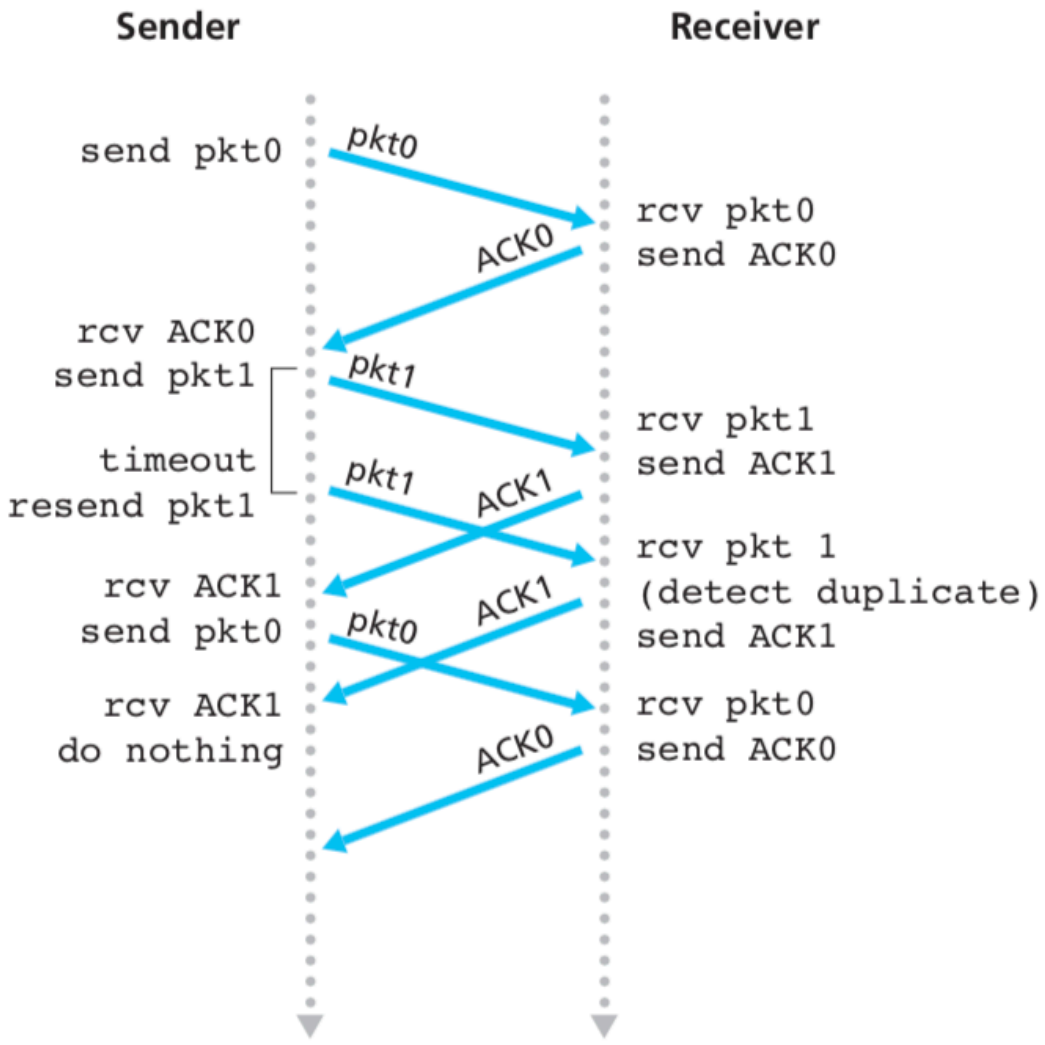

rdt_rcv(rcvpkt)&& notcorrupt(rcvpkt) && has_seq0(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(ACK,checksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && corrupt(rcvpkt)
sndpkt=make_pkt(NAK,checksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && corrupt(rcvpkt)
sndpkt=make_pkt(NAK,checksum)
udt_send(sndpkt)
Wait for 0 from below
Wait for 1 from below
rdt_rcv(rcvpkt)&& notcorrupt (rcvpkt)&&has_seq1(rcvpkt)
sndpkt=make_pkt(ACK,checksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt)&& notcorrupt (rcvpkt)&&has_seq0(rcvpkt)
sndpkt=make_pkt(ACK,checksum)
udt_send(sndpkt)
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && has_seq1(rcvpkt)
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(ACK,checksum)
udt_send(sndpkt)





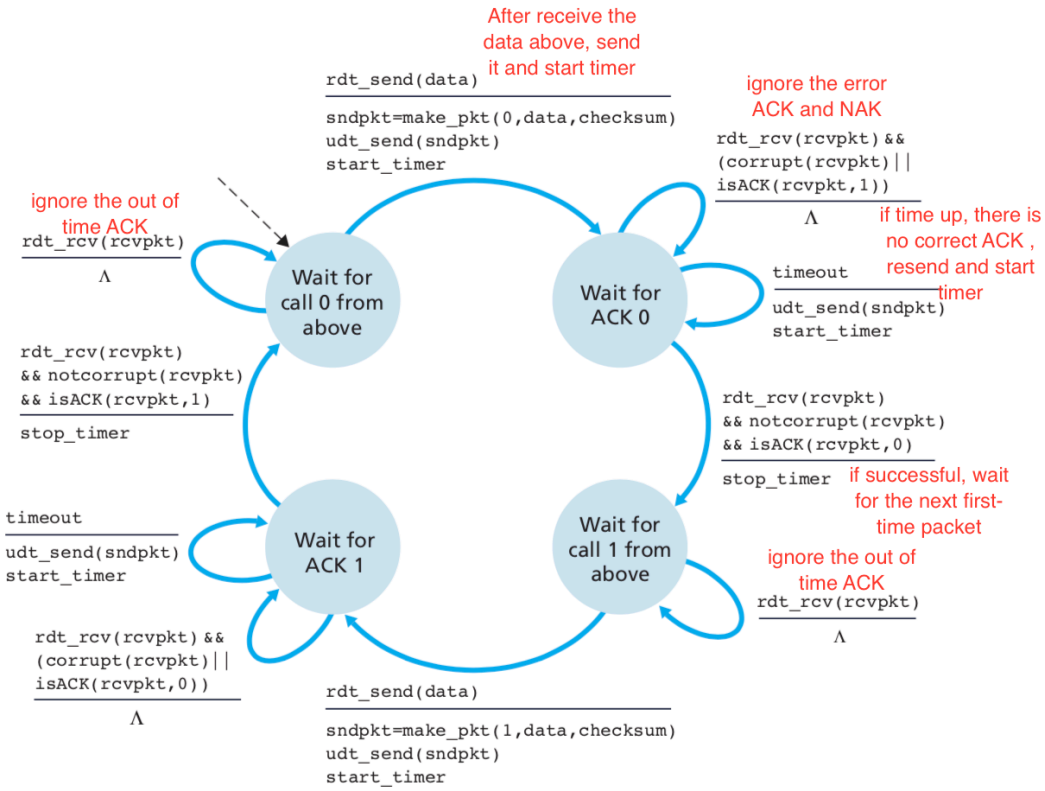
- Reliable Data Transfer over a Lossy Channel with Bit Errors: `rdt3.0`
  - When data loss occur => put the burden of detecting and recovering from lost packets on the sender
    - Suppose one of the pkt or the ACK of pet get lost => **wait for a long time** to ensure that the data is lost, then retransmit
    - How long? => At least as long as a round-trip delay between the sender and receiver plus whatever amount of time is needed to process a packet at the receiver
  - Problem of resent the lost packet

- If ACK is not back on time or has error (means it lost or error), just resent => If the time is not set properly, if packet experiences a particularly large delay, the sender may retransmit the packet even though neither the data packet nor its ACK have been lost => **duplicate data packets** (use sequence number to handle this problem)



d. Premature timeout

- **countdown timer** can interrupt the sender after a given amount of time has expired, thus the sender need to do:
  - start the timer each time a packet (first-time packet or retransmission) is sent
  - respond to a timer interrupt
  - stop the timer



- reason for **duplicate packet**: corrupted ACK or timeout
- duplicate ACK cases(for GBN)
  - 一直不成功(前一个成功的duplicate)
  - timer太快导致来不及传回ACK，相同包被传两次都导致duplicate

3.4.2 Pipelined Reliable Data Transfer Protocols

- `rdt3.0` is a functionaly correct, but slow => reason is it is a stop-and-wait protocol
- **utilization** : 真正传输时间占比
  - L is the size of packets, R is transmission rate of the packet
  - when the last bits of the packet comes to the receiver side

$$t = RTT/2 + L/R \tag{10}$$

  - assume that the ACK packet is extremly small, the time back is $RTT/2$
  - the **utilization** of the sender is fraction of time the sender is actually busy sending bits into the channel
  - we find that the number is very small => time is not efficiently used, *a lot of time to wait*

$$U_{sender} = \frac{L/R}{RTT + L/R} \tag{11}$$

- Solution of stop-and-wait => **pipelining** (流水线)
  - more efficient than before (分子为pipeline传输完的时间， 分母是第一个packet的ACK返回的时间)

$$U_{sender} = \frac{NL/R}{RTT + L/R} \tag{12}$$

  - Range of sequence number : increased (a lot of packets each time, need a identical number)
  - Sender and receiver sides of the protocols may have to buffer more than one packet
    - at least buffer the packets that have been transmitted but not yet acknowledged
    - Buffering of correctly received packets may also be needed at the receiver

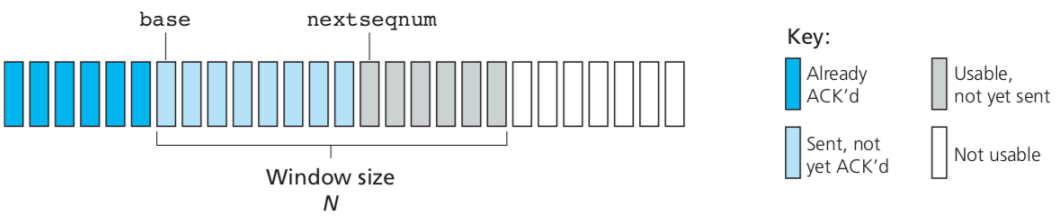
- Range of seq# and buffering requirments will depend on the manner in which a data transfer protocol responds to lost, corrupted, and overly delayed packets, 2 basic error recovery : **Go-Back-N**(seq# at least window size - 1) and **selective repeat** (seq# at least window size / 2)

2. Suppose that selective repeat protocol is used. The sequence number space is {0, 1, 2, 3}. **Suppose the sender window size is 3 (where this is actually not allowed. For illustration purpose, let us assume that the sender window size is 3)**. The sender sends packets with sequence number {0, 1, 2}. The receiver receives packets with sequence number {0, 1, 2} and sends ack 0, ack 1, and ack 2 respectively. However, all these three acks are lost. ↵
   ↵
   a) After the receiver sends ack 0, ack 1, and ack2, what will be the receiver-base? What packets are expected/acceptable at the receiver side?↵
   #3 {3,0,1}↵
   b) After the timeout interval for packet 0, the sender retransmits packet 0. Suppose that the receiver receives the packet. What the receiver will do for that packet? ↵
   Resend ACK(0) because the 0 is reused, caused problem↵
   ↵

4. Suppose that Go-back-N protocol is used. The sequence number space is {0, 1, 2, 3}. **Suppose the sender window size is 4 (where this is actually not allowed. For illustration purpose, let us assume that the sender window size is 4)**. The sender sends packets {0, 1, 2, 3}. The receivers receive all these 4 packets in order and generate acks. Suppose that all acks are lost.↵
   a) After the receiver sends out all acks, will be the expected sequence number at the receiver side? 0↵
   b) After the timeout interval for packet 0, the sender retransmits packet 0. Suppose that the receiver receives the packet. What will the receiver do? Can recongonize the new one and old one↵

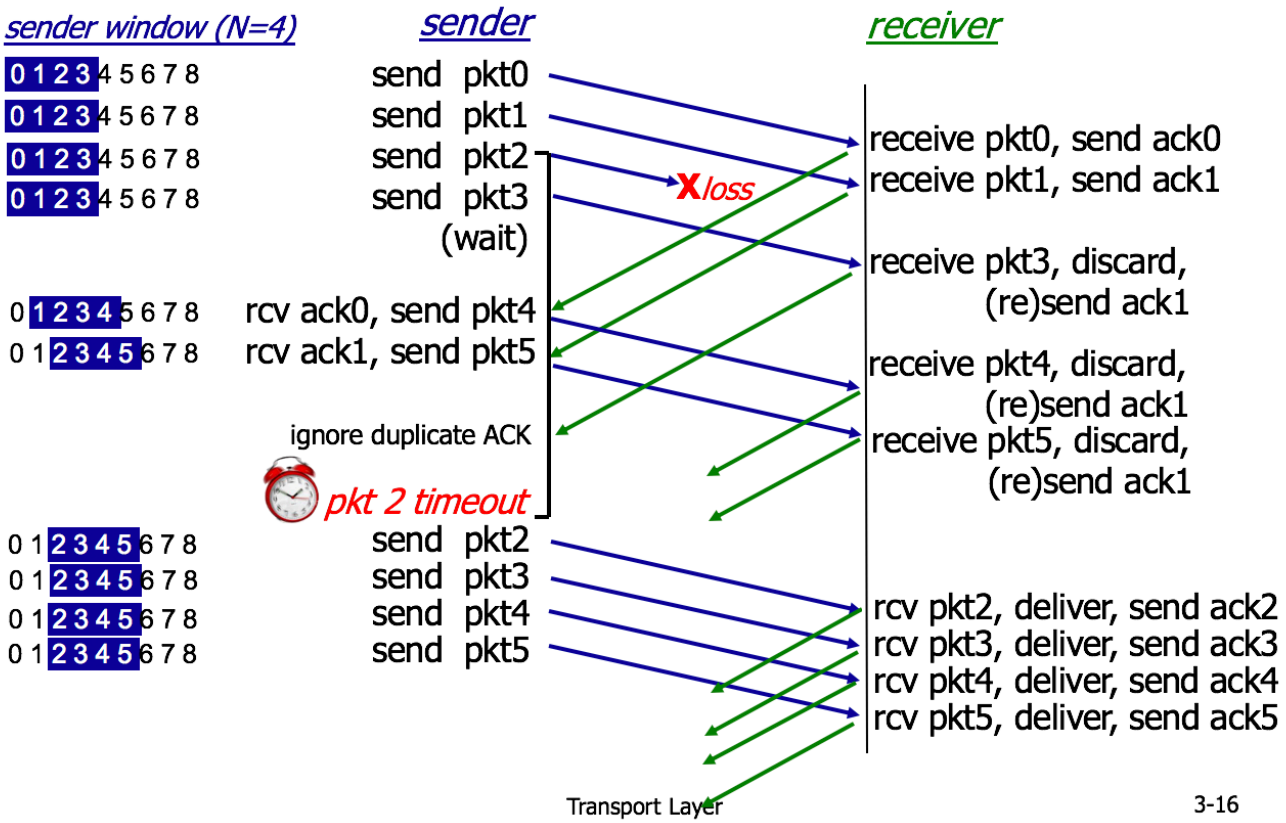
3.4.3 Go-Back-N (GBN)

- Sender is allowed to transmit multiple packets(no more than N) without ACK
- Define
  - `base` : sequence number of the oldest unacknowledged packets
  - `nextseqnum` : Smallest unused sequence number (next first-time packet to be sent)
  - Seq# `[0,base-1]` :packets that have already been transmitted
  - Seq# `[base,nextseqnum-1]` : packet that have been sent but not yet acknowledged
  - Seq# `[nextseqnum,base+N-1]` : useable but not sent
  - Seq# >= `base+N` : cannot be used until there is an ACK come back
  - window size : N
  - GBN itself : A **sliding-window protocol**



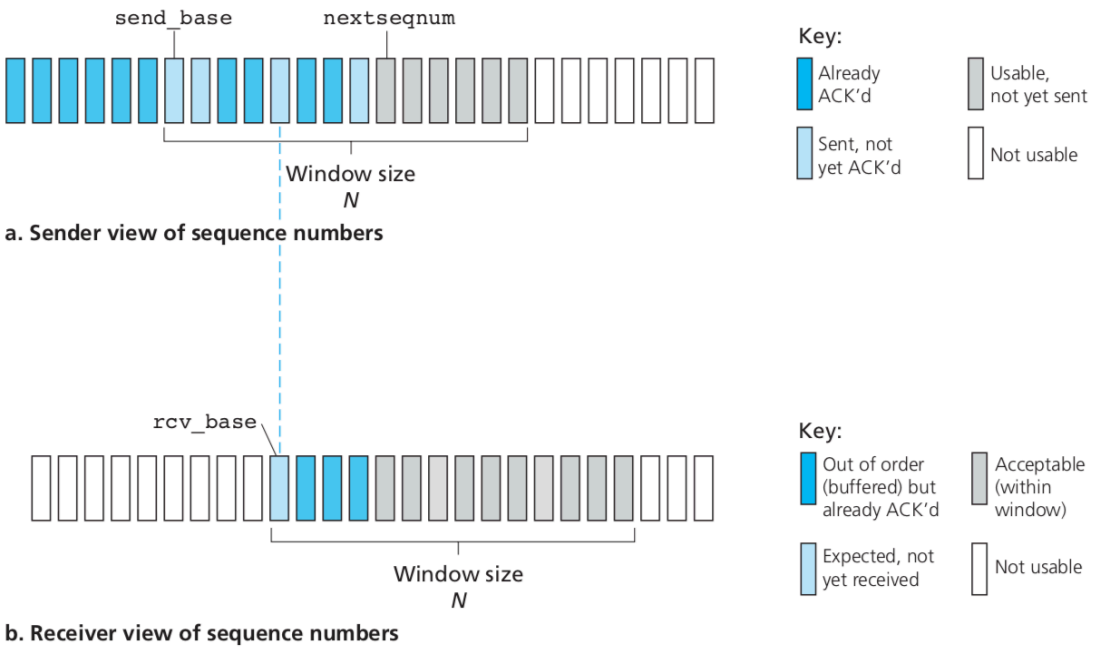
- In practice
  - If *k* is the number of bits in the packet sequence number field, the range of sequence numbers is thus $[0, 2^k - 1]$ => make sure not overflow
  - Every time $seq_n = (seq_{n-1} + 1)\%2^n$
  - stop-and-wait case : k = 1; TCP case: k = 32
- Extended *FSM* : ACK-based NAK-free GBN protocol
  - extended means: add 2 variable to the function : `base` and `nextseqnum`
- Sender
  - *Invocation from above* : when there is data comes from above, the sender will check whether the window is full
    - Full: return the data back to the upper layer (in practice : buffer the data or make the upper layer send only when the window is not full)
    - Not full : add to the window
  - *Receipt of an ACK* : In GBN, acknowledgment for a packet with seq# n will be taken to be a **cumulative acknowledgment** => indicating that all packets with a **seq# <= n** have been correctly received (n代表从0积累到n的packet都被收到了)
    - if sender receive an ACK which is earlier than base => result of short timer, just discards it (different with selective repeat)
  - *timeout event* : As in stop-and-wait protocol, a timer will be again be used to recover from lost data or lost ACK packets . When time out, *sender resend all the packet previously sent but not acknowledged*
    - i.e. all packet after the oldest unACKed packet(says kth)
      - Because, on receiver side, if there is an out of order one it will send back the last correct ACK (k-1) which is in front of the current one(k)
      - what if the ACK(k) is lost or corrupted => sender also resend from kth packet
    - Thus, only **one timer** needed to record the oldest not ACKed packet
    - If an ACK is received but there are still additional transmitted but not ACKed packets, resend and timer restart to record the next unACKed one
    - If no outstanding, unACKed packets, timer stop
- Receiver
  - If a packet with sequence number *n* is received correctly and in order (i.e. the last one is n-1)
    - send ACK for packet n=> i.e. ACK(n)
  - In all other case => discards the packet and resends an ACK for the most recently received in-order packet => may generate duplicate ACKs (also duplicate when timer is too short, packets are sent more than 1 times successfully with ACK)
    - duplicate ACK cases
      - 一直不成功(前一个成功的duplicate)
      - timer太快导致来不及传回ACK，相同包被传两次都导致duplicate
    - Out-of-order pkt
      - Discard (don't buffer) : no receiver buffering
      - re-ACK packet with highest in-order seq#
    - Thus, need only remember `expectedseqnum`, i.e. the next seq# after the last inorder seq# because don't have out of order members
    - Cons: more retransmission would be required
  - Since Receiver deliver the data to the upper layer, if kth packet delivered, then the seq#<= k packets are also delivered => Thus, cumulative acknowledgments is a natural choice for GBN

Go-back-N in action



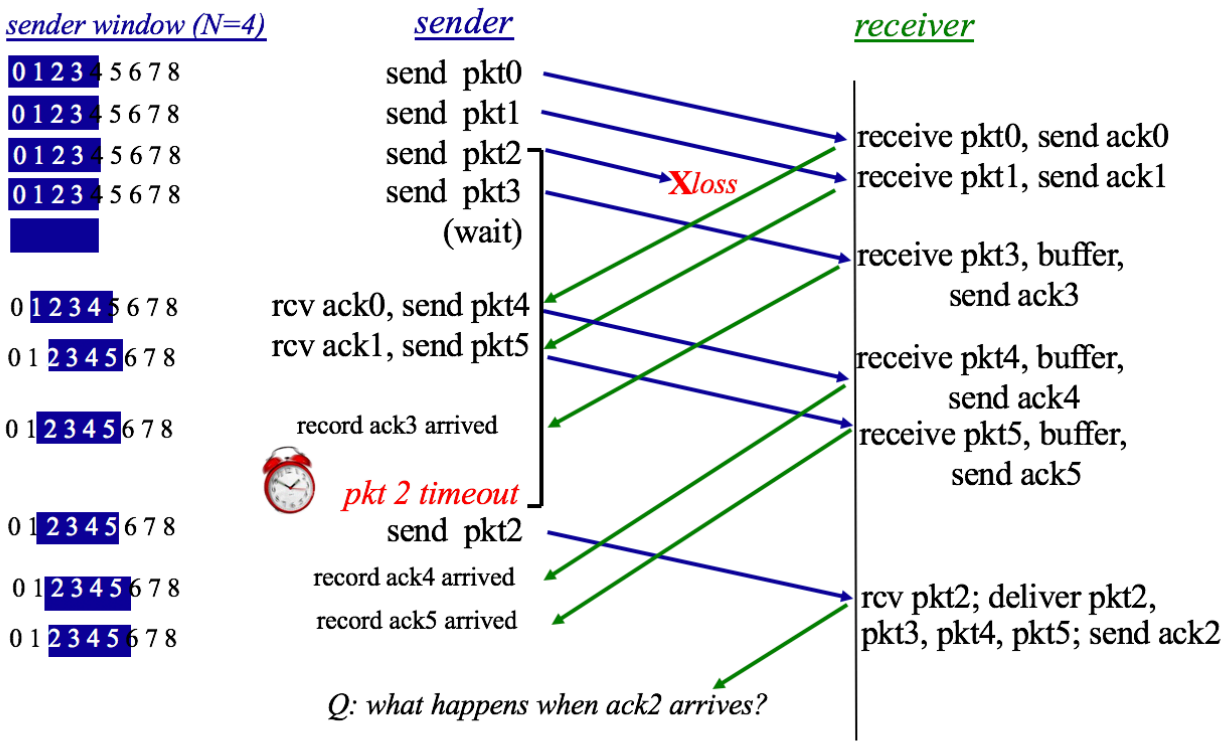
3.4.4 Selective Repeat (SR)

- Problem of GBN :
  - window size and bandwidth-delay product are both large => many packets can be in the pipeline => single packet error cause GBN retransmit a large number of packets
- Selective Repeat :
  - avoid unnecessary retransmissions by having the sender **retransmit only** those packets that it **suspects were received in error**
  - Require receiver *individually* acknowledge correctly received packets (not just expected seq#)
  - Require sender keep some of the ACKed packets in the window

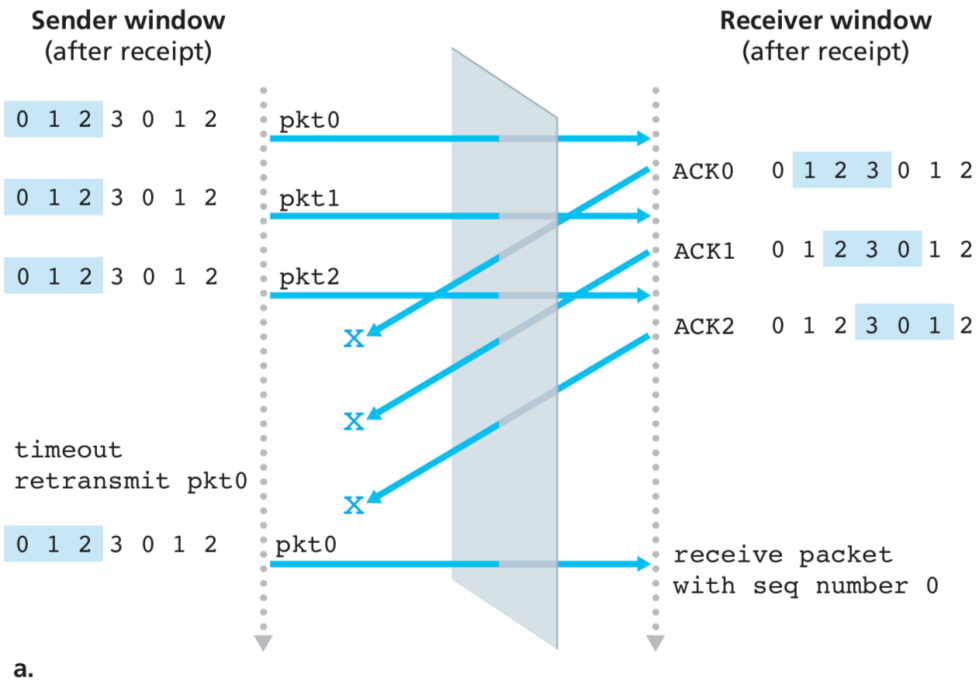


    - SR receiver will acknowledge a correctly received packet **whether or not it is in order**
        - also for packet behind the window base, for correctly move the sender window
- Sender
    - *Data received from above*: similar as GBN
    - *Timeout* : Timers are again used to protect against lost packets. But each packet has its own timer (a single packet will be transmitted)
    - *ACK received* : If ACK received, SR sender marks that packet as having been received but may not move the window
        - If `base` packet is ACKed, move then window to the minimum unACKed packets in the window
        - If windows moved, and there are **untransmitted packets** with seq# that now fall within the window, these packets are transmitted
- Receiver
    - *Packet with seq# in* `[rcv_base,rcv_base+N-1]` *is correctly received* : The received packet falls within the receiver's window and a selective ACK packet is returned to the sender
        - If the packet was not previously received, it is buffered (maybe repeat when ACK of receiver side lost or corrupted)
        - If the packet has a sequence number equal to the base of the receive window, then the consecutively buffered packets after the bases are delivered to upper layer. The window move to the first unreceived packet after the delivered sequence
    - *Packet with sequence number in* `[rcv_base-N, rcv_base-1]` *is correctly received* (i.e. the packet is arrive after the window shift => maybe caused by loss or corrupt of ACK packet) , even though this is a packet that the receiver has previously acknowledged(because the sender side need to be terminated)
    - Otherwise, ignore(receive a packet out of window, or a duplicate packet in the window)

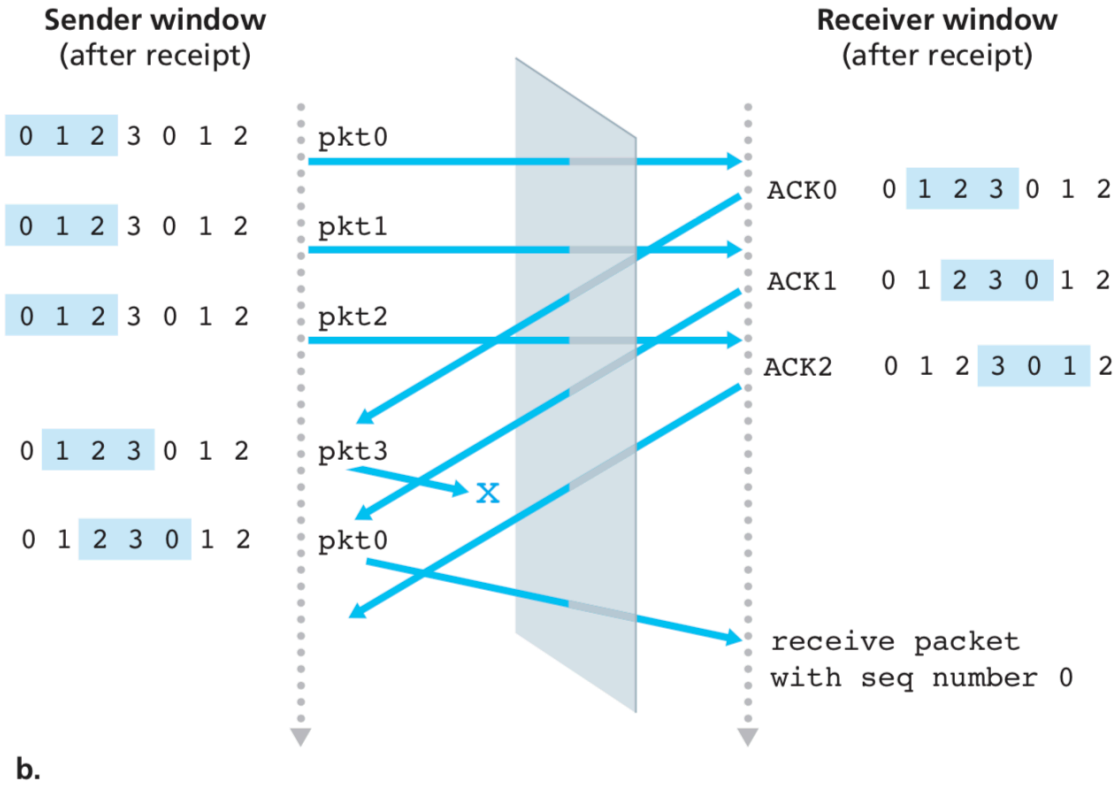
Selective repeat in action



- Lack of synchronization (缺乏同步)
  - suppose the length of sequence field is 4 => only 0,1,2,3 four numbers
  - suppose the window size is 3
  - if 0,1,2 is successfully received => sender window becomes 3,0,1
  - 2 scenario
    - ACKs for the first 3 packets are lost and the sender retransmit these packets
      - The receiver thus receive a 0 which is caused by the fail ACK transmission
      - Cause chaos



- ACKs for the first three packets are all delivered correctly, but the packet with seq# 3 id lost, and the seq# 0 after it deliveted correctly



- There is no way to identify these two scenarios
- Conclusion : The window size must be less or equal to half of the size of the sequence number space for SR protocols (i.e. no same seq# in adjacent window shifting)

- Remaining assumption
  - we assume that packets cannot be reordered within the channel between the sender and receiver
  - possible when there is only one path to transmit, but there is a lot of roads to a town

    - If reordered, there maybe a packet with seq# x, that x is neither in sender's window nor in receiver's window. i.e. x is buffered by the channel
    - Seq# may be reused (%2^n) : sender need to be sure that there is no pre-sent packet with seq# x still in the network
    - Set the lifetime of the packet to be approxiamately 3 minutes
- TCP
    - connection oriented : Explicit set-up and tear-down of TCP session
    - Reliable, in-order delivery
        - checksum => bits error
        - Acknowledgments & retransmissions for reliable delivery
            - Sender retransmits lost or corrupted data
            - Timeout based on estimates of round-trip time (wait for the data, maybe just slow instead of loss)
            - Fast retransmit algorithm for rapid retransmission
        - Sequence numbers to detect losses and reorder data => detect missing data and put the data it back in order
    - Flow control : prevent overflow of the receiver's buffer space
    - Congestion control
- Summary of GBN and SR
    - Pipelining : increased utilization : $U = N * U$ (N is maximum packets per pipeline)
    -
        - Go-back-N
            - sender can have up to N unacked packets in pipeline
            - Receiver sends $cumulative\ ack$ => doesn't ack packet if there is a gap (i.e. receive 5th without 4th packets, there is a gap between 3 and 5)
            - Sender has timer for oldest unacked packet. Resend in sequence from the oldest unacked
            - For acked packets after the oldest unack packet, just delete them from the buffer
        - Selective Repeat
            - sender can have up to N unacked packets in pipeline
            - Receiver sends $individual\ ack$ for each packet
            - sender maintains timer for each unacked packet

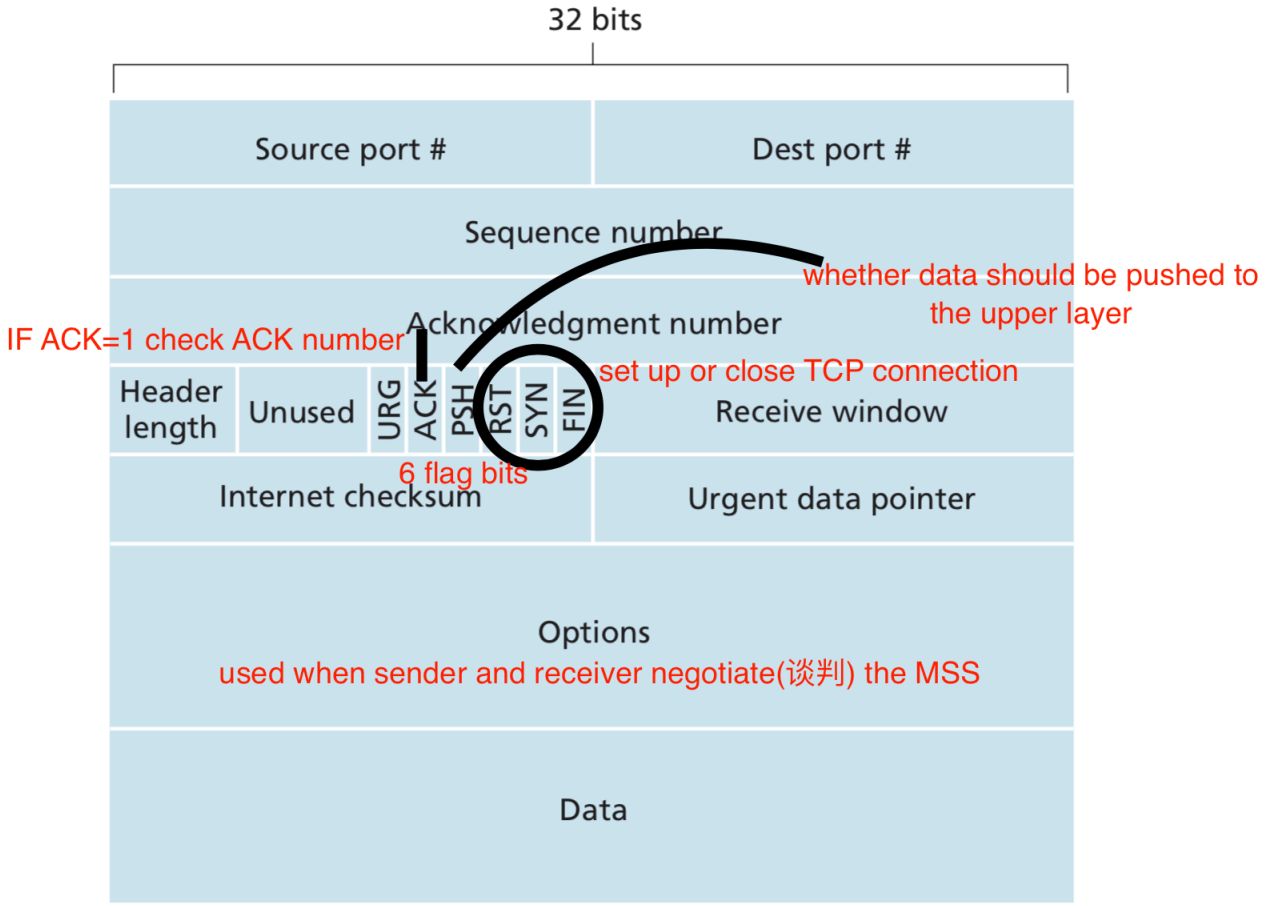
3.5 Connection-Oriented Transport: TCP

3.5.1 The TCP Connection

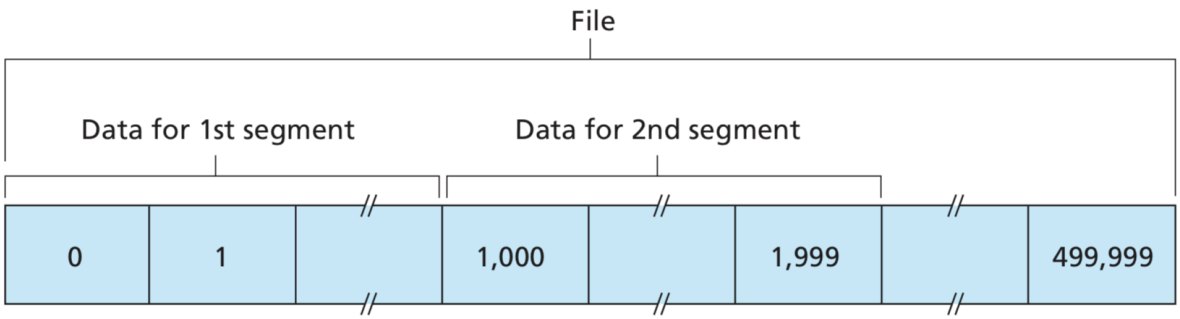
- Review of TCP
    - Connection-oriented : handshaking (Three-way-handshake)
    - TCP only runs in end systems (not in routers or link-layer switches)
    - **Full-duplex service** : Same transmission speed for both side of host connected by the same TCP
    - point to point : single sender and single receiver
- Some terms
    - **send buffer** : one of buffers set up during the initial three-way handshake
    - **maximum segment size(MSS)** : set by first determining the length of the largest link-layer frame that can be sent by the local sending host (maximum application layer data size rather than transport layer)
    - **maximum transmission unit(MTU)** : largest link-frame

$$MTU = MSS + TCP/IP\ header \tag{13}$$

3.5.2 TCP Segment Structure



- seq# and ACKs#
  - Seq# : **byte stream "number"** of first byte in segment's data(i.e. seq#-1=the last bytes has been received) => TCP 可以借用SR的优点的本质原因在于seq#的选取非常巧妙
    - initial the seq# randomly (prevent it is same as earlier used numbers)
    - neighbor segment. seq# - this.seq# = MSS
  - ACK# : Since the TCP is full-duplex, everyone could be sender or receiver
    - $ACK_A = last\ correct\ PKT_B + 1$
    - $ACK_B = last\ correct\ PKT_A + 1$
    - i.e. the next seq# the **receiver expecting**
      - GBN : the currently received packet
    - cumulative ACK : only ask for the first expected (when there is a gap, ask for the first bytes of the gap)
    - buffer the out-of-order segments



3.5.3 RTT Estimation and Timeout

- timeout value : longer than RTT (BUT RTT varies)
  - too short : **premature timeout**, unnecessary retransmissions
  - too long : slow reaction to segment loss
- Measurment
  - measured time from segment transmission until ACK receipt ($SampleRTT = t_{ack} - t_{sent}$) => average the several *recent* measurements rather than the current one
  - At any point in time, the `SampleRTT` is being estimated for only one of the transmitted but currently unACKed segments => leading to a new value of `SampleRTT` once every RTT
  - ignore the retransmitted segments
  - Because the value will fluctuate(波动) due to the congestion in the routers and to the varying load on the end systems
    - Use the average value `EstimatedRTT`

$$EstimatedRTT = (1 - \alpha) * EstimatedRTT + \alpha * SampleRTT \tag{14}$$

    - Weghted combination of previous value and new value of Sample
    - recommended value is 0.125
    - Also need to measure the variability(变化) of the RTT, define `DevRTT` as an estimate of how much `SampleRTT` typically deviates from (do not test)`EstimatedRTT` :

$$DevRTT = (1 - \beta) * DevRTT + \beta * |SampleRTT - EstimatedRTT| \tag{15}$$

    - DevRTT is an EWMA(exponential weighted moving average) of the difference between SampleRTT and EstimatedRTT
- Setting and Managing the Retransmission Timout Interval(do not test)
  - The interval should be larger than `EstimatedRTT` (stop and wait) => otherwise, unnecessary retransmissions
  - not to large => otherwise, when segment lost, TCP wouldn't quickly retransmit the segment
  - Therefore, set it to be `EstimateRTT` plus some margins

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

  - Initial value of the interval should be set to 1 sec
  - timeout occurs => double the interval => to avoid premature(过早的) timeout occurring for a subsequent segment that will soon be acknowledged => However, next time the value will be computed again by the fotrmula

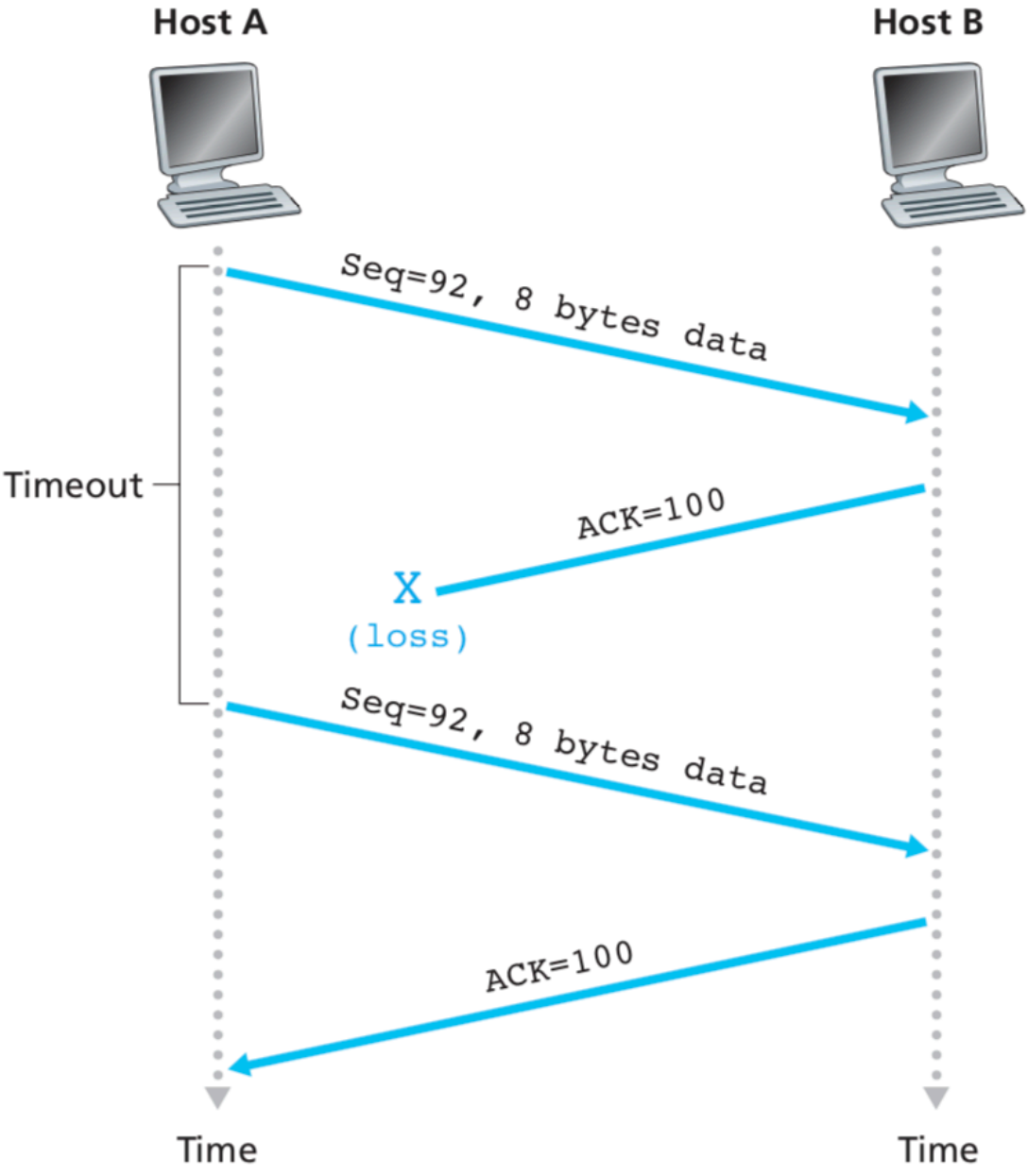
3.5.4 TCP Reliable Data Transfer

- Review : with IP service datagrams can overflow router buffers and never reach their destination, datagram can arrive out of order, and bits in the datagram can get corrupted
- first, don't consider duplicated ACK and window control
- Sender :
  - ACK(n) $all\ byte\ before\ n$ is correctly received (window remove to the expected one)
  - Single timer for segment loss => associated with the oldest unACKed segment
  - Three basic events : data received above, timer timeout, ACK receipt
  - Timer timeout
    - resend the segment caused timeout
    - restart timer
  - ACK receipt
    - change the window base to y (expected)
- Receiver
  - If no gap, send the next expect ACK
  - If there is gap, send back the first bytes of the gap as seq# of ACK

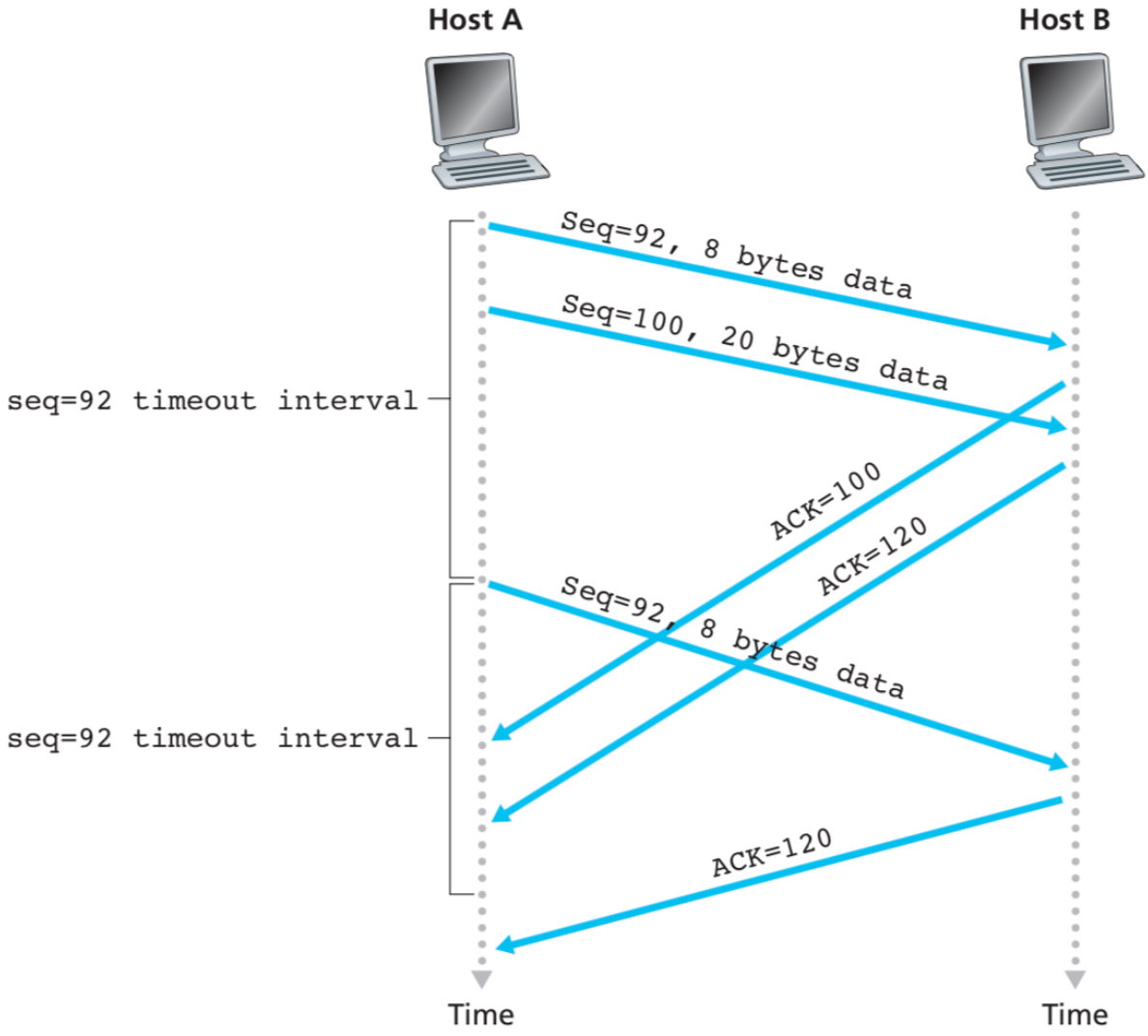
- Duplicate packets to receiver(due to the timeout interval is too short, the ACK is not back, sender resend the pkt) : ignore duplicate, just send expected ACK
- loss or corrupted ACK : receiver will just send expected

Event	Action
arrival correct seq# pkt, and all data up to expected seq# already acknowledged	Delayed ACK : wait for the next segment for about 500ms
correct seq#, one other in-order segment waiting for ACK judgement	Immediately send single cumulative ACK, ACKing both in-order segments
arrival of out-of-order segment higher-than expect seq# (Gap)	immediately send back duplicate ACK
arrival of segment partially or completely fills gap	Immediately send ACK, provided that segment starts at lower end of gap or the next expected

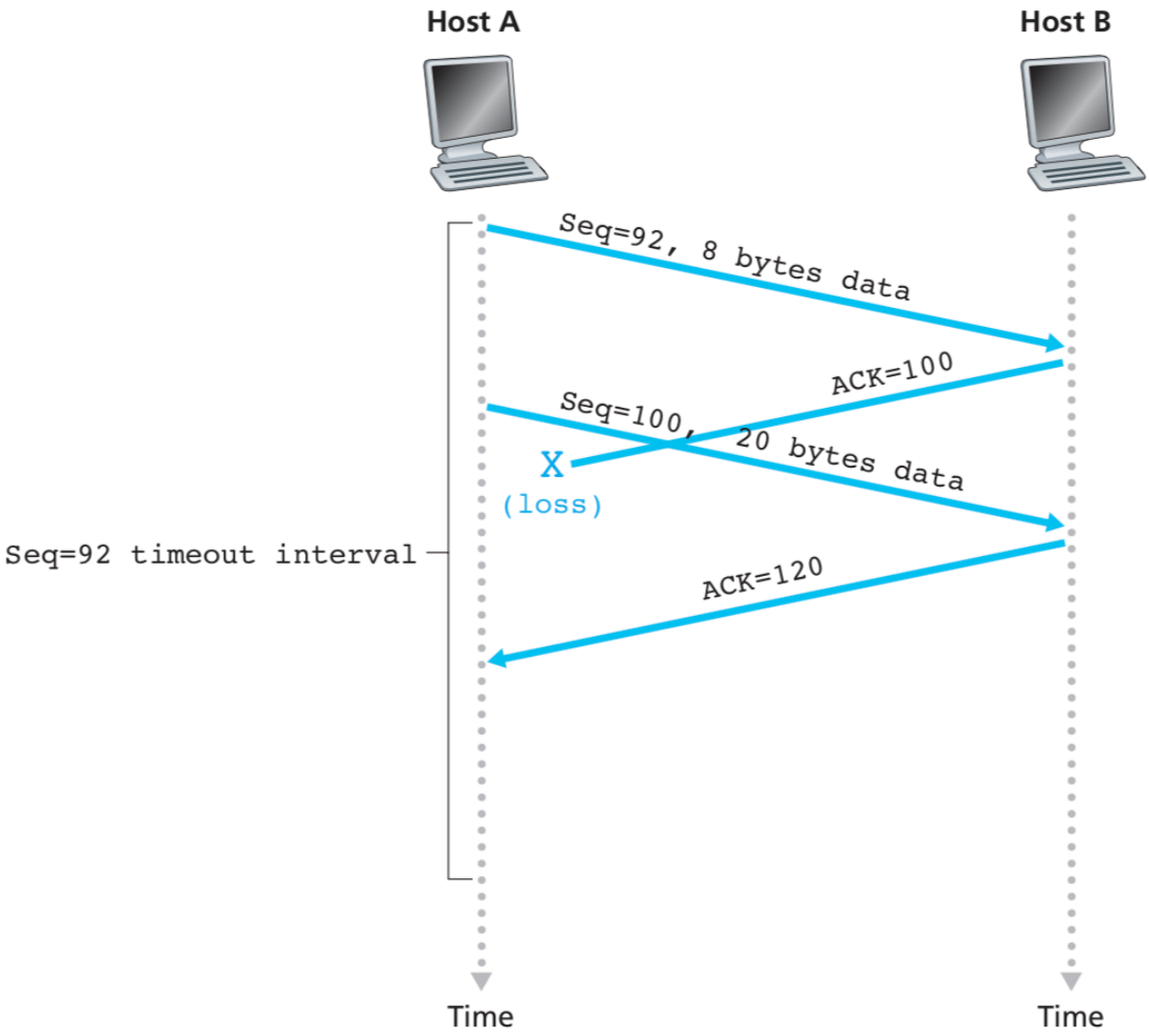
- A few interesting scenarios
  - first case : lost ACK , duplicate segment, resend ACK



- Second case : 2 continuously segments' ACK arrive late, duplicate segment, resend last ACK (Sender restart timer, duplicate ACK occurs)



- Third case : ACK before the last one lost => ignore

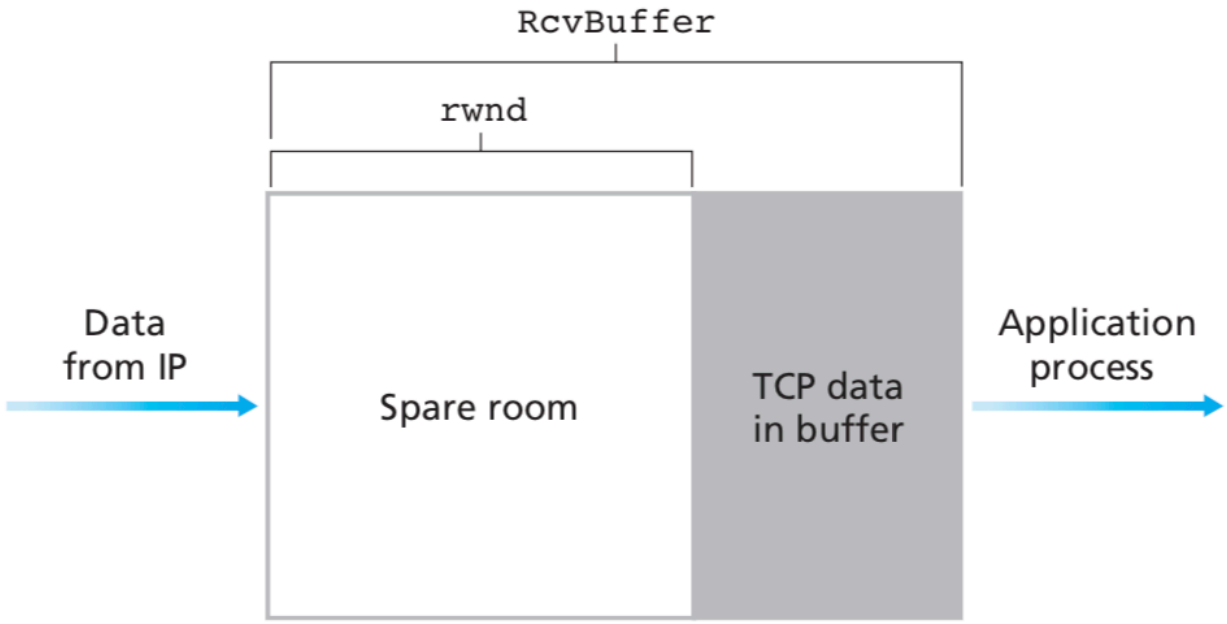


- **Doubling the Timeout Interval** : now discuss a few modifications that most TCP implementations employ
  - Timeout interval will be doubled if the timeout occurs => interval grow exponentially after each continuously retransmission
  - Timeout interval will be recalculated by the formula when other 2 events occur
  - Timer expiration is most likely caused by congestion in the network (i.e. too many packets in the same router causing packet loss or long queuing delay)
- **Fast** Retransmit(By using duplicate ACK)
  - timeout events => long delay before resending lost packet
    - Problems of timeout-triggered retransmission : if timeout interval is long, need to wait for a long time before retransmitting
  - duplicate ACK => segment is lost, a lot of segment after it received by receiver, then receiver will send back a lot of same ACK(只是优化， 没有补充timeout检查不到的问题)
    - if duplicate ACK exist, there might be error, but also possible to be correct => thus receive $3ACKs$ for same data then resend unacked segment with smallest seq#
    - If 3 duplicate occurs, sender performs a **fast retransmit**
- Compare with GBN and SR
  - GBN
    - Same : cumulative ACK, only maintain 1 timer , 1 sendbase and NextSeqNum
    - Diff : buffer correctly received but out of order segments also selective repeat

3.5.5 Flow Control

sender may send faster than receiver buffer send the data to upper layer => overflow of receiver buffer

- Thus Flow control => speed-matching service
- congestion control => solve the problem of whole network congestion (use same method on the sender side as flow control but for different reason)
- Method : have the sender maintain a variable called **receive window**(rwnd) => give sender



  - receiver

$$rwnd = RcvBuffer - [LastByteRcvd - LastByteRead] \tag{16}$$

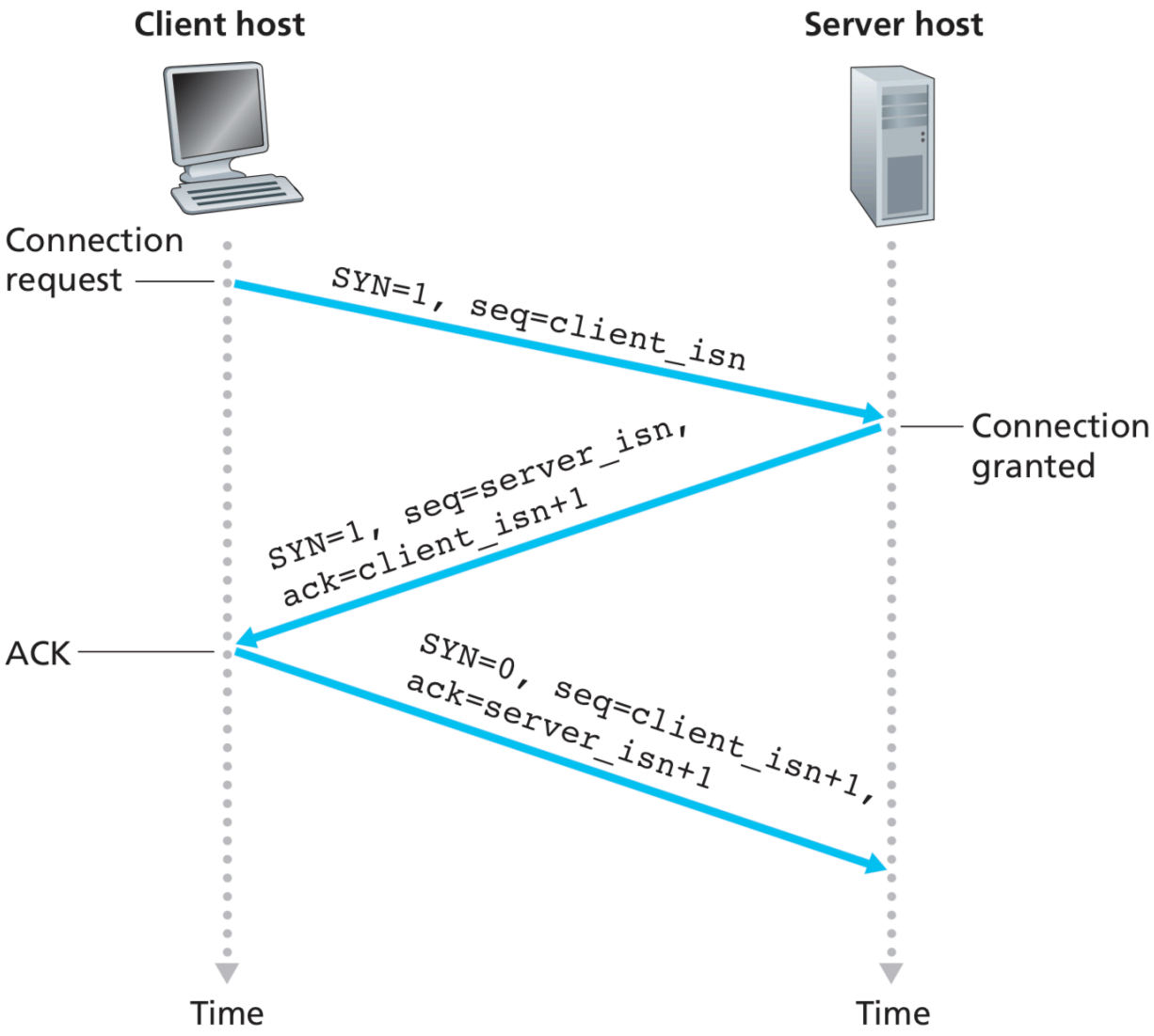
  - Sender

$$LastByteSent - LastByteAcked \leq rwnd \tag{17}$$

  - Problem : when receiver buffer is full, now sender stop sending, but as receiver's application layer empty the buffer without sending back any information (only when receives segment it will send ACK) , sender thus blocked
  - Solve : Host A continue to send segments with one data byte when rwnd is 0, this segment will be ACKed and return the window size

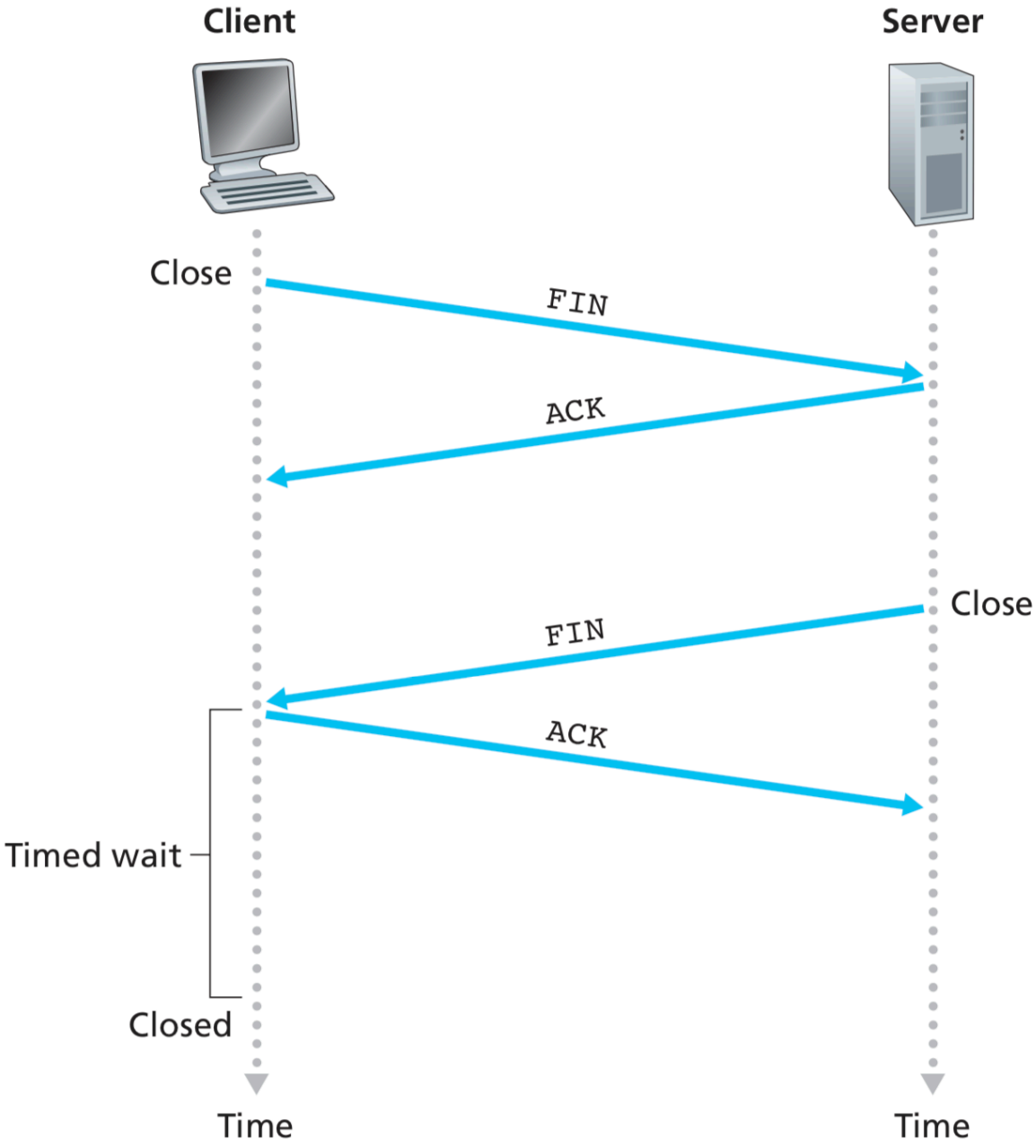
3.5.6 TCP Connection Management

- establish a connection : three ways hand shaking(1.5 RTT, then 0.5 RTT for html)



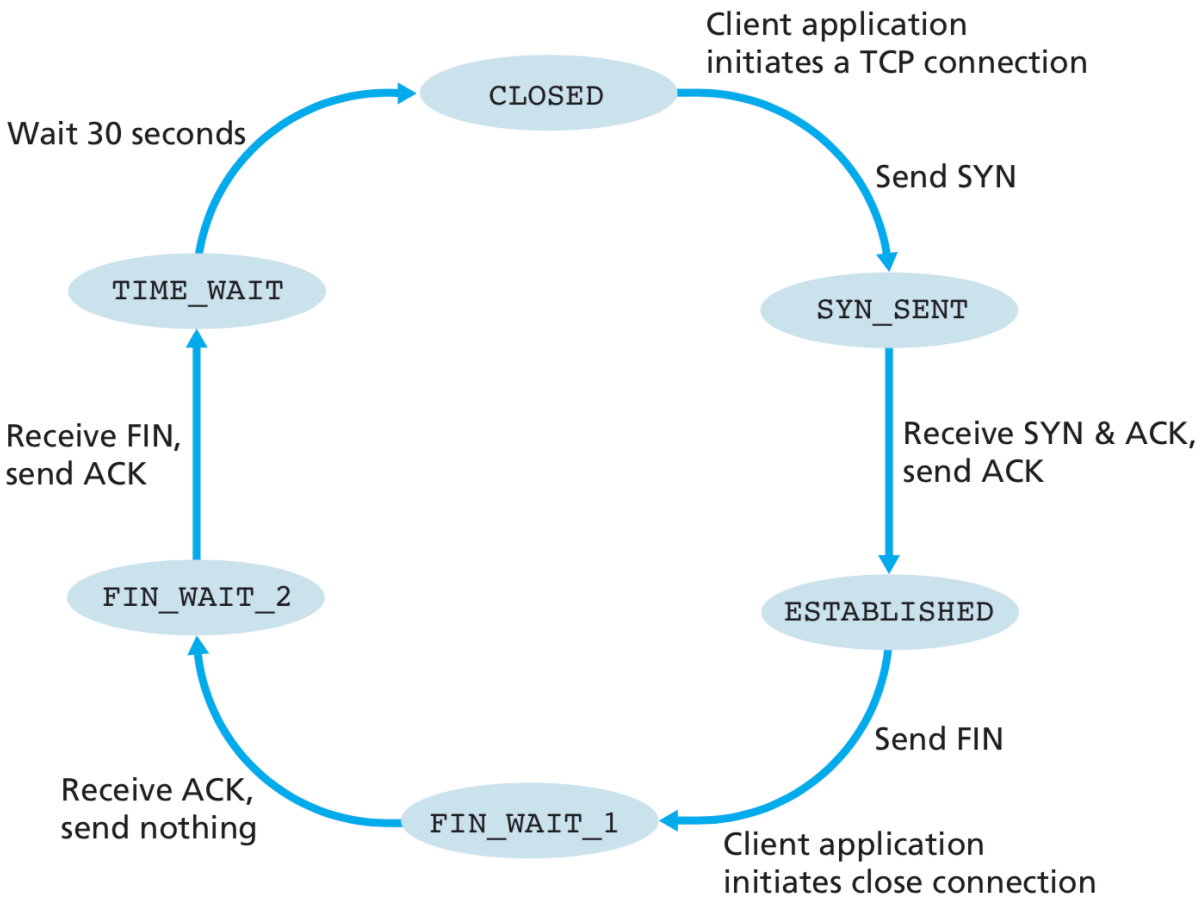
- step1 : client-side TCP first sends a special TCP segment(no application data) to the server-side TCP, the SYN bit of the segment header is set to 1
  - The special TCP segment
    - SYN = 1;
    - application data = null
    - Seq# = random # denote as client_isn + 1
- Step 2 :
  - The IP datagram containing the TCP SYN segment arrives the server, the server extracts the TCP SYN segment, **add the buffers** and variables to the connection
  - Then send a connection-granted segment back
    - SYN = 1
    - ACK = client_isn +1
    - Seq# : random# server_isn
  - The connection granted segment is refered as a **SYNACK segment**
- Step 3 :
  - Upon receiving the SYNACK segment, the client also allocates buffers and variables to the connection, and then the cliet put the server_isn +1 as ACK number
  - The SYN is set to 0, begins transfer the data

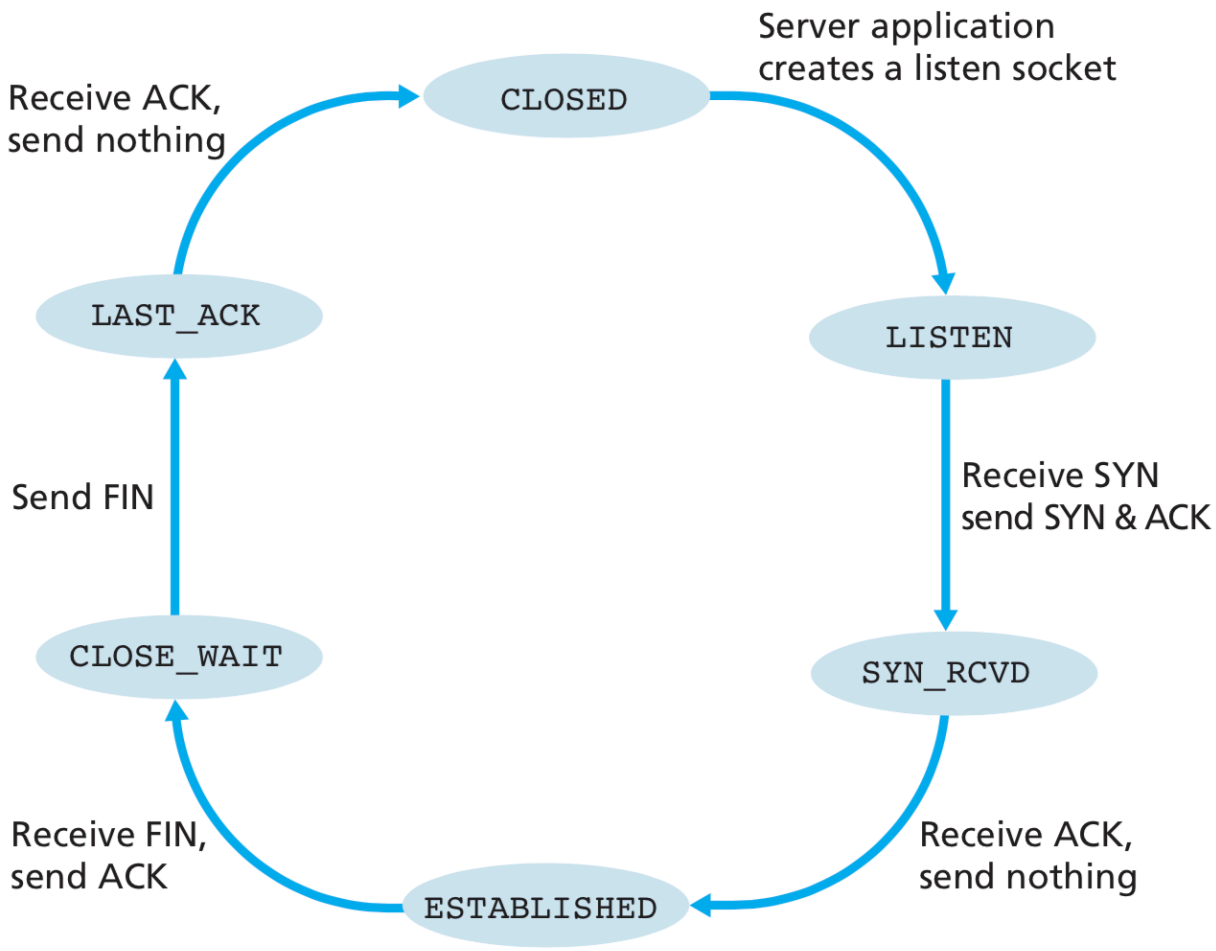
- Close a connection : The "resources"(variables and buffer) should be deallocated



- First, the application layer of sender declare a command of closing the connection, client side send a special segment FIN = 1, then **client can no longer send data**
- Second, the server send back ACK for FIN and wait for a while(**server still can send data**) , then sends its own shutdown segment, which has the FIN set to 1(**server can no longer send data**)
- Thirdly, the client side acknowledges the server's shutdown segment => at this point, all the resources in the two hosts are now deallocated

- During life TCP connection


CLOSED
Client application
initiates a TCP connection
Send SYN
SYN_SENT
Receive SYN & ACK,
send ACK
ESTABLISHED
Send FIN
FIN_WAIT_1
Client application
initiates close connection
Receive ACK,
send nothing
FIN_WAIT_2
Receive FIN,
send ACK
TIME_WAIT
Wait 30 seconds



- Situation : The host receives a TCP SYN packet with destination port 80,. but the host is not accepting connections on port 80
  - The host will send a special reset segment to the source. (RST bit set to 1)

3.6 Principles of Congestion Control

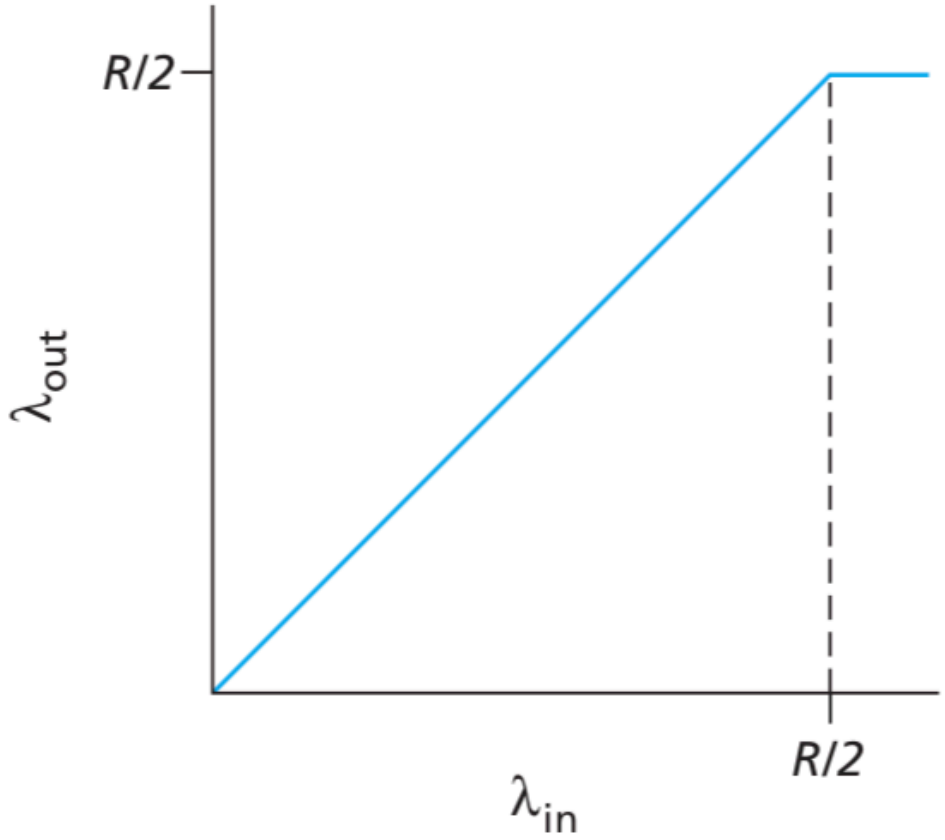
Data loss or long delay => overflowing of router buffers as the network becomes congested

Retransmission => Does not treat the cause of network congestion-too many sources attempting to send data at too high a rate

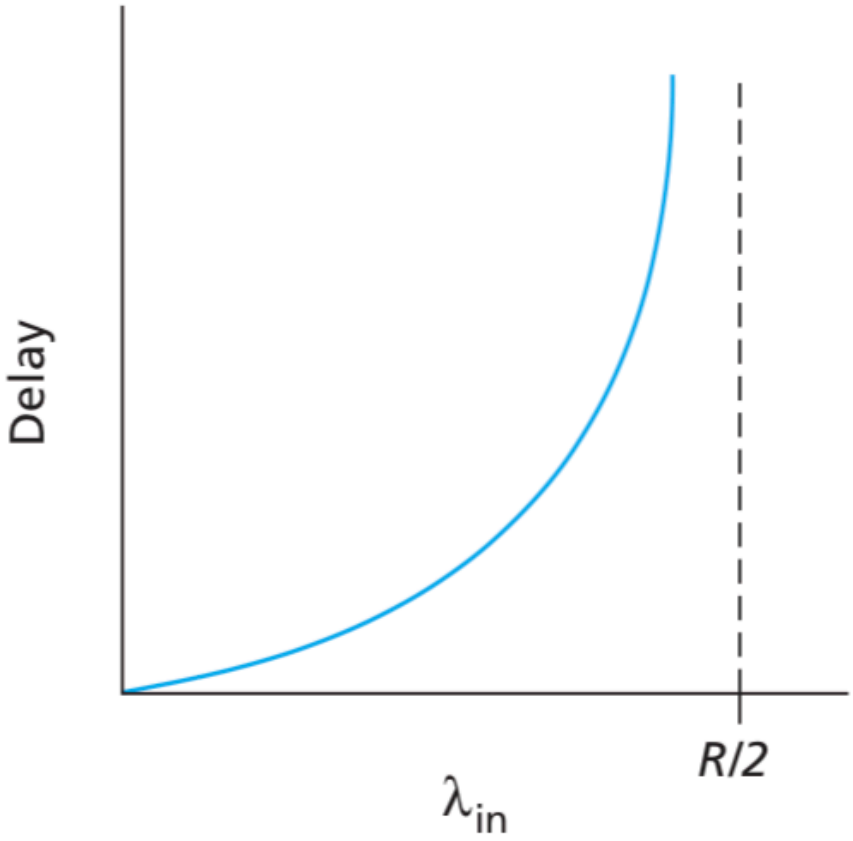
available bit-rate (ABR) service in **asynchronous transfer mode(ATM)** networks

3.6.1 The Causes and the Costs of Congestion (not in class)

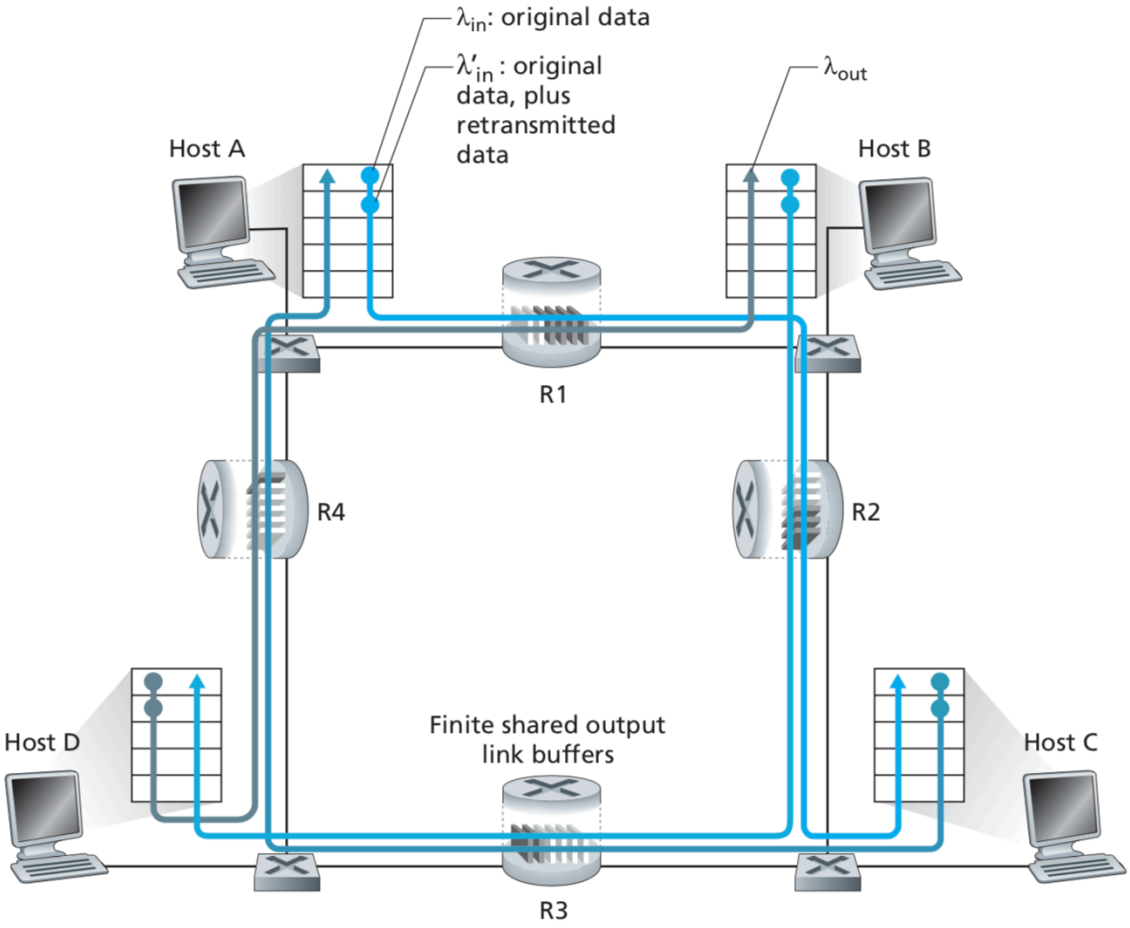
- Simplest scenario : 2 hosts each have a connection that shares a single hop between source and destination (Suppose the Transport layer protocol is weak)
  - A sending data in to the connection at ave rate of λ_{in} bytes/sec => igoring the adding header time, the rate at which Host A traffic to the router in this first scenaro is thus λ_{in} bytes/sec
  - **per-connection throughput**(number of bytes per second at the receiver) as a function of the connection-sending rate (when sender rate larger than R/2, it will not increase)



- Operating near link capacity
  - As the sending rate exceed R/2 the average number of queued packets in the router is unbounded, and the average delay between source and destination become infinite => In the simplest scenario, there is a cost of congested network - large queuing delays are experienced as the packet arrival rate nears the link capacity



- Second Scenario : 2 senders and a Router with Finite Buffers => with data loss
  - sending rate : packets can be retransmitted we denote the term *sending rate* as
    - λ_{in} when the application sends original data
    - λ_{in}' (**offer load** to the network)when the transport layer sends segment(both original and retransmited data are considered (by using plus))
  - Case 1 : Buffer is free => no loss occur
    - $\lambda_{in} = \lambda_{in}'$, Throughput = λ_{in}
    - Ave host sending rate cannot exceed R/2
  - Case 2 : The sender retransmits only when a packet is known for certain to be lost (Also beyond the realistic because when timeout is not large enough the packet will be resent)
    - Cost of retransmission
  - Case 3 : The sender may time out prematurely and retransmit
    - Cost of unneeded retransmission
- Third Scenario : 4 senders, Routers, with Finite Buffers and Multihop paths



- For extremely small value of λ_{in} the sender side, overflows are rare. So in this case a little change of λ_{in} effects a lot of λ_{in}'
- For large value, arrival rate of B-D traffic at R2 can be much larger than that of the A-C traffic
  - because A-C and B-D complete at router R2 for the limited buffer space
  - rate of A-C goes to zero in the limit of heavy traffic
  - When a packet is dropped at second-hop router, the work done by teh first-hop in forwarding a packet to the second router ends up being "wasted" (好不容易过来，前面白努力了)

3.7 TCP Congestion Control

TCP needs to adjust the rate of transmission by perceiving（察觉） the network congestion

- simple congestion detection
  - packet loss
    - timeout
    - Triple-duplicate ACK
  - packet delay
    - RTT estimate

Three question:

(1) How does a TCP sender limit the rate at which it sends traffic into its connection

- Solution

```
Sender side congestion-control mechanism operating at the sender keeps track of an       additional variable, the **congestion window** (denote ```cwnd```), imposes a constraint on the rate at which a TCP sender can send into the network
```

$$LastByteSent - LastByteAcked \leq min\{cwnd, rwnd\} \tag{18}$$

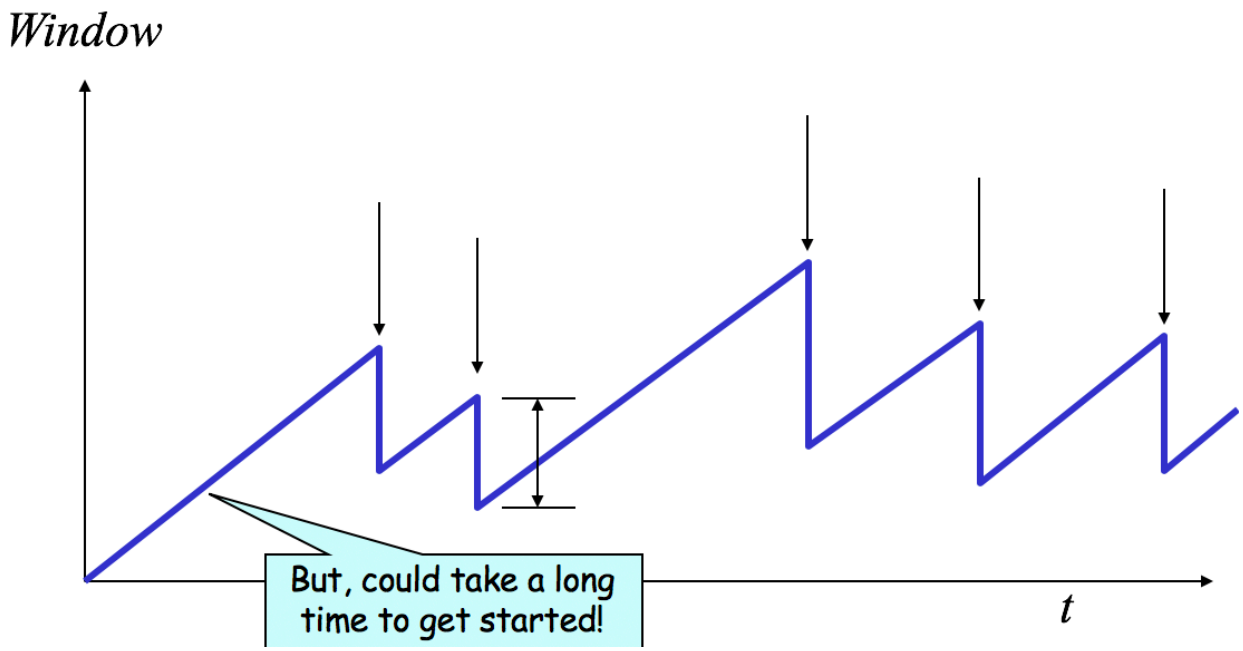
- Suppose : The TCP recieve buffer is so large => the amount of unACKed data at sender is solely limited by `cwnd`

  Thus the sender's send rate is roughly $cwnd/RTT$ *bytes/sec. By adjusting the value of cwnd, the sender can therefore adjust the rate at which it sends data into its connection*

  - stable rate means : the utilization is 1, thus it is always transmitting

(2) How does a TCP sender perceive that there is congestion on the path between itself and the destination

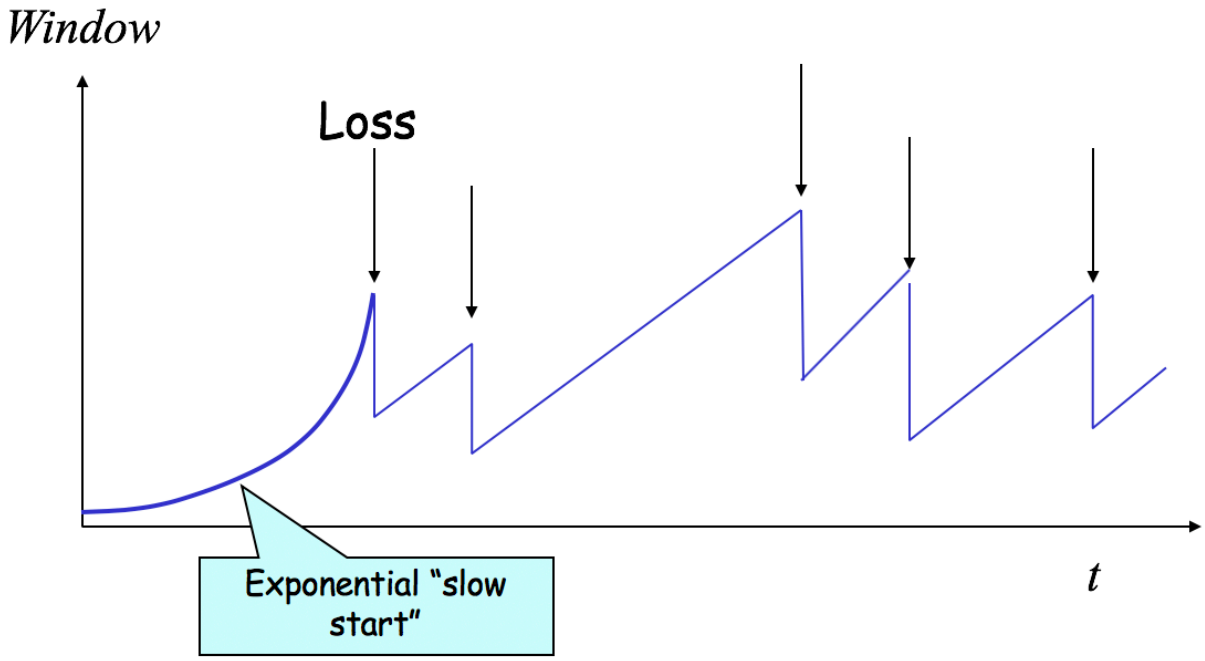
- Define "loss event" at TCP sender as : **Timeout** or **receipt of 3 duplicate ACKs**
- when congestion free
  - when a loss event doesn't occur => Increasing the cwnd size (optimistically exploring)
  - If the ACK arrives at low rate, then the window size will increase in a low rate
  - decrease upon losing a packet(backing off)
  - Use ACK to trigger increasing cwnd => called **self-clocking**
- Control the cwnd (don't congest the network, but make full use of bandwidth) => additive increase, multiplicative decrease



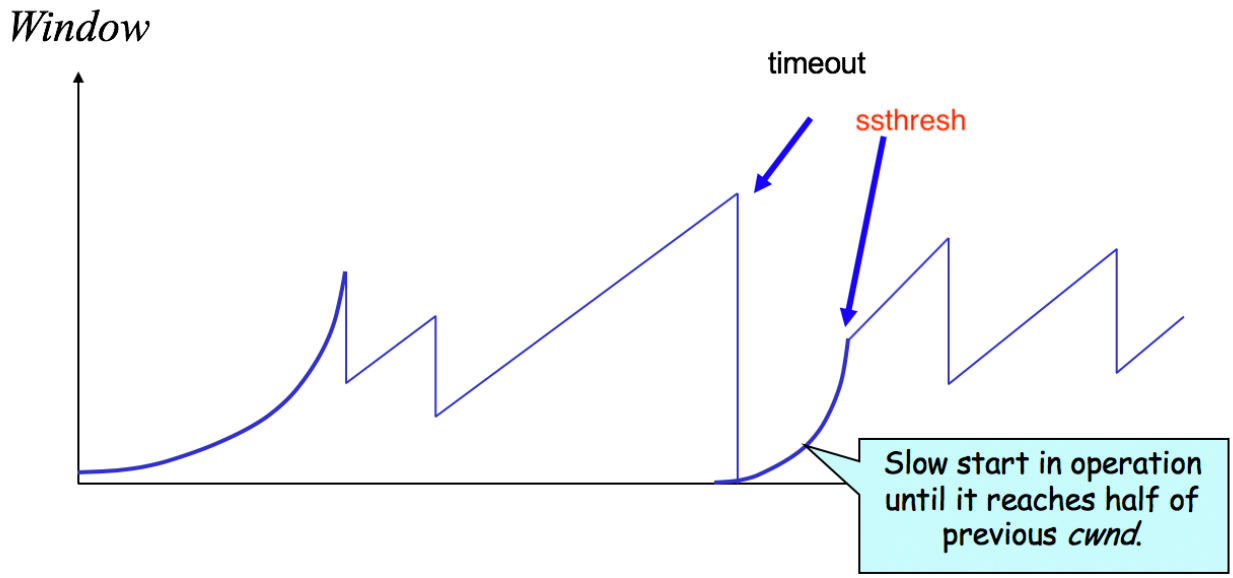
  - Loss => decrease the cwnd(to the half)
  - ACK => increase the cwnd (by 1, increase linearly)
  - Bandwidth probing => increase the rate until loss occurs
  - practical detail:
    - increase by MSS on success for **last window of data**
      - packets per window: $\frac{CWND}{MSS}$
      - Increment per ACK: $\frac{MSS}{(\frac{CWND}{MSS})}$
    - Decrease: never drop congestion window below 1 MSS

(3) What algorithm should the sender use to change its send rate as a function of perceived end-to-end congestion

- slow start : mandatory(必须)



- Initialize : cwnd = 1 MSS, rate = MSS/RTT
- Increase the value of `cwnd` begins at 1 MSS and increases by 1 MSS every time a transmitted segment is first ACKed
  - 1 ACKed packet will increase 1 (i.e. 2 packets from the ACKed packet) => increase exponentially (2^n)
- When to end the increasing (end the start state i.e. the first curve at the following picture)

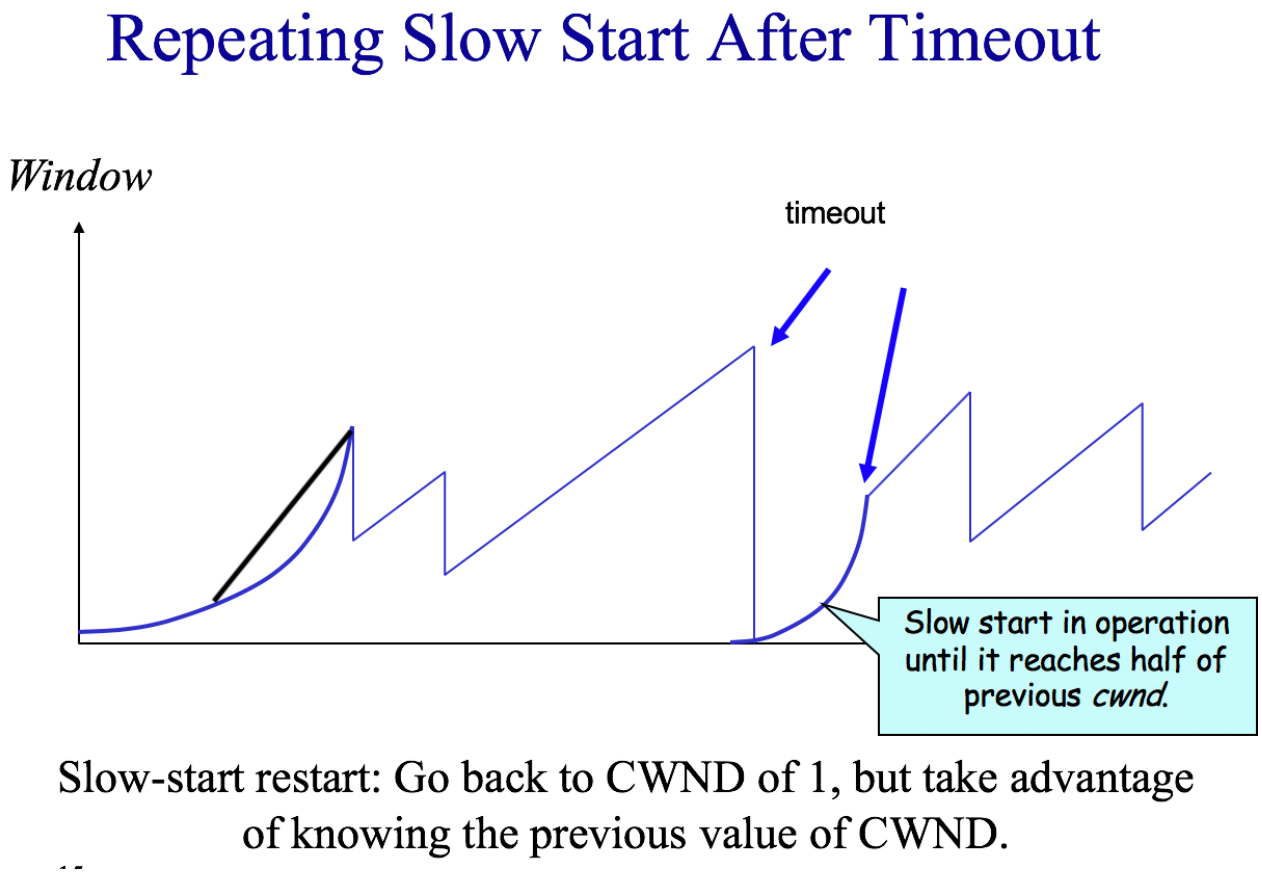


  - The first way: 3 duplicated ACKs received(first 2 deecreases on the graph)
    - do a multiplicative decrease
  - The second way: If there is a loss event(congestion) indicated by a time out (third decrease)
    - TCP sender sets the value of `cwnd` to **1** , because timeout represents that the **packets after the first unACKed packet are all possibly lost**, the network is seriously congested
    - Sets the value of second state variable `ssthresh` to be `cwnd/2` , where `cwnd` is the window size caused time out
    - after that, do a fast recover, slow-start again until meat the `ssthresh` (third step)
  - The third way :
    - Since the `ssthresh` is half of the value last detected, it maybe reckless to keep doubling the cwnd, thus fast recover
- congestion avoidance : mandatory(必须)

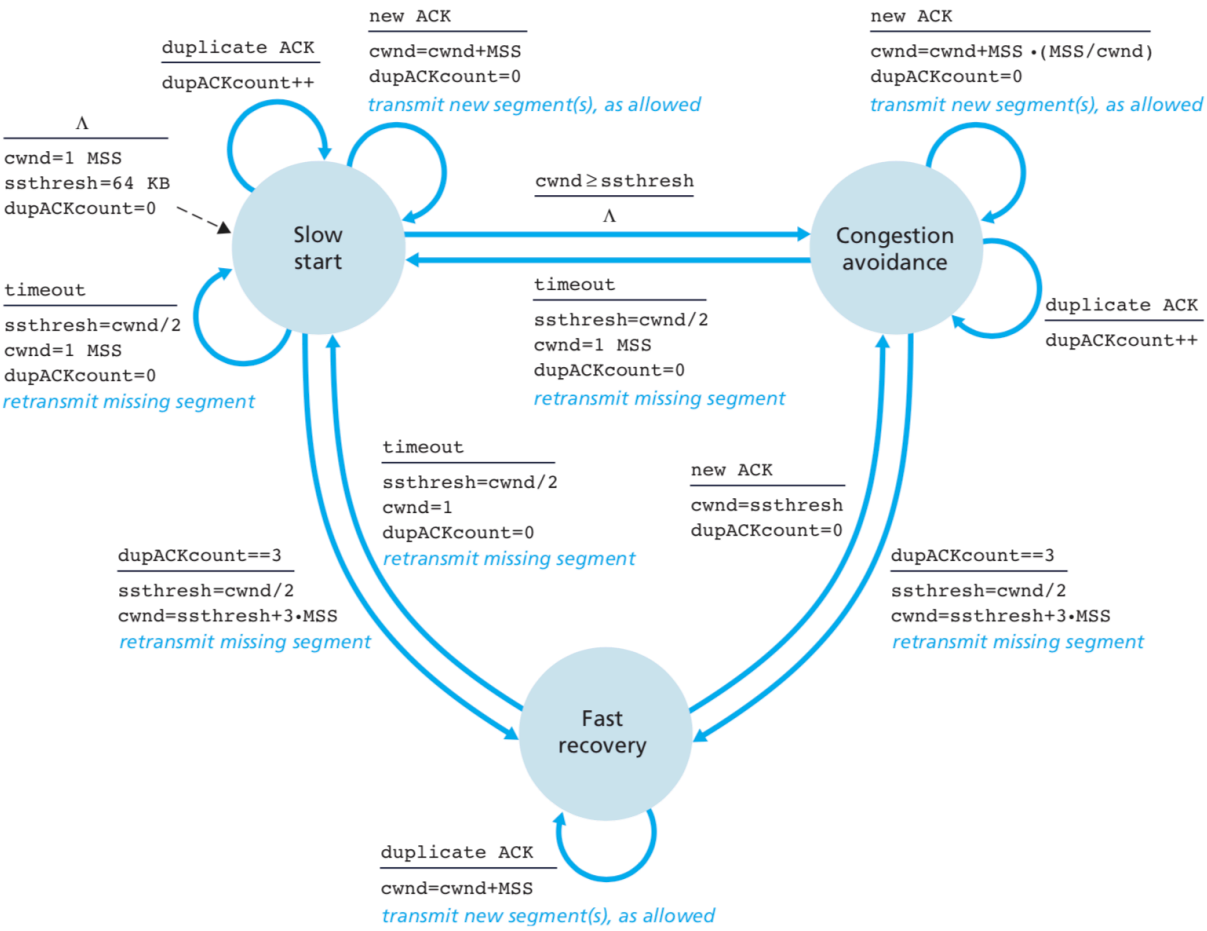
- The value of cwnd is approximately half its value when congestion was last encountered -- congestion could be just around the corner
- Thus increase 1 MSS every RTT=> many ways to accomlish it
  - increase `cwnd` by MSS bytes whenever a new ACK arrives (exclude duplicate)

    ```
    cwnd+=MSS*(MSS/cwnd) // increase per ACK (MSS/num of pkts)
    ```

  - When timeout occurs, the value of `cwnd` set to 1 MSS, value of `ssthresh` updated and slow start again
  - When 3 duplicate ACKs occurs, since only the duplicate one may lost, so just halve the value of `cwnd`



- fast recovery : recommended
  - When duplicate ACK occurs, need to restart => fast start (exponentially increase)
  - until reach the `ssthresh`



- The tutorial
  - Go-back-N must has at most seq-1 window size
  - selective repeat must has most seq/2 window size
  - In TCP, whatever the sender send, the receiver will reply accumlative ACK
  - When using AIMD (addition.. multiple ..) TCP, the average throughput when the pkt size from 4MSS increased to 12MSS is

$$\frac{4MSS + 5MSS + \ldots + 11MSS}{8RTT} \tag{19}$$

Chapter 4 Network Layer

- Scope of the protocol: every host, router
- Two main function of network layer:
  - forwarding function
  - routing function
- Difference between Forwarding and routing functions of network layer:
  - Forwarding : The transfer of a packet from an **incoming link** to an **outgoing link** within a *single* router (getting through single interchange)
    - Look inside the router => at its harware architecture and orgnization
    - Look at packet formarding in the Internet scope => IP and network address translation(NAT)
  - Routing : Involves all of a network's routers, whose collective interactions via routing protocols determine the paths that packets take on their trips from sourse to destination node(process of planning trip from source to destination)
    - network layer routing function => the job of a routing algorithm is to determine good paths

        - Link-state algorithm
        - Distance-vector algorithm
    - Complexity of routing algorithms grows considerably as the number of network routers increases => Hierarchical routing approaches will also be interest

---

4.1 Introduction

4.1.1 Forwarding and Routing

- Definition : The role of network layer just to move packets from a sending host to a receiving host => 2 function are identified
    - *Forwarding* : When a packet arrives at a router's input link, the router must move the packet to the appropriate output link (all links are stored in the router as an interface)
    - *Routing* : The network layer must determine the route or path taken by packets as they flow from a sender to a receiver. The algorithms that calculate these paths are referred to as **routing algorithms**
- **Forwarding table** : Every router has a forwarding table, a router forwards a packet by examining the value of a field in the arriving packet's header, and then using this header value to index into the router's forwarding table. (Key=header , value=output link)
    - Depending on the network-layer protocol, the header value could be the *destination address* of the packet or an *indication of the connection to which the packets belong* (hierarchy structure)
    - How farwarding tables are configured ? => exposes important interplay between routing and forwarding
        - Routing algorithm determines the values that are inserted into the routers' forwarding tables (router receive a routing protocol message in either(all) case below)
            - The algorithm may be centralized (executing on a central site and downloading the routing information to each of the routers)
            - Also may be decentralized (with a piece of teh distributed routing algorithm running in each router)
- packet switches : A general packet-switching device that transfers a packet from input link interface to output link interface
    - **Link-layer switches** : Base their forwarding decision on values in the **fields of the link-layer frame** => thus switches(generally) refered to as link-layer devices
    - **Routers** : Base their forwarding decision on the value in the **network-layer field** => thus routers are network-layer devices (but must implement layer2 protocols as well)
- Connection Setup (third important network-layer function)
    - This function allows the sender and receiver to set up the needed state information (similar to handshaking)

4.2 Datagram Networks

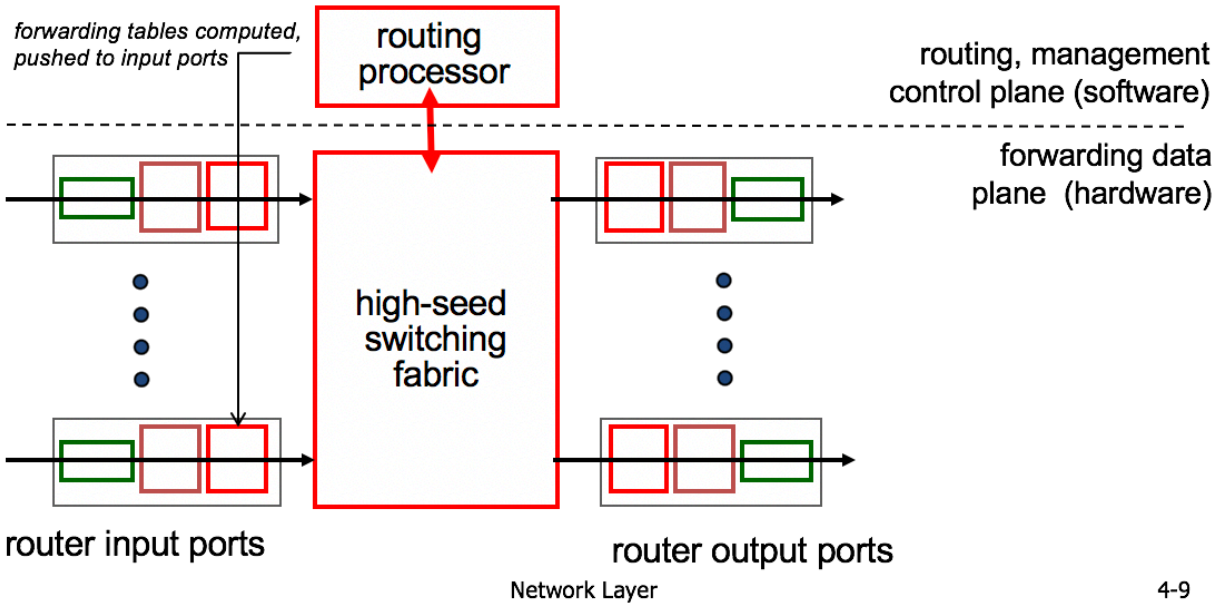
- Datagram network : In a datagram network, each time an end system wants to send a packet
    - Beginnig : it **stamps**(贴邮票) the packet with the address of the destination end system and then pops the packet into the network
    - Transmitting : Passes through a series of routers => each of them uses the packet's destination address to forward the packet
    - In a router : The routers use the packet's destination address to look up the **appropriate output link interface** (likes a gate to the right path) in the forwarding table
- Longest prefix matching : Suppose that all dest addresses are 32 bits (length of the destination address in an IP datagram)
    - Brute-force implementation : Forwarding table would have 1 entry for every possible destination address => more than 4 billion possible addresses => too comlex=>make groups
    - Further suppose that our router has four links, numbered 0 through 3

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

the packet goes in to the interface which has **longest match digits** with its prefix

4.3 What's Inside a Router

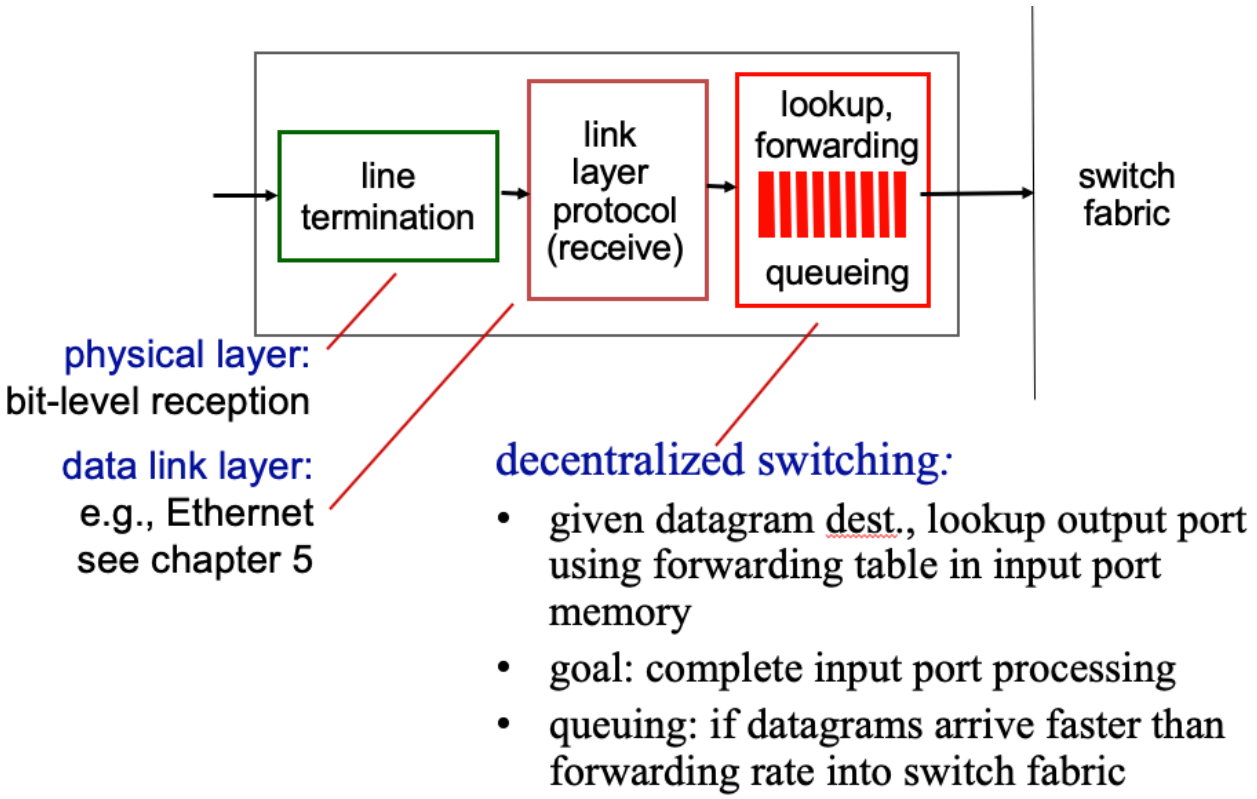
- Terminology : *Forwarding* is same as *switching*
- Architecture :
  - *Input ports* : several key functions
    - Physical layer function of terminating an incoming physical link at a router (used in left most box input and right most box in output)
    - Link-layer function to interoperate with the link layer at the other side of the incoming link (middle box in 2 sides)
    - Look up function determine the router **output port** to which an arriving packet will be forwarded via the switching fabric.
      - *Control packets* (special packet carrying routing protocol information) are forwarded from an input port to the routing processor
      - *port* here means **physical input** and output interface
  - *Switching fabric* : connects the router's inout ports to its output ports
  - *Output ports* : An output port stores packets received from the switching fabric and transmits theses packets on the outgoing link by performing the necessary link-layer and physical-layer functions
    - Link are bidirectional(both directions traffic), output port will typically be paired with input port for that link on the same line card
  - *Routing processor* : execute the routing protocols
  - control plane : routing service
  - forwarding plane : forwarding service



Network Layer 4-9

4.3.1 Input/Output Processing

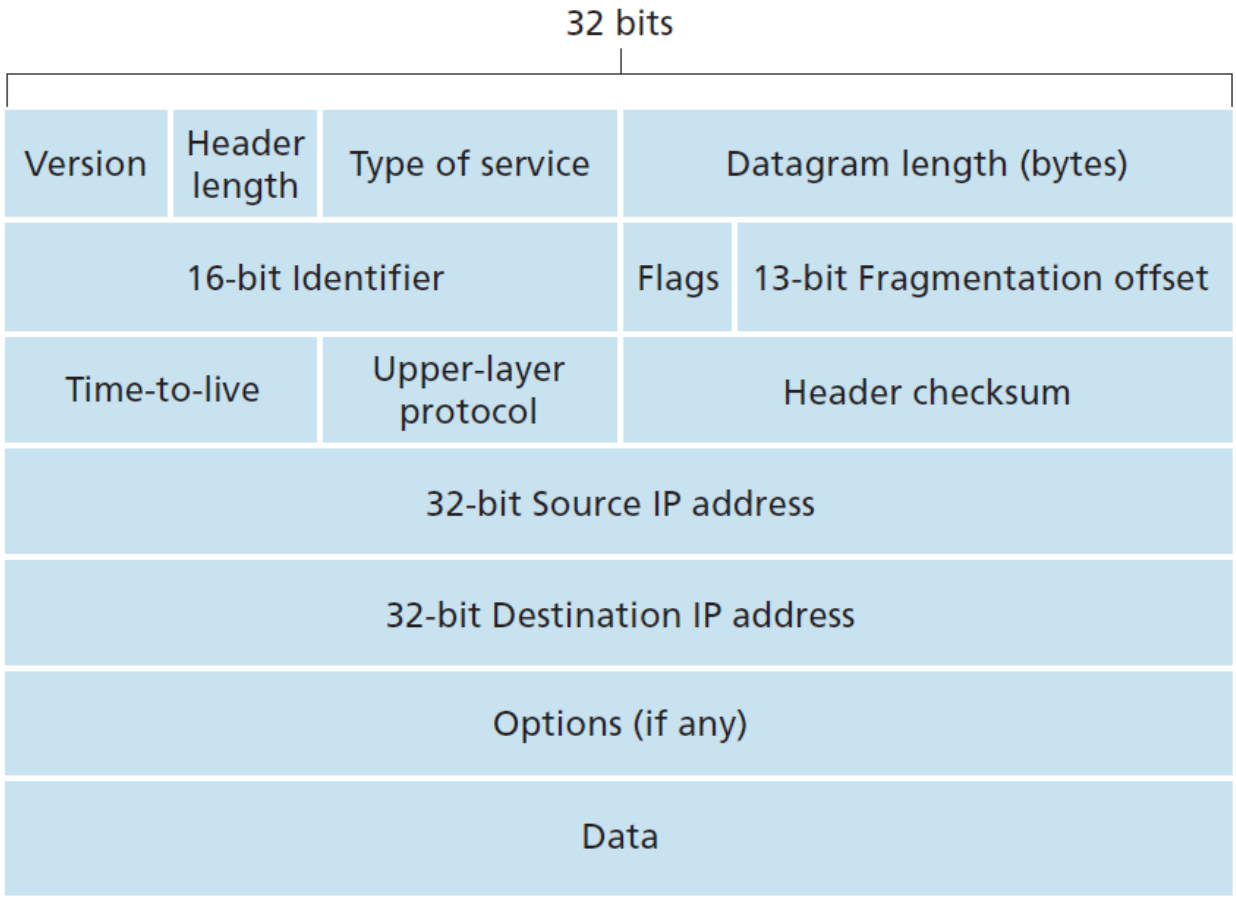
- Input
  - Forward table computed by the routing processor, and copied by the line cards over a separate bus (match plus action (look up the table and send the data))
  - queuing: if datagrams arrive faster than forwarding rate into switch fabric
  - Decentralize switching:



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

4.4 The IP : Forwarding and Addressing in the Internet

4.4.1 Datagram Format

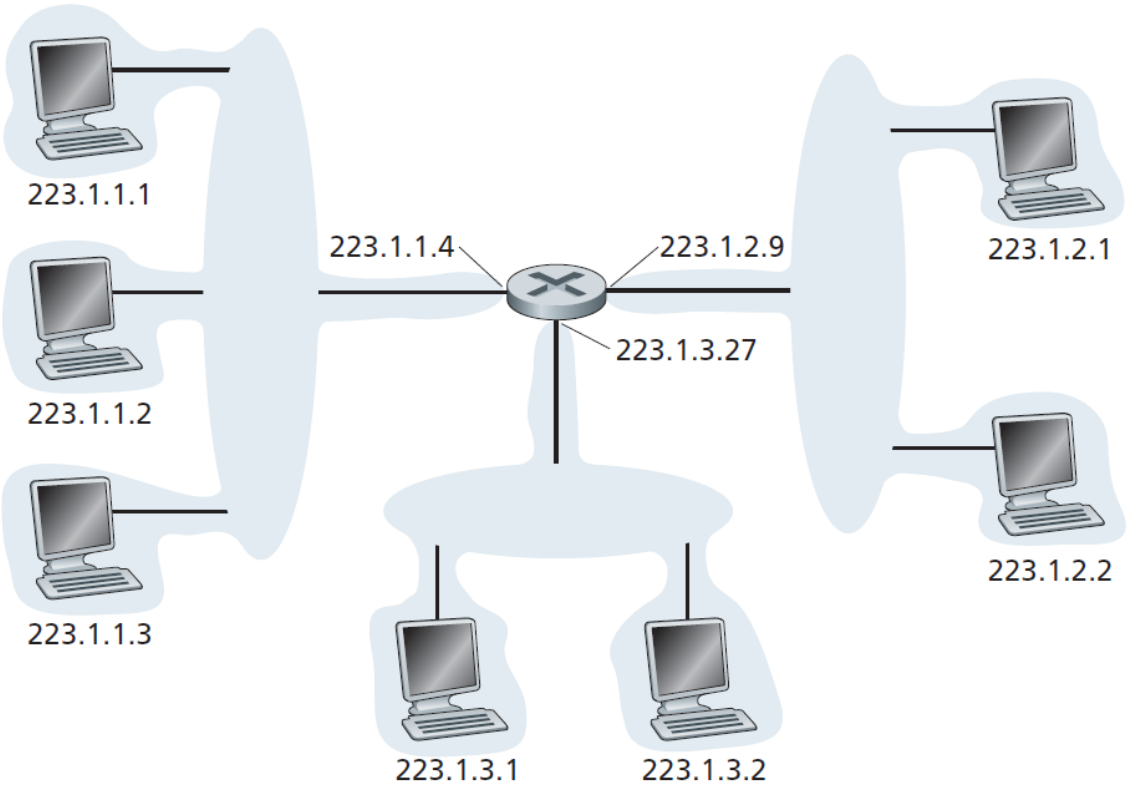


- Terminologies
  - *type of service* : The type of service (TOS) bits were included in the IPv4 header to allow different types of IP datagrams to be distinguished from each other
  - 16-bits identifier : length of the header
  - *Time to live* : The time-to-live (TTL) is included to ensure that datagrams do not circulate forever (may due to a long-lived routing loop) in the network
  - *Protocol* : Used only when IP datagram reaches its final destination => indicates the specific transport-layer protocol to which the data portion of this IP datagram should be passed (glue that binds the network and transport layers together)
  - *options* : Allow an IP header to be extended => header options were meant to be used rarely => hence the decision to save overhead by not including the information in options fields in every datagram header
  - Total : suppose IP datagrams has a total of 20 bytes of header.

4.4.2 IPv4 Addressing

- IP address basic
  - Host-Router link : A host typically has only a single link into the network
  - **Interface** : The boundary over the host and the single **physical link**
  - A router thus has multiple interfaces, one for each of its links
  - Because every host and router is capable of sending and receiving IP datagrams, IP requires each host and router interface to have its own IP address.
  - **IP address** is technically associated with an interface rather than with the host or router containing that interface
  - Each IP address is 32 bits long, and there are thus a total of 2^{32} possible IP addresses about 4 billion possible IP addresses.

Use dotted-decimal notation (convert every 8 bits to the decimal)
    - **In the same scope**, the IP address should be unique
- **Subnet** : Network interconnecting three host interfaces and one router interface forms a subnet



    - IP addressing assigns an address to this subnet: 223.1.1.0/24 (/24 notation is a **subnet mask**) => left most 24 bits of the 32-bit quantity define the subnet address
        - mask(24) written as 11111111 11111111 11111111 00000000, '1' means it belongs to the network portion, '0' belongs to the host portion
        - **prefix** : network portion of the address
        - **host** : host portion of the address
    - IP definition of a subnet is *not* restricted to Ethernet segments that connect multiple hosts to a router interface.

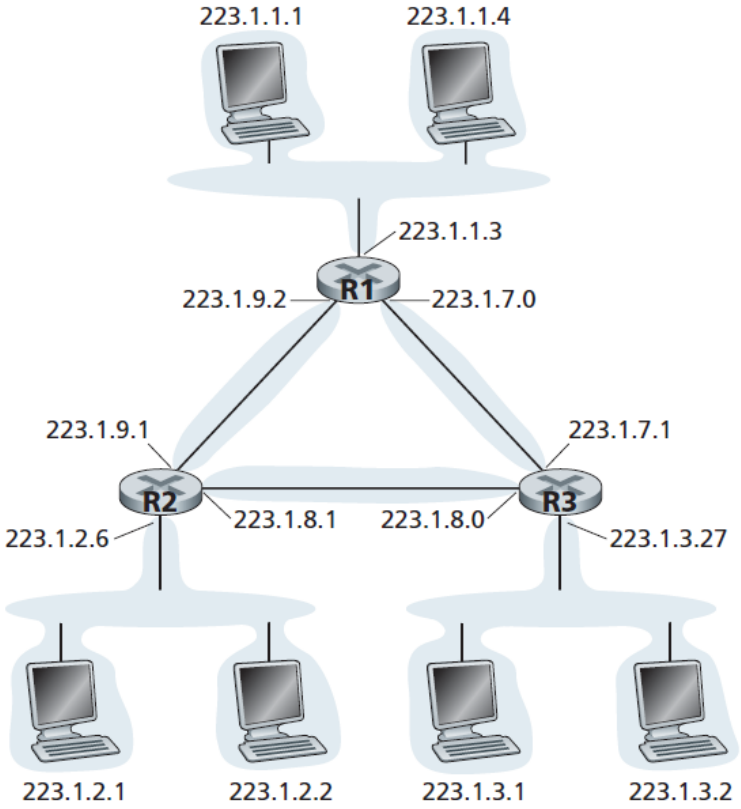


Figure 4.17 ♦ Three routers interconnecting six subnets

- 233.1.9.0/24, 223.1.8.0/24, 223.1.7.0/24 are also subnet
- to determine subnet, treat the host and routers as end points, the connected parts is a subnet

- Global Internet
  - With multiple Ethernet segments and point-to-point links will have multiple (all devices on a given subnet having the same subnet addresses)
  - In principle, the different subnets could have quite different subnet addresses, but in practice, their subnet addresses often have much in common
- **Classless Interdomain Routing** (CIDR) DONNOT TEST PART
  - Rule : the 32-bits IP address is divided into two parts and again has teh dotted-decimal form $a.b.c.d/x$ cconstitute the network protion of the IP address (refered to as the **prefix**)
  - ISP advertises to the outside world that it should be sent any datagram whose *network prefix* of first n bits are match a mask(refrered as **address aggregation**)
  - If one host(or subnet) of one subsidiary moves to another, the host (or subnet) => The subsidiary it goes to now also advertises the bolock of addresses of the move host
  - When other routers in the larger Internet see the address blocks 200.23.16.0/20 (from Fly-By-Night-ISP) and 200.23.18.0/23 (from ISPs-R-Us) and want to route to an address in the block 200.23.18.0/23, they will use longest prefix matching (see Section 4.2.2), and route toward ISPs-R-Us, as it advertises the longest (most specific) address prefix that matches the destination address.

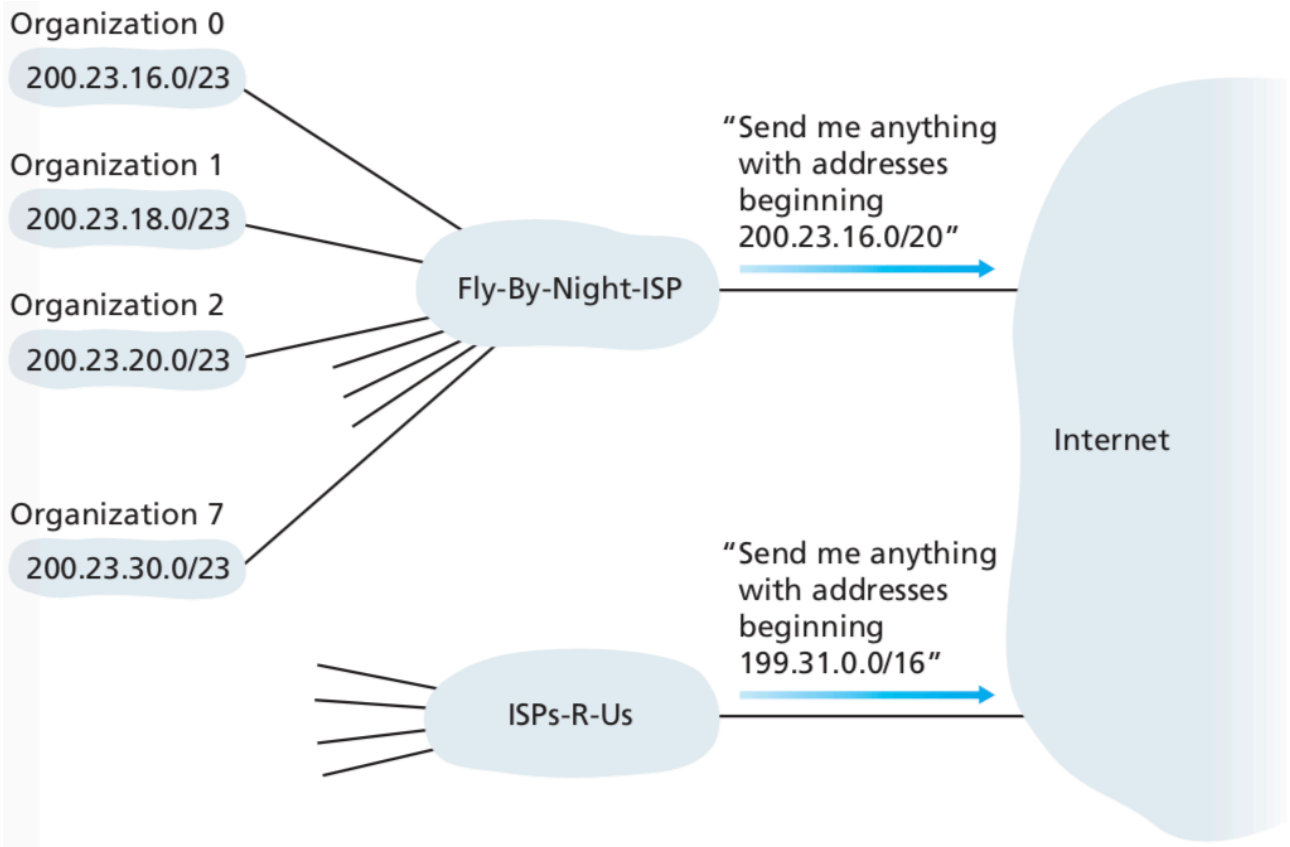
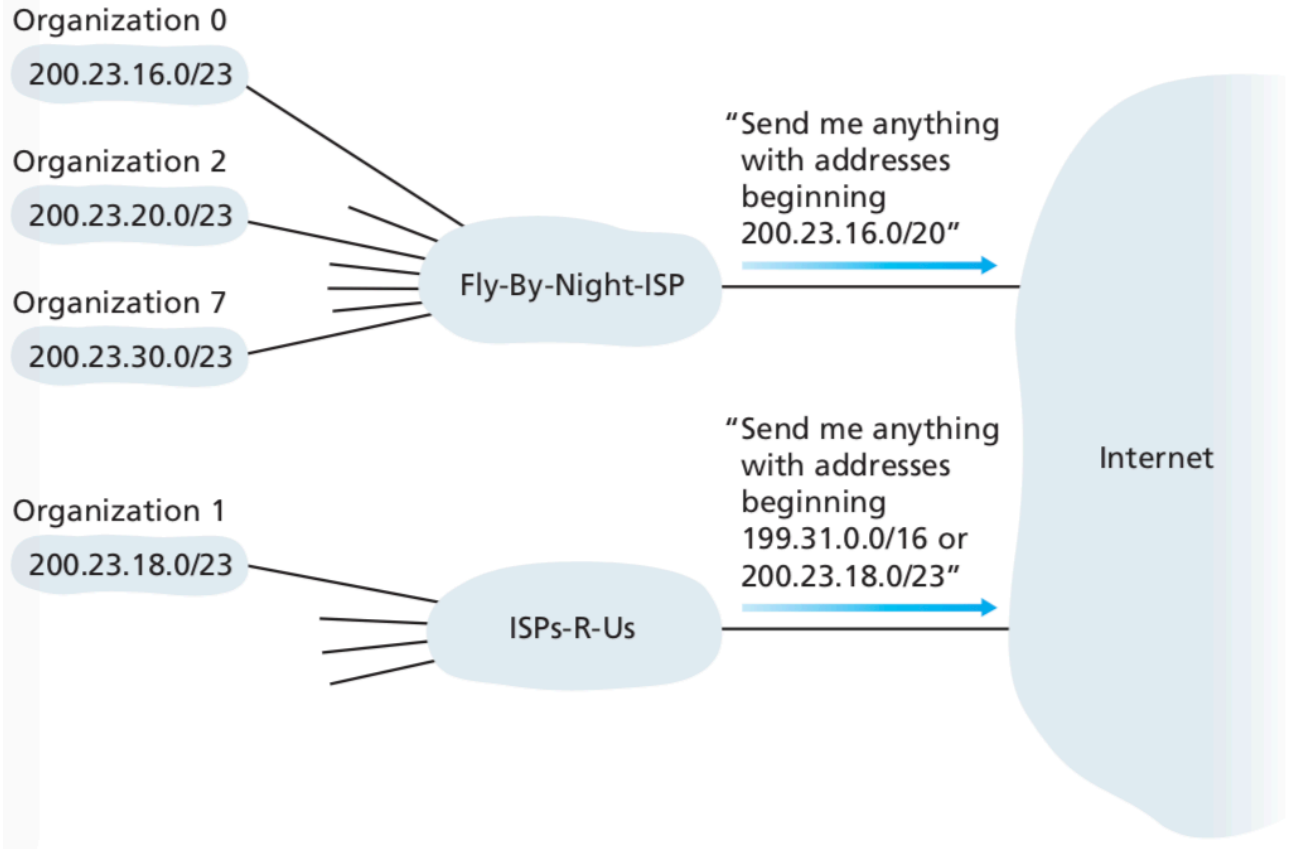


Figure 4.18 ♦ Hierarchical addressing and route aggregation



- remain 32-x bits of an address can be thought of as distinguishing among the device s *within* the organization, all of which have the same network prefix. These are the bits that used when forwarding packets at routers within the organization, (
- Remain bits may have an additional subnetting structure, i.e. it identify the specific hosts in the organization

- The network portions of an IP address were constrained to be 8, 16, or 24 bits in length, an addressing scheme known as **classful addressing** (before teh CIDR is invented)
  - 8- : class A => 1 bytes
  - 16- : class B => 2 bytes(inner can most holds $2^{16} - 2$ =65634 hosts, 2 numbers are for special use, one is the address of subnetwork, the other is the broadcast address)
  - 24- : class C => 3 bytes (inner can most holds $2^{8} - 2$ =254 hosts, 2 numbers are for special use)
  - The class B is too large(wasteful) and class C is too small => use mask for any bits
  - get the prefix and remain
- Calculate 2 parts of address by using mask

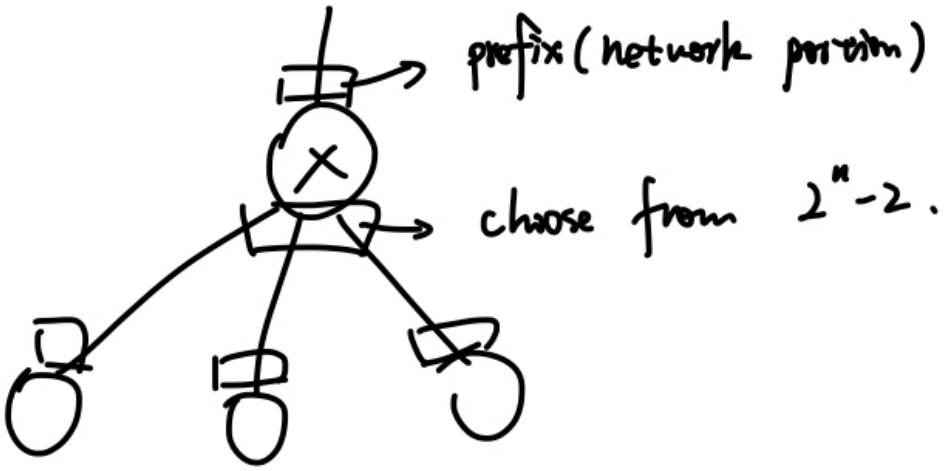
Working with Addresses

Write down the IP address.	11000000 10101000 01100100 01010000 192 168 100 80
If you have a prefix length, just wrote down the number of 1's. If you have a network mask, computer the binary as with the IP address.	11111111 11111111 11111111 11000000 8 +8 +8 +2 == 26
AND these two.	11000000 10101000 01100100 01000000
Convert back to dotted decimal. This is the network address.	192 168 100 64

Working with Addresses

Write down the IP address.	11000000 192	10101000 168	01100100 100	01010000 80
If you have a prefix length, just wrote down the number of 1's. If you have a network mask, computer the binary as with the IP address.	11111111 8	11111111 +8	11111111 +8	11000000 +2 == 26
Inverse every bit in the mask	00000000	00000000	00000000	00111111
AND IP address with the inversed mask	00000000	00000000	00000000	00010000
Convert back to dotted decimal. This is the host address.	0	0	0	16

- IP broadcast address
  - When a host sends a datagram with destination address 255.255.255.255, the message is delivered to all hosts on the same subnet (Router optionally(可以选择) forward the message into neighboring subnets as well)
  - *calculate the number of host can be held* : exclude the prefix (all '0') and broadcast (all '1') => $2^n - 2$, **the subnet side interface of the router is also assigned an IP address in this scope**.



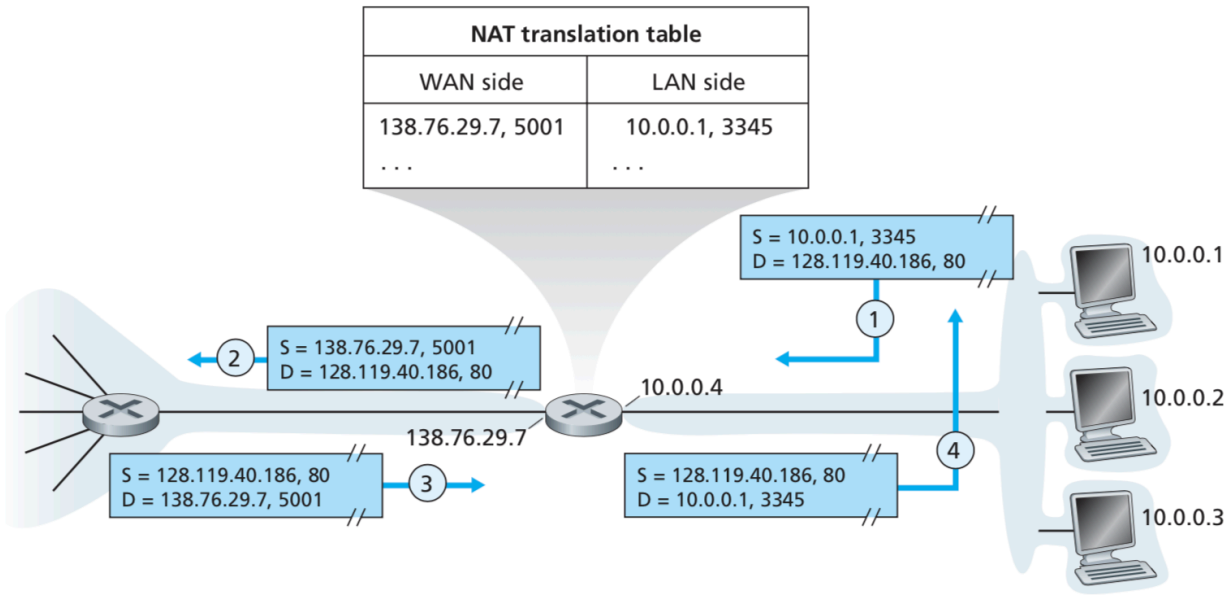
- Obtaining a Block of Addressses => for subnetwork

- Method 1
  - In order to obtain a block of IP addresses for use within an organization's subnet, a network administrator might first contact its ISP, which would provide addresses from a larger block of addresses that had already been allocated to the ISP(ask which portion to use)

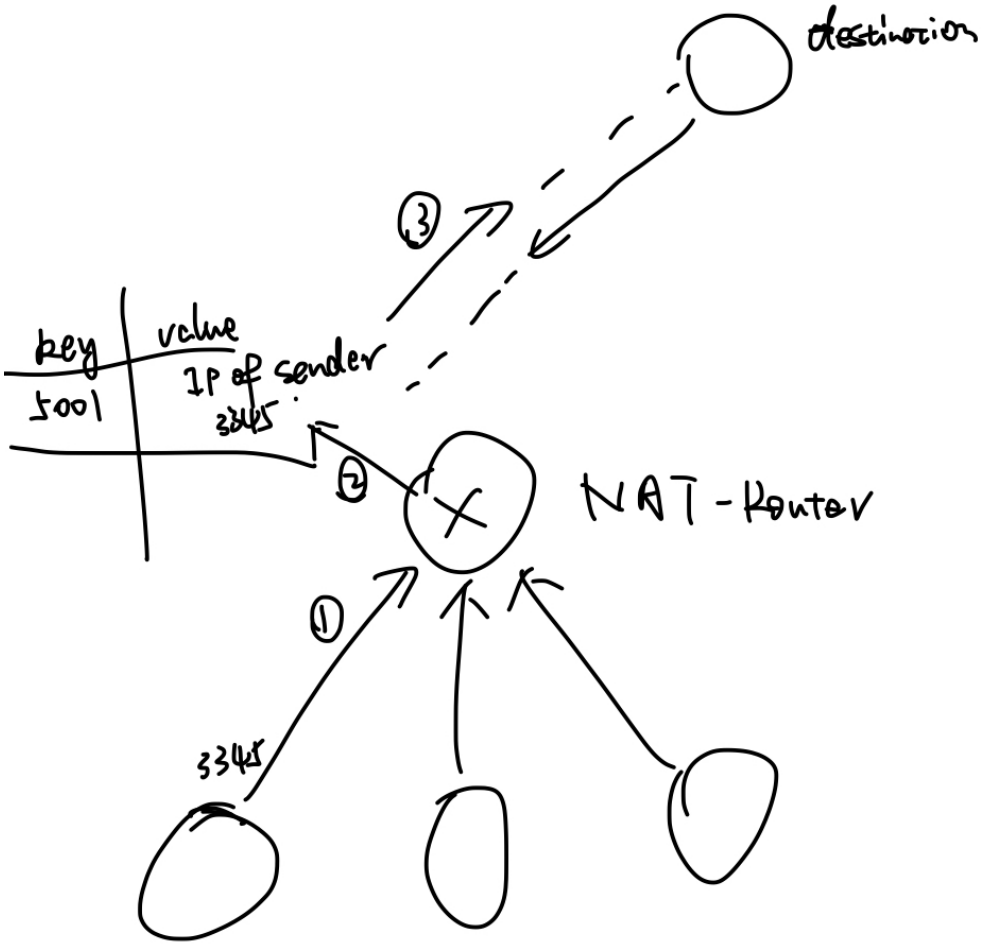
ISP's block	200.23.16.0/20	11001000 00010111 00010000 00000000
Organization 0	200.23.16.0/23	11001000 00010111 00010000 00000000
Organization 1	200.23.18.0/23	11001000 00010111 00010010 00000000
Organization 2	200.23.20.0/23	11001000 00010111 00010100 00000000
. . .	. . .	. . .
Organization 7	200.23.30.0/23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

  - For example, the ISP may itself have been allocated the address block 200.23.16.0/20. The ISP in turn, could divide its address block into eight equal-sized contiguous address blocks and give one of these address blocks out to each of up to eight organizations (0-7)
- Method 2 : ISP itself should find a way to get the IP address
  - There is a global authorithy that has ultimate responsibility for managing the IP address space and allocating address blocks to ISPs and other organization
  - Under authority of the Internet Corporation for Assigned Names and Numbers (**ICANN**) (who also manage the DNS root servers) => for ISP

- Obtain a Host Address : the Dynamic Host Configuration Protocol(**DHCP**)=> for a single host(为什么连不同的网络ip不一样)
  - DHCP allows a host to obtain(be allocated) an IP address automatically
  - A network administrator can configure DHCP so that a given host receives the same IP address each time it connects to the network (assignt the first time, remember then)
  - Or a host may be assigned a **temporary IP address** that will be different each time the host connects to the network => DCHP is a **plug-and-play protocol** (即插即用)
  - DHCP also allows a host to learn additional information, such as its subnet mask, the address of its first-hop router, the address of its local DNS server
- Network Address Translation (NAT)
  - The situation : If a subnet allocated by the ISP grows bigger=> larger block of addresses would have smartphones and networked => simpler approach to address allowcation : **network address translation (NAT)**



- A **realm**(领域) with private addresses refers to a network whose IP addresses **only have meaning to devices within that network**
- NAT-enabled router
  - NAT-enabled router has an interface that is part of the home network(10.0.0.4), and all four interfaces in the home network have the same subnet address of 10.0.0/24
  - A NAT-enabled router looks like a *single* device with a *single* IP address (i.e. all messages go out of the network has the source IP address of 138.76.29.7) => hiding the details of the home network from the outside world
  - NAT-enabled routers' IP addresses are assigned by the upper DHCP service, and run DHCP servers to assign address to lower hosts
- **NAT translation table** => include port numbers as well as IP addresses in the table entries (key = port number, value = **intra** network IP address)
  - The host (says a PC) ask for a Web server (port 80) with **Inner IP address** 128.119.40.186 .
  - At first, the host itself will assign an arbitrary source port number says 3345 and sends the datagram into the LAN
  - The NAT router receives the datagram, generates a new source port number says 5001 (**different from the Transport layer port, can be any number do not exist in the current translation table**), replace the source IP address with its WAN-side IP address 138.76.28.7, replace the old port number 3345 by the new 5001
  - (source IP, port#)=>(NAT IP,new port# used as a key)
  - The Web server doesn't know the port has been manipulated by the NAT route, responds with a datagram whose destination address is the IP address of the NAT router
  - When NAT server receives the datagram, it search for the 5001 entry on the table, and rewrites the datagram's destination by the appropriate IP address and port stored in the table
  - The NAT router can change out side interface without changing the arrangement inside or change arrangement inside without notifying outside world



4.5 Routing Algorithm

Datagram service : packets go through different routes

VC service : packets go through same path

Recall routing algorithm : From sender to receiver through the network of routers, to find the good paths (i.e. the least cost path)

Terminologies

- **Default router** : The router directly connected to the host is called default router for **the host** . (also called **first-hop router**)
  - **source router** : default router of source host
  - **destination router** : default router of destination host
- **graph** : G = (N,E) is a set N of nodes and a collection E of edges, where each edge is a pair of noes from N. In computer network, Node is routers, edge is physical links
- For any edge (x, y) in E, we denote $c(x, y)$ as the cost of the edge between nodes x and y, if (x, y) does not belongs to E, we set c(x,y) to be ∞ , and we only concern undirected graphs i.e. $c(x, y) = c(y. x)$
- Cost of path :

$$c(x_1, x_2, x_3, \ldots, x_p) = c(x_1, x_2) + c(x_2, x_3) + \ldots + c(x_{p-1}, x_p) \tag{20}$$

- Least-cost paths are paths have the least cost of path from source to destination. When the cost of all paths are the same, the least-cost paths are also shortest paths
- Types of routing algorithms
  - **global routing algorithm** :
    - the machine compute it using complete, global knowledge about the network
    - takes the connectivity between all nodes and all link costs as inputs
    - require the algorithm obtain this information before actually performing the calculation
    - Refered to as **link-state(LS) algorithms**
  - **decentrailized routing algorithm** :
    - Calculation of the least-cost path is carried out in an iterative, distributed manner.
    - Each node begins with only the knowledge of the costs of its own directly attached links
    - Iterativly update datas to neighbors (distance-vector algorithm)
- Another way of dividing the algorithm
  - **static routing algorithms** : routes change very slowly overtime, often as a result of human intervention
  - **dynamic routing algorithms** : routes change more frequently
    - Can be run either periodically or in direct response to topology or link cost changes
- Third way to classify
  - **load-sensitive algorithm** : link costs vary dynamically to reflect the current level of *congestion* int the underlying link

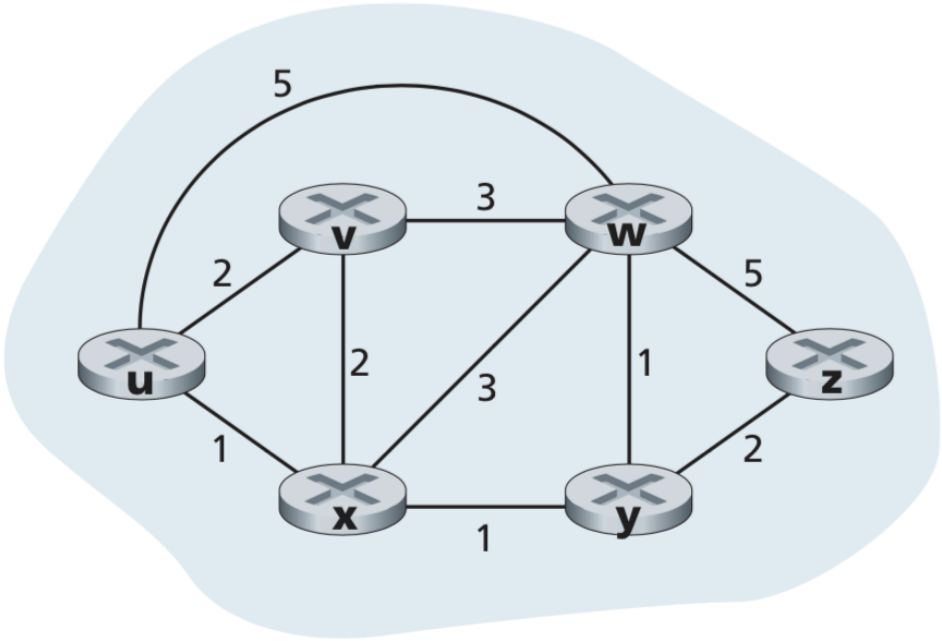
4.5.1 The Link-State (LS) Routing algorithm

- Way to get the information : Each node broadcast link-state packet containing the identities and costs of its attached links (**link-state broadcast** algorithm).
- *Dijkstra's algorithm*
  - In Dijkstra's algorithm, the source router is the one who do the calculation
  - Define the notation:
    - $D(v)$: cost of the least-cost path from the source node to destination v as of this iteration of the algorithm.
    - $p(v)$: previous node (neighbor of v) along the current least-cost path from the source to v.
    - N' : subset of nodes; v is in N if the least-cost path from the source to v is defin- itively known.
  - Illustration

```
Initialization:
N' = {u} //only one elelment at first
for all nodes v
```

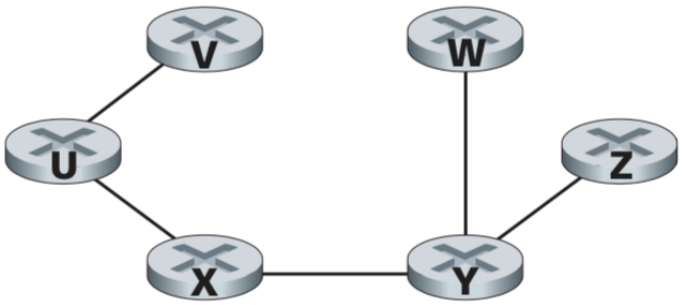
```
  if v is a neighbor of u
    then D(v) = c(u,v)
  else D(v) = ∞
Loop
  find w not in N' such that D(w)is a minimum //minimum means can't be
updated
  add w to N'
  update D(v) for each neighbor v of w and not in N':
  D(v) = min( D(v), D(w) + c(w,v) )
/* new cost to v is either old cost to v or known
least path cost to w plus cost from w to v */
until N'= N
```

- Example



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

- After running the LS algorithm, all nodes store the least-cost path from itself to a series of destinations, so the forwarding table just store the next-hop node on the least-cost path from current router to the destination (u remembers x, x remembers y,if the target is z) => directly stored in the forwarding table of u



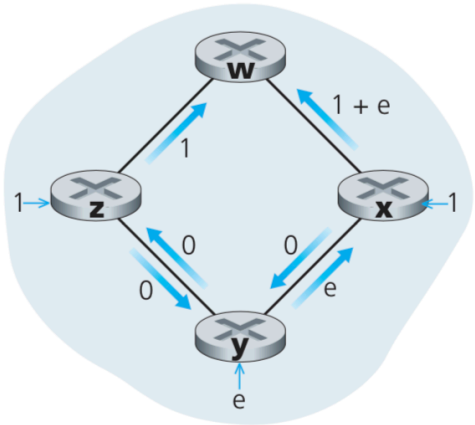
Destination	Link
v	(u, v)
w	(u, x)
x	(u, x)
y	(u, x)
z	(u, x)

- Complexity:
  - Each iteration: check all nodes not in N:

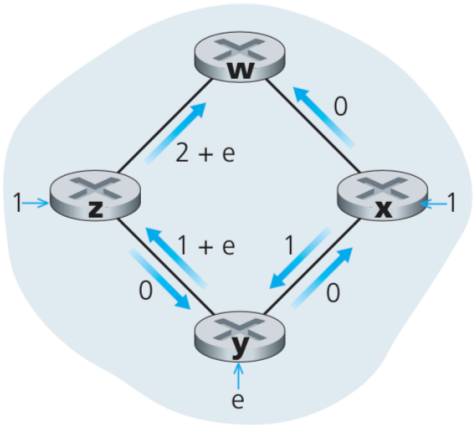
$$n + (n-1) + \ldots + 1 = \frac{n(n+1)}{2} = O(n^2) \tag{21}$$

  - possibly $O(nlogn)$, by using a heap rather than loop

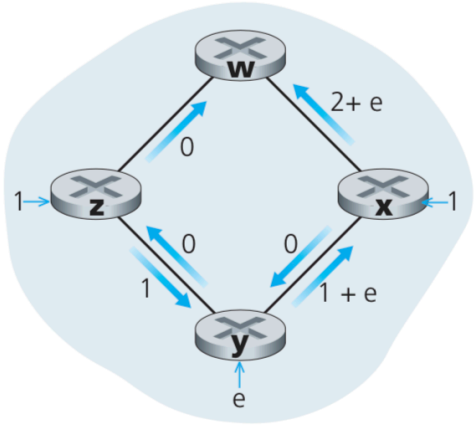
- Oscillation problem
  - When the cost may depends on the load, change the route will affect next output.



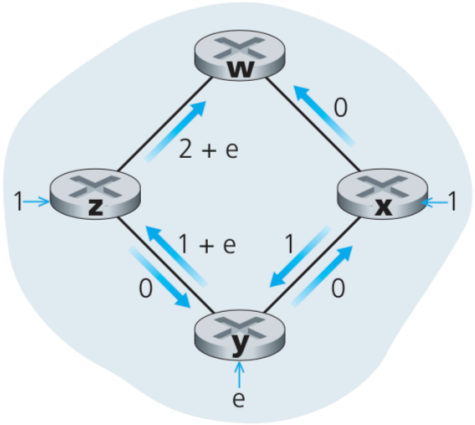
a. Initial routing



b. *x*, *y* detect better path to *w*, clockwise



c. *x*, *y*, *z* detect better path to *w*, counterclockwise



d. *x*, *y*, *z*, detect better path to *w*, clockwise

  - This situation can occur in any algorithm, not just LS

- Solution : Ensure that not all routers run the LS algorithm at the same time (Routers in the Internet can self-synchronize among themselves)

4.5.2 The Distance-Vector (DV) Routing Algorithm

- Each node receives information from one or more of its **directly attached** neighbors and do calculation, then send the result of its calculation back => do until no more information is exchanged between neighbors
- Denote $d_x(y)$ as the cost of the least-cost path from x to y, the least costs are related by the celebrated Bellman-Ford equation (min_v is the minimum value among all neighbors (a vector) of x)

$$d_x(y) = min_v\{c(x,v) + d_v(y)\} \tag{22}$$

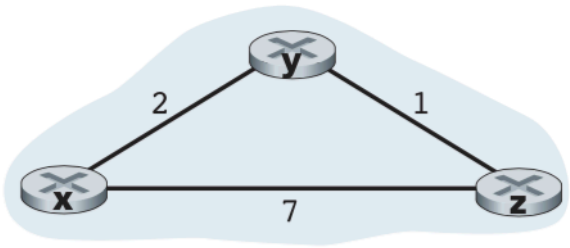
- The solution provides the entries in node x's forwarding table (v* gets by recursion is stored in the forwarding table)
- Each node x begins with $D_x(y)$, an **estimate** of the cost of the least-cost path from itself to node y, for all nodes in N(the whole network). Let $D_x = [D_x(y) : y\ in\ N]$ be node x's distance vector, which is the vector of cost estimates from x to all other nodes, y, in N. With the DV algorithm, each node x maintains the following routing information:
  - c(x,v) v is neighbor of x
  - x's itself distance vector D_x
  - distance vector of **each of its neighbors**, $D_v = [D_v(y) : y\ in\ N]$
- From time, each node sends a copy of its own distance vector to each of its neighbors, the receive re-calculate

$$D_x(y) = min_v\{c(x,v) + D_v(y)\} \tag{23}$$

- Illustration

```
Initialization:
  for all destinations y in N:
    Dx(y) = c(x,y) /* if y is not a neighbor then c(x,y) = ∞ */
  for each neighbor w
    Dw(y) = ? for all destinations y in N
  for each neighbor w
    send distance vector Dx = [Dx(y): y in N] to w
loop
  wait (until I see a link cost change to some neighbor w or
      until I receive a distance vector from some neighbor w)
for each y in N:
  Dx(y) = minv{c(x,v) + Dv(y)}
if Dx(y) changed for any destination y
  send distance vector Dx = [Dx(y): y in N] to all neighbors
forever
```

- Example



Node x table

cxy and cxz is the direct link cost

from	cost to x	cost to y	cost to z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

from	cost to x	cost to y	cost to z
x	0	2	3
y	2	0	1
z	7	1	0

Dy

Dz

from	cost to x	cost to y	cost to z
x	0	2	3
y	2	0	1
z	3	1	0

(2,0,1)=>(2,2+0,7+1)

(7,1,0)=>(7,2+1,7+0)

Node y table

from	cost to x	cost to y	cost to z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

from	cost to x	cost to y	cost to z
x	0	2	7
y	2	0	1
z	7	1	0

from	cost to x	cost to y	cost to z
x	0	2	3
y	2	0	1
z	3	1	0

Node z table

from	cost to x	cost to y	cost to z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

from	cost to x	cost to y	cost to z
x	0	2	7
y	2	0	1
z	3	1	0

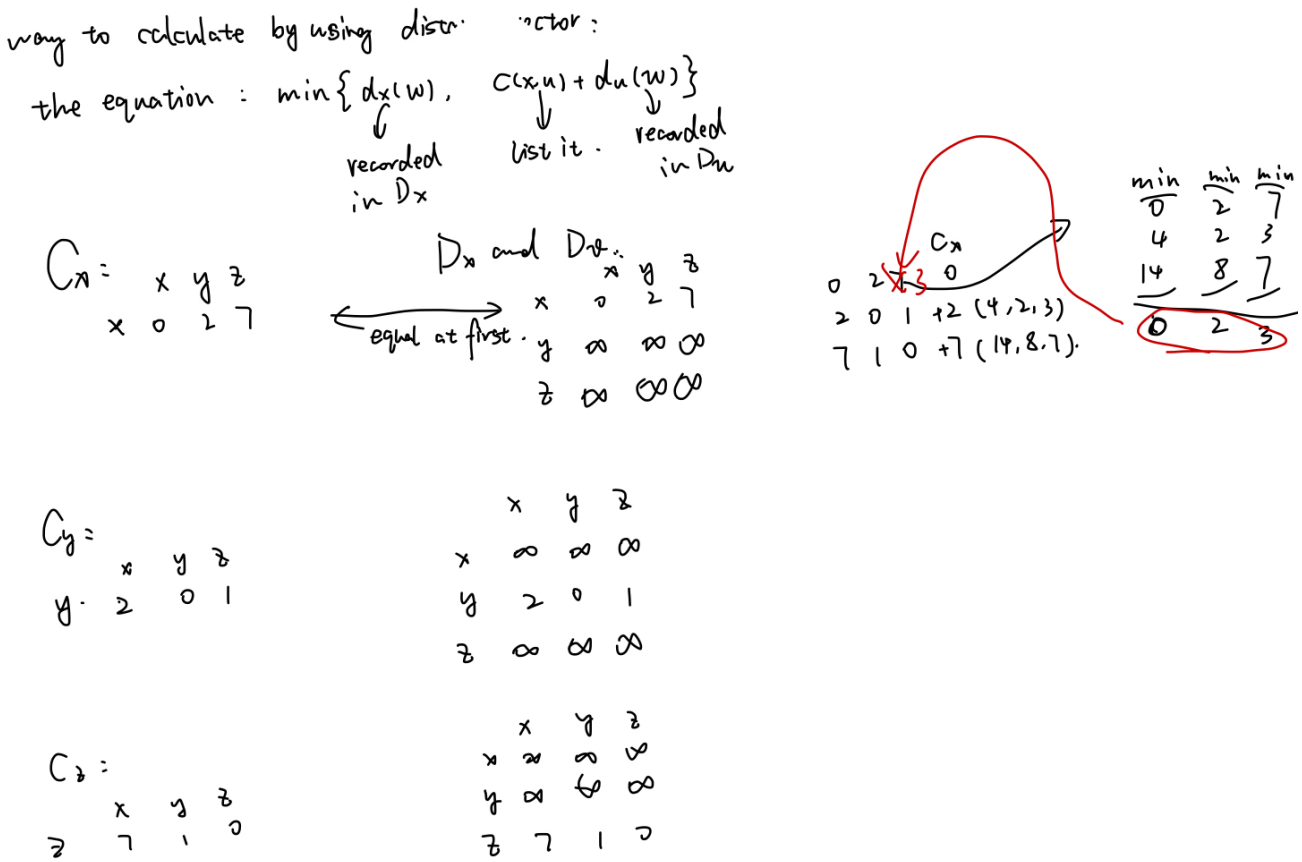
from	cost to x	cost to y	cost to z
x	0	2	3
y	2	0	1
z	3	1	0

Time

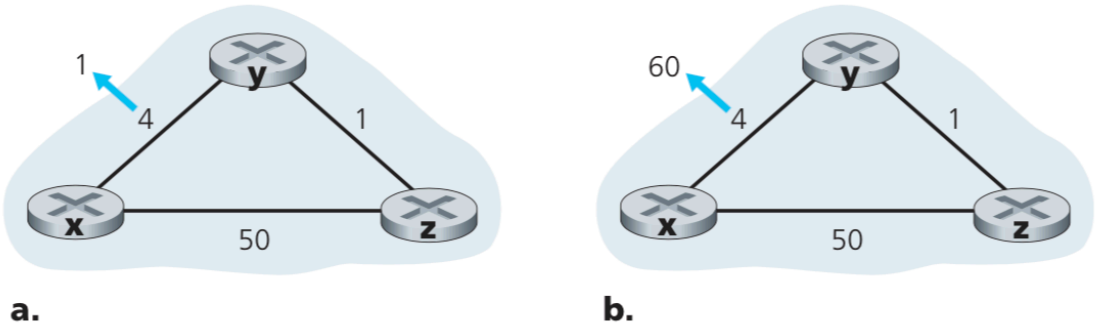
$$D_x(x) = 0$$

$$D_x(y) = \min\{c(x,y) + D_y(y),\ c(x,z) + D_z(y)\} = \min\{2 + 0,\ 7 + 1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z),\ c(x,z) + D_z(z)\} = \min\{2 + 1,\ 7 + 0\} = 3$$

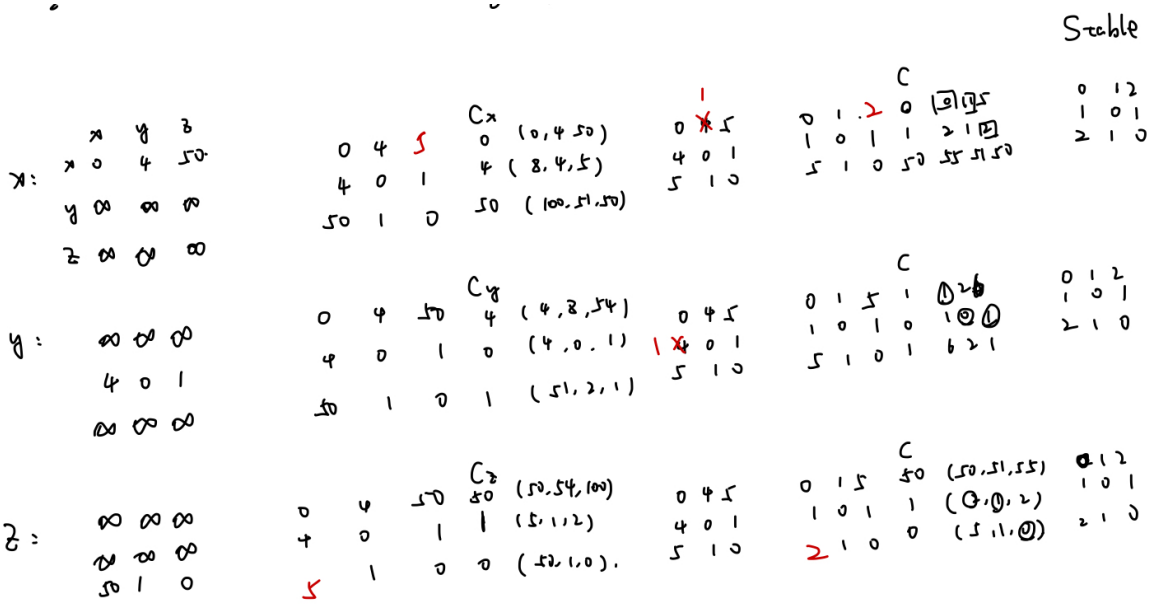


- Link-Cost Changes and Link Failure
  - Link cost changes:
    - Node detects local link cost change
    - Updates the distance table
    - If cost changes in least cost path, notify neighbors
    - compute when receive new vector, send when least cost path changes, do until the quiescent state



a. **b.**

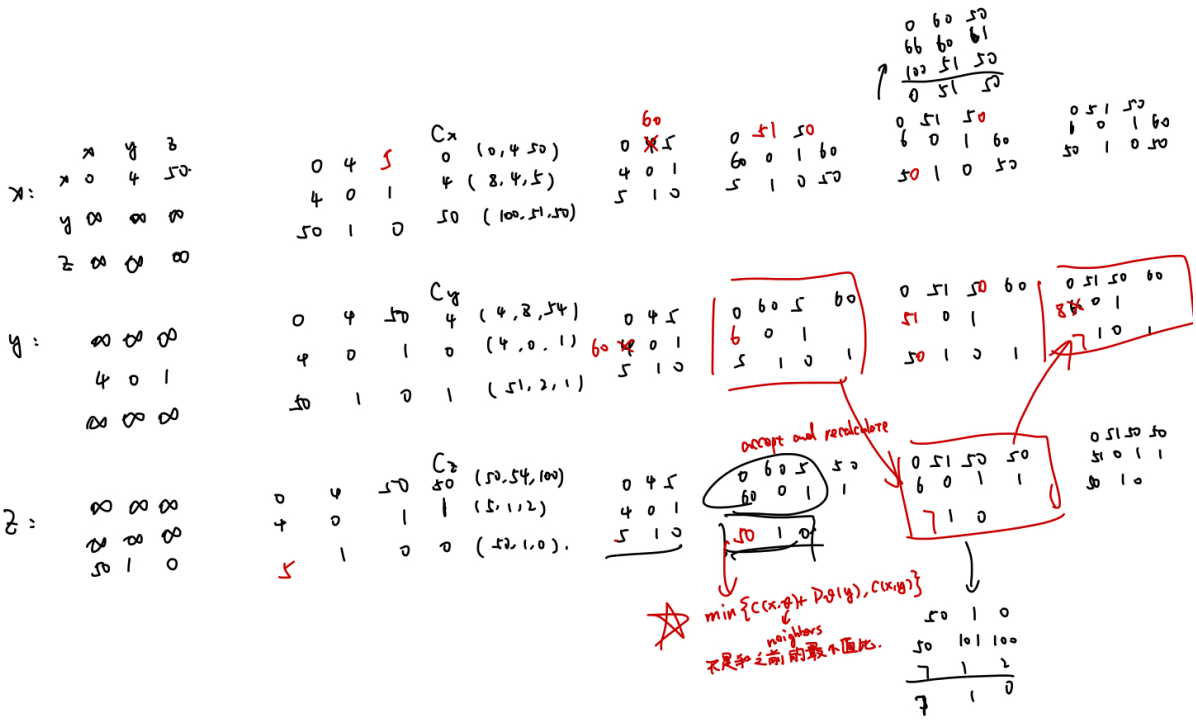
  - For a decreasing link cost (i.e. graph a)
    - At time t_0, y detects link-cost change and updates its distance vector, and informs its neighbors of this change
    - At time t_1 , z receives the update from y and updates its table. It computes a new least cost to x and sends its new distance vector to its neighbors
    - At time t_2 , y receives z's update and updates its distance vector, y's table no longer changed => quiescent state => only 2 iterations are required for the DV algorithm
    - good news about the decreased cost travels fast



- For a increasing link cost (i.e. graph b)
  - Before the link cost changes, Dy(x) = 4, Dy(z) = 1, Dz(y) = 1, and Dz(x) = 5. At time t0, y detects the link-cost change (the cost has changed from 4 to 60). y computes its new minimum-cost path to x to have a cost of

    $$D_y(x) = min\{c(y,x) + D_x(x), c(y,z) + D_z(x)\} = min\{60+0, 1+5\} = 6$$

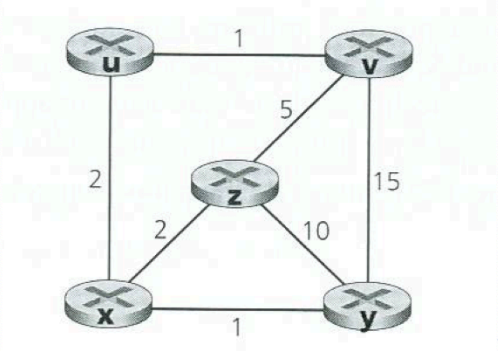
    y only know the direct cost to x is 60 and z has told y that z could get to x with a cost of 5. => In order to get to x, y route through z, which causes a routing loop -- z also route through y, every iteration the value of $D_y(x)$ increased by 1
  - infinite loop



  - Bad news about the increase in link cost has indeed traveled slowly

- Adding poisoned reverse
  - Basic idea : **If z routes through y to get to destination x, then z will advertise to y that its distance to x is infinity => avoid the infinite loop**
- more hops case:

1. Consider the network shown below, and assume that each node initially knows the costs to each of its neighbors. Consider the distance vector algorithm and show the distance table entry **at node z** at each step of the algorithm.



Initialization: Cost to

	u	v	x	y	z
v	∞	∞	∞	∞	∞
x	∞	∞	∞	∞	∞
y	∞	∞	∞	∞	∞
z	∞	5	2	10	0

Only one hop shortest path

	u	v	x	y	z
v	1	0	∞	15	5
x	2	∞	0	1	2
y	∞	15	1	0	10
z	4	5	2	3	0

2 hops shortest path for z, 1 hops shortest path for others

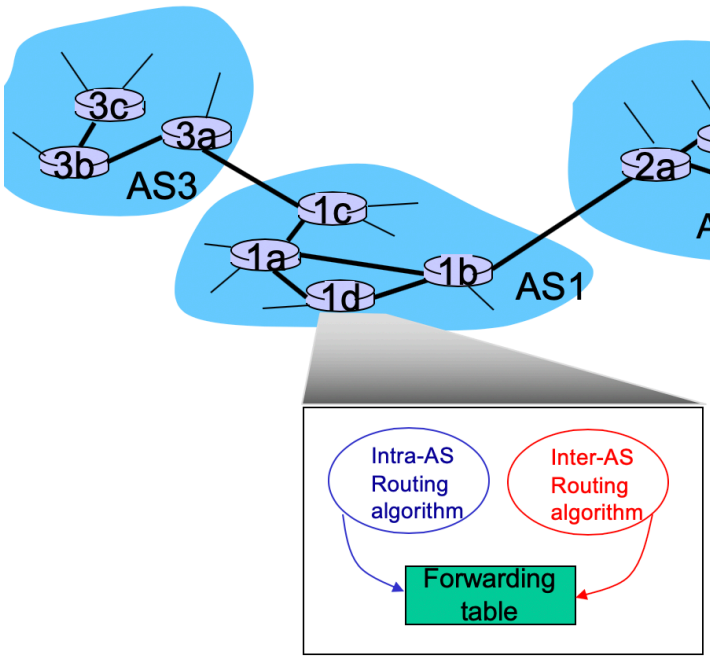
	u	v	x	y	z
v	1	0	3	15	5
x	2	3	0	1	2
y	3	15	1	0	3
z	4	5	2	3	0

	u	v	x	y	z
v	1	0	3	4	5
x	2	3	0	1	2
y	3	4	1	0	3
z	4	5	2	3	0

- A comparison of LS and DV Routing Algorithm
  - **Message** complexity:
    - LS requires each node to know the cost of each link in the network => O(|N| |E|) messages to be sent
      - 每个node E 个message， 一共N个router
    - DV algorithm depends on many factors (change of link-cost,convergence time etc.)
  - Speed of convergence(收敛)
    - LS converge when the $O(N^2)$ program finished => may have oscillations
    - DV converge only when there is no change of link cost(also count-to-infinity problem) => routing loops and count-to-infinity problem
  - Robustness(稳健性，强度)
    - LS : Router may broadcast an incorrect cost for one of its attached links, a router only computes its own forwarding table => provide degree of robustness
    - DV: A node can advertise incorrect least-cost path to all destinations => less robustness
  - Both are used in the Internet

4.5.3 Hierarchical Routing

- In practice, treat all routers as set of routers executing the same routing algorithm is a bit simplistic for at least 2 reasons :
  - *Scale* : As the **number** of routers becomes **large**, the overhead involved in computing, storing, and communicating routing information becomes prohibitive(高昂)
  - *Administrative autonomy* : The need of ability to let company deside which algorithm to use and don't let outsiders know the structure of inner network
- Solution : Organizing routers into **autonomous systems(ASs)** , with each AS consisting of a group of routers that are typically under the same administrative control (Routers in a same AS will run the same routing algorithm and have information about each other)
  - 1 ISP may consist 1 or more ASes

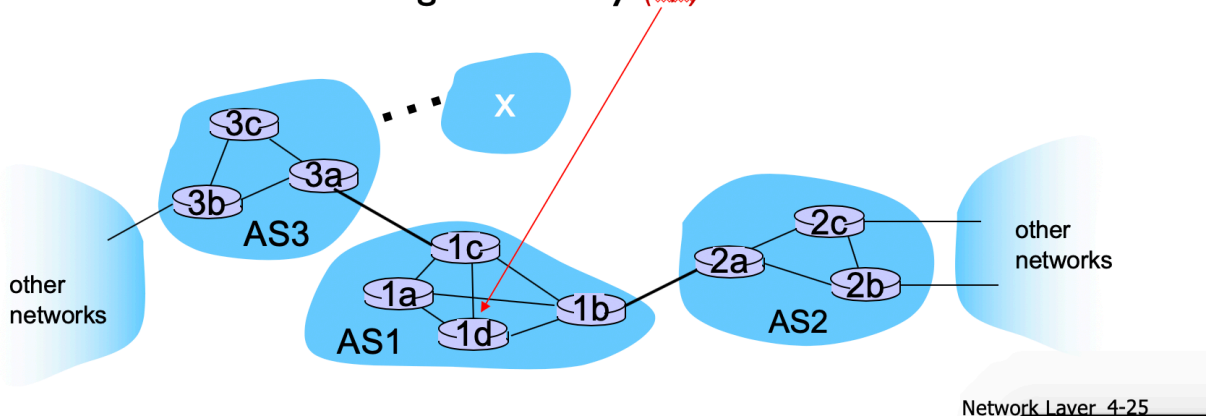


- forwarding table configured by both intra- and inter-AS routing algorithm
  - intra-AS sets entries for internal dests
  - inter-AS & intra-AS sets entries for external dests

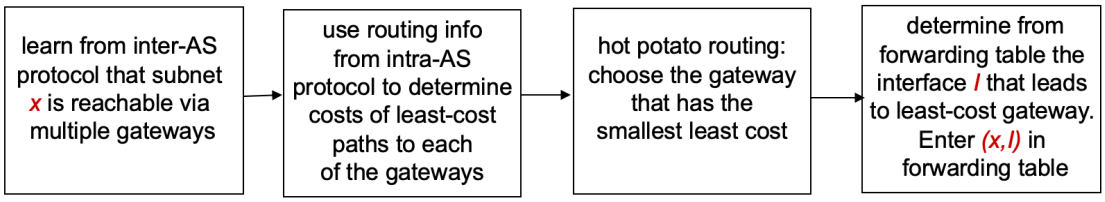
  - **intra-autonomous system routing protocol** : Routing algorithm running within an autonomous system
  - **gateway routers** : Connect ASs to each other, have the added task of being responsible for forwarding packets to destinations *outside* the AS (3a,1c,1b,2a)
  - Source router and destination router are not in the same AS
    - if there is only 1 gateway router
      - First find the best path to the gateway router
      - Second find the best path from the gateway router to the destination AS's gateway router
      - Third the destination AS's gateway router find the best way to the destination router
    - if there are 2 or more links of the AS (**inter-AS routing protocol**)
      - AS1 need to learn
        - Which destinations are reachable via AS2 and which dests are reachable via AS3, if both can reach, use hot-potato routing
        - propagate this reachability information to **all the routers within AS1** => each router can configure its forwarding table to handle external-AS dests
      - 2 communicating ASes must run the **same inter-AS routing protocol** (In internet all ASs run the same one called **BGP4**)
      - Each routers receive a intra-AS routing protocol and a inter-AS protocol
    - e.g.

Example: setting forwarding table in router 1d

- suppose AS1 learns (via inter-AS protocol) that subnet *x* reachable via AS3 (gateway 1c), but not via AS2
  - inter-AS protocol propagates reachability info to all internal routers
- router 1d determines from intra-AS routing info that its interface *I* is on the least cost path to 1c
  - installs forwarding table entry *(x,I)*



- If AS3 and AS2 can both access to AS1 => AS1 also propagate the information to all the routers
  - In order to configure the table, the router itself runs **hot-potato routing** to determine which path to go (done by having a router send the packet to the gateway router that has the smallest router-to-gateway cost => need to run intra routing)
  - When AS learns about a destination from a neighboring AS, the AS can advertise this routing information to some of its other neighboring ASs
  - total order:
    - Inter-AS protocol: check the accessability
    - Broadcast to all members within AS
    - Intra-AS protocol: hot-potato determine which gateway to use
    - Intra-AS protocol: since the best path has already determined, just store in the forwarding table

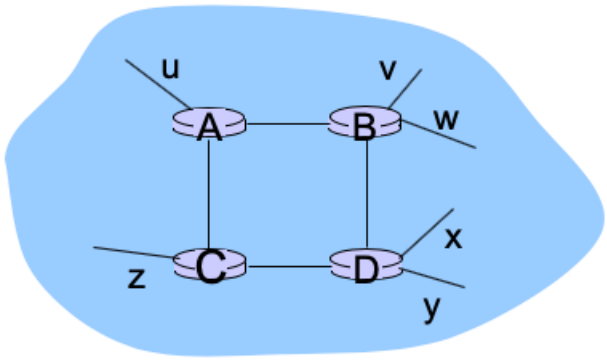


4.6 Routing in the Internet

4.6.1 Intra Routing algorithm (*interior gateway protocols*)

- RIP: **Routing Information** Protocol (based on DV)
  - distance metric: # hops (max = 15 hops), each link has cost 1 (just see number of links)
  - Hop: number of routers involved in the subnet along the source to the destination(involved means at least one interface is in the subnet)
  - DVs exchanged with neighbors every *30 sec* in response message (aka advertisement)
    - let neighbors know you are still alive

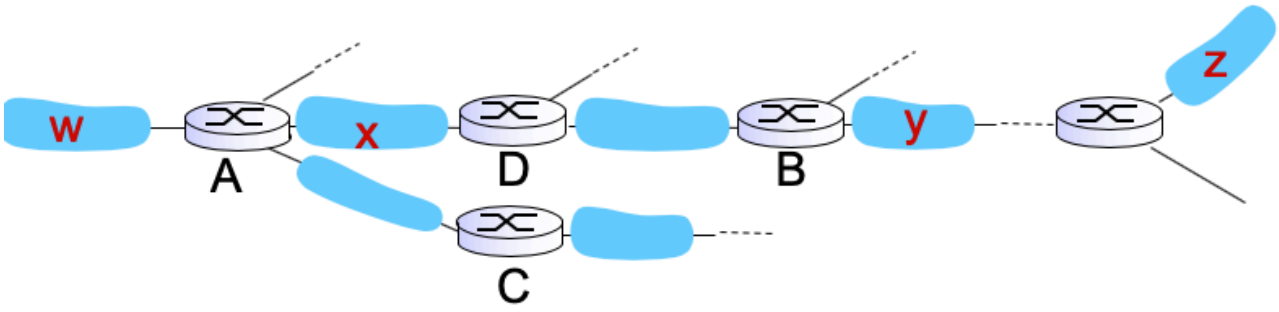
- each advertisement: list of up to 25 destination *subnets* (in IP addressing sense)



from router A to destination *subnets:*

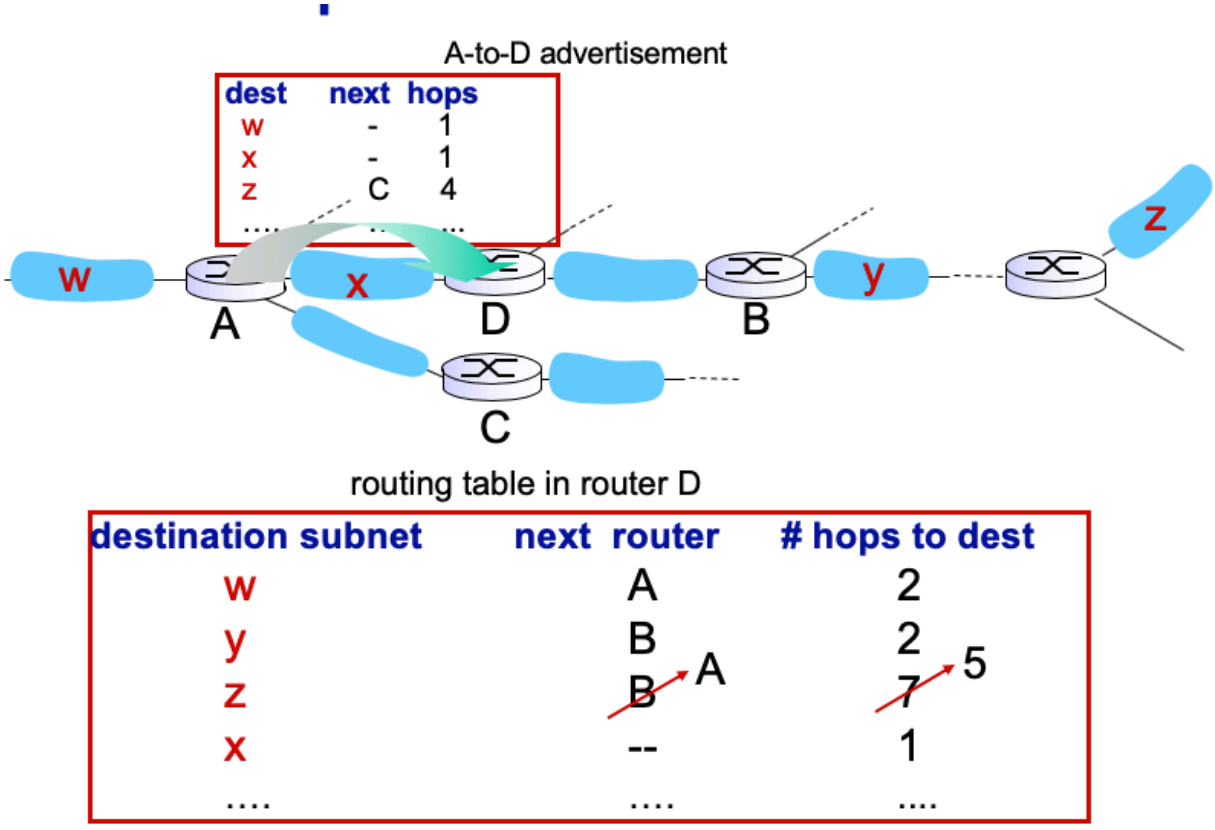
subnet	hops
u	1
v	2
w	2
x	3
y	3
z	2

- difference of the forwarding table using this method : need to maintain cost (hops), and the cost may be changed by the change of the network(some links broken)
- For example, for **router D**



routing table in router D

destination subnet	next router	# hops to dest
w	A	2
y	B	2
z	B	7
x	--	1
....	....	....



  - **link failure, recovery**
    - if no advertisement heard after 180 sec => neighbor/link declared dead => make some changes => send new advertisement to your neighbors => link failure into all the network
  - table processing
    - RIP routing tables managed by application-level process called route-d (daemon => means always run)
      - messages are generated by network layer and sent to the application layer
      - messages are used to tell others that you are alive
    - advertisements sent in UDP packets, periodically repeated
- OSPF: Open Shortest Path First (based on LS)
  - “open”: publicly available
  - also set cost of each link to 1 to achieve minimum hop routing, and 30 secs advertisement
  - If route changes, broadcast the information to all routers in the internet instead of only neighbors
  - Link-sate algorithm
    - LS packet dissemination(传播)
    - topology map at each node
    - route computation using Dijkstra’s algorithm
  - OSPF advertisement carries one entry per neighbor
  - intra-AS Routing in practice:
    - use hop routing instead of cost
    - use periodically advertisement

4.6.2 Inter-AS routing

- BGP(Border gateway protocol) : two BGP routers (gateway router“peers”) exchange BGP messages
  - Task of BGP
    1. Obtain subnet reachability information from neighboring ASs.
    2. Propagate the reachability information to all routers internal to the AS.
    3. Determine “good” routes to subnets based on the reachability information and on AS policy.
  - Thus, gateway router need to comunicate with:

        - internal routers to broadcast accessibility => **external BGP session**
        - external routers to get the accessibility => **internal BGP session**
    - advertising **paths** to different destination network prefixes (“path vector” protocol)
    - path vector determines the accessabilty
    - every prefix(represents an AS) there is a path vector stored in the gateway router
    - when AS3 advertises a prefix to AS1:
        - AS3 promises it will forward datagrams towards that prefix
        - AS2 can aggregate prefixes in its advertisement
            - There are 4 subnet attached to AS2 (3b-3a,3b-3c,3a-3c,3a-2a)
            - As2 can add the together and advertise

Difference of Intra and Inter

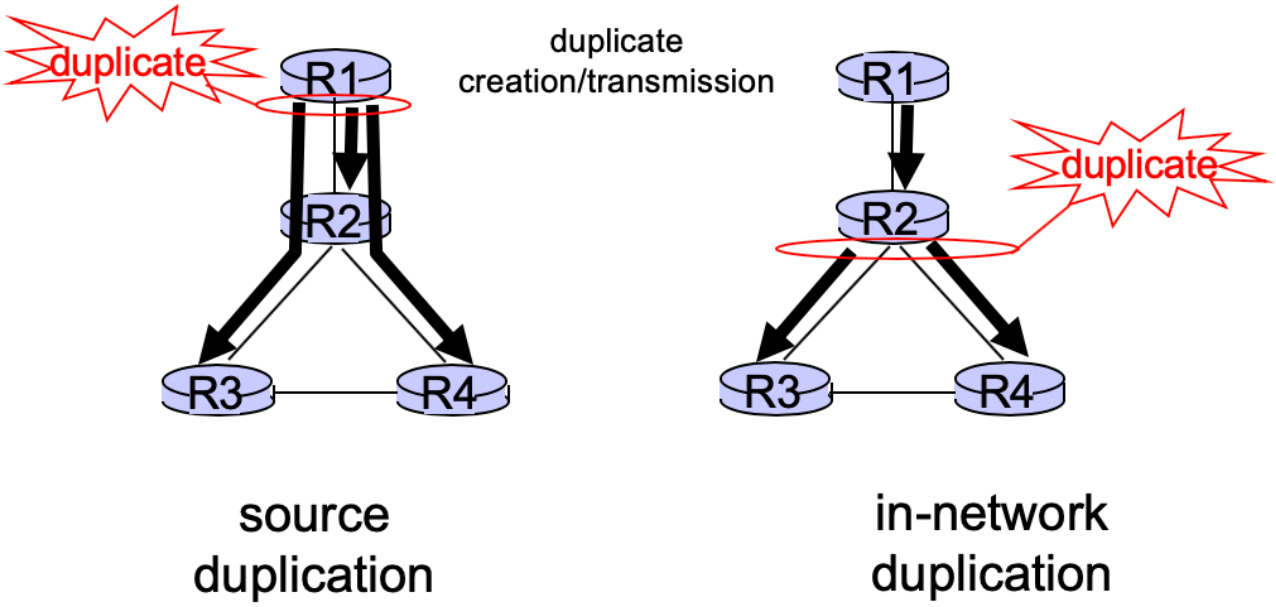
- Policy
    - inter-AS: admin wants control over how its traffic routed, who routes through its net.
    - intra-AS: single admin, so **no policy** decisions needed
- Scale : hierarchical routing saves table size, reduced update traffic
- Performance
    - intra-AS: can **focus on performance**
    - inter-AS: **policy may dominate** over performance

4.7 Broadcast and Multicast Routing

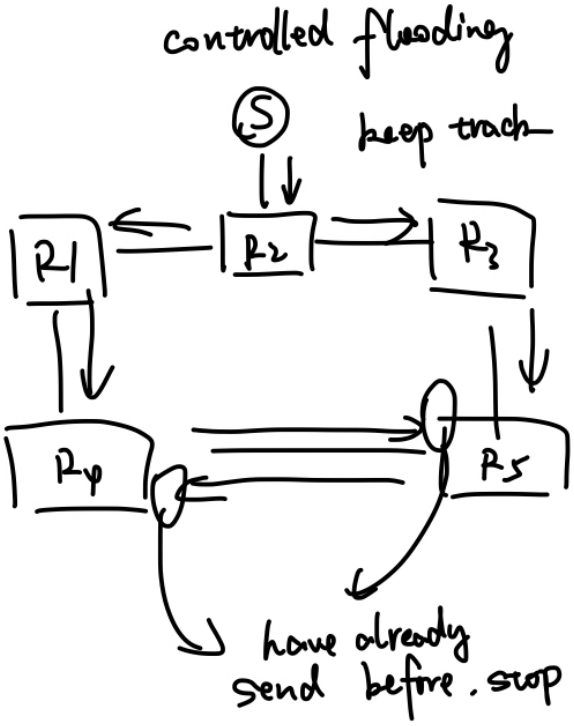
- Unicast
    - One-to-one
    - Destination – unique receiver host address
- Broadcast (hold by special IP address)
    - One-to-all
    - Destination – address of network
- Multicast (hold by special IP address)
    - One-to-many
    - Multicast group must be identified
    - Destination – address of group

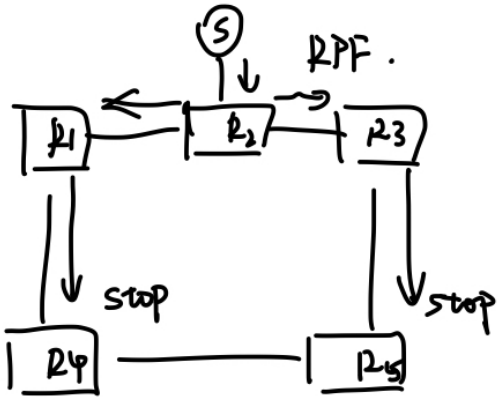
4.7.1 Broadcast routing algorithm

- deliver packets from source to all other nodes
- **source duplication** is inefficient



- Do In-network duplication
  - Flooding : when node receives broadcast packet, sends copy to all neighbors (recongonize mask)
    - problem : cycles & broadcast storm (neighbors are in a cycle)
  - controlled flooding: **node only broadcasts pkt if it hasn't broadcast same packet before**





    - node keeps track of packet ids already broadacsted (buffer the sender packet => to busy for router)
    - or **reverse path forwarding** (RPF):
      - only forward packet if it arrived on shortest path between node and source or For target router : only receive packets come from the shortest path from source router (comes from another way=> don't forward)
        - It is impossible to have the case that 2 nodes are on shortest way of each other => recall the Dijkastra algorithm
      - Thus a tree can be constructed
  - Spanning tree (go through every nodes only once => make the graph has no cycle)
    - first construct a spanning tree

        - Nodes then forward copies only along the spanning tree => maintains only once
        - Creation
            - determine a center node
            - each node sends unicast(one-to-one) **join** message to center node
            - message forwarded until it arrives at a node already belonging to spanning tree(i.e. each node want to join, just **find a path to one of spanning tree's leaf**)

4.7.2 Muticast Routing Algorithms

- IGMP(Internet Group Management Protocol) allows a host to communicate with l**ocal multicast router**
    - Using IGMP, a host can dynamically
        - join a multicast group
        - leave a multicast group
    - IGMP is part of IP
- IGMP messages:
    - Joining a group
        - membership report
    - Leaving a group
        - leave report
    - Maintenance
        - query (general : all multicast group used for checking, special : specific multicast group used for leaving)
        - membership report
- Joining
    - An application process sends group id to IGMP module(单元) within host
    - Host sends a $membership\ report\ message$ if this is the first entry for this group under the router
    - Local routers adds this group if it(the router) was not in database of active groups (only send the first request)(the source also has a database of router who has the interested host), and **propagates** appropriate messages(also a member ship message) to other routers
- Leaving
    - If no process is interested in a specific group, host sends a leave report
    - Router cannot immediately remove group (there may be other interested hosts)
    - Router send a **special query**(for perticular group) to ask whether anyone is interested in the muticast group
    - If no one interesting (after wait for sometime) => send leave to other routers
- Membership Monitoring
    - Imagine only one host for a group, and the host died **without sending leaving** message
    - Router periodically sends **general query message**
    - Router just expects the response message from each group (1000 hosts interest in 100 groups in total under a router => expect 100 messages according to each groups)
    - Hosts **delay response to reduce traffic** (max response time for query 100msec)
        - Hosts wait a **random time** between zero and maximum response time
            - hosts maintains a timer for each group to waiter for response of general query message
            - If it sees another in the same group response while waiting (responses are broadcast), it does NOT send response (cancels its timer)
- IP **multicast** routing
    - Purpose : share a Group information among routers, to implement a better routing for data distribution (send to interest hosts, cut don't interesting hosts)
    - Distribution tree structure

      G : number of groups for a router, S: number of source for each groups
        - source tree : root of tree is source
            - A tree consists of shortest paths from the source to each receiver
            - paths from different source to the same target is different => many entries in the forwarding table

        - Property
            - More memory $O(G * S)$ in routers
            - But optimal path from source to receiver, minimizes delay
            - Good for : **small number of senders**, many receivers such as **Radio broadcasting application**
    - shared tree : specify some node to be root
        - source unicast to the shared tree's root
        - Shared root then sends traffic to the multicast tree rooted at the shared root.
        - just maintain 1 entry for each target and how many group
        - property
            - less memory $O(G)$ in routers, one tree for each group
            - Sub-optimal path from source to receiver, may introduce **extra delay** (source to root)
            - May have duplicate data transfer (possible duplication of a path from source to root and a path from root to receivers)
            - Good for:
                - Many senders with low bandwidth
                - environment such as most part of the shared tree is identical to the source tree

---

Tutorial

---

1. Host A and B are directly connected with a 200Mbps link. There is one TCP connection between the two hosts, and host A is sending to Host B an enormous file over this connection. Host A can send application data into the link at 100Mbps but host B can read out of its TCP receive buffer at a maximum rate of 50Mbps. Describe the effect of TCP flow control.
    - When the receiver's buffer is full => B->(rwnd = 0)->A (A stops)
    - When rwnd>0, B->(rwnd>0)->A (A resumes data transmission)
2. At time t, a TCP connection has a congestion window of 4000 bytes. The maximum segment size used by the connection is 1000 bytes. What is the congestion window after it sends out 4 packets and receives acks for all of them? Suppose there is one ack per packet.

    a) If the connection is in slow start. : 8000 bytes

    b) If the connection is in linear increase mode. : 5000 bytes
3. Suppose that you are designing a reliable data transfer protocol. You have two choices. One is to use negative acknowledgement, i.e., the receiver sends NACK to let the sender know when the receiver detects a loss of packet. The other is to use positive acknowledgement such as what we have learned in class for Go-back-N, Selective Repeat, and TCP, to acknowledge what has been received.

    a) Suppose the sender sends data only *infrequently*. Would a NAK-only protocol be better or an ACK-based protocol be better? Why?
    - Only when the next one sent, the sender will receive NAK => sender wait a long time for a NAK (no timeout)

    b) Now suppose the sender has a lot of data to send and the end-to-end connection experiences few losses. In this second case, would a NAK-only protocol be preferable to a protocol that uses ACKs? Why?
    - Positive : Many ACK
    - Negative : very few NAK (in theory preferred)
4. Consider a network which has a lot of bursty losses (several packets lost in a burst/**batch**) on the links. Which of the two protocols functions better in such scenario, go-back-N or TCP? Why?
    - choose GBN, because TCP can only retransmit one packet

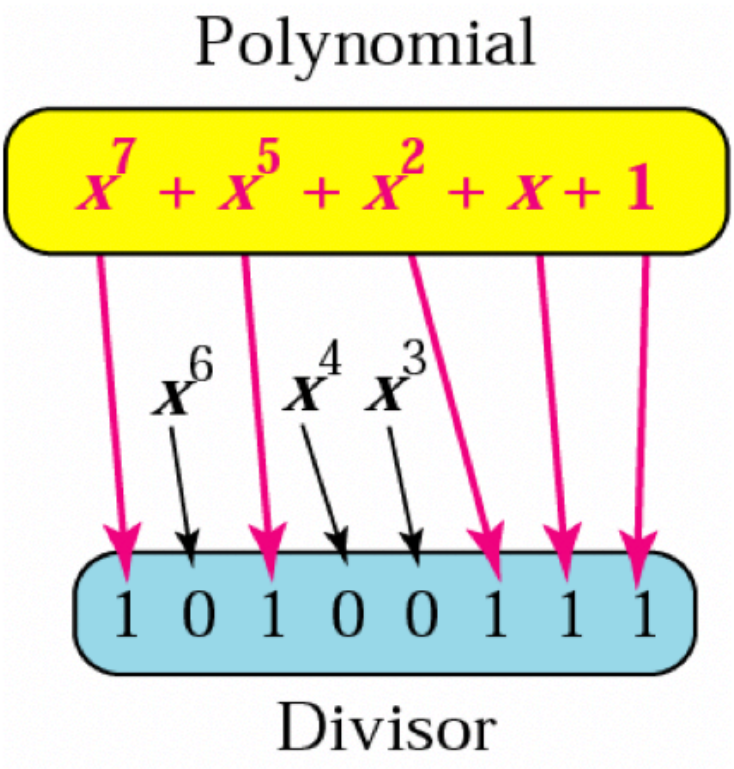
Chapter 5 Link Layer

- Links: communication channels that connect adjacent nodes along communication path

    - Wired link : Ethernet
    - wireless link : WIFI
    - Packet name : frame,application layer(message), transport layer(segment),network layer(datagram)
- Duties of data link layer
    - paketizing (divide datagram)
    - flow control (limit of router's buffer)
    - **Error detection, correction**
    - **Access control : whose turn to transmit(avoid collision)**
    - **Addressing**
- Position implemented
    - In each and every network device(switch only runs 2 protocol)
    - In each device, run in the NIC(network interface card)
        - application layer : run as applications
        - transport and network layer : defined in OS (operating system)
- Error detection
    - EDC : Error Detection and Correction bits (redundancy) => only detect error, if occurs, throw away the frame, retransmission is taken care by receiver(transport layer protocol)
    - methods : parity check, Cyclic redundancy check, checksum
    - parity check
        - Even parity : the total number of 1's in the data plus parity bit must be even (if odd, put 1 in EDC,else put 0)
        - Odd parity : the total number of 1's in the data plus parity bit must be odd (if even, put 1 in EDC,else put 0)
        - one bit error can be checked, but 2 bits error cannot be detected
    - 2-dimensional bit parity
        - do column checking except row checking for matrix of all datas
        - if 1 bit error occurs, you can correct it by locating it by row and column
        - 2 bits error can be absolutely detected, but it cannot be corrected (because you just know the 1 dimentional information)
        - cannot check all 4-bits error (the 2*2 square error cannot be detected)
    - Cyclic redundanct check (any number of error=>depends on redundant)
        - D is data (as binary form)
        - choose $r+1$ pattern (r is the max bits error you can check) G
        - Goal : choose r CRC bits R, s.t.
            - (D << r $\bigoplus$ R) % G = 0 (redefine the plus operation)
            - **In the redefined calculation, addition and substraction can be treated as a same operation (because there is no carrying bit)**
            - receiver knows G, divides D << r $\bigoplus$ R by G, if non-zero remainder: error detected
            - can detect all burst error less than $r+1$ bits (**why r+1 maintains a $at\ most$ r bits remainder**?)
                - r+1 bits => D << r has r '0' and the end, when do the division, there are only 2 cases
                    - last r+1 : 0000...00 => remainder less than r+1 digits
                    - Last r+1 : 1000...00 => remainder less than r+1 digiits
        - Proof

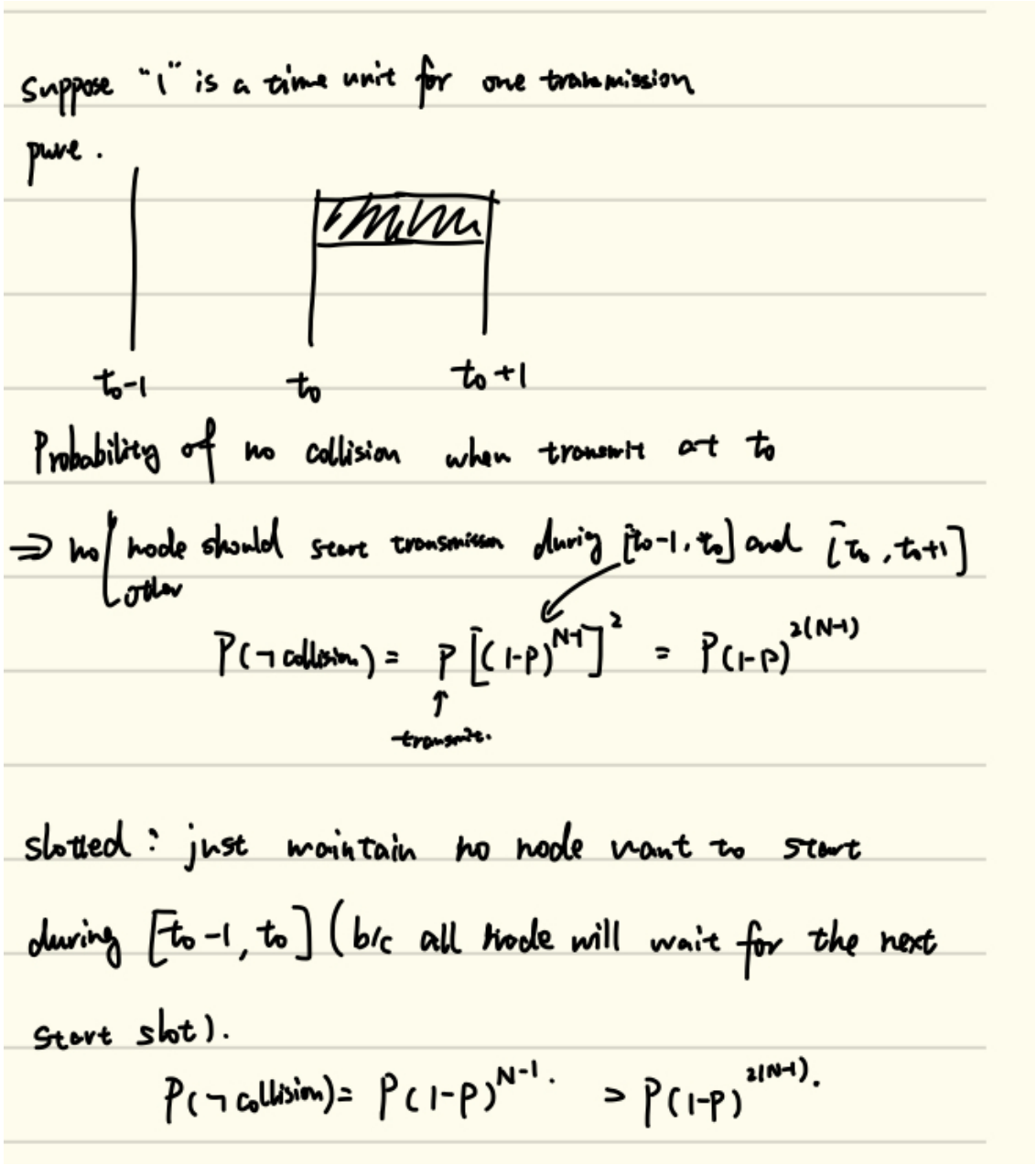
$$\begin{aligned}(D << r) \oplus R &= n * G \\ (D << r) \oplus R \oplus R &= n * G \oplus R \\ (D << r) &= n * G \oplus R\end{aligned} \tag{24}$$

            - here the meaning of xor is *plus without consideration of carrying or borrowing bits* , thus can be treated as plus, but symmetrically, in the division and multiplication operation, you should also use XOR as plus and minus
        - polynomial representing a divisor (x equals to 2)



- Multiple access links protocols(access control)
  - two or more simultaneous transmissions by nodes: interference (干涉)
    - collision if node receives two or more signals at the same time
  - Broadcast : shared wire or medium , only 1 source can tranfer at one time
    - **Multiple Access Control (MAC) Protocol** : distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
- MAC protocol :
  - channel partitioning
    - divide channel into smaller “pieces” (time slots, frequency, code)
    - allocate a piece to a node for exclusive use
    - TDMA:
      - access to chanel in "rounds"
      - each station gets fixed length slot in each round
    - FDMA:
      - channel spectrum divided into frequency bands
      - each station assigned fixed frequency band
    - TDMA and FDMA => share time or share frequency => unused time or frequency becomes idle => inefficient
  - random access
    - channel not divided, allow collisions (similar to try and catch)
    - when a node has a packet to send
      - transmit at full channel data rate R
      - no a priori coordination among nodes
    - When 2 or more transmitting nodes => collision => the protocol specifies how to detect collisions and recover from it
    - types
      - slotted ALOHA, ALOHA
      - CSMA, CSMA/CD, CSMA/CA
  - "taking turns"
    - nodes take turns, but **nodes with more to send can take longer turns**

        - But if someone get turn, it will hold it for a long time, others may wait for a long time (different with channel partitioning, it doesn't define when to give turn to others)
- Slotted ALOHA(detect:signal different, retransmission: retransmit in next beginning of the slot according to **probability p**)
    - assumptions
        - all frames have same size
        - time divided into equal size slots (time to transmit one frame)
        - nodes start to transmit only at the beginning of a slot (wait for next beginning)
        - **nodes are synchronized** (timer are same)
        - if 2 or more nodes transmit in a slot, all nodes detect collision (in hardware perspective, two or more transmitting has different signal with only one, thus can be detected)
    - Operation
        - when node obtains fresh frame, transmits in the beginning of the next slot
        - At the end of each slot,
            - if no collision: successful
            - if collision: node retransmits the frame in each subsequent slot with **prob. p** until it is successful
                - every time generate a random number, if more than p, then transmit (every one has a probability of p to retransmit => avoid next collision
                - p is the probability for **all the nodes** to retransmit
    - Pros :
        - single active node can continuously transmit at full rate of channel (than channel division)
        - highly decentralized: only slots in nodes need to be in sync
        - simple
    - Cons :
        - collision, wasting slots and idle slots (a slot of collision is completely wasted)
        - nodes may be able to detect collision earlier than the time to transmit packet (possible optimization by using **carrier sense** because it can detect collision before transfer)
        - difficult to implement clock synchronization
- Pure ALOHA (no time slots)
    - Whenever a user has a frame to send, it simply transmits the frame
    - If data doesn’t get through (receiver sends acknowledgement) then it retransmit with probability p, if it fail to retransmit(1-p), then retransmit after a random delay
    - collision probability increases:
        - frame sent at t_0 collides with other frames sent in $[t_0 - 1, t_0 + 1]$
    - not efficient as slotted ALOHA



- Carrier Sense Multiple Access (CSMA)： only listen at first
  - Procedure
    - Before start, check whether it is busy, listen to medium and wait until it is free
    - **if it is free, wait a random back off time then start talking**
  - Advantage :
    - fairly simple to implement
    - functional scheme that works (有效)
  - Disadvantages:
    - Did not detect collision => even if collision happens, the ongoing transmission still keeps going until the transmission finishes.
      - Inefficient waste of medium time
- CSMA/CD (collision detection) : keep on listen and special back off after collision
  - CD part
    - collisions can be detected within short time
      - each node listens to the channel while transmitting
    - colliding transmissions aborted, reducing channel wastage
  - difference with CSMA : not only listen at first, but also listen during transmitting

- Procedure
  - NIC sense chanel idle, starts frame transmission, otherwise wait until channel idle
  - If NIC transmit entire frame without detecting another transmission, NIC is done with frame
  - If NIC detects another transmission, aborts and **sends(broadcasts) jam signal**
    - inject some noise, because the original collision signal maybe weak
  - After aborting, NIC enters **binary(exponential) backoff** (waiting time) :
    - after m^{th} collision, NIC chooses K at random from $\{0, 1, 2, \dots, 2^m - 1\}$. NIC waits K **slots**, returns to Step 1
    - Slots can be defined by **Prof.** zlq as : a transmission time (i.e. $\frac{frame\ size}{bandwidth}$ the transmission time for one frame)
    - longer backoff interval with more collisions
    - At first the interval has only {0,1}, if collision still occurs, it will be increased to {0,1,2,3} etc. until no collision => exponential
- Minimum Ethernet Frame size (needed under the collision detection system)
  - To send a collision message : The longest time between start to transmit a frame and receiving the first bit of a jam sequence(collision message) is **twice the propagation delay** from one end to another
    - The first bit of frame reaches the other side 1 propagation delay
    - Other side send collition message(treated as 1 bit) : 1 propagation delay
  - Thus the length of each frame must be long enough to hold on for 2 propagation delay (else when the message goes back, the frame is already finished and dequeued)

$$2 * t_{prop} = \frac{frame\ size}{transmission\ speed(bandwidth)} \tag{25}$$

  - The size of the collide message is treated as 1 bits (the first bit arrive then the sender will know something, so just need to consider 1 bit)

Example #1: Cable = 400m, transmission speed (bandwidth) = 10 Mbit/sec, propagation speed = $2*10^8$ m/sec

Propagation delay time:

$$t_{prop} = \frac{d}{V} = \frac{400}{2 \times 10^8} = 2 \times 10^{-6} = 2\mu\text{sec}$$

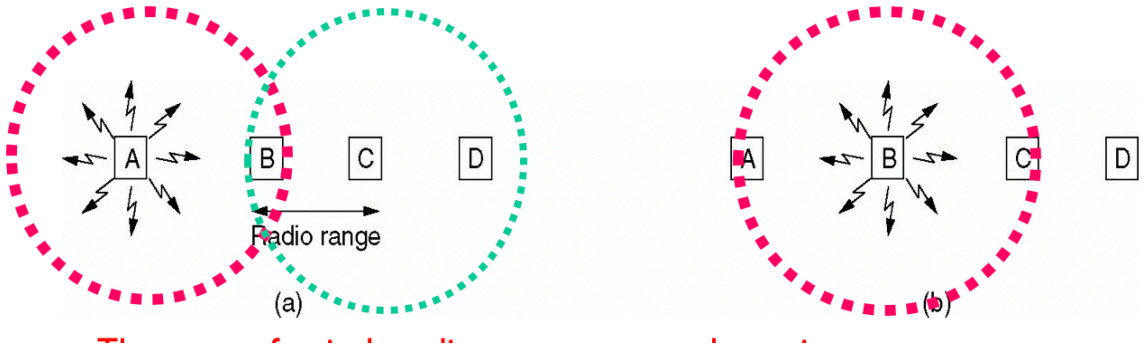
The round-trip propagation delay is, of course, twice this. Thus the round trip delay is $2 \times t_{prop} = 4\mu\text{sec}$

With a data rate of $R = 10Mbps$ *each bit has*

$$t_b = \frac{1}{R} = \frac{1}{10{,}000{,}000} = 0.1\mu\text{sec}\, duration$$

- Wireless network
  - Elements:
    - wireless hosts :
      - Laptop, PDA, IP phone
      - run applications
      - may be stationary(non-mobile) or mobile
    - Base station:
      - typically connected to wired network
      - Relay : responsible for sending packets between **wired** network and **wireless** hosts in its "area"
    - wireless link :

        - typically used to connect mobile(s) to base station
        - multiple access protocol coordinates link access
        - various data rates, transmission distance
- problem
    - Two transmitting stations will interfere with each other at the receiver if they send data simultaneously
    - The receiver will “hear” the sum of the two signals (which usually means garbage)
        - Similar to talking in a crowded room
        - Also similar to hub based Ethernet
    - Protocol required to coordinate access
        - i.e. transmitters must take turns
    - *Hard to detect collision* in wireless networks
        - Transmitted signal is MUCH stronger than received signal due to high path loss in the wireless environment (up to 100dB)(cannot base on difference)
        - Also transmitter may not even have much of a signal to detect due to geometry (signal cannot come back)
    - Hidden and Exposed station problem
        - range of a single radio may not cover the entire system



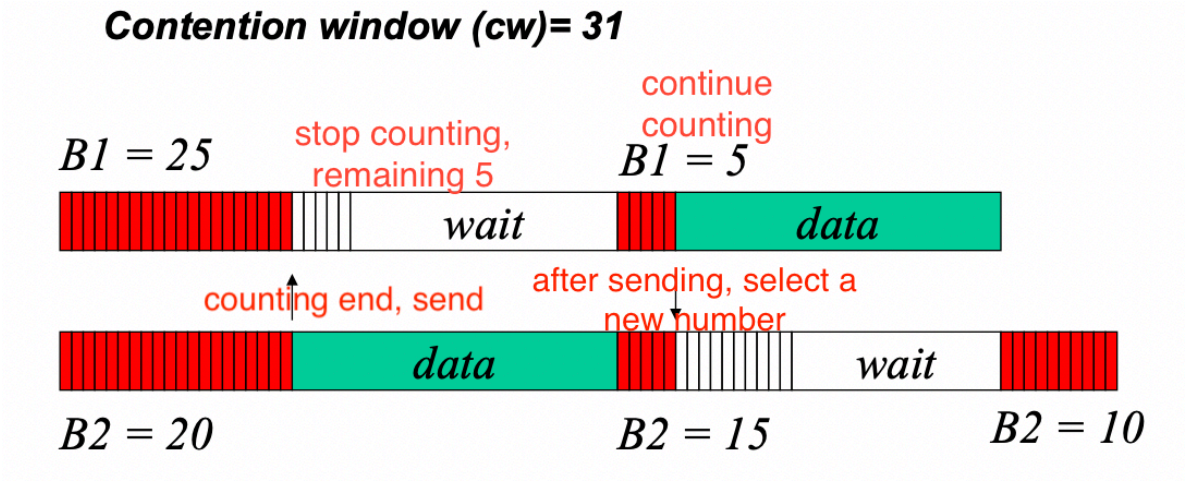
        - If we apply CSMA, collision will happen a lot => cannot avoid some cases
            - A->B, C->B since C does not hear A to judge collision
            - Hidden station problem (C hiden from A)
            - for wired system, C can be informed
        - If we apply CSMA directly, efficiency may not be high => may not need to make some routers silent
            - **B->A**, C hears B, C won’t send to D, reduced efficiency
            - Exposed station problem (C exposed to B)
            - for wired system, if C send message, it may affect A, but in this case, C is free to send
- Solution : CSMA/CA in Wireless Communication (CA is collision avoidance)
    - Idea : having a short frame transmitted from both sender and receiver before the actual transfer
    - Operation
        - **A** sending a short **RTS (Request to Send)** (30 bytes) to **B** with length of L
        - **B** respond with a **CTS(clear to send)** to A, whoever hears CTS except A shall remain silent(neither send nor receive) for the duration of L
        - **A** send data to **B**
        - **B** sends ACK to **A**
    - How it solve the two problems

      (1) C hears the CTS from B and will be silent => hidden solved

      (2) C will not hear the CTS from A and will send to D => expose solved
    - In the second case, if A->B, C cannot send or receive to D, why ?
        - Because the signal transfered to all the direction since it is wireless, C will affect B if it is not silent
    - Advantage
        - Small control frames lessen the cost of collisions
            - When data is large, the control message overhead to avoid collisions is affordable.
        - RTS + CTS provide “virtual” carrier sense which protects against hidden terminal collisions (where A can’t hear B)
    - Disadvantage :

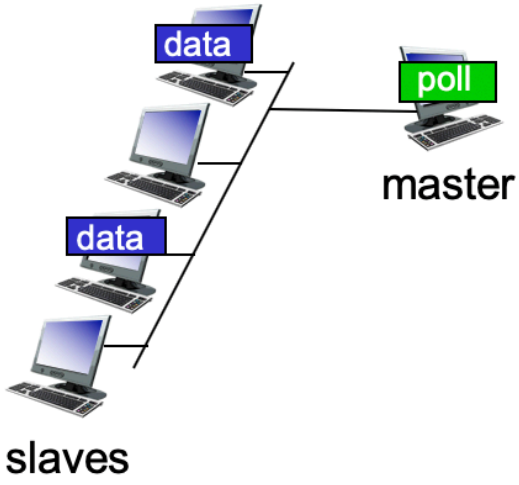
        - lack of efficiency compared to CSMA-CD, due to the silent period
        - RTS could have collision as well

          We should add some random time on top of CSMA/CA => Random Contention Access
    - Random Contention Access(also DCF => distributed coordination function)
        - Slotted contention period
            - Used by all carrier sense variants(not only for CSMA)
            - Provides random access to the channel
        - Operation
            - Each node selects a random back off number [0, contention window]
            - Waits that number of slots while monitoring the channel
            - If channel stays idle and reaches zero, then transmit
            - If channel becomes active, **stop counting** and wait until transmission is over, then start counting again



        - Choose DCF contention window
            - small value of CW
                - Less wasted idle slots time
                - large number of collisions with multiple senders
            - Large value of CW
                - too long waiting
            - optimal cw for known number of contenders & know packet size
                - depends on 2 variables : **transmission rate, and number of nodes**
                - Computed by minimizing expected time wastage (by both collisions and empty slots)
                - Tricky to implement because number of contenders is difficult to estimate and can be VERY dynamic =>still difficult to deal with the case that selected random numbers are the same => binary exponential backoff in DCF
            - Binary Exponential backoff in DCF
                - when node fails to receive CTS in response to its RTS, it increases the contention window(doubled)
                - **when a node successfully complete a data transfer, it restores cw to cw_{min} => sawtooth curve**
- Taking turns protocol
    - comparation
        - small load : channel partitioning MAC protocols is not efficient(wasted slots/partitions)
        - high load : random access protocols is not efficient(collision overhead)
        - taking turns : look for best of both worlds
    - term: latency : **Turn latency** is the difference between the start time of a **turn** and the end time of the previous **turn**.
    - types
        - polling (轮询)(dumb slave : slave node don't accept data)

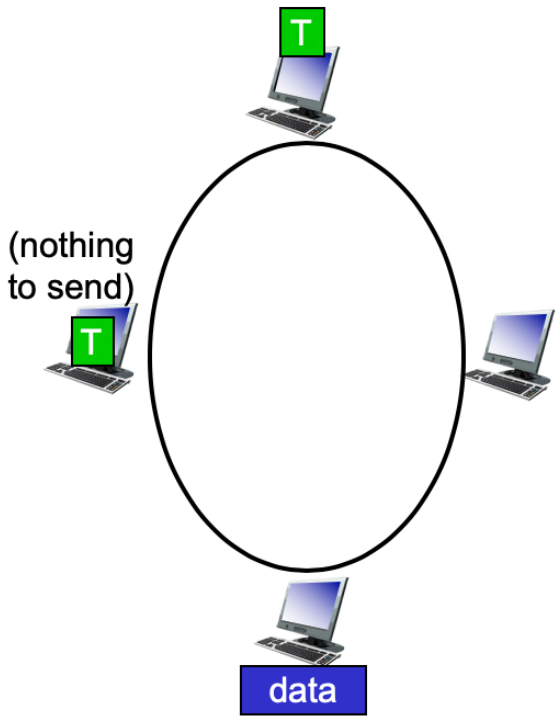
- master node "invites" slave nodes to transmit in turn
- typically used with "dumb" slave devices
- concerns:
  - polling overhead
  - latency
  - single point of failure (master)



- token passing (transmit permision)

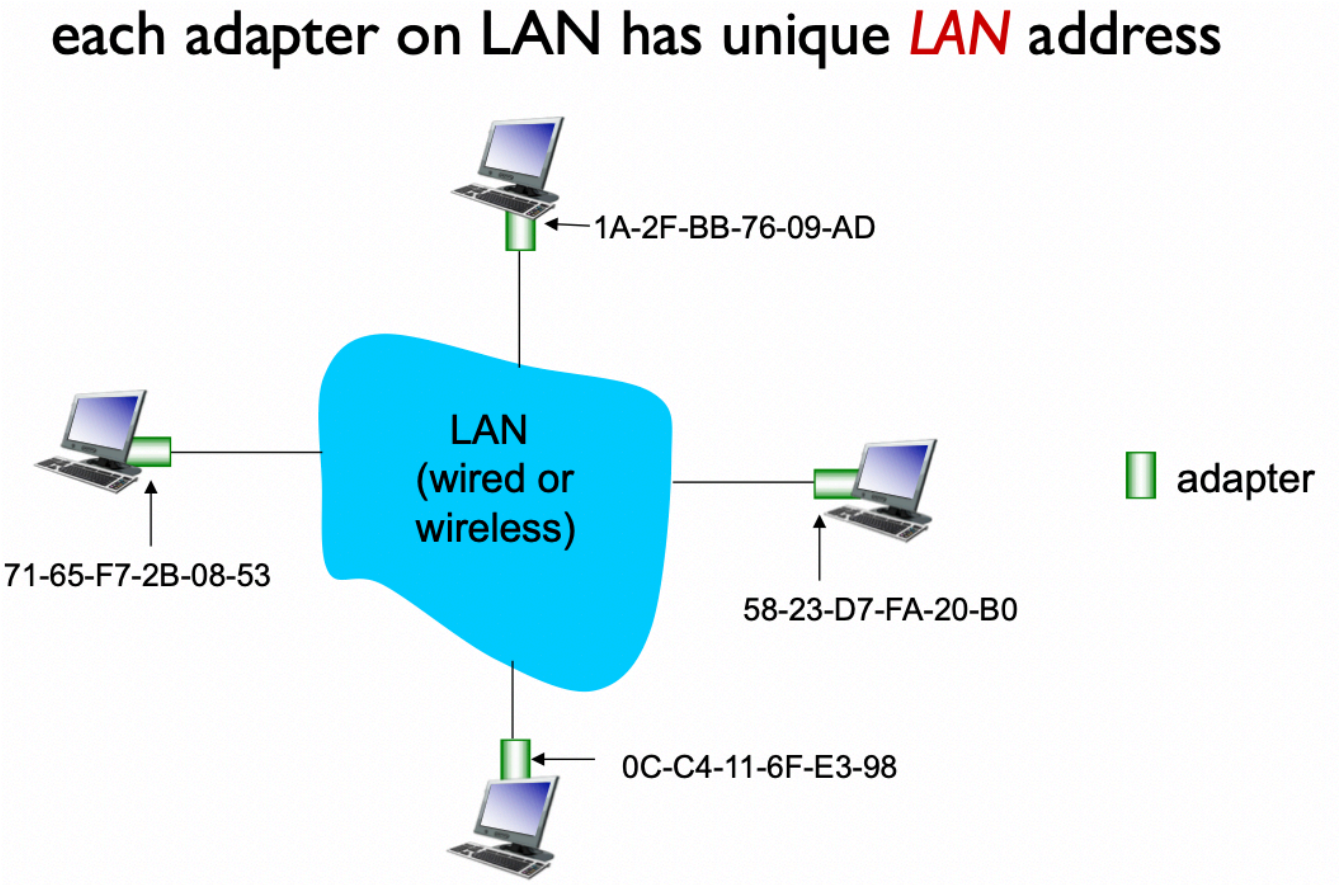
token passing:

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



- MAC address and ARP(address resolution protocol)
  - recall IP address : network-layer address for interface (**will not be changed in a datagram during transmitting**)
  - MAC (or LAN or physical or Ethernet)address (**will be changed during transmitting**):
    - function: used 'locally" to get frame from one interface to another **physically-connected** interface (same network, in IP-addressing sense)
    - 48 bit MAC address (for most LANs) burned in NIC ROM(**fix for every NIC**, different with IP address change with location),

      •e.g.: 1A-2F-BB-76-09-AD (hexadecimal notation) 48 bits



- MAC address allocation administered by IEEE
- manufacturer buys portion of MAC address space (to assure uniqueness)
- analogy:
  - MAC address: like Social Security Number
  - IP address: like postal address
- MAC flat address ➜ portability (可移植)
  - can move LAN card from one LAN to another
- IP hierarchical address *not* portable
  - address depends on IP subnet to which node is attached

- ARP: address resolution protocol
  - determine interface's MAC address knowing its IP address
  - ARP table : each node(host, router) on LAN has table
    - IP/MAC address mappings for some LAN nodes

      $< IP\ address; MAC\ address; TTL >$
    - TTL (time to live) : define time after which address mapping will be **forgotten**
  - operations

- A wants to send datagram to B
  - B's MAC address not in A's ARP table.
- A broadcasts ARP query packet **to all the neighbors**, containing B's IP address
  - dest MAC address = FF-FF-FF-FF-FF-FF (broadcast)
  - all nodes on LAN receive ARP query
- B receives ARP packet, replies to A with its (B's) MAC address
  - frame sent to A's MAC address (unicast)
- A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (times out)
  - soft state: information that times out (goes away) unless refreshed