Lec 2

TCP vs UDP

Internet transport protocols services

TCP service:

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

UDP service:

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

Q: why bother? Why is there a UDP?

UDP vs TCP

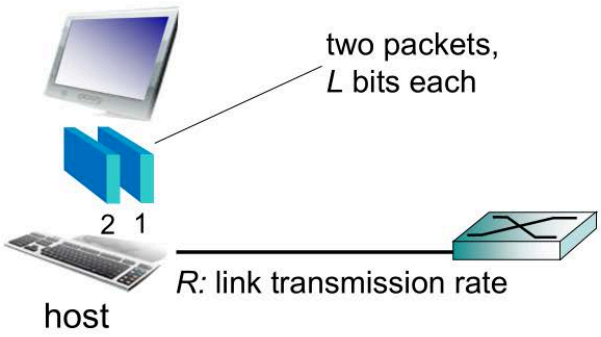
- TCP → Unicast
- UDP → Broadcast

Terms in Host side:

Host: sends *packets* of data

host sending function:

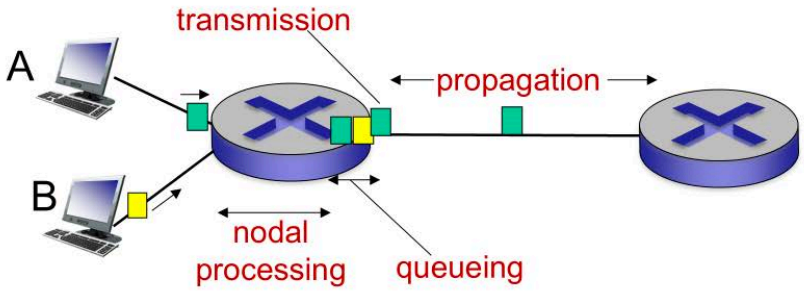
- takes application message
- breaks into smaller chunks, known as *packets*, of length *L* bits
- transmits packet into access network at *transmission rate R*
  - link transmission rate, aka link *capacity, aka link bandwidth*



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

Sources of Delay

Four sources of packet delay



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

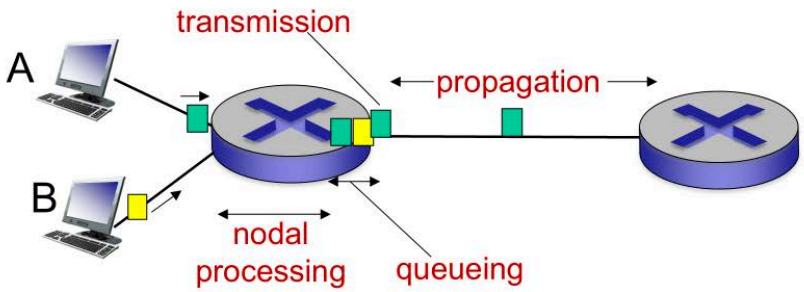
d_{proc}: nodal processing

- check bit errors
- determine output link
- typically < msec

d_{queue}: queueing delay

- time waiting at output link for transmission
- depends on congestion level of router

Four sources of packet delay



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

d_{trans}: transmission delay:
- L: packet length (bits)
- R: link *bandwidth (bps)*
- $d_{trans} = L/R$

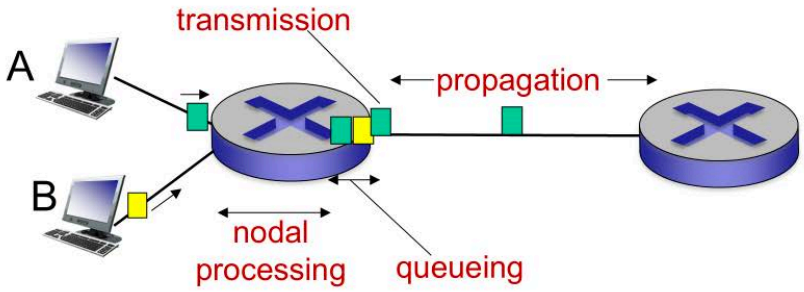
d_{trans} and d_{prop} *very* different

d_{prop}: propagation delay:
- d: length of physical link
- s: propagation speed (~2×10^8 m/sec)
- $d_{prop} = d/s$

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/
* Check out the Java applet for an interactive animation on trans vs. prop delay

Throughput

Four sources of packet delay



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

d_{trans}: transmission delay:
- L: packet length (bits)
- R: link *bandwidth (bps)*
- $d_{trans} = L/R$

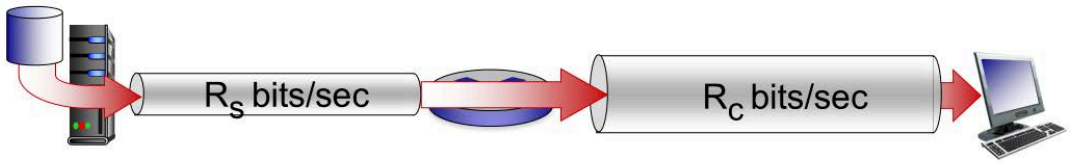
d_{trans} and d_{prop} *very* different

d_{prop}: propagation delay:
- d: length of physical link
- s: propagation speed (~2×10^8 m/sec)
- $d_{prop} = d/s$

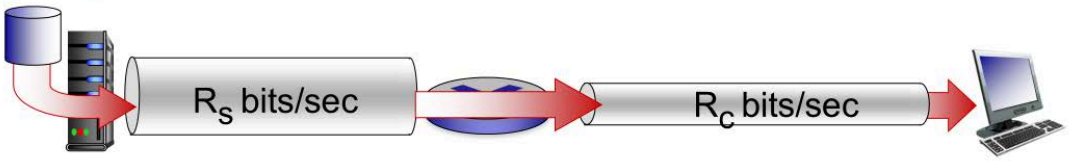
* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/
* Check out the Java applet for an interactive animation on trans vs. prop delay

Throughput (more)

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



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

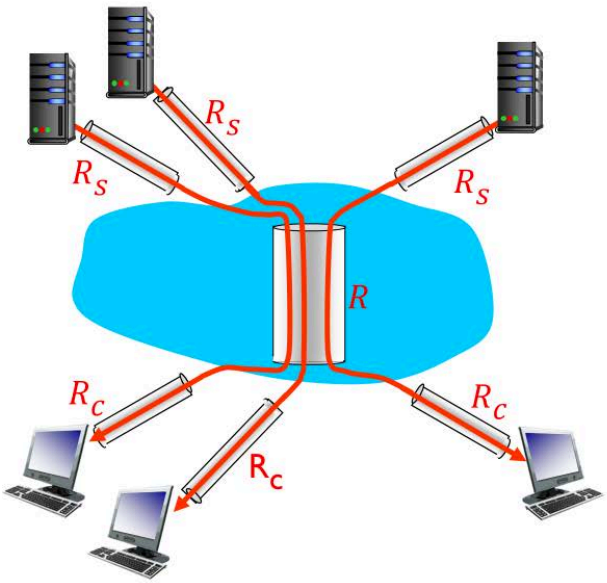


bottleneck link

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

Throughput: Internet scenario

- per-connection end-end throughput: $\min(R_c, R_s, R/10)$
- in practice: R_c or R_s is often bottleneck



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

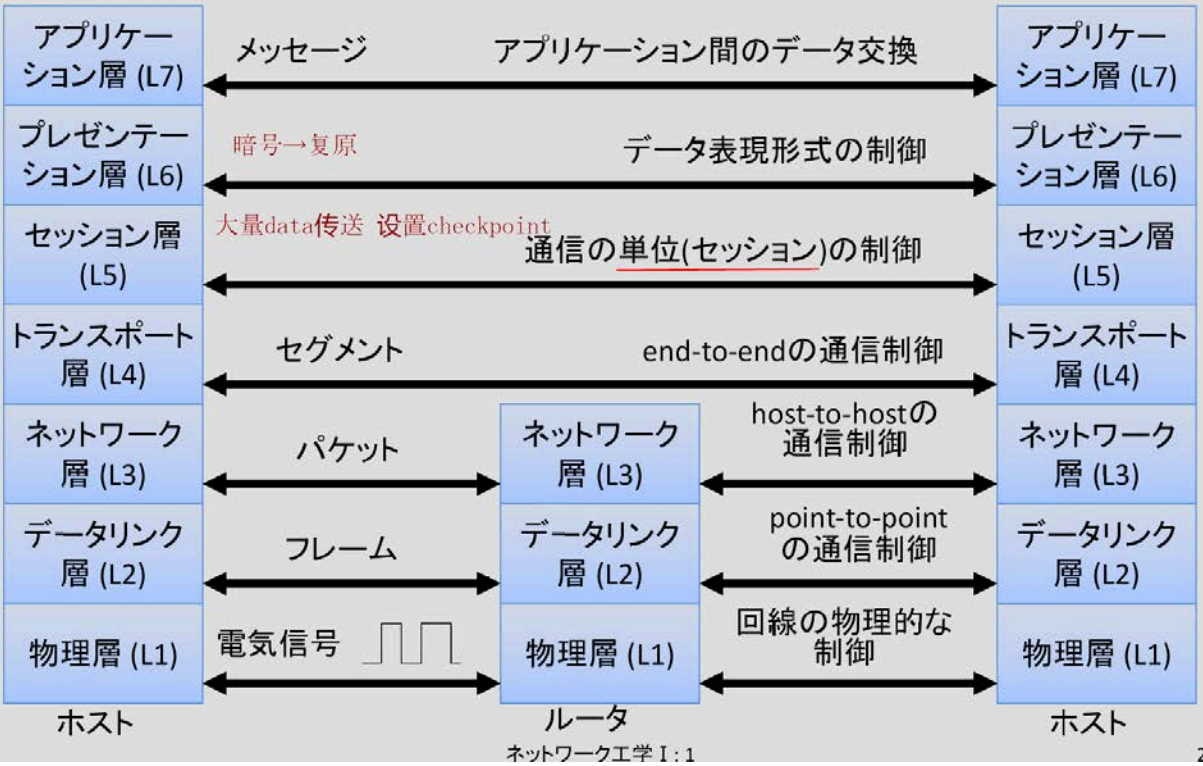
* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Key elements of a Protocol

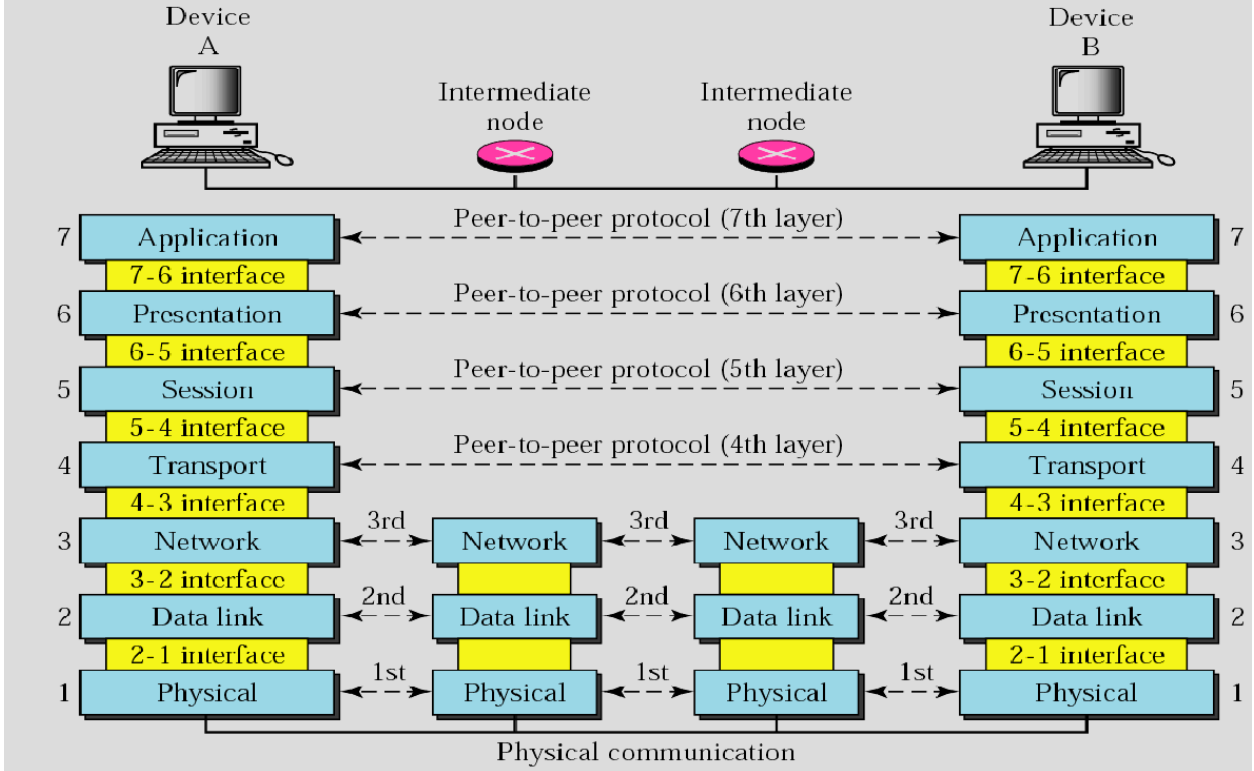
Key Elements of a Protocol

- The peer layers communicates by means of formatted blocks of data that obey a set of rules and conventions known as a **protocol**.
- Key elements:
  - **Syntax**
    - Format of the data blocks
    - Signal levels
  - **Semantics**
    - Control information for coordination and error handling
  - **Timing**
    - Speed matching
    - Sequencing

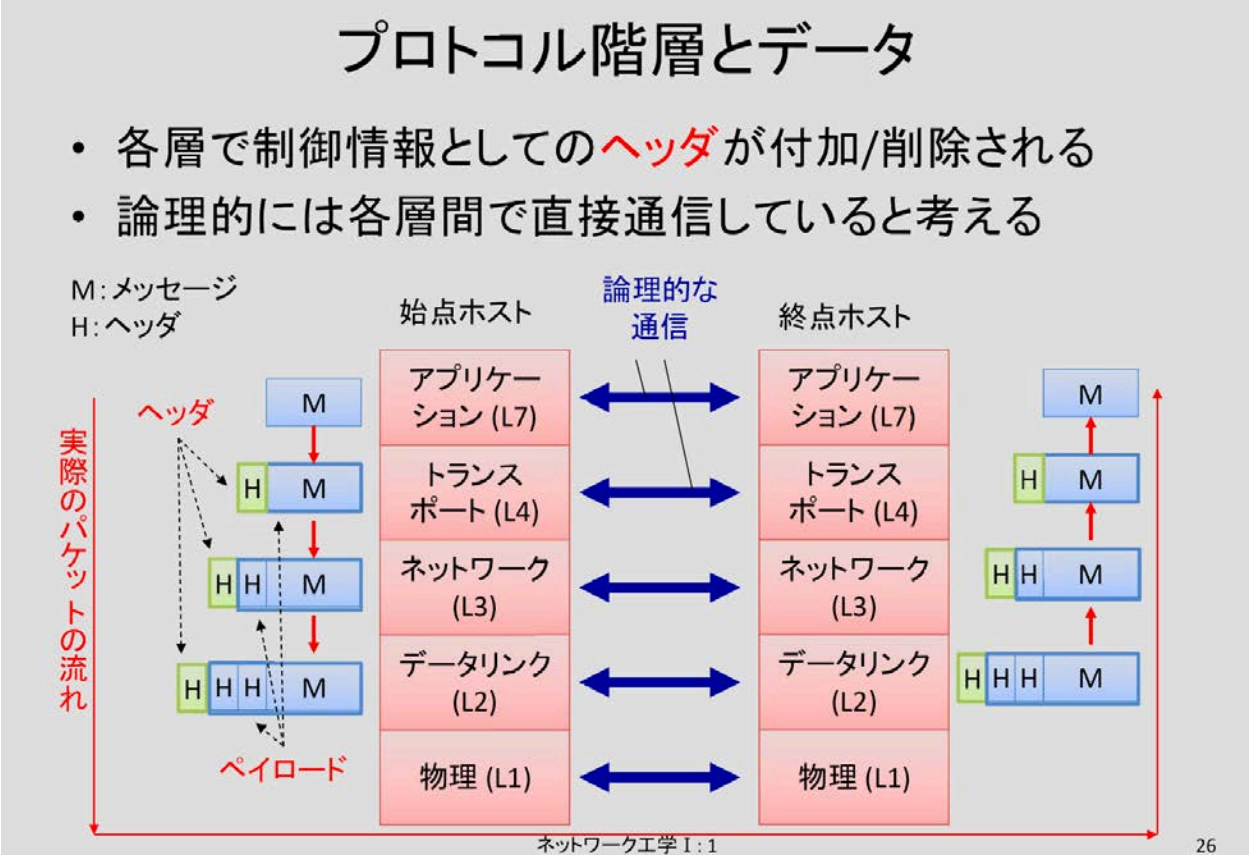
OSIの7層モデル


アプリケーション層 (L7)
メッセージ
アプリケーション間のデータ交換
アプリケーション層 (L7)
プレゼンテーション層 (L6)
暗号→复原
データ表現形式の制御
プレゼンテーション層 (L6)
セッション層 (L5)
大量data传送 设置checkpoint
通信の単位(セッション)の制御
セッション層 (L5)
トランスポート層 (L4)
セグメント
end-to-endの通信制御
トランスポート層 (L4)
ネットワーク層 (L3)
パケット
ネットワーク層 (L3)
host-to-hostの通信制御
ネットワーク層 (L3)
データリンク層 (L2)
フレーム
データリンク層 (L2)
point-to-pointの通信制御
データリンク層 (L2)
物理層 (L1)
電気信号
物理層 (L1)
回線の物理的な制御
物理層 (L1)
ホスト
ルータ
ホスト

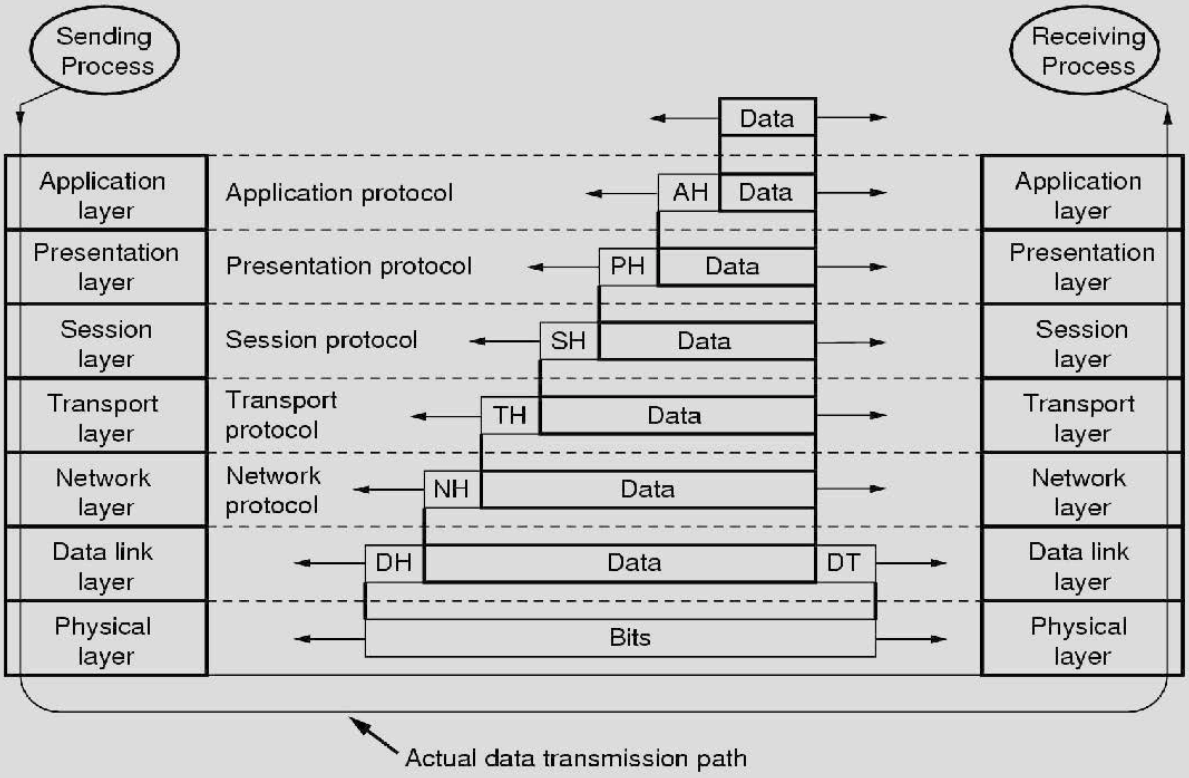
OSI Reference Model



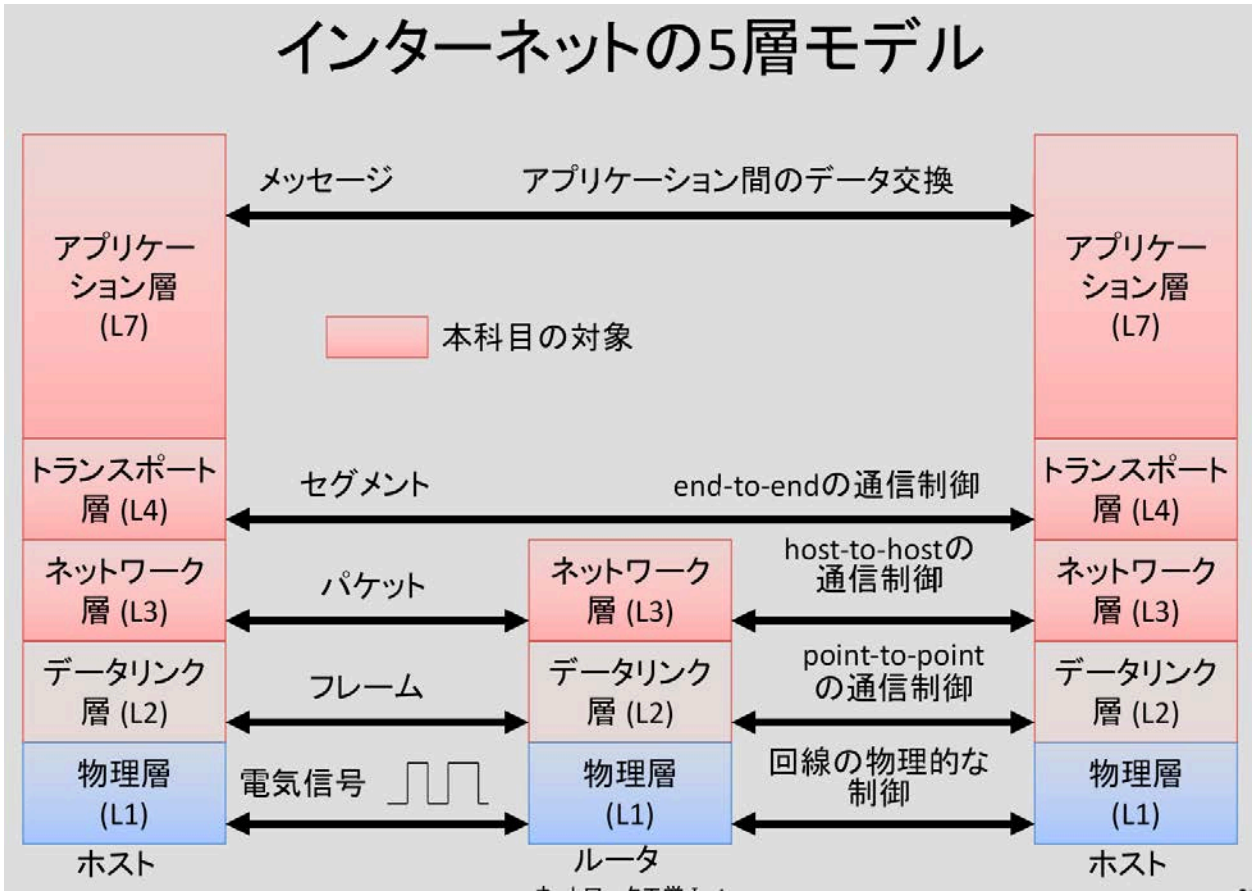
Data transfer in OSI



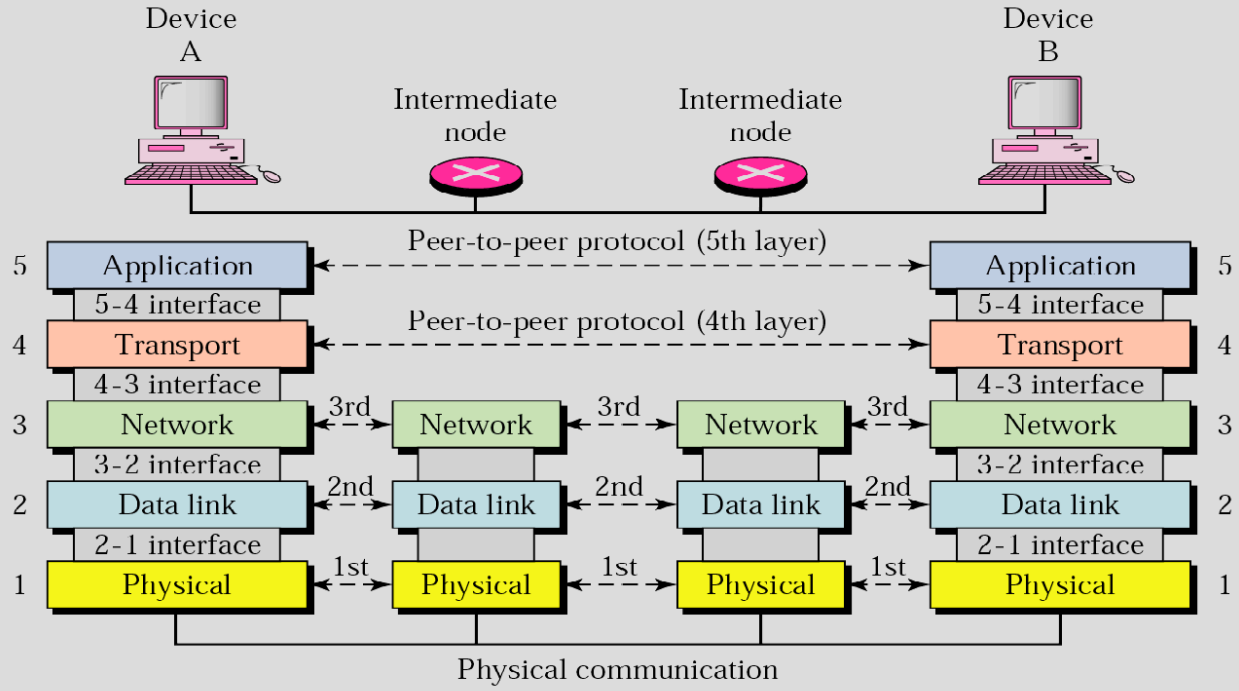
Data Transfer in OSI



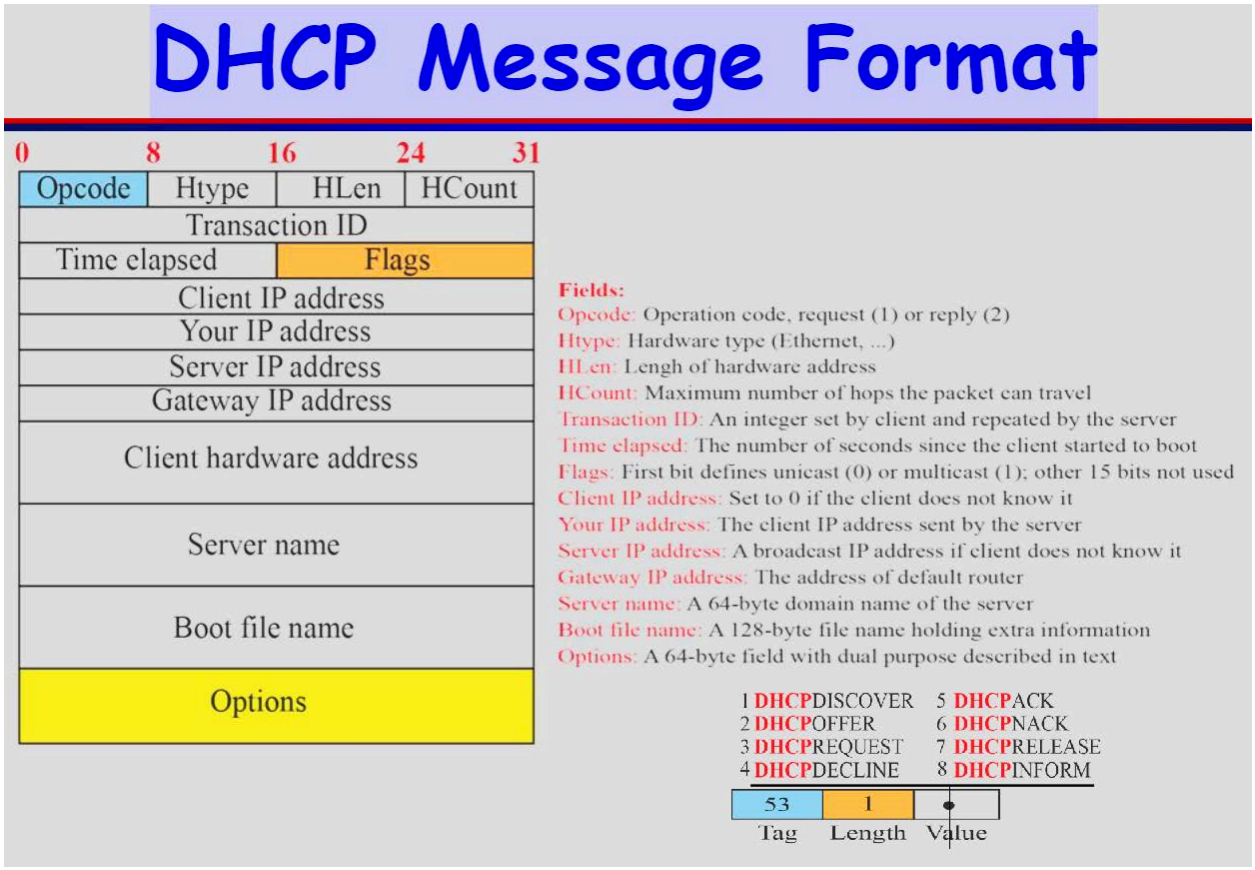
TCP/IP model

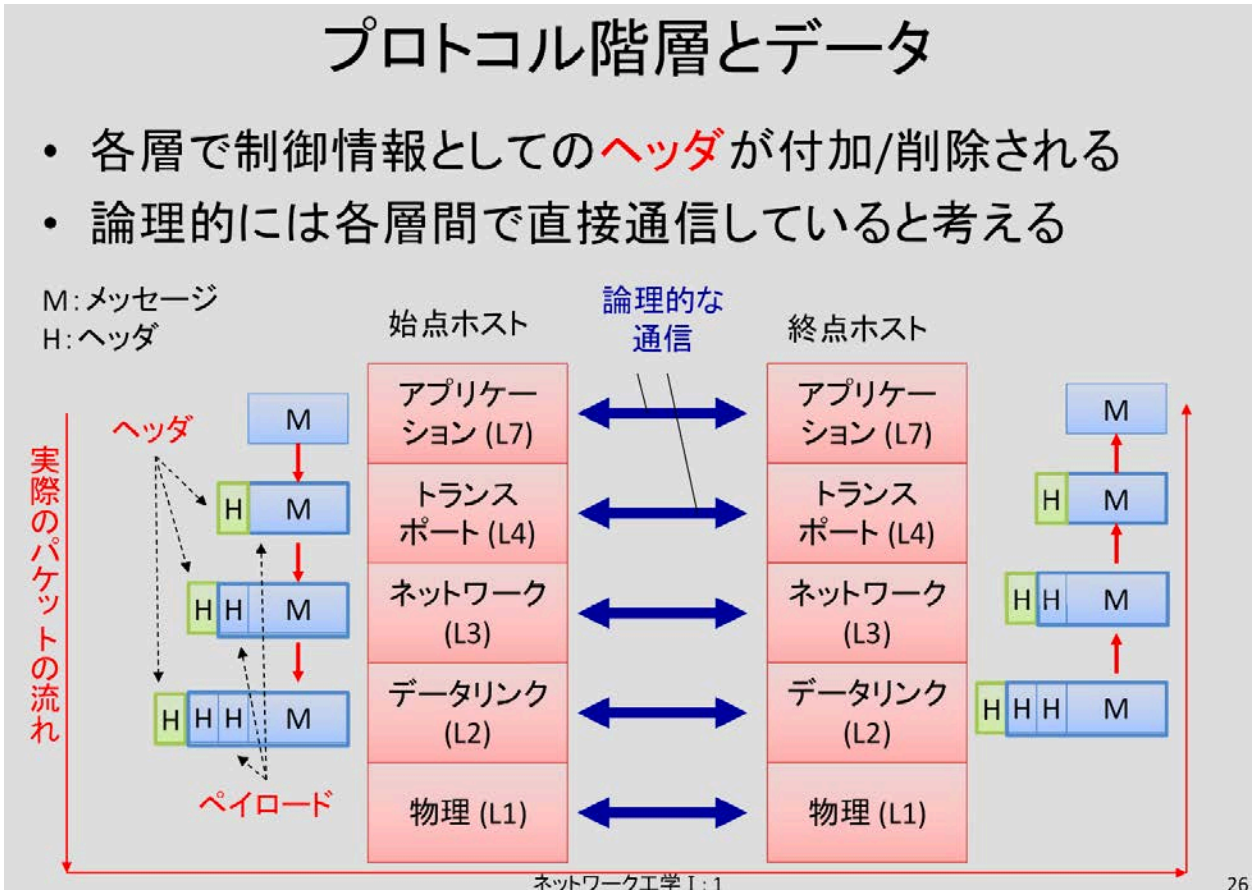

インターネットの5層モデル
アプリケーション層 (L7)
メッセージ
アプリケーション間のデータ交換
アプリケーション層 (L7)
本科目の対象
トランスポート層 (L4)
セグメント
end-to-endの通信制御
トランスポート層 (L4)
ネットワーク層 (L3)
パケット
ネットワーク層 (L3)
host-to-hostの通信制御
ネットワーク層 (L3)
データリンク層 (L2)
フレーム
データリンク層 (L2)
point-to-pointの通信制御
データリンク層 (L2)
物理層 (L1)
電気信号
物理層 (L1)
回線の物理的な制御
物理層 (L1)
ホスト
ルータ
ホスト

TCP/IP Model

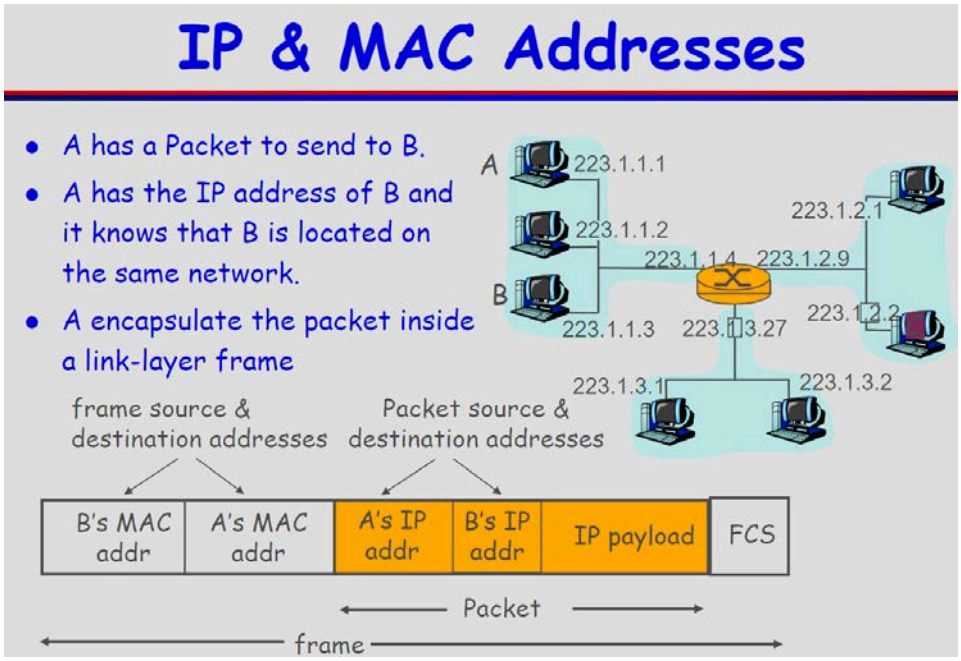

Device A
Device B
Intermediate node
Intermediate node
Peer-to-peer protocol (5th layer)
Peer-to-peer protocol (4th layer)
5
Application
5-4 interface
4
Transport
4-3 interface
3
Network
3-2 interface
2
Data link
2-1 interface
1
Physical
3rd
2nd
1st
Physical communication

DHCP Message Format





IP & MAC Addresses



IP & MAC Addresses (Cont.)

- 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 is a flat address ➜ portability
  - can move LAN card from one LAN to another
- IP is a hierarchical address ➜ not portable
  - address depends on IP subnet to which node is attached

IP & MAC Addresses (Cont.)

- **32-bit IP address:**
  - network-layer address for interface
  - used for layer 3 (network layer) forwarding
  - e.g.: 128.125.7.11 Decimal (base 10) notation (each "numeral" represents 8bits)
- MAC (or LAN or physical or Ethernet) address:
  - 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, also sometimes software settable
  - e.g.: 1A-2F-BB-76-09-AD

hexadecimal (base 16) notation (each "numeral" represents 4 bits)

Port Number

Port Numbers

- Port Numbers allow receiving host to determine to which local process the message be delivered
- Port numbers are integers between 0 and 65,535 = 2^16 (16bit)
- Client process defines itself with a port number chosen **randomly** by the underlying transport layer protocol.
- Server process defines itself by a **well-known port number**. The ports ranging from 0~1023 are well-known port numbers and are assigned and controlled by **ICANN**.

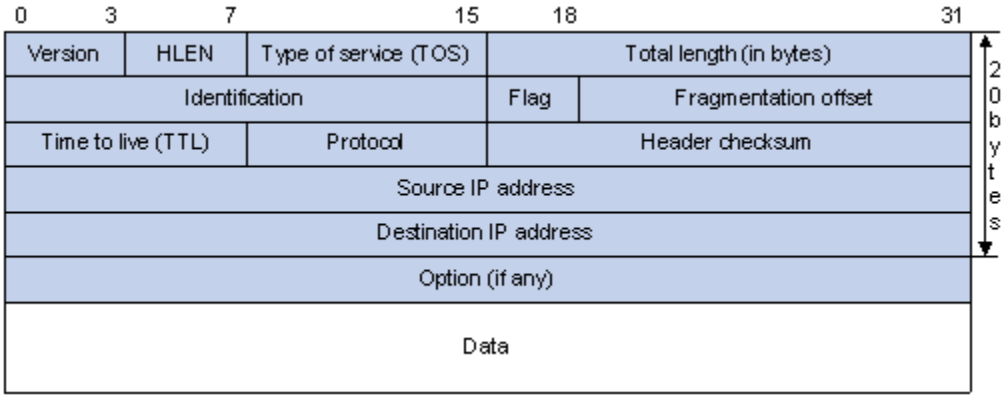
ARP packet

ARP Packet

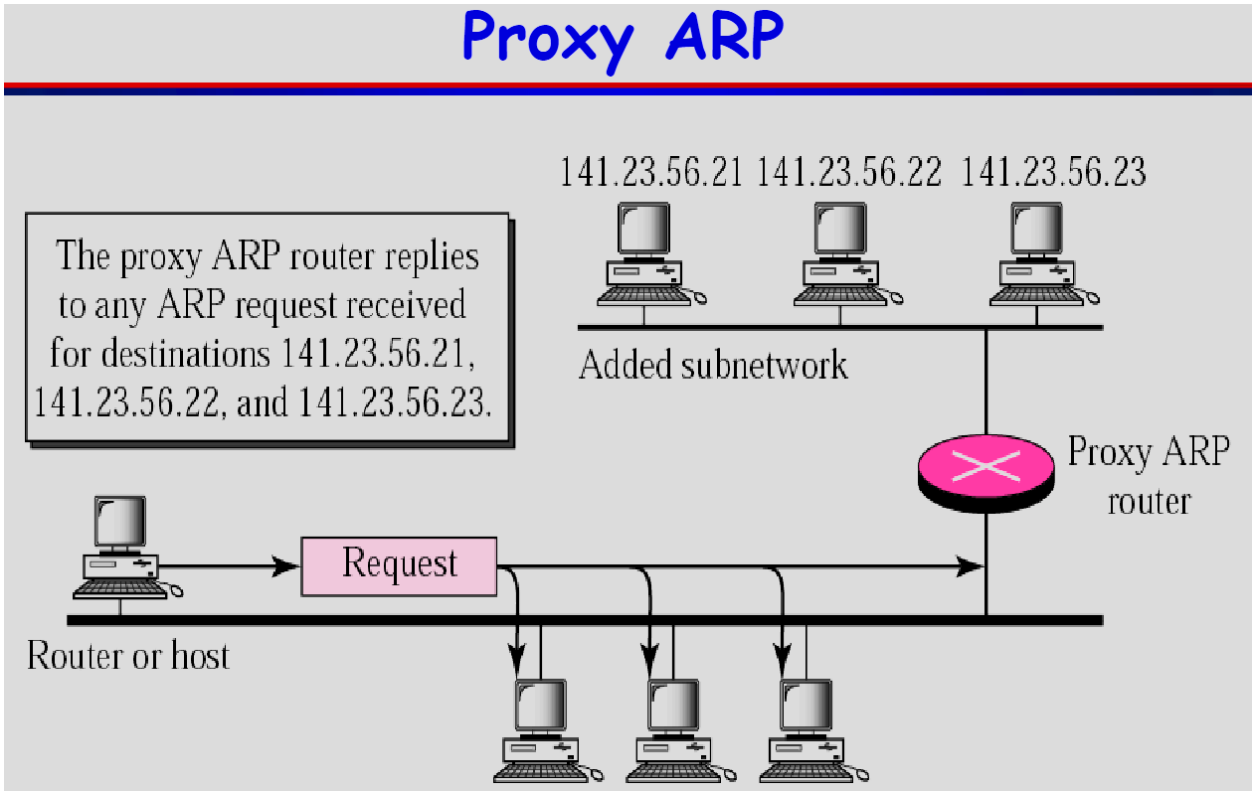
Hardware Type		Protocol Type
Hardware length	Protocol length	Operation Request 1, Reply 2
Sender hardware address (For example, 6 bytes for Ethernet)		
Sender protocol address (For example, 4 bytes for IP)		
Target hardware address (For example, 6 bytes for Ethernet) (It is not filled in a request)		
Target protocol address (For example, 4 bytes for IP)		

Diff with IP packet:

- ARP packet doesn't have a ***payload***!



Proxy ARP

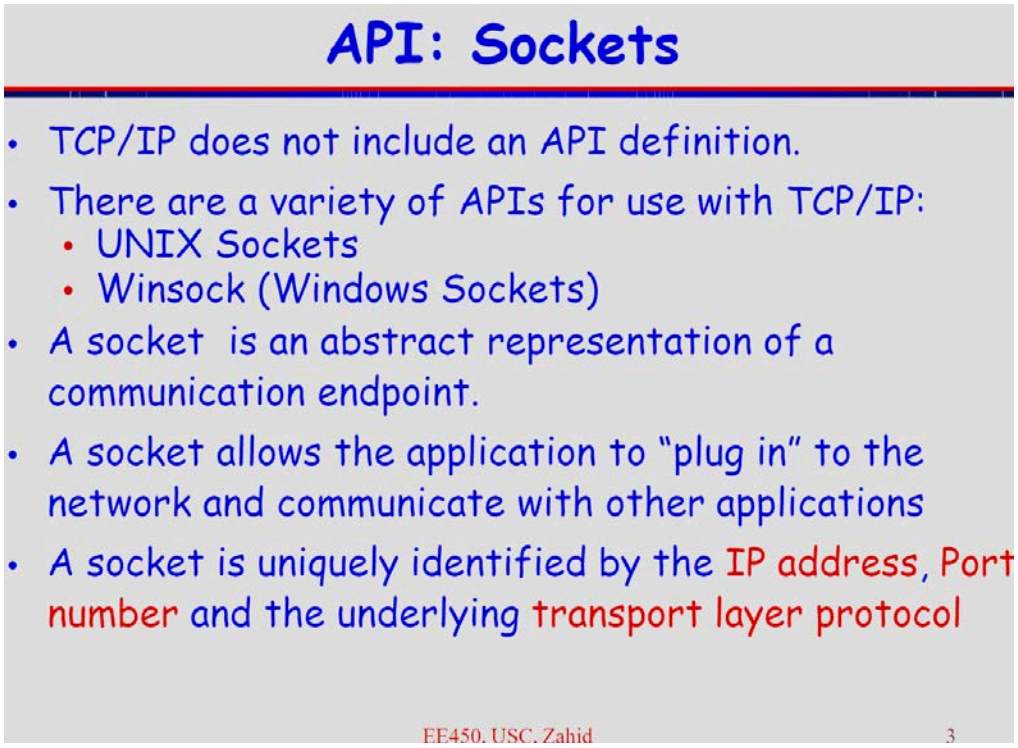

Proxy ARP
141.23.56.21 141.23.56.22 141.23.56.23
The proxy ARP router replies to any ARP request received for destinations 141.23.56.21, 141.23.56.22, and 141.23.56.23.
Added subnetwork
Proxy ARP router
Request
Router or host

DHCP (Dynamic Host Configuration Protocol)

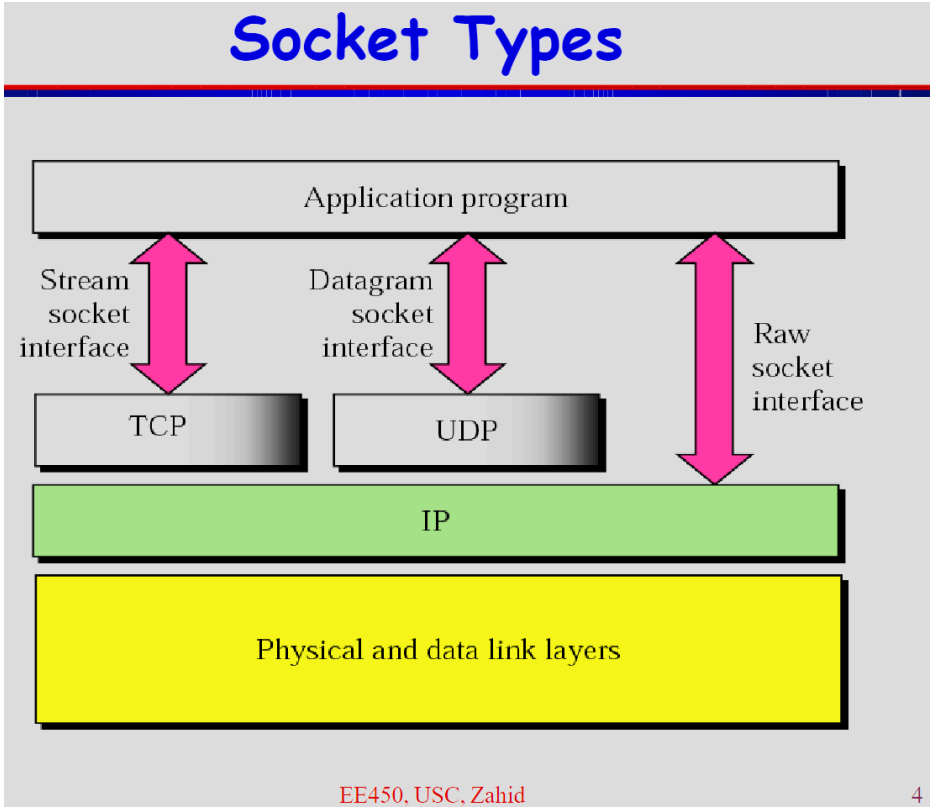
Dynamic Host Configuration Protocol (DHCP)

- DHCP is a client/server application designed to provide a centralized approach to configuring and maintaining IP addresses
- Four basic steps involved in obtaining an IP address:
  - Discovery Phase
  - Offer Phase
  - Request Phase
  - Acknowledgement Phase

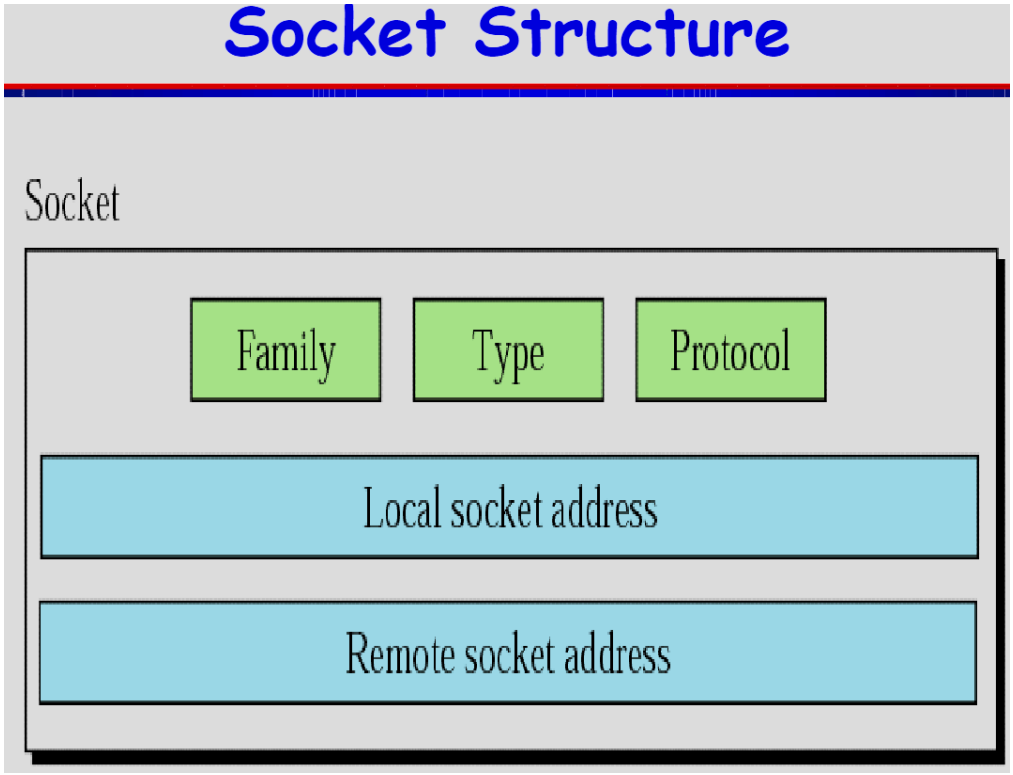
API: Sockets


API: Sockets
• TCP/IP does not include an API definition.
• There are a variety of APIs for use with TCP/IP:
• UNIX Sockets
• Winsock (Windows Sockets)
• A socket is an abstract representation of a communication endpoint.
• A socket allows the application to "plug in" to the network and communicate with other applications
• A socket is uniquely identified by the IP address, Port number and the underlying transport layer protocol
EE450, USC, Zahid
3

Socket Types



Socket Structure



Basic Sockets API

Basic Sockets API

- Clients and Servers differ in some aspects of their use of API and they are the same in other aspects
- Both client and server programs begin by asking the NOS to "create" a socket. The function to accomplish that is Socket ( )
  - int socket (int protocol family, int type, int protocol)
  - Protocol family: PF_INET
  - Type: SOCK_STREAM, SOCK_DGRAM
  - Protocol: TCP, UDP
- Return value of Socket ( ) is a non-negative integer called Socket Descriptor (or –1 if errors)

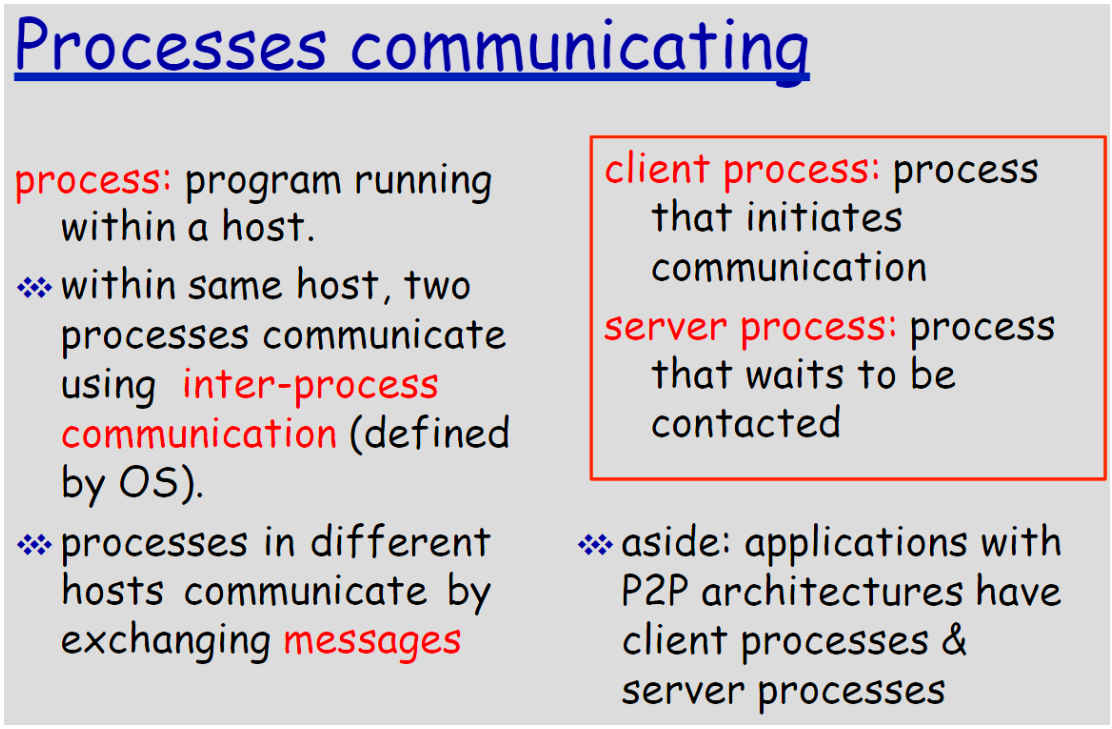
TCP Server

TCP Server

- Create a TCP socket using Socket ( )
- Assign a port number to the socket using Bind ( )
  - int bind (int socket, local address, address length)
  - Local address is the IP address and the port number of the server. It is this address that the client and the server need to agree on to communicate. Neither one need to know the client address.
  - Address length is length of address structure
  - If successful, Bind ( ) returns a 0, otherwise it returns -1
  - If successful, the socket at server side is "associated " to the local IP address and the local port number
- Listen to connections from clients using Listen ( )
  - int listen (int socket, int queue limit)
  - Queue limit is an upper bound on # of clients that can be waiting
  - If successful, Listen ( ) returns a 0, otherwise it returns -1

9.20 Discussion-Applications

Processes communicating



Web and HTTP

Web and HTTP

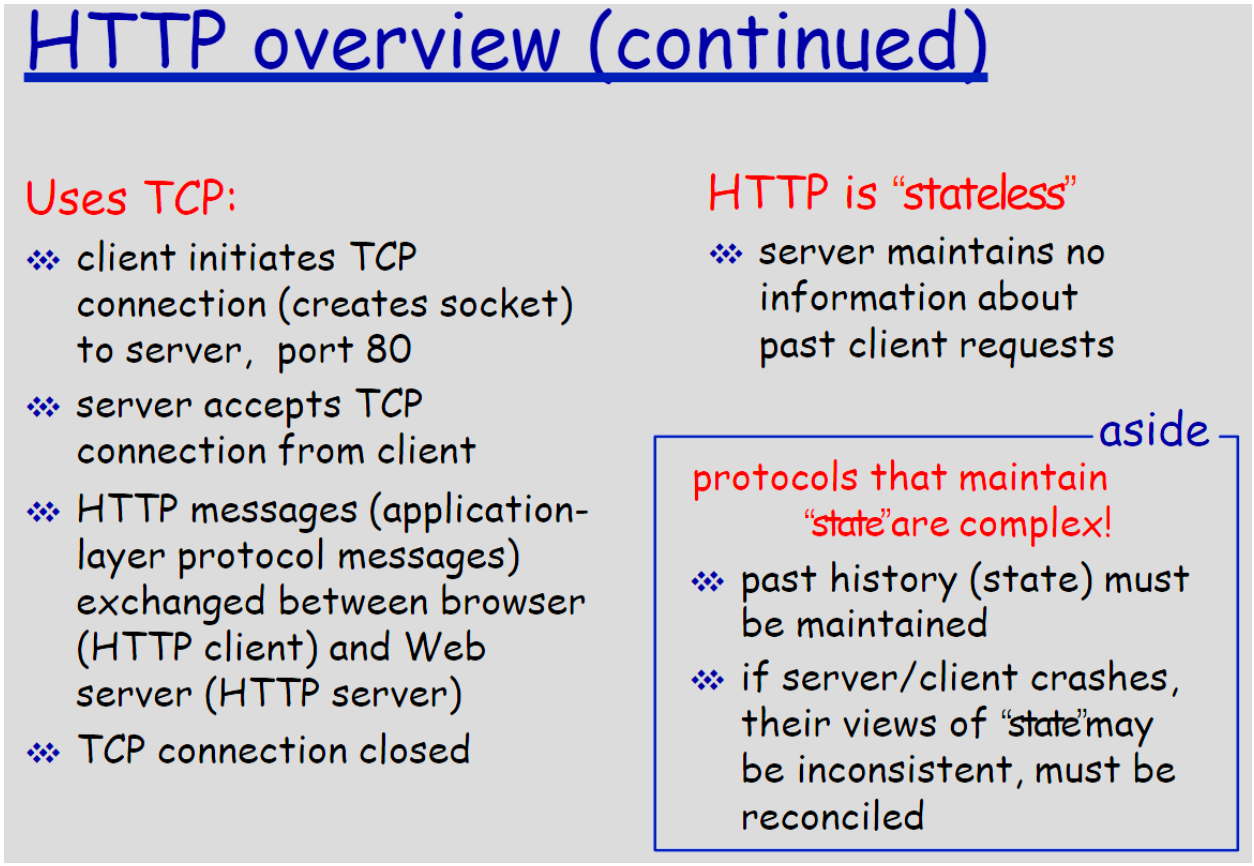
First, a review…

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

www.someschool.edu/someDept/pic.gif

host name | path name

HTTP overview (continued)



Types of HTTPs (persistent / non-persistent)

HTTP connections: two types

Non-persistent HTTP

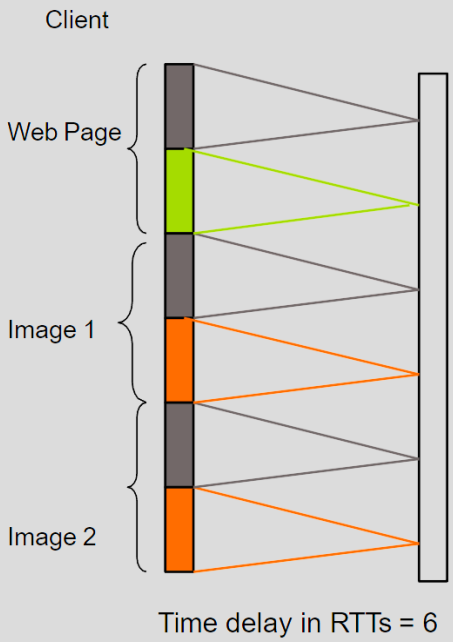
1. TCP connection opened
2. at most one object sent over TCP connection
3. TCP connection closed

downloading multiple objects required multiple connections

Persistent HTTP

- TCP connection opened to a server
- multiple objects can be sent over *single* TCP connection between client, and that server
- TCP connection closed

Non-Persistent: Rough calculation for number of RTTS

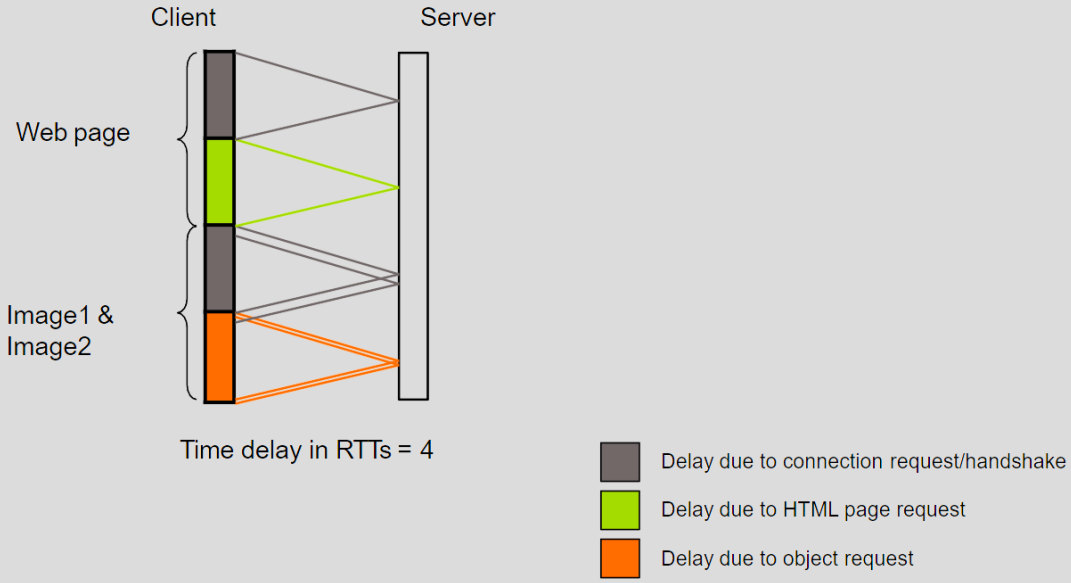


Time delay in RTTs = 6

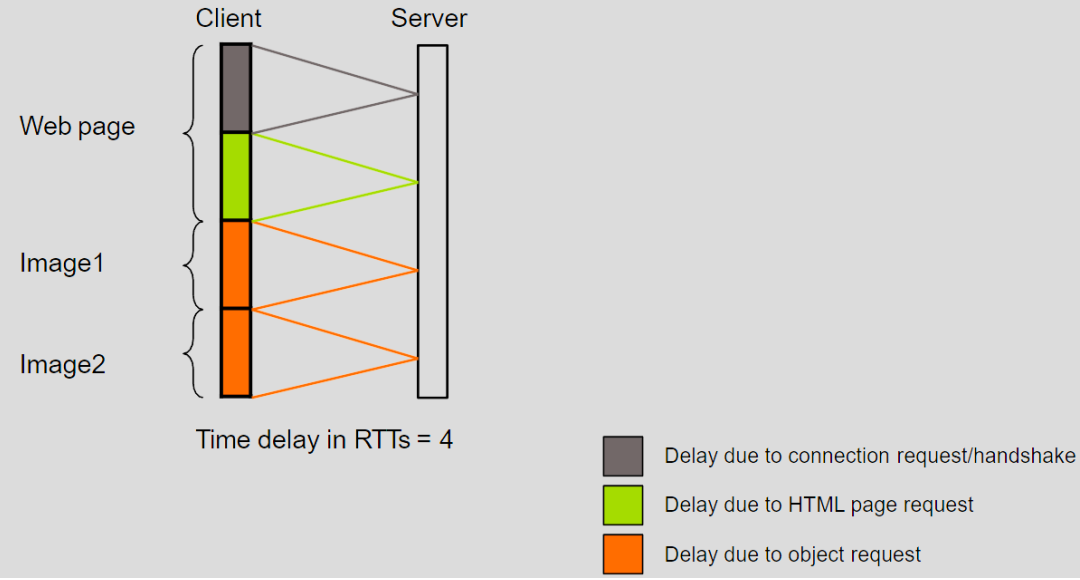
Can we reduce the number of RTTS?

Delay due to connection request/handshake
Delay Due to HTML Page Request
Delay Due to Object Request

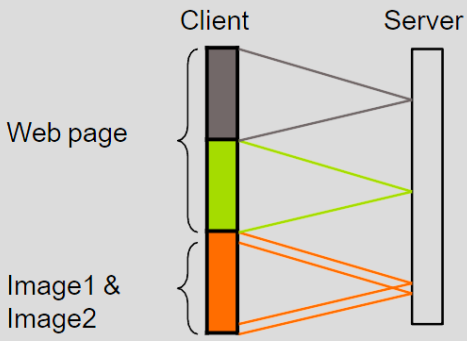
Non-Persistent with Parallel Connections: Rough calculation



Persistent Connection without Pipelining: Rough calculation



Persistent Connection with Pipelining: Rough calculation



Time delay in RTTs = 3

Delay due to connection request/handshake

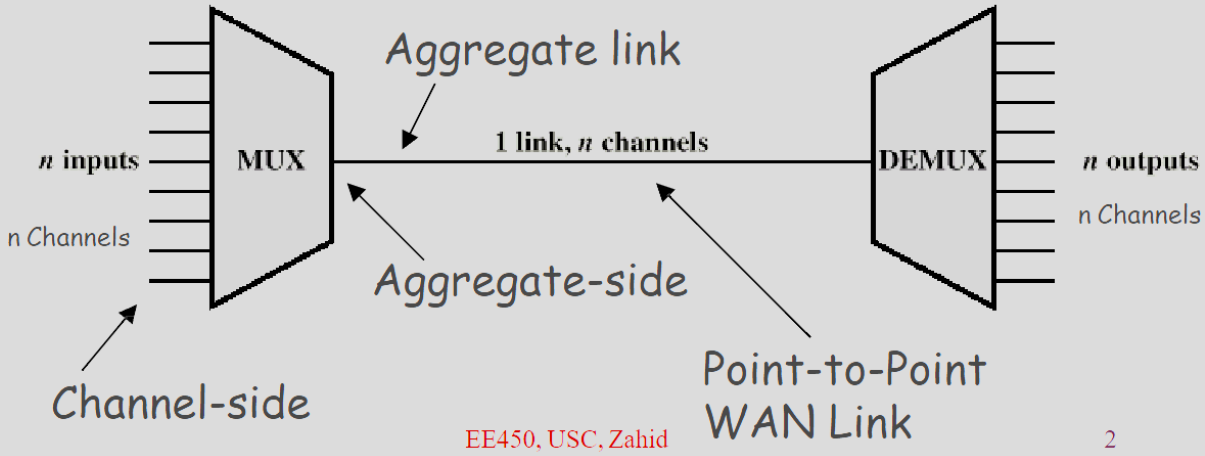
Delay due to HTML page request

Delay due to object request

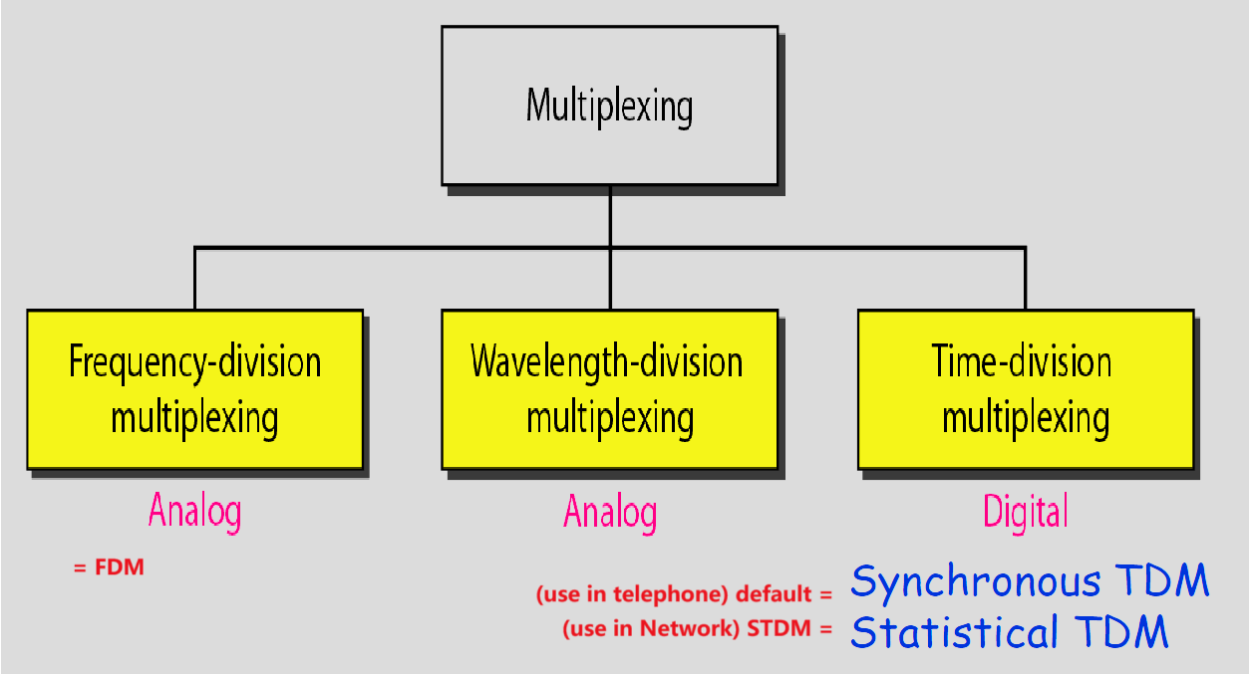
9.24 Multiplexing

Multiplexing

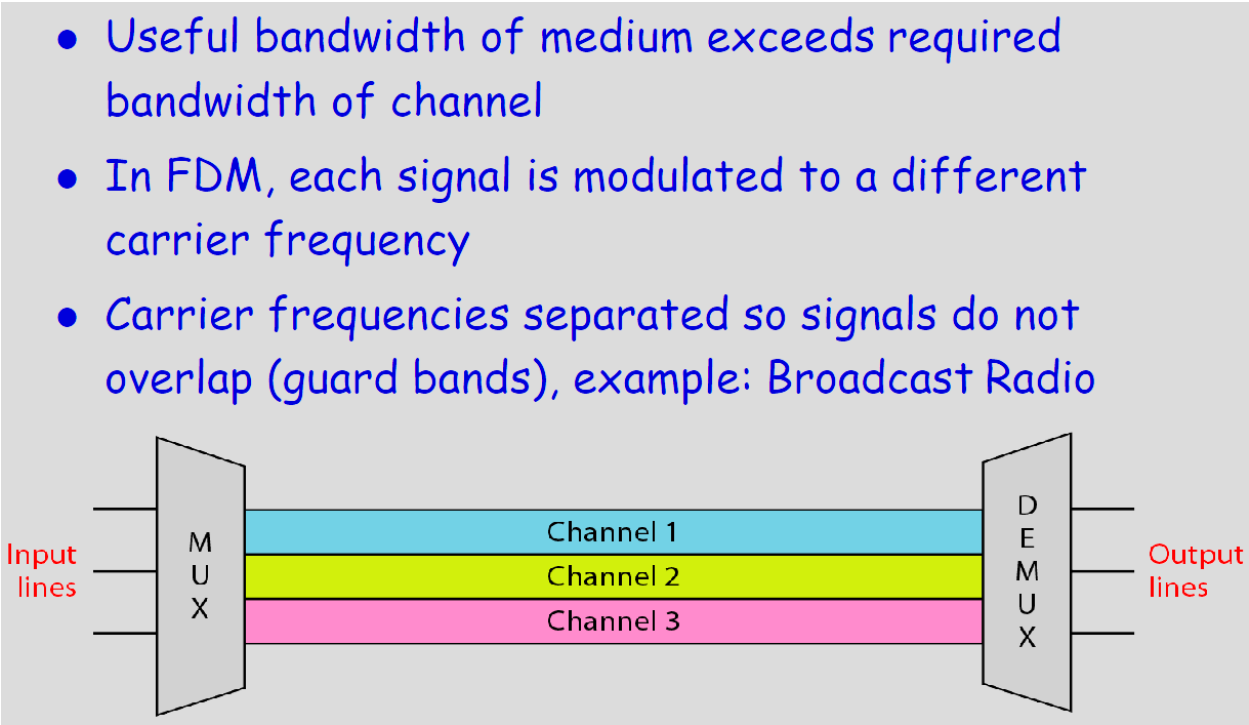
- Multiplexing is a resource sharing process allowing information from several information sources to be aggregated onto a single, high-speed link



Categories of Multiplexing

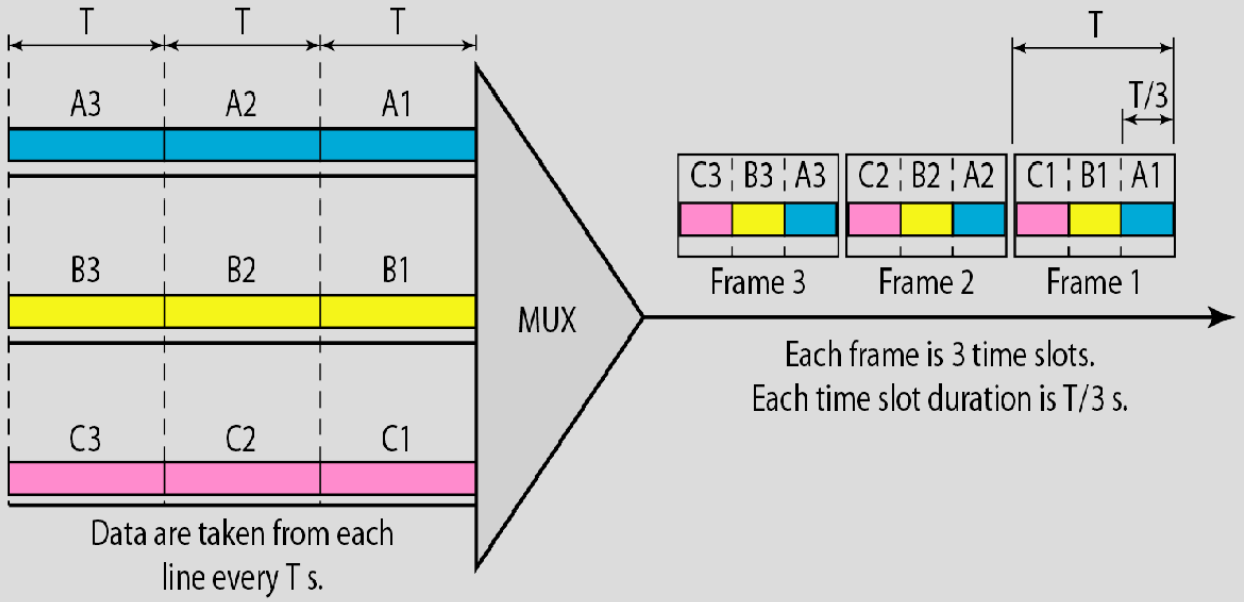


FDM (Frequency Division Multiplexing)



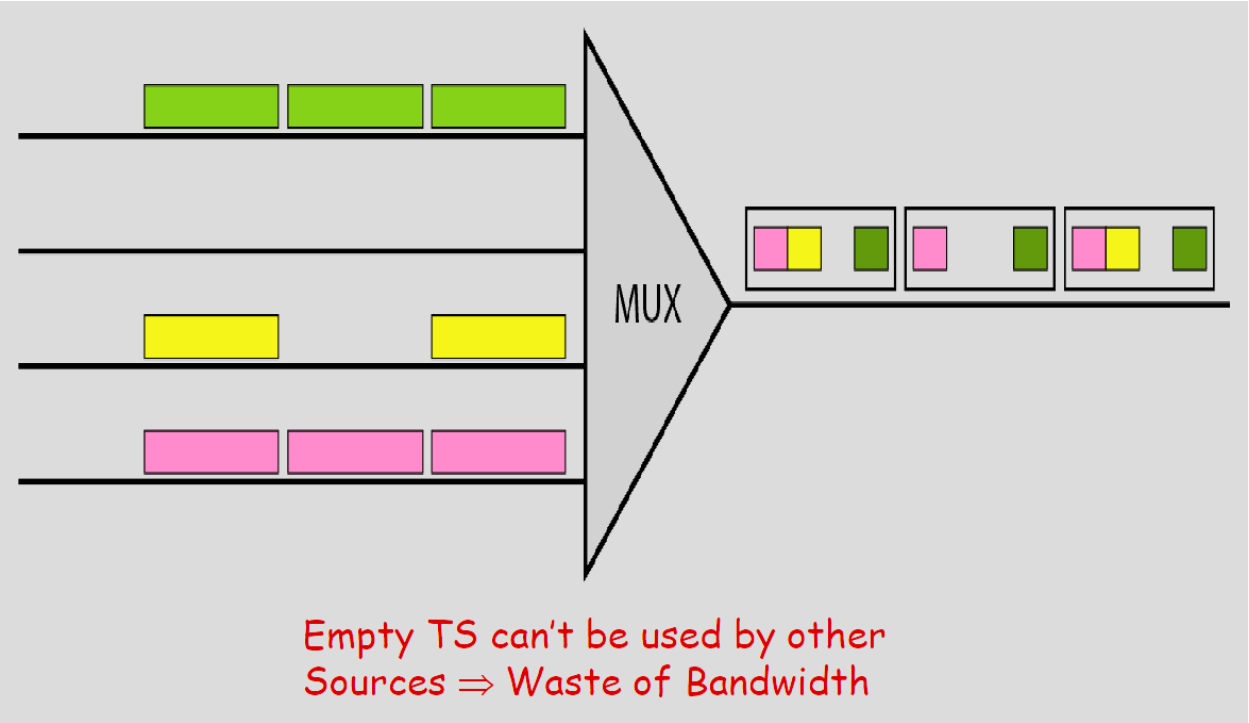
Synchronous TDM

- Data rate of medium exceeds data rate of digital signal to be transmitted
- Multiple digital signals interleaved in time
- May be at bit level or block of bits
- Time slots pre-assigned to sources and fixed
- Time slots allocated even if source is idle
- Time slots do not have to be evenly distributed amongst sources

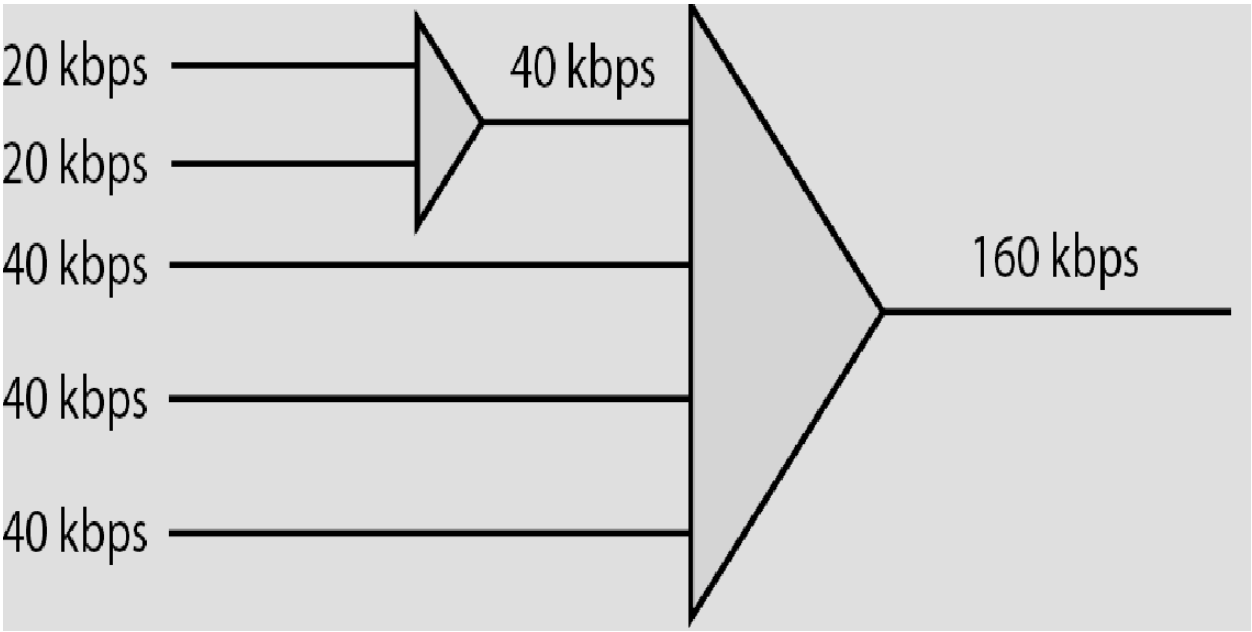


In synchronous TDM, the data rate of the link is *n* times faster, and the unit duration is *n* times shorter.

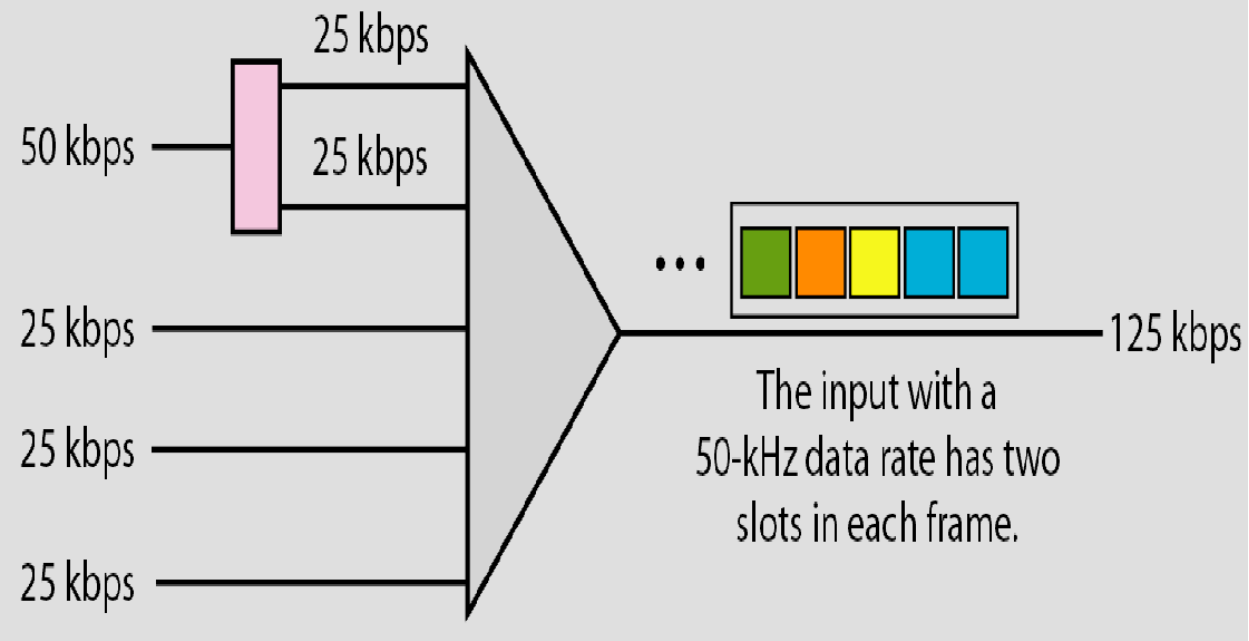
Empty Time Slots



Multilevel Multiplexing



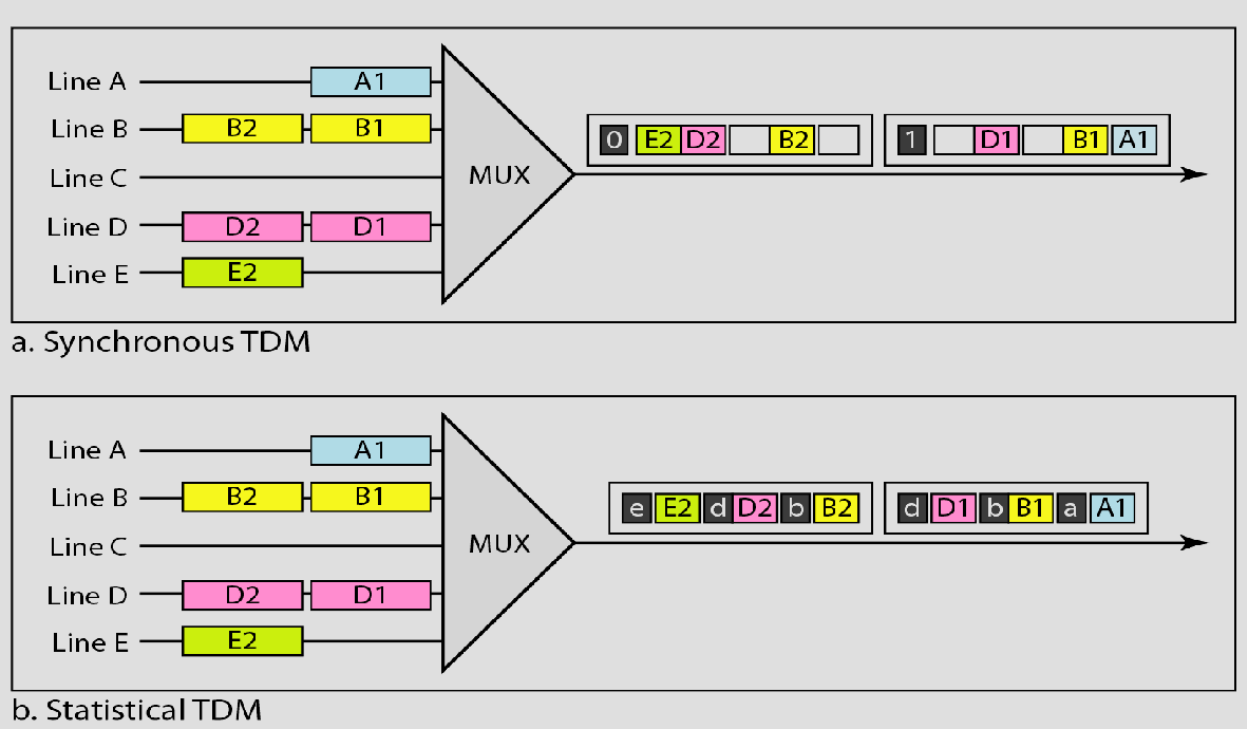
Multiple-Slot Multiplexing



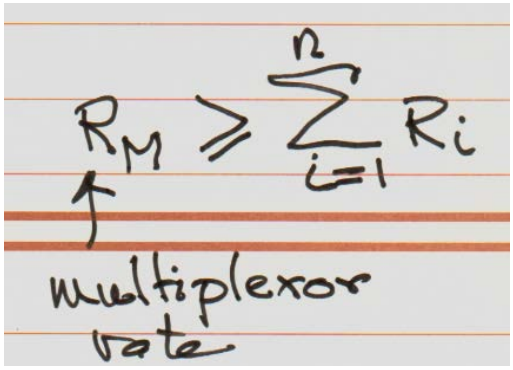
STDM, Statistical (Asynchronous) TDM

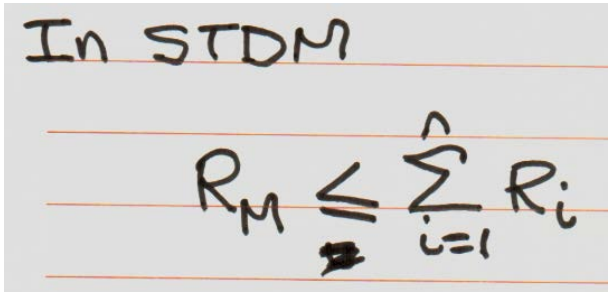
- In Synchronous TDM many slots are wasted
- Statistical TDM allocates time slots dynamically, i.e. based on demand
- Every Slot has to start with a header identifying the device (address)
- Multiplexer scans input lines and collects data until frame full
- Data rate on line lower than aggregate rates of input lines

Synchronous vs. Statistical



In Synchronous TDM vs. in STDM





Conclusions

- TDM – Guarantees the User a bandwidth but on the contrary wastes valuable carrier capacity. Suitable for streamy type traffic like voice (digitized)
- STDM – Utilizes unused time slots. Suitable for Bursty-type traffic such as data
  - More efficient use of capacity
  - When times are busy, user suffers delay

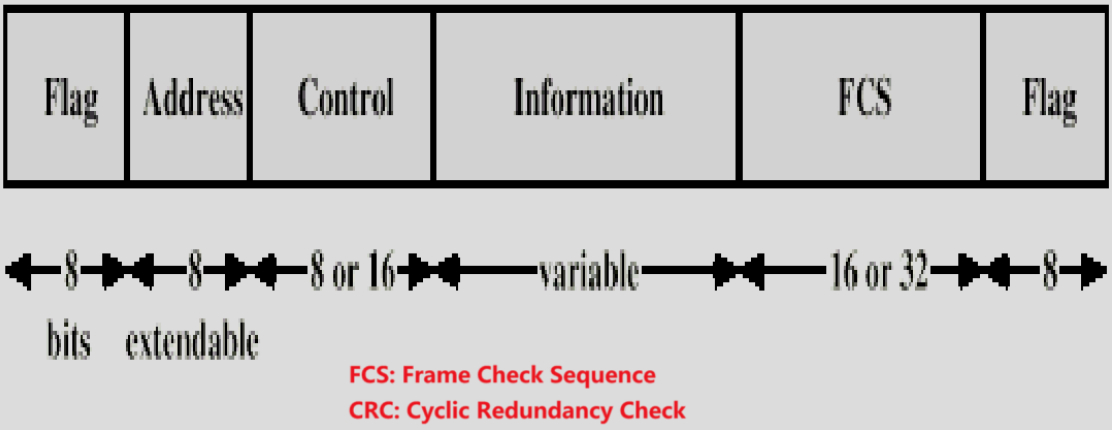
9.24 Data Link Control Protocols

Data Link Layer Services

- Framing
  - Encapsulate packet into frame, adding header/trailer
  - Establish frame synchronization
- Error Detection & Control
  - Errors caused by signal attenuation, noise.
  - Receiver detects presence of errors:
    - Receiver drops frame
    - Receiver requests retransmission (ARQ)
- Flow Control
  - Ensuring the sender does not overwhelm the receiver (i.e., preventing buffer overflow)

Data Frame Structure

Typical Frame Structure



FCS: Frame Check Sequence
CRC: Cyclic Redundancy Check

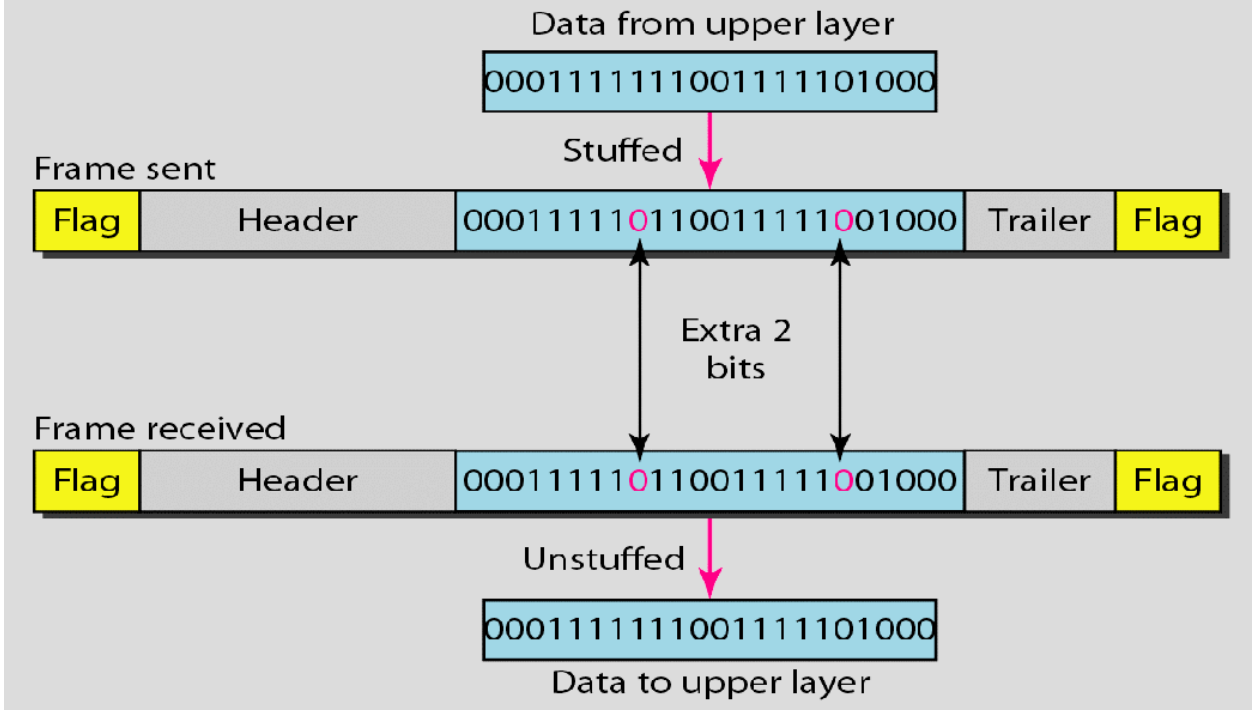
We shall see that the above structure does not work in a Multipoint link (like in LANs)

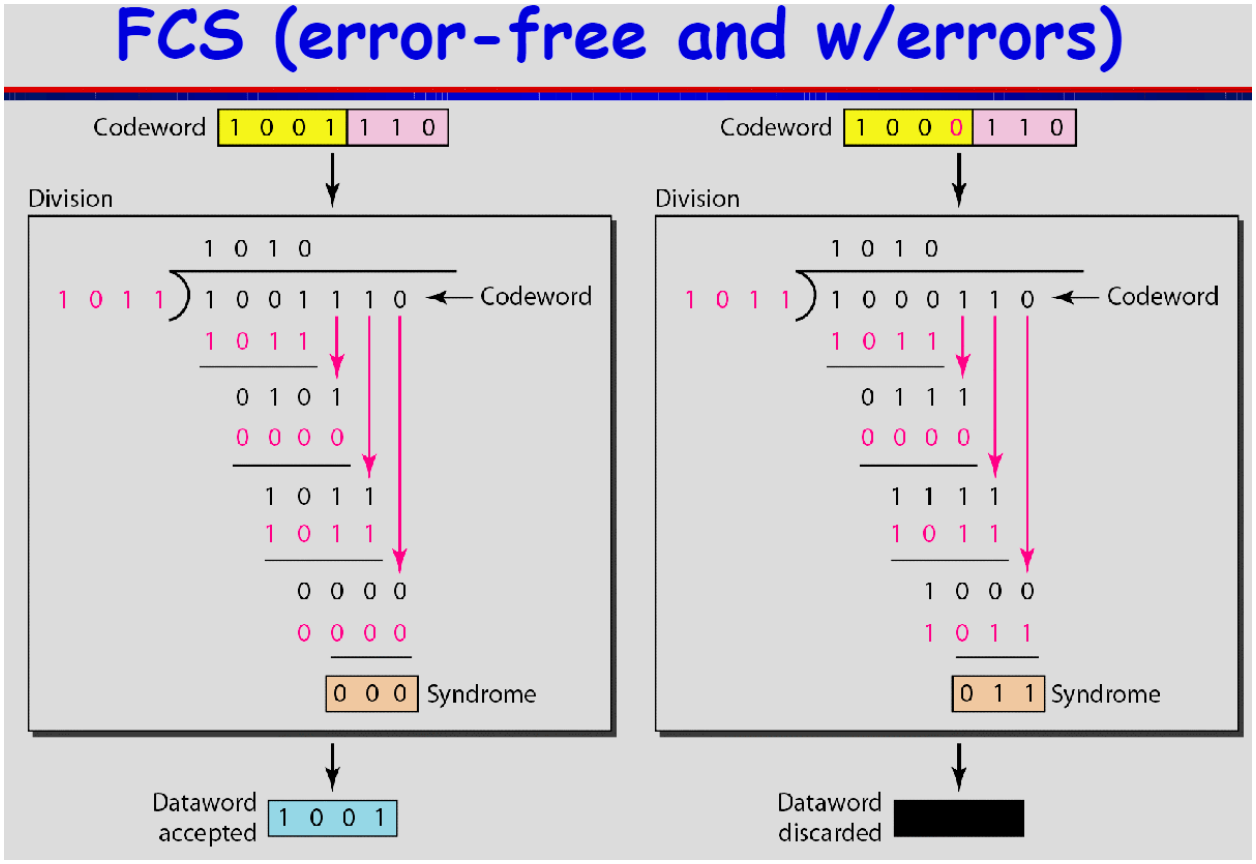
Frame Synchronization (Flag)

Frame Synchronization

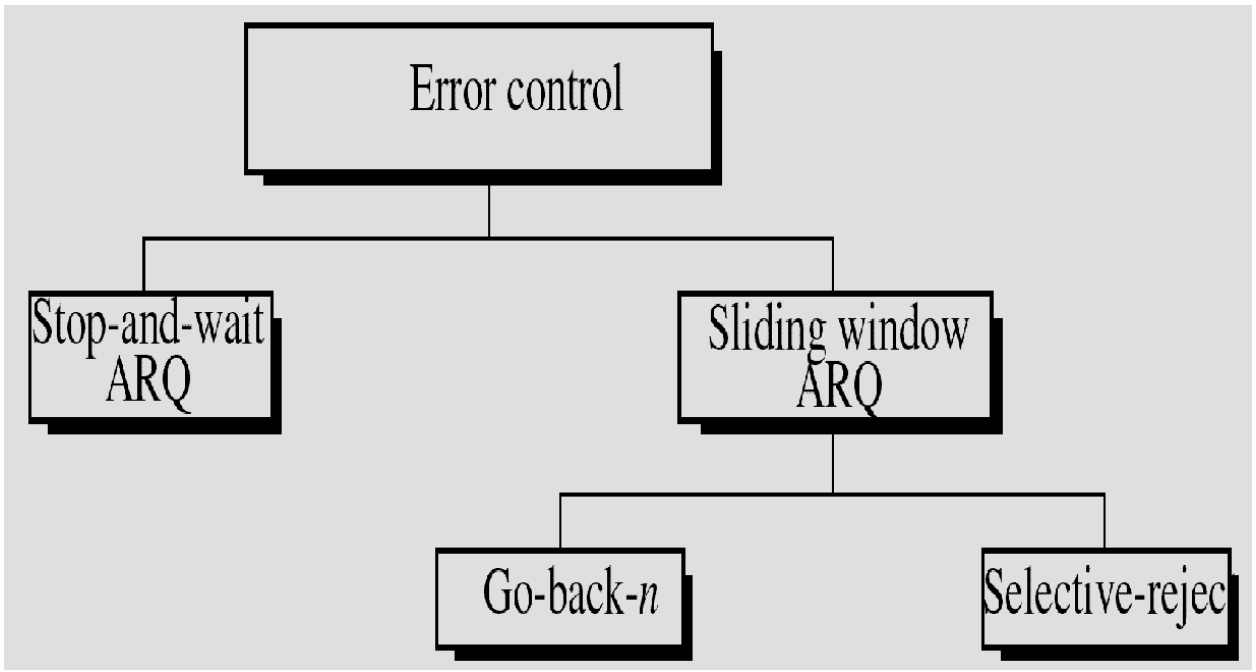
- A special pattern, called a Flag (01111110) appears at the beginning and the end of the frame
- Receiver hunts for flag sequence to synchronize
- **Bit stuffing** used to avoid confusion with data containing 01111110
  - 0 inserted after every sequence of five 1s
  - If receiver detects five 1s it checks next bit
  - If 0, it is deleted
  - If 1 and seventh bit is 0, accept as flag
  - If sixth and seventh bits 1, sender is indicating abort

Bit Stuffing and un-Stuffing

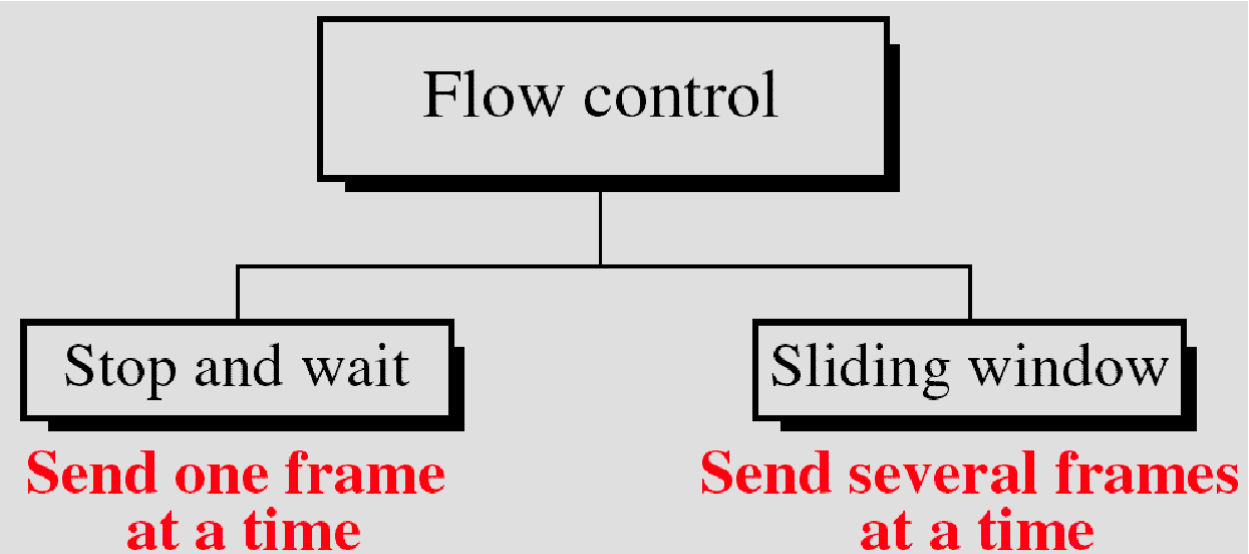




Error Control Procedures

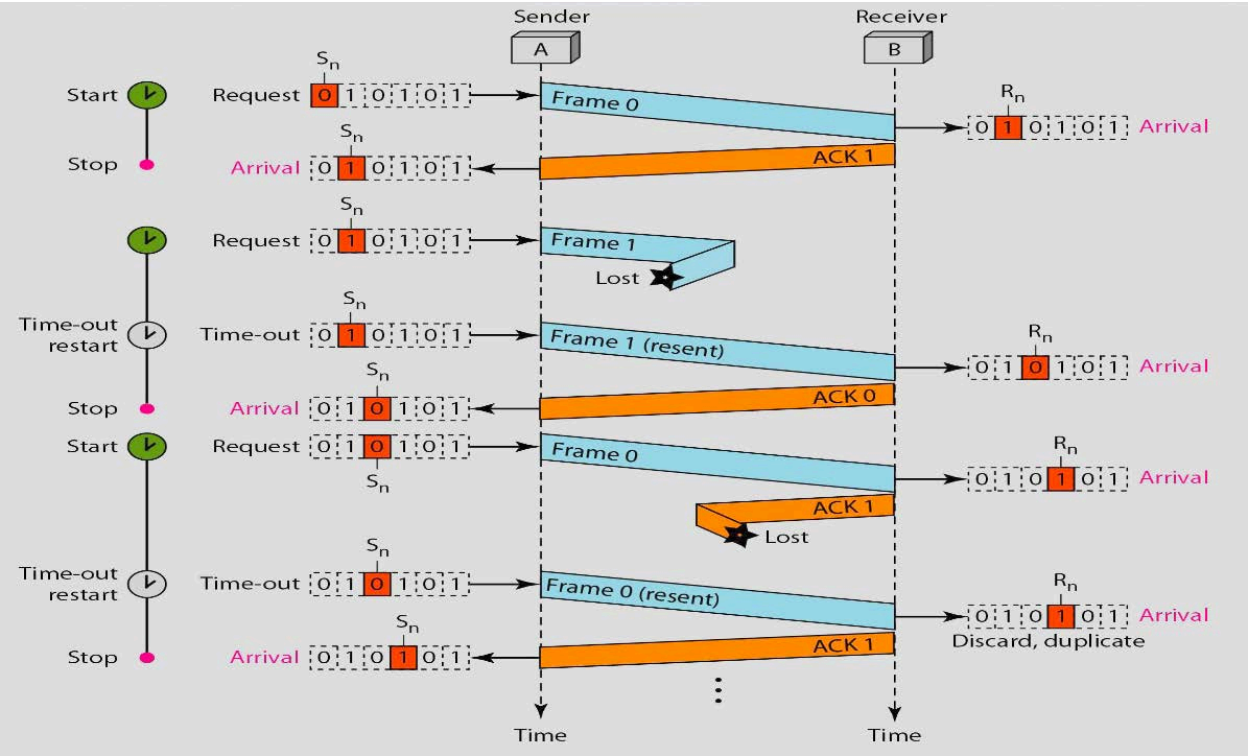


Flow Control Procedures



Stop and Wait ARQ

- Source transmits a single frame at a time
- Wait for ACK
- If received frame damaged, discard it
  - Transmitter has timeout timer
  - If no ACK within t_{out} = timeout, retransmit frame
  - Transmitter buffers copy of frame until ACK is received
- If ACK damaged, transmitter will not recognize it
  - Transmitter will retransmit
  - Receiver gets two copies of frame and discards one.
  - Use ACK_0 (recv'd frame 1) and ACK_1 (recv'd frame 0)



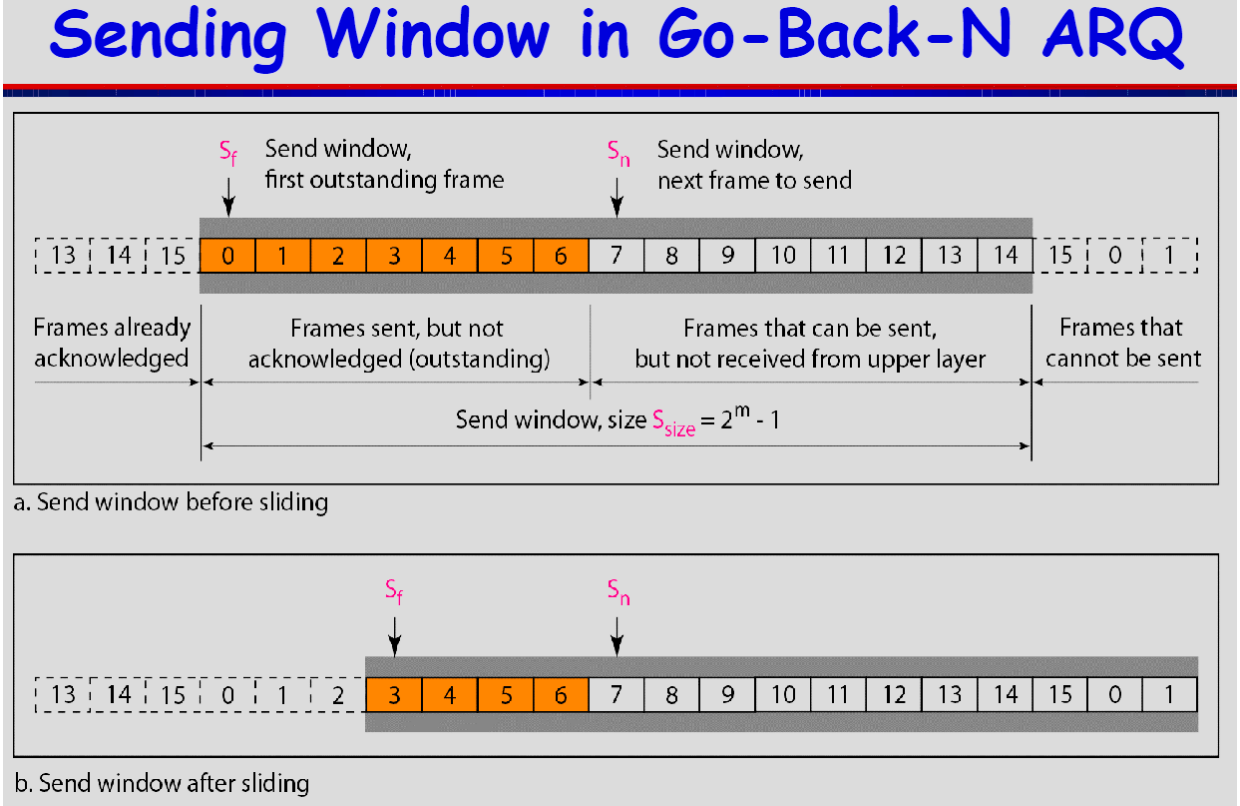
Link Utilization is Stop & Wait ARQ

- Link Bandwidth: 1 Mbps
- RTT: 20 msec
- Frame Length: 1000 bits
- BW x Delay Product = 20000 bits = 20 frames
- Sender can ONLY send 1 frame during RTT
- Hence Link Utilization is 5%
- Really Bad!

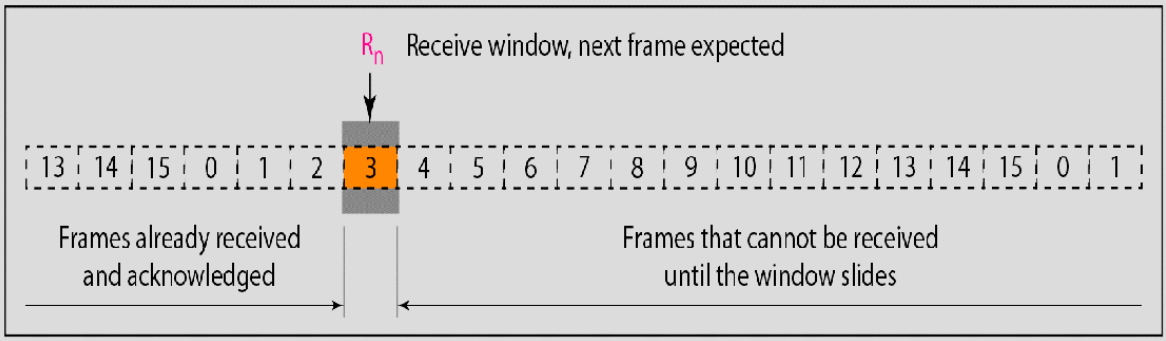
Go-Back-N ARQ

- Based on sliding window Protocol
- If no error, ACK as usual with next frame expected
- Use window to control number of outstanding frames
- If error, reply with rejection
  - Discard that frame and all future frames until error frame received correctly
  - Transmitter must go back and retransmit that frame and all subsequent frames

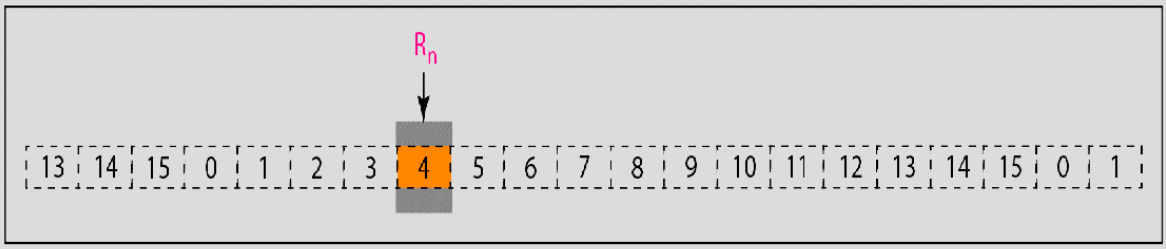
Window in Go-Back-N ARQ



Receiver Window in Go-Back-N ARQ

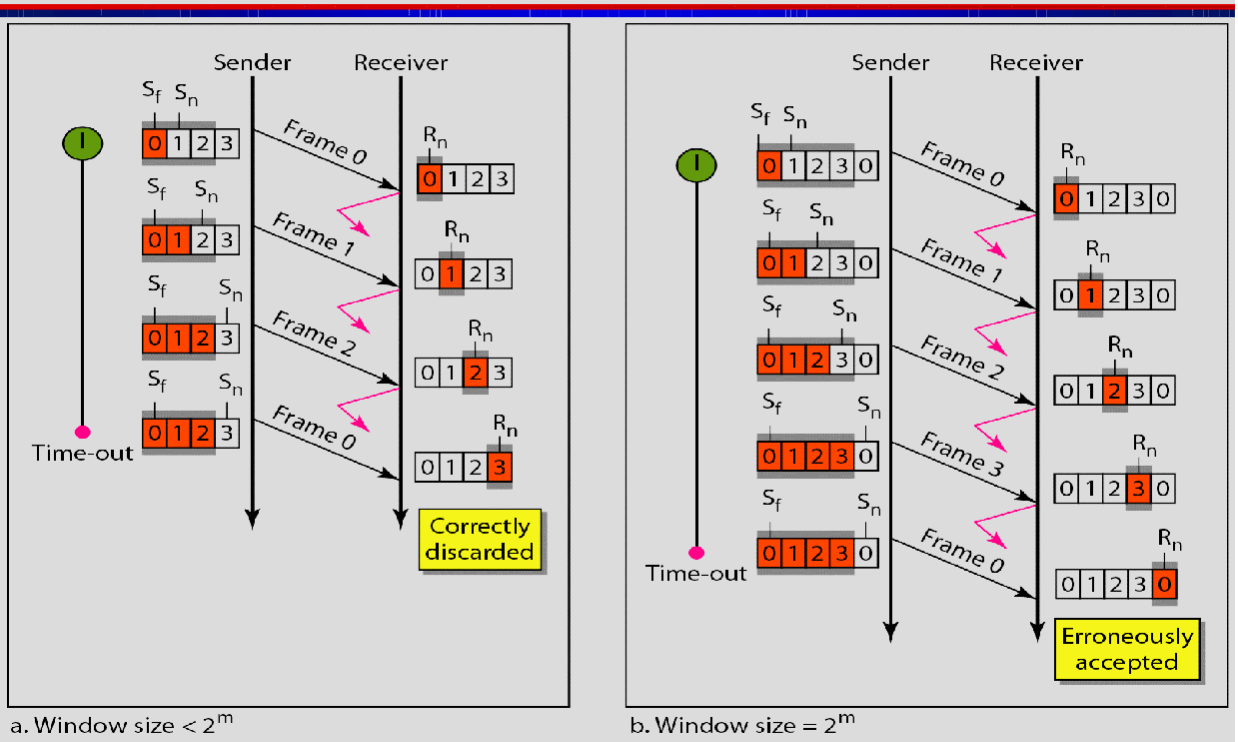


a. Receive window



b. Window after sliding

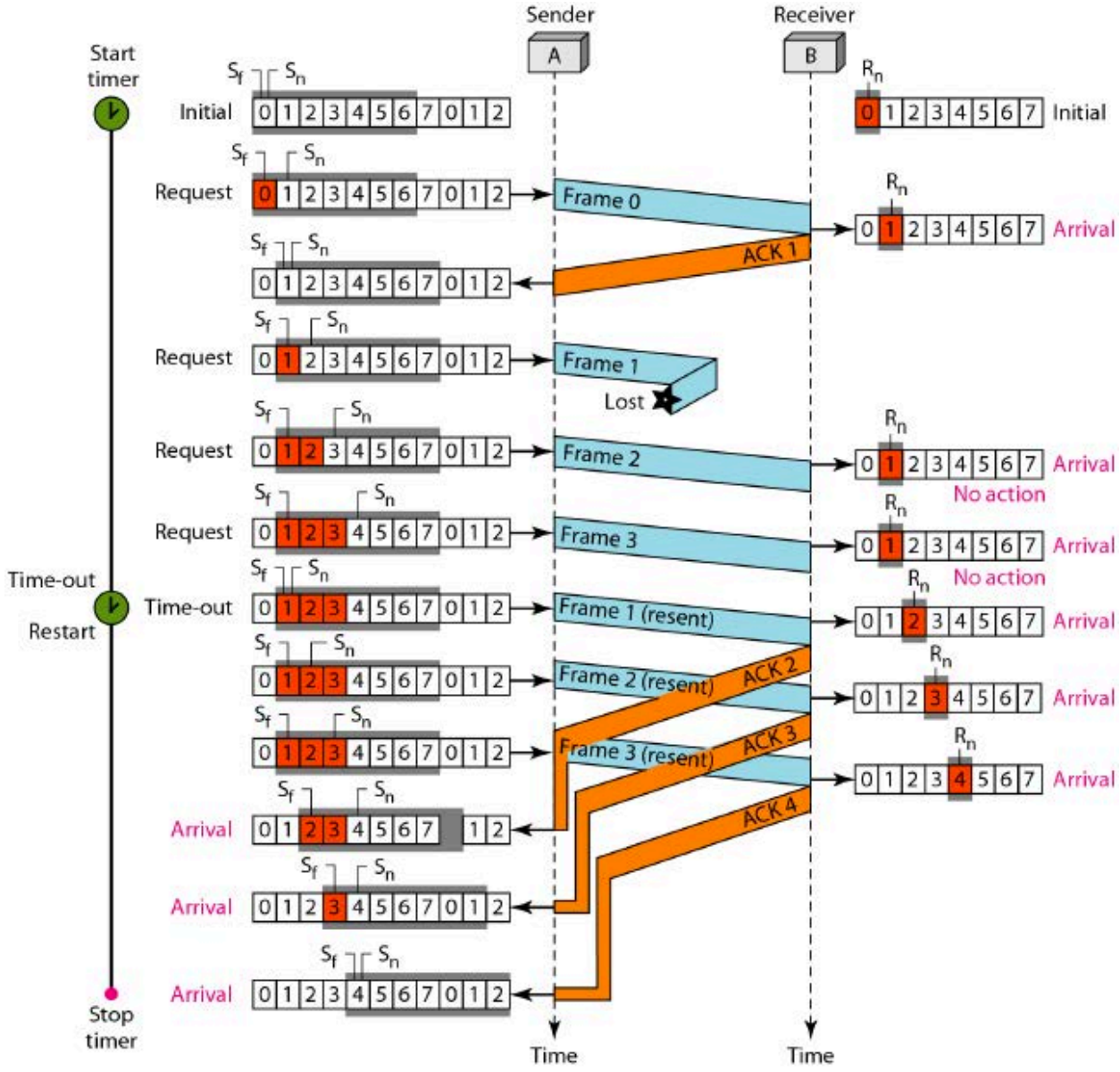
Window Size in Go-Back-N ARQ



Summary Notes for Go-Back-N ARQ

- In the Go-Back-N Protocol, the sequence numbers are modulo 2^m, where m is the size of the sequence number field in bits.
- The send window can slide one or more slots when a valid acknowledgment arrives.
- In Go-Back-N ARQ, the size of the send window must be less than 2^m; the size of the receiver window is always 1.
- The receive window of size 1. The window slides when a frame with no detected errors arrive; i.e. sliding occurs one slot at a time. Receiver will drop any out-of-order frames

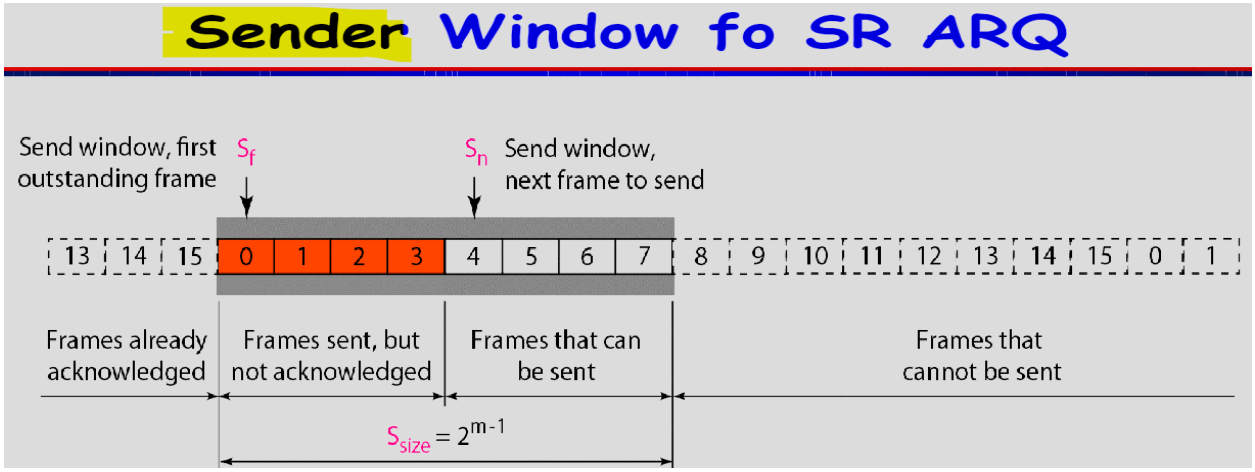
Example: Un-reliable Channel

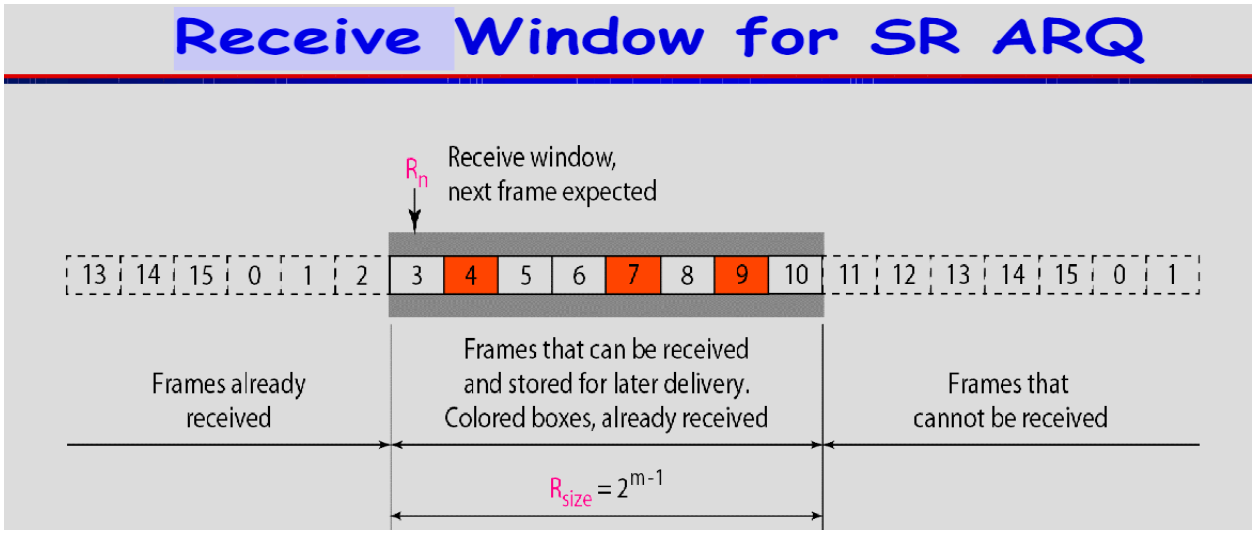


Selective Repeat (Reject) ARQ

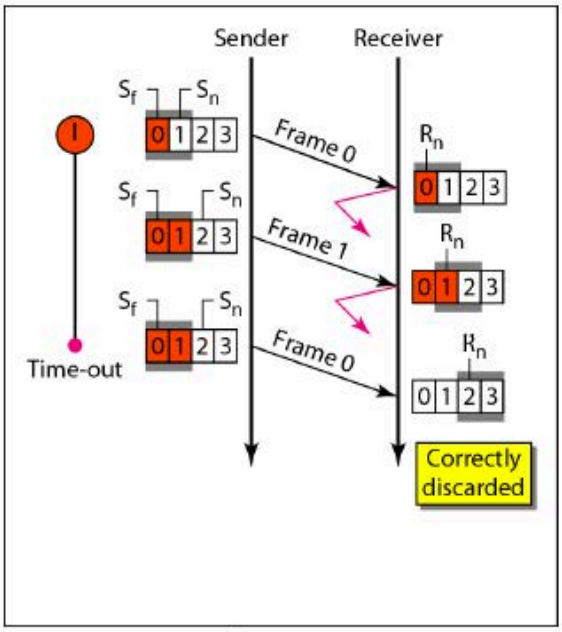
- Only rejected frames are retransmitted
- Subsequent frames are accepted by the receiver and buffered
- Minimizes retransmission
- Receiver must maintain large enough buffer
- More complex transmitter

Window fo SR ARQ

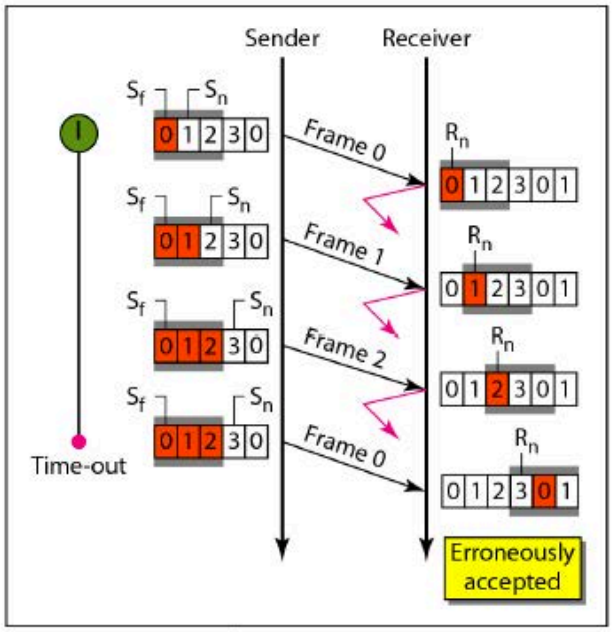




SR ARQ Window Size



a. Window size = 2^{m-1}

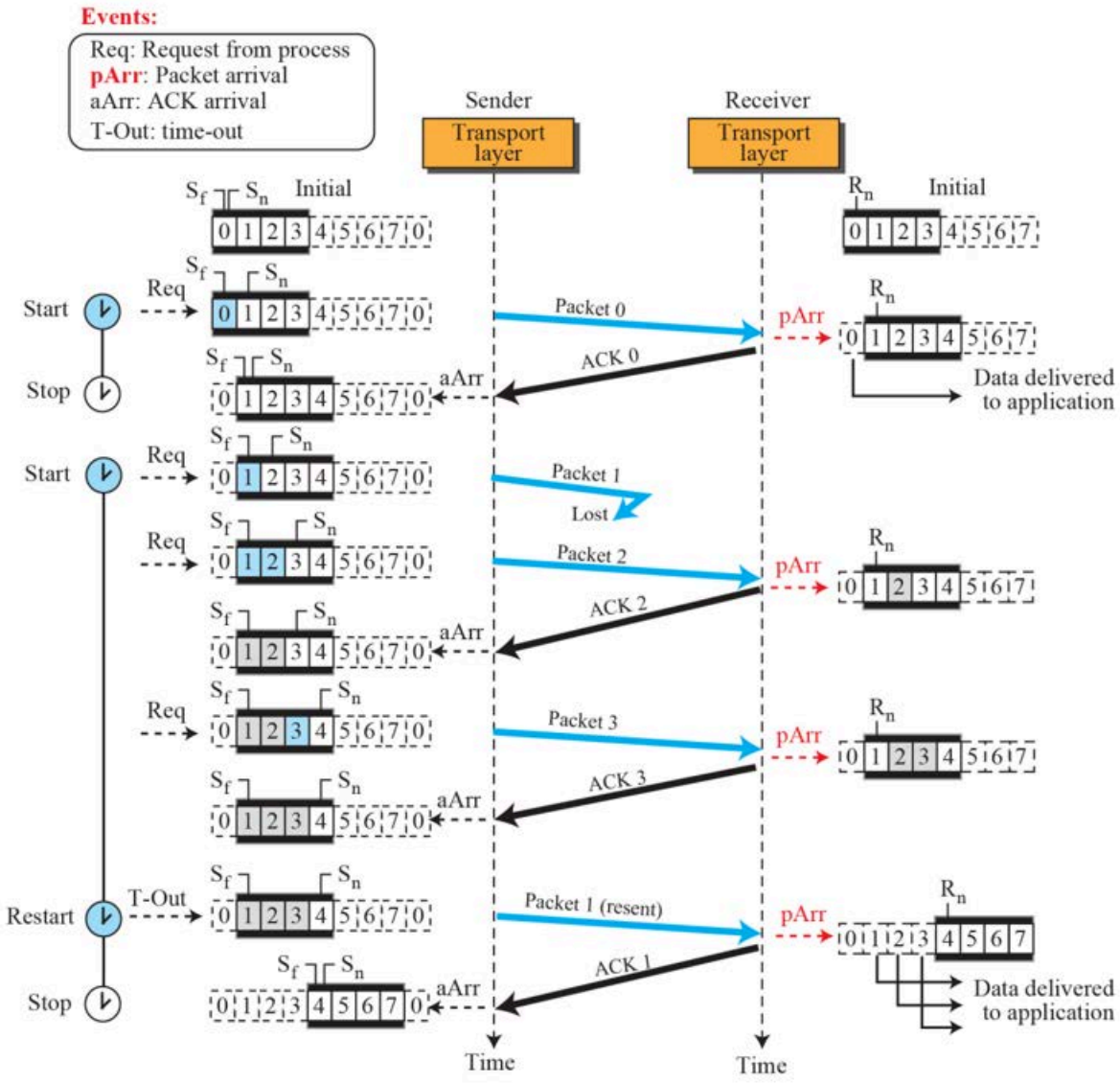


b. Window size > 2^{m-1}

Summary Notes for SR ARQ

- In Selective Repeat ARQ, the size of the send window must be at most 2^{m-1}; the size of the receiver window is usually (but not necessarily) the same as that of the sender window. Receiver will buffer any out-of-order frames
- Receiver can acknowledge frames that are out of sequence but it can't pass them to the network layer. The sequence number of the ACK corresponds to the sequence number of the frame being acknowledged

Example of SR ARQ



===Final===

9.29 MAC (Medium Access Control)

Medium Access Control

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

- **Multiple Access Protocol**
    - Distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
    - Communication about channel sharing must use channel itself!

Multiple Access Protocol（多路访问协议） 被认为是分布式算法的一种，主要是因为它解决了在多个节点共享通信信道的分布式系统中协调资源访问的问题。

为什么它属于分布式算法？

节点间没有中心控制

- 分布式特性:
  多路访问协议运行在多个独立节点上，这些节点相互平等，没有中心协调者来决定哪个节点可以使用信道。节点必须自主决策如何访问共享信道。
- 举例:
    - 在以太网的 **CSMA/CD**（载波侦听多路访问/冲突检测） 中，节点通过监听信道状态自行决定是否可以发送数据，没有中央控制器。

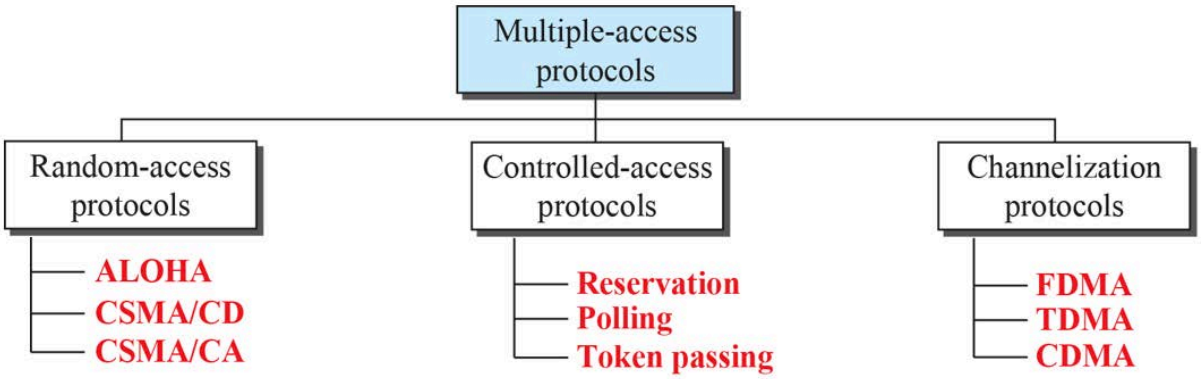
协调机制通过局部信息实现

- 分布式算法的关键特征:
  节点的决策基于局部信息（如信道状态、历史冲突记录），不需要全局信息。
- 举例:
    - 在 Wi-Fi 的 **CSMA/CA**（载波侦听多路访问/冲突避免）中，每个节点通过检测信道是否空闲来决定发送数据，无需了解其他节点的具体状态。

解决共享资源的访问冲突

- 多路访问协议通过算法解决分布式系统中资源共享的问题:
    - 共享的通信信道是一种稀缺资源。
    - 如果多个节点同时使用，会发生冲突（如数据包碰撞）。
- 分布式算法的本质: 协调多个节点在共享资源上的操作，避免冲突并优化性能。

Taxonomy of MAC Protocols



Three Broad Classes of MAC Protocols

1. ***Channel Partitioning***
    - Divide channel into smaller “pieces” (time slots, frequency, code)
        - for example TDMA, FDMA or CDMA
    - Allocate piece to node for exclusive use
2. ***Random Access***
    - Channel not divided, allow collisions (contention)
        - Examples: ALOHA, CSMA/CD, CSMA/CA
    - “Recover” from collisions
        - for example：via delayed retransmissions
3. “***Taking turns***” (or ***Round Robin***)
    - Tightly coordinate shared access to avoid collisions
    - Nodes take turns, but nodes with more to send can take longer turns.
        - Examples: Polling, Token Passing

CSMA (Carrier Sense Multiple Access)

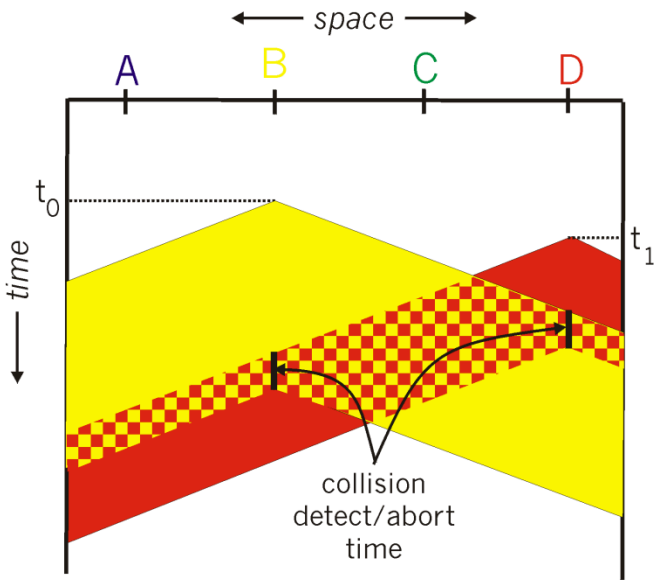
- **CSMA/CD**: Carrier sense, multiple access with collision detection
    - collisions detected within short time
    - colliding transmissions aborted, reducing waste
    - Persistent, non-persistent and P-persistent retransmission
- **Collision Detection**:
    - On baseband bus, collision produces much higher signal voltage than transmitted signal
    - For twisted pair (Hub-topology) activity on more than one port is collision

Ethernet CSMA/CD algorithm

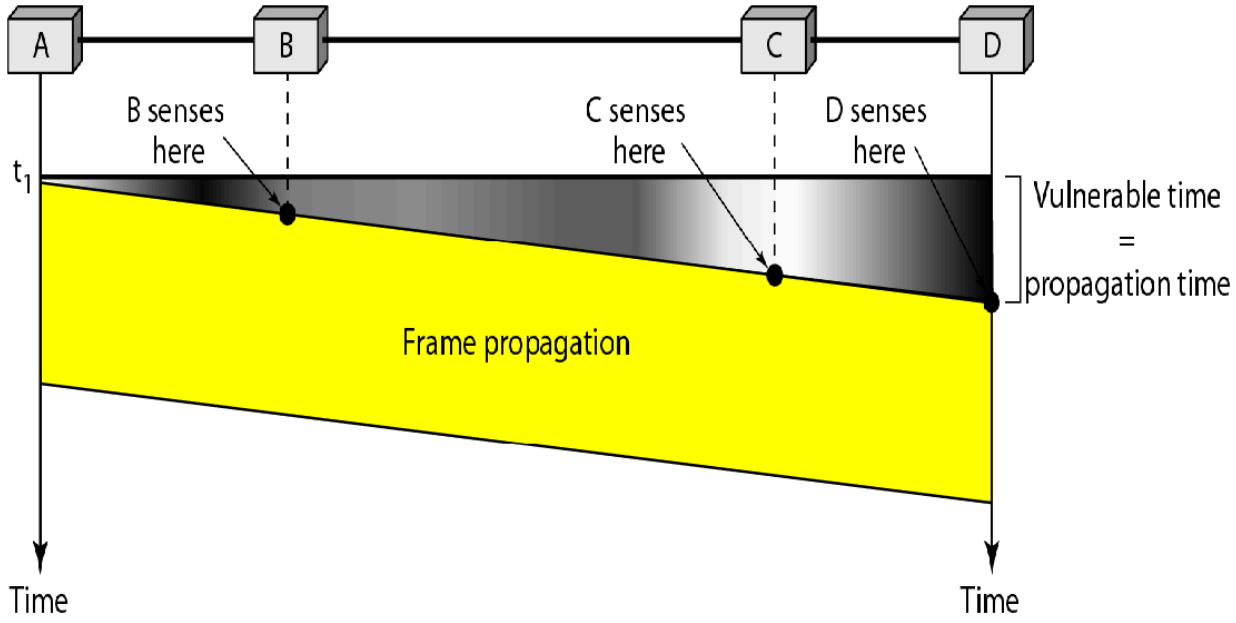
1. NIC receives datagram from network layer, creates frame
2. If NIC senses channel idle, starts frame transmission. If NIC senses channel busy, waits until channel idle, then transmits.
3. If NIC transmits entire frame without detecting another transmission, NIC is done with frame !
4. If NIC detects another transmission while transmitting, aborts and sends jam signal
5. After aborting, NIC enters *binary (exponential) backoff:*
   - after *m*th collision, NIC chooses *K* at random from $\{0, 1, 2, \ldots, 2^m-1\}$. NIC waits K·512 bit times, returns to Step 2
   - longer backoff interval with more collisions

Collisions in CSMA/CD

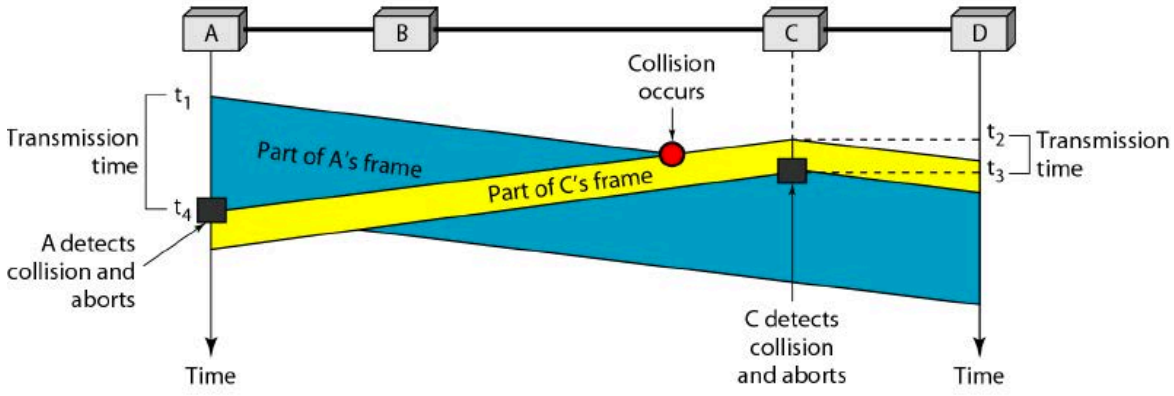
- **Collisions can still occur**: propagation delay means two nodes may not hear each other's transmission
- When collision occur, entire frame is wasted
- Collision is detected by comparing transmitted and received signal strengths (Hard to do in WLANs, TBD)



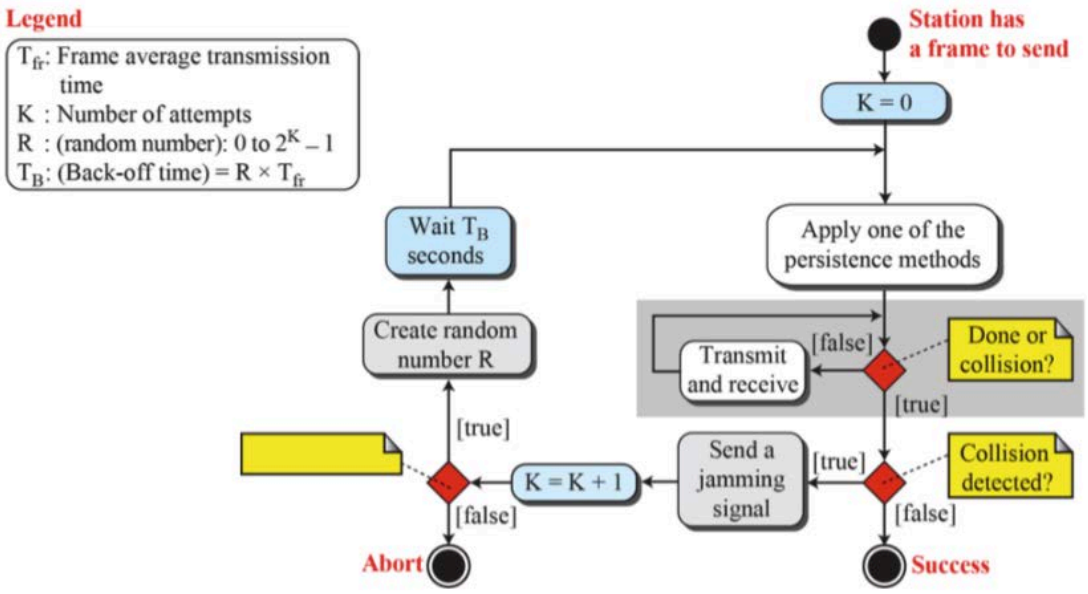
Vulnerable Time in CSMA



Collision Detection



Flow Chart of CSMA/CD



现在，我们根据流程图简单看一下流程。

1. 准备帧传输。

   首先，准备一个用于传输的帧。而在这个过程中，代表“尝试次数”的K被初始化为0。
   然而，您不应该关注数字 0，而应该关注包含“尝试次数”的概念，该概念类似于 IP 标头的 TTL。

2. 检测载波，检查是否空闲，然后发送。

   这部分对应于“载波侦听”。它检测载波并确定线路是否正在使用。我很好奇如何确定这一点，所以我做了一些研究，发现可以通过有线 LAN（以太网）中的电压变化轻松测量它。

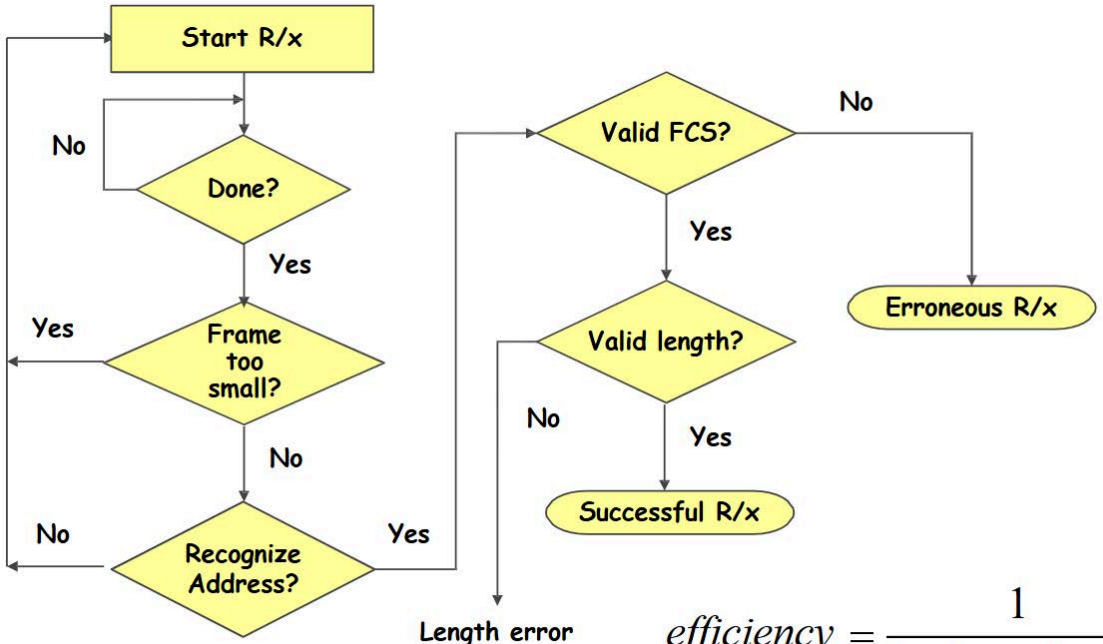
   然而，即使线路未使用，它也并不总是立即传输，并且它根据所使用的持续方法而变化。简单来说，持久化方法是一种如果判断线路为空则决定何时传输的算法。

3. 发送完毕后，检查是否发生冲突，如果发生则重新发送。

   发送一帧后，检查是否发生冲突。如果发生冲突，则发送干扰信号并且增加K值。如果K值超过一定值，则中止。否则，它使用退避算法等待一段随机时间，然后等待重传。

   这里，据说可以通过电压的变化容易地检测到冲突检测，这与确定介质是否处于空闲状态相同。

Receive Process in IEEE802.3



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

★ $efficiency = 1 / (1 + 5 * t_{prop}/t_{trans})$

CSMA/CD efficiency

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

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

- efficiency goes to 1
  - as t_{prop} goes to 0
  - as t_{trans} goes to infinity
- better performance than ALOHA: and simple, cheap, decentralized!

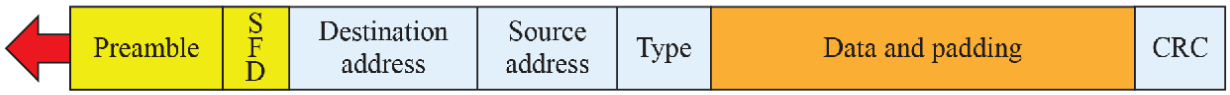
IEEE802.3 MAC Frame

Sending adapter encapsulates IP Packet in Ethernet frame

Minimum payload length: 46 bytes
Maximum payload length: 1500 bytes

Preamble: 56 bits of alternating 1s and 0s

SFD: Start frame delimiter, flag (10101011)



Physical-layer header

Minimum frame length: 512 bits or 64 bytes
Maximum frame length: 12,144 bits or 1518 bytes

06:01:02:01:2C:4B

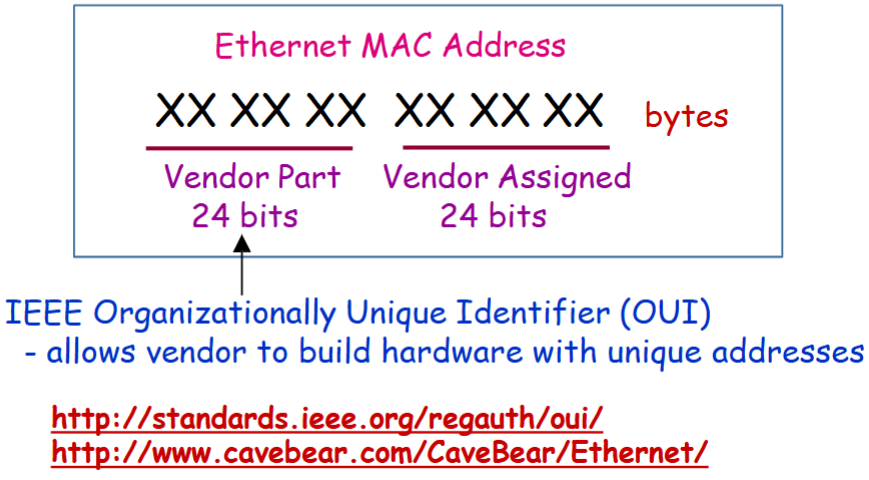
6 bytes = 12 hex digits = 48 bits

MAC address is burned in NIC ROM
Type: Indicate Network Layer Protocol (mostly IP)
A NIC card will process a frame if it recognizes its MAC address or Broadcast Address, Otherwise discards the frame

MAC Addresses

- Source and destination MAC addresses.
    - These are the hardware addresses.
    - They are 48-bits long each

- Source and destination MAC addresses. These are the hardware addresses. They are 48-bits long each



http://standards.ieee.org/regauth/oui/
http://www.cavebear.com/CaveBear/Ethernet/

Types of MAC Addresses

- ***Unicast***: one interface to one interface
- ***Broadcast***: all 1's destination address
    - means that every attached interface to a LAN should read the frame.
    - MAC Address: FF:FF:FF:FF:FF:FF
- ***Multicast***: an interface can be configured to read frames sent to **one or more** *multicast* addresses.

Multicast 多播地址的特征

- 第一个字节最低位为 1（01-xx-xx-xx-xx-xx）
- 区别于单播地址（最低位为 0）
- 区别于广播地址（FF-FF-FF-FF-FF-FF）

多播地址的生成规则（以 IPv4 多播为例）

- 前 25 位固定为 01:00:5E
- 第 25 位必须为 0
- 最后 23 位来自 IP 多播地址的后 23 位

传输方式的对比

- 单播：一对一，需要多次发送
- 广播：一对所有，网段内所有设备都收到
- 多播：一对多，只有订阅的设备收到

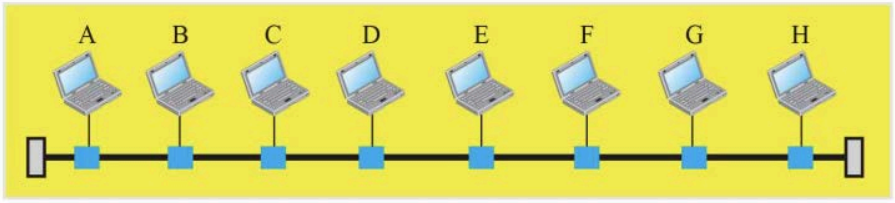
工作机制

发送方：
- 将数据发送到多播MAC地址
- 一次发送，多处接收

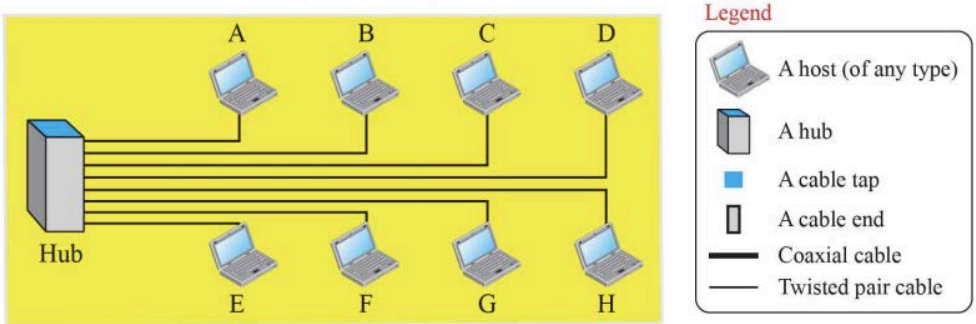
接收方：
- 网卡配置接收特定多播地址
- 硬件过滤，只接收订阅的多播地址
- 可以同时订阅多个多播地址

Shared Ethernet Implementations

- 以太网LAN主要可以通过两种方式实现：总线型拓扑 (Bus) 和星型链路型拓扑 (Star)

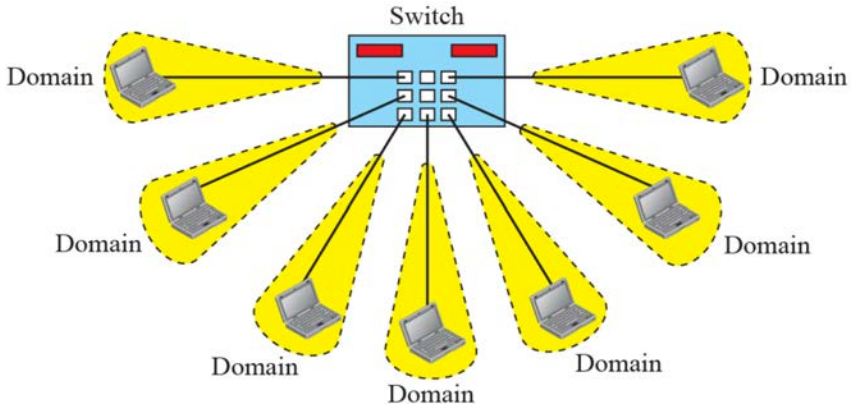


a. A LAN with a bus topology using a coaxial cable

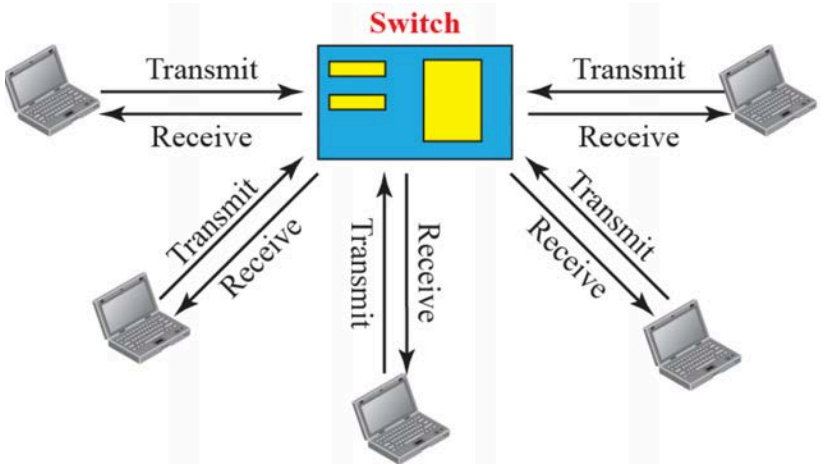


b. A LAN with a star topology using a hub

- 交换以太网
  - 交换机隔离流量



- 全双工以太网 (Support FDX)
  - 无需 CSMA/CD→这是因为数据发送和接收是同时发生的。

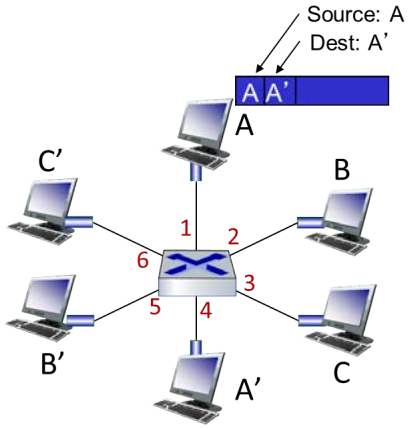


Ethernet switch

- **link-layer device: takes an active role**
  - store, forward Ethernet frames
  - examine incoming frame's MAC address,
    selectively forward frame to one-or-more outgoing links
    when frame is to be forwarded on segment, uses CSMA/CD to access segment
- **Transparent**
  - hosts are unaware of presence of switches
- **plug-and-play, self-learning**
  - switches do not need to be configured

Switch: *self-learning*

- switch learns which hosts can be reached through which interfaces
  - when frame received, switch "learns" location of sender: incoming LAN segment
  - records sender/location pair in switch table



MAC addr	interface	TTL
A	*1*	*60*

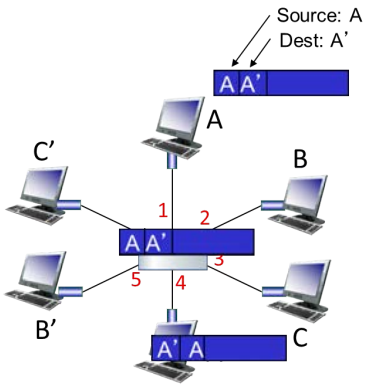
Switch table (initially empty)

Switch: frame filtering/forwarding

- when frame received at switch:
  1. record incoming link, MAC address of sending host
  2. index switch table using MAC destination address
  3. if entry found for destination
     then {
     if destination on segment from which frame arrived
     then drop frame
     else forward frame on interface indicated by entry
     }
     else flood /* forward on all interfaces except arriving interface */

Example: Self-learning, forwarding

- frame destination, A', location unknown: **flood**
- destination A location known: **selectively send on just one link**

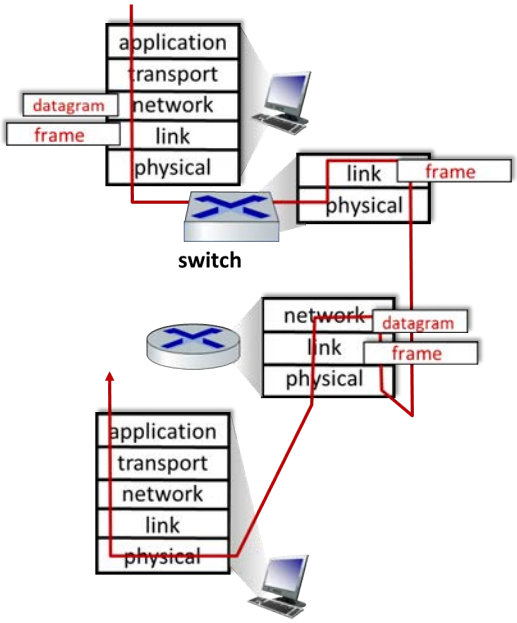


MAC addr	interface	TTL
A	*1*	*60*
A'	*4*	*60*

switch table (initially empty)

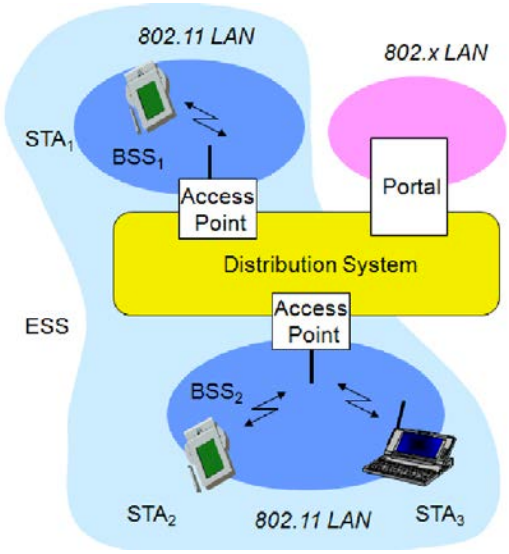
Switches vs. routers

- both are store-and-forward:
  - *routers*: network-layer devices (examine network-layer headers)
  - *switches:* link-layer devices (examine link-layer headers)
- both have forwarding tables:
  - *routers:* compute tables using routing algorithms, IP addresses
  - *switches:* learn forwarding table using flooding, learning, MAC addresses



802.11 Infrastructure Network

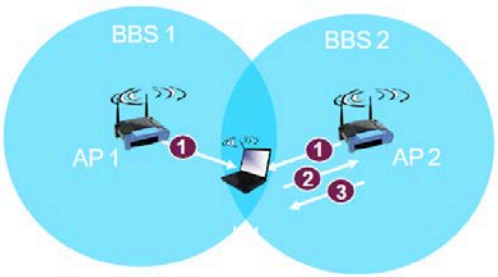
- ***Station*** (***STA***)
  - terminal with access mechanisms to the wireless medium and radio contact to the access point
- ***Basic Service Set*** (***BSS***)
  - group of stations using the same radio frequency
- ***Access Point***
  - station integrated into the wireless LAN and the distribution System
- ***Portal*** (***Bridge***/***Router***)
  - to other (wired) networks
- ***Distribution System***
  - interconnection network to form one logical network (**EES**: Extended Service Set) based on several BSS



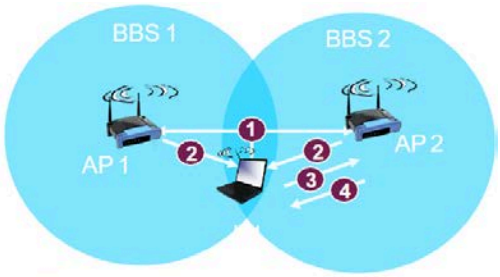
Channel Association

- 802.11b: 2.4GHz-2.485GHz spectrum divided into 11 channels at different frequencies
    - AP admin chooses frequency for AP
    - Interference possible: channel can be same as that chosen by neighboring AP!
- Host: must associate with an AP
    - Scans channels, listening for **Beacon frames** containing AP's name (SSID) and MAC address
    - Selects AP to associate with
    - May perform authentication
    - Run DHCP to get IP address in AP's subnet

802.11: Passive/Active scanning



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



active scanning:
(1) Probe Request frame broadcast from H1
(2) Probe Response frames sent from APs
(3) Association Request frame sent: H1 to selected AP
(4) Association Response frame sent from selected AP to H1

IEEE802.11 MAC Protocol

- Avoid collisions: 2+ nodes transmitting at same time
- **802.11: CSMA** - sense before transmitting
    - Don't collide with other transmissions
- 802.11: No collision detection!
    - Difficult to receive (sense collisions) when transmitting due to weak received signals (fading)
    - Can't sense all collisions in any case: hidden terminal, fading
    - Goal: avoid collisions:
        - CSMA/C(ollision)A(voidance)

802.11 MAC Procedures

- **Traffic services**
  - Asynchronous Data Service (mandatory) – ***DCF***
  - Time-Bounded Service (optional) - ***PCF***
- **Access methods**
  - DCF CSMA/CA (mandatory)
    - collision avoidance via randomized back-off mechanism
    - ACKs for data frames (not for broadcasts)
  - DCF w/ RTS/CTS (optional)
    - avoids hidden terminal problem
  - PCF (optional)
    - access point polls terminals according to a list

DCF (Distributed Coordination Function，分布式协调功能)

- 基本的访问方法，必选实现
- 使用 CSMA/CA 机制
- 完全分布式，没有中央控制
- 工作过程
  1. 想发送数据时先监听信道
  2. 如果信道空闲，等待 DIFS 时间
  3. 如果还是空闲，开始随机退避计数
  4. 计数到0且信道仍空闲，才发送数据
  5. 接收方收到后等待 SIFS 时间发送 ACK

PCF (Point Coordination Function，点协调功能)

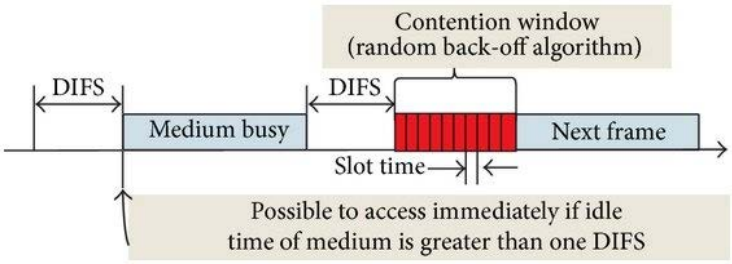
- 可选的访问方法，建立在 DCF 之上
- 需要接入点(AP)作为中央控制器
- 使用轮询机制
- 工作过程：
  1. AP 发送 Beacon 帧开始免竞争周期
  2. AP 轮询每个注册的站点
  3. 被轮询的站点获得发送权
  4. 发送完成或超时，AP 轮询下一个站点

主要区别：

1. 控制方式：DCF 分布式 vs PCF 集中式
2. 实现要求：DCF 必选 vs PCF 可选
3. 时延保证：DCF 不确定 vs PCF 可预测
4. 使用场景：DCF 适合普通数据 vs PCF 适合实时数据

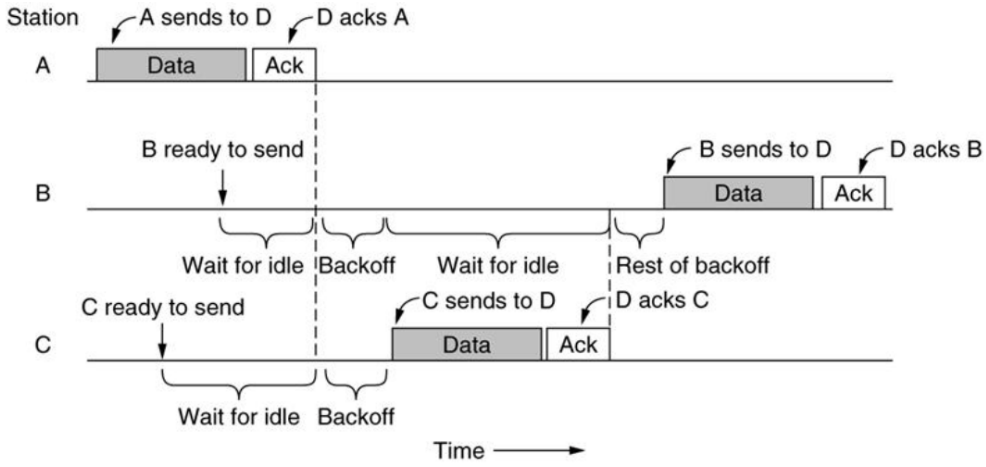
现在很多设备主要使用 DCF，因为它更简单且适应性更强。PCF 虽然能提供更好的服务质量保证，但实现复杂度高，使用较少

DCF: CSMA/CA



- station ready to send starts sensing the medium (Carrier Sense based on CCA, Clear Channel Assessment)
- if the medium is free for the duration of an Inter-Frame Space (IFS), the station can start sending (IFS depends on service type)
- if the medium is busy, the station has to wait for a free IFS, then the station must **additionally** wait a random back-off time (collision avoidance, multiple of slot-time)
- if another station occupies the medium during the back-off time of the station, the back-off timer stops (fairness)

Sending a Frame in CSMA/CA

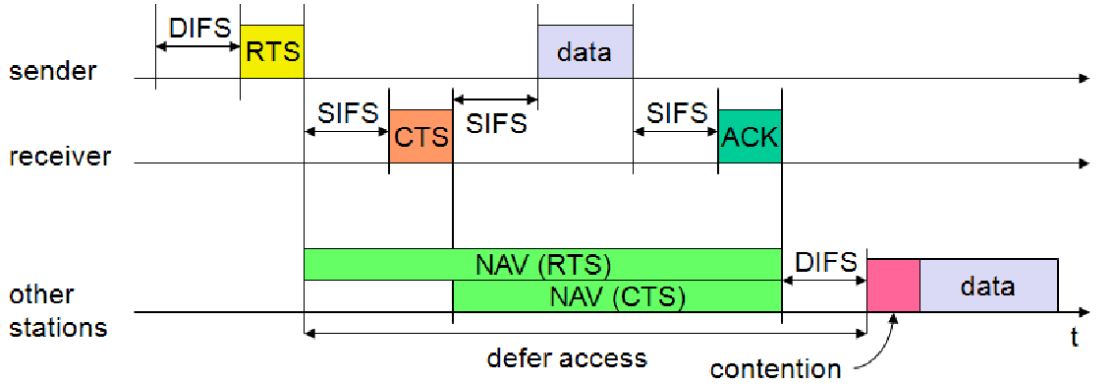


Avoiding Collisions (***RTS, CTS***)

- Idea: allow sender to “reserve” channel rather than random access of data frames: avoid collisions of long data frames
- Sender first transmits small request-to-send (***RTS***) frames to BS using CSMA
  - RTSs may ***still collide*** with each other (but they’re short)
- BS broadcasts clear-to-send ***CTS*** in response to ***RTS***
- ***CTS*** heard by all nodes
  - Sender transmits data frame
  - Other stations defer transmissions

avoid data frame collisions completely using small reservation packets!

DCF w/RTS & CTS



- Station send RTS with reservation parameter (amount of time the data frame needs the medium) after waiting for DIFS
- Acknowledgement via CTS after SIFS by receiver (if ready to receive)
- Sender can now send data at once, acknowledgement via ACK
- Other stations store medium reservations distributed via RTS and CTS

⭐ **SIFS** vs. **DIFS**

- **SIFS** (短帧间间隔, Short Interframe Space)
  - ***Purpose***: Provides **highest priority** for time-sensitive transmissions like acknowledgments (ACK), CTS (Clear to Send), or data fragments in a burst.
  - ***Duration***: **Shorter than DIFS** to ensure immediate access to the channel for high-priority frames.
- **DIFS** (DCF 帧间间隔, DCF Interframe Space)
  - ***Purpose***: Used by stations to initiate new data transmissions after sensing that the medium has been idle for a sufficient time.
  - ***Duration***: **Longer than SIFS**, ensuring priority for ongoing transmissions (like ACKs or control frames) before allowing new data to access the medium.

工作流程示例：

数据传输：

```
发送方 ----[等待 DIFS]----> 数据帧 ---->
接收方 --------------------|等待 SIFS|---> ACK
```

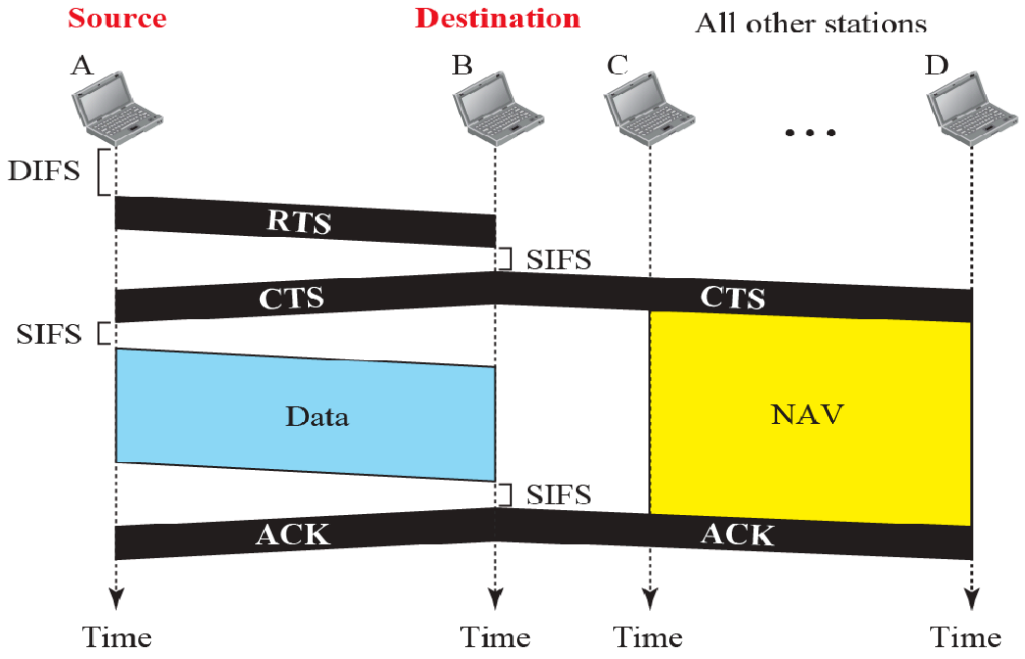
为什么需要不同的间隔？

- 优先级区分：重要的控制帧（如 ACK）用 SIFS，确保可以优先发送
- 避免冲突：DIFS 较长，给其他站点更多时间检测到传输
- 效率考虑：对于需要快速响应的帧使用 SIFS，减少延迟

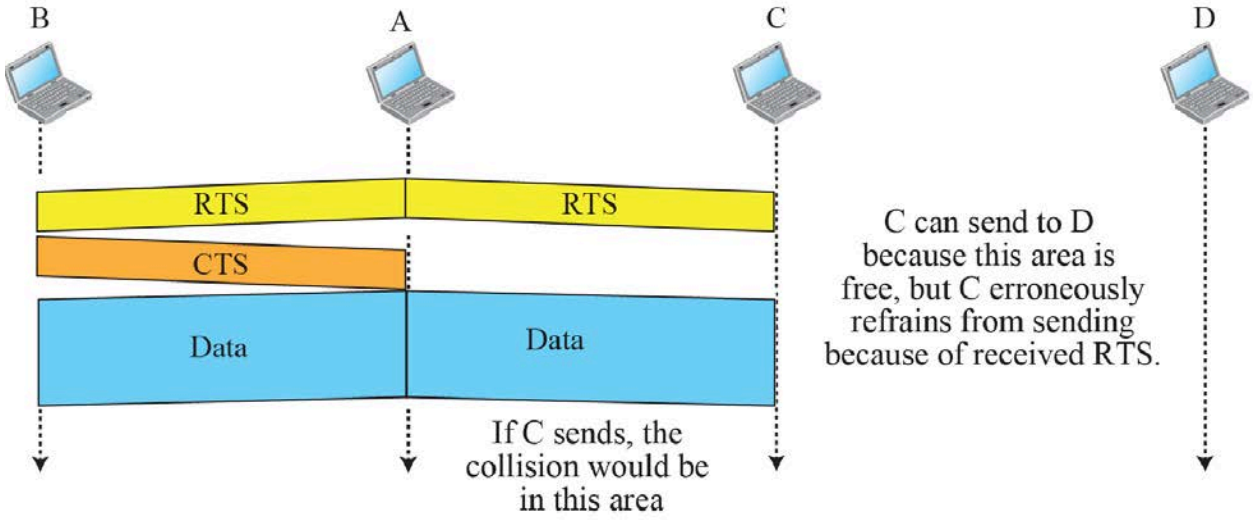
IFS (Interframe Space):

- An IFS is a mandatory waiting period between frames to avoid collisions and prioritize specific types of traffic.
- Different IFS durations help establish priority among various types of transmissions.

Collision Avoidance using RTS/CTS



Exposed Terminal Problem

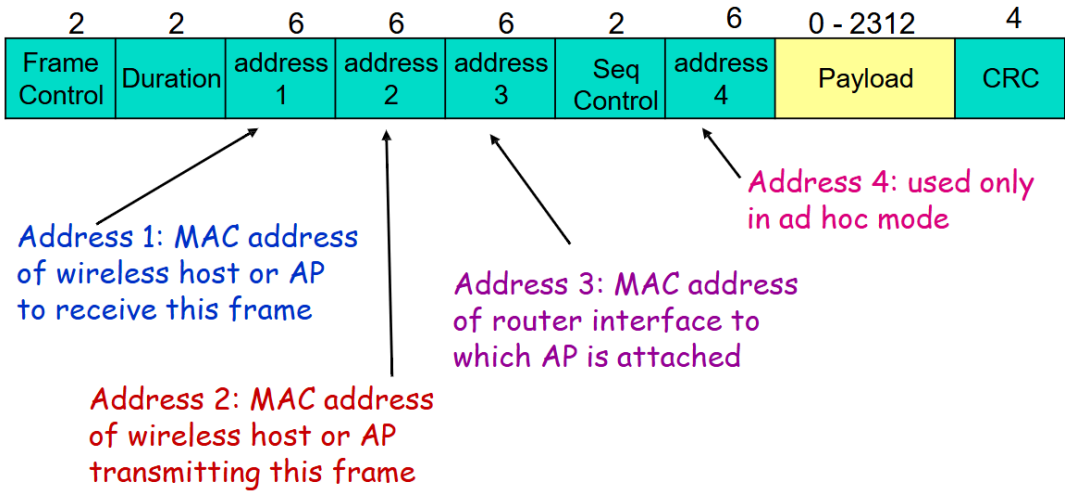


暴露站问题是用于解决隐藏站问题的RTS和CTS干扰具有不同交换方法的站之间的通信的问题。

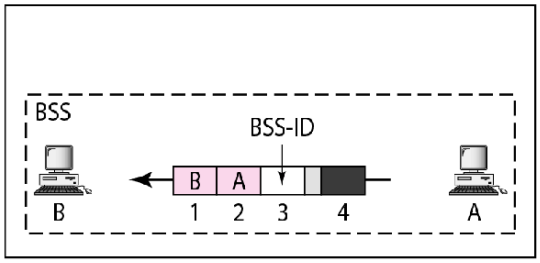
如果A发送了RTS向B发送数据，C也收到了该RTS。
C 想要向 D 发送数据，但 C 由于 A 的信号而认为通道繁忙，从而导致问题。

⭐ How to fix？ → Not specified in lecture

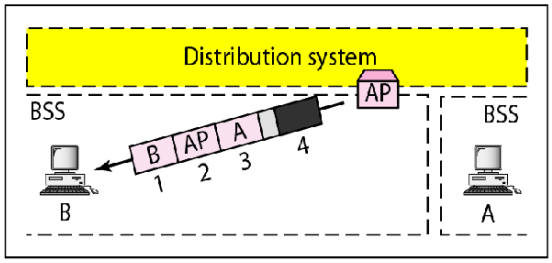
EEE802.11 Frame Structure



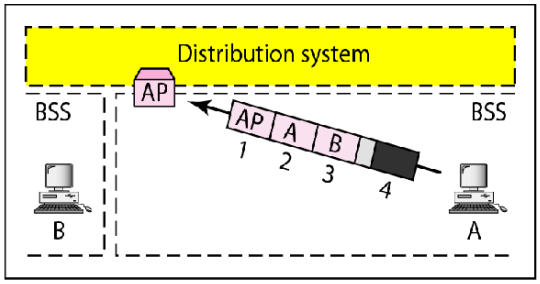
Addressing Mechanisms



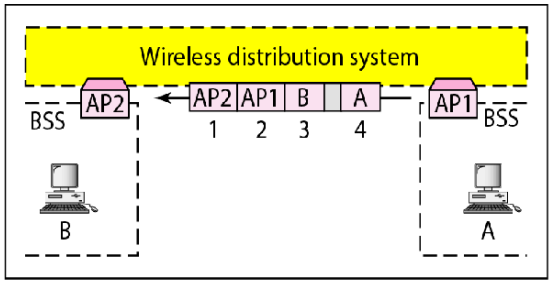
a. Case 1



b. Case 2



c. Case 3



d. Case 4

1) 方案一：A和B 在同一个IBSS，A->B (Ad hoc无线自组网中的数据帧的地址格式)。

2) 方案二：从AP发出的无线数据帧中的地址格式。

3) 方案三：发到AP的无线数据帧中的地址格式。

4) 方案四：通过无线分布系统传输的无线数据帧中的地址格式。

10.17 IP (Internet Protocol) “The” Network Layer

Network-layer services and protocols

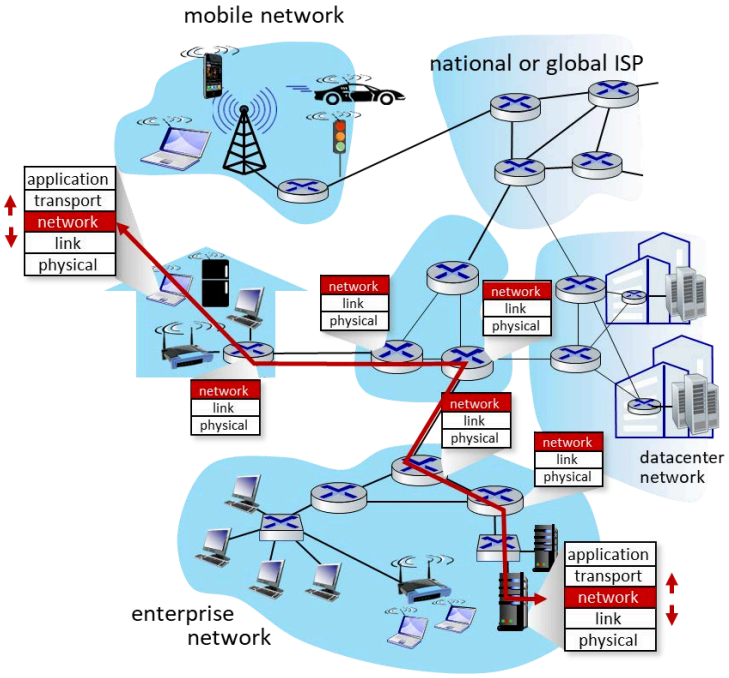
Transport segment from sending to receiving host

- ***sender***: encapsulates TCP segments (or UDP Datagrams) into **Packets** passes to link layer
- ***receiver***: delivers segments to transport layer protocol

Network layer protocols in ***every*** *Internet device*: hosts, routers

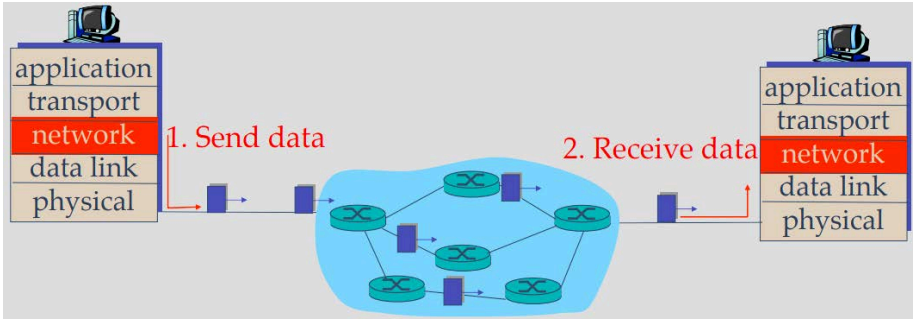
Routers:

- examines header fields in all IP datagrams passing through it
- moves datagrams from input ports to output ports to transfer datagrams along end-end path

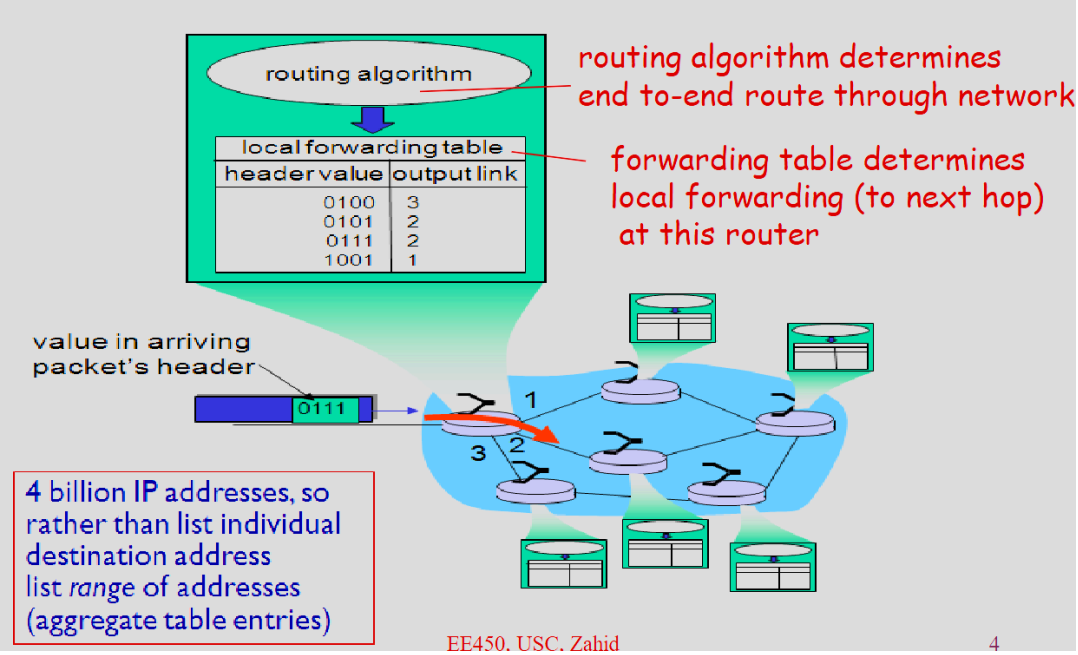


Packet Switched Network (Review)

- No call setup at Network layer
- Routers: no state about end-to-end connections
    - Packets may arrive out-of-order
- Packets forwarded using destination IP address



Routing vs. Forwarding



network-layer functions:

- *forwarding:* move packets from a router's input link to appropriate router output link
- *routing:* determine route taken by packets from source to destination
    - *routing algorithms*

analogy: taking a trip

- *forwarding:* process of getting through single interchange
- *routing:* process of planning trip from source to destination

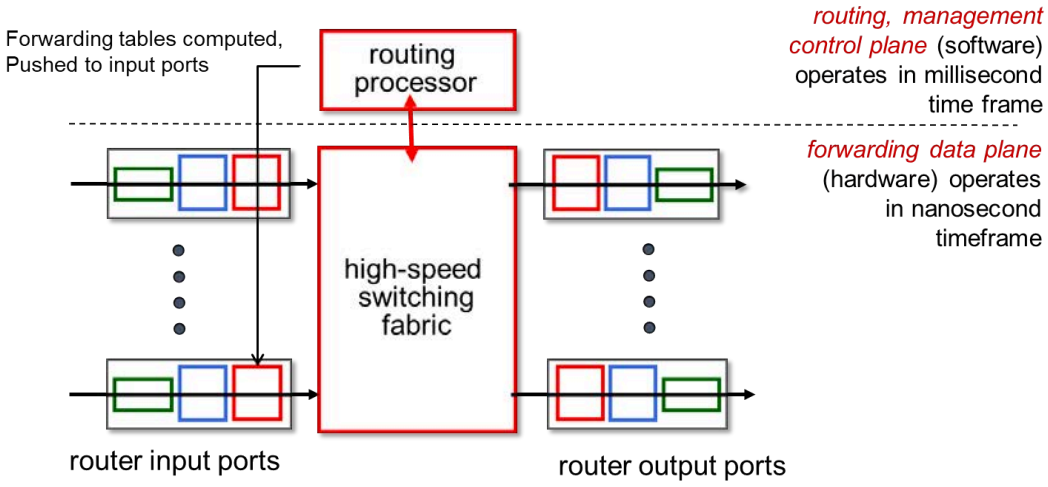
forwarding

routing

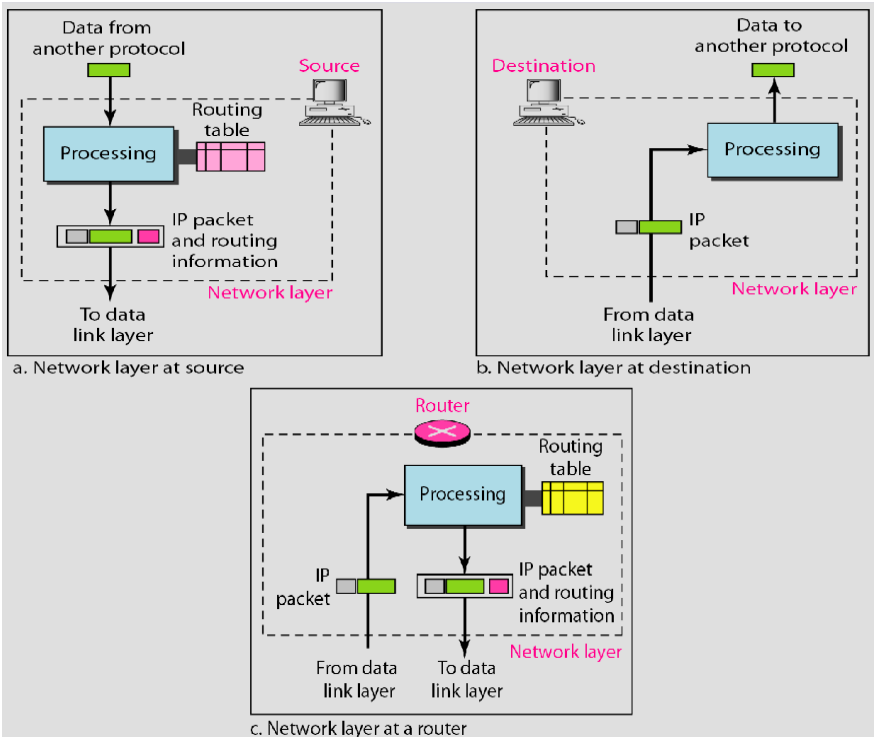
Router Architecture

two key router functions:

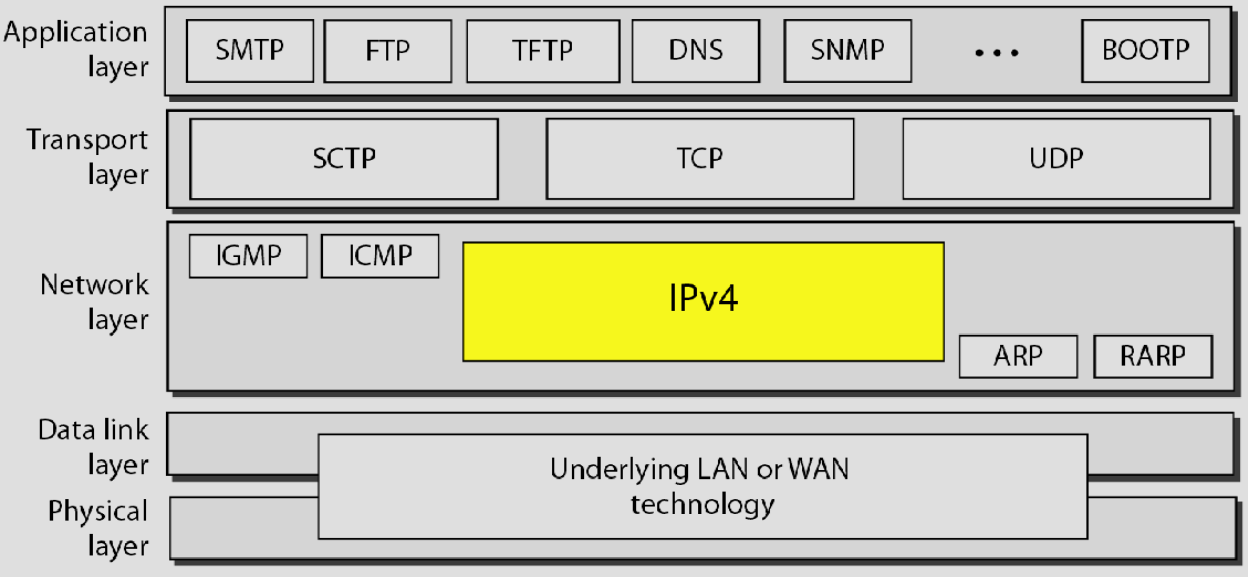
1. Run Routing algorithms/protocols (RIP, OSPF, BGP)
2. forwarding packets from incoming to outgoing link



Network Layer at Source/Router/Destination



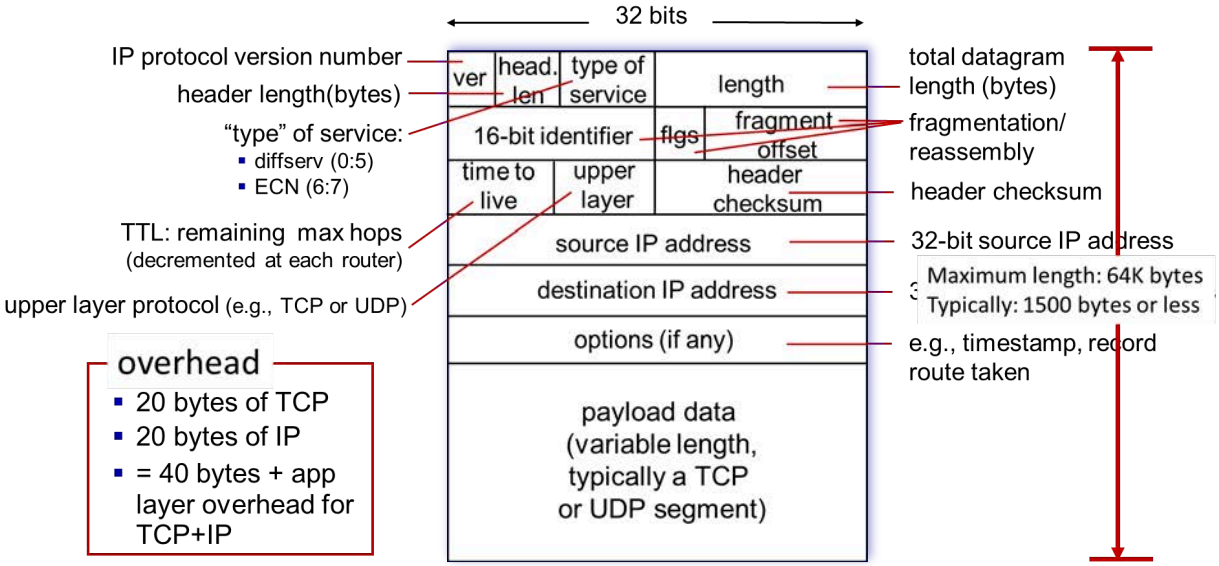
The IP Protocol



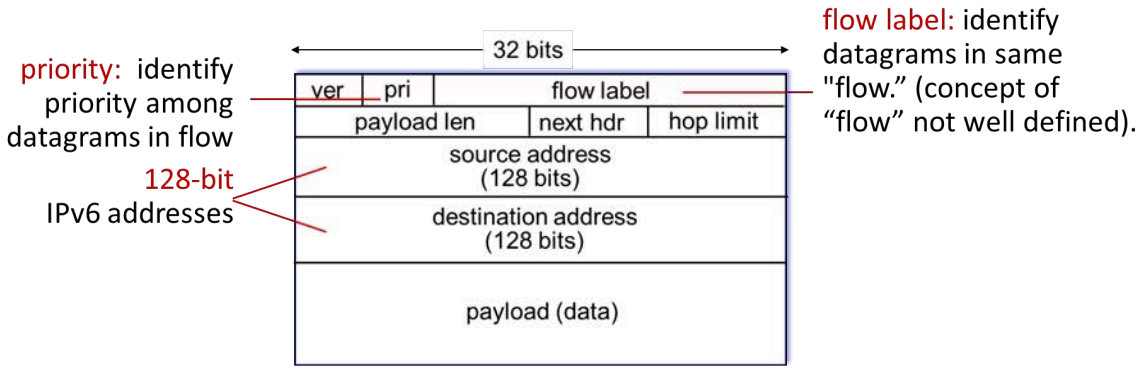
The Internet Protocol

- IP is a connection-less, unreliable network layer protocol designed to be used in a connection-less packet switched network such as the Internet.
- IP provides best effort services in the sense
    - There is no guarantee of ***delivery of error-free*** packets
    - There is no guarantee of ***ordered delivery*** of packets
    - There is no guarantee of ***delivery of packets***,
        - i.e. some packets may be lost, some packets may be duplicated
- IP relies on upper layer transport protocols (TCP) to take care of these problems

IP Packet Format



IPv6 format



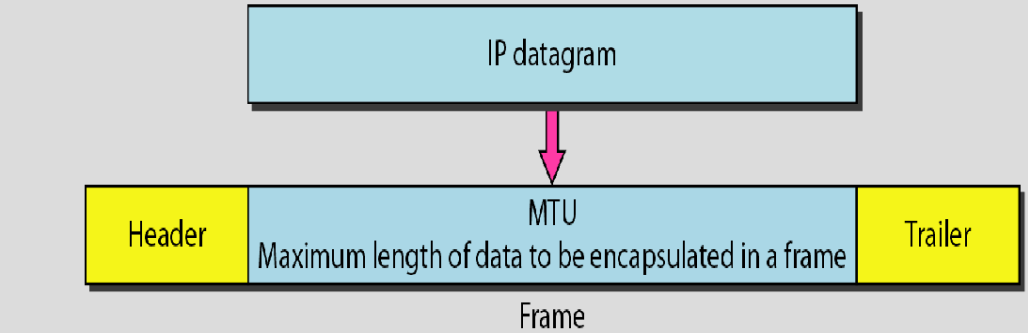
What's missing (compared with IPv4):

- no checksum (to speed processing at routers)
- no fragmentation/reassembly
- no options (available as upper-layer, next-header protocol at router)

Fragmentation

- IP packet may travel over different networks (LANs and WANs)
- A router de-capsulate an IP packet from the frame it receives, process it, and encapsulate it in another frame
- Frame size and format varies depend on the data link protocol used by the physical network through which the frame is traveling
- **MTU** (**Maximum Transmission Unit**) is the maximum size of the **data field** (**payload**) in the ***frame*** → **MTU 为 L2 datalink 层的概念**
- If Packet size > MTU, Need for Fragmentation

MTU: Maximum Transfer Unit



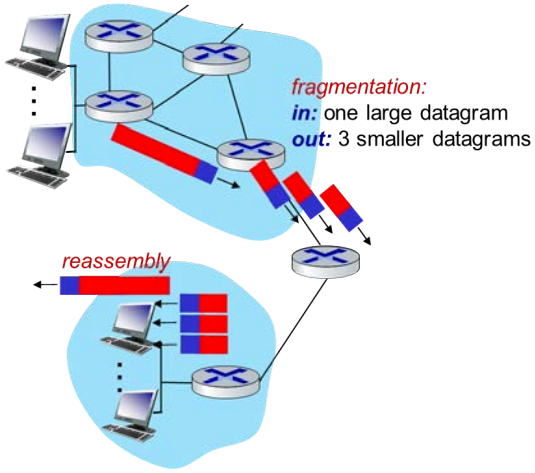
Protocol	MTU (Octets)
Ethernet	1500
Token Ring (4 Mbps)	4464
Token Ring (16 Mbps)	17914
FDDI	4352
X.25	576
PPP	296

Fragmentation (Continued)

- Each fragment has its own header (most of fields are copied, some will change, including the total length, the Flags and the fragmentation offset fields)
- A fragmented datagram may itself be fragmented if it encounters a network with smaller MTU
- A packet can be fragmented by a source host or by any router in the path. Re-assembly of the packet must be done at the destination host because those fragments become independent packets and may travel different routes

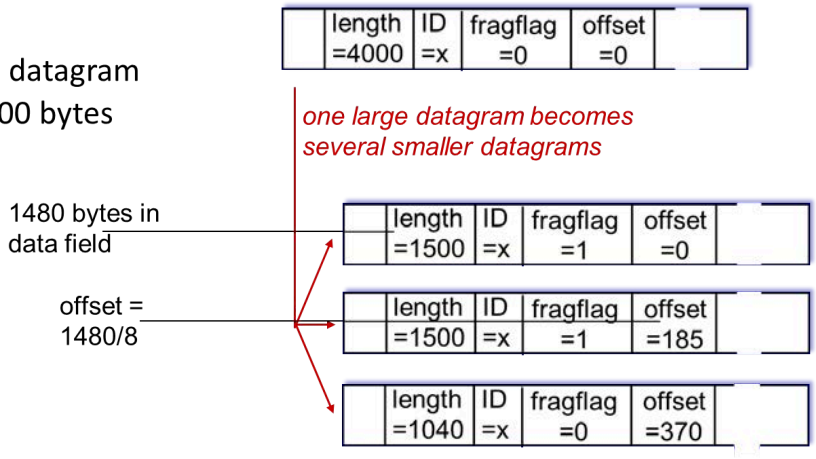
IP fragmentation/reassembly

- network links have ***MTU*** (***max***. ***transfer size***)
  - MTU - largest possible link-level frame
  - different link types,
  - different MTUs
- large IP datagram divided ("fragmented") within net
  - one datagram becomes several datagrams
  - "reassembled" only at *destination*
  - IP header bits used to identify, order related fragments

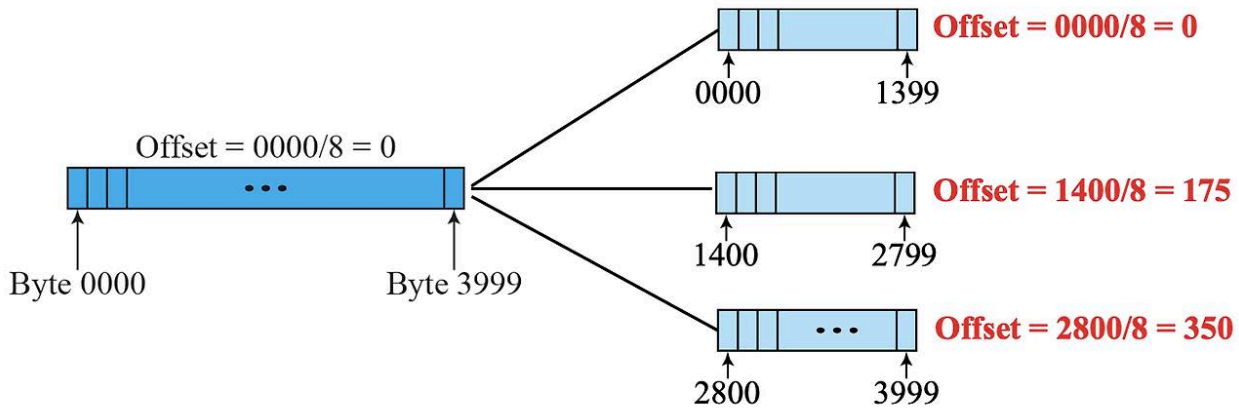


example:

- 4000 byte datagram
- MTU = 1500 bytes



当4000字节的数据包通过1400字节的以太网传输时:

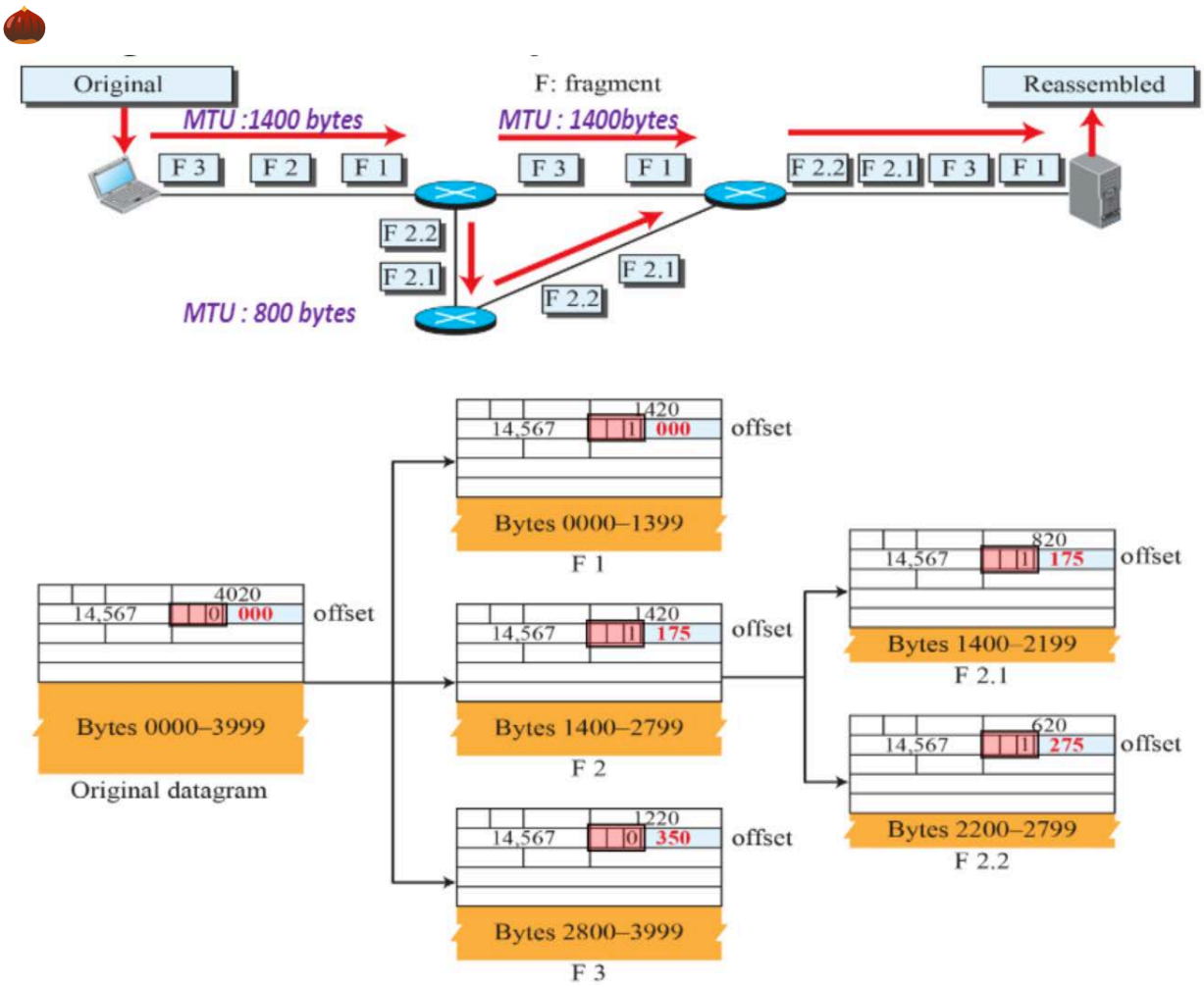


Fields related to Fragmentation

- Identification (16 bits): All fragments of a packet has the same ID number which is the same as that of the original packet. The R/x knows that all fragments having the same ID should be assembled into one packet
- Flags (3 bits):

D: Do not fragment
M: More fragments

- Fragmentation Offset (13 bits): Relative position of the fragment to the whole packet **measured in units of 8 Bytes** → Offset 需要是8字节的倍数



14,567：原始IP包的标识(Identification)字段的值。

IP 的总长度(IP Total Length)：

- F1 (Bytes 0000-1399): 总长度 = 1420 bytes = IP头部(20字节) + 数据(1400字节)
- F3 (Bytes 2800-3999): 总长度 = 1220 bytes = IP头部(20字节) + 剩余数据(1200字节)

Offset：F1: offset = 0：表示这是第一个分片；数据范围：bytes 0000-1399

F2: offset = 175；175 × 8 = 1400 bytes；数据范围：bytes 1400-2799

F2.1: offset = 175；继承自F2；数据范围：bytes 1400-2199

IP分片偏移量(Fragment Offset)字段需要是8字节的倍数，这是因为：

1. 在IP头部中，Fragment Offset字段是13位
2. 这13位表示的值需要乘以8才是真实的字节偏移量
3. 这样做的原因是：
    - 通过乘以8，13位可以表示更大范围的偏移量
    - 13位最大值是8191，乘以8后可以表示到65528字节的偏移量
    - 如果不乘8，13位只能表示到8191字节的偏移量

举个🌰：

- 如果Fragment Offset字段值为61
- 实际偏移量 = 61 × 8 = 488字节
- 这意味着这个分片的数据应该放在重组后的数据包的第488字节处

⭐ 所以当我们进行分片时：

- 每个分片的大小（除去IP头）必须是8的倍数
- 每个分片在原数据包中的位置（偏移量）也必须是8的倍数
- 唯一的例外是最后一个分片，它的大小可以不是8的倍数
- 每个分片都有完整的IP header（20字节）

IP v4. Addressing (Cont.)

- The Internet is made of combination of LANs and WANs connected via routers
- A host needs to be able to communicate with another host without worrying about which physical network must be passed through
- Hosts must therefore be identified uniquely and globally at the network layer
- For efficient and optimum routing, routers must also be identified uniquely and globally at the network layer

- IPv4 address is a 32-bit address, implemented in software, is used to uniquely and globally identify a host or a router on the Internet
- A device can have more than one IP address if it is connected to more than one network (multi-homed)
- An IP address have two parts, the netid and the hostid. They have variable lengths depending on the class of the address
- All devices on the same network have the same netid

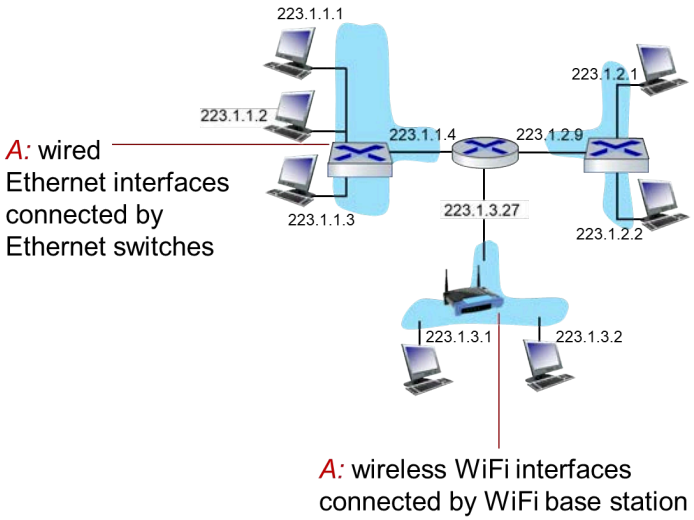
IP Addresses (Example)

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

interface: connection between host/router and physical link

- router's typically have multiple interfaces
- host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)

IP addresses associated with each interface



Q: how are interfaces actually connected?
A: See Data Link layer

dotted-decimal IP address notation:

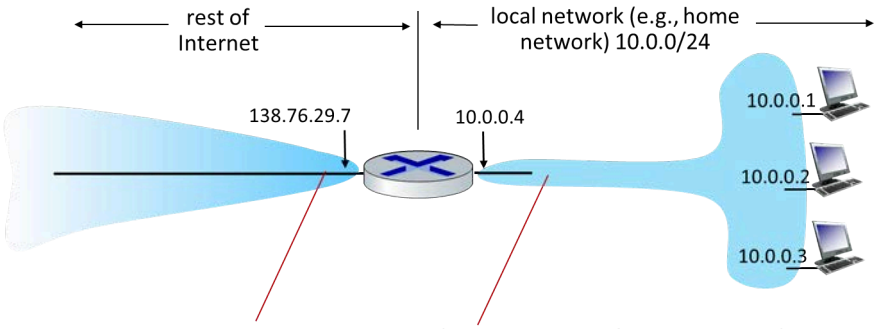
223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1

NAT: Network Address Translation

- NAT is a protocol that maintains a translation table for mapping an internal private IP address to a globally unique IP addresses and vice versa
- May be **static** (one-to-one) or **dynamic** (from a pool of global IP addresses)
- Implemented in the border (access) router separating the private network and the public Internet or as a stand-alone **multi-homed Server**
- NAT is a special type of **Proxy Server**. It was introduced with Windows 2000

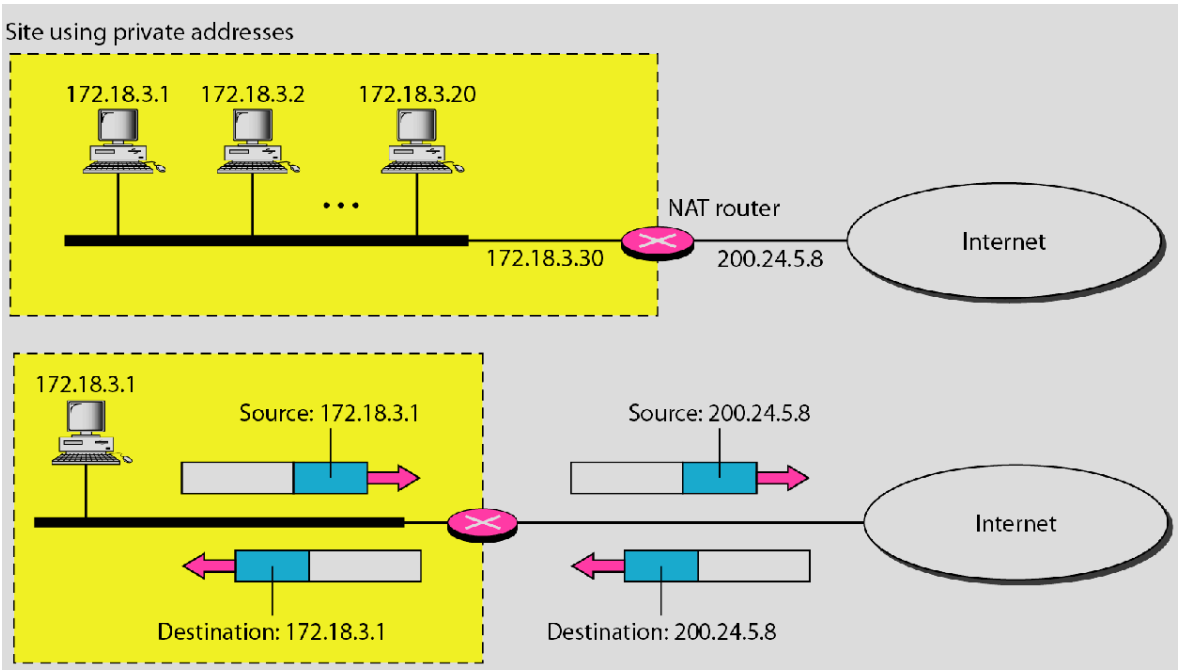
NAT: all devices in local network share just one IPv4 address as far as outside world is concerned



all datagrams *leaving* local network have *same* source NAT IP address: 138.76.29.7, but *different* source port numbers

datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

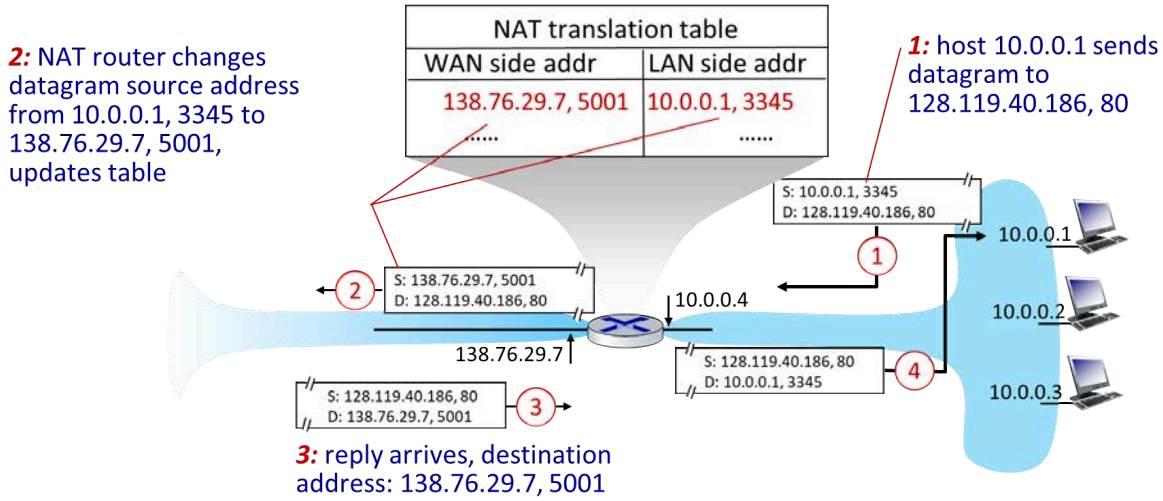
NAT Implementation



implementation: NAT router must (transparently):

- outgoing datagrams: replace (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
    - remote clients/servers will respond using (NAT IP address, new port #) as destination address
- remember (in NAT translation table) every (source IP address, port #) to (NAT IP address, new port #) translation pair
- incoming datagrams: replace (NAT IP address, new port #) in destination fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

NAT & PAT



Motivation for NAT

- Local network uses just one IP address as far as outside world is concerned:
- Range of addresses not needed from ISP: just one IP address for all devices
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network
- devices inside local net not explicitly addressable/visible by outside (a security plus)

解释:

1. 统一对外 **IP** 地址
- 整个局域网对外部世界来说只需要一个公网 IP 地址
- 不管局域网内有多少设备,外部网络只能看到这一个地址
- 这就像一栋大楼只需要一个门牌号,不管里面住了多少户人家
2. 节省 **IP** 地址资源
- 不需要向 ISP(网络服务提供商)为每个内网设备申请公网 IP
- 特别是在 IPv4 地址资源稀缺的情况下,这点非常重要
- 一个公网 IP 就可以满足整个局域网的需求

3. 灵活修改内网配置
- 可以随意更改内网设备的 IP 地址,而不会影响与外网的通信
- 因为外网只认识那个统一的公网 IP,不关心内网怎么变
4. 更换 **ISP** 更方便
- 就算更换了网络服务提供商、公网 IP 发生变化
- 内网设备的 IP 地址可以保持不变
- 不需要重新配置内网的每台设备
5. 提升安全性
- 外网无法直接访问或看到内网的设备
- 内网设备被"隐藏"在 NAT 之后
- 相当于给内网增加了一层保护,降低了被攻击的风险

Subnets

What's a subnet ?

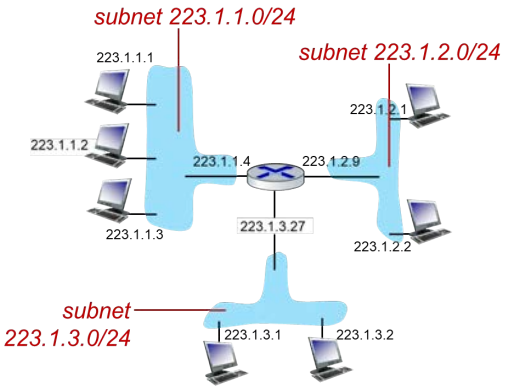
- device interfaces that can physically reach each other without passing through an intervening router

IP addresses have structure:

- subnet part: devices in same subnet have common high order bits
- host part: remaining low order bits

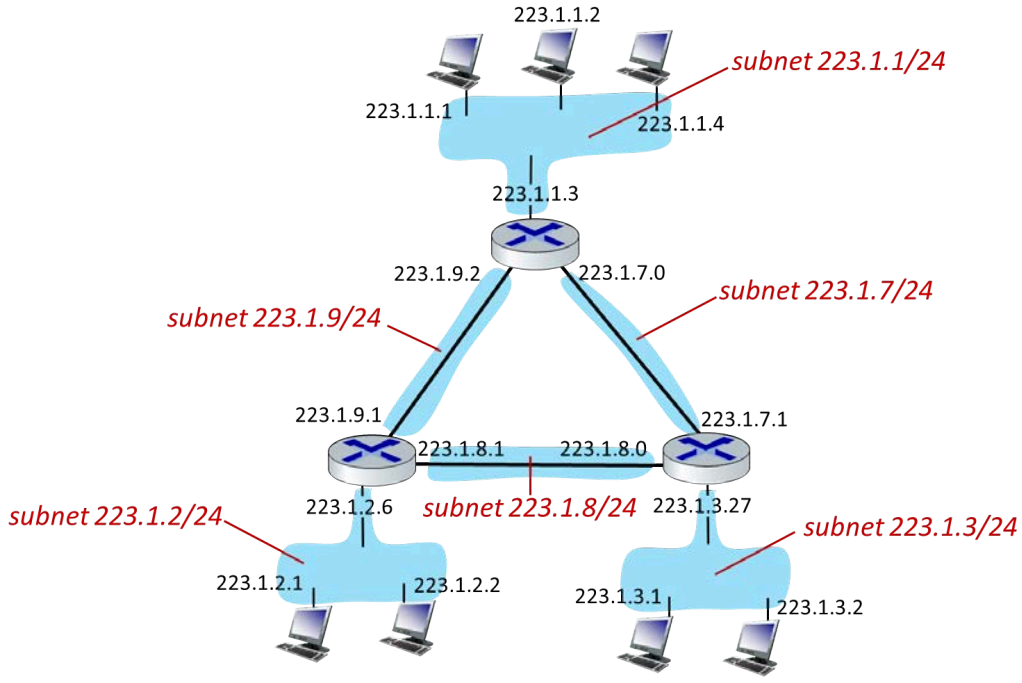
Recipe for defining subnets:

- detach each interface from its host or router, creating "islands" of isolated networks
- each isolated network is called a *subnet*



subnet mask: /24
(high-order 24 bits: subnet part of IP address)

(from prev. Slide. IP Address Expample)



Subnet Masking

- Subnetting is achieved by “stealing” some bits from the hostid field to represent the subnet portion of the address
- Those bits used for the subnetid are identified through the use of a **subnet mask**
- Masking is the process of extracting the address of the physical network (if subnetting is not used) or the subnet address (if subnetting is used) from an IP address
- A subnet mask is a 32-bit pattern having a “1” in every netid and subnetid locations and a “0” in every hostid location

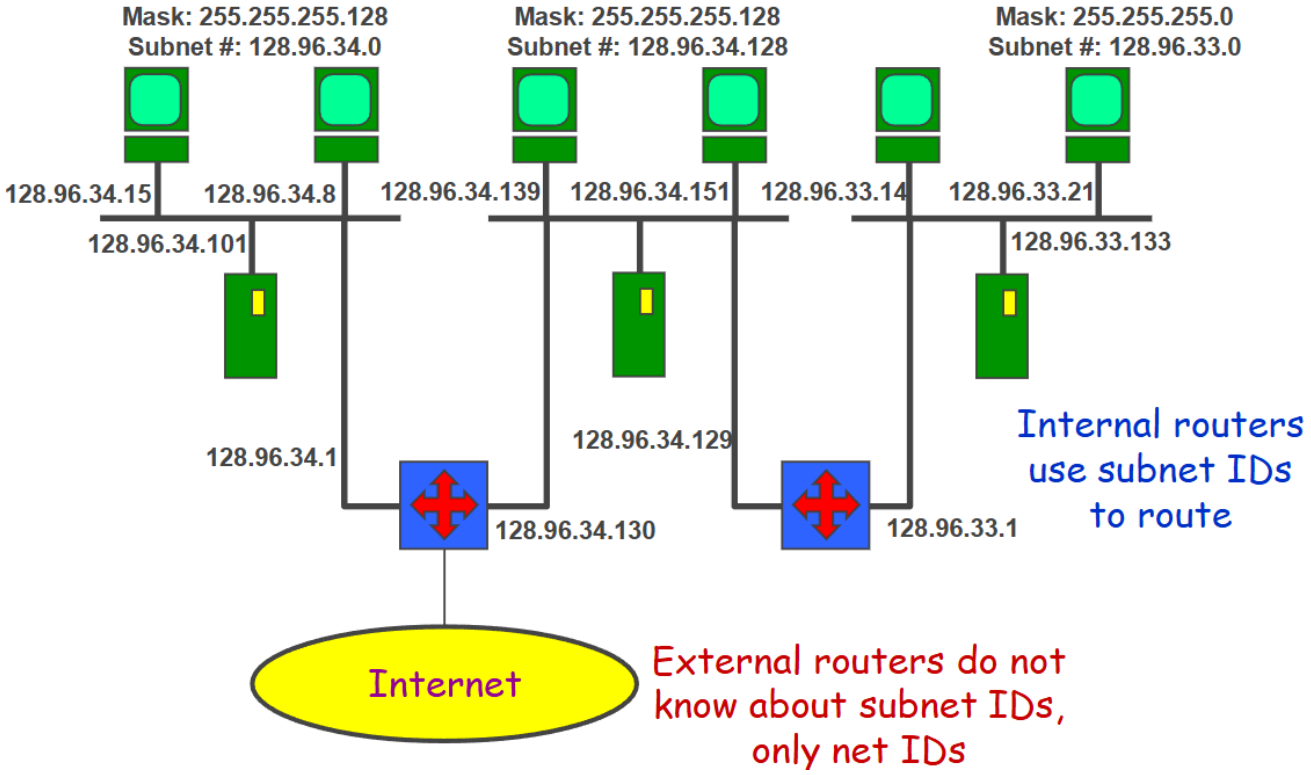
- Subnet masking is performed (both at the host and at the router) by applying “bit-wise-AND” operation between the IP address and the subnet mask

🌰 Class B network without subnetting

- 141.14.2.21 10001101.00001110.00000010.00010101
- 255.255.0.0 11111111.11111111.00000000.00000000
- “Bit-wise and” 10001101.00001110.00000000.00000000

IP address 141.14.2.21 → Mask 255.255.0.0 → Network address 141.14.0.0

of Subnetting



Subnet Router Routing Table

Partial Table in Router 128.96.33.1

Subnet Number	Subnet Mask	Next Hop
128.96.34.0	255.255.255.128	Left Router
128.96.34.128	255.255.255.128	Left Interface
128.96.33.0	255.255.255.0	Right Interface

```
for each table entry do
  if (DestAddr & Subnet Mask) = SubnetNumber
     if NextHop is an Interface
        deliver packet directly to DestAddr
     else
        deliver packet to Router
```

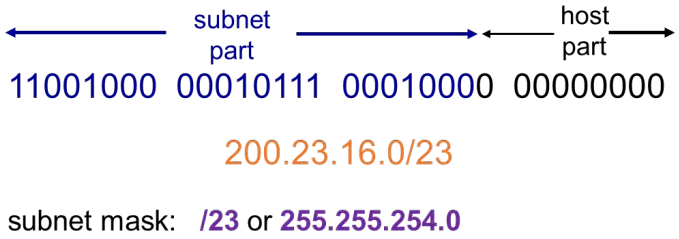
CIDR Notation

CIDR: Classless InterDomain Routing (pronounced “cider”)

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

CIDR: Classless InterDomain Routing (pronounced “cider”)

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



10.29 Routing Algorithms (week 10)

IP Packet Delivery

- Two Processes are required to accomplish IP packet delivery, namely the ***Routing*** Process and the ***Forwarding*** Process
  - ***Routing*** is the process of discovering and selecting the path to the destination according to some metrics. (1. 发现 2. 选择)
  - ***Forwarding*** is the process of inserting the IP packet into a Layer-2 frame and forwarding the frame to the next hop (which could be the destination host or another intermediate router). (1. encapsulation 2. forwarding)

Network-layer functions

forwarding: move packets from router’s input to appropriate router output → ***data plane***

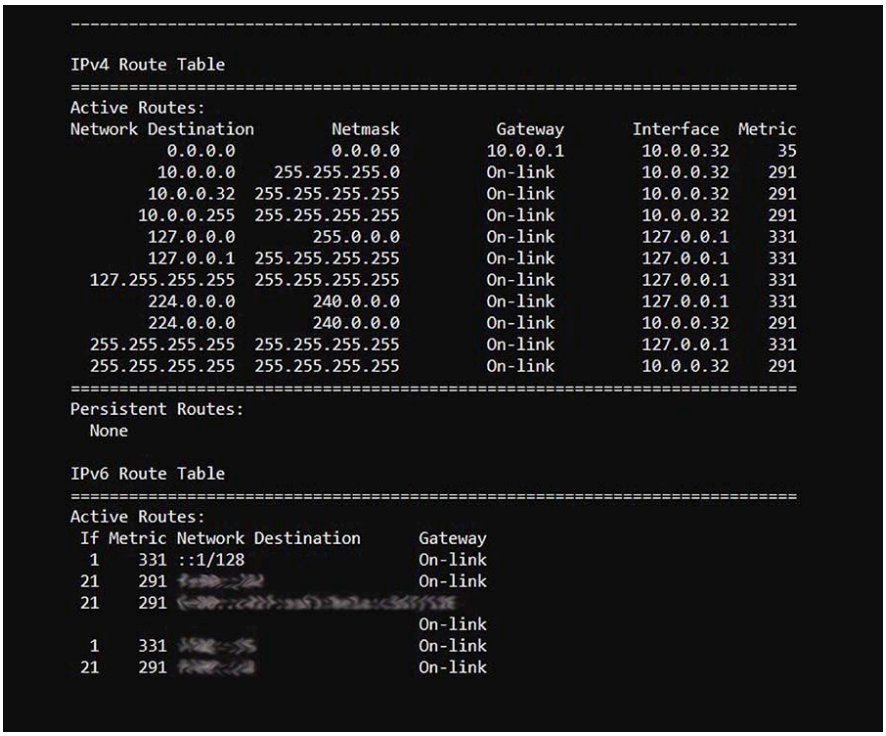
routing: determine route taken by packets from source to destination → ***control plane***

Two approaches to structuring network control plane:

1. per-router control (traditional)
2. logically centralized control (software defined networking)

Routing Tables

- Routing Tables are built up by the routing algorithms.
  They generally consist of:
  - ***Destination Network Address***: The network portion of the IP address for the destination network
  - ***Subnet Mask***: used to distinguish the network address from the host address
  - ***The IP address of the next hop*** to which the interface forwards the IP packet for delivery
  - ***The Interface*** with which the route is associated



```
---------------------------------------------------------------------------

IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0       10.0.0.1        10.0.0.32     35
         10.0.0.0    255.255.255.0       On-link         10.0.0.32    291
        10.0.0.32  255.255.255.255       On-link         10.0.0.32    291
       10.0.0.255  255.255.255.255       On-link         10.0.0.32    291
        127.0.0.0        255.0.0.0       On-link         127.0.0.1    331
        127.0.0.1  255.255.255.255       On-link         127.0.0.1    331
  127.255.255.255  255.255.255.255       On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0       On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0       On-link         10.0.0.32    291
  255.255.255.255  255.255.255.255       On-link         127.0.0.1    331
  255.255.255.255  255.255.255.255       On-link         10.0.0.32    291
===========================================================================
Persistent Routes:
  None

IPv6 Route Table
===========================================================================
Active Routes:
 If Metric Network Destination      Gateway
  1    331 ::1/128                  On-link
 21    291 [redacted]               On-link
 21    291 [redacted]
                                    On-link
  1    331 [redacted]               On-link
 21    291 [redacted]               On-link
```

典型字段:

- **Destination (目的地)**: 目标网络地址或主机地址（通常以CIDR表示）。
- **Next Hop (下一跳)**: 转发的下一跳路由器地址。
- **Interface (接口)**: 应使用的网络接口。
- **Metric (度量值)**: 用于选择最佳路径的指标，例如跳数或带宽。

Forwarding Tables

- After the routing lookup is completed and the next hop is determined, The IP packet is forwarded according to a local or remote delivery models
- ***Local delivery model***
    - is when the destination and the host are on the same local network. In this case, the IP packet is inserted into a MAC-frame which is forwarded directly to the destination
- ***Remote delivery model***
    - is when the destination and the host are on different networks. In this case, the IP packet is inserted into a Layer-2 frame which is forwarded to the next hop router

⭐ *L3* Routing Table vs. Forwarding Table

L3 ***Routing Table*** 示例:

Destination	Next Hop	Metric	Protocol	Flags
192.168.1.0/24	192.168.0.1	10	OSPF	C
192.168.1.0/24	192.168.0.2	20	OSPF	C
10.0.0.0/8	10.1.1.1	5	BGP	D

L3 ***Forwarding Table*** 示例:

Destination	Next Hop	Interface	Metric
192.168.1.0/24	192.168.0.1	eth0	10
10.0.0.0/8	10.1.1.1	eth1	5

差异:

- ***Routing Table*** 包含多个路径选项，决定最佳路径后，将其同步到 **L3** ***Forwarding Table***
- **L3** ***Forwarding Table*** 只包含最优路径，用于实际转发。

工作流程中的关系

1. ***Routing Table*** 由静态配置或动态路由协议（如 OSPF、BGP、RIP）更新。
2. 路由器从 Routing Table 中选择最佳路径，生成精简的 **L3** ***Forwarding Table***。
3. 数据包转发时，仅查找 **L3 Forwarding Table**，以提升转发效率。

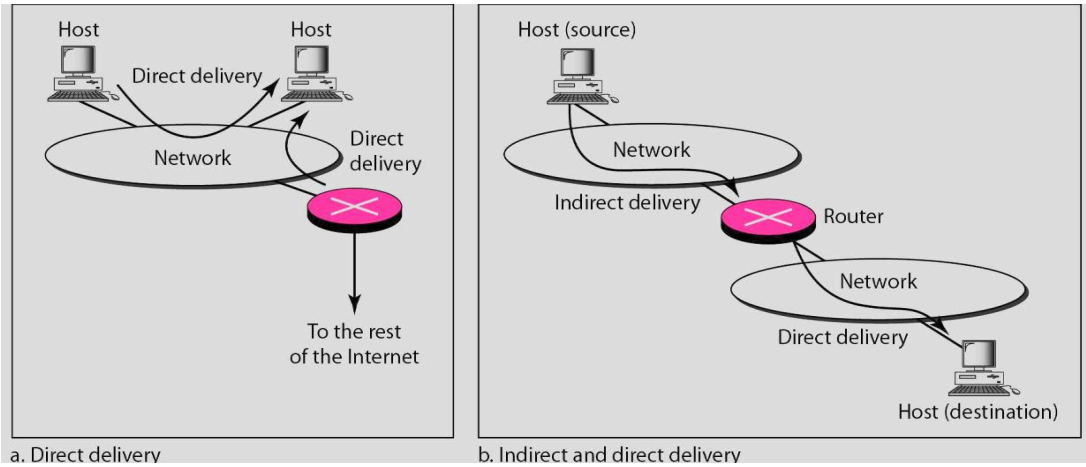
⭐ vs. L2 Forwarding Table

示例:

MAC Address	Port	VLAN ID	TTL
00:1A:2B:3C:4D:5E	3	100	300s
11:22:33:44:55:66	5	200	500s
FF:FF:FF:FF:FF:FF	All	-	N/A

- 典型字段:
  - **MAC Address**: 设备的目标MAC地址。
  - **Port**: 数据帧应该通过哪个交换机端口转发。
  - **VLAN ID**: 如果网络支持VLAN，则包含VLAN相关信息。
  - **TTL (Time to Live)**: 条目在表中的寿命，用于清理不活动的记录。

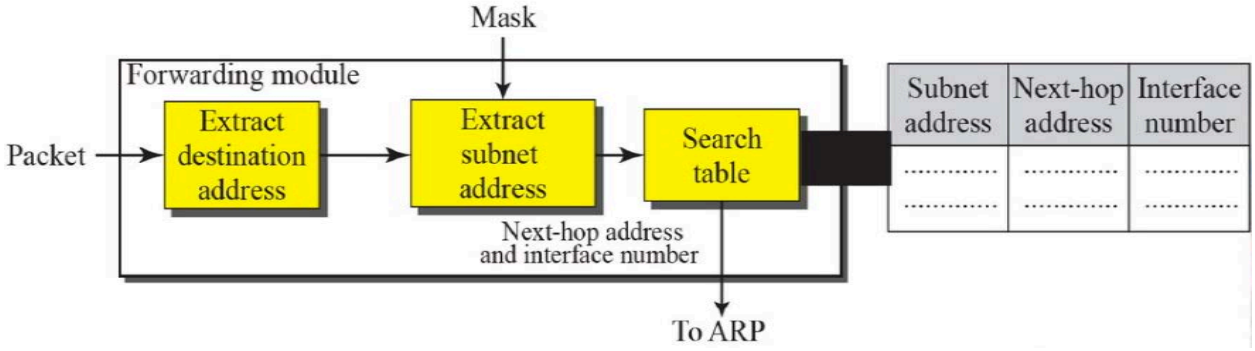
Local (Direct) vs. Remote Delivery



★ Note: 同一网络内都是 Local, Router 的内网 interface 也算

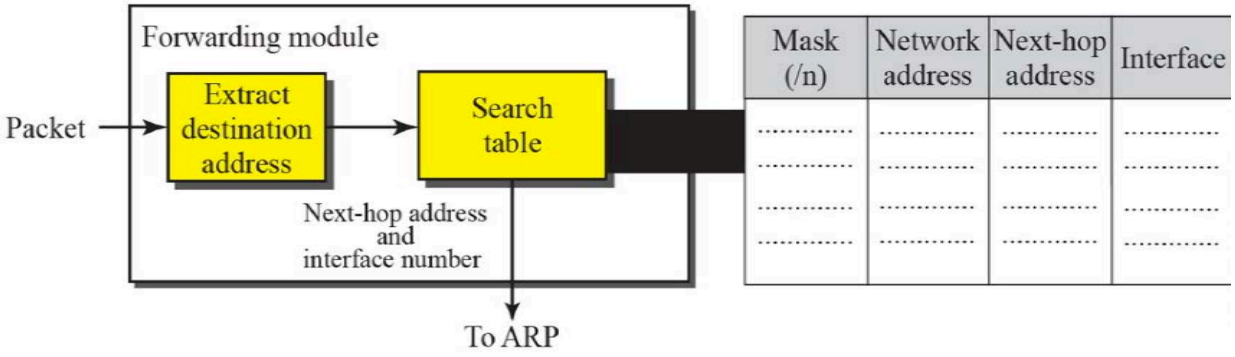
Forwarding Module

- 不带子网转发



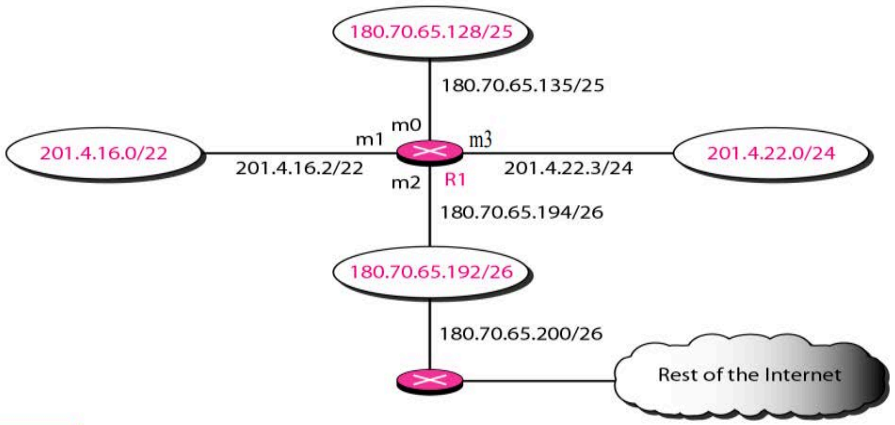
1. 模块提取数据包的目的地址。
2. 如果目标地址与表中主机指定的地址相同，则从表中提取下一跳和接口号
3. 使用目标地址和掩码提取子网地址
4. 查表查找下一跳地址并查找接口号。如果没有匹配的条目，则使用默认值
5. 下一跳地址和接口号发送到 ARP。

- 用于具有子网划分的基于类别的地址的单一化转发模块



  - 整个地址空间是一个单一的实体（没有类的概念）
  - 这意味着每个块需要一行信息。
  - 基于类的寻址需要路由表中的三列，但无类寻址至少需要四列（因为必须包含掩码 (/n)）。

🌰 Forwarding Table



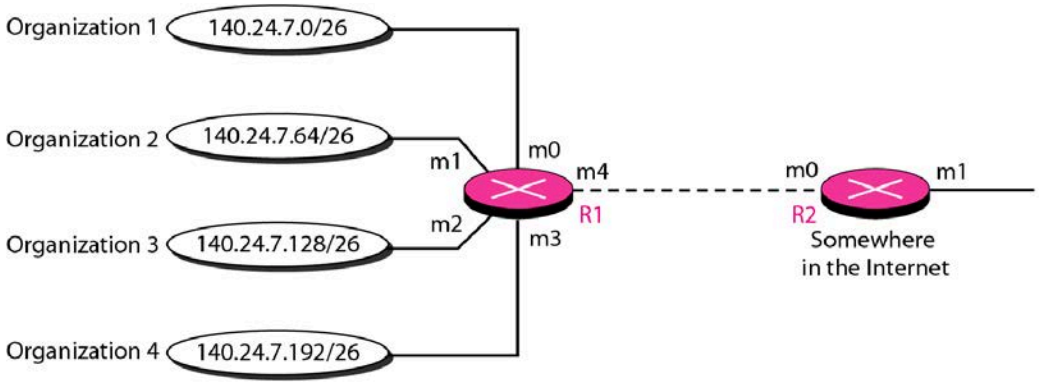
Solution ***Routing table for router R1 in Figure above***

Mask	*Network Address*	*Next Hop*	*Interface*
/26	180.70.65.192	—	m2
/25	180.70.65.128	—	m0
/24	201.4.22.0	—	m3
/22	201.4.16.0	....	m1
Any	Any	180.70.65.200	m2

The table is sorted from the longest mask to the shortest mask.

- Suppose that **R1** receives a Packet destined to `180.70.65.140`.
  The router performs the following steps:
  - The ***first mask (/26)*** is applied to the destination address.
    Result is `180.70.65.128`. → No match
  - The ***second mask (/25)*** is applied to the destination address.
    Result is `180.70.65.128`. → A match.
    The next hop address (in this case it is the destination host address) and the interface ***m0*** is then passed to the ARP module to get the MAC address

CIDR: Address Aggregation



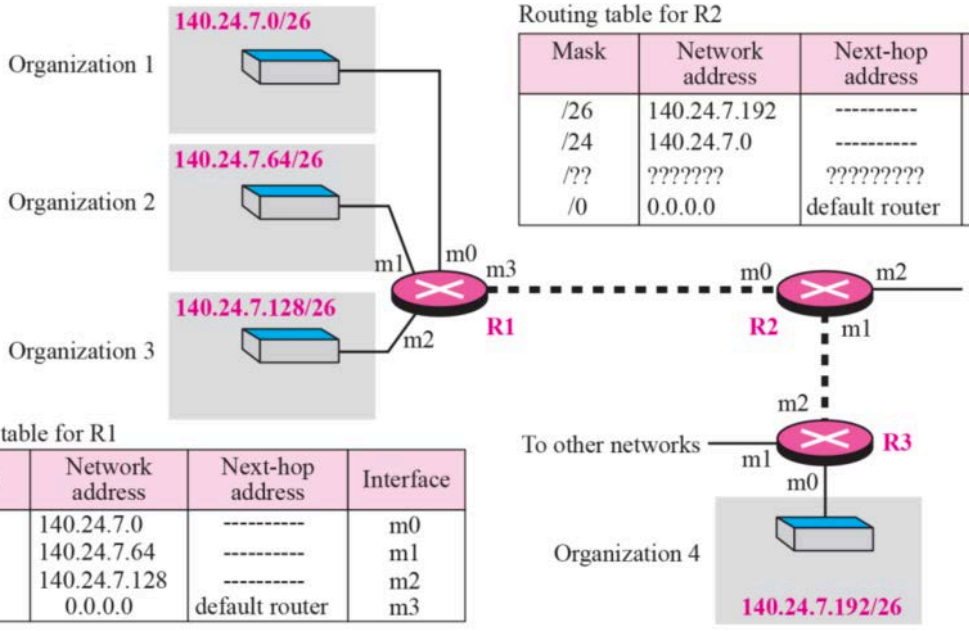
Mask	Network address	Next-hop address	Interface
/26	140.24.7.0	----------	m0
/26	140.24.7.64	----------	m1
/26	140.24.7.128	----------	m2
/26	140.24.7.192	----------	m3
/0	0.0.0.0	Default	m4

Routing table for R1

Mask	Network address	Next-hop address	Interface
/24	140.24.7.0	----------	m0
/0	0.0.0.0	Default	m1

Routing table for R2

Longest Mask (Prefix) Matching



Routing table for R2

Mask	Network address	Next-hop address	Interface
/26	140.24.7.192	----------	m1
/24	140.24.7.0	----------	m0
/??	???????	?????????	m1
/0	0.0.0.0	default router	m2

Routing table for R1

Mask	Network address	Next-hop address	Interface
/26	140.24.7.0	----------	m0
/26	140.24.7.64	----------	m1
/26	140.24.7.128	----------	m2
/0	0.0.0.0	default router	m3

Routing table for R3

Mask	Network address	Next-hop address	Interface
/26	140.24.7.192	----------	m0
/??	???????	?????????	m1
/0	0.0.0.0	default router	m2

: Longest Prefix Matching

Longest prefix matching

longest prefix match

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

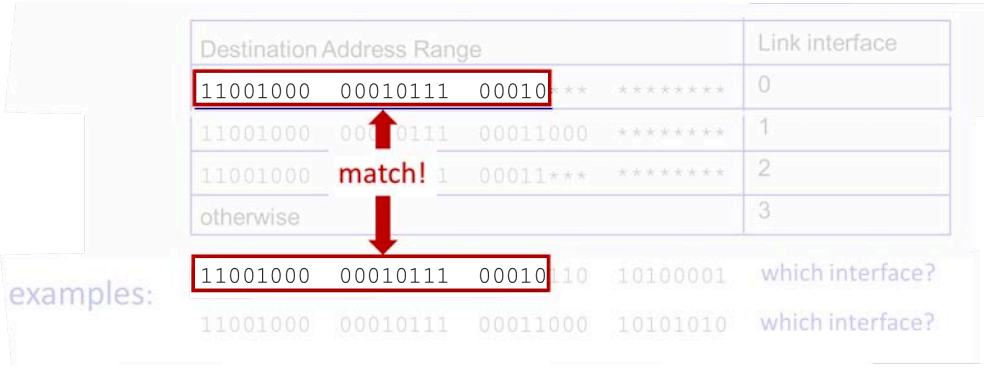
Destination Address Range	Link interface
11001000 00010111 00010*** ********	0
11001000 00010111 00011000 ********	1
11001000 00010111 00011*** ********	2
otherwise	3

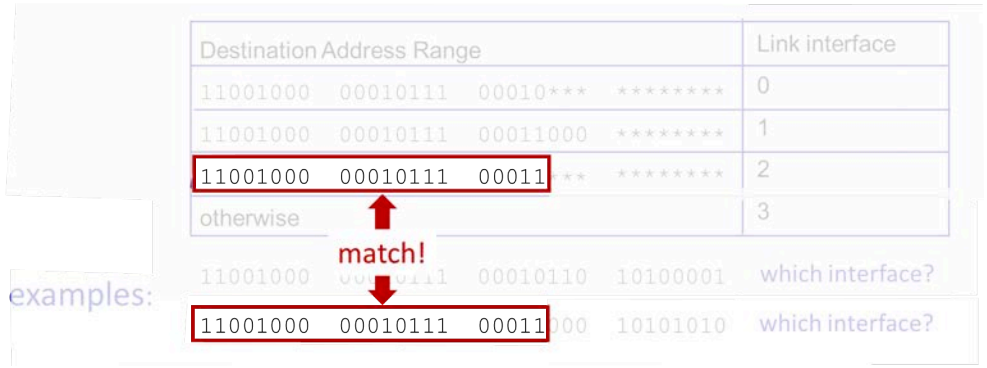
examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

Ans:





Static vs. Dynamic Routing

Static Routing

- **Definition**: Static Routing Tables are entered **manually**.
- ***Strengths***:
    1. **Ease of Use**: Simple to configure for small or stable networks.
    2. **Reliability**: Fixed paths ensure predictable behavior.
    3. **Control**: Administrators have full control over routing decisions.
    4. **Security Through Obscurity**: Not shared with other routers, making it less vulnerable to attacks.
    5. **Efficiency**: Minimal processing overhead as no dynamic updates are needed.
- ***Weaknesses***:
    1. **Not Scalable**: Inefficient in large or frequently changing networks.
    2. **Not Adaptable to Link Failures**: Manual updates are needed when network conditions change.

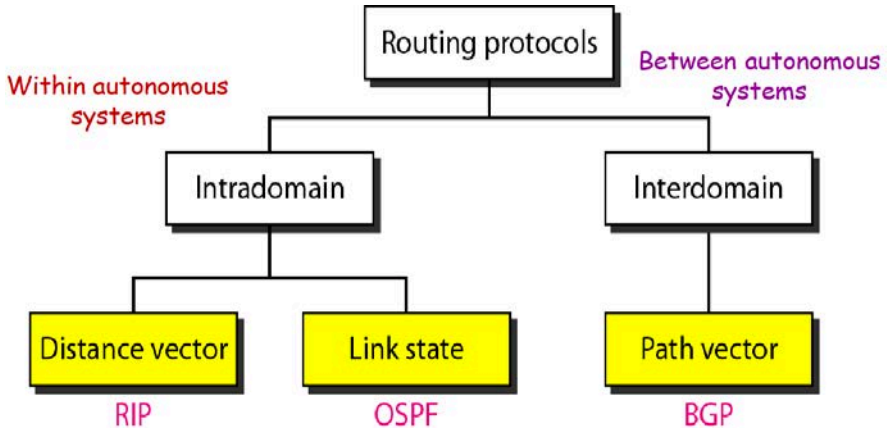
Dynamic Routing

- **Definition**: Routing tables are **automatically** created and updated by routers exchanging information about network status.
- **Types**:
    - ***Distance Vector Protocols***:
        - Example: **RIP (Routing Information Protocol)**.
        - Routers share routing tables with neighbors.
    - ***Link State Protocols***:
        - Example: **OSPF (Open Shortest Path First)**.
        - Routers maintain a global view of the network topology.
- **Key Features**:
    - Adapts to network changes, such as link failures.
    - More suitable for large, dynamic networks.

Routing Metrics

- Used by ***dynamic routing*** protocols to establish preference for a particular route.
- Goal: support Route Diversity and Load Balancing
- Most Common routing metrics:
    - Hop count (minimum # of hops)
    - Shortest distance
    - Bandwidth/Throughput (maximum throughput)
    - Load (actual usage)
    - Delay (shortest delay)
    - Reliability
    - Cost

Popular Routing Protocols



Distance Vector (DV) Routing

- Example: ***RIP***: ***Routing Information Protocol***
  - Based on an algorithm by ***Bellman-Ford*** (Dynamic Programming)
  - Each router on the network compiles a list of the networks it can reach (in the form of a distance vector) and exchange this list with its ***neighboring routers only***
  - Upon receiving vectors from each of its neighbors, the router computes its own distance to each neighbor. Then, for every network X, router finds that neighbor who is closer to X than to any other neighbor. Router updates its cost to X. After doing this for all X, router goes to the first step.

Distance vector algorithm

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

Bellman-Ford equation

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

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

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

min taken over all neighbors v of x

direct cost of link from x to v

Distance vector algorithm

key idea:

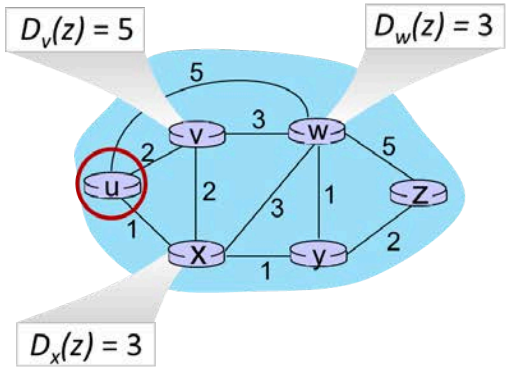
- from time-to-time, each node sends its own distance vector estimate to neighbors
- when *x* receives new DV estimate from any neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow min_v\{c_{x,v} + D_v(y)\} \text{ for each node } y \in N$$

- under minor, natural conditions, the estimate $D_x(y)$ *converge to the actual least cost* $d_x(y)$

Bellman-Ford Example

Suppose that *u*'s neighboring nodes, *x,v,w,* know that for destination *z*:

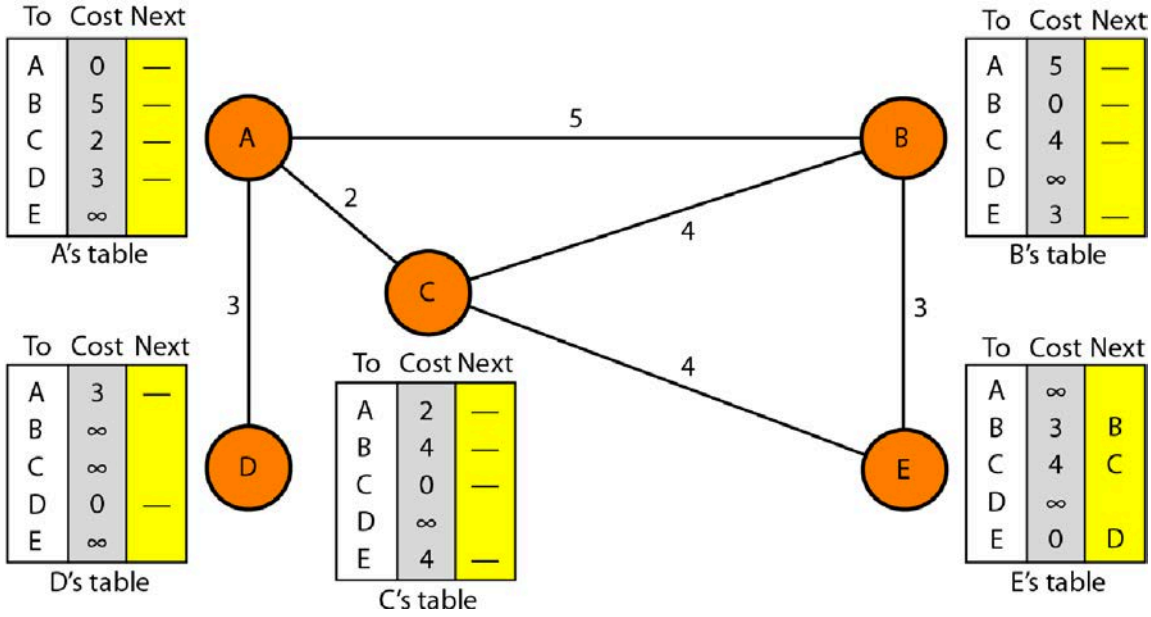


Bellman-Ford equation says:

$$D_u(z) = \min \{ c_{u,v} + D_v(z), \; c_{u,x} + D_x(z), \; c_{u,w} + D_w(z) \}$$
$$= \min \{2 + 5, \; 1 + 3, \; 5 + 3\} = 4$$

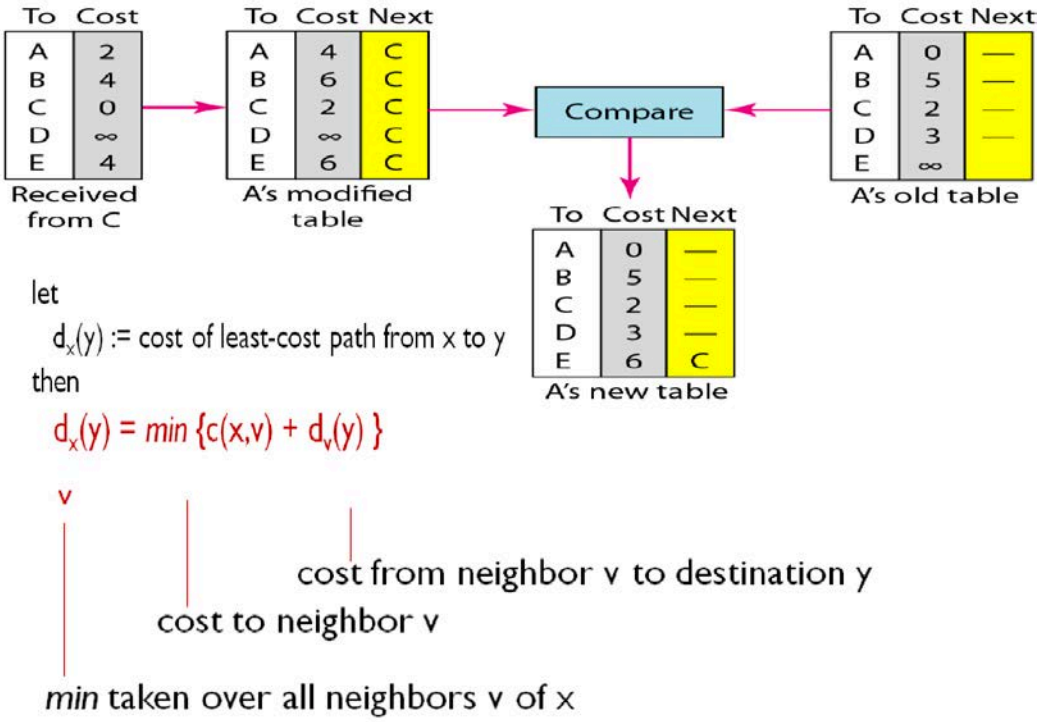
node achieving minimum (x) is next hop on estimated least-cost path to destination (z)

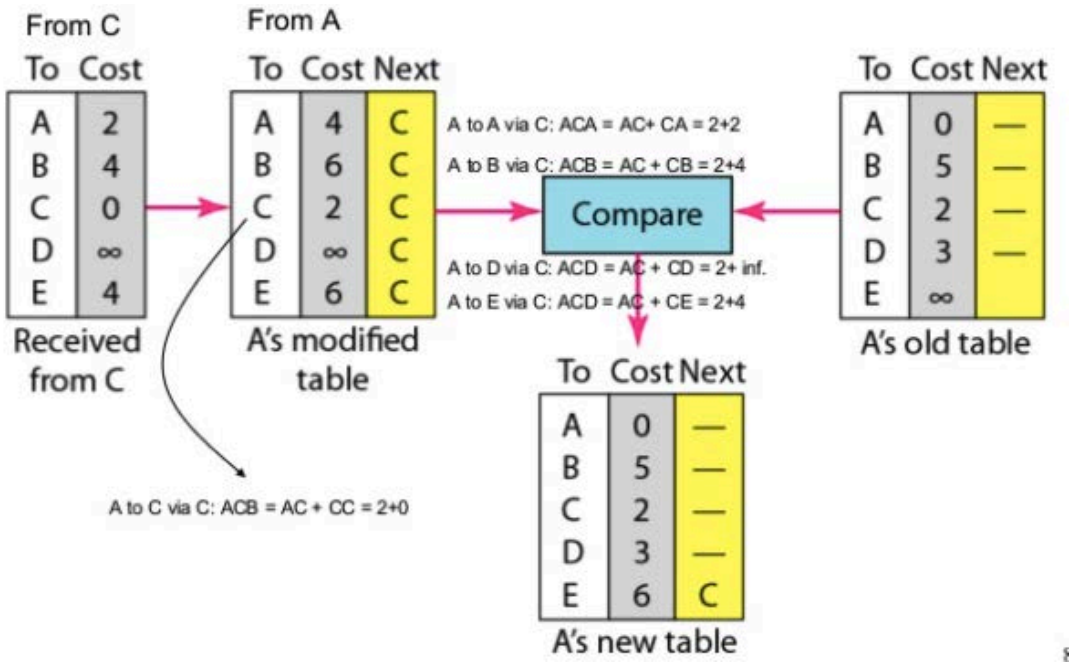
Initial Distance Vector Tables



- 从路由器A的角度来看，当去A时，成本是0，因为它要去自己，而当去B时，成本是5.....当去E时，你不知道如何到达那里。

Updating Distance Vector Tables

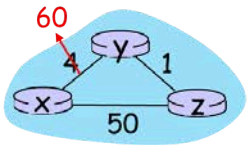




- 从C的角度来看成本，去A需要花费2。当A收到C的信息时，每张表都会加上2的成本，并且每张表都会与A现有的表进行比较。
- 从A的角度来看，发送给原始人时的成本为0，而通过C发送时的成本为4。这样的话就不需要通过C发送了，所以最终表中从A到A的cost就变成0了。
- 另外，现有表中没有关于去E的信息，但是查看从C收到的信息，它说花费6，所以在最终表中存储为从A到E花费6 。
- 这样，就可以对信息进行一一比较，识别出最优路由，构建路由表。

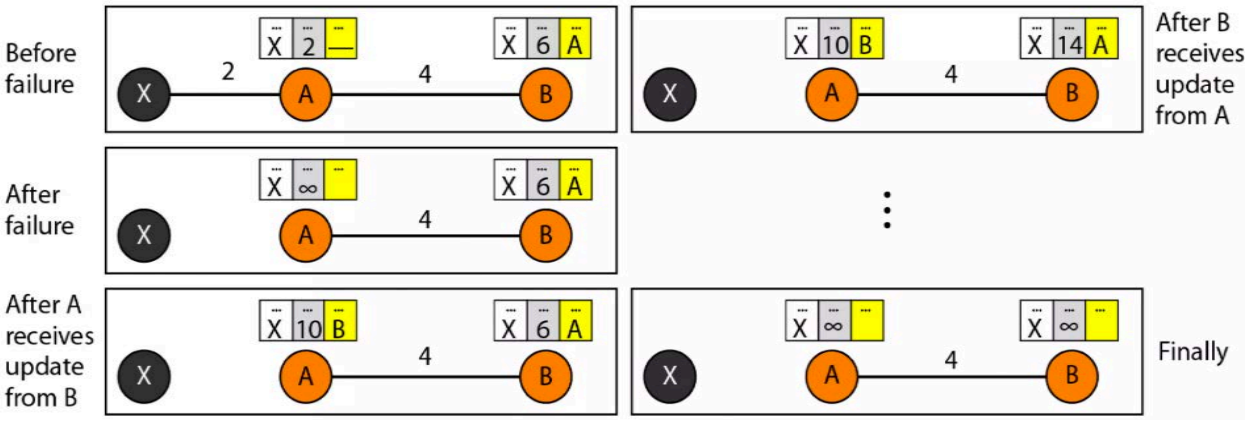
count-to-infinity Problem (Instability)

link cost changes:

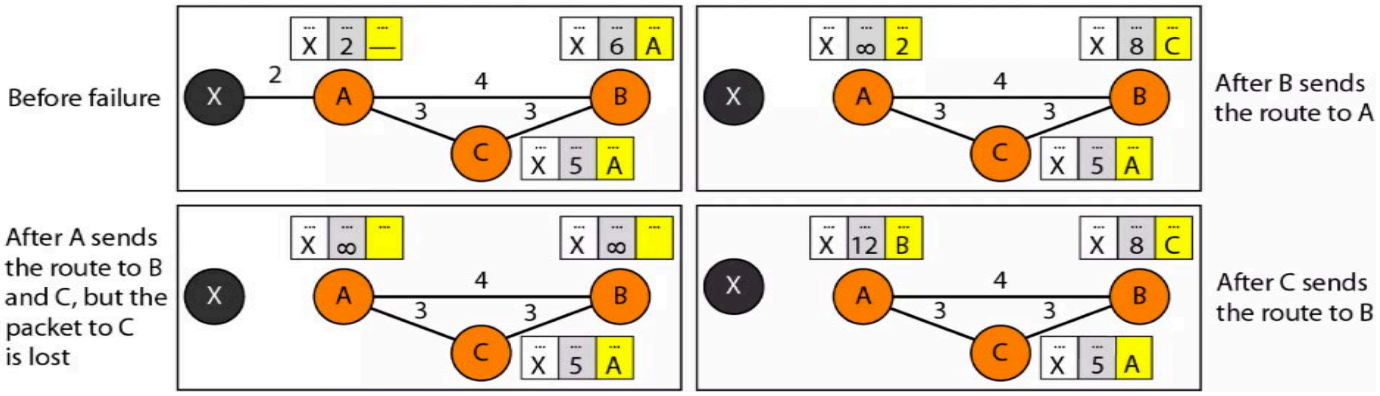


- node detects local link cost change
- "bad news travels slow" – ***count-to-infinity problem***:
  - *y* sees direct link to *x* has new cost 60, but z has said it has a path at cost of 5. So *y* computes "my new cost to x will be 6, via z); notifies *z* of new cost of 6 to *x*.
  - *z* learns that path to *x* via *y* has new cost 6, so *z* computes "my new cost to *x* will be 7 via y), notifies *y* of new cost of 7 to *x*.
  - *y* learns that path to *x* via *z* has new cost 7, so *y* computes "my new cost to *x* will be 8 via y), notifies *z* of new cost of 8 to *x*.
  - *z* learns that path to *x* via *y* has new cost 8, so *z* computes "my new cost to *x* will be 9 via y), notifies *y* of new cost of 9 to *x*.
  - ...
- see text for solutions. *Distributed algorithms are tricky!*

2-Node Instability



3-Node Instability

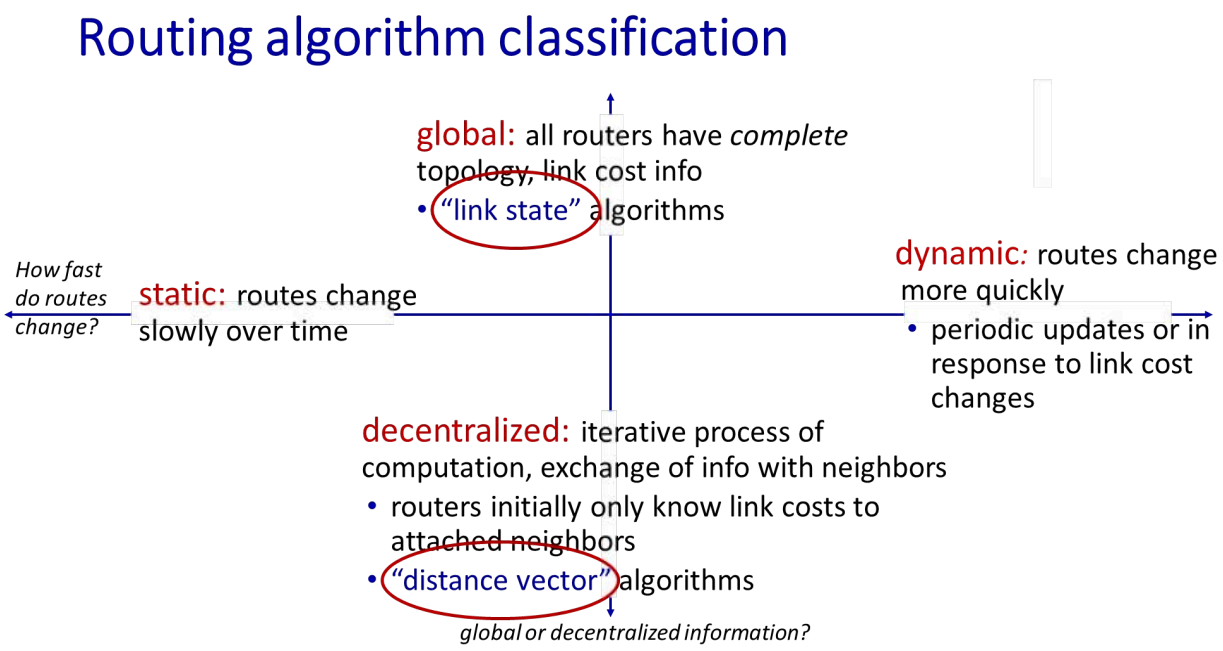


⭐ *Q. 2-Node Instability* 里，最后表里到*X*的距离都是无穷，好像符合实际没有问题？

问题的关键在于这个过程本身,而不是最终结果。虽然最后确实都会达到无穷大,但存在以下问题:

1. 收敛时间问题:
   - 路由器们不是直接认识到X不可达
   - 而是通过反复交换信息、不断增加距离值的方式慢慢达到无穷大
   - 这个过程可能需要很长时间(称为收敛时间)
   - 在这个过程中,网络处于不稳定状态
2. 资源消耗问题:
   - 在达到无穷大之前,路由器们会不断交换更新信息
   - 每次更新都会占用网络带宽
   - 同时消耗路由器的处理资源
3. 临时环路问题:
   - 在收敛过程中,可能会产生临时的路由环路
   - 比如A认为可以通过B到达X
   - B又认为可以通过A到达X
   - 导致数据包在A和B之间循环传递

Routing algorithm classification



Link State (***LS***) Routing

- ***Link-State*** (***LS***) Protocols
  - Based on an algorithm by **Dijkstra**
  - Each router on the network is assumed to know the **state of the links** to all its neighbors (Cost, Operating Status, Bandwidth, Delay, etc...)
  - Each router will **disseminate** (传播, 宣传 via reliable flooding of link state packets, LSPs) the information about its link states to all routers in the network.
  - In this case, every router will have enough information to build a **complete map** of the network and therefore is able to construct a ***Shortest Path Spanning Tree*** from itself to every other router

Link State Packets

- The link state packets consist of the following information:
  - The address of the node creating the LSP
  - A list of directly connected neighbors to that node with the cost of the link to each neighbor
  - A sequence number to make sure it is the most recent one
  - A time-to-live to insure that an LSP doesn't circulate indefinitely
- A node (router) will only send an LSP if there is a failure (change of status) to some of its links or if a timer expires

Dijkstra's link-state routing algorithm

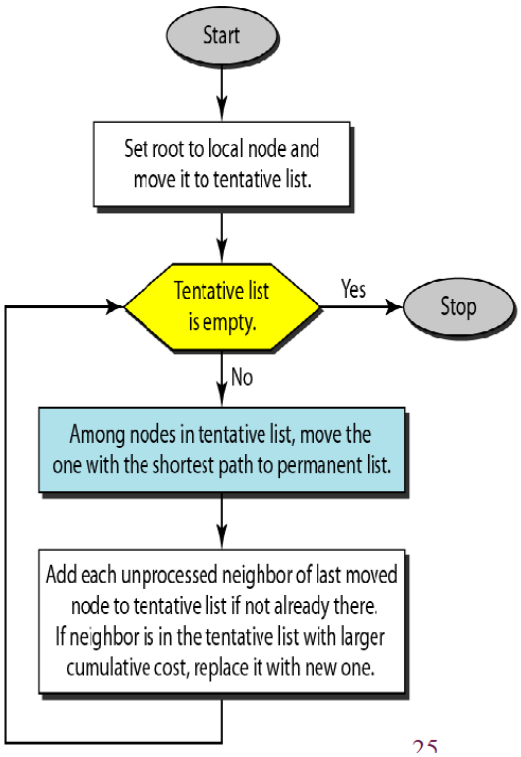
- centralized: network topology, link costs known to *all* nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- computes least cost paths from one node ("source") to all other nodes
  - gives *forwarding table* for that node
- iterative: after *k* iterations, know least cost path to *k* destinations

notation

- $c_{x,y}$: direct link cost from node *x* to *y*; = ∞ if not direct neighbors
- *D(v): current* estimate of cost of least-cost-path from source to destination *v*
- *p(v):* predecessor node along path from source to *v*
- *N':* set of nodes whose least-cost-path *definitively* known

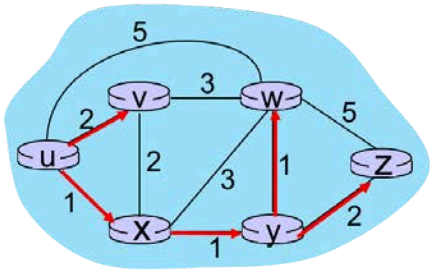
Dijkstra Algorithm

- SPT = {a}
- for all nodes v
  - if v adjacent to a then D(v) = cost (a, v)
  - else D(v) = infinity
- Loop
  - find w not in SPT, where D(w) is min
  - add w in SPT
  - for all v adjacent to w and not in SPT
    - D(v) = min (D(v), D(w) + C(w, v))
- until all nodes are in SPT

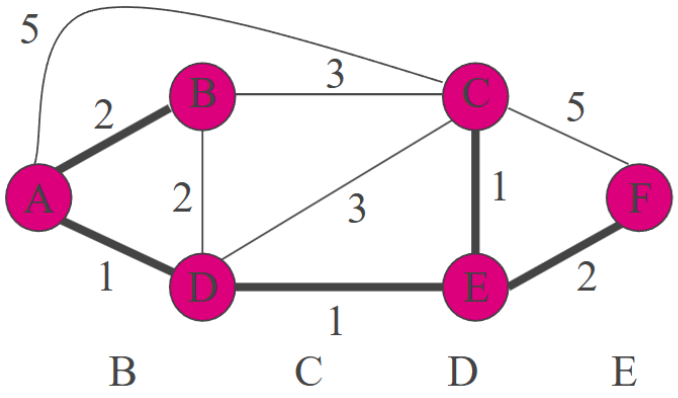


Dijkstra

Step	N'	v D(v),p(v)	w D(w),p(w)	x D(x),p(x)	y D(y),p(y)	z 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					



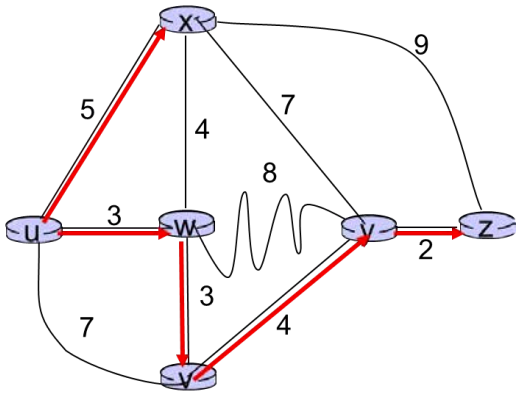
8 *Loop*
9 find *a* not in *N'* such that *D(a)* is a minimum
10 add *a* to *N'*
11 update *D(b)* for all *b* adjacent to *a* and not in *N'* :
D(b) = min (*D(b)*, *D(a)* + $c_{a,b}$)



step	SPT	D(b), P(b)	D(c), P(c)	D(d), P(d)	D(e), P(e)	D(f), P(f)
0	A	2, A	5, A	1, A	~	~
1	AD	2, A	4, D		2, D	~
2	ADE	2, A	3, E			4, E
3	ADEB		3, E			4, E
4	ADEBC					4, E
5	ADEBCF					

2

Step	N′	v D(v), p(v)	w D(w), p(w)	x D(x), p(x)	y D(y), p(y)	z D(z), p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					

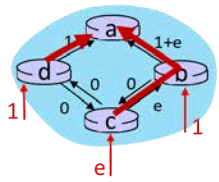


notes:

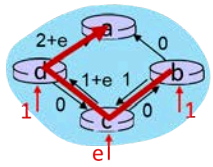
- construct least-cost-path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)

Dijkstra's algorithm: oscillations possible 振荡

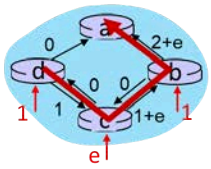
- when link costs depend on traffic volume, route oscillations possible
- sample scenario:
  - routing to destination a, traffic entering at d, c, e with rates 1, e (<1), 1
  - link costs are directional, and volume-dependent



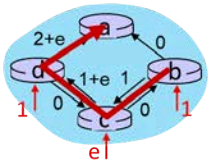
initially



given these costs, find new routing.... resulting in new costs



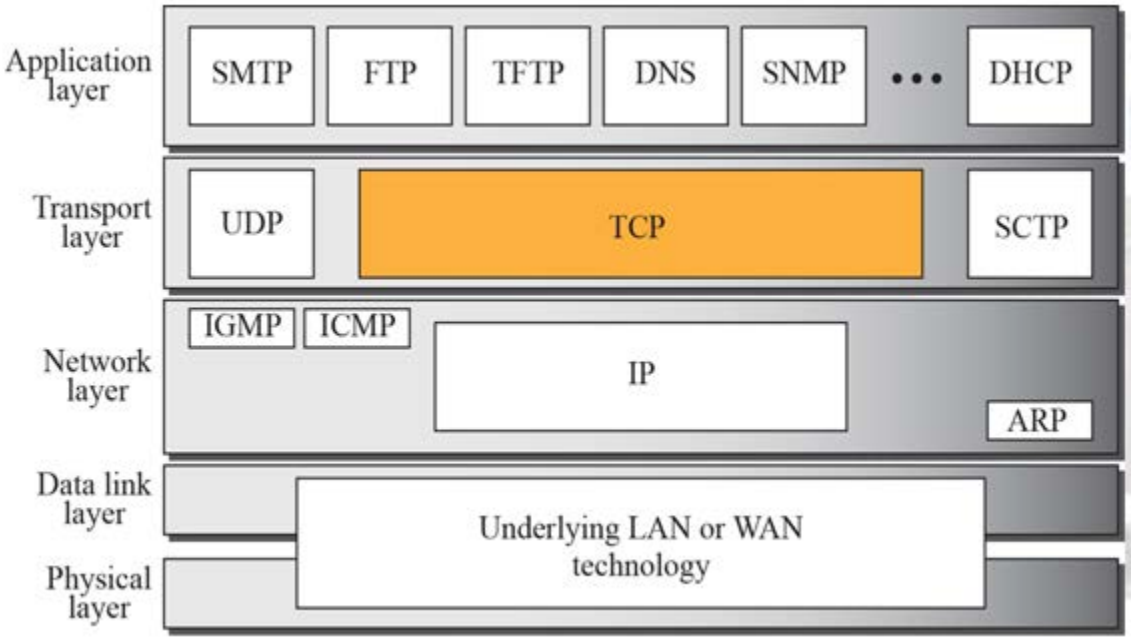
given these costs, find new routing.... resulting in new costs



given these costs, find new routing.... resulting in new costs

11.05 Transport Layer Protocols

L4 stack:



Transport services and protocols

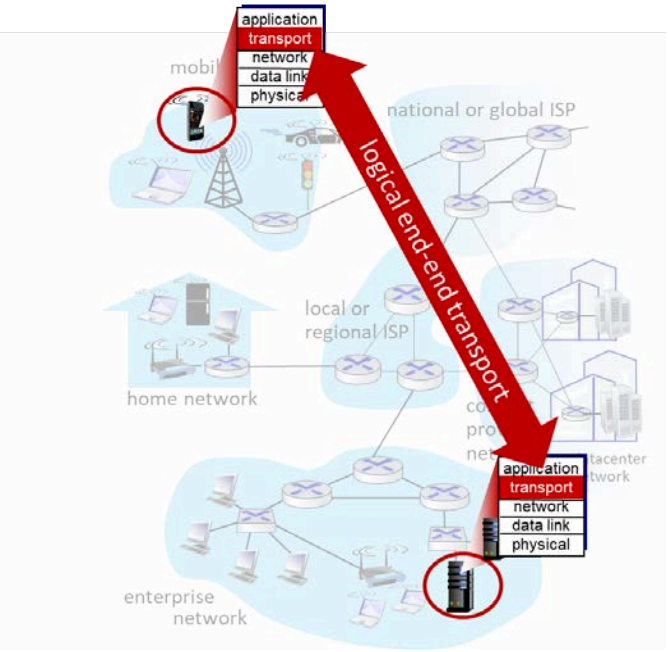
provide *logical communication* between application processes running on different hosts

transport protocols actions in end systems:

- *sender*: breaks application messages into *segments*, passes to network layer
- *receiver*: reassembles segments into messages, passes to application layer

two transport protocols available to Internet applications

- TCP, UDP



Functions of Transport Protocols

Functions of the transport layer protocols include:

- Provide for ***Process-to-Process*** communications.
  To accomplish this task, Port Numbers are used to identify the process, at both the client and at the server side
- Provide for **end-to-end** ***Error Checking*** (both *TCP* and *UDP*),
  Error Control and ***Flow*** and ***Congestion control*** (only *TCP*)
- TCP is a reliable protocol, UDP is an unreliable Protocol

⭐ Neither ***TCP*** nor ***UDP*** provides for "Guaranteed Delay" or Guaranteed Bandwidth"

- **TCP:** Transmission Control Protocol
  - reliable, in-order delivery
  - congestion control
  - flow control
  - connection setup
- **UDP:** User Datagram Protocol
  - unreliable, unordered delivery
  - no-frills extension of "best-effort" IP
- services ***not*** available:
  - delay guarantees
  - bandwidth guarantees

Transport vs. network layer services and protocols

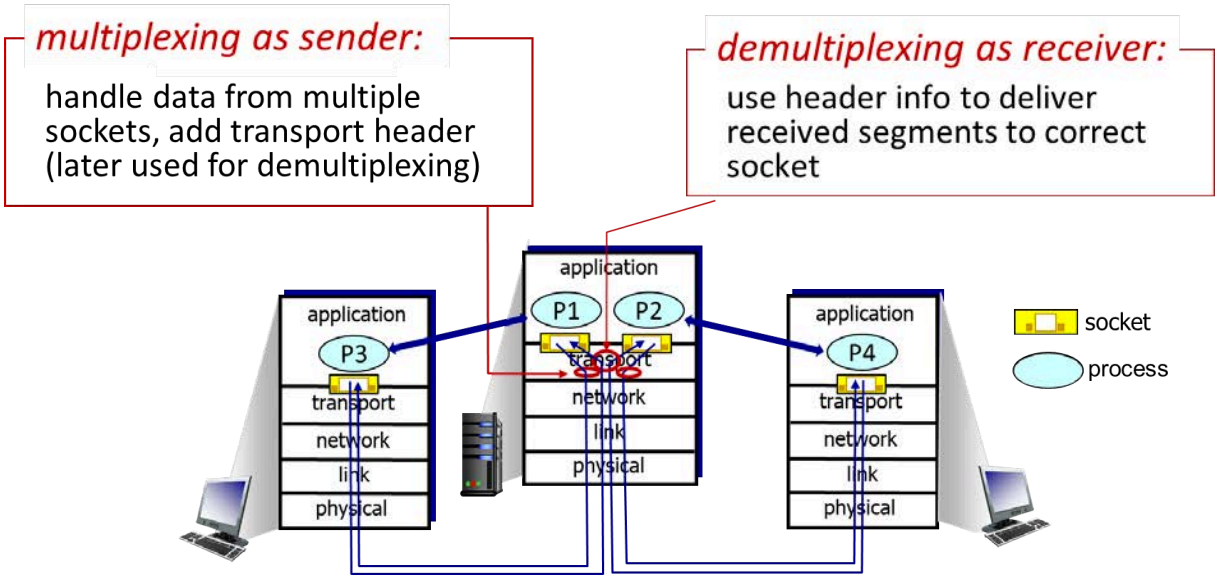
- transport layer: communication between *processes*
  - relies on, enhances, network layer services
- network layer: communication between *hosts*

household analogy:

12 kids in Ann's house sending letters to 12 kids in Bill's house:

- hosts = houses
- processes = kids
- app messages = letters in envelopes

Multiplexing/Demultiplexing



⭐ *Q: how did transport layer know to deliver message to Firefox browser process (application P3) rather then Netflix process or Skype process (application Px) ?*

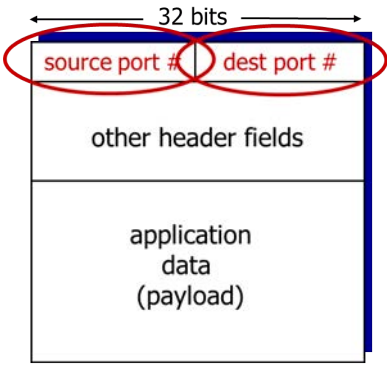
→ Demultiplexing

How demultiplexing works

host receives IP datagrams

- each datagram has source IP address, destination IP address
- each datagram carries one transport-layer segment
- each segment has source, destination port number

host uses *IP addresses & port numbers* to direct segment to appropriate socket



TCP/UDP segment format

Connectionless demultiplexing

Recall:

- when creating socket, must specify *host-local* port #:

```
DatagramSocket mySocket1
 = new DatagramSocket(12534);
```

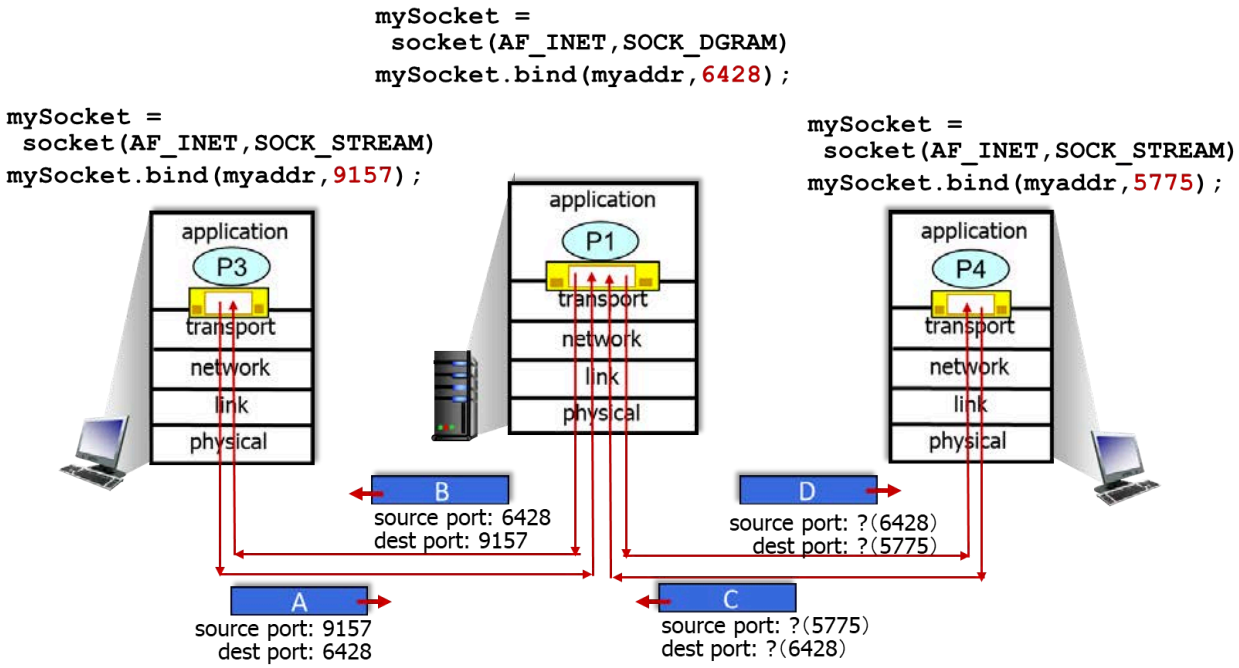
- when creating datagram to send into UDP socket, must specify
  - destination IP address
  - destination port #

when receiving host receives *UDP* segment:

- checks destination port # in segment
- directs UDP segment to socket with that port #

IP/UDP datagrams with *same dest. port #,* but different source IP addresses and/or source port numbers will be directed to *same socket* at receiving host

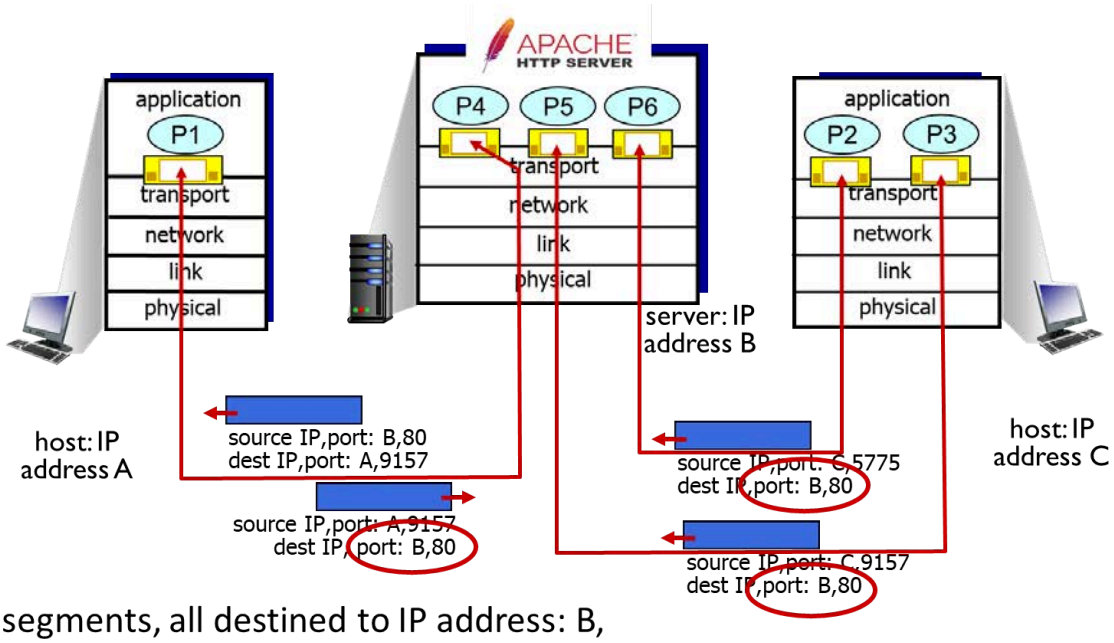
Connectionless (UDP) Demultiplexing



Connection-oriented demultiplexing

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses *all four values (4-tuple)* to direct segment to appropriate socket
- server may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
  - each socket associated with a different connecting client

Connection-Oriented (TCP) Demultiplexing



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

Summary

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

UDP: User Datagram Protocol

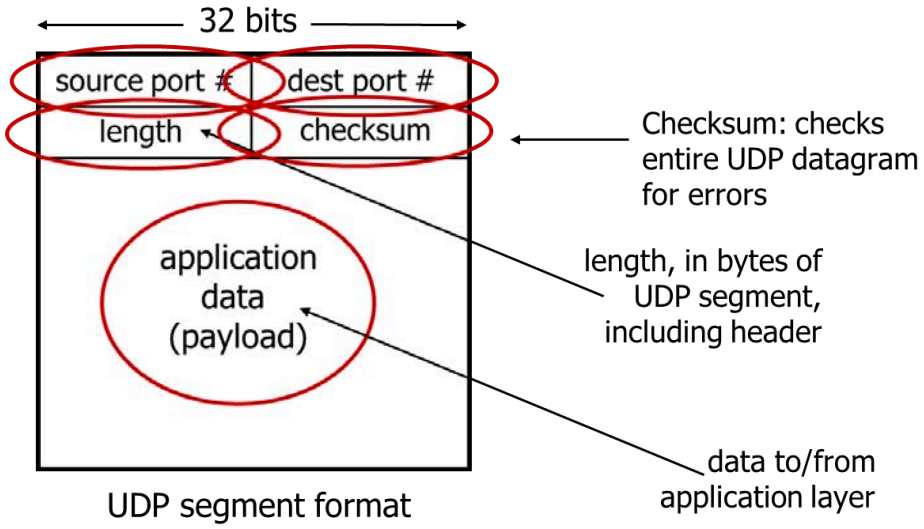
- UDP is a connection-less, unreliable end-to-end transport layer protocol that provides
  - *Process-to-process communications*
  - End-to-end *error* ***checking*** **only**
- UDP **does not** provide for end-to-end *error* or *flow* ***control***
- UDP services is used by
  - Applications that involves short request/response
    - DNS, SNMP, RIP, etc…
  - Applications that can't tolerate connection- setup delay
    - multimedia applications, internet telephony, streaming audio/video, etc...

- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out-of-order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

- no connection establishment (which can add RTT delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control
  - UDP can blast away as fast as desired!
  - can function in the face of congestion

UDP segment header



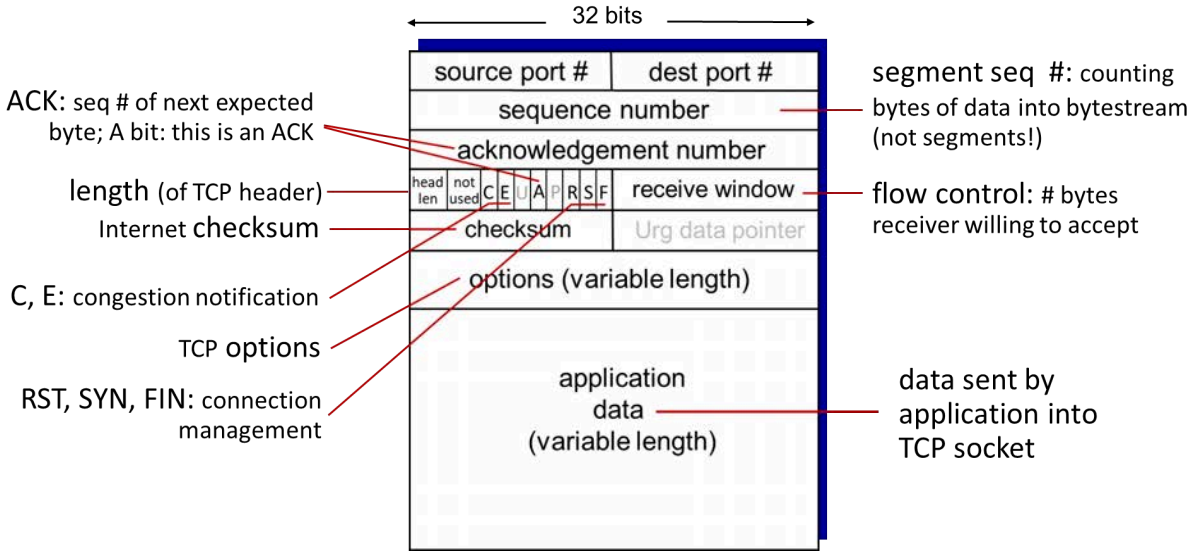
UDP segment format

TCP: Transport Control Protocol

- TCP is a point-to-point, connection-oriented, reliable, end-to-end protocol that provides
  - Process-to-process communications
  - End-to-end error, flow and congestion control
  - FDX service
- TCP services is used by
  - Applications that can tolerate packet losses but can tolerate the additional delay required to set up the logical connection.
    - Such applications include HTTP, SMTP, FTP, TELNET, etc...
- The unit of data using TCP is called a Segment
- TCP is a Byte-Oriented Protocol (No message boundary)

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no “message boundaries"
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- cumulative ACKs
- pipelining:
  - TCP congestion and flow control set window size
- connection-oriented:
  - handshaking (exchange of control messages) initializes sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

TCP Segment Format



TCP sequence numbers, ACKs

Sequence numbers:

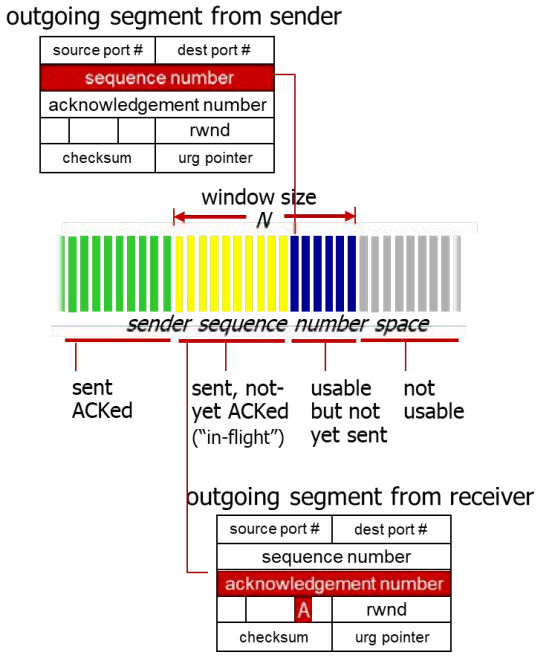
- byte stream "number" of first byte in segment's data

Acknowledgements:

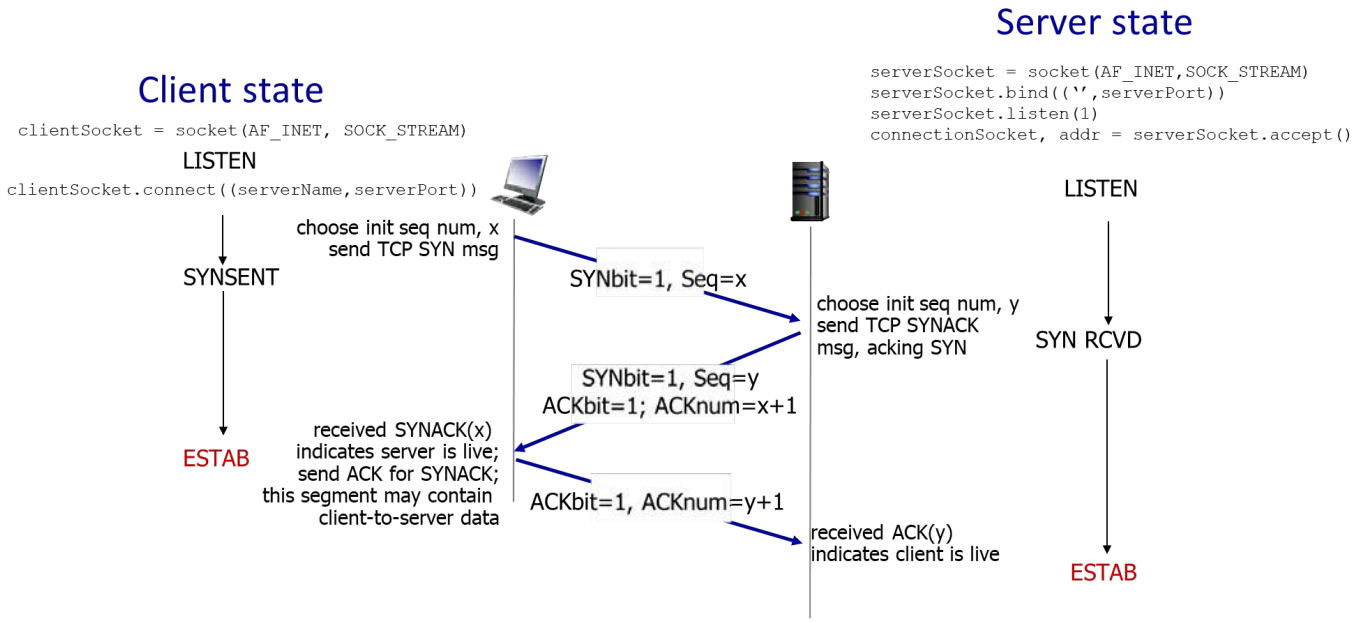
- seq # of next byte expected from other side
- cumulative ACK

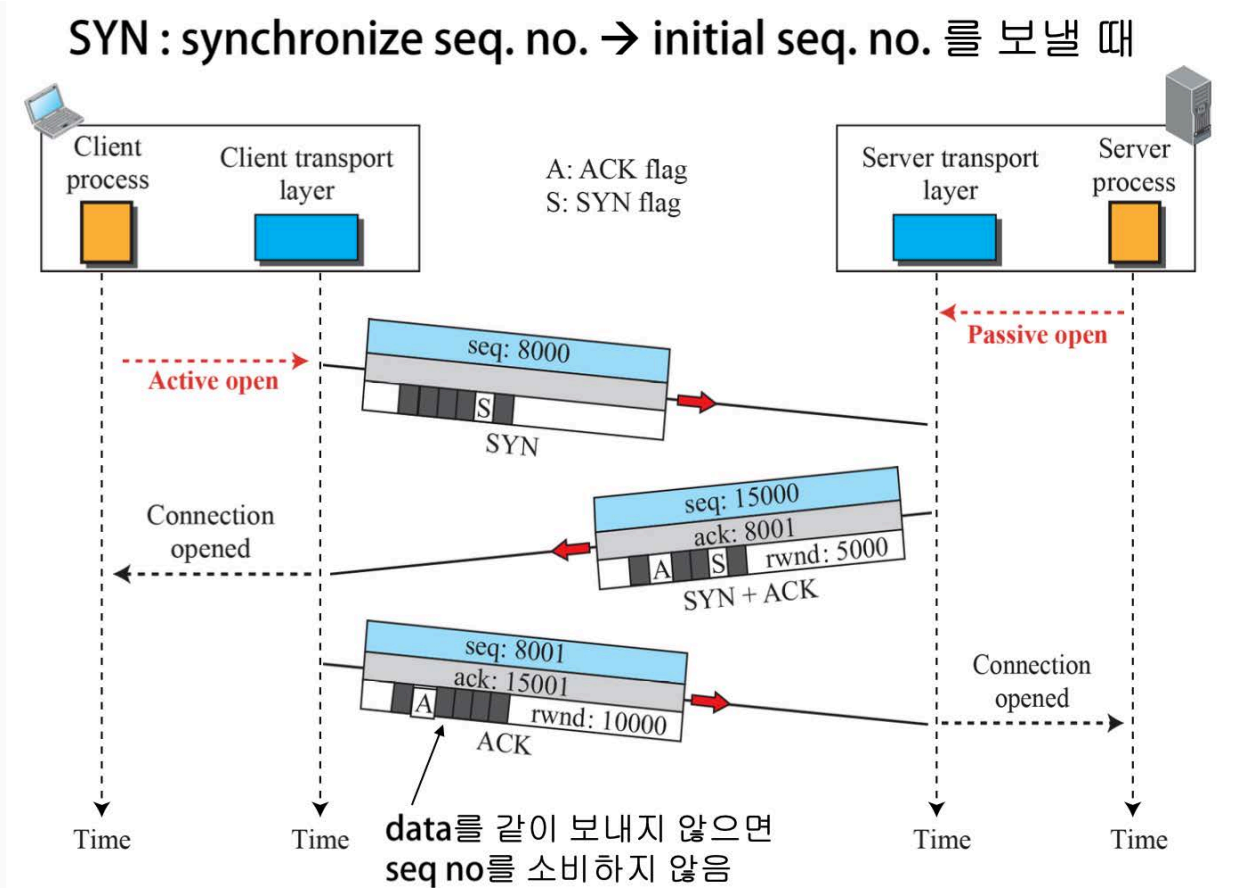
Q: how receiver handles out-of-order segments

- *A:* TCP spec doesn't say, - up to implementor



TCP Connection Set-up





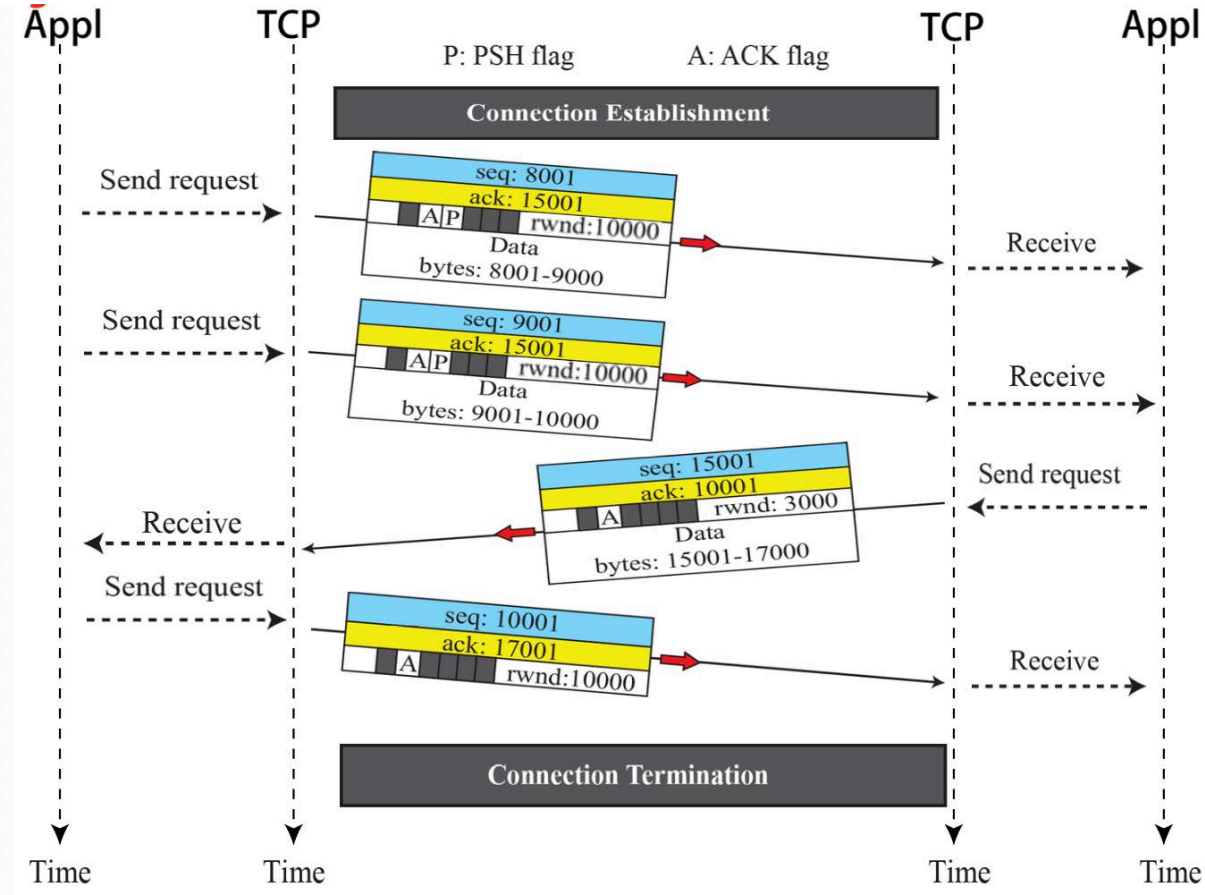
- A SYN segment doesn't carry data, but it consumes one sequence number.
- A SYN/ACK segment doesn't carry data, but it consumes one sequence number
- An ACK segment can carry data

TCP 的连接的拆除需要发送四个包，因此称为四次挥手(Four-way handshake)，也叫做改进的三次握手。客户端或服务器均可主动发起挥手动作，在 socket 编程中，任何一方执行 `close()` 操作即可产生挥手操作。

- 第一次挥手(FIN=1，seq=x)
  假设客户端想要关闭连接，客户端发送一个 FIN 标志位置为1的包，表示自己已经没有数据可以发送了，但是仍然可以接受数据。
  发送完毕后，客户端进入 `FIN_WAIT_1` 状态。
- 第二次挥手(ACK=1，ACKnum=x+1)
  服务器端确认客户端的 FIN 包，发送一个确认包，表明自己接受到了客户端关闭连接的请求，但还没有准备好关闭连接。
  发送完毕后，服务器端进入 `CLOSE_WAIT` 状态，客户端接收到这个确认包之后，进入 `FIN_WAIT_2` 状态，等待服务器端关闭连接。

- 第三次挥手(FIN=1，seq=y)
  服务器端准备好关闭连接时，向客户端发送结束连接请求，FIN 置为1。
  发送完毕后，服务器端进入 `LAST_ACK` 状态，等待来自客户端的最后一个ACK。
- 第四次挥手(ACK=1，ACKnum=y+1)
  客户端接收到来自服务器端的关闭请求，发送一个确认包，并进入 `TIME_WAIT`状态，等待可能出现的要求重传的 ACK 包。
  服务器端接收到这个确认包之后，关闭连接，进入 `CLOSED` 状态。
  客户端等待了某个固定时间（两个最大段生命周期，2MSL，2 Maximum Segment Lifetime）之后，没有收到服务器端的 ACK ，认为服务器端已经正常关闭连接，于是自己也关闭连接，进入 `CLOSED` 状态。

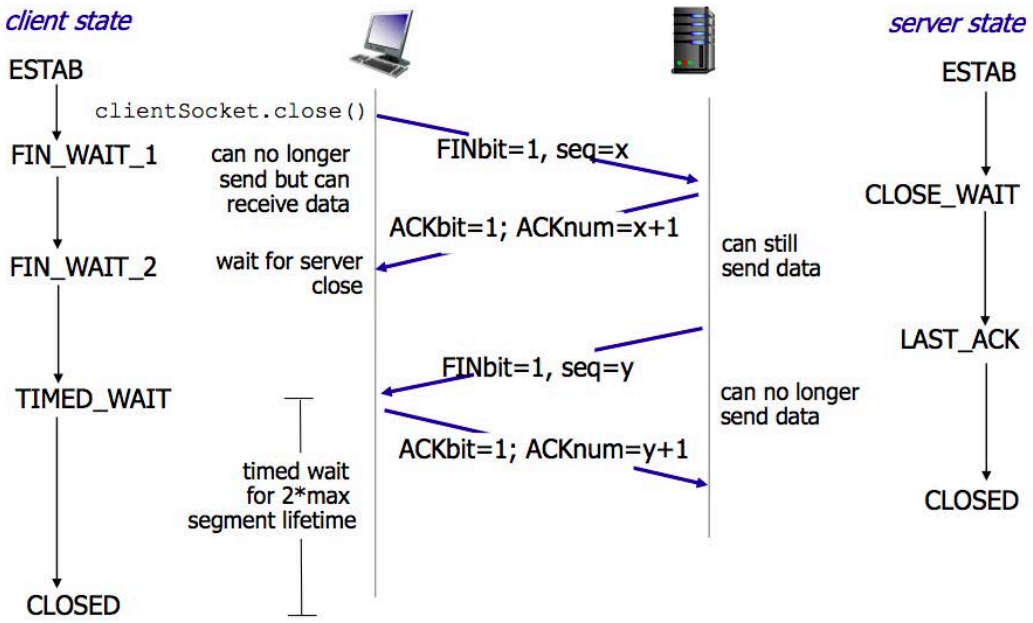
Data Transfer Phase



据传输：连接建立~连接终止

- PUSH标志：TCP接收方收到时设置
- URG（紧急）位 = 设置 URG 位并在段的开头发送紧急数据，并在末尾定义紧急字段。

TCP Connection Termination (四次挥手)

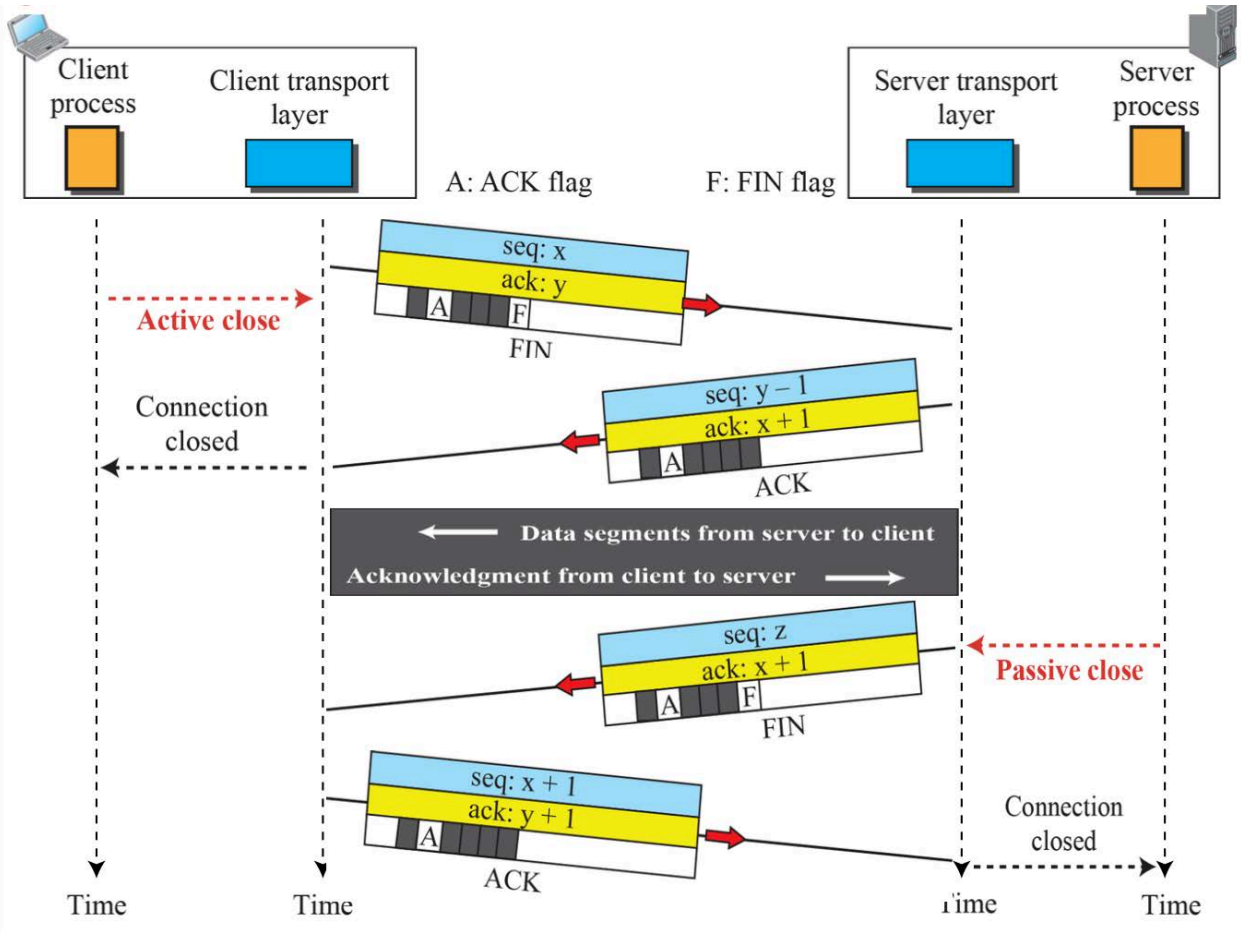


具体过程：

- 客户端打算关闭连接，此时会发送一个 TCP 首部 FIN 标志位被置为 1 的报文，也即 FIN 报文，之后客户端进入 FIN_WAIT_1 状态。
- 服务端收到报文后，就向客户端发送 ACK 应答报文，接着服务端进入 CLOSE_WAIT 状态
- 客户端收到服务端的 ACK 应答报文后，之后进入 FIN_WAIT_2 状态。
- 等待服务端处理完数据后，也向客户端发送 FIN 报文，之后服务端进入 LAST_ACK 状态
- 客户端收到服务端的 FIN 报文后，回一个 ACK 应答报文，之后进入 TIME_WAIT 状态
- 服务端收到了 ACK 应答报文后，就进入了 CLOSE 状态，至此服务端已经完成连接的关闭
- 客户端在经过 2MSL 一段时间后，自动进入 CLOSE 状态，至此客户端也完成连接的关闭

你可以看到，每个方向都需要一个 FIN 和一个 ACK，因此通常被称为四次挥手。

★ Note: 这里一点需要注意是：主动关闭连接的，才有 `TIME_WAIT` 状态。



SYN洪泛攻击（DOS攻击 ）

- 恶意攻击者冒充多个IP的客户端，在短时间内向服务器发送大量SYN报文段的攻击。
- 服务器每次接收到 SYN 段时都会分配资源（创建传输控制块表……）。
- 如果大量的SYN报文进来，服务器的资源就会被耗尽，即使有正常客户端的SYN报文进来，服务也会被拒绝。
- 通过复杂的 ISN 号码进行预防。

Reliability in TCP

- Components of Reliability
  - Sequence Numbers/Acknowledgements
  - Retransmissions
  - Timeout Mechanism(s): function of the round trip time (RTT) between the two hosts (is it static?). How to set TCP timeout value?
    - Longer than RTT, but RTT varies???
    - Too short, but that may mean premature timeouts and hence unnecessary retransmissions
    - Too long, but that means slow reaction to segment losses

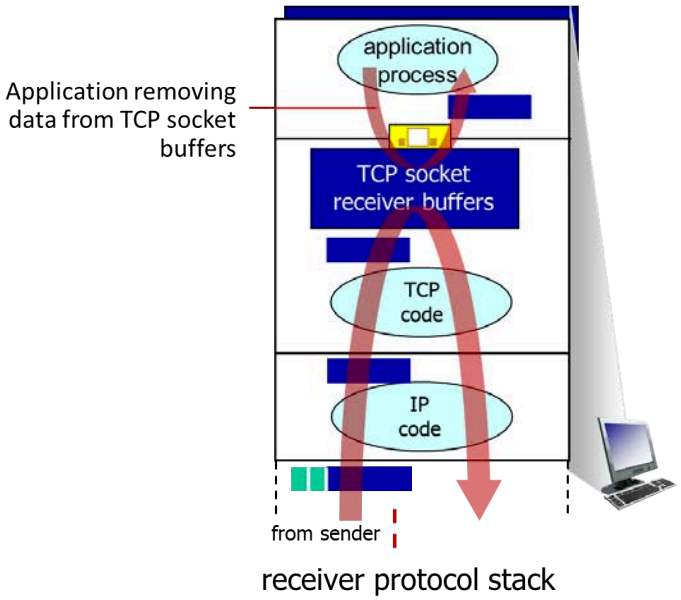
TCP rdt (Reliable Data Transfer)

- TCP creates reliable service on top of IP's unreliable service
- Pipelined segments
- Cumulative ACKs
- TCP uses single retransmission timer
- Retransmissions are triggered by:
  - Timeout events
  - Duplicate ACKs
- Initially consider simplified TCP sender:
  - Ignore duplicate ACKs
  - Ignore flow control, congestion control

TCP flow control

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?

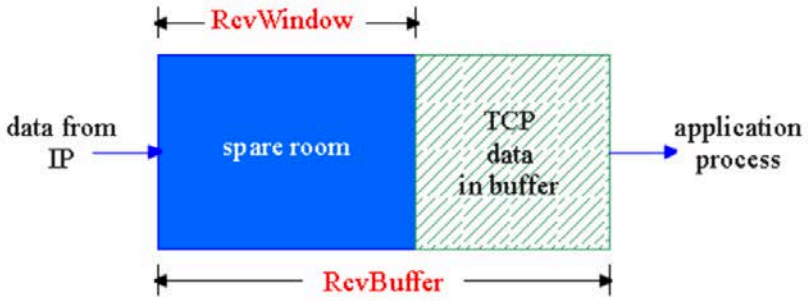
flow control
receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast



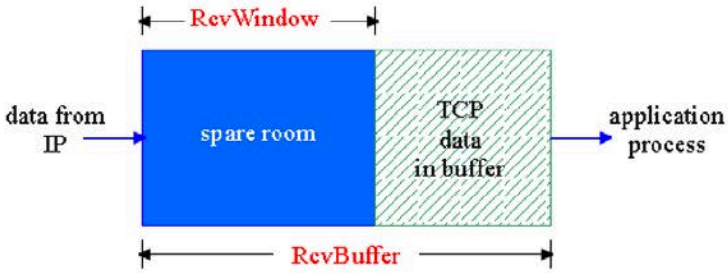
流量控制

- 目的：防止淹没接收方
- 方法：将接收窗口大小捎带方式传递给发送端
  - 接收方在其向发送方的TCP段头部的rwnd字段通告其空闲buffer大小
    - RcvBuffer大小通过socket选项设置（典型默认大小为4096字节）
    - 很多操作系统自动调整RcvBuffer
  - 发送方限制未确认字节个数接收方发送过来的rwnd值

- receiver side of TCP connection has a receive buffer:



- app. process may be slow at reading from buffer
- speed-matching service: matching the send rate to the receiving app's drain rate
  - 缓存中可用的空间
    =RcvWindow
    =RcvBuffer-[LastByteRcvd-LastByteRead]



(Suppose TCP receiver discards out-of-order segments)

- Spare room in buffer

= **RcvWindow**

= **RcvBuffer-[LastByteRcvd - LastByteRead]**

- Receiver advertise spare room by including value of **RcvWindow** in segments
- Sender limits un-ACKed data to **RcvWindow** ⇒ No overflow

Congestion Control

Congestion:

- informally: “too many sources sending too much data too fast for *network* to handle”
- Manifestations:
  - long delays (queueing in router buffers)
  - packet loss (buffer overflow at routers)
- different from flow control!
- a top-10 problem!

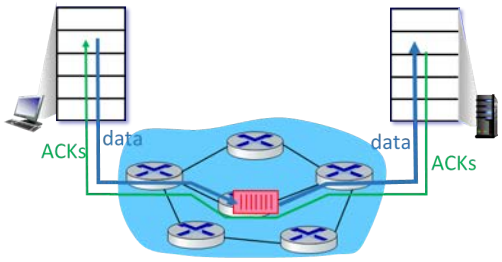
congestion control: too many senders, sending too fast

flow control: one sender too fast for one receiver

Approaches towards congestion control (Appendix.)

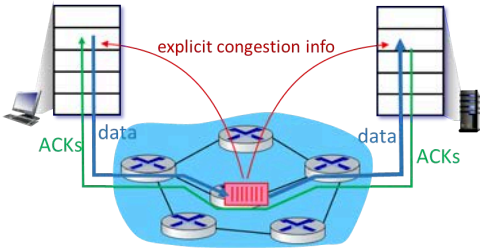
End-end congestion control:

- no explicit feedback from network
- congestion *inferred* from observed loss, delay
- approach taken by TCP



Network-assisted congestion control:

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

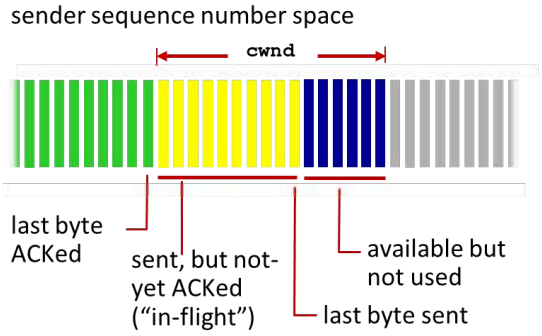


wa , wc Advertised window, Congestion window

- The receiver window (**advertised window**, w_a) ensures that receiver buffer will never overflow, however it does not guarantee that buffers in intermediate routers will not overflow (congestion) 由接收方的可用缓冲区大小决定
- IP does not provide any mechanism for congestion control. It is up to TCP to detect congestion
- **congestion window**, w_c : determines the maximum number of bytes that can be transmitted without congesting the network 是发送方基于网络状况自行维护的窗口大小

⭐ **Max # of bytes that can be sent =** $min\ (w_a, w_c)$

TCP congestion control: details (Appendix.)



TCP sending behavior:

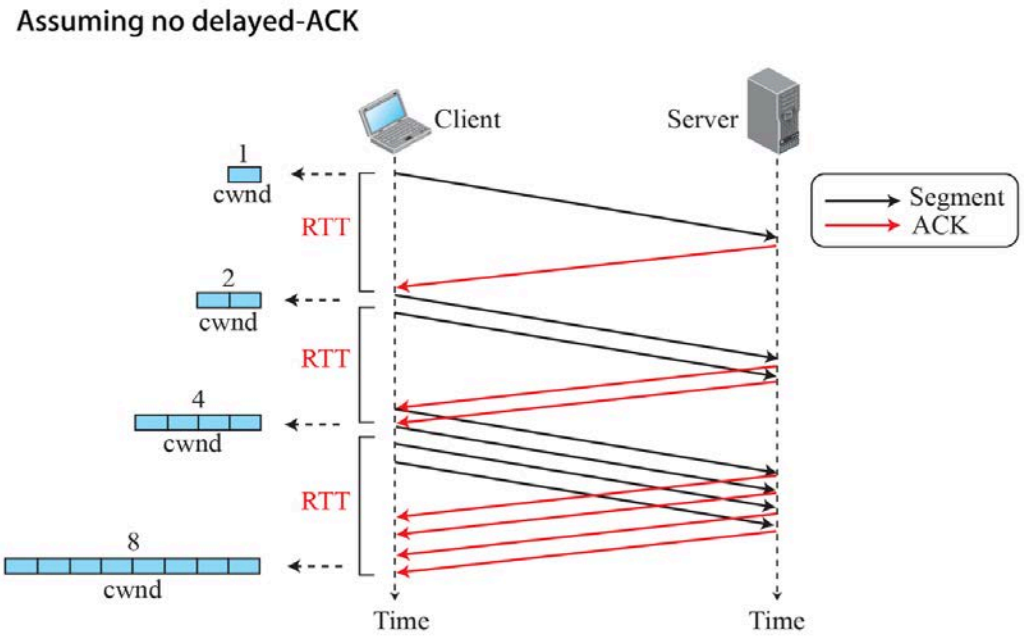
- *roughly:* send `cwnd` bytes, wait RTT for ACKS, then send more bytes

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

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

TCP slow start (Tahoe)

- **Phase 1**: Start by setting the congestion window, wc to one MSS. Each time the sender receives an ACK it increases its congestion window by one and so on.
  - Hence, $wc = wc + 1$ **for every ACK received**.
  - This phase is referred to as the "***Slow Start Phase***".
    - In SS the congestion window increases ***exponentially***
- **Phase 2**: As the congestion window reaches a threshold value, the congestion window starts to increase ***linearly***.
  - This phase is referred to ***Congestion Avoidance Phase***.
  - In this phase the congestion window is increased by one segment every RTT, i.e. $wc = wc + (1/wc)$ **for every ACK received**



- TCP连接建立后，第一个cwnd值以1MSS开始。（MSS大小由发送方和接收方协商确定）
- 此后，每收到一个ACK，cwnd的值就增加1个MSS。
- 此时rwnd值会很大，所以发送窗口的大小最初是由cwnd的值决定的。
- 在 TCP 连接开始时，cwnd 的值每个 RTT 都会加倍（指数增加），如图所示。
- 应用慢启动算法，直到 cwnd 的值达到慢启动阈值 (ssthresh)。

  ★ Note: 也就是说cwnd增加了1倍，但是从RTT角度来看，增加了2倍。

Phase 1 慢启动

TCP 在刚建立连接完成后，首先是有个慢启动的过程，这个慢启动的意思就是一点一点的提高发送数据包的数量，如果一上来就发大量的数据，这不是给网络添堵吗？

慢启动的算法记住一个规则就行：当发送方每收到一个 **ACK**，拥塞窗口 **cwnd** 的大小就会加 **1**。

这里假定拥塞窗口 cwnd 和发送窗口 swnd 相等，下面举个栗子：

- 连接建立完成后，一开始初始化 cwnd = 1，表示可以传一个 MSS 大小的数据。
- 当收到一个 ACK 确认应答后，cwnd 增加 1，于是一次能够发送 2 个
- 当收到 2 个的 ACK 确认应答后，cwnd 增加 2，于是就可以比之前多发 2 个，所以这一次能够发送 4 个
- 当这 4 个的 ACK 确认到来的时候，每个确认 cwnd 增加 1，4 个确认 cwnd 增加 4，于是就可以比之前多发 4 个，所以这一次能够发送 8 个。

可以看出慢启动算法，发包的个数是指数性的增长。

那慢启动涨到什么时候是个头呢？

有一个叫慢启动门限 ssthresh（slow start threshold）状态变量。

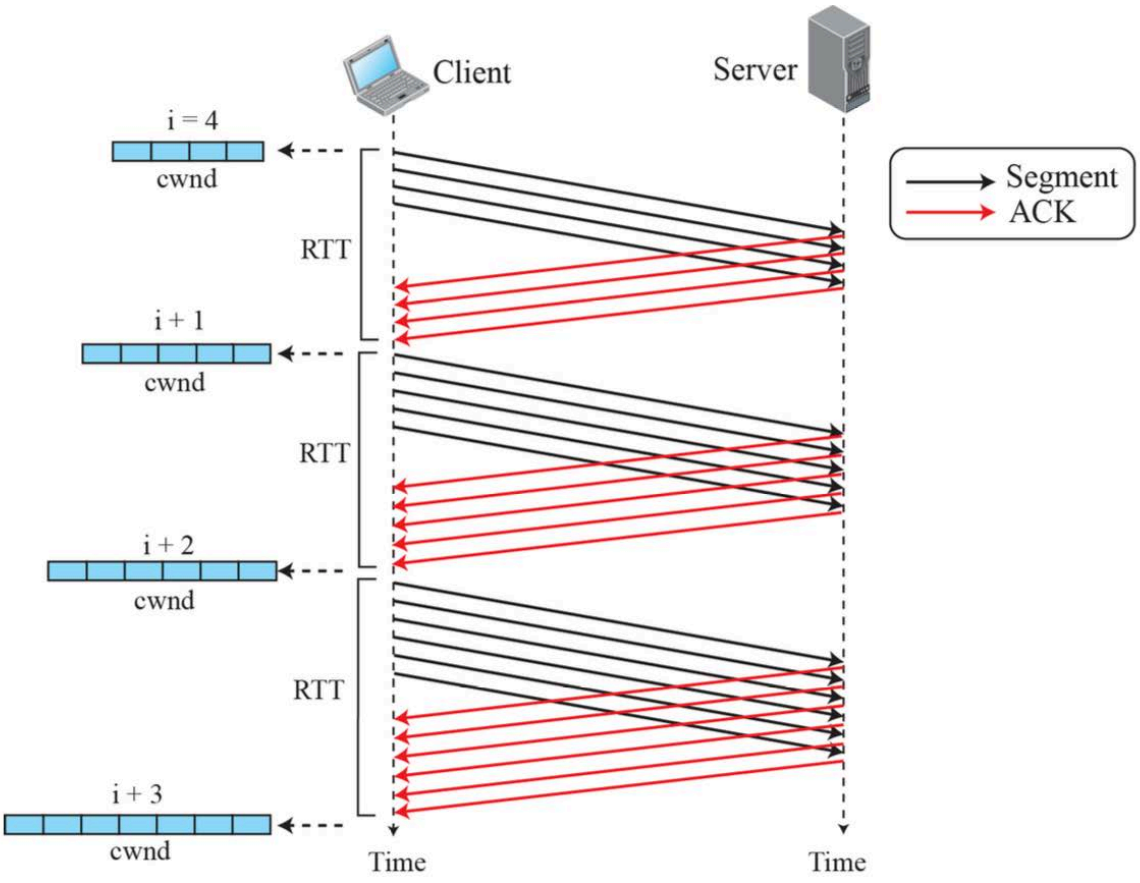
- 当 cwnd < ssthresh 时，使用慢启动算法。
- 当 cwnd >= ssthresh 时，就会使用「拥塞避免算法」。

Phase 2 拥塞避免算法

如果cwnd的值无限制地呈指数增长，传输速度最终会变得太大，从而导致网络拥塞。

为了提前避免这种情况（避免拥塞），在 ssthresh（阈值）之后，每次收到 ACK 时，cwnd 的值都会增加（1/ cwnd）。

这相当于当整个发送窗口收到ACK时，每个RTT将cwnd的值加1。结果，如图所示，cwnd 的值在每个 RTT 处增加 1，并且当绘制为时间的函数时，cwnd 的值呈现出以加性增加的线性增加。



拥塞避免算法

前面说道，当拥塞窗口 cwnd「超过」慢启动门限 ssthresh 就会进入拥塞避免算法。

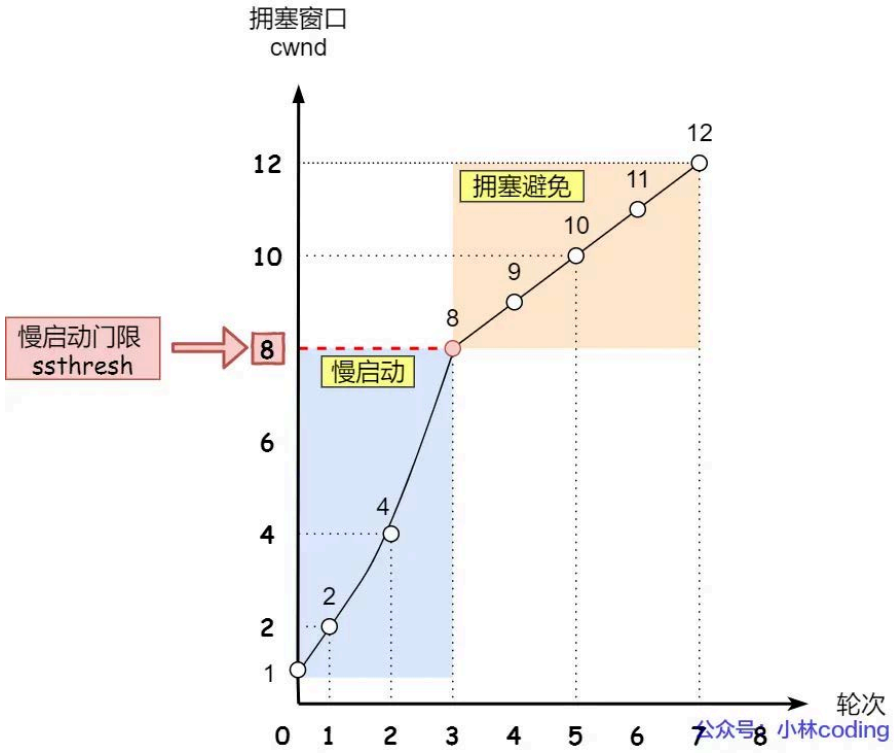
- 一般来说 ssthresh 的大小是 65535 字节。

那么进入拥塞避免算法后，它的规则是：**每当收到一个 ACK 时，cwnd 增加 1/cwnd**。

接上前面的慢启动的栗子，现假定 ssthresh 为 8：

- 当 8 个 ACK 应答确认到来时，每个确认增加 1/8，8 个 ACK 确认 cwnd 一共增加 1，于是这一次能够发送 9 个 MSS 大小的数据，变成了线性增长。

拥塞避免算法的变化过程如下图：



所以，我们可以发现，拥塞避免算法就是将原本慢启动算法的指数增长变成了线性增长，还是增长阶段，但是增长速度缓慢了一些。

就这么一直增长着后，网络就会慢慢进入了拥塞的状况了，于是就会出现丢包现象，这时就需要对丢失的数据包进行重传。

当触发了重传机制，也就进入了「拥塞发生算法」。

- **Phase 3:** The congestion window stops increasing when the client TCP detects the network is congested. This happens when an ACK doesn't arrive before the time-out expires. In this phase the congestion threshold is set to 1/2 the current window size which is the min (wa , wc). The congestion window is then reset to one segment and the slow start phase is repeated.

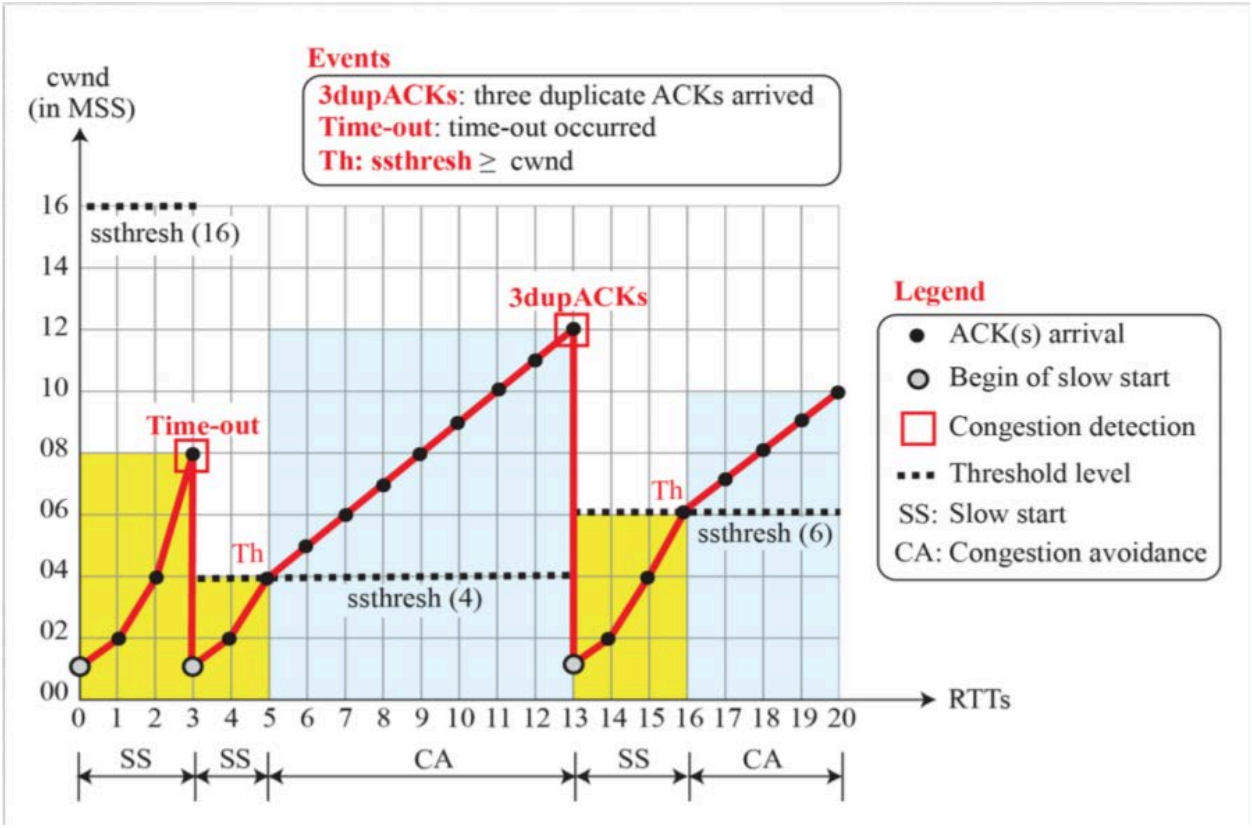
  This phase is referred to as **Congestion Control**

Phase 3: Congestion Control

TCP 中的经典拥塞

- **TCP Tahoe** (1988):慢启动算法(SS)、拥塞避免算法(CA)
- **TCP Reno** (1990):SS、CA、快速恢复 (FR) -> 最常见的版本之一

🌰: TCP Tahoe



Fast Retransmission/Recovery

Fast retransmit:

- receiver sends Ack with last in-order segment for every out-of-order segment received
- when sender receives 3 duplicate ACKs it retransmits the missing/expected segment

Fast recovery: when 3rd dup Ack arrives

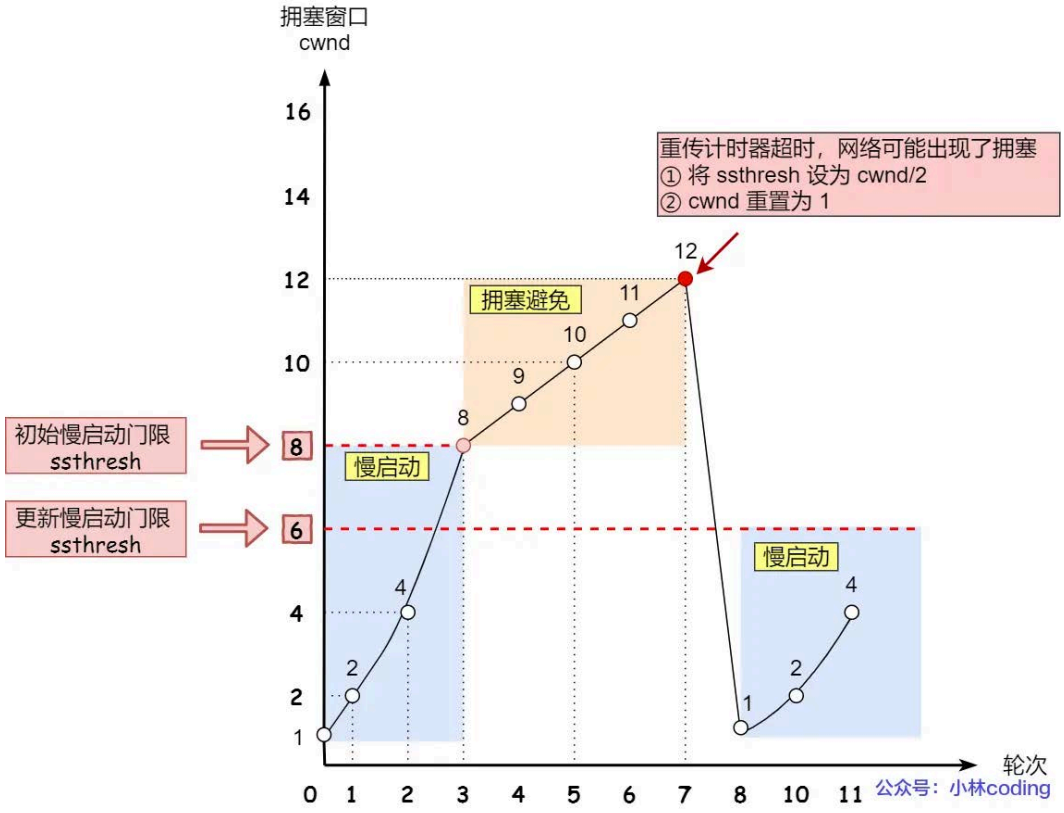
- ssthresh=CongWin/2
- retransmit segment, set CongWin=ssthresh
- Enter congestion avoidance phase, i.e. skip SS

当网络出现拥塞，也就是会发生数据包重传，重传机制主要有两种：

- 超时重传
- 快速重传 → 快速恢复

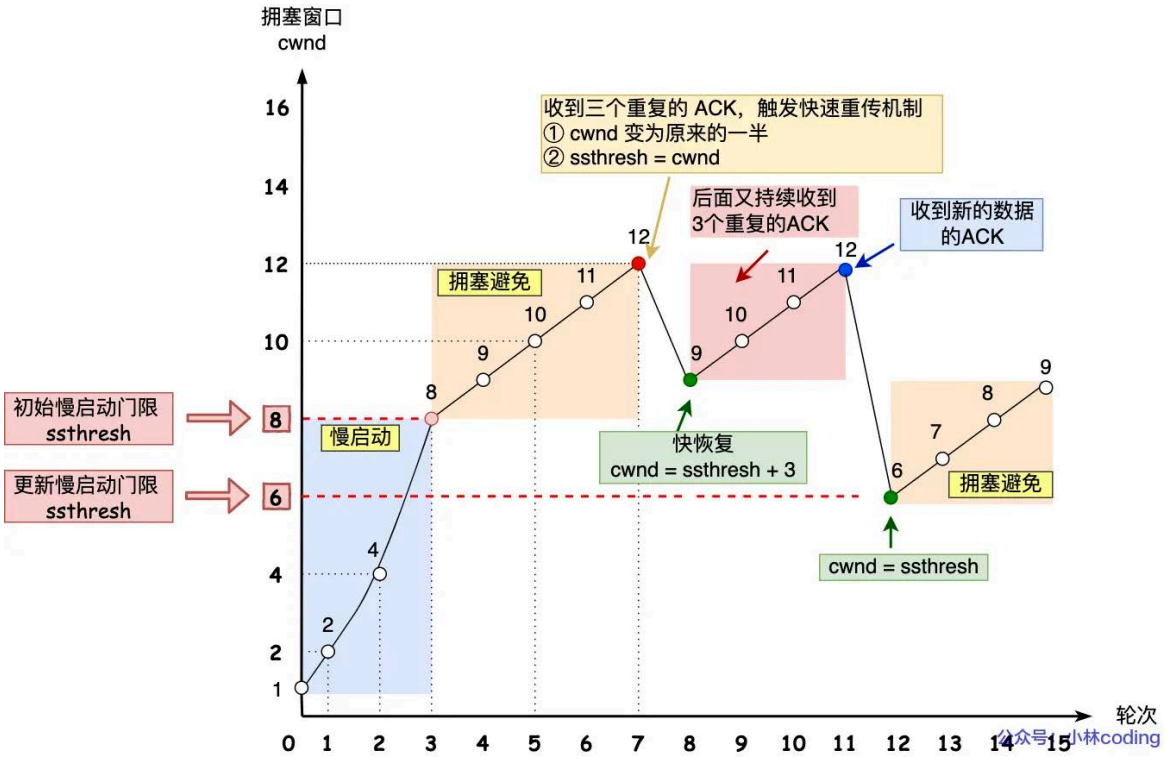
1. 超时重传

- 当发生了「超时重传」，则就会使用拥塞发生算法。
- 这个时候，ssthresh 和 cwnd 的值会发生变化：
  - ssthresh 设为 cwnd/2，
  - cwnd 重置为 1（是恢复为 cwnd 初始化值，我这里假定 cwnd 初始化值 1）



2. 快速重传 → 快速恢复

- 快速重传和快速恢复算法一般同时使用，快速恢复算法是认为，你还能收到 3 个重复 ACK 说明网络也不那么糟糕，所以没有必要像 RTO 超时那么强烈。
- 正如前面所说，进入快速恢复之前，cwnd 和 ssthresh 已被更新了：
  - cwnd = cwnd/2 ，也就是设置为原来的一半;
  - ssthresh = cwnd;
- 然后，进入快速恢复算法如下：
  - 拥塞窗口 cwnd = ssthresh + 3（3 的意思是确认有 3 个数据包被收到了）；
  - 重传丢失的数据包；
  - 如果再收到重复的 ACK，那么 cwnd 增加 1；
  - 如果收到新数据的 ACK 后，把 cwnd 设置为第一步中的 ssthresh 的值，原因是该 ACK 确认了新的数据，说明从 duplicated ACK 时的数据都已收到，该恢复过程已经结束，可以回到恢复之前的状态了，也即再次进入拥塞避免状态；

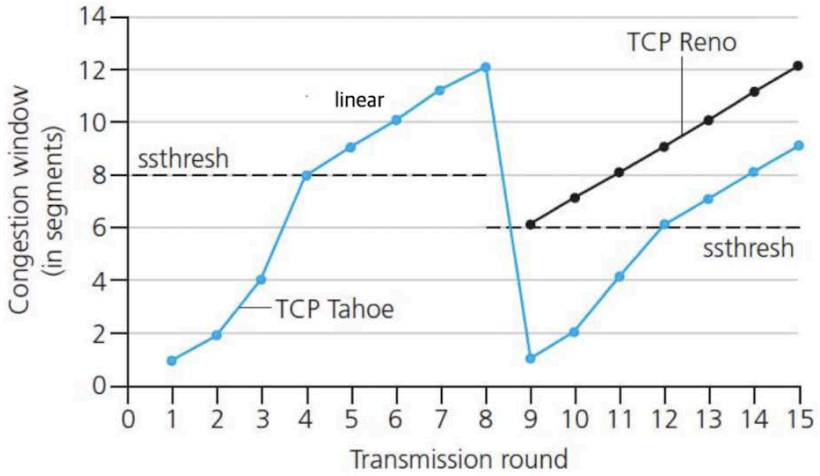


TCP Reno

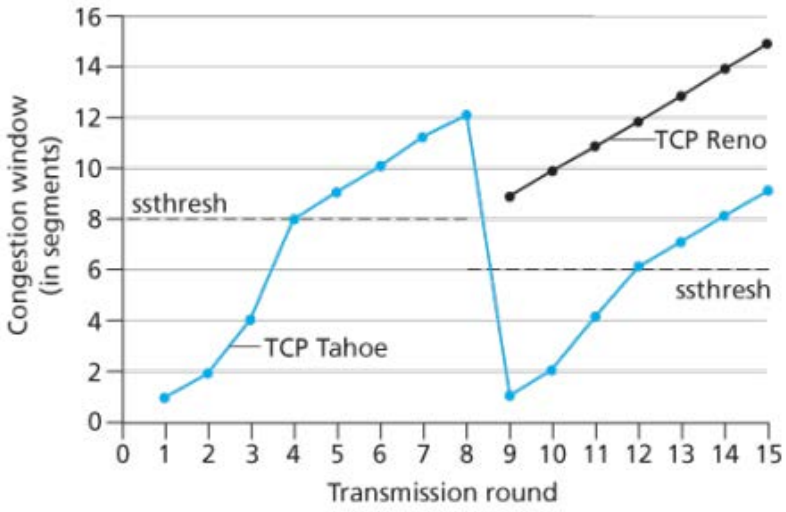
- Tahoe：超时事件和3个冗余ACK处理一样（无快速恢复）
  - TCP Tahoe cwnd = 1,
  - 无条件将CongWin减至1

- Reno：超时事件和3个冗余ACK处理不一样
  - Timeout:
    - TCP Tahoe 和 TCP Reno 一样：cwnd = 1
  - 3个冗余ACK
    - TCP Reno cwnd = ssthresh = cwnd / 2　→ 之前发生拥塞的点的1/2

  - 超时事件（丢包）：Threshold=CongWin/2，CongWin=1MSS，进入SS阶段
  - **3个冗余ACK**：快速重传，Threshold=CongWin/2，CongWin=Threshold + 3MSS



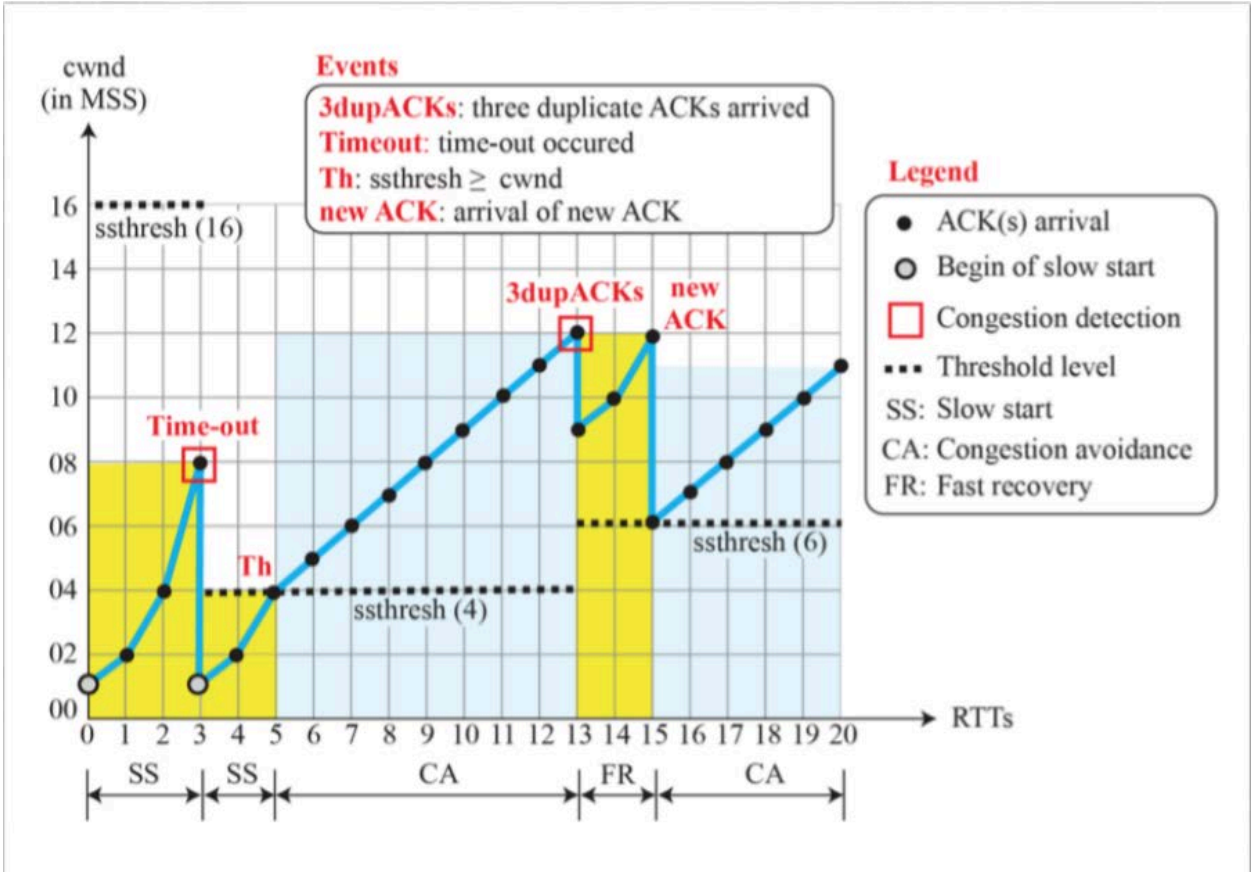
（超时事件（丢包）Threshold=CongWin/2）



（3个冗余ACK　Threshold=CongWin/2 + 3MSS）

- 3 dup ACKs indicates network capable of delivering some segments, Network is not that badly congested
- Timeout indicates a “more alarming” congestion scenario

🌰: TCP Reno



该数据展示了Reno TCP的拥塞控制过程。

减少到 6MSS（与 Taho TCP 相同），但将 cwnd 设置为高于 1MSS 的值 (ssthresh + 3 = 9MSS)。
Reno TCP 转换到快速恢复状态。此外，我们假设两个重复的 ACK 随 RTT 15 到达，并且 cwnd 呈指数增长。
此时，一个新的 ACK（非重复）到达，宣布新的丢失段。
Reno TCP 切换到拥塞避免状态。但首先，我们忽略快速恢复状态，将拥塞窗口减少到6MSS，比如在之前的轨道上向后移动。

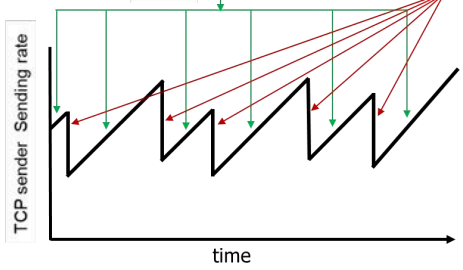
AIMD Approach (No SS)

TCP congestion control: AIMD

- *approach:* senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event

Additive Increase
increase sending rate by 1 maximum segment size every RTT until loss detected

Multiplicative Decrease
cut sending rate in half at each loss event



AIMD sawtooth behavior: *probing* for bandwidth

11.12 Network Security

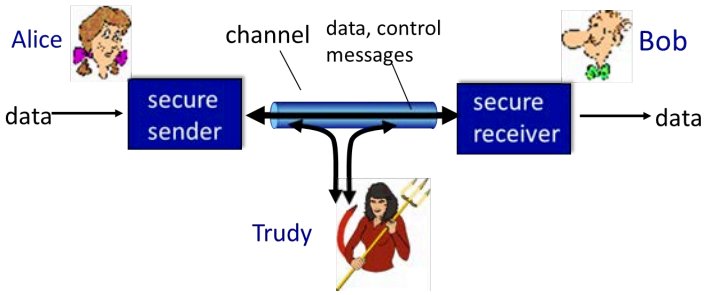
⭐ Final: Only multi-choice questions

What is network security?

- **confidentiality**: only sender, intended receiver should "understand" message contents
  - sender encrypts message
  - receiver decrypts message
- **authentication**: sender, receiver want to confirm identity of each other
- **message integrity**: sender, receiver want to ensure message not altered (in transit, or afterwards) without detection
- **access and availability**: services must be accessible and available to users

Scenario: Friends and enemies: Alice, Bob, Trudy

- well-known in network security world
- Bob, Alice (lovers!) want to communicate "securely"
- Trudy (intruder) may intercept, delete, add messages

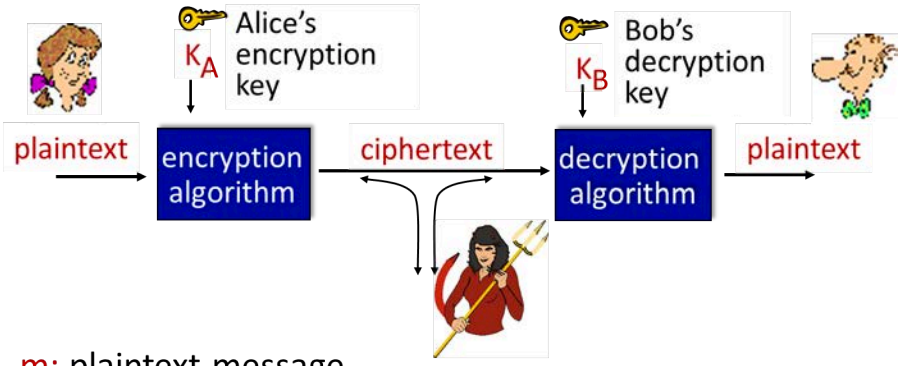


Q: What can a "bad guy" do?

A: A lot! (recall section 1.6)

- eavesdrop: intercept messages 窃听
- actively insert messages into connection
- impersonation: can fake (spoof) source address in packet (or any field in packet)
- hijacking: "take over" ongoing connection by removing sender or receiver, inserting himself in place
- denial of service: prevent service from being used by others (e.g., by overloading resources)

Cryptography



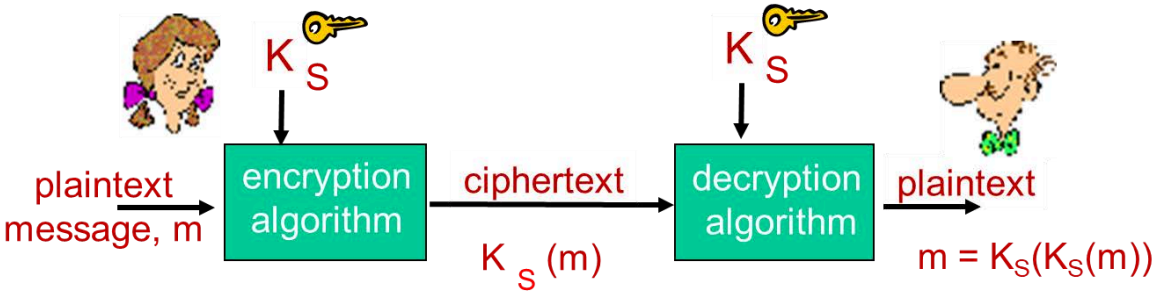
m: plaintext message

$K_A(m)$: ciphertext, encrypted with key K_A

$m = K_B(K_A(m))$

- ***Symmetric key*** crypto: sender, receiver keys identical
- ***Public-key*** crypto: encryption key public, decryption key secret (private)

1. Symmetric key cryptography



symmetric key crypto: Bob and Alice share same (symmetric) key: K_S

- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher

Q: how do Bob and Alice agree on key value?

A: Ex: Data Encryption Standard, DES

Simple encryption scheme

- *substitution cipher:* substituting one thing for another
  - mono-alphabetic cipher: substitute one letter for another

```
plaintext:  abcdefghijklmnopqrstuvwxyz
            |                       |
ciphertext: mnbvcxzasdfghjklpoiuytrewq
```

e.g.:

```
Plaintext: bob. i love you. alice
ciphertext: nkn. s gktc wky. mgsbc
```

Encryption key: mapping from set of 26 letters to set of 26 letters

2. Public Key Cryptography

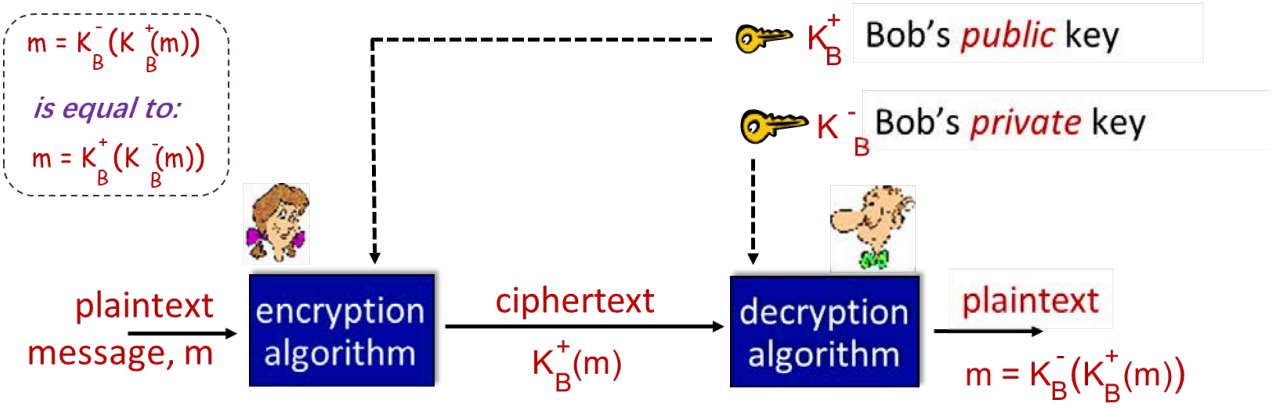
symmetric key crypto:

- requires sender, receiver know shared secret key
- Q: how to agree on key in first place (particularly if never "met")?

public key crypto

- *radically* different approach [Diffie-Hellman76, RSA78]
- sender, receiver do *not* share secret key
- *public* encryption key known to *all*
- *private* decryption key known only to receiver

Public Key (Asymmetric) Cryptography

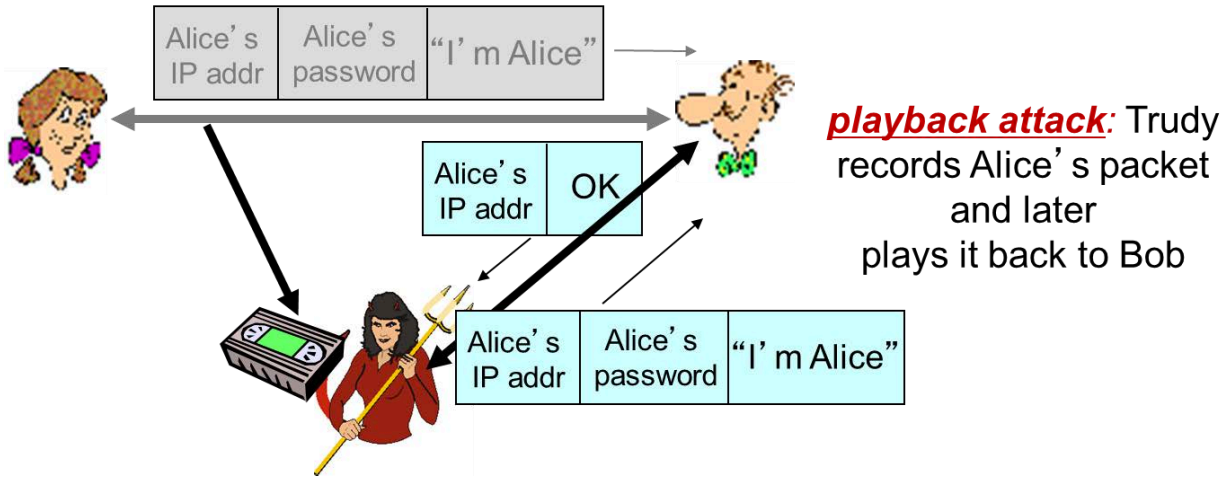


Sender, *Receiver* **do not** share secret key **public** encryption key known to **all private** decryption key known only to receiver

- Ex: Rivest, Shamir, Adelson (RSA) Algorithm

Playback Attack?

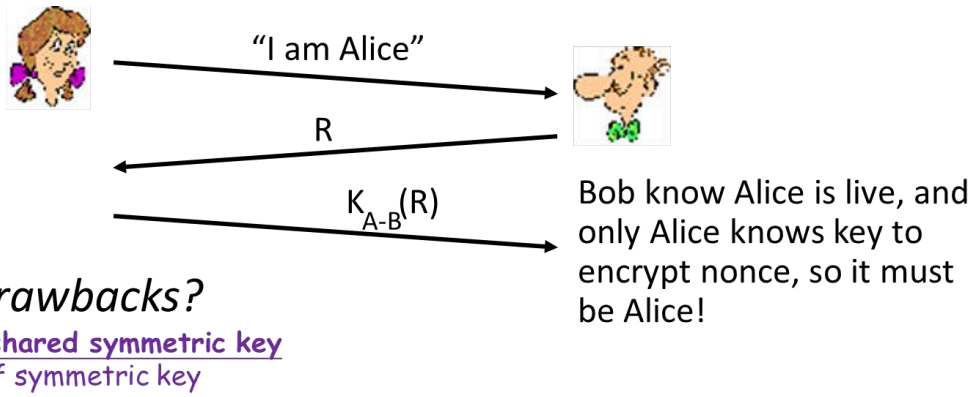
Even if Alice encrypt her password, playback attack is still applicable



playback attack: Trudy records Alice's packet and later plays it back to Bob

Authentication using a Nonce

- **Goal**: avoid playback attack
- **nonce**: number (R) used only ***once-in-a-lifetime***
- protocol ap4.0: to prove Alice "live", Bob sends Alice *nonce*, ***R***.
    - Alice must return ***R***, encrypted with shared secret key

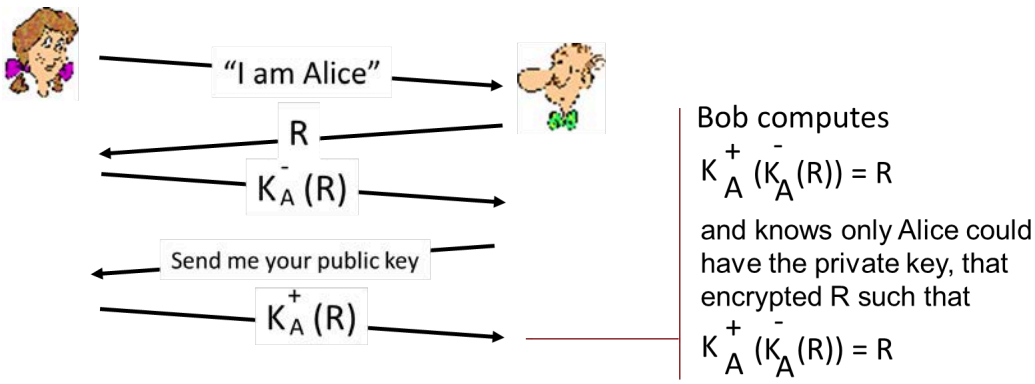


Failures, drawbacks?

ap4.0 requires **shared symmetric key**
→ drawback of symmetric key

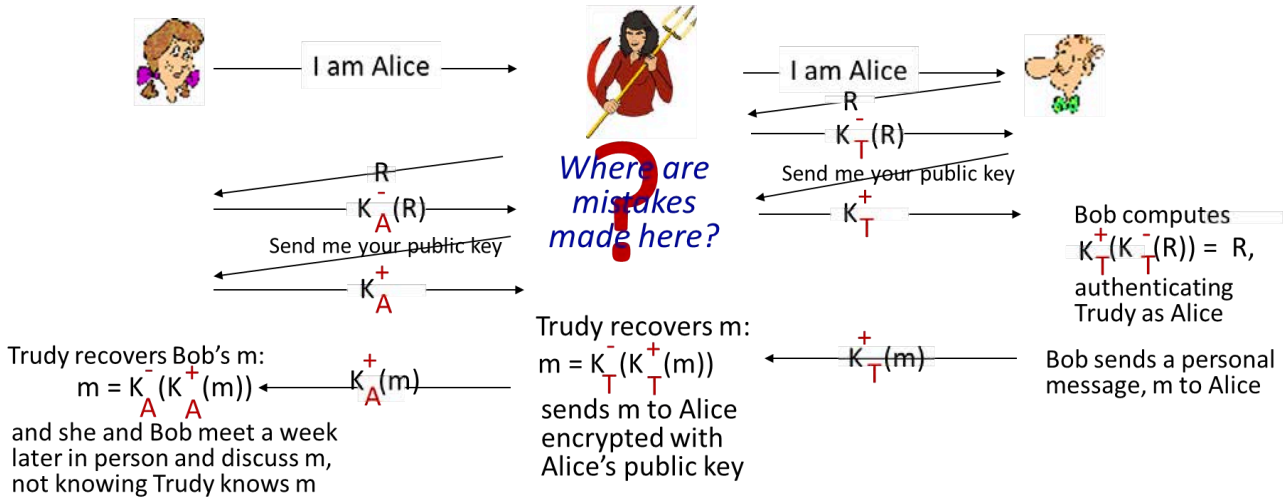
Ap5.0: Nonce Authentication with Public Key

- ap4.0 requires ***shared symmetric key***
    - can we authenticate using public key techniques?
- ap5.0: use ***nonce***, ***public key*** cryptography



ap5.0 – still a flaw!

man (or woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



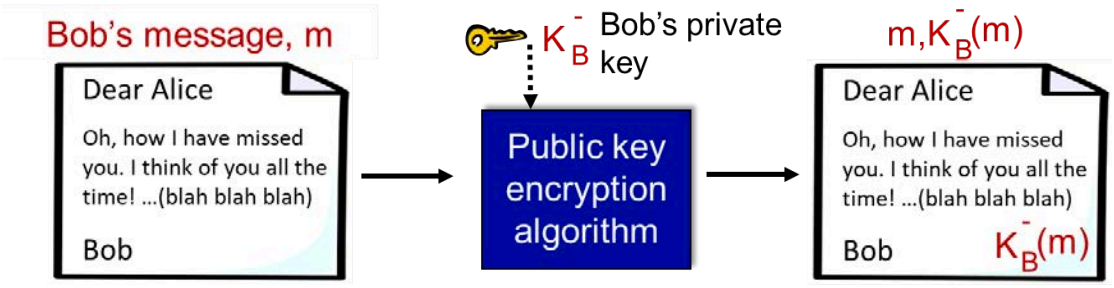
Message Integrity: Digital Signature

cryptographic technique analogous to hand-written signatures:

- sender (Bob) digitally signs document: he is document owner/creator.
- *verifiable, nonforgeable:* recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document

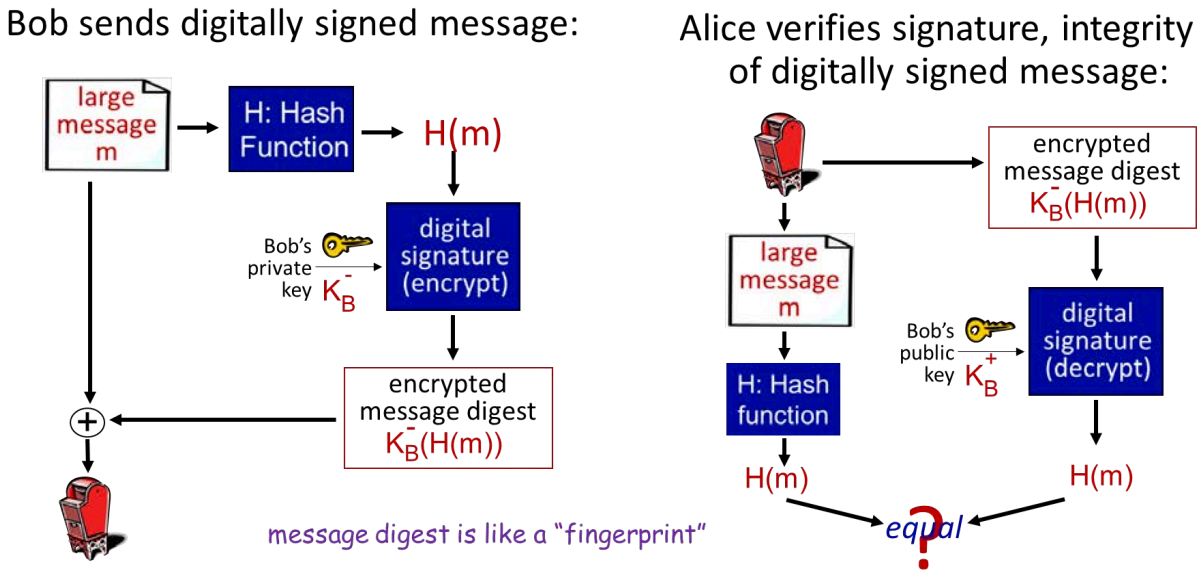
simple digital signature for message m:

- Bob signs m by encrypting with his private key K_B, creating "signed" message, $K_B^-(m)$



⭐ Digital signature **does not** provide privacy.
If there is a need for privacy, another layer of encryption/decryption must be applied.

Digital signature = signed message digest



Alice can verify that:

✅Bob signed ***m***.

✅No one else signed ***m***.

✅Bob signed ***m*** and not ***m'***.

Non-repudiation:

✅Alice can take ***m***, and signature $K_B^-(m)$ to court and prove that Bob signed ***m***.

⭐ how Alice can know Bob's hash function $H(m)$?

(chatGPT)

1. Standardized Hash Functions

- Many systems specify a standard hash function (e.g., SHA-256) for digital signatures. If both parties operate within the same protocol or system, Alice already knows the hash function.

2. Certificate Information

- Bob's public key may be included in a digital certificate issued by a trusted Certificate Authority (CA). These certificates often specify the hash function used alongside the key pair for signing.

3. Message Metadata

- Bob can include information about the hash function in the message metadata or signature block. This way, Alice can determine which H(m)H(m)H(m) to use for verification.

4. Pre-Shared Agreement

- If Alice and Bob have a pre-established agreement on the hash function H(m)H(m)H(m), this information is predetermined and doesn't need to be transmitted with every message.

Key Problems

***Symmetric key* problem:**

- How do two entities establish shared secret key over network?

Solution:

- trusted **key distribution center** (**KDC**) acting as intermediary between entities

***Public key* problem:**

- When Alice obtains Bob's public key (from web site, e-mail, diskette), how does she know it is Bob's public key, not Trudy's?

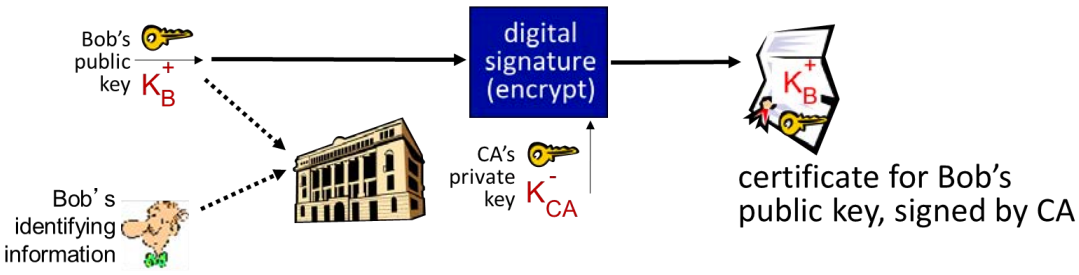
  ★ Note: Trudy 可以用他自己的 private key 来做 signature，然后发送给 Alice 他自己的 public key，并说这是 Bob's public key。这样 Alice 同样可以 verifies signature，并以为是 Bob 的 signature.

Solution:

- trusted **certification authority** (**CA**)

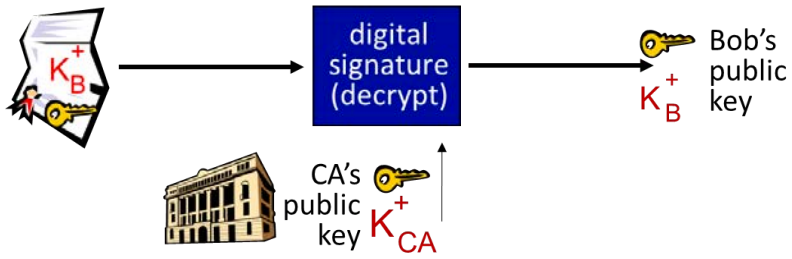
Public key CA (Certification Authorities)

- certification authority (CA): binds public key to particular *entity*, *E*
- *entity* (person, website, router) registers its public key
  - *E* provides "proof of identity" to CA
  - CA creates certificate binding identity *E* to *E's* public key
  - certificate containing E's public key digitally signed by CA: CA says "this is *E's* public key"



when Alice wants Bob's public key:

- gets Bob's certificate (Bob or elsewhere)
- apply CA's public key to Bob's certificate, get Bob's public key



★ Note:

- CA不是用来交换公钥的，而是用来"证明公钥属于声称的拥有者"。这解决了在不安全信道上进行身份认证的问题。 → 防止中间人攻击 (***the middle attack***)

⭐ Q. 如何防止中间人攻击 (***the middle attack***) ？

1. 没有CA时的问题：
   - 假设Alice直接从Bob那里获得公钥
   - 攻击者Eve可以截获通信，假装自己是Bob
   - Eve可以发送自己的公钥给Alice，声称"这是Bob的公钥"
   - Alice无法确认收到的真的是Bob的公钥，还是Eve的
2. CA提供的解决方案：
   - CA作为可信任的第三方机构会严格验证Bob的身份（域名所有权、组织资质等）
   - CA用自己的私钥给Bob的公钥签名，生成证书
   - 所有浏览器/操作系统预装了CA的公钥
   - Alice可以用CA的公钥验证证书，从而确认Bob的公钥确实属于Bob
3. 为什么这样更安全：
   - 攻击者Eve即使截获通信也无法伪造证书
   - 因为Eve没有CA的私钥，无法伪造CA的签名
   - CA的信任是预先建立的（通过预装的根证书 ）
   - 形成了一个可信任的链条：CA → Bob → Alice

实际防护过程：

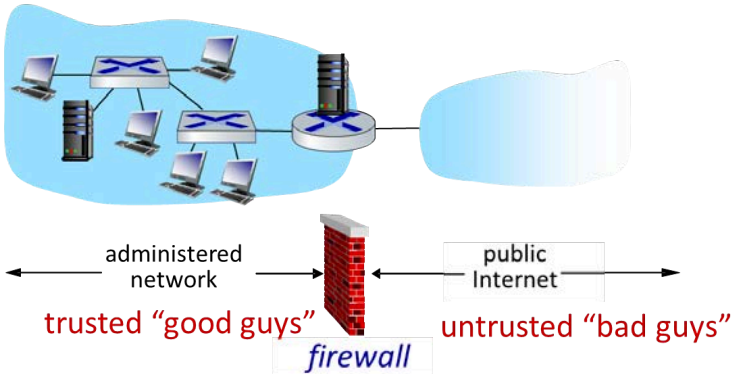
- 当Alice连接到"声称是Bob"的服务器时：
  - 服务器提供证书
  - Alice的浏览器：检查证书
    1. 检查证书是否由受信任的CA签发
    2. 验证CA的签名是否有效
    3. 检查证书的域名是否匹配
    4. 检查证书是否在有效期内
    5. 查询证书是否被吊销

- 如果Eve在中间：
  - Eve无法提供有效的证书（因为无法获得CA对伪证书的签名）
  - Alice的浏览器会显示证书错误警告
  - TLS连接将无法建立

Firewalls

firewall

isolates organization's internal network from larger Internet, allowing some packets to pass, blocking others



Why necessary:

prevent denial of service attacks:

- **SYN flooding**: attacker establishes many bogus TCP connections, no resources left for "real" connections

prevent illegal modification/access of internal data

- e.g., attacker replaces CIA's homepage with something else

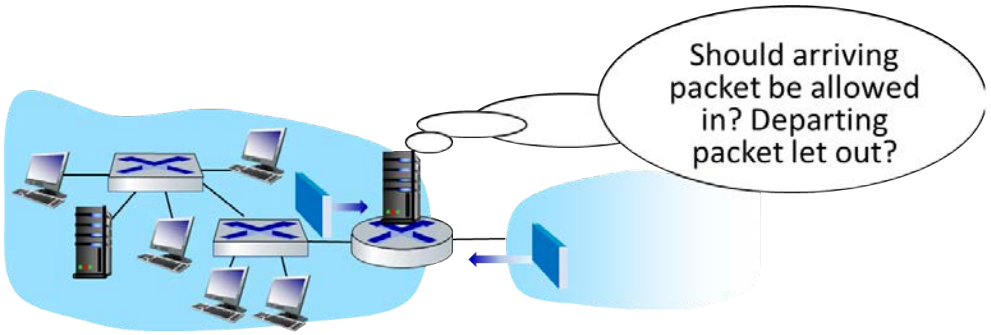
allow only authorized access to inside network

- set of authenticated users/hosts

three types of firewalls:

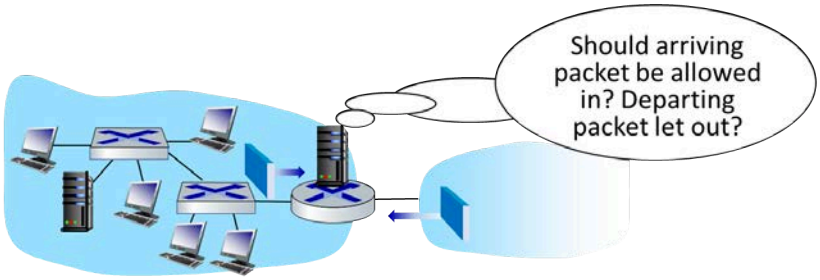
- stateless packet filters
- stateful packet filters
- application gateways

Stateless packet filtering



@ the network Or transport layer

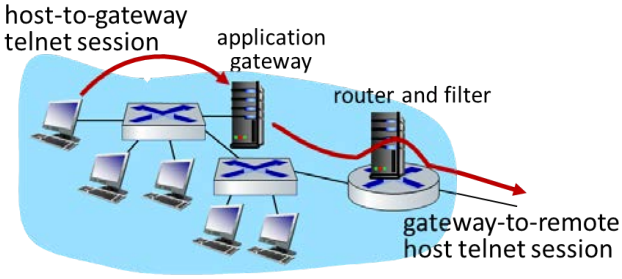
- internal network connected to Internet via router firewall
- filters packet-by-packet, decision to forward/drop packet based on:
  - *source IP* address, *destination IP* address
  - *TCP/UDP source*, *destination port numbers*
  - *ICMP message type*
  - *TCP SYN, ACK* bits



- example 1: block incoming and outgoing datagrams with IP protocol field = 17 and with either source or dest port = 23
  - result: all incoming, outgoing UDP flows and telnet connections are blocked
- example 2: block inbound TCP segments with ACK=0
  - result: prevents external clients from making TCP connections with internal clients, but allows internal clients to connect to outside

Application gateways

- filter packets on application data as well as on IP/TCP/UDP fields.
- *example:* allow select internal users to telnet outside



1. require all telnet users to telnet through gateway.
2. for authorized users, gateway sets up telnet connection to dest host
   - gateway relays data between 2 connections
3. router filter blocks all telnet connections not originating from gateway

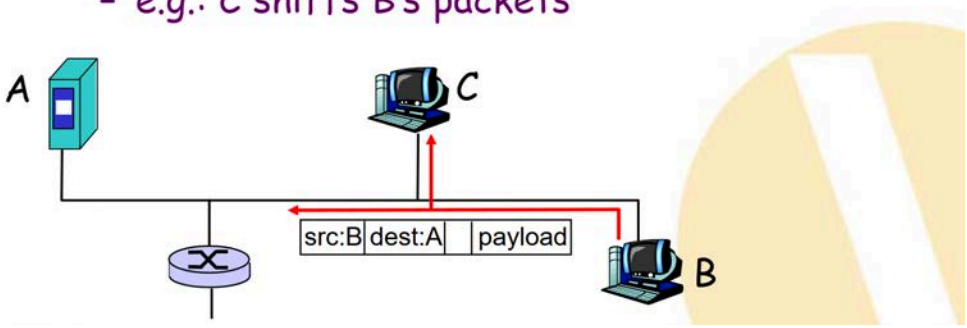
Limitations of firewalls, gateways

- **IP spoofing**: router can't know if data "really" comes from claimed source
- if multiple apps need special treatment, each has own app. gateway
- client software must know how to contact gateway
  - e.g., must set IP address of proxy in Web browser
- filters often use all or nothing policy for UDP
- ***tradeoff:*** degree of communication with outside world, level of security
- many highly protected sites still suffer from attacks

Security Threats

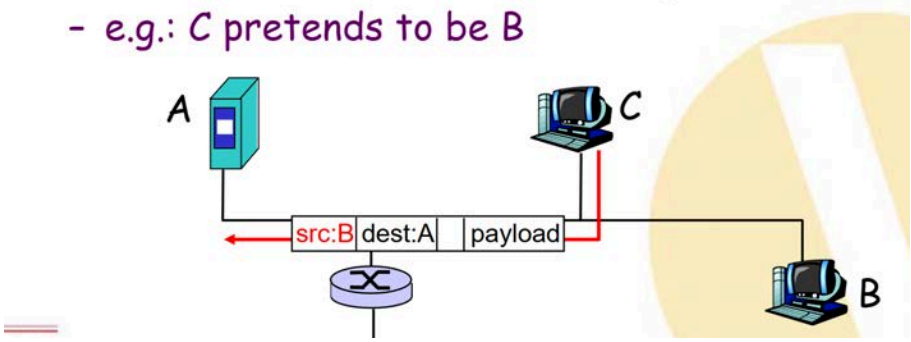
Packet sniffing 数据包嗅探:

- Packet sniffing:
  - Broadcast Traffic
  - Promiscuous NIC reads all Packets passing by
  - Can read all unencrypted data (e.g. passwords)
  - e.g.: C sniffs B's packets



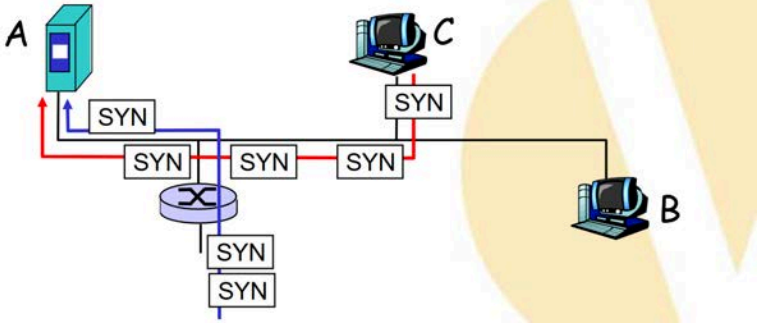
IP Spoofing IP欺骗:

- IP Spoofing:
  - can generate "raw" IP packets directly from application, putting any value into IP source address field
  - receiver can't tell if source is spoofed
  - e.g.: C pretends to be B

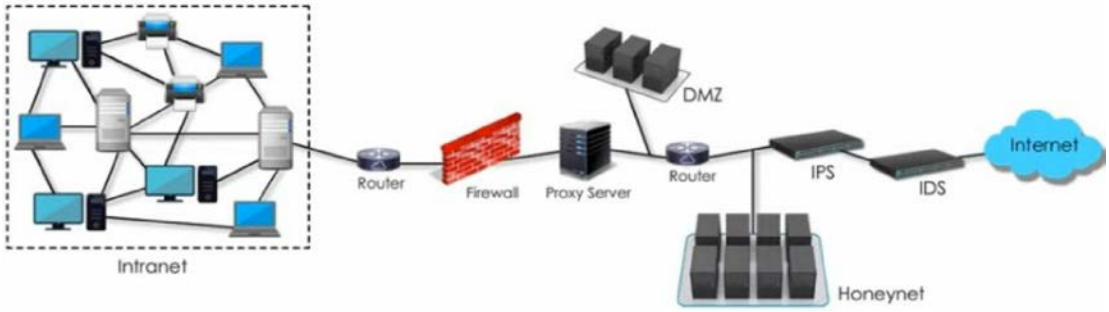


Denial of service (DOS):

- Denial of service (DOS):
  - Flood of maliciously generated packets "swamp" receiver
  - Distributed DOS (DDOS): multiple coordinated sources swamp receiver
  - e.g., C and remote host SYN-attack A



11.19 Security Devices



DMZ (sometimes referred to as a "**demilitarized zone**")

- a subnetwork containing an organization's exposed, outward-facing services. It acts as the exposed point to an untrusted network, commonly the internet.
  - ***Least*** secure
- 作用:
  - an extra layer of security to the computer network by restricting remote access to internal servers and information, which can be very damaging if breached.

Honeynet

- A honeynet is a network set up with intentional vulnerabilities hosted on a decoy server to attract hackers. The primary purpose is to test network security by inviting attacks.
- 蜜网是一种故意设置漏洞的网络，托管在诱饵服务器上以吸引黑客。其主要目的是通过诱使攻击来测试网络安全。这种方法可以帮助安全专家研究实际攻击者的活动和方法，以提高网络安全性。

IDS (**Intrusion detection systems**)

- packet filtering:
  - operates on TCP/IP headers only
  - no correlation check among sessions
- IDS: intrusion detection system
  - DPI (deep packet inspection): look at packet contents (e.g., check character strings in packet against database of known virus, attack strings)
  - examine correlation among multiple packets
    - port scanning, network mapping, DoS attack

IPS (**intrusion Prevention system**)

- IPS 是一种网络安全设备或软件，用于实时监测网络流量，主动阻止潜在的网络攻击。IPS 不仅能像 IDS 一样检测到入侵，还能采取措施阻止攻击

简单类比:

- IDS像监控摄像头:只能观察和报警
- IPS像保安:不仅能发现问题，还能直接采取行动

性能影响: IPS 在主动阻止攻击时可能会消耗更多的系统资源，影响网络性能。

11.19 VPN

⭐ Final: Only multi-choice questions

VPN: Virtual Private Network

- Virtual Private Network (VPN) is defined as network connectivity deployed on a shared infrastructure with the same policies and security as a private network.
- A VPN can be between two end systems, or it can be between two or more networks.
- A VPN can be built using tunnels and encryption. VPNs can occur at any layer of the TCP/IP protocol stack.
- A VPN is an alternative WAN infrastructure that replaces or augments existing private networks that use leased-line or enterprise-owned networks

- ***Advantages***:
    - Leased lines are secured
    - Privacy and QoS Guaranteed
- ***Disadvantages***
    - Leased lines are very expensive
    - No of links required grows exponentially if full mesh connectivity is required and network expands.
    - More CPE ports are required
    - Network complexity increases as network grows. All existing sites require reconfiguration in case of a new site addition.

Critical Functions of VPN

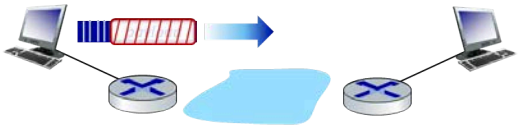
VPNs provide three critical functions:

- ***Confidentiality*** (***encryption***)
    - The sender can encrypt the packets before transmitting them across a network.
    - By doing so, no one can access the communication without permission. If intercepted, the communications cannot be read.
- ***Data integrity***
    - The receiver can verify that the data was transmitted through the Internet without being altered.
- ***Origin authentication***
    - The receiver can authenticate the source of the packet, guaranteeing and certifying the source of the information.
- ***Access Control:***
    - Restricting unauthorized users from the network

11.9 IP Sec (Network Security)

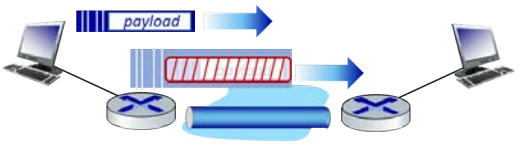
IP Sec

- provides datagram-level encryption, authentication, integrity
  - for both user traffic and control traffic (e.g., BGP, DNS messages)
- two “modes”:



transport mode:

- *only* datagram *payload* is encrypted, authenticated

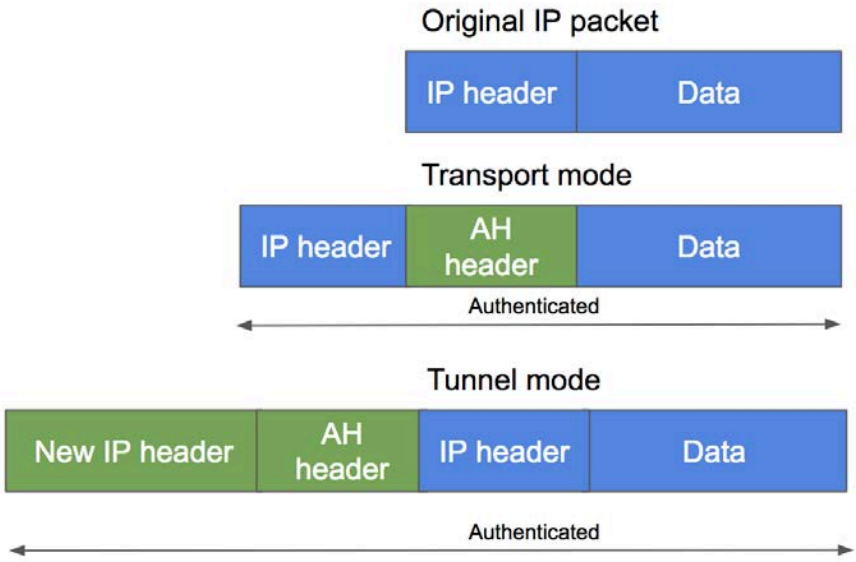


tunnel mode:

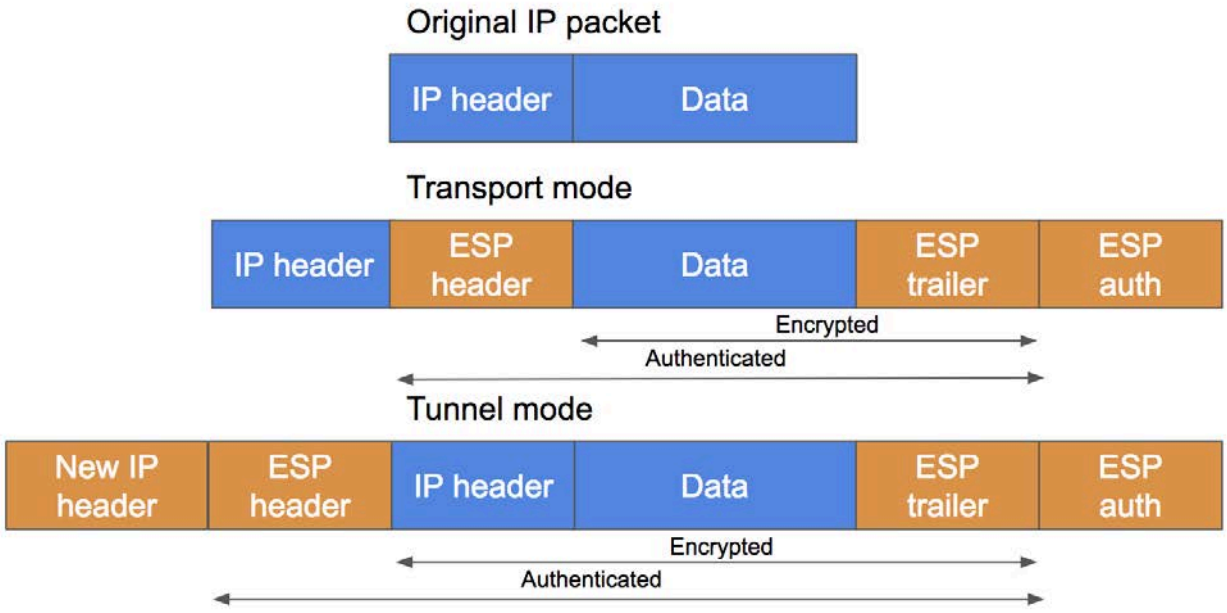
- entire datagram is encrypted, authenticated
- encrypted datagram encapsulated in new datagram with new IP header, tunneled to destination

Two IPsec protocols

- **AH** (Authentication Header) protocol [RFC 4302]
  - provides source authentication & data integrity but *not* confidentiality
- **ESP** (Encapsulation Security Protocol) [RFC 4303]
  - provides source authentication, data integrity, *and confidentiality*
  - more widely used than *AH*



AH (Authentication Header)



ESP (Encapsulation Security Protocol)

Transport mode vs. Tunnel mode

- Transport mode 和 Tunnel mode 都可以使用 **AH** 和 **ESP** 任意一种
- Transport mode: 通常是直接建立在兩台主機上，因為不需要再多加一個 IP header，整體來說較省頻寬，在這個模式下，兩邊的主機都要安裝 IPSec 的 protocal，而且不能隱藏主機的 IP 位置。
- Tunnel mode: 針對 Firewall 或是 Gateway proxy，一般來說我們會用這個模式，因為他們不是原本的發送收端。

11.9 TLS (Network Security)

TLS (Transport-layer security)

- widely deployed security protocol above the transport layer
  - supported by almost all browsers, web servers: https (port 443)
- provides:
  - confidentiality: via *symmetric encryption*
  - integrity: via *cryptographic hashing*
  - authentication: via *public key cryptography*

  } *all techniques we have studied!*
- history:
  - early research, implementation: secure network programming, secure sockets
  - secure socket layer (SSL) deprecated [2015]
  - TLS 1.3: RFC 8846 [2018]

TLS 的改进之处：

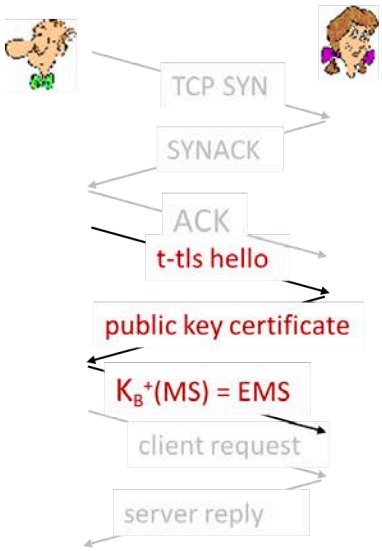
1. 强制使用强加密算法：
   - 弃用了弱加密算法（如 RC4 和 MD5），引入了更强的加密标准（如 AES-GCM 和 SHA-256）。
2. 握手协议增强：
   - TLS 提供更安全的握手过程（如加入 HMAC），防止中间人攻击和重放攻击。
3. 支持前向保密（**Forward Secrecy**）：
   - TLS 支持临时密钥（如使用 Diffie-Hellman 密钥交换），即使密钥泄露，历史数据仍然安全。
4. 版本更新：
   - **TLS 1.2** 和 **TLS 1.3** 添加了更多的安全功能和性能改进（如减少握手轮次）。

what's needed for TSL ?

- let's *build* a toy TLS protocol, *t-tls,* to see what's needed!
- we've seen the "pieces" already:
  - handshake: Alice, Bob use their certificates, private keys to authenticate each other, exchange or create shared secret
  - key derivation: Alice, Bob use shared secret to derive set of keys
  - data transfer: stream data transfer: data as a series of records
    - not just one-time transactions
  - connection closure: special messages to securely close connection

t-tls: initial handshake

t-tls: initial handshake



t-tls handshake phase:

- Bob establishes TCP connection with Alice
- Bob verifies that Alice is really Alice
- Bob sends Alice a master secret key (MS), used to generate all other keys for TLS session
- potential issues:
  - 3 RTT before client can start receiving data (including TCP handshake)

t-tls: encrypting data

- recall: TCP provides data *byte stream* abstraction
- Q: can we encrypt data in-stream as written into TCP socket?
  - *A:* where would MAC go? If at end, no message integrity until all data received and connection closed!
  - *solution:* break stream in series of "records"
    - each client-to-server record carries a MAC, created using M_c
    - receiver can act on each record as it arrives
- t-tls record encrypted using symmetric key, K_c, passed to TCP:

- possible attacks on data stream?
  - *re-ordering:* man-in middle intercepts TCP segments and reorders (manipulating sequence #s in unencrypted TCP header)
  - *replay*
- solutions:
  - use TLS sequence numbers (data, TLS-seq-# incorporated into MAC)
  - use nonce

t-tls: connection close

- truncation attack:
  - attacker forges TCP connection close segment
  - one or both sides thinks there is less data than there actually is
- solution: record types, with one type for closure
  - type 0 for data; type 1 for close
- MAC now computed using data, type, sequence #

SDN

⭐ NOT in Final

Virtualization

⭐ NOT in Final