

## 01. Introduction

- **Network Edge** - Hosts (Clients and servers)
- **Access Networks** - Wired and wireless communication links
- **Network Core** - Network of interconnected routers

### Network Core

- **Store and Forward** - Entire packet must arrive at router before being transmitted to next link

### Key Functions of Network Core

- **Routing** - Determines source-destination routes taken by packets (How we get the hashtable)
- **Forwarding** - Move packets from router's input to correct router output

### Packet-Switching

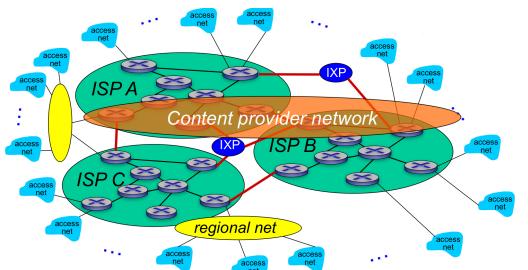
- Host breaks messages into packets of  $L$  bits
- Transmits packets into access network at transmission rate  $R$  (aka Link bandwidth, capacity)

$$\text{Packet Transmission Delay} = \frac{\text{Packet Size (bits)}}{\text{Transmission Rate (bits/sec)}}$$

### Circuit Switching

- Resources reserved for call between source and destination
- End to end resources allocated and reserved
- Set up and teardown required
- Circuits like (guaranteed) performance cos circuit reserved per usage
- Pros: Better performance
- Cons: More resources (Unable to share with other processes)

### Internet Structure

- 
- The diagram illustrates the Internet structure. At the bottom, multiple 'access net' icons connect to 'ISP A', 'ISP B', and 'ISP C'. 'ISP A' and 'ISP B' are part of a 'Content provider network', which then connects to 'ISP C'. 'ISP C' is part of a 'regional net'. 'ISP C' connects to a 'Content provider network' which then connects to 'ISP A' and 'ISP B'. 'ISP A' and 'ISP B' both connect to an 'IXP' (Internet eXchange Point). 'ISP A' and 'ISP B' also have direct connections to 'Content provider network' nodes. 'Content provider network' nodes are shown with blue dots and 'access net' icons. 'ISP' nodes are shown with green dots and 'access net' icons. 'IXP' nodes are shown with red dots and 'access net' icons.
- End systems connect to Internet via **Access Internet Service Providers (ISPs)** (autonomous systems → regional ISP → Access ISP → hosts)
  - ISPs connect to larger global ISPs (usually competitors)
  - Large ISPs connect via **peering links** or **internet exchange points (IXP)**
  - **IXP** - Physical place with routers from different ISPs
  - **Regional Networks** - Smaller ISPs
  - **Content Provider Networks** - Provide content close to end users

## Loss, Delay, and Throughput

### Packet Loss

- If Arrival Rate > Transmission Rate, packets will queue and can be dropped if buffer fills up
- Solutions: Lost packets can be retransmitted

### Packet Delay

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

- **Nodal Processing** - ( $d_{\text{proc}}$ ) Check for bit errors and determine output link

- **Queueing Delay** - ( $d_{\text{queue}}$ ) Time at queue waiting for transmission, router congestion

- **Transmission Delay** - ( $d_{\text{trans}}$ ) Time to load packet onto link

$$\bullet d_{\text{trans}} = \frac{L}{R} \text{ where } L \text{ is packet length and } R \text{ is link bandwidth}$$

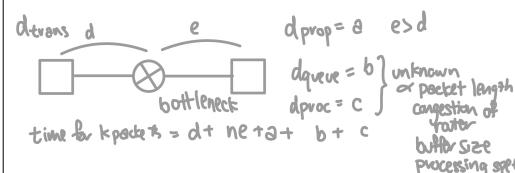
- **Propagation Delay** - ( $d_{\text{prop}}$ ) Time for 1 bit to reach end of link

$$\bullet d_{\text{prop}} = \frac{d}{s} \text{ where } d \text{ is length of link and } s \text{ is propagation speed}$$

- Does not depend on packet size, number of packets or number of intermediate nodes

### Throughput

- Rate at which bits transferred between hosts
- Different from transmission rate (Theoretical upper bound)
- Measures end-to-end, even through intermediaries
- Irregardless of the size of packets
- Dealing with bottlenecks over  $n$  packets:



- Average: Rate over long period of time
- Instantaneous: Rate at given point in time

### Protocol Layers and Service Models

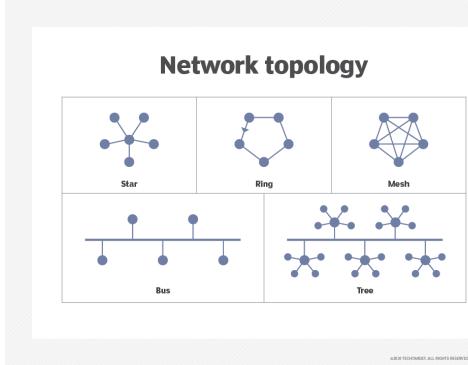
- **Protocol** - Defines format, order of messages sent and received, and actions taken on message transmission
- **Layering** - Each layer implements a service by doing something within layer and relying on services provided by layer below it
  - Explicit structure allows us to make sense of complex components
  - Easy maintenance (Like OOP, change in 1 layer should not affect others)

## Internet Protocol Stack

1. **Application** - eg. FTP, SMTP
2. **Transport** - TCP, UDP
3. **Network** - IP, Routing protocols
4. **Link** - Ethernet, WiFi, PPP
5. **Physical** - Wires, bits

- **Encapsulation** - Take information from a higher layer and adds a header to it, treating the higher layer information as data

### Topology



- **Star** - Lowest cost and simplest

- **Mesh** - High redundancy, increased cost but highest transmission speeds

## 02. Application Layer

- Programs that run on end systems, and not on network-core devices

### Client-server Architecture

- **Server**: Always-on host, Permanent IP address, waits for incoming req and provides service to client
- **Clients**: Communicates with server, Intermittently connected, Dynamic IP addresses, Do not communicate with each other directly, initiate connection with server to request services

### P2P Architecture

- Peers request service from other peers and provide service in return
- No always-on server, Intermittently connected, Dynamic IP addresses
- **Self Scalability** - New peers offer new services and demands

### Process

- **Inter-process Communication** - How 2 processes in 1 host communicate
- **Socket** - Process sends/receives messages to/from its socket (like a door)
  - Outside of socket, transport layer delivers message

### Addressing Processes

- Motivation: IP address is not enough to address process, since many processes can be running on same host
  - **Identifier** - IP address and port number
  - **IP** - 32-bit address for identifying host
  - **Port Number** - 16-bit to identify specific process on host

## Services

- **Data Integrity** - Reliable data transfer
- **Timing** - Low delay/latency
- **Throughput** - Minimum amount of throughput for effectiveness
- **Security** - Encryption, data integrity

### Transport Protocol Services

1. **TCP** - Transmission Control Protocol
    - Reliable transport
    - Flow control: Sender does not overwhelm receiver
    - Congestion control
    - Connection-oriented: Setup required between client and server
  2. **UDP** - User Datagram Protocol
    - Unreliable data transfer
    - Fast
- Does not guarantee minimum throughput and timing

### App-layer Protocol

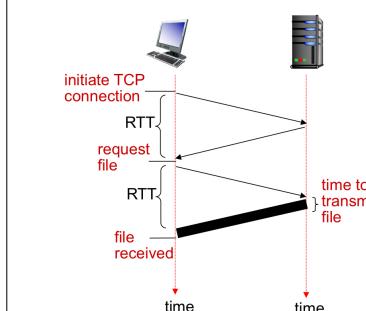
- Types of messages exchanged (e.g. Request or response)
- Message syntax: How fields are delineated
- Messages semantics: Meaning of information in fields

### HTTP - 80

- **Hypertext Transfer Protocol** - Web's application layer protocol
- Motivation: Web page consists of objects (HTML, images). Need method to request/send web objects.
- Follows client/server model
- Uses TCP
- **Stateless** - Server maintains no information about past requests

### Non-persistent HTTP 1.0

- At most 1 object sent over TCP connection
- Downloading multiple objects requires multiple TCP connections
- New TCP connection for each web resource/object

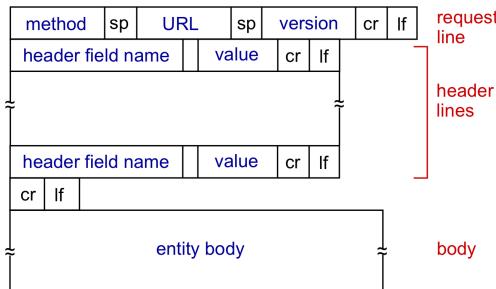


- Server closes TCP connection after sending file
- **Return Trip Time** - (RTT) Time for small packet to travel from client to server and back
- RTT is  $2 * d_{\text{prop}}$
- Response Time:  $2 \text{ RTT} + \text{File transmission time}$

## Persistent HTTP

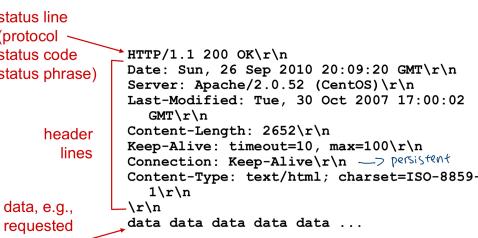
- Multiple objects can be sent over single TCP connection
- Server leaves TCP connection open after sending response
- As little as one RTT for all referenced objects

## HTTP Request Message



- To upload form input: **POST method** - Input uploaded via entity body and **URL method** - Input uploaded in URL field of GET method
- **HTTP/1.0** - GET, POST, HEAD (Ask server to leave request object out of response)
- **HTTP/1.1** - GET, POST, HEAD, PUT, DELETE

## HTTP Response Message



## Error codes

200 OK

301 Moved permanently

304 Not modified (during conditional GET)

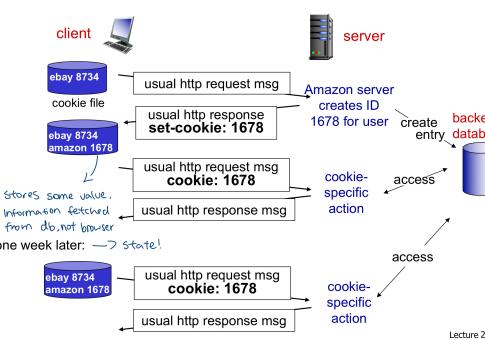
400 Bad request

404 Resource not found

505 HTTP Version Not Supported

## Cookies

- Maintains state on client side
- Components: Cookie header for HTTP response, Cookie header for HTTP request, Cookie file on user's host (Key-value pair), Database on server



## Web Cache (Proxy Server)

- Goal: Fulfill request without involving origin server via caching
- Browser sends all HTTP requests to cache
- Pros: Faster, Reduces traffic to origin server
- Cons: What if origin server updates?
- **Conditional GET** - Origin server doesn't send object if cache has updated version
- Cache: Specifies date of cached copy in HTTP request to origin (If-modified-since)
- Origin Server: Response contains no object if cached object is updated (Empty content body)

## Domain Name System - 53

- Maps between hostname (e.g. yahoo.com) and IP address
- Implemented using distributed and hierarchical databases
- Application-layer protocol (Why?)

Build complexity in edges to simplify inner core networks

- Uses UDP (Much faster and smaller packet overhead)
- Services provided

**Host/Email aliasing** Additional name for complicated hostnames

**Load balancing** Providing multiple IP addresses for the same URL

- **Local DNS Name Server** - Local cache of name-to-address mapping. Forwards query into hierarchy.

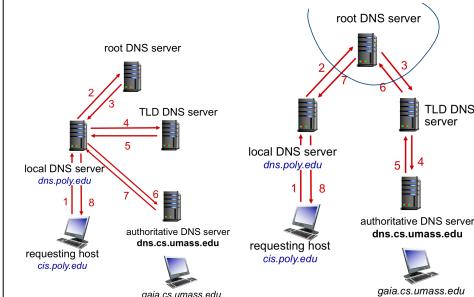
- **Time to Live** - (TTL) Cached mappings disappear after some time

- **Root Name Server** - Contacted by local name server that cannot resolve name. Provides IP address of TLD servers.

- **Top-level Domain Server** - (TLD) Provides IP address of authoritative server

- **Authoritative DNS Server** - Organization's own DNS server. Provides mappings for organization's named hosts.

- Iterated query: "Not sure, ask this server"



- Recursive query: "Okay, let me find for you"

- Heavy load on upper levels of hierarchy

Local DNS servers does both

## DNS Caching

**Cache entry** - Cache record of recent name mapping

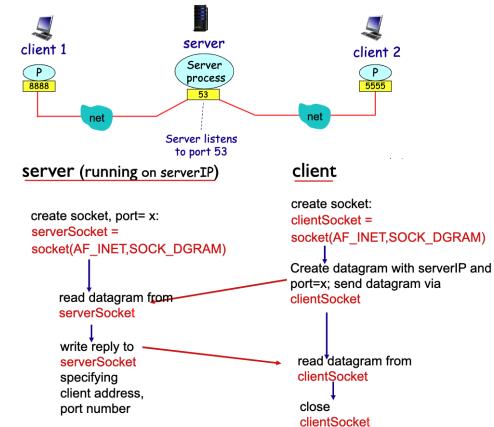
- Timeout after some time based on TTL
- Best effort name-to-address translation (can be out-of-date)
- **Resource record** - Mapping btw hostname and IP
- Format: {NAME, TYPE, VALUE, TTL}

## DNS Cache poisoning

- DNS spoofing attack - Rogue DNS records introduced in resolver's cache, leading to incorrect IP address reply

### 03. Socket Programming with UDP and TCP

#### UDP Socket



- No connection beforehand. Just send it.

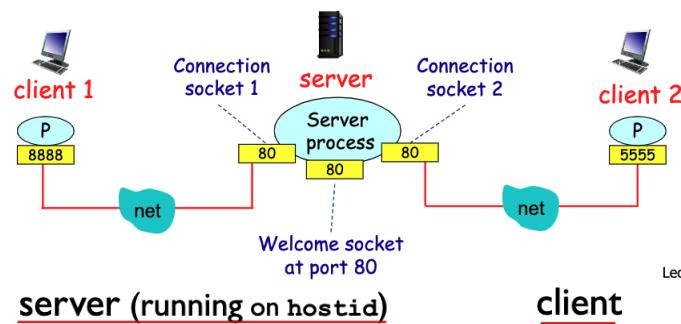
- Server has 1 socket to serve all clients

- Sender attaches destination IP address and port number (**Stateless**)

- Unreliable datagram: Data may be lost or out-of-order

- **Datagram** - Group of bytes

#### TCP Socket



- Client establishes connection to server via welcome socket
- Server makes new socket for each client
- Server identifies client via connection (**Stateful**)
- Reliable stream pipe: Data always in order

#### Remarks

- SOCK\_STREAM - TCP, SOCK\_DGRAM - UDP
- Can only specify destination port numbers, source port numbers randomly assigned by OS

### 04. UDP and Reliable Protocol

#### Transport Layer Services

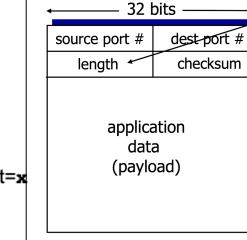
- Provide logical communication between processes in different hosts
- VS network: Process-to-process VS host-to-host
- Sender: Breaks app messages into segments
- Receiver: Reassembles segments

#### UDP

- On top of network layer, UDP adds:

1. Connectionless multiplexing/de-multiplexing
  - UDP segments contain both source and destination ports
  - **Multiplexing** - Sent to target processes
  - **Demultiplexing** - Delivering the received segments at the receiver side to the correct app layer processes
2. Checksum

#### UDP Segment Header



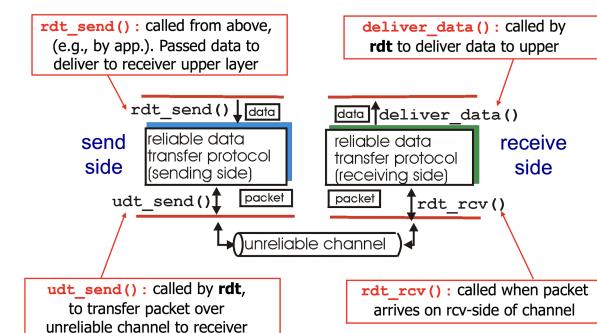
Note that length includes 8 byte header + body content

#### Checksum

- Goal: Detect errors in received segment
  1. Treat UDP segment as sequence of 16-bit integers
  2. Add every 16-bit integer (Carry added back to result)
  3. Invert to get UDP checksum (1's complement)
  4. When receiving, sum segment again. All 1s if correct.

#### Reliable Data Transfer (rdt)

- Characteristics of unreliable channel will determine services provided by rdt
- Cannot fix unordered packets

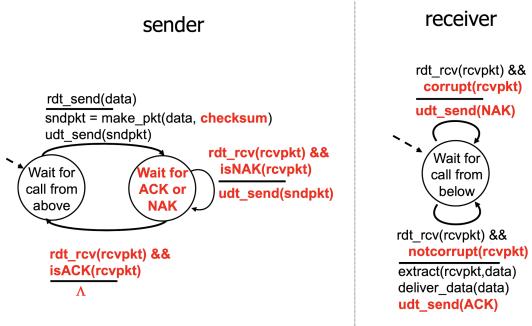
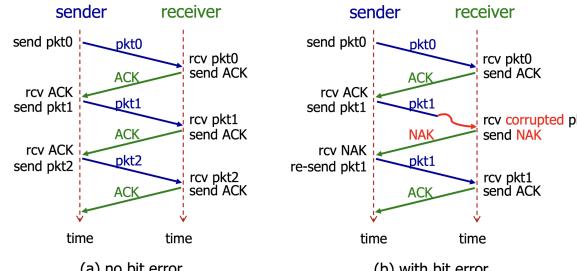


## rdt 2.0

- New problem: **Bit error** - May flip bits in packet

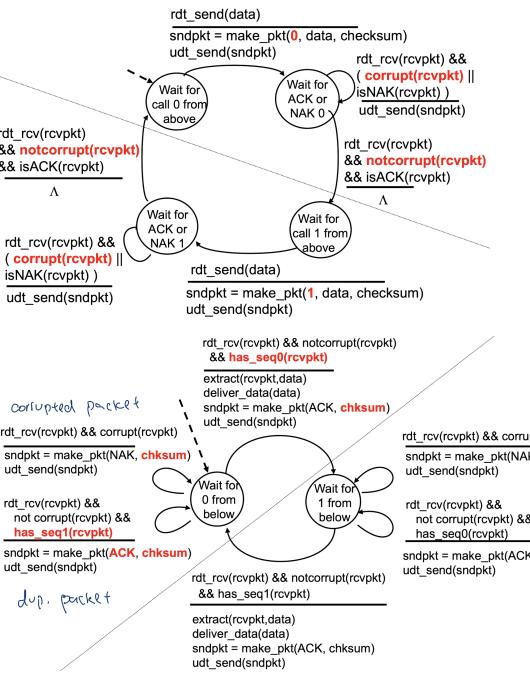
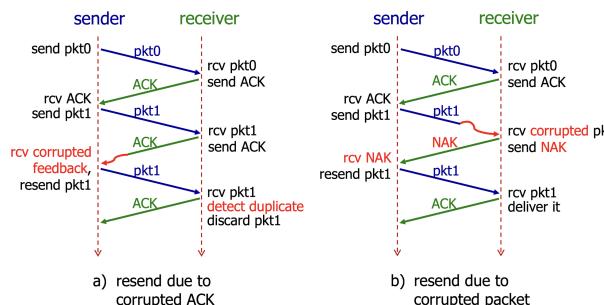
### Solution:

- Perform checksum to detect bit errors
- **Stop and wait protocol** - Sender sends one packet at a time and wait for response
- **ACK** - Receiver tells sender that packet received is ok
- **NAK** - Receiver tells sender that packet received has errors (retransmission)



## rdt 2.1

- New problem: ACK/NAK may be corrupted
- Solution: Sender retransmits packet after receiving corrupted ACK/NAK
- New problem: Duplicate packets during retransmission
- Solution: Sender adds **sequence number** to each packet and receiver discards duplicate packet (Only 0 and 1 needed)

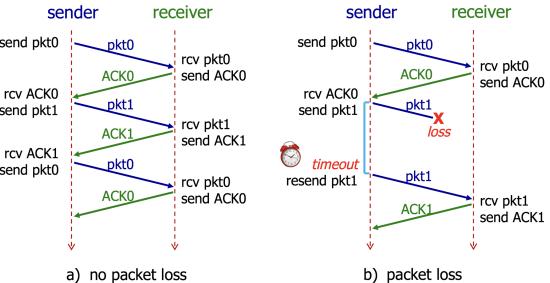
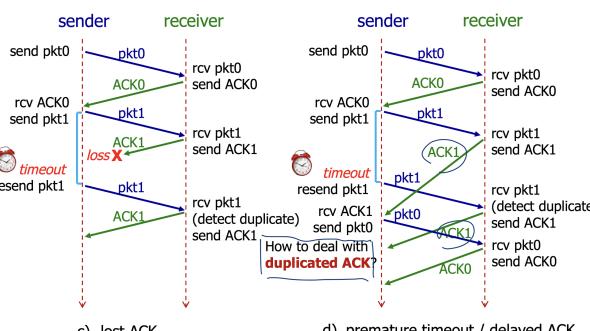


## rdt 2.2

- No NAK, only ACKs
- Receiver sends ACK for last packet received. **ACK must include seq number of packet**.
- Receiver send duplicate ACK to retransmit current packet

## rdt 3.0

- New problem: Lost packets
- Solution: Sender waits for some time for ACK and retransmits packet
- What if duplicate packet? Sequence number handles this.
- What if duplicate ACK? Do nothing.



## Performance of rdt 3.0

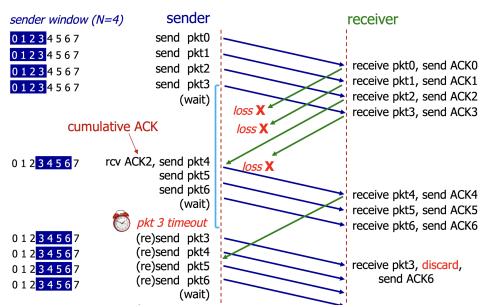
- **Stop-and-wait protocol** - Sender sends 1 packet at a time, then waits for receiver response
- Performance is bad. Stop-and-wait protocol limits use of resources
- **Utilization** - Fraction of time sender is busy sending

- Given: 1 Gbps link, 15 ms prop. delay, 8000 bit packet
- $D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microsecs}$
- $U_{sender} = \frac{D_{trans}}{RTT + D_{trans}} = \frac{0.008}{30.008} = 0.027\%$
- Pipelined Utilisation =  $U * \text{window\_size}$

## Pipelined Protocols

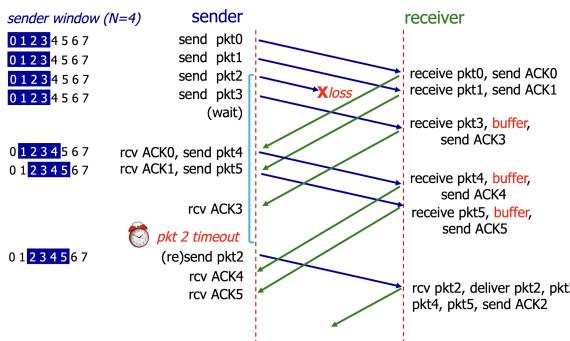
- **Pipelining** - Sender allows multiple not-yet-ACKed packets
- Need more sequence numbers
- Buffering at sender and receiver
- Pipeline vs Parallel (HTTP): Parallel connections need to retrieve HTML webpage first before retrieving data elements
- Pipeline can retrieve HTML and data elements all at once

## Go-Back-N



- Intuition: Sliding window
  - Sender:
    - Timer for oldest in-flight packet
    - If timeout( $n$ ), retransmit packet  $n$  and other pkts in window
  - Receiver:
    - **Cumulative ACK** - ACK for correct pkt with highest in-order sequence
    - Only ACK packets that arrive **in-order**
    - Discard **out-of-order** packets
  - Not efficient, since future packets discarded if out-of-order

## Selective Repeat



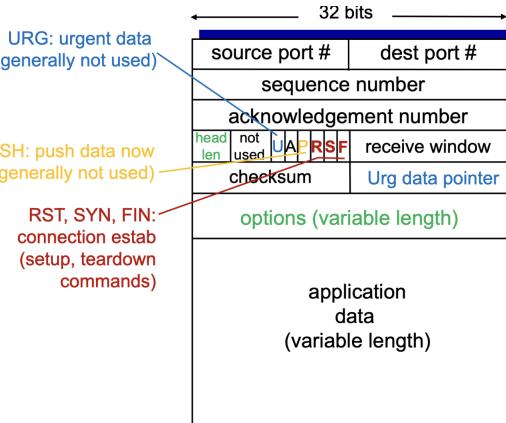
- Receiver individually ACKs correct pkts (Not accumulative) and sender maintains timer for each unACKed pkt
  - Sender:
    - If timeout( $n$ ), retransmit packet  $n$  only
    - If ACK( $n$ ) and  $n$  is smallest unACKed pkt, slide window
  - Receiver:
    - Once receive pkt  $n$  in window, send ACK( $n$ ). If out-of-order, buffer. If in-order, deliver and slide window
    - Once receive pkt  $n$  outside of window, still send ACK( $n$ )
    - ACK m == packet m received but has no implication on receipt of other packets

## 05. TCP

## TCP Features

- **Point-to-point** - 1 sender and 1 receiver
  - **Connection-oriented** - Handshaking before exchange data
  - **Full duplex** - Bi-directional data flow in connection
  - Reliable, in order and pipelined
  - Buffers on both hosts
  - Limited by **Maximum Segment Size**  $\approx$  1460B
    - Refers only to content length (no header)
    - Limited by **MTU** - Maximum transmission unit, size of largest IP transaction

## Structure

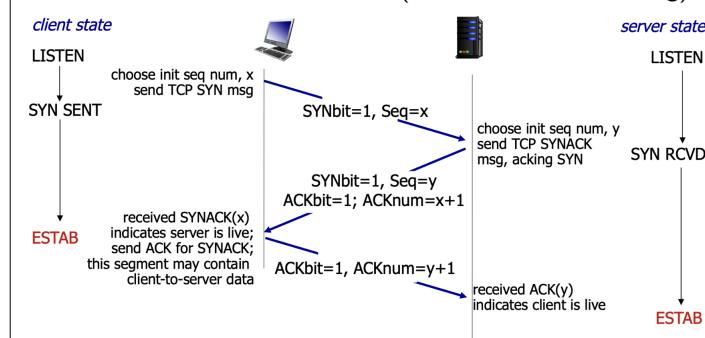


Header is 20-60 bytes long

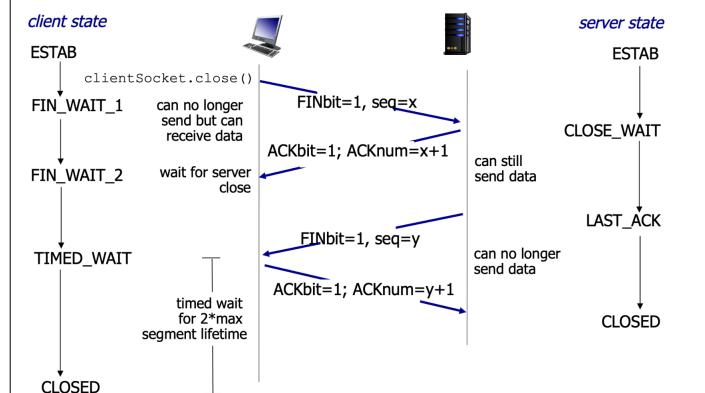
1. Sequence #
    - Let receiver know if packet is retransmission or new data
    - Value only measures the body length in bytes
    - Initial number randomly generated btw  $0 - (2^{32} - 1)$  (Why?)
      - Security
      - Prevent duplicate sequence numbers when link is used extensively
    - ACK # = next byte expected by receiver (initial seq # + data length in bytes)
    - SYN # = byte number of first byte(expected ACK (of sender) - data length)
  2. ACK number
  3. **rwnd** - Receive window
    - Number of bytes the receiver is willing to accept
    - **Flow control** - Prevent overflow of receiver's buffer
  4. Flags - Connection establish or priority

### Establish TCP connection

Creates  $n+1$  sockets for  $n$  clients (additional 1 for listening)



## Close TCP connection



Why TIMED\_WAIT ( $2 * \text{MSL}$  - Maximum segment lifespan)

- Guarantee the port has been closed completely
  - Prevent delayed data segments from being received by other TCP connections

RDT

## Features

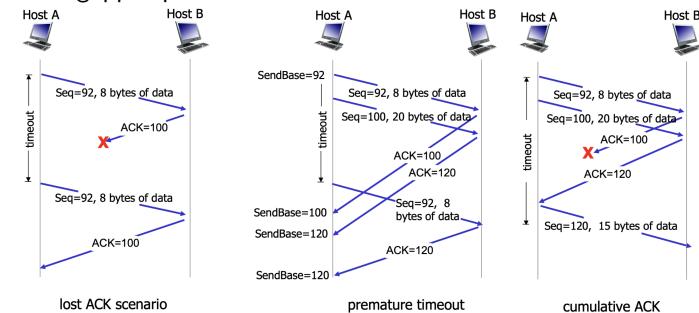
- Pipelined segments
  - Cumulative ACKs (like GBN)
  - Single retransmission timer
  - Retransmit only missing packets (SR)

## Retransmission Scenarios

Send ACKs every other packet OR after timeout of packet OR retransmission event at receiver

	TCP receiver action
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately send single cumulative ACK, ACKing both in-order segments
arrival of out-of-order segment higher-than-expect seq #. Gap detected	immediately send <b>duplicate ACK</b> , indicating seq. # of next expected byte
arrival of segment that partially or completely fills gap	immediate send ACK, provided that segment starts at lower end of gap

receive gapped packet



## RTT

- Problem: Estimate timeout value for retransmission
- Solution: Average recent **SampleRTT** - Measured time from segment transmission until ACK
- $\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$
- Exponential weighted moving average
- **DevRTT** - Margin due to deviation from Estimate RTT
- $(1 - \beta) * \text{DevRTT} + \beta * (\text{Sample RTT} - \text{Estimated RTT})$
- **Timeout Interval** -  $\text{EstimatedRTT} + 4 * \text{DevRTT}$

$$\alpha \approx \frac{1}{8}, \beta \approx \frac{1}{4}$$

## Fast retransmission

- Retransmit unacked segment **IMMEDIATELY** with smallest seq # after receiving 4 ACKs for same data

## Midterm tips

1. Bits are b and bytes are B (8 bits = byte)

## 06: IP Addressing

### Function

1. **Forwarding** - Move packets from router input to output

### 2. Routing

- Determine route taken by packets from source to destination
- Plan trip from source to destination

### Planes

#### 1. Data plane

- Local per router function
- Determines how datagram forwarded to output port

#### 2. Control plane

- Network wide logic
- Routing algorithms

### Subnet

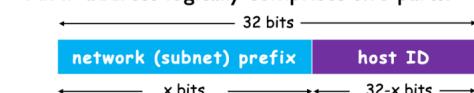
- Network formed by directly interconnected host
- Internal host can communicate without router
- Connect to external networks with router
- Same network prefix
- Valid subnet masks
- 254, 252, 248, 240, 224, 192, 128

### Special subnet

- Private IP addresses - 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16
- Loopback address 127.0.0.0/8
- Broadcast 255.255.255.255/32

### CIDR

- An IP address logically comprises two parts:

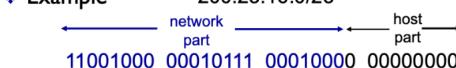


- **CIDR: Classless InterDomain Routing**

- arbitrary length for the subnet portion (network prefix)
- address format: a.b.c.d/x, where x is # bits in subnet portion of address

- **Example**

200.23.16.0/23



## Subnet mask

- **Subnet mask** is made by setting all network prefix bits to "1"s and host ID bits to "0"s.

- example: for IP address 200.23.16.42/23:

IP address in binary  $11001000\ 00010111\ 00010000\ 00101010$  network prefix host ID

Subnet mask  $11111111\ 11111111\ 11111110\ 00000000$

Subnet mask in decimal 255.255.254.0

- used to determine which network an IP address belongs to (use bitwise AND operation)

### Hierarchical addressing

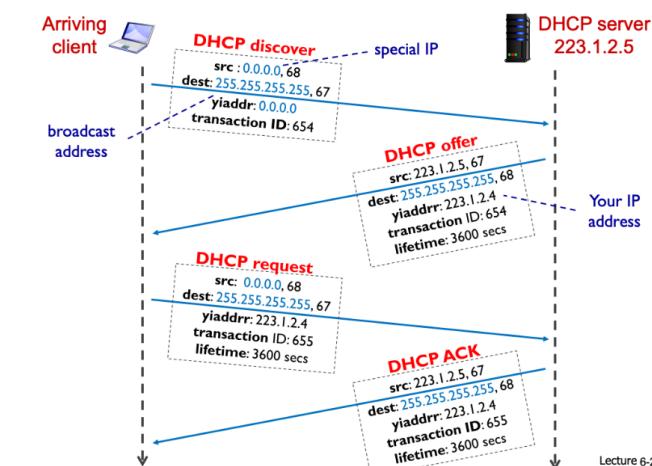
**Route aggregation** - Advertise routing information by grouping similar IP addresses  
Longest Prefix matching

Routes to subnet with the greatest subnet portion of CIDR

### DHCP

**Dynamic Host Configuration Protocol** - Lease IP address from network server when joining network

- Runs over UDP, Client #: 68
- Router can act as DHCP
- Server #: 67, sends "DHCP ack" and "DHCP offer"
- Client #: 68, sends "DHCP request" and "DHCP discover"



What happens when multiple servers in network?

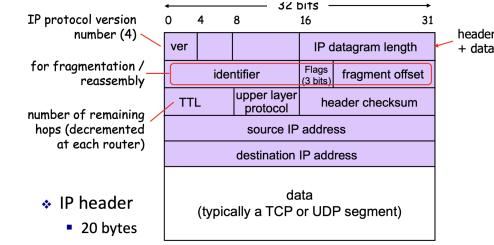
- Transaction ID specifies the DHCP connection

## 07. IP (cont)

### IP fragmentation

**MTU** - Maximum Transfer Unit, maximum amount of data a link-level frame can carry

**Fragmentation** - Large datagrams that are broken up by routers



- Frag flag = 1 if there are more fragments from same segment, 0 = last fragment
- Offset - in units of 8-bytes specifying the offset of fragment wrt beginning of unfragmented IP datagram

### Problem:

Not enough IP addresses to specify all devices

### Network Address Translation

Using private IP addresses to map devices within an organisation and enable communication with the internet

- WAN (Wide area network) vs LAN (Local area network) - private vs public IP address

### Implementation

- Routers replace source IP address, port num of every outgoing datagram to NAT IP address and new port num
- Remember (in NAT translation table) the mapping of the source vs NAT info
- Replace NAT ip address and port num in destination fields of incoming datagrams with corresponding source IP/port stored in NAT translation table

### Motivation

- No need to rent a range of public IP addresses from ISP: just 1 for NAT
- Hosts using private IP address can have variable IP addresses without notifying other hosts
- Can change ISP without changing addresses of host in LAN
- Hosts in Lan are not explicitly addressable and visible by outside (Security)

### Challenges

- Difficult to have p2p applications across WANs

### Routing

Hierarchical routing on internet due to size and decentralised administration

Goal: Find the least cost to connect 2 hosts through routers

### Intra Autonomous system(AS) routing

- Finds good path between routers within AS
- Single admin so no policy decision needed
- Performance impt in routing
- Protocols - RIP and OSPF

### Inter AS routing

- Handles interfaces between AS
- Admin wants control over traffic routing
- Policy performance
- Protocol - BGP

Cost associated along each link based on various factors

- Inversely related to bandwidth
- Related to congestion of router
- Distance, \$

### Distance vector

Iterative Bellman-Ford computation

1. Starting router queries all the neighbours around it
  2. Neighbours could have cached the distance vector to other routers
    - (a) Router can send the table of distance vectors to the new router (routing tables)
  3. No/incomplete distance vector record: Wait for (change in local link cost or message from neighbour)
  4. Recomputes DV estimates using DV received from neighbour
  5. Notify neighbours if DV to any destination changes
- Iterative and asynchronous: local iteration caused by
    - Local link cost changes
    - DV update messages from neighbours
  - Distributed and self stopping
    - Each node notifies neighbours **only** when DV changes (if necessary)

### Link state algorithms

- Periodically broadcasting link costs to each other
  - All routers have complete knowledge of topology and link cost
  - Use Dijkstra algorithm to compute least cost path locally
- Routing Information Protocol(RIP)**
- Implements DV algorithm
  - Uses hop count as cost metric
  - **Self-repair** - If no updates from router for 3 minutes will assume router fails
  - Exchanges routing table every 30s (UDP - 520)

### Internet Control Message Protocol - ICMP

Used by hosts and routers to communicate network information

When TTL is 0, packet is discarded and ICMP error message is sent to datagram's source address

### Error reporting

Unreachable host/network/port/protocol

### Echo Request

**Headers** Starts after IP header carried in IP datagrams

Type	Code	Description
8	0	echo request (ping)
0	0	echo reply (ping)
3	1	dest host unreachable
3	3	dest port unreachable
11	0	TTL expired
12	0	bad IP header

### Ping

- Used to echo and test connection between different remote hosts
- Format: *XX* bytes from *client ip*: *icmp\_seq=X* *ttl=X* *time=X ms*
- Time == time taken for ping command to execute completely
- ttl = number of hops between client and remote host
- icmp\_seq = index of the datagrams sent to remote host (ascending order, default 0..6)

### Traceroute

- Sends series of small packets with different TTL to display route to get to host
- Mechanism
  - TTL decrements every hop from initial TTL, X
  - Once TTL has reached 0, it will immediately reply with ICMP error message, addressed from the current device to the original source
  - Extract source IP address from the reply to get the device X hops away