Common Conversions

Time: 1 second = 1,000 milliseconds
Data Transfer: 1 gigabit (Gb) = 1,000 megabits (Mb) → 1 megabit (Mb) = 1,000 kilobits (Kb) → 1 kilobit (Kb) = 1,000 bits (b)
Storage: 1 gigabyte(GB) = 1,024 megabytes (MB) → 1 megabyte (MB) = 1,024 kilobytes (KB) → 1 kilobyte = 1,024 bytes

Section 0: Basis

What is the internet?

- **Hosts or End Systems:** The millions of connected devices that run **network apps**
- **Communication Links:** fiber / copper/ radio/ satellite etc.
- **Bandwidth:** the transmission rate / bits per second
- **Packet Switches: Routers / Switches** that forward packets / chunks of data between devices
- **Internet**: Interconnected ISP's (Internet Service Providers = org that provides access to int services)
- **Protocols**: Control sending and receiving (transmission) of msgs. (TCP, IP, HTTP, Skype, 802.11)
- **Internet Standards**: RFC (Request for comments), IETF (Internet Engineering Task Force)

A service view of the internet

- The internet is an infrastructure that provides services to applications
  - Web / VoiP, email, games, ecommerce, social nets etc.
- Provides programming interfaces to apps
  - Hooks that allow sending / receiving app programs to connect to the internet
  - Provides service options, analogous to a postal service.

What is a protocol?

- They are basically a set of rules.
- **Protocols define the format, order of msgs sent and received among network entities and actions taken on msg transmission and receipt.**
- All communication activity on the internet is governed by protocols
- **TCP (Transmission Control Protocol)** is one of the most popular protocols

The network structure

- **Network Edge**: The end devices which connect to the network
  - Hosts = clients and servers. Servers are often in data centres.
- **Access Network / Physical Media**: What allows you to to connect to the internet.
  - Wired, wireless communication links etc.
- **Network Core**: Interconnected routers / network of networks

Access networks

- How do we connect the End Systems to the Edge Router?
  - Residential access nets, institutional access networks (school, company), mobile access networks
- Things to keep in mind about access networks:
  - Bandwidth (bits/sec) of access network? Shared or dedicated bandwidth?
  - DSL = dedicated bandwidth. Cable = shared bandwidth.
  - Congestion if too many people are using shared network.
- Access network example #1: Home network
  - Wireless Access Point (54 Mbps), Router, Firewall, Wireless Devices, cable or DSL modem, wired ethernet etc.
- Access network example #2: Enterprise network
  - Ethernet switch, institutional mail / web servers, institutional router, institutional link to ISP.
  - 10mbps, 100mbps, 1gbps, 10gbps transmission rates.
- Wireless Access Networks
  - Shared wireless access network connects end systems to the router
  - **Wireless LANs** = within building, 802.11 (WiFi), 11 / 54 / 450 mbps transmission rate
  - **Wide-area wireless access** = provided by telco (cellular operation, 10's kms), 1 to 10mbps, 3G / 4G: LTE

Physical Media

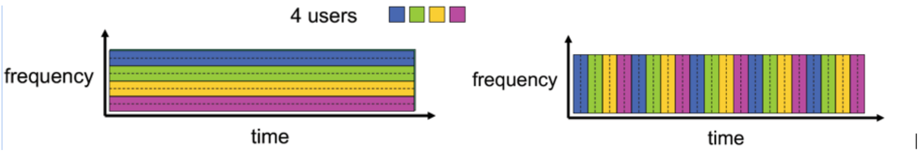
- **Bits**: Propagates between transmitter / receiver
- **Physical Links:** what lies between transmitter / receiver
- **Guided Media:** Copper / Fiber / **Unguided Media:** signals propagate freely i.e. radio
- **Twisted Pair:** two insulated copper wires.

- **Fiber Optic Cable**: glass fiber carrying light pulses, each pulse = 1 bit
  - Low error rate / high-speed operation 10's-100's gbps
- **Coaxial Cable:** two copper conductors / broadband
- **Radio**: Signal carried in electromagnetic spectrum (no physical wire)
  - Propagation environment effects: Reflection / Obstruction by objects / Interference
  - Radio link types: **Microwave (45mbps), LAN (WiFi) (1/45/450mbps), Wide-Area (~10mbps), Satellite (kbps to 45mbps)**

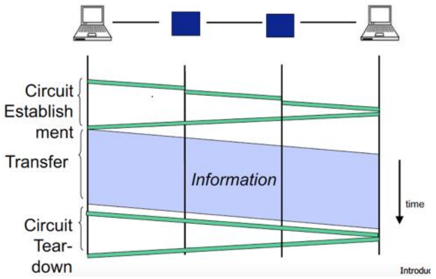
The Network Core is the mesh of interconnected routers / switches. Two forms of switched networks:

1. Circuit Switching: Used in legacy systems / traditional telephone networks

- End-to-end resources allocated / reserved for "call" between source and destination. Dedicated resources: no sharing
- If a connection is established with customer, even if customer is idle, no one else can use it (**no sharing**)
- Two technologies for Circuit Switching:
  1. **FDM (Frequency Division Multiplexing)**: Users share divided frequency / use frequency simultaneously
  2. **TDM (Time Division Multiplexing)**: Frequency is given to one user who gets to use the whole frequency / round robin to share use



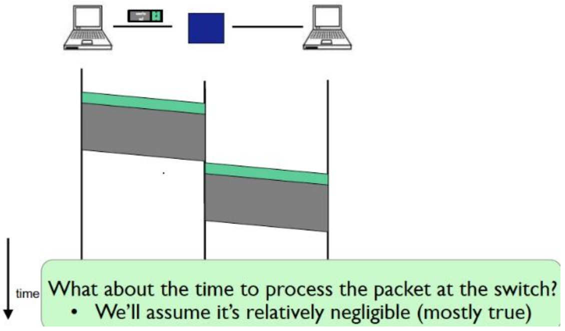
- Timing in Circuit Switching: **Circuit Establishment → Transfer of Information → Circuit Tear-Down**



- Pros and cons of circuit switching?
  - PRO: Uninterrupted connection, reserved and dedicated for you.
  - PRO: Potentially faster depending on how much reserved / generally super performance
  - CON: Reserved channel even for idle connections, can't be used by anyone. I.e. Waste of resources.

2. Packet Switching: Used in the internet

- Data is chopped into small chunks of formatted **bits** (These chunks are **Packets**)
- Packets consist of a **Header** and **Payload**
- **Payload**: The data you want to send which is split up into different packets / chunks
- **Header**: Header holds the instructions to the network for how to handle the packet
  - I.e. Who does it go to and how is it routed, what is the quality that the network should provide for this payload
  1. **Internet Address**
  2. **Age (TTL: Time To Live)**: To avoid looping, every time it goes through a router it decrements ctr. Packet will be dropped when counter goes to zero.
  3. **Checksum to protect header**: For error correction. If bits are flipped, you know how to fix it or you know something is wrong with the data.
- **Switches** help "forward" packets based on their headers.
- Timing in Packet Switching: **Sending header + packet → Arrives @ Router → Finds next dest → Sends H+P to next**



What about the time to process the packet at the switch?
- We'll assume it's relatively negligible (mostly true)

When a packet comes, we need to:

- Find out next address
- Modify the TTL field (decrement for each router)
- Other error checking etc.

The processing time is pretty much negligible.
Diagram to the left is a **Time Series Diagram.**
Can the switch start transmitting as soon as it has processed the header? **Yes, it is called Cut-Through switching**. **Fast**

- We will always assume that we are not using Cut-Through switching, i.e. a switch processes / forwards a packet after it has received it entirely. This is called **Store and Forward** switching.
- Packet switching leverages **Statistical Multiplexing**: No link resources are reserved in advance, unlike Circuit Switching.
  - Dynamically allocates bandwidth to each channel on an as-need basis.
  - A communication channel is divided into an arbitrary number of variable bitrate digital channels/data streams.

    - Each stream is divided into packets that normally are delivered asynchronously in **first-come first-serve basis**. Alternatively, the packets may be delivered according to a **scheduling discipline** or **fair queuing**.
    - **It usually implies "on-demand" service rather than one that pre-allocates resources for each data stream.**

In data networks, communication flow is typically **Bursty**

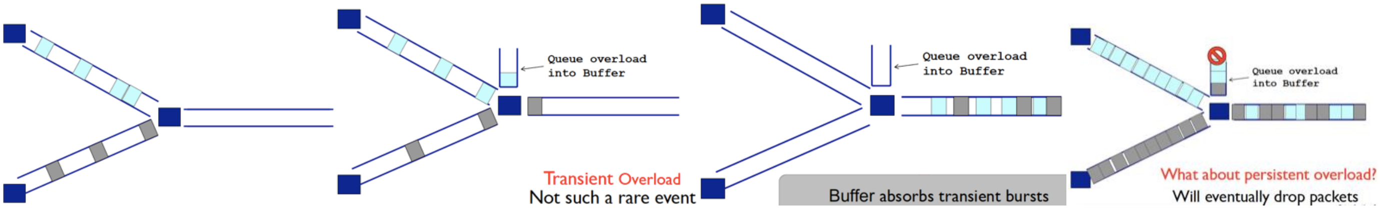
- There is a period of time where you are transmitting a lot of data and then a period where you are sending nothing.
- E.g. Voice communications: Periods of speaking vs. pause in speaking. It is not a constant stream of data.
- Video too (if compression is present), where periods of no movement on the video = less bits being sent through. However, uncompressed video will be constant.
- Without Statistical Multiplexing, everytime you're exceeding bandwidth, you'll have overload.
- All these multimedia and application characteristics impact network design.

What happens when communication flow shares the total capacity.

- If flow is shared, over a total period of time, you can allow other active sources to share the bandwidth where some nodes are not that active = **NO OVERLOADING**
- **Statistical multiplexing relies on the assumption that not all flows burst at the same time**.
- Overflow can still occur, but not frequently if there are bursts which all occur at the same time.

Statistical Multiplexing: A pipeline view.

- The packet size varies for different technologies.
- A router switch is simply another computer that forwards packets.


Queue overload into Buffer
Transient Overload
Not such a rare event
Queue overload into Buffer
Buffer absorbs transient bursts
Queue overload into Buffer
What about persistent overload?
Will eventually drop packets

Packets get forwarded → Back-to-back packets (Transient Overload), placed in a queue → Persistent Overload = packet loss

- A buffer is only finite. It will fill eventually and you'll have to drop packets.
- Why can't we have a very large buffer? You could, but the buffer would just keep getting filled up and there would nevertheless be a huge amount of delay. E.g. Talking to someone on Skype and having huge lag / silence in between.

Pros and Cons of packet switching

- PRO: Requires less infrastructure. With circuit switching, you have dedicated linking so will require more infrastructure.
- CON: Packet loss.

Packet Switching VS. Circuit Switching

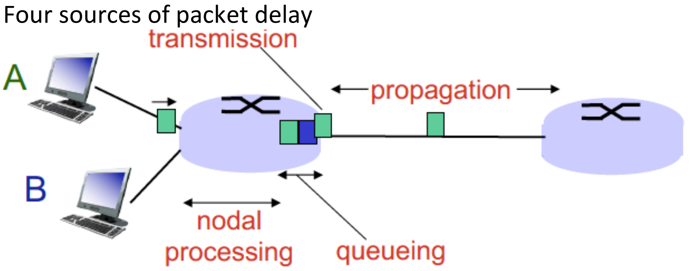
- Packet switching allows more users to use the network
    - E.g. 1mbps link → Each user has 100kb/s when active, are active 10% of the time.
- Circuit switching: 10 users for example.
- Packet switching: 35 users, probability > 10 active at the same time is less than 0.0004%
- Is packet switching a "Slam-dunk winner"?
    - **Great for Bursty Data because of statistical multiplexing**: Resources sharing + simpler / no call setup
    - **Excessive congestion is possible:** Packet delay and loss
        - Protocols are needed for reliable data transfer and congestion control.
    - How to provide circuit-like behaviour?
        - Bandwidth guarantees need for audio/video apps.
        - Still an unsolved problem for internet-wide domain, but possible for within enterprise-level domains.

Internet structure: The internet is a Network of networks

- End systems connect to internet via. **Access ISP's**: Residential, company and university ISP's(ISP:互联网服务提供商)
- Access ISP's in turn must be interconnected: Any two hosts can send packets to each other.
- Resulting network of networks is very complex.
- Given millions of access ISP's, how do we connect them together? (E.g. Optus, Telstra, TPG etc.)
    - **OPTION #1: Interconnect all ISP's.(用单一的全球承载 ISP 互联接入所有 ISP)**
        - Can be bad, because connecting each access ISP doesn't scale = $O(N^2)$ connections
    - **OPTION #2: Connect each ISP to a global transit ISP. Customer / Provider ISP's have an economic agreement(数 10 万接入 ISP 和多个全球承载 ISP 构成)**
        - Can be bad, because single point of failure. Attacking the global transit ISP = whole internet breaks
    - **OPTION #3: Multiple global ISP's as different businesses (each are competitors, e.g. Optus / Telstra / TPG)(多层 ISP)**
        - ISP's can have a **Peering Arrangement** with each other to interconnect.
        - **Internet Exchange Points (ISX)** is a building full of high-speed switches / routers which help connect the global ISP's.
        - **Regional Networks** may arise to connect access nets to ISP's. Smaller companies using bigger ISP's.

- **Content Provider Networks** (E.g. Google, Microsoft etc.) may run their own network to bring services, content close to end users. Faster access to services for customers, as their customers don't need to go through multiple hops through the network to reach their services.

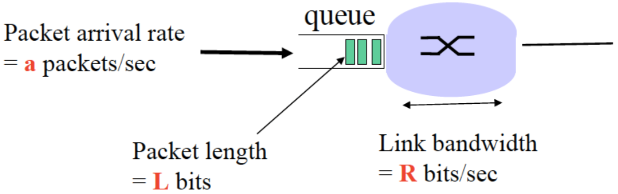
Four sources of packet delay



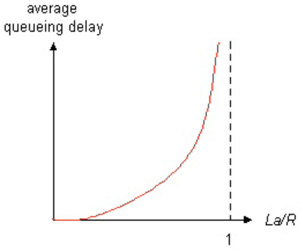
$$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$$

- **处理时延 Nodal processing dPROC**
  - Checking bit errors, determine the output.
  - Typically this process is done very quickly (< millisecond), can mostly be ignored in calculations
- **排队时延 Queuing delay dQUEUE**
  - If there is an overload, the packet will be put into a buffer = time sitting in the buffer
  - Time waiting at the output link for transmission
  - Queuing delay depends on the congestion level of the router.
- **传输时延 Transmission delay dTRANS**
  - The amount of time taken to transmit the **whole packet of size L**.
  - **L:** packet length (bits)
  - **R:** link bandwidth (bits/s)
  - **dTRANS = L / R**
- **传播时延 Propagation delay dPROP**
  - The amount of time taken to transmit **1bit** through the physical link E.g. Fiber optic from SYD → New York
  - **d:** length of physical link
  - **s:** propagation speed in medium (~2 x 10^8 m/sec = speed of light / fiber optic cables)
  - **dPROP = d / s**

More on Queuing Delay



- Every second: **aL** bits arrive to queue
- Every second: **R** bits leave the router
- If aL > R → Queue will fill up, packets drop
  - **Traffic Intensity = aL / R**
- 1 packet arrives every L/R seconds
  - **Arrival Rate (a)** = **1 / (L/R) = R/L** (packets per sec)
  - If arrival rate is R/L and Bandwidth is R, traffic intensity = (R/L) * (L/R) = 1
- What is the Average queuing delay (queue is empty at time 0)?
  - **{0 + L/R + 2L/R + . . . + (N-1)L/R} / N = L / (R*N) {1 + 2 + . . . + (N-1)} = L (N – 1) / (2R)**
  - 1st packet has no delay, 2nd packet has to wait for 1 packet, 3rd has to wait for 2 and so on.
  - General rule of thumb: **Traffic Intensity = 1 is BAD / at 0.8 you should be fixing or expanding your network**
- **La/R ~ 0**: average traffic intensity. Queuing delay is small.
- **La/R approaches -> 1**: delays become large
- **La/R > 1**: more "work" than can be serviced, average delay is infinite (when **a (arrival rate) is random**) = collapse of sys



"Real" Internet delays and routes

- What do real internet delay and loss look like?
- **Traceroute** program provides delay measurement from source to router along end-to-end Internet path towards dest.

For all *i* routers:

Send 3 packets towards destination router → router returns packets → interval between transmission / reply is timed

Section 1: Application Layer

DNS:

Resource: tuple of len 4(name, val, type, ttl)

- Type=A => Name: local domain name, val: IP
- Type=NS => Name: domain edu.au , val: host domain name
- Type=CNAME => Name: 别名, val: 真名
- Type=MX, youjian

Section 2: Transport Layer

Transport Layer: Responsible for delivering data to applications on host computers.

Multiplexing / De-multiplexing

Multiplexing: Multiple data streams from different sources are combined and transmitted over a single shared medium.

- Handles data from multiple sockets, adding transport header which is used in de-multiplexing.

De-multiplexing: At the receiving end, the reverse occurs, separating data streams from the single channel and routing to corresponding receivers / destination.

- Use the transport header info to deliver received segments to the correct socket.

Connectionless De-multiplexing (UDP)

- IP datagrams with the same dest port but different source IP/port will be directed to the same socket at the dest.

Connection-orientated De-multiplexing (TCP)

- Receiver uses all four values of TCP 4-tuple to direct segments to the appropriate socket.
- A server host may support simultaneous TCP sockets.
- Web servers have different sockets for each client. Non-persistent HTTP will have a different socket for each request.

UDP: User Datagram Protocol

UDP is a communications protocol used primarily for establishing **low-latency** and **loss-tolerating** connections between apps. It is **connection-less**, where each UDP segment is handled independently of others.

UDP Header (8 bytes): SRC PORT # | DEST PORT # | LEN (BYTES) | CHECKSUM | PAYLOAD

UDP Checksum

- Treat segment content + header fields as a sequence of 16-bit integers.
- Checksum = binary addition of segment contents (sum) → invert the bits to get checksum (complement of sum)
- Sender puts checksum value into UDP checksum field.
- Receiver adds all segment content with the checksum. Result should = 1111 1111 1111 1111, else there are errors.

Reliable Data Transfer (RDT) – STOP AND WAIT PROTOCOLS

With RDT, transferred data is **NOT CORRUPTED | NOT LOST | DELIVERED IN ORDER**. TCP offers this service model to apps.

RDT 1.0 – Transfer over a perfectly reliable channel (not a realistic model)

- All packet flow is from sender → receiver, no need for receiver-side to provide feedback to sender.
- Assume the receiver is able to get data as fast as the sending of data, thus no need for flow/congestion control.

RDT 2.0 – Transfer over a channel with bit errors (more realistic model)

- In this model, we assume packets can be corrupted.
- Recover from errors through **ARQ: Automatic Repeat Requests Protocols**.

ARQ - Stop-and-Wait Protocol: Sender sends packet and waits for an ACK or NACK. ACK = not corrupted | NACK = corrupted.
Sender can't receive more data from upper layer while waiting.
Flaw with Stop-and-Wait: ACK/NACK can be corrupted themselves.
Solution to flaw: Number packets with a sequence number #0 or #1

RDT 2.1 – Protocol includes sequence numbers #0 #1 to track expected packets

- Sender: Check ACK/NACK + Remember whether expected packet = seq #0 or #1
- Receiver: See if packet is a duplicate (duplicate if expected seq# != received seq#).

RDT 2.2 – NAK-free protocol

- Same as 2.1 but only using ACKs. Instead of sending NACK, send the same ACK for the last successfully received packet.
- E.g. Server PKT_0 → Client | Client ACK_0 → Server | Server PKT_1 → Client | **Client ACK_0 → Server [NACK]**

RDT 3.0 – Transfer over a channel with bit errors and loss

- New assumption: In addition to bit errors, the channel can also lose entire packets.
- New concerns: (1) How to detect packet loss? (2) What to do when packet loss occurs?

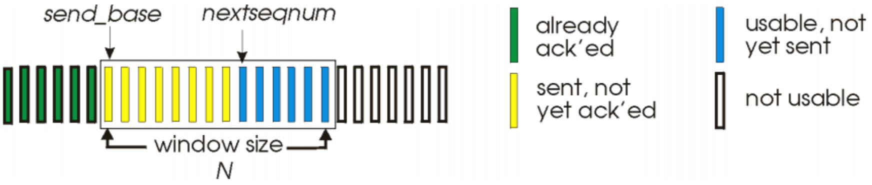
- **Time-Based Packet Retransmissions** that can interrupt the sender after an amount of time waiting for an ACK expires. The sender will need to:
  (1) Start the timer after each packet
  (2) Respond to timer interrupt – take appropriate actions i.e. retransmit packet
  (3) Stop the timer
- Retransmission is an all-in-one solution: doesn't matter if packet is LOST or LARGE DELAY. Seq #s will handle duplicates.

Reliable Data Transfer (RDT) – PIPELINED PROTOCOLS: Go-Back-N, Selective Repeat

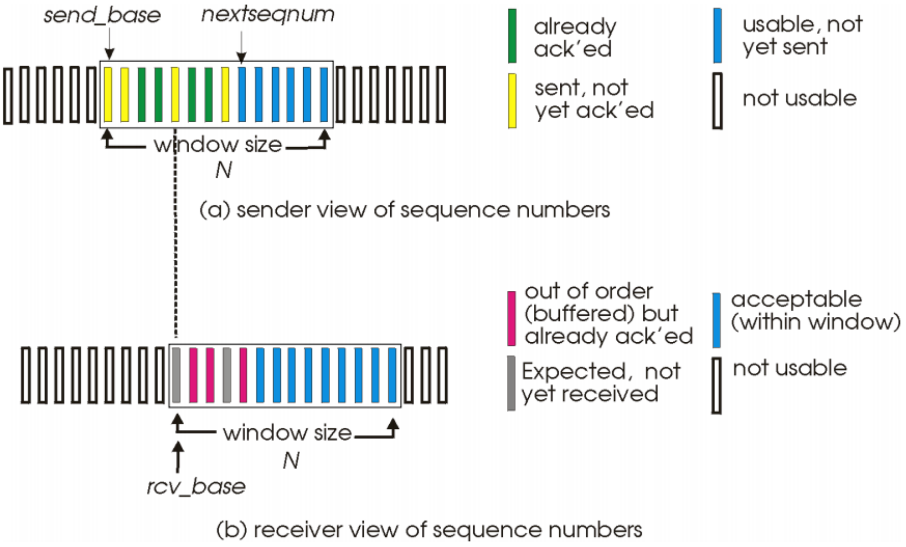
Pipelined protocols allow for multiple "in-flight", un-acknowledged packets and increases utilisation.

Go-Back-N (GBN) Sender continues to send pkts specified by window-size N without receiving ACKs.	**Selective Repeat (SR)** Receiver individually ACKs all received pkts. Buffers packets for eventual in-order delivery to the upper layer.
• Sender window size N of consecutive un-ACK'd packets. • On timeout/loss of packet P: Receiver – discards P + resend ACK of last successful pkt. Sender - retransmit all pkts of higher seq# in window. • On success: Advance send_base	• Sender window size N consecutive seq #s. • On timeout/loss of packet P Receiver: buffer the out of order pkt. Sender: retransmit P only • On success: Send P + all following in-order packets Advance send_base

GBN Sender Window:



Select Repeat Sender / Receiver Windows:

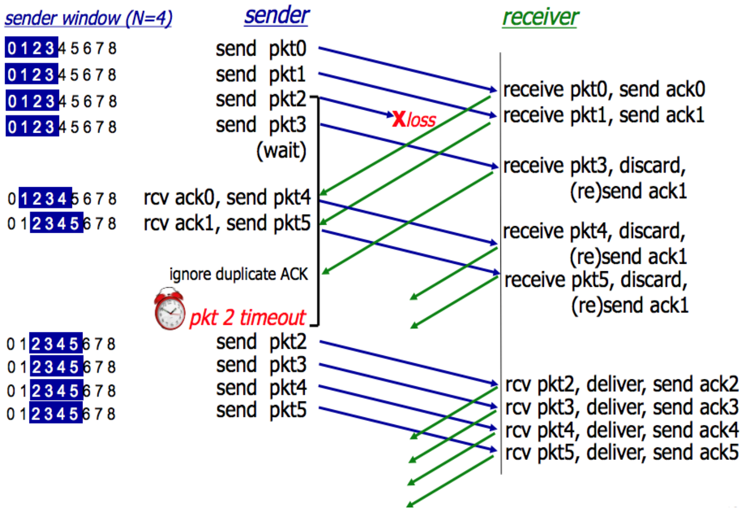


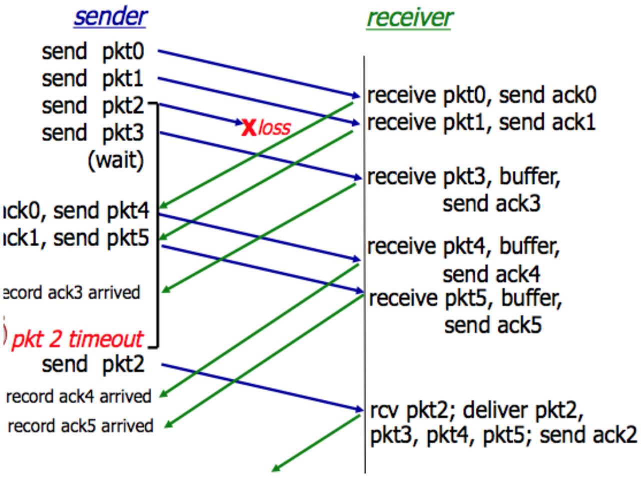
(a) sender view of sequence numbers

(b) receiver view of sequence numbers

Go-Back-N

Selective Repeat

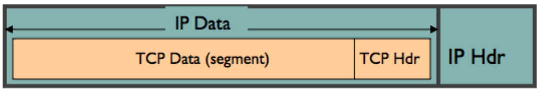




Transmission Control Protocol (TCP) – Segment structure

TCP Header (20 bytes): UDP fields + seq#, ack#, receiver window #bytes, connection establishment + teardown, options.

TCP Packet



IP Packet: No bigger than **Max Transmission Unit (MTU)**

TCP Data/Segment: No more than **Max Segment Size (MSS)**

MSS = MTU – IP Header – TCP header

TCP – Process, Timer + Retransmissions

TCP Sender / Receiver Process:

- Sender: Sends packet of SEQ# = X. Packet len = B bytes [X , X+1 , X+2 . . . X + B–1]
- Receiver: If data prior to X has already been received, send ACK# = X+B
  X+B = next expected seq # from the next packet.
  If highest-order byte received is Y, where Y+1 < X → resend ACK Y+1
- Next Seq# = ACK#

What else can TCP do?

- Receivers can buffer out-of-sequence packets like Selective Repeat / NOT drop out-of-seq packets.
- Senders can maintain a single retransmission timer like Go-Back-N and retransmit on timeout.

Set up TCP timeout by choosing a value > RTT.

- Choose value too short: premature timeout, unnecessary retransmission
- Choose value too long: slow reaction to segment loss and lower throughput for connection

1. Measure EstimatedRTT:

Exponential Weighted Moving Average

$$\text{EstimatedRTT}_{\text{CURR}} = (1 - a) * \text{EstimatedRTT}_{\text{PREV}} + a * \text{SampleRTT}_{\text{RECENT}}$$

- SampleRTT
  - Time measured from segment transmission until ACK receipt (ignoring retransmissions)
  - Current value of RTT
- Typical value of a = 0.125

2. Measure timeout interval: EstimatedRTT + "Safety Margin"

RTT Deviation is calculated by

$$\text{DevRTT} = (1 - b) * \text{DevRTT} + B * |\text{SampleRTT} - \text{EstimatedRTT}|$$

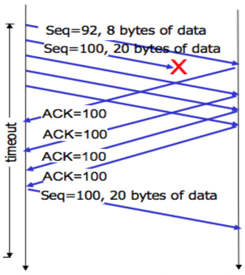
- Typical value of b = 0.25

Timeout Value is calculated by

$$\text{Timeout Interval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$

- 4 * DevRTT = The "Safety Margin"

TCP – Fast Retransmission

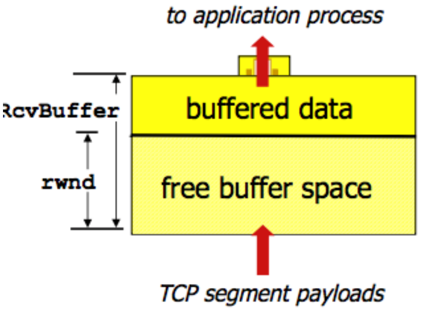


fast retransmit after sender receipt of triple duplicate ACK

TCP has a Fast Retransmissions feature that uses duplicate ACKs to trigger early retransmission.

- If sender receives 3 duplicate ACKs for the same data, resend the un-ACK'd data with the smallest sequence #.
- Timeout periods are often long, so there is a long delay before resending lost packets. No need to wait for timeout.

TCP – Flow Control



TCP Flow Control is where the receiver controls the sender, so the sender won't overflow the receiver's buffer by transmitting too much, too fast.

Receiver Advertised Window (RWND): Advertises available recv buffer space in the RWND value in TCP header.

Sender limits the amount of un-ACK'd data to receiver's RWND value.

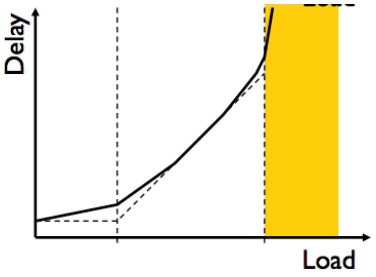
TCP – Connection Management

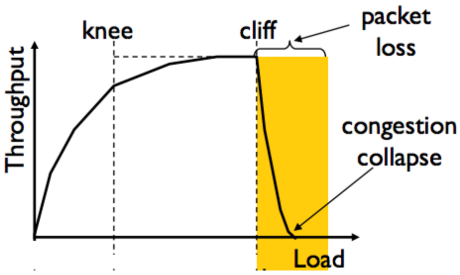
C Establishment: (1) SYN → (2) SYN-ACK → (3) ACK + DATA → Data exchange
C Teardown: Data exchange → (1) FIN → (2) ACK-FIN → (3) ACK → (4) WAIT / Retransmit ACK → (4) CLOSE CONNECTION

RST: Reset Flag. Possibly because application has crashed on one end, or socket is closed, or there is a firewall.

TCP – Congestion Control

Congestion Control is needed if a network node/link/router is taking in more data than it can output, leading to collapse.
Congestion Collapse: Throughput starts to drop to zero, delays approach infinity.





Knee Point
Throughput increases slowly
Delay increases really fast

Cliff Point
Throughput begins to drop to zero
(Congestion collapse)
Delay approaches infinity

CWND: Congestion Window i.e. how many bytes can be sent without overflowing routers?

- Sender varies the window size to control the sending rate.

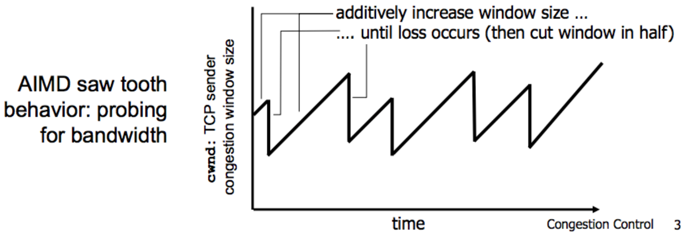
TCP sending rate =~ CWND / RTT bytes per second.
Sender-Side Window: minimum { RWND , CWND }
Basics of sender rate adjustment:

- Upon receiving ACK → increase rate
- Upon detection of loss → decrease rate

(1) Bandwidth Discovery with Slow Start (SS)

- When connection begins, initial rate is slow (for safety) then increase exponentially until the first packet loss event.
- <u>Initial CWND = 1 MSS</u> → <u>Double CWND every RTT</u> or alternate <u>Increment CWND for every ACK received</u>

(2) Additive Increase Multiplicative Decrease (AIMD)


additively increase window size …
…. until loss occurs (then cut window in half)
AIMD saw tooth behavior: probing for bandwidth
cwnd: TCP sender congestion window size
time
Congestion Control 3

- Slow start gave an estimate on available bandwidth. Now we want to track variations of this bandwidth via. probing.
- Additive Increase: Sender increase CWND / transmission rate, probing until a loss event occurs
- Multiplicative Decrease: Cut CWND in half after a loss occurs

Slow-Start Threshold (SSThresh) is used to determine when a sender should stop slow-start and start AIMD.

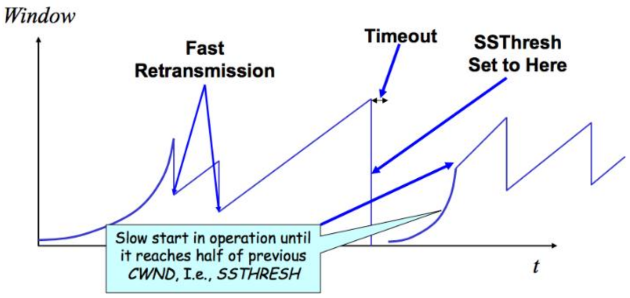
- SSThresh is initialised to a large value. On timeout, SSThresh = CWND/2.
- If CWND < SSThresh = Slow-Start
- If CWND > SSThresh = AIMD / Congestion Avoidance

Congestion Control Rate Increases:

- Slow-Start: CWND += MSS
- Congestion Avoidance/AIMD: CWND += MSS/CWND

Congestion Control Rate Decreases:

- DupACKs:
  SSThresh = CWND/2
  CWND = CWND/2
- Timeout/Loss Event:
  SSThresh = CWND/2
  CWND = 1 MSS

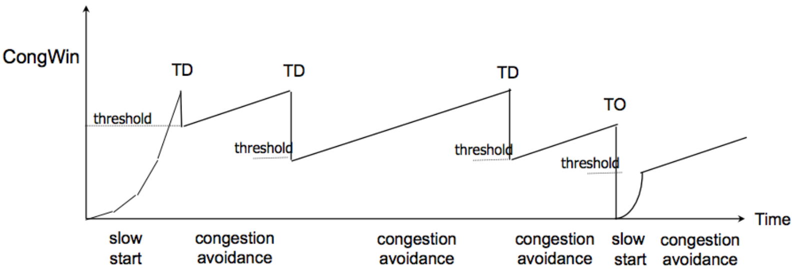

Window
Fast Retransmission
Timeout
SSThresh Set to Here
Slow start in operation until it reaches half of previous CWND, I.e., SSTHRESH
t

Slow-start restart: Go back to CWND = 1 MSS, but take advantage of knowing the previous value of CWND

TCP - Flavours

TCP Tahoe: CWND = 1 on DupACK and Timeout
TCP Reno: Same as above.
TCP New-Reno: TCP Reno + improved fast recovery

TD = Triple Duplicate ACKs
TO = Timeout


CongWin
TD
TD
TD
TO
threshold
threshold
threshold
threshold
Time
slow start
congestion avoidance
congestion avoidance
congestion avoidance
slow start
congestion avoidance