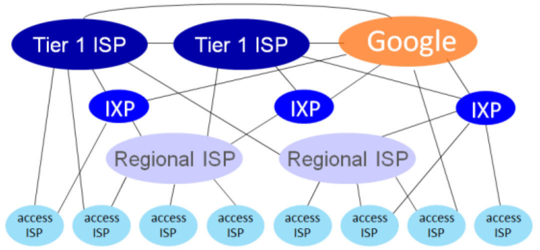
Intro

1Mbps = 0.125MB/s = 1000000bit/s = 125000Byte/s

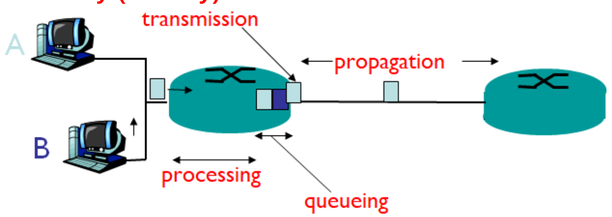
- **Internet structure:** "Network of Networks"



Tier -1 ISP: 类似 AT&T

Content provider network: 例如谷歌的私有网络；bypass tier-1 和 regional ISP

- **Layered architecture**
  - application: supporting network application //HTTP, DNS
  - transport: host-host (socket2~) data transfer //TCP, UDP
  - network: rout of datagram from src to dst //IP, route 协议
  - link: data transfer within neighbor network //PPP, Ethernet
  - physical: bits "on the wire"
- **Network performance metrics**
  - **Delay (latency)**



1. Processing: check bit errors & determine output link
2. Queueing: 在 output link 等待传输的时间；取决于 router 拥堵程度；容量不足 drop
   (packet size/trans rate)*(n-1) //n 是队伍中的第几个
3. Transmission delay: time to send bits into link = L/R

L=packet length (bits); R=link bandwidth (bps)

4. Propagation delay: d/s

d=physical link length; s=propagation speed (~$2x10^8$ m/sec)

若 dprop > dtrans，dtrans 时：first bit 已经离开 A 而没到 B；反之则已经到 B。无论如何 last bit 都已离开 A。

5. End-to-End Delay: 以上四种 delay 的相加

Store-and-forward transmission: 可以查看每个节点并添加

- **Throughput：** 受最低链路带宽限制 min(Ri)

- **circuit switching vs packet switching**
  - circuit(per connection): 网络资源(bandwidth)切成 pieces 分配；若当前用户 inactive，则资源块闲置，不共享
    适合应用于 constant & predictable transmission rate，因为可以保留资源而不浪费 bandwidth。建立和断开 connections 的开销会在长时间内被分摊。
  - packet(per packet): 将数据拆成更小的 packet，每个包按照需求独立处理，适合突发数据；可以解决 transient overloads (congestion: delay & loss)；需要对应 protocols
  - 对比(multiplexing): **link**: 1Gb/s – **each user**: 100 Mb/s when "active"; active 10% of time

Circuit: 10 users;

Packet: 35 users, P(>10 active)= .0004

$$Pr(users = k) = \frac{n!}{k!(n-k)!}p^k(1-p)^{n-k}$$

Application Layer

- **Transport layer service models: TCP vs UDP**
  - **UDP:** unreliable 数据传输；sender 自订发送速率；适合实施交付可容忍损失（如音视频）；丢失数据无法恢复
  - **TCP:** 连接可靠，传输有序；保证数据接收；**Flow control；Congestion control;** 需要 set up overhead
- **Application architectures: Client-server vs P2P**

CSA: 专用服务器（最常见我们所用的）- server 始终开启连接到网络，ip 固定公开，位于数据中心确定端口接受请求；client 不需要固定公开 ip，不直接交流

P2P: 用户直接交流，并不一定总在线

- **HTTP**: Stateless & text-based protocol for web pages //资料无法存储，用 cookies
  - **Non-persistent：单个 TCP 连接最多一个对象**
    vs persistent: 单个 TCP 连接允许多个对象
  - **Parallel connections**: 同时请求多个对象
    Pipelining：发送多个请求而不等待响应来减少响应时间

Non-persistent, serial: ~ $2RTT\ x\ n$

Non-persistent, m parallel connections: ~$2RTT\ x \left\lceil \frac{n}{m} \right\rceil$ (向上取整)

Persistent (non-pipelined): ~ $(1+n)\ x\ RTT$

Persistent, pipelined: ~ $2\ x\ RTT$ for first set of requests, RTT after connection established

RTT=2* one-way propagation delay

total object 数量+1(for base file); //作业问题

- **Http2** multiplex: 对象分帧，不同对象的 frame 可以交错
- **Caching & CDNs：** 临时存储已检索的结果于（如 browser）

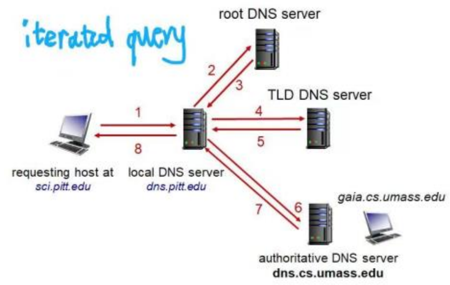
ISP 在网络中部署 proxy servers(许多用户访问相同内容从而增加命中几率)

LAN utilization = access link rate/LAN rate

access link utilization = avg data rate to brw/access link rate

Content Distribution Networks: 存储内容副本于 node；用户就近从 CDN 请求副本

- **DNS:** map hostname(url) to IP(机器识别)
  Runs over UDP on port 53；含有映射的分布式数据库
  - **Hierarchical organization of namespace & servers**



root: 域名解析的根本

tld: .com/.edu

authoritative: pitt

local: 不属于层次结构，充当 proxy

例：检索网页最小 DNS 查询数？3or4 (是否绕过 local DNS Server)，用 **recursive query**

DNS Caching: 每个级别缓存对应响应

Transport Layer

- **Network layer service model**
- **UDP Multiplexing:** sent in to socket 时：指定 dst IP 和 dst port；host 接收时检查 segment

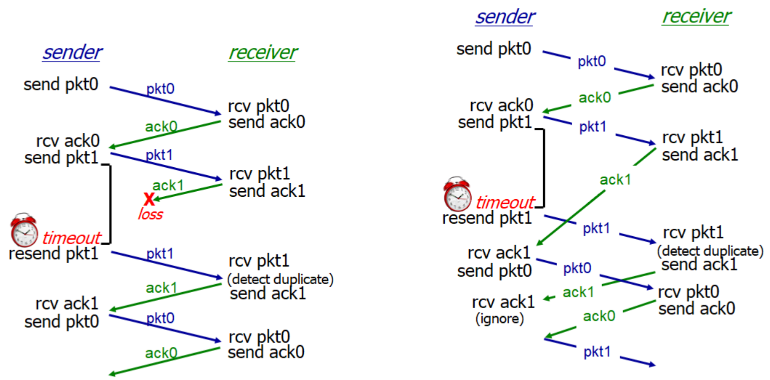
中的 port，再将 segment 传给对应 port 的 socket

/TCP demultiplexing：socket 由 src IP, src port, dst IP, dst port 所定义；用这些值把 segments 指引给 sockets

- Error detection：CheckSum(received segment 中 bit 翻转)

Sender 根据 segment 计算出值将其存于 cheksum filed；receiver 进行相同计算；若不匹配，则 segment 损坏 //切成 16-bit words，相加，取反，放入 field；rcv 到的+checksum 为 1__1 即可

- Reliable data transfer（unreliable channel）
  - ACK/NAK: sender 用于确认 receiver 是否收到正确 packet 的标识符；NAK 需要 resend the packet
  - Retransmission: sender should **retransmit** to recover a lost/corrupted packet，timeout 后 resend oldest unacknowledged segment
  - timeout: 通道可能会丢失数据，无响应就 resend，可能导致 duplicate packet
    检测（可能的）丢失，决定何时重传
  - sequence number: 识别 packet，区分新旧，用 1-bit 验证，允许重复数据删除
  - stop-and-wait: sender 发送一个数据包，然后等待 receiver 的反馈；如此往复（以上机制都是为了实现它）



t = RTT + L / R

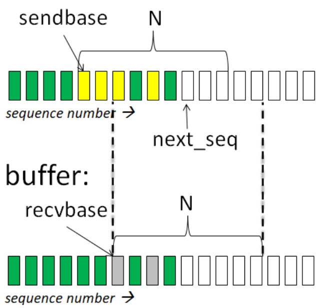
?如何提升性能 Pipelining

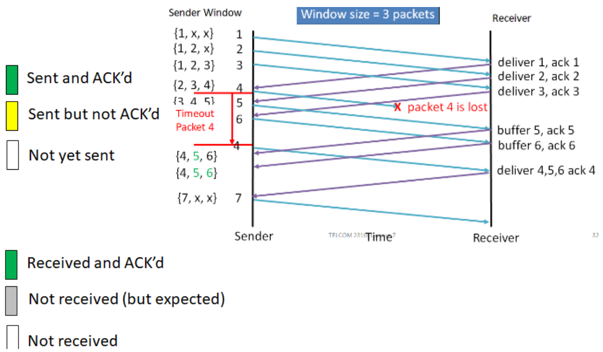
Window: sender buffer(**up to N unacknowledged packets**)

- go-back-N: sender 一次发 N 个未确认的 packet；receiver 发送累积的 ACK；timeout 时，resend 所有 NAK packets
- selective repeat: Receiver individually acknowledges each correctly received packet and buffers out-of-order packets; Sender maintains separate timer for each un-ACK'd packet to retransmit individually

Sender buffer:

Receiver buffer:





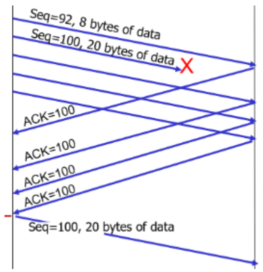
- RTT Estimation: Timeout 应该接近 RTT，因此要预估

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

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

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

Safe Margin



- fast retransmit：TCP 一旦接收到 **3 duplicate ACKs of a segment**, retransmit it，而无需等待 timeout

重复的 ACK 是孤立丢失的标志

fifth packet (seqno 500) is lost, but no others; ACKs will be: 200, 300, 400, 500 (seqno:600), 500 (seqno:700)…

- Flow Control: sender 限制 NAK packets，防止过载 receiver
- Congestion Control: sender 限制 NAK packets，防止过载 network

Sender 如何推断存在拥塞？ - Assume **loss** implies congestion

Sender 如何调整 sending rate in response？ - Maintain a **window**，当检测到拥塞时收缩（当没有检测到拥塞时增加）

Adjust window size 来限制 sending rate: -探测 **available bandwidth**；-ACK: segment rcved → network not congested → increase window size；-Lost segment: (timeout or 3 duplicate ACKs) →network is congested→decrease window size

Event	State	TCP Sender Action	Commentary
ACK receipt for previously unacked data	Slow Start (SS)	cwnd = cwnd + MSS, If (cwnd > ssthresh) set state to "Congestion Avoidance"	Resulting in a doubling of cwnd every RTT
ACK receipt for previously unacked data	Congestion Avoidance (CA)	cwnd = cwnd + MSS * (MSS/cwnd)	Additive increase, resulting in increase of cwnd by 1 MSS every RTT
Loss event detected by triple duplicate ACK	SS or CA	ssthresh = cwnd/2, cwnd = ssthresh, Set state to "Congestion Avoidance"	Multiplicative decrease. cwnd will not drop below 1 MSS.
Timeout	SS or CA	ssthresh = cwnd/2, cwnd = 1 MSS, Set state to "Slow Start"	Enter slow start
Duplicate ACK	SS or CA	Increment duplicate ACK count for segment being acked	cwnd and ssthresh not changed

Ss 阶段指数级快速上升；CA(normal)阶段加法级增加、乘法级下降。

Timout：系列丢失；回 SS re-do bandwidth 预测 // ssthresh = cwnd/2; cwnd = 1 MSS

3dupACK：孤立丢失；微调 // ssthresh = cwnd/2; cwnd = cwnd/2 + 3 MSS

slow start threshold：慢启动阈值，决定是否 SS

cwnd：堵塞窗口