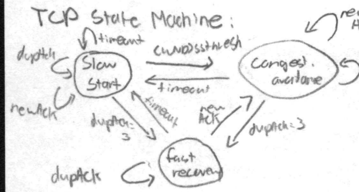
Congestion Control

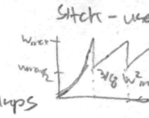
Internet traffic is bursty, may cause delays/drops
TCP has endhosts adjust rate, feedback by ACKs/drops
Goals: Discover bottleneck bandwidth, adjust to variations in bandwidth and share bandwidth between flows
TCP uses dynamic adjustment for rates but assumes that endhosts will be good citizens
Congestion Window (CWND): bytes w/o overflowing routers
Advertised window (RWND): Receiver -> sender, buffers
Sender-side window = minimum{CWND, RWND}
① Slow Start - Grow window exponentially by +1 for each ACK received, until a loss is experienced or ssthresh is reached
② AIMD - Increase window by $\frac{1}{CWND}$ per ACK and CWND = CWND/2 on packet loss
On timeout, set ssthresh = CWND/2, CWND = 1
AIMD allows us to achieve efficiency and fairness
With dupACKs, if 3 then ssthresh = CWND/2, CWND = CWND/2 (fast retransmit)
Fast Recovery - if dupACK count == 3, ssthresh = CWND/2, CWND = ssthresh + 3
CWND += 1 for each dupACK and CWND = ssthresh w/ new ACK to exit

TCP State Machine:



Tahoe
CWND = 1 on 3 dupACK
Reno
CWND = 1 on timeout
CWND /= 2 on 3 dupACK
new Reno
TCP-Reno + improved fast recovery
SACK - use SACKs as well

TCP Throughput Equation
Average ¾ Wmax / RTT per packet, ½ Wmax RTT between drops



$$\text{Throughput} = \sqrt{\frac{3}{2}} \frac{1}{RTT\sqrt{p}}$$

① TCP unfair if we have different RTT's
② TCP requires extremely low drop rate to have high speed, pipe/persis. We can increase AIMD faster after thresh, use multiple, or router-assisted
③ Can follow TCP throughput to send at constant rate

TCP Problems
① Will cut rate even for corruption rather than congestion
② Short flows never leave slow start, hard to trigger retransmit
③ TCP fills up queues by default, which delays everyone!
④ Can cheat by increasing window faster, open mult. connections, start w/ large CWND
⑤ CC == reliability (maybe try SACKs?), complicates evolution
Routers can fix by telling endpoints about congestion, give rate to send at and by enforcing fair sharing

Routers and Fairness
Packets are classified into flows and served fairly on definition
Max-min fairness: $a_i = \min(f, r_i)$, f such that $\text{sum}(a_i) = C$, so if you don't get full demand, no one gets more
Fair Queuing: Compute time for last bit to leave router if served bit by bit and serve by ↑ order of deadlines
Allows cheat prevention, bandwidth not dep on RTT. Flows can pick any rate adjustment scheme they want, but more complex
RCP: Packets have "rate field", f fair share, endhosts set rate to f
ECN: single bit in header for congestion, can interpret as drop
Helps distinguish corruption (congestion and early indicate congest.)
Perhaps one can also charge money for ECN
Highlights importance of performance, shows we can do better
RC3: Transmit high but use priorities (CWND #1)

Application Layer

Host Names and Addresses (DNS)
Addresses used by protocols ("where"), host names by people ("who")
DNS maps to each other, directory service that decouples them
We want scalability, availability, correctness and performance
Partition namespace, distribute partition administration and name resolution
Hierarchical namespace, administration and servers
Namespace: "top level domains", domain = subtree, name: leaf-to-root path
Administration: Each zone corresponds to admin for that part
Servers: Root servers -> Top-level domain servers -> Authoritative DNS serv
All know DNS addr, DNS knows TLD, auth DNS stores name-to-addr ("resource records") for own, anyone can discover server in hierarch.
Root server replicated by anycasting: deliver packet to closest machine
DNS resource records -> (name, value, type, TTL)
Type A (Address) name = hostname, value = IP addr; Type = MX (mail exchanger) name = domain in email addr, value = name of mail server
Type NS (Name server) name = domain, value = name of domain DNS server
Insert RR by register domain name w/ IP addr + name of auth name server, then RR added to TLD server, store RR in your own server

HTTP and the Web

Uniform Record Locator (URL) protocol://host-name[:port]/directory-path/resource
protocol: http, ftp... host: DNS, IP... port: default to standard, dir, path: hierarch., file sys resource
HTTP (TCP, Port 80)
Client-server architecture (server "always on" and "well known")
Synchronous request/reply protocol, stateless, ASCII format
① Est connection (TCP SYN, SYNACK) ② Client request (TCP ACK + HTTP GET)
Request line: method, resource, and protocol version
Request headers: provide information or modify request
Body: optional data (e.g. to "POST" data to the server)
③ Request response ④ TCP FIN/FINACK
Status line: protocol version, status code, status phrase
Response headers: provide information, Body: optional data
Statelessness and Cookies
Treat each request-response independently
Improves scalability on server-side (failure easier, higher rate, no order dep.)
but some applications need persistent state -> cookies
Cookies: client stores small state for server and sends in future
Performance goals (User: fast + available, content provider: happy user, cost-effective, Network: avoid overload)
Improve HTTP to compensate for TCP, caching + replication, economies of scale
Concurrent Requests and Responses
Use multiple connections in parallel w/o regard to order
Good for client and content provider, bad for network
Persistent Connections
Maintain TCP connection across multiple requests
Avoid setup/teardown overhead, get accurate RTT, allow TCP CWND ↑
This is the default in HTTP/1.1
Pipelined Requests and Responses
Batch together requests and responses to reduce number of packets, allowing for multiple requests in one TCP segment
Comparison Between Small and Large Objects

	Small	Large
	Time dominated by latency	Time dominated by bandwidth
one-at-a time	~2nRTT	~nF/B
Concurrent	~2[n/m] RTT	~[n/m] F/B
Persistent	~(n+1) RTT	~nF/B
Pipelined	~2RTT	
	~2RTT, then RTT	

Caching
How? Modify GET to include "If-modified-since", reply not or OK w/ latest using Expires (how long cache, TTL), No-Cache (always ask server)
Where? Reverse Proxies: Close to server to decrease server load, content provider
Forward Proxies: Close to clients to reduce network congest. + latency, ISP
Replication: Copy across many machines, load balance by DNS response
Content Dist Networks: Caching and replication as a service, pull caching at client req and push replication high access rate material
In cost-effective cont delivery, usually see multiple sites hosted on same shared physical infrastructure allowing use of statistical mult, economies of scale, amortization of human operator costs

Data Link Layer

Broadcast Medium Access
Random Access MAC Protocols
Transmit w/o coordination but know how to detect/recover from collisions
Aloha signaling: site sends packet to hub via uplink, hub broadcasts packet
Carrier Sense Multiple Access (CSMA)
Transmit entire frame if channel idle else defer
CSMA/CD: collision detected in short time and abort
Limits on max wire length
Carrier sense: wait for idle link
Collision detection: listen while transmit, jamming signal if collision
Random Access: binary exponential back-off - $\{0, \ldots, 2^{m-1}\}$ s after m th collision

$$E \sim \frac{P/b}{P/b + Kd}$$

Large packets, small d -> high efficiency
E ↓ as bandwidth increases, high-speed LAN -> switched

DNS Usage

DNS Usage: obtain DNS name and request local DNS server
Local DNS may query root name server by iterative/recursive
Query sent, reply received over UDP Port 53
DNS servers replicated w/ primary and secondary servers
If > one up, try alternate on timeout, exp backoff on same
Caching of entries at all levels w/in TTL field, effective since top-level servers rarely change
Negative caching can reduce future invalid query time
Attack DNS by impersonating DNS server, DDoS root or TLD servers, poison DNS server cache

Switched Ethernet

Point-to-point links from switches to hosts
enable concurrent communication (no collis., 2ms, length constraint)
Frame: | Preamble | Dst Addr | SRC Addr | Type | Data | CRC |
Preamble: 7 bytes clock synch, 1 byte start of frame
Addrs: 6 bytes Type: 2 bytes protocol Data: 46-1500 bytes CRC: 4 bytes error
Two hosts must exchange frames, want to know where frames start and end by link layer
Sentinel bits (01111110 start, 01111111 end)
bit stuffing, sender inserts 0 after 5 1's
Receiver removes 0 after 5 1's
Medium Access Control (MAC) Address
Flat namespace 48 bits (6 in HEX)
Unique, hard-coded in network adapter, like SSN
First 24 bits by vendor, last 24 assigned by vendor
Routing in Ethernet
No DV/LS due to scalability issues and broadcast Ethernet compatibility, plug-n-play
In broadcast, sender transmits frame on broadcast link and receiver link layer passes frame to network layer if dst = receiver MAC or FF:FF:FF:FF:FF:FF
"Broadcast storm"; want to eliminate loops
Spanning Tree Protocol
Need to pick a root and compute shortest paths to root to form spanning tree
Send messages (Y, d, X) from X, Y as root with distance d
① Each switch proposes self as root
② Switches update view of root
If Y's id < current root: set root = Y and add 1 to shortest distance from neighbor
Root switch sends periodic announcement
Timeout (soft state) if no word from root
Flooding/Forwarding on Spanning Tree
Flood packets to all ports on spanning tree
Learn paths based on flooded packets for efficiency
Map src MAC to incoming port in switch table and store with TTL
Ethernet -> zero config, simple but inefficient, unpredictable

ARP and DHCP

Address Resolution Protocol and Dynamic Host Config Protocol are link layer discovery protocols that allow discovery of local end-hosts + comm w/ remote end-hosts
DHCP
Used to discover own IP addr, netmask, DNS/router IP (first hop default)
① Client broadcast discovery FF:FF:FF:FF:FF:FF
one or more local DHCP servers have info w/ IP addr pool, netmask, DNS server (UDP Port 67)
② One or more DHCP servers responds with "offer" (proposed IP, lease time)
③ Client broadcasts DHCP request message to specify accepted offer and echo parameters
④ Selected DHCP responds w/ ACK
"soft state": addr allocations have lease period, server sets timer and client must request refresh, reset timer if refresh else reclaim address
ARP
Table with (IP addr -> MAC addr) pairs
consult when sending packet else broadcast
If dest is remote, look up first-hop router MAC (IP by DHCP, netmask also by DHCP)
ARP Header

Hardware Type	Protocol Type
Hardw Addr Length / Protocol Addr Length	Opcode
Sender Hard. Addr	Sender protocol Addr
	Target Hardware Addr
Target Protocol Addr	

H Type: 1 - Ethernet
P Type: 0x0800 - IPv4
H Len: 6 - ethernet
P Len: 4 - IPv4
Opcode: 1 - req, 2 - reply

Broadcast, caching and soft state are important

Steps in end-to-end communication
① Use DHCP to discover own IP addr, DNS IP, Router IP (Eth, IP, UDP, DHCP)
② Resolve domain name by having local DNS contact remote DNS (Eth, IP, UDP, DNS)
Use ARP to get MAC addr of router (Eth, ARP)
Then send the DNS query to your DNS server
③ DNS server returns appropriate IP address
④ HTTP to get content from the server (Eth, IP, TCP, HTTP)

Wireless

Frequency and wavelength affect distance (free-space loss), penetration/reflection, antenna size, energy proportionally
Characteristics: broadcast medium, cannot receive while transmitting, signals don't always arrive intact
Free Space Path Loss: $\left(\frac{4\pi d}{\lambda}\right)^2 = \left(\frac{4\pi d f}{c}\right)^2$
Also consider multipath effects
The lower SNR, higher the bit error (power vs. range, interf.)
If loss, lower bitrate if free-space/multi-path fading, raise if external interference (short burst)

802.11 Architecture
- Access points (AP) for specific channels
- Broadcast beacon messages w/ SSID (Service Set Identifier) and MAC address periodically
- Hosts scan all channels to discover AP's, then associate

CSMA/CA
- Hidden Terminals (A) (B) (C)
  - A and C cannot hear each other, so carrier sense ineffective
- Exposed Terminal (A) (B) (C) (D)
  - B and C can hear each other and may refrain from transmitting even though zero chance of collision (from CS)
- No concept of global collision, collision only at receiver
- Goal: check if receiver can hear, tell interfering senders to stop
- ① Sender sends Request to Send (RTS) frame giving length of transmission and destination
- ② Receiver responds with Clear to Send (CTS) frame
  - other senders who hear shut up until ACK -> over
- ③ Sender sends data ④ Receiver finishes w/ ACK
- Can also prevent collisions by partitioning frequency spectrum into multiple channels, or partition space into non-overlapping cells

Multi-Hop Wireless Ad-hoc Networks
- Ideally w/ self-healing and multi-path routing
- ① Node cannot use all links at once due to sharing
- ② Overlapping ranges further decrease utilization, contention from RTS and CTS
- ③ TCP ACKs produce bidirectional traffic
- Store and forward technique halves hop bandwidth, doubles latency and increases interference

Datacenters

Organization: servers in racks, headed by "Top of Rack" switch
Aggregation switches connect ToR switches, to outside by core switches -> 2x redundancy for fault tolerance
Why?: Scale changes, new applications: "big data", customer-facing revenue generating services
Service model: cloud, and multi-tenancy
Scale: Need scalable design (no flooding), low cost, high utilization, tolerate frequent failure, automation
Service model: performance guarantees, isolation guarantees, portability (achieve through network virtualization)
Applications: High bandwidth "bisection", low latency is critical, worst-case ("tail") latency critical
- Partition Aggregate "North-South"
  - Ext clients and datacenter query/response
  - Front-end web servers, mid-tier app server, back-end database
- Map Reduce "East-West"
  - Traffic between servers in datacenter ("big data" computation) may shift on small timescales (< minutes)
- "Elephants" vs. "Mice" flows (95% flows, 5% bytes, 5% flows 95% bytes)

Bandwidth
- Ideally, have all servers talk to others at full access link rate
- Solution: Build network of switches ("Fabric") w/ high bisection bandwidth (split network in two, minimum bandwidth between any two)
- Full bisection bandwidth: bis. band in N node is N/2 times bandwidth of a single link
- "Scale up" is inefficient for traditional tree topology
- "Scale out" "Clos" topology - multi-stage network where all switches have k/2 ports up, k/2 down, all same speed
- To be efficient, routing must be able to exploit all paths

Now have single administrative domain, control over network and endpoints, placement of traffic source/sink

Datacenters (cont.)

Forwarding
- Per-packet load balancing spreads round-robin
  - Spreads traffic equally but interacts badly with TCP due to drop/reordering, Multipath TCP
- Per-flow load balancing (ECMP, equal cost multi path)
  - Hash five-tuple allows following of single path but can be non-optimal due to elephants

Routing
- Extending DV/LS w/ ECMP allows simple reuse of solutions but is not scalable N -> millions
- ① Topology-aware Addressing
  - Addresses embed location in regular topology
  - "Hard coded" topology allows no computation, constant # entries w/ localized link failure detection
  - But VM migration or misconfiguration may be problematic
- ② Centralize and Source Routes
  - No entries in switch, routing = broadcast from controller -> server
  - Switches simple and scalable, end-points control route selection
  - Scalability/design is often issue

Transport Protocol Design
- Want low queue occupancy and high throughput
- DC-TCP
  - Use ECN (treat like drop), react to extent not presence of cong.
  - Switch: Set ECN bit if Queue len > K (instantaneous)
  - Sender: Maintain avg of fraction packets marked (α) and adapt window to: $W \leftarrow (1 - \frac{\alpha}{2}) W$
    - At each RTT, F = # marked ACKs / Total ACKs => $\alpha \leftarrow (1-g)\alpha + gF$
    - When $\alpha = 1$, produces TCP-like behavior
  - Overall, ↓ latency by avoiding large buildup in queues, maintain high throughput by not over-reacting and reduces variance in sending rates to ↓ queue buildup
- Flow-Completion Time (FCT) [pFabric]
  - Use priorities (remaining flow size)
  - Switches have small queues + send high/drop low priority
  - Servers transmit/retransmit start at high, drop on timeout
  - Similar to job scheduling where "Shortest Job First" is a heuristic to approximate as remaining flow

IPv6

IPv4 address space exhaustion is primary motivation w/ 128-bit rather than 32-bit (8 16-bit blocks), ~ easier
Routing (addr diff), Forwarding (addr, but least prefix matching same)

Version	Traffic class	Flow Label	
Payload Length		next header	hop limit
Source address			
dest address			

version: 6
Traffic class: DSCP/ECN
Flow label: New, ID flows!
Payload length: Everything after
next header: 'Protocol' in IPv4
src: IPv6 src dst
hop limit: TTL

No fragmentation, checksums, optional stuff in extension headers (no options) and can be chained

Addressing
- 64 bits addressable interface and 64 bits interface identifier
- /64 gives one subnet, longest-prefix matching still applies
- Special: host-local (localhost) ::1 ≡ 127.0.0.1
  - link local (not routed): fe80::/10 ≡ 169.254.0.0/16
  - site-local (not routed globally)
  - global unicast
  - multicast
- EUI-64 Identifier: [ab] -> [a|ff|fe|b], flip 7th bit of a

Neighbor Discovery Protocol
- ICMPv6 messages for neighbor/router solicitation/advertisement, redirects to get Layer-2 address

Stateless Address Auto Configuration (SLAAC)
- Network (top 64) obtained through Router Advertisements, Interface Identifier (bottom 64) by EUI-64 or DHCPv6 (stateless/ful)

Software-Defined Networking

Internet built as an artifact w/ complexity
"network management" requires simplifying ideas to build an academic discipline
Define abstractions for the control plane
Want isolation (VLANs), Access Control (ACL: <header, out>) and traffic engineering (centralized computation) (configuration)
Abstraction -> interfaces -> modularity

Control Plane Abstractions
- ① Be compatible with low-level hardware/software
  - Want independence of proprietary HW/SW (forwarding)
  - OpenFlow <header, action> as standardized interface
  - switches accept ext control msg, standard flow entry format
- ② Make decisions based on entire network
  - Abstract (network state) as global network view by annotated network graph through API (NETOS)
  - Runs on servers ("controllers") w/ info to form view and config to forwarding
  - Control by function (view), just a program -> graph algor.
- ③ Compute configuration of each physical device
  - Abstract view of network to simplify configuration
  - Control prog -> virtualization layer -> Network OS
  - SDN separates data and control planes to tackle complexity (simplifies interface, allows reusability)
  - Routing (graph, compute, give to SDN platform, routers)
  - Access control / Control
    - Control program -> SDN platform -> ACL