ECE4016 Computer Networks – CP, Dec 2024

Basics: packet, circuit switching, statistical multiplexing, link characteristics, packet delays. (L1-L2)
Network: a system of "**links**" that interconnect "**nodes**" to move "**information**" between them.
End-system (PC, phone, …) – link – switch – fibers – router. Cannot use N^2 to connect. Need to share switched networks: Circuit switching (per connection, separated resources) / packet switching via statistical multiplexing (per packet, shared, on-demand). Compare: circuit faster once established, predictable performance, no need for congestion control; packet more efficient, simpler, robust for route around trouble.
Evaluation: Delay: transmissionDelay(↓when bandwidth↑, push all bits into link)=packetSize(bits)/transmissionRateOfLink(i.e. bandwidth, bits/sec, bps), propagationDelay(move one bit to end)=linkLen(m)/propagationSpeedOfLink(m/s), queuingDelay(Little's law): [L]enforWaitingQueue=[A]rrivalRate*avg[W]itingTime, count L every second, processingDelay: Negligible 忽略不计, RTT(RoundTripTime=source->destination->back); Loss: fraction of packet being dropped; Throughput: destination receiving data rate, [R]ate =[F]ileSize(bit)/([T]ransferTime-propagationDelay), end-to-end: take min.
OSI layers (Open Systems Interconnection model): L7 Application (& L6 Presentation, L5 Session), L4 Transport (reliable or unreliable, in OS), L3 Network (best-effort global packet delivery, in OS), L2 Data link (best-effort local packet delivery, HW), L1 Physical (physical transfer of bits, HW).
Layers: part of system, with well-defined interfaces to other parts. Only interact with above and below layer, only through the interface. Protocol: define peer layer communication syntax (header+payload). Host/End-system: L1-L7. Switch (local delivery): L1, L2. Routers (global): L1, L2, L3. Pros: complexity↓, flexibility↑; Cons: overhead↑, missing cross-layer info. IP (L3) is the waist of layering hourglass, single, allowing arbitrary networks to interoperate; decouples low-level networking tech; simulation support; hard to change it. End-to-end arguments: Dumb network (keep IP simple) and smart end systems (try put everything here), Fate sharing: fail together or don't fail at all.

Application Layer: HTTP, DNS, CDN, streaming. (L3-L5)
Socket: a *host-local*, *application-created*, *OS-controlled* interface into which application process can **both send and receive** messages to/from another application process; bind to one process/thread. TCP (reliable transfer bytes): Server: socket() create=>bind() optional assign a port=>listen() for incoming conn=>accept() incoming conn, create a new file descriptor=>read()/write()/close(). Client: socket()=>bind()=>connect() to remote server via IP&Port=>read()/write()/close().
HTTP (Hyper Text Transfer Protocol): Stateless (scalable, sequence did not matter / some app need state), ASCII format. Req&Resp: TCP SYN=>TCP SYN+ACK=>TCP ACK+HTTP GET=>…=>Close. Req: GET /somedir/page.html HTTP/1.1**\n**(header)**\n**(blank line)**\n**(data). Resp: HTTP/1.1 200 OK**\n**(header)**\n**(blank line)**\n**(data). Improve performance: **1)** Persistent connects, **2)** parallel/concurrent connections, **3)** pipeline transfer over the same connection, **4)** Caching (forward & reverse), **5)** Replication.
Scorecard: **1)** Getting n small objects, Time dominated by latency: One-at-a-time: ~2n RTT, m concurrent: ~2[n/m] RTT, Persistent: ~(n+1) RTT, Pipelined: ~2 RTT, Pipelined and Persistent: ~2 RTT first time; RTT later for another n from the same site. **2)** Getting n large objects each of size F, time dominated by TCP throughput B_C (<= B_L), where link bandwidth is referred by B_L. One-at-a-time: ~nF/B_C, m concurrent: ~nF/(mB_C) (assuming each TCP connection gets the same throughput and mB_C <= B_L), Pipelined and/or persistent: ~nF/B_C (the only thing that helps is higher throughput).
Caching: Req header: If-modified-since, Resp header: Expires, No-cache. Reverse Proxies: by content provider, decrease load. Forward Proxies: by ISP or enterprise, reduce traffic. Replication: help cannot cacheable content, spread to closer. CDN: Caching and replication, lower latency and load, better speed and price.
DNS: a name-address mapping service. Server hierarchy: Root=>Top-level domain (TLD)=>Authoritative DNS. Server store resource records (RRs): [A]dress, [N]ame[S]erver (name of DNS server), [C]anonical [NAME] (refer to other name), [M]ail e[X]changer. Two ways: Recursive: ask server to do it, Iterative: ask server who is next to be asked. Application-layer protocol: Query and Reply messages, same format: Header (identifier, flags, etc) + resource info. Reliability: replicated server, UDP, same identifier. Caching: time to live TTL to decide whether server delete cache. Negative caching: remember not working address. Properties: Administrative delegation 委托 and hierarchy enables: Easy unique naming, Reasonable trust model, Caching increases scalability and performance. Indirection: change IP underneath, multiple-IP for load balance, alias domain name.

HTTP streaming: Compression: spatial 空间 & temporal 时间 coding. Client sent GET request for the video URL, server sent the video as stream, client first buffers for a while to minimize interruptions later, once reached threshold (foreground: play, background: download more frames). DASH: dynamic adaptive streaming over HTTP, keep multiple resolution with a manifest in HTTP server, client get the manifest first, then asks for chunks and measures bandwidth: low bandwidth=>lower bitrate, high bandwidth=>higher bitrate. SO: Streaming Video=encoding+DASH+playout buffering.
Data center network: Bisection bandwidth: cut network into two equal parts, minimum bandwidth between two parts is the bisection bandwidth; full ~: in N-node network, ~ is N/2*singleLinkBandwidth=>any two halves can communicate at full speed. Traffic Type: North-South: between worker and master, East-West: between worker servers, <mins. Traffic Characteristic: High bandwidth (most bytes comes from large flows, idea: view as a giant switch, practical: build a network of switches ("fabric") with high "bisection bandwidth"), low latency(most flows are small). Traditional: Rack=>Aggregation=>core=>cloud. Low bandwidth(Oversubscription in [T]op[o]f[R]ack-[Agg]regation link), few paths(load balancing, recovery issues). Modern: more bandwidth and paths, e.g., Clos topology: k pods, each pod has two layers of k/2 switches, k/2 up and k/2 down; all link same b/w; at most K^3/4 machines.

Transport Layer: TCP, UDP. Reliability and flow control. Congestion control. (P34-P53, L6-L9)
Role: 1) communication between application processes: mux and demux from/to application processes, using port implement, 2) provide e2e service for app layer: reliable, in-order, well-paced data delivery.
UDP vs. TCP: data abstraction (packets⇔byte stream), service (best-effort⇔reliability, in-order, congestion control, flow control). Reliable transport: Checksum for error detection, timers for loss detection, acknowledgements for data (cumulative or selective claim data), sequence number (detect duplicates, accounting), sliding windows for efficiency. Two way: Stop and wait, correct but inefficient, throughput is DATA/RTT; sliding window, using pipelining, n packets at a time to increase throughput, MIN(n*DATA/RTT, link bandwidth). Reliability: +fast retransmit to detect faster-than-timeout loss, cumulative ACKs,

buffers at sender sent until ACKs arrive, buffers at receiver to reorder packets before sent to APP. Connection establish/terminate: three-way handshake: A=>SYN=>SYN+ACK=>ACK. Three-way handshake to terminate. Congestion Control: why: network might be bottleneck, sharing capacity with other flows and adapt to available capacity changes; how: restrict window to CWND to make sure network not overwhelmed; implementation: sender (CWND small constant, ssthresh large constant, dupACKcount, timer), event (ACK for new data: slow start phase{CWND++ if CWND < ssthresh else CWND = CWND + 1/CWND}=>congestion avoidance phase, dupACK for old data: dupACKcount++==3 then ssthresh=CWND/2 & CWND=CWND/2 [fast retransmit], timeout: ssthresh=CWND/2, CWND=1). Flow Control: why: receiver buffer may be overwritten, need to control sending speed; how: "advertised window" filed in TCP header, restrict windows size to RWND. CC+FC: min{RWND, CWND}, CWND: congestion window, ssthresh: Slow Start Threshold. TCP variants: **TCP-Tahoe** resets CWND to 1 on 3 dupACKs; **TCP-Reno** resets CWND to 1 on timeout and halves CWND on 3 dupACKs; **TCP-newReno** = **TCP-Reno** + improves fast recovery; **TCP-SACK** incorporates selective acknowledgments for better efficiency. TCP throughput model=3/8*W_{max}^2.

Network Layer: Routers, Intra-domain routing, Inter-domain routing. (P55-P91, L10-L15)
Network layer/IPv4: Present everywhere, perform addressing, forwarding (data plane, directing one data packet using local routing state), and routing (control plane, jointly computing forward tables that guide packet by routers using distributed algorithm). IP header: Parse the packet by extracting the IP version (4 bits), packet length (16 bits), destination IP address (32 bits), and handling issues like TTL (loops, 8 bits), checksum (corruption, 16 bits), and fragmentation fields (packet too large, 32 bits). IPv6 Header: IPv4 (version, header length, type of service, total length, identification, flags, fragment offset, time to live, protocol, header checksum, source and destination IP addresses) vs. IPv6 (version, traffic class, flow label, payload length, next header, hop limit, and source and destination addresses): simplified structure (new options mechanism: next header, no header length) and no checksum/fragmentation(save TTL, leave these to end).
Router: data=>input linecards(challenge: processing speed, task: update packet header & [L]ongest[P]refix[M]atching in tree look up dest addr)=>Interconnect/Switching Fabric(crossbar interconnect, row: N input, col: N output, non-blocking)=>output linecards(packet classification: map to flows, buffer management: drop packet, scheduler: when and which to transmit QoS). Max-Min fairness: first consider to fulfill the min req, then avg to others. Given demand r_i, total bandwidth C, allocation: $a_i = \min(f, r_i)$, where f is the unique value such that $\mathrm{Sum}(a_i)=C$. State: Local routing state (forwarding table in a single router, cannot evaluated) vs. Global state (all routers' forwarding table=>path, valid if produce forwarding decision that deliver to dest). Valid routing state determine: valid⇔no dead ends(next-hop, outgoing link), no loops(cycles). Least-cost routes: avoid loops, generate a spanning tree for each dest (dest as root).
Intra-domain routing 域内路由: routing and forwarding within [A]utonomous[S]ystem(follow same policy). Algo: [L]ink-[S]tate(Dijkstra's algo, broadcast neighbors' info to everyone, [O]pen [S]hortest [P]ath [F]irst) vs. [D]istance[V]ector(Bellman-Ford algo, gossip to neighbor about everyone, [R]outing [I]nformation [P]rotocol): 1) both are shortest-path based routing, sharing same regulation judge cost metric "link weights", used inside organization; 2) messaging: LS: N node, E links, O(NE) message; DV: exchange between neighbor; 3) convergence speed: LS: fast, DV: varies, Count-to-infinity problem; 4) robustness: LS: can advertise incorrect link cost, node can computer own table, DV: can advertise incorrect path cost, page used by other=>errors propagate. Address aggregation: is crucial for scalable inter-domain routing: state(smaller forwarding table), churn(limited change rate). Classful addressing: Class A (8-bit network prefix), B (16-bit, ~65k), C (24-bit, <500);
[C]lassless[I]nter[D]omain[R]outing: flexible division between network and host addresses, better tradeoff {routing table size}⇔{efficient use of IP address space}. Hierarchy in IP addressing: 32-bits partitioned into prefix(network, for inter-domain) and suffix(host). Inter-domain routing 域间路由: administrative structure shapes this, ASes want pick routes based on policy, autonomy, privacy. Business relationship: traffic: provider=>customer (paid), peer=>peer (don't pay). BGP differs from Distance-Vector routing: allowing policy-based route selection, using Path-Vector to prevent loops, enabling selective route advertisement that can affect reachability, and supporting route aggregation for scalability. BGP: an AS advertises/exports its best routes to one or more IP prefixes, each AS selects the "best" route it hears advertised for a prefix. BGP's selection (whether/how data leave the AS) & export (whether/how data enter the AS). Policy: (dest prefix advertised by) customer=> (export to) all; peer=>customer; provider=>customer. Selection priority: (highest) LOCAL PREF>(shortest) ASPATH>(loweset) MED>eBGP>iBPG>Router ID. Types: eBGP(BGP sessions between broader routers in different ASes, to external dest), iBGP(BGP … in the same AS, distribute externally learned routes internally), IGP(Interior Gateway Protocol=Intra-domain routing protocol).

Link layer: ethernet. (P93-P105, L16-L17)
Services: Framing(encapsulate network layer data), Link access([M]edium [A]ccess [C]ontrol protocal), Reliable delivery(primarily for high error rate like wireless), Error detection and correction. Point-to-point (pairwise communication) ⇔ broadcast (shared wire or medium). Random access MAC protocols: detect and recover from collision (made by two or more nodes trying to transmit at full channel data rate w/o coordination, e.g., ALOHA, CSMA, CSMA/CD, CSMA/CA(wireless).
[C]arrier [S]ense [M]ultile [A]cess: listen before transmit, if channel sensed idle=>transmit entire frame, if busy=>defer transmission, don't eliminate all collisions.
CSMA/[C]ollision[D]etection: add collisions detection, abort transmission when collision detected, easy in wired (broadcast), hard in (wireless) LANs, limitation: restricted length, it takes latency d, resulting in wait 2d to detect collision. For 10 Mbps, max len of wire: 2.5km, min len for frame: 64 bytes. Ethernet Switches: enable concurrent communication, no constrains on link len; use source MAC address to store mapping in switch table, reduce TTL to eventually forget mapping.
[A]ddress[R]esolution[P]rotocal and [D]ynamic[H]ost[C]onfiguration[P]rotocal: link layer discovery protocols, confined to a single [L]ocal-[A]rea [N]etwork, relay on broadcast(make contact, limited size for scalable), caching(reduce overhead), soft state(w/ TTL, forget past eventually, key for roubustness). Remote: checking via netmask (via DHCP), get router IP (via DHCP), send to router MAC (via ARP).

```
/*
 *                   _oo0oo_
 *                  o8888888o
 *                  88" . "88
 *                  (| -_- |)
 *                  0\  =  /0
 *                ___/`---'\___
 *              .' \\|     |// '.
 *             / \\|||  :  |||// \
 *            / _||||| -:- |||||- \
 *           |   | \\\  -  /// |   |
 *           | \_|  ''\---/''  |_/ |
 *           \  .-\__  '-'  ___/-. /
 *         ___'. .'  /--.--\  `. .'___
 *      ."" '<  `.___\_<|>_/___.' >' "".
 *     | | :  `- \`.;`\ _ /`;.`/ - ` : | |
 *     \  \ `_.   \_ __\ /__ _/   .-` /  /
 * =====`-.____`.___ \_____/___.-`___.-'=====
 *                   `=---='
 *           佛祖保佑       CP完整       考试高分
```