Computer Networks

> Author: Guanzhou (Jose) Hu 胡冠洲 @ MIT 6.829
>
> Teachers: Manya Ghobadi & Mohammad Alizadeh

Overview

Computer Networking is quite "*Engineering-ish*". It:

- Measure/Build real things that impact the real world
- Is *interdisciplinary* - Interplay with other fields and based on various theories
- Relatively young and immature
- **Defining the problem** is as least as important as proposing a solution

Internet Architecture & Protocals

A **Network Architecture** defines: Functionality & Placement - what to do & where to do it. A **Protocol** defines: How to communicate between components in the architecture. They combine with actual **engineering and deployment** together to become an operational computer networking system.

History of the Internet Architecture

Brief history of **the Internet**:

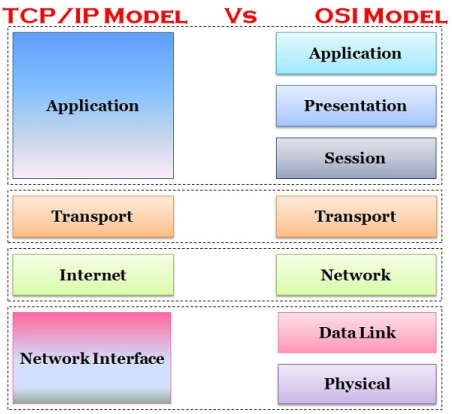
- ARPANET: the first *packet-switching* network implemented, 1969.10.29 the first communication from UCLA to Stanford SRI, widely called "the birthday of Internet"
- Internet: 1970s, linking different packet-switching local networks together, using *Gateways* (nowadays *routers* supporting IP)
- Splitted into a layered TCP/IP model in 1978
  - End-to-end principles in 1980s: DNS, DHCP, ...
  - The problem of *congestion collapse* becomes urgent in 1986, leading to reserches on *congestion control*
- NSFNET: entering an era of world-wide large-scale distributed networks in 1990s
  - Leading to CIDR, BGP, QoS (Quality of Service)...
  - HTTP → World Wide Web & URL
  - Multicasting (v.s. Unicast, Broadcast, Anycast): Explanation
  - ...
- 2000-2010, The Internet matures:
  1. Peer-to-Peer (P2P) networks
  2. Security against DDoS & Spams
  3. User-generated content
  4. Wireless & Mobile networking
  5. Cloud computing & Datacenters

The Internet architecture still has its weak parts waiting for improvements! E.g., security issues, mobility, higher availability...

OSI Model v.s. TCP/IP Protocol

The 2 models:

- **OSI** (Open System Interconnection) Model
  - From ISO (International Organization for Standardization), 1977
  - A purely conceptual model
- **TCP/IP** Protocol, a.k.a., Internet Protocol Suite (IPS), TCP/IP Stack, DoD (Department of Defense) Model
  - From DARPA (Defense Advanced Reserach Projects Agency, previously ARPA), 1978
  - A practical model that actually guides implementation, tightly related to the Internet



Sub protocol components in the TCP/IP protocol suite:

Abstract TCP/IP Layer	Protocol Candidates Related to this Layer
Application layer	BGP, DHCP, DNS, FTP, HTTP, HTTPS, IMAP, POP, SMTP, Telnet, SSH, ...
Transport Layer	TCP, UDP, TLS/SSL, ...
Internet Layer	IP (IPv4, IPv6), ICMP, ...
Network Interface Layer	ARP, NDP, Tunnels, PPP, MAC (Ethernet, Wi-Fi, DSL, ...), ...

Packet-switching

A major contribution of the TCP/IP protocol stack is the **packet-switching** scheme. Packet-switching scheme is different from Circuit-switching. Data streams are splitted into and sent as multiple packets (each having some necessary metadata, like id and target address, in the header). Different end-points can thus *multiplex* a global data link.

TCP Packets

For long-lived **TCP** (Transmission Control Protocol) connection, they ensure high reliability via ACK & retransmissions:

- Allowing maximum 64KB packet length, but practically much smaller (depending on the physical limit of the network interface)
- Packet header includes: source & target port, seq#, ack#, data offset, flags, window size, checksum, ...
  - A seq# marks the position of a byte in a TCP connection
    - Starts with a random initial number (ISN), and creating a connection will consume one number
    - Packet's seq# is the seq# of its first byte
  - A ack# = 4201 means the receiver has got all bytes upto seq# 4200
- Creating a connection: *three-way handshake*
  1. [Client $\rightarrow$ Server] `SYN` with seq# A
  2. [Server $\rightarrow$ Client] `SYN/ACK` with seq# B & ack# $A + 1$
  3. [Client $\rightarrow$ Server] `ACK` with seq# $A + 1$ & ack# $B + 1$
  4. [Client] Start sending; [Server] `ACK` received, start listening
- During a connection, when transmitting data:
  - Every packet received, send back a corresponding `ACK` message
  - Packet gets **retransmission** after certain timeout (RTO) not receiving corresponding ACK
  - Two different sending choices:
    - Send a packet when ready, or send at a certain rate, ✗
    - A **window** of several packets is sent w/o receiving `ACK`, then wait for all `ACK`s to comeback to start sending a new window, ✓
  - *Fast-retransmission*: retransmit immediately on duplicate ACK for a missing packet $\Rightarrow$ *Selective ACK* optimization. READ HERE
- Ending a connection: *four-way handshake*
  - [Initiator (either side) $\rightarrow$ Receiver (the other side)] `FIN`
  - [Receiver $\rightarrow$ Initiator] `ACK`
  - [Receiver $\rightarrow$ Initiator] `FIN`
  - [Initiator $\rightarrow$ Receiver] `ACK`
  - [Initiator] Wait for certain time before closing; [Receiver] `ACK` received, close

UDP Messages

For **UDP** (User Datagram Protocol) messages, they are efficiently sent w/o reliability insurance. Applications must allow some missing / incorrect / repeated messages:

- Message header only includes: source & target port, datagram length, checksum
- Called **Best-effort**

IP Fragments

IP-level (Internet Protocol) also splits into packets (called *fragments*):

- IP fragments do NOT ensure 100% reliably sent. The upper-level transport protocols (e.g., TCP) should handle this
- Fragment header includes: version, header length, total length, flags, TTL (time to live), checksum, source addr, target addr, ...

- A TCP packet should normally be small enough so as not to be fragmented at IP level

> Length of actual data in a TCP/IP packet: `IP.total_length - (IP.header_length + TCP.data_offset) * 4` in bits.

DNS & Routing Architecture

See This blog & This wiki. Also check the domain routing section below.

And a very good exercise for understanding routing & CDNs HERE.

Congestion Control

Packet-base **Congestion Control** (CC) is a major research field on computer networking, aiming to improve the availability of the Internet. We suppose using the TCP protocol by default in this section.

Problem Definition

"Without knowing the capacity of link c and the number of senders n with me, how / whether / where to send to avoid / solve congestions?"

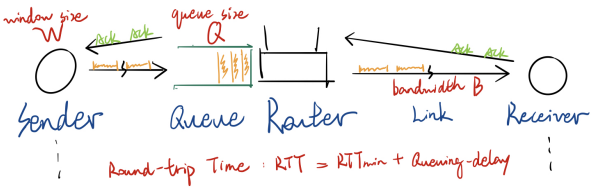
Goals:

1. *Efficient* - high utilization of link bandwidth, i.e., *High Throughput*
2. *Fairness* - connections are served somewhat fairly (user-level unfairness might still occur, where an accelerator simply opens a lot of connections to do one task)
3. *Low Latency*, i.e. *Low Delay*
4. *Fast Convergence* - figuring out a proper configuration as fast as possible

Architecture base:

End users + Routers with **buffers** that can buffer an incoming **queue** (when following links are congested - bandwidth fully occupied and cannot serve more packets at that time).

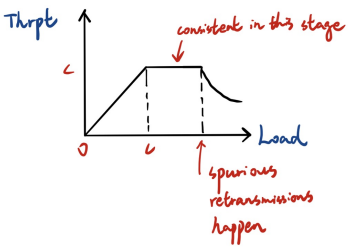
How many of these components are involved & How they are placed and connected with **links** are called the **Topology** of the network.



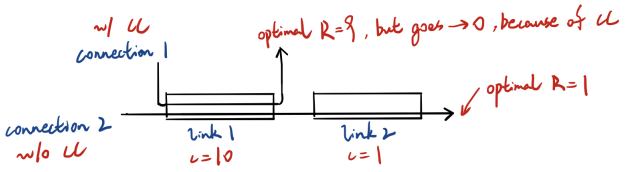
When Collapses Happen

Scene 1: End-to-End on a link with capacity c in packets.

- When load reaches capacity c, *queueing* starts, and the throughput reaches the limit c
- When "spurious retransmission" starts happening (a packet is not dropped but waiting in queue, but triggers retransmission)



Scene 2: Needs global view of the network.



End-to-End Congestion Control

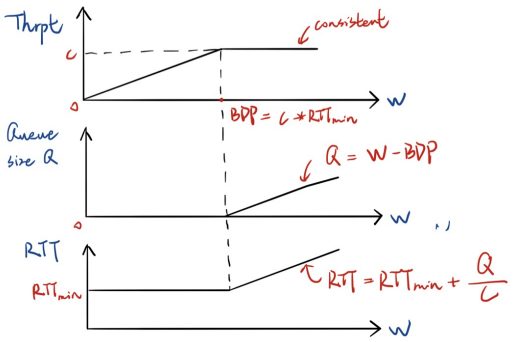
E2E congestion control only happens on end points, w/o involving routers. Includes: Reno, Cubic, BBR, ...

TCP Reno

Traditional TCP CC, linear & loss-based. READ HERE.

Method

- *Mean+Variance* method to dynamically determine timeout: $RTO = RTT_{smooth} + 4 \cdot Var_{RTT_{smooth}}$
- Robust "`ACK` *self locking*" by windowing (with window size w in bytes):
  - Sending in rate $R = \frac{1}{\text{packet\_interval}}$ in packets $\Rightarrow$ Hard to know & dynamically adjust to optimal rate
  - With windowing (packet conservation), magic happens:
    - Throughput $Thrpt = \frac{w}{RTT}$, where RTT is the actual round-trip time $\geq RTT_{min}$ (i.e., RT propagation delay, the unavoidable delay in hardware transmissions)
    - Maximum window w/o triggering queuing $= c \cdot RTT_{min}$, called Bandwidth-Delay Product BDP



- Uses *Additive Increase Multiplicative Decrease* (AIMD) to *probe* & *maintain* the window size (w in bytes, or `cwnd` in packets, congestion window size). This results in a typical **TCP Sawtooth** graph as shown below:
  - After every RTT: $cwnd \leftarrow cwnd + \alpha$, in Reno $\alpha = 1$
  - At packet drop: $cwnd \leftarrow m \cdot cwnd$, in Reno $m = \frac{1}{2}$
  - (During slow start $cwnd \leftarrow m \cdot cwnd$, $m = 2$, after each RTT, so it is a special-case Multiplicative Increase)

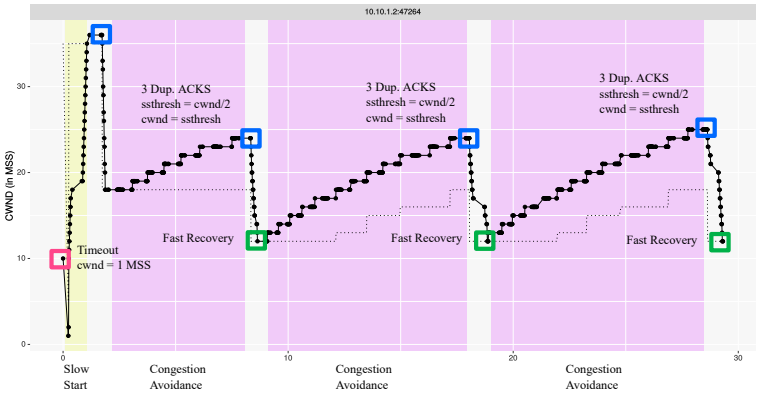
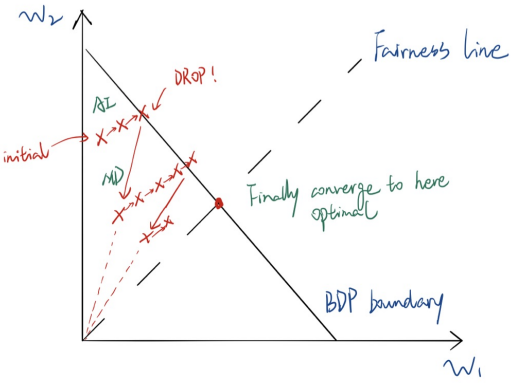


Figure from this blog.

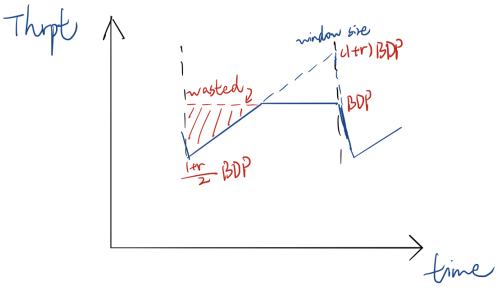
Fairness

AIMD can finally converge to fairness when flows have the same RTT. Check the following phase plot. When different flows have different RTTs, the one with longer RTT will grow slower during AI, resulting in unfairness issues in TCP.



Buffer Size Effects

Let the router's buffer size to be $r \cdot BDP$. Window size will reach a peak at $(1+r)BDP$ and then packet drop happens, making it shrink downto $\frac{1+r}{2}BDP$. However, as queuing packets stay in the buffer, actual throughput cannot exceed BDP:



It means to fully utilize the link bandwidth:

- Buffer size $= 0 \Rightarrow \overline{Thrpt} = \frac{3}{4}c$, not enough
- Buffer size $\geq BDP \Rightarrow \frac{1+r}{2} \geq 1 \Rightarrow \overline{Thrpt} = 100\%\ c$, fully utilized

But big buffers also lead to much longer latency and unwanted retransmissions.

#####Throughput Equation

When assuming no packet loss (under utilized): $Thrpt = \frac{w}{RTT}$.

On a fully utlized dynamic path, we have packet drops. Denote packet loss rate as p, then one loss every $\frac{1}{p}$ packets. Since the windows enlarge from $\frac{1}{2}w_m \to w_m$ by step size 1 and then triggers that loss, summing up the series gives $\frac{3}{8}w_m^2 = \frac{1}{p} \Rightarrow Thrpt = \sqrt{\frac{8}{3}} \cdot \frac{MSS}{RTT \cdot \sqrt{p}}$, where MSS (*Maximum Segment Size*) is the size of data in a packet in bytes.

Google BBR

New from Google, bottleneck-based. READ HERE.

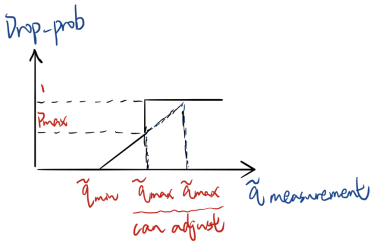
Network-Assisted Congestion Control

Network-assisted congestion control means involving the routers into the control procedure. Includes: AQMs such as RED, PIE, ..., ECN, and XCP. With the assistance of routers, congestion control performance are normally better. However, it requires change of implementation in routers, making it hard to deploy and incompatible with normal Internet infrastructure.

AQM: RED, PIE

Active Queue Managements (AQM) apply on switch buffers. It specifies a drop rate r to randomly drop enqueuing packets, in order to mitigate problems like *bufferbloat*.

RED (*Random Early Detection*):



- Very sensitive to parameters, usually turned off in today's routers

- With number of TCP flows increases, queue length also increases

PIE (*Proportional Integral controller Enhanced*): READ HERE. Sets a target queue size measure $\tilde{q}_{ref}$ (actually using delay in ms). Then, at each update interval, update drop rate as $p \leftarrow p + \alpha(\tilde{q} - \tilde{q}_{ref}) + \beta(\tilde{q} - \tilde{q}_{old})$, to both control the step size of adjustment based on *how far we are away* from the target queue size, and the *history trend* of the queue size. (This is the method used in *PI controllers*.)

ECN & XCP

ECN (*Explicit Congestion Notification*): router sets the ECN bit in IP header to 1 if it sees congestion.

```
|-----IP-Header--[ECN Data]--|-----TCP-Header-----|----------Payload----------|
```

XCP (*eXplicit Control Protocol*): advanced version of ECN, READ HERE.

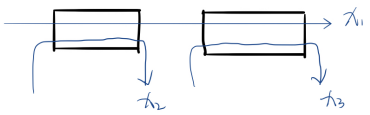
- When multi hop $\rightarrow$ routers take the minimum value
- Has reaction delay $\rightarrow$ may have problem in heterogeneous / dynamic networks
- Complex header $\rightarrow$ requires change in current router implementations, so very hard to actually deploy

Network Utility Maximization (NUM)

What various TCP congestion control protocols are actually solving is: an *optimization* problem.

Given a flow x, the *utility function* is a *increasing* + *concave* curve that vaguely models a flow's utilization of the network. E.g., $U(x) = \sqrt{x}, \log x, \ldots$. In any topology, **NUM** abstracts congestion control as maximizing the sum of utility functions $\sum_i U_i(x_i)$ of all flows, when subject to the constraint $A \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ c_l \end{bmatrix}$, where c_j is the capacity of link j, and A is a matrix of size $l \times n$ that gives which flows go through which links.

Example:



- Maximize $\log x_1 + \log x_2 + \log x_3$ when subject to $x_1 + x_2 \leq 1$ and $x_1 + x_3 \leq 1$
- Solution: $U'(x_1) = \frac{1}{x_1} + \frac{1}{1-x_1} + \frac{1}{1-x_1} = 0 \Rightarrow x_1 = \frac{1}{3}, x_2 = x_3 = \frac{2}{3}$
- Different choice of function U means different tradeoff between *throughput* & *fairness*

A distributed NUM algorithm:

- Each link l gives a congestion "price" p_l
- On links perspective: $p_l(t+1) = p_l(t) + k(r_l(t) - c_l)$, where r is current rate and c is capacity
- On flows perspective: e.g.., pick $U_i(x_i) = \log x_i$, we wanna maximize $\log x_i - q_i x_i$, where $q_i =$ sum of all link prices that x_i goes through
  - This gives optimal $x_i = \text{some constant} \cdot \frac{1}{q_i}$
  - TCP Reno's choice is $U(x_i) = -\frac{1}{x} \Rightarrow x_i \propto \frac{1}{RTT_i \sqrt{q_i}}$
- Will finally run into convergence

Underlay Networks

The building blocks of nowaday's computer networks.

Datacenter Networks

A **Datacenter Network**'s goal is to provide low latency, high bandwidth, less oscillation and agility in placing application tasks.

Layer 2 v.s. Layer 3

The *layer-2* network refers to the **Ethernet routing** layer. It:

- Is based on MAC address instead of IP address, so can have nodes with conflict IPs $\Rightarrow$ Plug & Play, convenient
- Has a broadcast scale limit because of ARP
- Typically using *spanning-tree protocol* $\Rightarrow$ No *multipathing*

The layer-3 network refers to the **IP routing** layer. It:

- Needs IP address configuration ⇒ Not flexible

In a traditional datacenter network architecture:

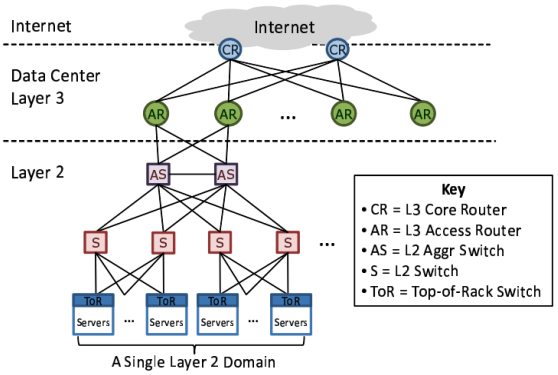


Figure from the VL2 Paper.

There exists serious **oversubscription** problem: message from one pod to another must go through upper layers, so upper layer switches handle a huge amount of messages.

VL2 Design

VL2 provides the illustion of a huge layer-2 switch based on the *Clos* topology. Architecture:

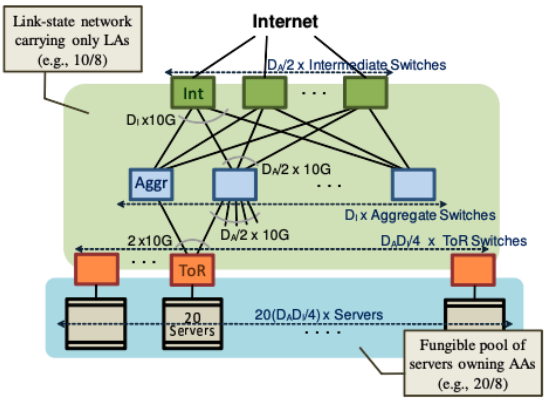
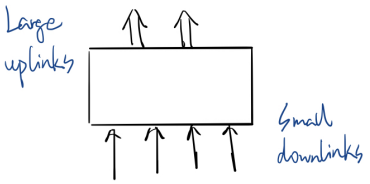


Figure from the VL2 Paper.

1. Maintains layer-2 semantics for communications
2. Multipathing ⇒ Maximizes utilization & Provides performance isolation
    - ECMP Valiant Load balancing (VLB), with double-encapsulation + anycast
    - Fatter uplinks improve throughput, see below
3. May have trouble with hash collision and when there are "elepant" flows, but they are often transient, not long-lived
4. *Needs TCP CC to operate*, because aggregated upper layer bandwidth is high, but bandwidth from a TOR to a node is limited. Without TCP CC, packets are sent in multiple paths through the upper layer but only to be dropped at the target node's TOR

Speed Differential on Switches

Speed differential between downlink and uplink on a switch can actually improve the performance:



- With small downlinks → same number of uplinks, one hash collide then cannot get 100% utilization rate
- With small downlinks → several large uplinks, then hash collide can happen and we will have a significantly larger chance to maintain 100% utilization rate

Software-Defined Networks (SDN)

A **Software-Defined Network (SDN)** is a network architecture where *control* is physically separated from the *forwarding devices*, and it controls a set of devices from a logically centralized view. It is one step towards a higher-level of *abstraction* for networking research. Becoming a hot topic with the first release of **OpenFlow** protocol and **Network OS (NetOS)** at 2008.

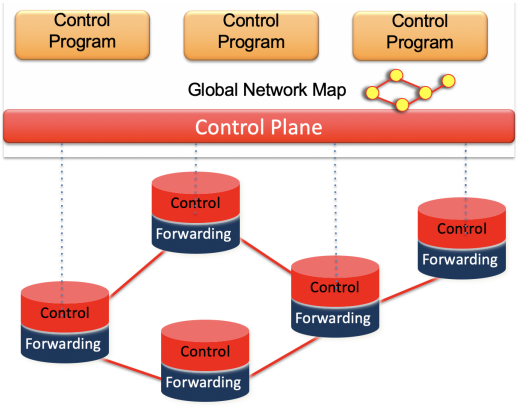


Figure from Prof. Ghobadi's slides.

- **Data plane**: physical forwarding devices in the network, functioning at packet scale (nano secs)
- **Control plane**: logically centralized control layer, changes at event scale (seconds)
- **Management plane**: things like *Access Control* (ACL), changes at human scale (minutes)

Wide-Area Networks (WAN) & Interdomain Routing

Let's scale our view beyond a local network, to explore the problems in a higher-level of networking across the world.

Wide-Area Networks (WAN)

A **Wide-Area Network** (WAN) is a network **backbone** connecting different datacenters across a large geographic area. The interconnection is mostly enabled by physical fiber cables under the oceans.

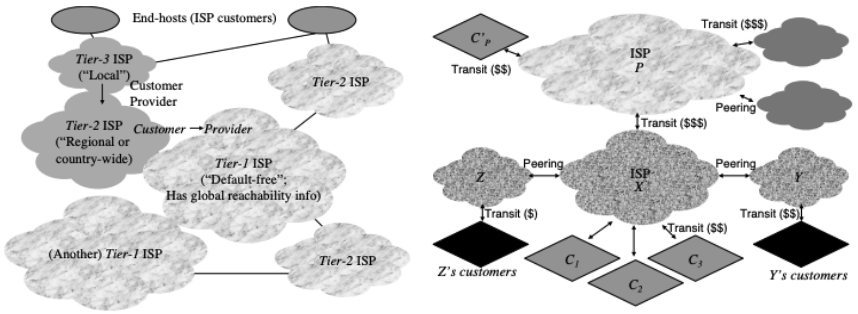
WAN is different from the general "Internet" in that it is *private*, often belongs to one company / organization for connect its own datacenters.

Problems that might occur in WAN:

- Efficiency: achieve high average utilization
- Sharing
- Fault tolerance and Security issues
- Cost

Interdomain Routing & BGP

Regarding the **Internet** architecture nowadays, it consists of end users and three level of ISPs (*Internet Service Providers / Access Network*, AN, Tier-1 (Global), Tier-2 (Regional), Tier-3 (Local)). Each ISP has its own "Intranet", which we call as an **Anonymous System** (AS). Different ASs connect together, routing messages between each other:



Figures from Prof. Balakrishnan's book.

- **Provider-Customer Transit**: a customer ISP pays the provider ISP for its routing capability
- **Peering** neighbors: two ISPs collaborate and expose their routing to each other, typically not paid; peers do not transit for other peers, they only transit for their respective customers

IP address contains the topology information: the *subnet prefix* tells you which intranet you belong to (e.g., `18.*` for MIT). From a single ISP's point of view, if an IP destination of a message is not one of my local end users, then a **Route** means a mapping $f: \text{subnet prefix} \mapsto \text{neighbor (or link) to send this message}$. That neighbor does not have to be the final destination (in this case he should know where to send further), but it should be ensured that all messages finally get to their destinations. Such mappings are recorded in the *Routing Table*.

How does an ISP know where to send a message? The **BGP Protocol** (*Boarder Gateway Protocol*) defines a way of doing *Route Advertisement*, where eash ISP broadcast "*I can route messages with prefix* `A`, `B`, *and* `C` *to their destinations*" information to neighbors. This is called the *import / export* policy. After knowing which links can I send a message to, we should then have a *ranking* scheme to choose which is the best link to send it.

Export policy (Announcement rules):

- To customers: announce all routes learned from peers, providers, and customers, also self-origining routes
- To peers / providers: announce routes learned from customers and self-origining routes

Detailed **path selection policy** in use:

Priority	Rule	Remarks
1	LOCAL PREF	Highest LOCAL PREF (§3.2.3). E.g., Prefer transit customer routes over peer and provider routes.
2	ASPATH	Shortest ASPATH length (§3.3.5) *Not* shortest number of Internet hops or delay.
3	MED	Lowest MED preferred (§??). May be ignored, esp. if no financial incentive involved.
4	eBGP > iBGP	Did AS learn route via eBGP (preferred) or iBGP?
5	IGP path	Lowest IGP path cost to next hop (egress router). If all else equal so far, pick shortest internal path.
6	Router ID	Smallest router ID (IP address). A random (but unchanging) choice; some implementations use a different tie-break such as the oldest route.

Figure from Prof. Balakrishnan's book.

- `LOCAL_PREF`: **customer** > **peering neighbor** > **provider**
- `ASPATH`: number of ASs a message will go through
- `MED` (*Multi-Exit Discriminator*): used when two ISP neighbors have different inter-connection links, connecting their different edge routers

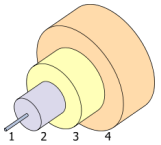
Today, BGP-4 has the following problems:

- Simple concept $\Rightarrow$ Complicated & abused implementation
- Security issues: subject to *hijacking*, e.g., Youtube diverted to Pakistan, Feb 24, 2008
- Videos put tremendous pressure on the interaction between Content Providers (CPs) and ANs

Optical Networks

Optical Fibers

Optical fibers (光纤) are a new form of new physical links that transmits information through optical light signals.



1. Core, 2. Cladding, 3. Buffer, 4. Jacket. From Wikipedia.

It works as follows:

- Glass-like materials that can transmit light signals through *Total Internal Reflection*, thus
  - Is cheaper than sending electronic signals over copper **Cables** in long-range transmissions
  - Light signal decays much slower than electronic signal over cables, so needs fewer *Amplifiers*
  - But theoretically cannot transmit faster than electrons
- **Transceivers** (*Transponders*) can translate between electronic signals and optical signals (E $\leftrightarrow$ O), which happens at
  - End hosts
  - Electronic swithes: OEO translation is fast (ns scale) compared to long-range transmission time, but cannot be ignored in fast datacenter networks
- One optical link can bear multiple signals: READ HERE & READ HERE
  - *Single-mode* (SMF): Waves are distributed in space in the same way (giving a single "ray" of light in the fiber)

        - Each trace can still bear mulitple signals of different wavelengths by *Wavelength Division Multiplexing* (WDM, coarse $\rightarrow$ dense); needs MUX & DeMUX
        - Stable over long-distance transmissions
    - *Multi-mode* (MMF): Waves can go through the same fiber in different traces at the same time
        - Larger data rate but limits the maximum length of fiber due to dispersion

It has the following disadvantages:

- Subject to bending & damage $\Rightarrow$ Hard-to-detect data corruption $\Rightarrow$ Important to use tools like checksums
- Theoretical RTT = Physical distance / Speed of light in fibers, but actually much larger than that because fibers are not laid in straight lines & there are overheads (often use $\frac{2}{3}c$ as an estimate, or ~1ms delay per 100km in practice)

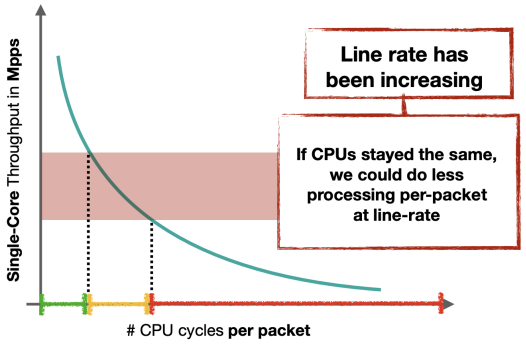
Optical Switches

Besides traditional electronic packet switching switches, optical links can use optical **Circuit Switches**: No OEO translation, directly guides the light through one port to another port.

- Reconfigurability is an issue - can achieve by using mirrors
- Essentially becomes a *matching* problem
- How to buffer light? Can we utilize indirect paths?

SmartNICs with FPGAs

Original (dumb) Network Interface Cards (NICs) are responsible for L1 + part of L2 logics, and all other computations are handled by CPU. With netlink rate dramatically increasing these days, CPUs can not keep up with this pace (Moore's law is coming to slow down):



Figures from Mina's slides.

We need to find a way to **offload** part of higher layer logics downto NICs.

- *ASICs* are too rigid, not programmable enough
- *Multi-core SoCs* are not performing so well
- *FPGAs*, with programmable hardware configurations, turn out to be a feasible solution

Overlay Networks

The upper-level applications and software designs that the networks people concern about.

Video Streaming

Video Streaming Workflow

Videos' 4D quality metric:

1. No interruption (no rebuffering)
2. High quality (high bit rate)
3. Quick start (small playback delay)
4. Stable quality (small variation)

Workflow: *Encoding* to different files of different bitrates $\Rightarrow$ *Deploy* to CDNs $\Rightarrow$ *Control* plane $\Rightarrow$ *Quality adaption*.

Video streaming is more sensitive to throughput than to delay (unless under special live streaming cases).

Adaptive Bitrate Algorithms (ABR)

The problem with video streaming is a tradeoff between:

- Avoid unneccessary rebuffering
- Avoid video quality to be too low

ABR algorithms dynamically choose which bitrate to fetch for the next frames, based on you network condition. Different kinds of ABR:

1. **Rate-based**: depends on an estimate to you net link's rate; not working well under highly varying throughput
2. **Buffer-based**: depends purely on your buffer usage ($bitrate = f(B)$)
3. Using advanced techniques like *Model Predictive Control* (MPC)

Remain problems that ABRs cannot handle very well:

- Large scale of *short* videos scenario
- Other metrics other than bitrate maximization (e.g., QoE)

Networking with RL

Not really understanding it well, so details ~~omitted~~ here.

Figure from Dr. Sen's slides.

Reward calculation:

- The deployed policy: $\frac{1}{n}\sum_{i=1}^{n} r_i$, where r is reward
- The estimated policy: $\frac{1}{n}\sum_{i=1}^{n} \frac{r_i \cdot I_i}{p_i}$, where p is the deployed policy's probability of choosing that action, and I is match indicator (0 when not match & 1 when the estimated policy will also do that action)

Networking with Blockchain

Blockchain is an attempt to establish *Distributed Trust*. Nakamoto's Bitcoin paper presents three essential components to achieve a workable blockchain system:

1. One-way hash that theoretically cannot be inverted + Chain structure where a block contains information of all the previous blocks by taking the previous block's hash value as part of the input of its own hash
2. *Proof-of-Work*, a way to pick a winner over multiple miners (in Bitcoin, a hash threshold that can be tuned with mining rate f)
3. Longest chain protocol + k-deep confirmation to maintain global consensus

Read Nakamoto's Initial Paper HERE.

The great thing about this design is that, the key thing matters to prevent **Private Double-spending** problems is the hash power (computation power) itself, not number of nodes / requests, so it is hard for attackers (non-honest users) to take over.

Downsides of traditional Bitcoin structure:

- Throughput ($\mathrm{Thrpt} \propto f$) is set to 1 block / 10 mins $\Rightarrow$ 5 transactions / sec, very low
- Confirmation latency ($\mathrm{Latency} \propto \frac{1}{f}$) can be hours for a transaction, to maintain a sufficient depth of k (e.g. 6)

Why not simply increase the mining rate f? Because then there will be much more **forking** and this will make private double-spending attacks much easier. To improve its performance while maintaining reliability, there are some possible directions:

- Longest chain protocol $\rightarrow$ Other advanced protocols based on DAGs
- ...

Cluster Resource Managing

Task scheduling (not VM scheduling here for this section) has been one of the important topics in cloud datacenters research. Legacy schedulers:

- Pure distributed schedulers (not global view): *Sparrow*, *Apollo*, ...
  - $\uparrow$ scale, utilization
  - $\downarrow$ no multi-frameworks, i.e., performs poorly when task sizes vary a lot
- To support multi-frameworks besides MapReduce, *Hadoop YARN* has taken the RM (Global Resource Manager) + AM (Per-job Application Manager) + NM (Per-node Node Manager) structure
  - $\uparrow$ multi-frameworks, good fairness
  - $\downarrow$ scale, utilization (because of heartbeats)

- Naive queuing can be used on NMs
  - ↑ utilization
  - ↓ task latency grows because of gang-scheduling / early-binding / ...

Hydra (YARN++) adopts sub-clusters design to scale YARN upto 50K nodes scale.

P2P & Distributed Hash Table (DHT)

Essential task for P2P content locating is a lookup function: `lookup(item) = host`, where host is the machine that actually stores the item.

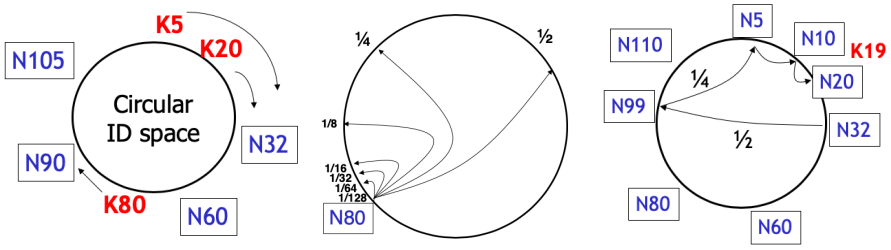
- *Napster*: uses a centralized index server, efficient but limits scalability and is prone to faults
- *Gnutella*: flooding requests to neighbors with TTL, will cause serious congestion when scaling up
- **Distributed hash tables** (DHT): *Chord*, *Pastry*, ..., totally decentralized and no flooding
  - Load balance - keys evenly distributed
  - Efficient for lookups both in time and bandwidth
  - Low management overhead - machine come & go in low overhead

Degree-Diameter tradeoff: we want small degree (for low management overhead) & small diameter (for efficient lookups) at the same time.

- Given average degree d, up to distance h, can reach $1 + d + d^2 + \cdots + d^h \sim d^h$ nodes
- So for n nodes in total, $d^h > n \Rightarrow h > \log_d n$

Consistent Hashing

The *Chord* algorithm is built upon **consistent hashing**: Both items and nodes are mapped onto the same geometric space. In this case, a *circular ring*. Then `h(item)` is the `host` who's key directly successes its key. It brings good load balance and minimal key movement when machines join & leave.



Left: Key assignment through consistent hashing; Middle: Finger table; Right: Lookup procedure.

Figure from Prof. Alizadeh's Slides.

Chord Operations

Lookups take $\frac{1}{2}\log N$ hops in average. Each node only keeps a $O(\log n)$ small routing table (called *finger table*), then do a "binary" search on the ring to find the node directly successing the item. It decouples *correctness* from *performance*:

1. Correct successor pointers ensure lookup correctness;
2. Accurate finger tables ensure good performance. Avg. lookup = $\frac{1}{2}\log n$ hops.

Node joining is achieved by a linked list insertion operation + *stabilization*.

1. Lookup joining node n_1's key k_{n_1}, find successor n_2
2. n_1 copies objects on n_2 whose key is smaller than k_{n_1} to itself
3. n_1 sets its successor pointer to n_2
4. In a *periodic stabilization* step of n_1, it notifies n_2 to set its predecessor pointer to n_1
5. In a *periodic stabilization* step of n_2's previous predecessor n_0, it checks its successor (n_2)'s current predecessor, and finds out that it is something (n_1) between itself (n_0) and its successor (n_2), so:
   - n_0 sets its successor to n_1
   - notifies n_1 to set its predecessor to n_0

To achieve higher availability & fault tolerance, we can store a object with mulitple copies on, say, 3 nodes successing it. And a `lookup` operation returns a list of nodes saying you can find the object on these nodes, pick whichever you want.

Other Topics References

Some topics are not covered in detail in this note.

- **Content Distribution Networks** (*CDN*s): Refer to HERE & HERE.
- **OS with Networking**: CPU core allocation, refer to HERE.
- **Programmable Routers**: Refer to HERE & HERE
- **Network Verification**: Header space analysis approach, refer to HERE.