19/01/28 INTRODUCTION

- What is cloud? Computing as a utility.
- Key advantage of cloud is economy of scale, being able to provide a lot of compute at relatively low cost compared to having your own data center.

Networks review

- Network Address Translators (NATs) - IPv4 addresses are limited, and NATs allow you to have a public IP address and a subnet of private IP addresses. The NAT itself translates addresses from the global to local IP addresses. If two users are behind NATs and want to communicate, then you need to use a rendezvous (e.g. Skype)
- 4 layer model - Application, Transport, Network, Link, allows for modular design
- Network layer is special because it uses IP, which makes a best effort attempt to delivers datagrams yet makes no promises. IP is very simple so that it is fast and lower cost to build and maintain. You can build what you need on it, e.g. you can run TCP on top of IP for reliability. IP runs on any link layer, and makes very few assumptions.
- End to end principle - where possible, implement features in the end hosts

19/01/30 BACKGROUND, SDN

Networks review, cont.

- Use layering to achieve modularity (separation of concerns), reuse, and continuous improvement without changing other components
- Challenges of networks:
    - Performance (latency, throughput)
    - Security
    - Scalability
    - Resilience (packet loss)
- Network layer: OSPF or IS-IS for shortest paths, BGP for ISPs and global routing
- Transport layer: TCP has 3 way handshake, UDP simply sends the data. TCP offers:
    - Reliable delivery (acknowledgements to indicate correct delivery, checksums to detect corrupted data, and sequence # to detect missing data)
    - Flow control via feedback to adjust sliding window
    - Congestion control

19/01/30 SDN

Openflow

- OSPF and IS-IS are distributed routing protocols that perform shortest paths
- SDN separates control and data plane, and allows you to run apps on top of a network OS (the controller)

- Openflow switch checks match action table, if it knows what to do, then use that entry. Tend to match on header fields, act by forwarding, dropping, or modifying header. If not a match, send it to the controller.
- Advantages: simplified programming because of centralized control, open standard API, encouraging research innovations
- Disadvantages: security, scalability, performance since the latency depends on communication to controller, which is slow, we want closer to line rate
- Challenges: resilience of logically centralized controller, imperfect knowledge of network state, consistency issues, and scalability
- SDN was applicable to hard problems: inter-datacenter traffic engineering, cloud virtualization

Firewall bug

- If internal protected by a firewall, no bug, since we install the user sends traffic through the firewall, and updates the app and firewall itself. The traffic reaches the external, and is able to be sent back through the firewall
- If there are multiple firewalls managed by the firewall app, there might be a race condition, if the internal sends to its firewall, and then sends traffic to external. The external to internal firewall might not be updated in time, so the external is blocked.

19/02/06 DC ARCH, CONGESTION CONTROL

Architecture

- Ideally, we would only have one big switch connected to all the racks, and we try to achieve this abstraction
- Racks contain a top of rack switch and a bunch of servers
- Key features:
    - Agility - use any server for any service at any time, rapid installation of a service's code via VMs, access data from anywhere via distributed file systems, and quick communication via topology
- Scalable commodity architecture
    - Nonblocking bandwidth - line rate
    - Commodity switches
    - CLOS/fattree topology
    - Disadvantages:
        - Routing mechanism, suboptimal throughput
        - Fault tolerance because upward failures are easier to fix than downwards failures, since you only have one path down. The solution to fault tolerance is to introduce a bit of asymmetry (AB clos network), but then you have to hack it a bit to make it work.

    - Rigid structure - need to have a certain number of servers, switches, etc. otherwise it won't work, so you can't incrementally deploy more servers. Overutilizing switches is not good and leaving ports for later is wasteful

Jellyfish

- Idea is to replace links randomly. This increases the amount of servers that can be at full throughput and minimizes average path length so that more servers are reachable in fewer hops.
- Advantages: incremental expansion, higher throughput
- Disadvantages: cabling, congestion control, what routing would you even use?

TCP control

- Review: 3 way handshake, sliding window
- On connection established, send full window, then, if a full window is not received, wait a bit, then resend
- Key properties: efficiency (throughput) and fairness

19/02/11 DC ARCH, CONGESTION CONTROL

Max-min fairness

- Max-min fairness means that you cannot increase the rate of one flow without decreasing the rate of another flow with lower or identical rate.
- To calculate max-min fairness, if a user requests less than their fair share, distribute their leftovers to everyone else
- Chiu Jain plot plots fairness and efficiency. There is a line where are are fair and efficient (neither overloading nor underloading)
- Additive increase, multiplicative decrease will eventually converge to the fairness and efficiency point. If the network can handle more, additively increase your window. If it's congested, half your window.
- Add an exponential slow start because otherwise it might take a long time to get started

Why is TCP broken in DC?

- Tends to fill queues - fix w/ shallow buffer switches, ECN
- Unfair to short flows - fix w/ larger initial rate or window size
- Slow to converge - fix w/ faster timeouts and ECN
- Loss is not equal to congestion - fix w/ ECN
- Equal rates is not necessarily fair or best - fix w/ traffic classification and prioritization
- Does not handle asymmetry well - assumes that everyone is running TCP, and works well only in symmetric topologies
- Congestion collapse in data center

ECN

- Once a buffer hits a certain fullness, send feedback to the sender.

BGP

- A money routing protocol first, does not necessarily do shortest paths
- Contains all the problems of intradomain routing, and more! Bigger scale, multiple parties without centralized control, conflicting interests, greater volume and diversity of attacks, harder to change architecture
- A router takes in updates from its neighbors, applies import policies, then selects a route to save. Then, applies export policies to send updates to neighbors
- Provider charge customers, peers are mutual and do not pay each other
- Gao-Rexford policies:
    - Prefer sending traffic over customer > peer > provider
    - Export all routes to customers
    - Export customer routes to everyone and nothing else

19/02/13 DATACENTERS

SDN, cont. Firepath

- One of the problems with SDN is communication with controller.
- Firepath alleviates this problem by sending topology to switches, so that not only the controller can perform shortest paths, switches can also perform shortest paths and adapt to changes in the topology.

MPLS

- Simple protocol with high performance forwarding, and VERY flexible. Good for VPNs, control backup paths, and traffic engineering/load balancing.
- Each packet has a header, and a router can match on a tag. It can then swap a label and forward it on a port.
- This does not really belong to any layer.
- But not good for inter-datacenter wide area networks
    - Lack of coordination between background and non-background traffic. However, this can be solved by simply running background traffic only when there is less non-background traffic.
    - Local, greedy resource allocation. If one flow goes in one path first, it could block other paths from using their shortest paths
    - Poor sharing. Not always max-min fair.
- If limited switch memory, just install a working set of paths, nad use scratch capacity to enable disruption-free updates to the set

MS SWAN

- Software driven wide area network, which tries to have more flexible sharing policies by coordinating across services and centralizing resource allocation
- Controller collects information about the topology, and allocates bandwidth to service hosts and service brokers. This provides feedback about the traffic demand, which the controller responds to by updating the network configuration
- One use case is for congestion-free network updates.

Congestion-free network updates

- We want to update flows to be globally optimal.
- If you have scratch capacity, you can split a flow into multiple parts, and move it over to a different link in parts.
- If you do not have scratch capacity, then this is impossible.
- So always leave some scratch capacity
- We would also like to preserve against a larger set of invariants, e.g. bad actors, but this is hard

19/02/18

DCTCP

- ECN, but rather than halving, use ECN marks to determine how congested. The more ECN marks, the more you want to cut the window, but if only one or two packets are congested, then it's okay, just decrease a little. This keeps queue length a lot smaller.
- We want programmable switches that are fast, so we can't really use software routers.

dRMT

- RMT aggregates resource into stages that provide a fixed ratio of memory, match, action resources
- dRMT is disaggregated, so allocate resources independently. Each stage uses as much memory, compute as it needs. Improves on RMT by coupling memory allocation with match and action allocation e.g. splitting a large table over multiple stages, allows throughput to degrade gracefully as program size grows.
- Tbh we didn't really talk about this too much

19/02/23 Changes and Verification

- Problem: transient intermediates of updates might be wrong e.g. bad actor example

Consistent updates

- Per packet consistency - each packet uses old or new version of rules, but not both

- Install new rules, keep the old rules. Packet can either use old rules, which are correct, or new rules. Eventually, old rules can be removed.

Consistent routing

- Per flow consistency - apply updates after all routers updated
- Responsiveness - liveness property, a system reacts quickly to failures or policy changes
- Consistency - safety property, a router forwards packet along a path adopted by upstream routers
- Routers compute and forward BGP route announcements, but do not apply them to forwarding table. Take a distributed snapshot, and mark updates in transit or being processed as incomplete. Router send list of incomplete updates to consolidators, which perform a consensus algorithm to agree on the set of incomplete updates. Consolidators flood the incomplete set to all routers, and the routers apply completed updates.
- Flooding and applying updates cannot be done atomically. The solution is that if a packet reaches a router that has not applied the updates, use old rules.

Verification via header space analysis

- Networks are super hard to debug and to even think about it. It's almost magic.
- HSA performs static checking. It basically views each router/switch as a function and headers as the input, and by looking at the space it takes up in the output, you can answer question such as can A talk to B, what are all packets from A that can reach B? We can also find loops, prove that two groups are isolated, good stuff.

Veriflow

- Generate equivalence classes - packets experiencing the same forwarding actions throughout the network, basically looks at a header and divides it into different regions based on the actions that can be performed.
- Generates forwarding rules
- Runs queries
- Tells us which packets are affected by an invariant.

Control plane versus data plane verification

- Control plane analyzes all possible network states while data plane looks at current network state (unable to capture change over time)
- HSA and veriflow are data plane verification.

19/02/27 VERIFICATION, VIRTUALIZATION

Control plane verification

- Batfish parses configuration and metadata dynamic state and intent and network models, and analyzes for correctness based on constraints. This is hard to scale. Also, this does not address the crux of the issue, which is that we want to make things easier to configure.
- Propane analyzes policy in the propane language, compiles it, then creates BGP configurations. Uses lots of good theory (regex, reverse automata, product graphs)

VL2
NVP

19/03/06 Big Data Systems

Map-Reduce

- Split data > map by extracting something you care about from each record > write to file system > reduce by aggregating, summarizing, filter, sorting, etc. > write to output
- Fault tolerance by rerunning
- Master orchestrates everything
- Stragglers - one machine is slow, have to wait for it… solution is just to schedule multiple copies of last tasks
- With more tasks, finer grained load balancing, spread failed worker load over many machines, and overlap map and shuffle, shuffle and reduce. However, with fewer bigger intermediate files with less overhead
- Maps probably scale, reduce probably scale, but network may limit scaling (low bisection bandwidth, incast)
- Not good for graphs, since each iteration needs to be stored to disk

Spark

- Rather than storing to disk through network, save it to memory
- RDDs - immutable, coarse-grained deterministic transformations, efficient fault recovery using lineage (if lost, rebuild it)
- K-V stores are good for transactional workloads, RDDs are good for batch workloads

19/03/11 Scheduling

- FIFO - first in first out, average completion time can be high, since unfair to short flows
- STF - shortest task first, but it is optimal. However, hard to know expected running time of the task, good for batch operations
- Round robin - good for interactive interactions

Hadoop

- Schedule map and reduce tasks, multi-tenant datacenters
- FIFO, Hadoop capacity scheduler, and Hadoop fair scheduler
- HCS does it by hierarchy of queues
- HFS divides clusters into pools, and focuses more on fairness, so preemption is allowed

Goals of a good scheduler

- Isolation - can't sabotage others
- Efficient resource usage - if as user is at full capacity, the resource is not idle
- Flexibility - priority functionality
- Max-min fairness - share guarantee in that each user gets at least 1/n resource, and it is strategy proof so no user can gain the system

Dominant Resource Fairness

- Different tasks might have different resource needs, e.g. CPU or memory
- Apply max-min fairness to dominant share
- May have some wasted resources

Competitive Equilibrium from Equal Incomes

- Give users 1/n of every resource, let users trade in a competitive environment
- Limits wasted resources, but some users get screwed over

Mesos

- No single framework is optimal for all applications
- Before, static partitioning, meaning that each framework is on a single cluster
- We want dynamic scheduling, distributed among different clusters
- Goals: high utilization of resources, support diverse frameworks, scalability, reliability
- Using resource offers, fine grained sharing