CS 268
Project Ideas

Sylvia Ratnasamy

Spring 2021

Presented in class (2/11)

1. Make TCP robust to high degrees of packet reordering
   - *starting point: SACK (selective ACKs)*
2. A "ban elephants" transport protocol
   - *Starting point: multi-path TCP*
3. Getting RTT estimates from *all* packets in a TCP connection
4. A client-side "BlameIt" tool
5. A "BlameIt" tool to detect short-term congestion
6. How consistent is cloud network performance?
7. A datacenter topology that's robust to misconfiguration?
8. An Internet architecture that prioritizes accountability
9. An Internet architecture that prioritizes diagnostics/measurement

Legend

Well-defined project

Less well defined

You need to define

A “return to sender” primitive

- High availability in the network is a long-standing problem
  - Hard because repairing a route fundamentally incurs some convergence time

- Insight: hosts increasingly have multiple access links
  - E.g., wifi and cellular; SD-WAN services; multi-homed enterprises

- Proposal
  - Instead of dropping a packet, the router upstream from the failed link/router returns the packet to the sender
    - high probability the path to the sender works!
  - Sender tries its alternate/backup link
  - Design and evaluate the usefulness of this primitive

Using anycast to build DDoS-resilient services

- IP anycast offers network-level DDoS protection
  - Advertise the same address from multiple locations
  - Network naturally routes a sender's packets to the closest location
  - Almost impossible for attackers to force concentration of traffic
- Common wisdom: doesn't work for TCP-based services
  - Because route changes might take you to a different server mid-connection
- Proposal: replicated services using TCP-over-anycast
  - E.g., based on migrating connection state across anycast-addressed servers

Applying ML/RL to Congestion Control

- Challenge in CC stems from the large variety of environments we need to optimize for
  - differ by orders of magnitude in almost any relevant metric
- Project: leverage recent advances in ML and RL to develop a solution that quickly adapts to different network environments
  - Show limitations of the existing ML/RL algorithms for CC
  - Or prove a general solution that matches any state-of-the-art CC
- Starting point: Remy@ SIGCOMM'13; Alizadeh's work @ MIT

From Ion Stoica

“Active scheduling” for fine-grained traffic engineering in the datacenter

- Many network phenomena occur on short timescales
  - e.g., queuing delays, incast problems, non-deterministic failures
  - hard to diagnose, debug, or react to (control path is too slow, switch statistics are averages)

- Proposal: packets carry forwarding instructions conditioned on switch statistics
  - e.g.: “change packet’s priority to X if queue length exceeds Y”; “generate control message if table > X”
  - Make it safe: “instructions” can be inserted by operator
  - Make it backward compatible…: implemented at hypervisor
  - Make it practical: compatible w/ emerging programmable switches (P4)

Deflection Switching in Datacenter Networks

- Common wisdom: statistical multiplexing requires buffering in switches
  - When two packets arrive for the same output port, transmit one and buffer the other
- But buffers in switches are a pain
  - Introduce latency, expensive, power hogs
- Idea: when two packets contend for an output port P, send one to P, the other to a random unused output port
  - Deflected packet may find another path to the destination or may loop back for another try
- Result:
  - equal/improved performance at lower cost?
  - A new lower bound on buffer requirements
  - A new result on the feasibility of buffer-free switches

How predictable is a job's traffic matrix?

- Study a number of typical datacenter workloads
  - E.g., Amplab's big data benchmark; ML benchmarks
- Run them over a range of configurations
  - Varying data set sizes, framework configurations, resource allocations (memory, CPU, etc.)
- Answer: is a job's network behavior predictable?
  - E.g., throughput, latency, burstiness, loss, flow completion times, etc.
- Results (positive or negative) has implications for job scheduling, traffic engineering, etc.

Containing Datacenter Traffic

- Full bisection bandwidth fat-trees makes sense if
  - there's ongoing high-volume all-to-all traffic, or,
  - there's occasional high-volume any-to-any traffic and we can't predict when/where it occurs
- Measurement studies suggest a small number of large jobs generate most traffic
- An alternate architecture
  - Interconnect (say) 10% of racks by a high-capacity switch
  - Interconnect the remaining 90% racks with a cheaper, lower-capacity interconnect (tree?)
- Schedule large jobs at the well-connected 10%
- Result: equal/better performance at lower cost?

Buffer Sizing vs. BBR (both later readings)

- Revisit the Appenzeller results on buffer sizing with two key differences:
  - Assume that endpoints run BBR
  - Measure impact on applications (i.e., measure FCT)
- Repeat their analysis / simulation / implementation
  - any one is plenty!
- Extend your study to other CC protocols (e.g., Cubic)

Putting BBR through its paces

- BBR paper's evaluation is thin
- Project: compare BBR to (say) XCP, Cubic, RCP
  - Over a range of topologies, flow sizes, bottlenecks, link latencies, etc.
  - Using a range of metrics, especially fairness and flow completion time
  - Use simulation and/or implementation
  - Bonus: evaluate robustness to misconfigured/malicious hosts

Revisiting Datacenter Topologies

- Cabling complexity is often an afterthought in the literature on datacenter topologies [Clos, Jellyfish]
- Explore design options that are cabling friendly
- E.g., hybrid designs
  - Mix of high and low speed links
  - Mix of random and structured topologies
- Prove fundamental tradeoffs between cabling complexity, throughput and cost
  - Through theory or simulation

Policy Compliant Link-State Routing

- Path vector routing (BGP)
  - Supports policy (only policy-compliant paths advertised)
  - Slow and complex convergence
- Link state routing
  - Much simpler and faster convergence
  - What about policy?
- Traditional link state advertisements (LSA) don't convey policy
- Project: Design a policy-compliant link state routing solution
  - E.g., extend LSAs to express a domain's policy for the traffic it is willing to route on that link
- Challenge: quantify whether this leaks more policy information than traditional BGP

The End-to-End Principle, Part 2

- The E2E paper informs the placement of function between the ends *vs.* the network

- But modern network involve many more parties
  - Between management controllers *vs.* network routers (SDN)
  - Between the "fast path" *vs.* "slow path" (hardware *vs.* software)
  - Between datacenters vs. wide-area networks (APLOMB@SIGCOMM12)

- Articulate the equivalent to the E2E argument for the above

The Future of Stat Muxing

- Statistical multiplexing has been a foundational principle of the Internet's design
  - Allows many bursty sources to efficiently share network resources
- Looking ahead: 50B+ IoT devices by 2024 [Gartner]
  - Many (most?) IoT devices are periodic, not bursty, sources
- Questions: implications of IoT for stat-muxing?
- Possible results:
  - A model for IoT traffic sources
  - Quantify the efficiency of stat-muxing with increasing periodic sources

Resources

- Internet topology maps:
    - https://www.caida.org/projects/ark/topo_datasets.xml
    - http://www.routeviews.org/routeviews/
    - http://projectsweb.cs.washington.edu/research/projects/networking/www/rocketfuel/

- Network simulators
    - https://www.nsnam.org (the most commonly used simulator in the research community
    - https://github.com/NetSys/simulator (a simpler simulator for quick prototyping but far fewer features)

- Traffic distribution models:
    - Most simulators include traffic sources (Zipfian, uniform, etc)
    - Traffic in datacenters:
        - https://cseweb.ucsd.edu/~snoeren/papers/fb-sigcomm15.pdf
        - DCTCP paper

- Software switches:
    - OVS: https://www.openvswitch.org
    - Click: https://github.com/kohler/click
    - BESS: https://github.com/NetSys/bess