Network Management Beyond SDN

Jeff Mogul
Google Platforms/Network Infrastructure
mogul@google.com

SDN solves network management, right?

Improving Network Management with Software Defined Networking

Hyojoon Kim and Nick Feamster, Georgia Institute of Technology

Network Management and Software-Defined Networking (SDN)

EE122 Fall 2013

Scott Shenker

(understudy to Sylvia Ratnasamy)

Well, somewhat

It depends on:

- how you define “SDN”
- how you define “Solves Network Management”

Defining “SDN”

For the purposes of this talk, SDN means:

- Cleanly separate “data plane” from “control plane”
  - Data plane (HW or SW) handles most of the packets
  - Control plane (SW, typically in a different place) tells the data plane what to do
  - Some sort of protocol (e.g., OpenFlow, but not always) to link the two
  - Relies on a straightforward data-plane abstraction: parse, match, apply actions, count
- Enable complex control-plane software systems (“network operating systems”)

This is oversimplified (other people define SDN more broadly), but it will do.

Defining “Solves Network Management”

Scott Shenker’s definition (from his slides):

- “Everything having to do with the control plane”
- “What is the control-plane problem? compute forwarding state”

“Everything” here includes (among other things):

- Data-plane functions on packets: routing, access control, isolation, modification
- Autonomous control-plane functions: routing protocols, traffic engineering

So, what doesn’t SDN solve?

That’s what I’m going to tell you

A broader view of “network management”

Some things that we have to do:

- Decide what networks we want (or how to improve them)
- Design our networks/improvements under a set of goals + constraints
- Plan how to deploy these networks
- Keep track of physical and virtual assets, and what we need to order
- Send humans out to wire these up
- Test what got wired up, and fix it if necessary (it’s always necessary)
- Install and upgrade SDN controller software
- Install and upgrade switch-stack software
- Control forwarding tables
- Configure the network elements (switches and controllers)

… and also

- Decide what to enable and disable (we say “drain” and “undrain”)
- Decide when to drain and undrain things (without losing too much capacity)
- Drain and undrain things
- Expand (or contract) a network while it is running
- Monitor the network for faults and failures
- Monitor stuff like “running out of memory” or “getting too hot”
- Root-cause and fix faults and failures
- Analyze the history of the network to see if we need to make changes

Now, the same lists, but using green to show what SDN can do

(or orange for “maybe”)

A broader view of “network management”

Some things that we have to do:

- Decide what networks we want (or how to improve them)
- Design our networks/improvements under a set of goals + constraints
- Plan how to deploy these networks
- Keep track of physical and virtual assets, and what we need to order
- Send humans out to wire these up
- Test what got wired up, and fix it if necessary (it’s always necessary)
- Install and upgrade SDN controller software
- Install and upgrade switch-stack software
- Control forwarding tables
- Configure the network elements (switches and controllers)

… and also

- Decide what to enable and disable (we say “drain” and “undrain”)
- Decide when to drain and undrain things (without losing too much capacity)
- Drain and undrain things
- Expand (or contract) a network while it is running
- Monitor the network for faults and failures
- Monitor stuff like “running out of memory” or “getting too hot”
- Root-cause and fix faults and failures
- Analyze the history of the network to see if we need to make changes

What are we trying to achieve?

Goals for network management

Top-level goals for a happy network:

- Provides enough bandwidth, between endpoints in its scope, to meet demand
- Cheap hardware, software, and operations
- Highly available: limited planned downtime, infrequent failures, fast repairs
- Support rapid evolution: expand, contract, add new HW, add new features/fixes

Goals for network management

Top-level goals for a happy network that are always in conflict with each other:

- Provides enough bandwidth, between endpoints in its scope, to meet demand
- Cheap hardware, software, and operations
- Highly available: limited planned downtime, infrequent failures, fast repairs
- Support rapid evolution: expand, contract, add new HW, add new features/fixes

Goals for network management

Top-level goals for a happy network that are always in conflict with each other:

- Provides enough bandwidth, between endpoints in its scope, to meet demand
- Cheap hardware, software, and operations
- Highly available: limited planned downtime, infrequent failures, fast repairs
- Support rapid evolution: expand, contract, add new HW, add new features/fixes

This leads to some next-level goals:

- Automate as much as possible (but support manual ops for emergencies)
- Avoid correlated failures
- Design for good modularity and conceptually-simple interfaces

Network management data

Some definitions

- **Topology**: graph of network elements and how packets flow through them
  - May involve multiple layers of abstraction (e.g., one IP link contains a number of fibers)
  - Also includes SDN controllers, physical containment, locations, addresses, etc.
- **Network policy**: rules for how we want to use (not use) parts of the topology
  - Access controls, routing policy, priorities for different classes of service
- **Configuration**: specific per-element settings that implement topology+policy
  - Includes settings for SDN controllers
- **Management policy**: rules for operating the management plane
  - Who can make what changes, and when.
- **Telemetry**: what is actually going on in the network
  - What's up, what's down, what's happening to packets, what's overheating, etc.

Representing network-management data

We try to limit the number of different data representations:

- This avoids an N^2 problem when N systems each define their own format
- We can share a lot of infrastructure code
- Data modeling is hard, and it helps to share the expertise

Formats that we use include (among others):

- “Unified Network Model” (UNM) for topology -- Google-specific
- OpenConfig (based on YANG) for configuration -- vendor-neutral, by design
  - OpenConfig also supports telemetry
- Domain-specific high-level declarative languages for policy
- Databases for “allocatable resources” such as IP addresses

Modeling ain’t easy

Just for topology modeling, we have:

- O(10K lines of schema definition)
- 260+ distinct “entity kinds” (switches, ports, links, etc.)
- Growing or changing, more or less every day
- Weekly meetings of a review board, to provide guidance & consistency
- A lot of software infrastructure (model storage, validation, viewing, etc.)
- Lots of technical debt:
  - Things we need to model, but have not yet figured out
  - Things we wish we had modeled differently, but schema-change is hard (really hard)

If declarative modeling is hard, why do it?

Because embedding knowledge in imperative code is much worse

Infrastructure

A network-management infrastructure

A complete infrastructure includes (incomplete list!):

- Capacity planning tools
- Network design tools
- Testing systems
- Configuration-generation pipeline
- Operational services (e.g., to “drain” a switch or upgrade its software)
- Monitoring and alerting services
- Systems to detect invalid topologies, policies, and operations
- Storage systems for topology, policy, generated config, monitoring data, secret keys, IP addresses

Topology planning has lots of moving parts

Topology planning includes:

- Measurements and forecasts of demand (workload)
- Policies (e.g., tradeoff between risk tolerance and the cost of excess capacity)
- Risk analysis (will we meet capacity goals given predicted failure rates?)
- Info about available resources (e.g., what submarine cables can we use?)
- Algorithms to compute new or expanded network topologies
  - This is a complex problem; see B. Schlinker, R. Niranjan Mysore, S. Smith, J. C. Mogul, A. Vahdat, M. Yu, E. Katz-Bassett, and Michael Rubin. "**Condor: Better Topologies Through Declarative Design.**" In *SIGCOMM '15*
- Systems to allocate IP addresses, choose the number of SDN controllers, compute cable lengths and how they should be routed, etc.

What is config?

What is configuration (or “config”)?

- What we have to tell a device, in order for it to do the right things
- Ditto for controller software
- Historically, this included what SDN calls “flow-table configuration”
  - OpenFlow only really covers flow-table configuration, plus a tiny set of switch configs
- OFCONFIG defines SDN switch-config … but it didn’t really cover enough stuff

Examples of configurable things ***besides*** **flow-table config**:

- IP addresses of my OpenFlow controllers; Secret keys for authenticating controllers; Allocation of buffer space among QoS classes; Port speeds and modes; Light levels for optical ports; Interface MTUs; Fan-speed controls; VLAN IDs; sFlow parameters; ECMP profile; Link-aggregation settings; NTP peers; ….

Config-related challenges

We need a way to specify ***declaratively***, and at different levels of abstraction:

- High-level (human-edited) “intent”
- Low-level config consumable by devices and controllers

We need to ***generate*** low-level config from high-level intent (kind of like compilation)

- and validate the results

We need ways to ***reliably*** “push” config to devices and controllers

- with sequencing across multiple targets, to maintain consistency

Other pieces of the network management infrastructure

Some “infrastructure for the infrastructure”

- Storage services for topology, config, telemetry data
- Log services to record actions and alerts, and tools to analyze logs

And

- Access-control mechanisms and audit trails
- Intrusion-detection systems
- Auditing systems, to ensure that reality doesn’t drift too far from intent
- Inventory-management systems, to keep track of stuff that has part numbers

(A lot of this is a legal requirement, for compliance with laws like “SOX” and “HIPPA”)

Don’t forget testing!

Network operators are pushing for increasingly high availability

- “5 nines” (99.999% available) allows 5.26 minutes downtime/year
- 6 nines: 31.5 seconds of downtime/year

Getting high availability:

- make “Mean Time To Recovery” (MTTR) really low
- make “Mean Time Between Failures” (MTBF) really high

The problem is unanticipated (vs. probabilistic) global failures:

- it’s hard to automatically recover from unanticipated failures …
- … so they need to be extremely rare
- … so your testing infrastructure has to do a lot of work to find them first

Summary

- SDN is great, but “classic SDN” only addresses part of “network management”
- Historically, the rest has been haphazard, but we’re gradually getting better
  - We’re searching for principles as elegant as those behind SDN, but it’s a complex world
- There are a lot of moving parts in a network-management system
  - and I’ve left out a lot of them
- I hope Scott still invites me to his next birthday party

Q&A

Short links for jobs

Internships: wait until next year (when the deadlines will probably be similar)

- ~~MS/PhD SW Eng students: g.co/swegradintern - deadline was Jan 31~~
- ~~Undergrad SW Eng: g.co/sweintern - deadline was Nov 30~~
- ~~Hardware Eng: http://g.co/HardwareEngIntern - deadline was Jan 25~~
- ~~Freshmen/soph "Engineering Practicum": g.co/engpracticum - deadline was 11/30~~

Full-time jobs:

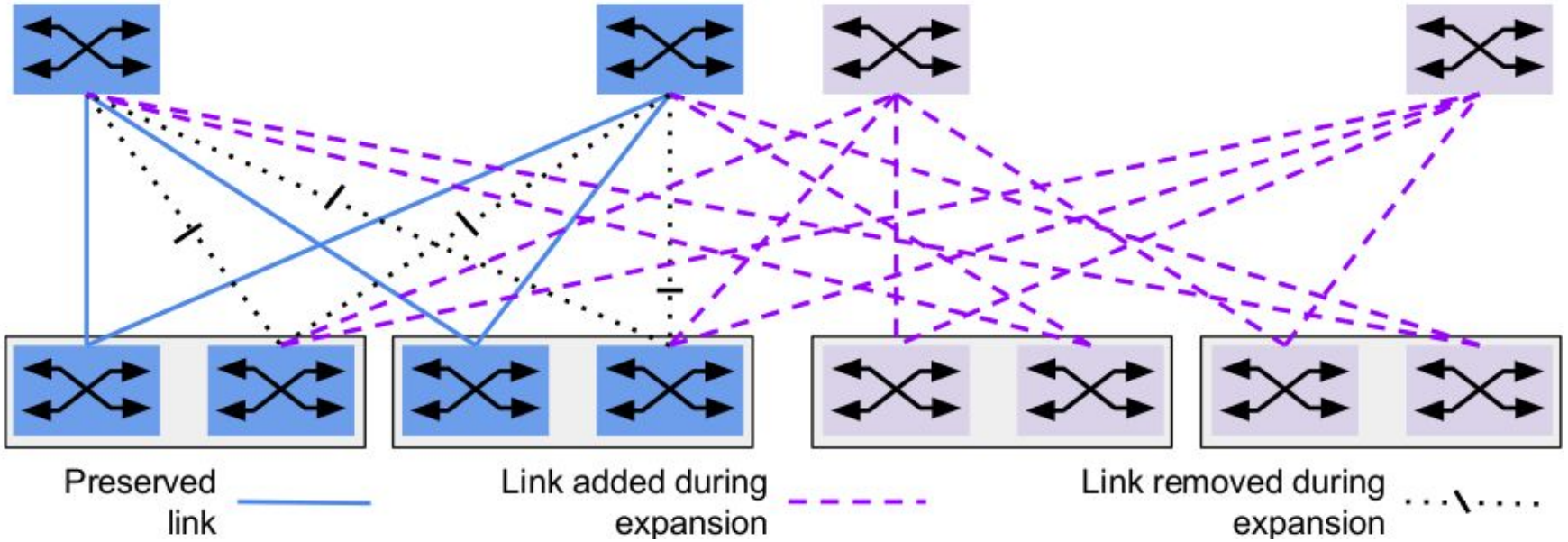
- PhDs - g.co/phdgrad
- Bachelors and Masters - g.co/swegrad
- research.google.com/teams/netsys/ for our team specifically

If you apply: please let me know (mogul@google.com) so I can keep track of you

Questions? StanfordStudents@google.com

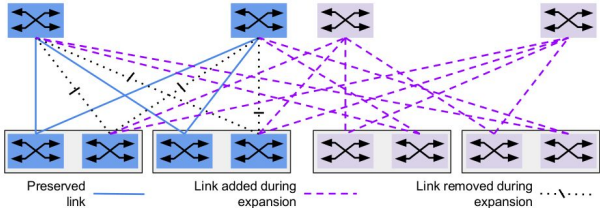
Backup slides

Fabric expansion example: doubling the size of a fat-tree



From: “Condor: Better Topologies Through Declarative Design”

In depth: executing a fabric expansion



Given an existing network and a plan for an expanded version:

- Split the expansion into “capacity-preserving phases”
  - We expand networks while they are carrying live traffic
  - Expansions typically require moving cables, so we have to “drain” some capacity
  - We need to ensure we don’t drain too much
- For each phase,
  - 1: Drain the stuff that we’re fiddling with (or that we might accidentally bump into)
  - 2: Ask some humans to move cables around, and tell us when they are done
  - 3: Test the cables to make sure nothing is mis-wired or dirty
  - 4: Undrain the new cables, and the ones we drained in step 1
  - 5: Check to make sure nothing is broken, and traffic is flowing as expected

Everything except step 2 can be automated; automating step 5 is hard!