Don't Mind the Gap: Bridging Network-wide Objectives and Device-level Configurations

Ryan Beckett[1], Todd Millstein[2], Jitendra Padhye[3], Ratul Mahajan[3], David Walker[1]
[1]Princeton University, [2]University of California Los Angeles, [3]Microsoft Research

Motivation

It is well known that traditional network configuration is highly error-prone [1,2,3]. Surveys of network operators often point to human error as a leading cause of misconfigurations [4,5].

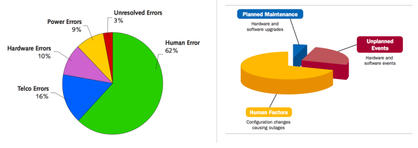


Figure 1. Operator survey on network misconfiguration causes [4,5]

High-level Idea: Traditional networks rely on distributed mechanisms to compute forwarding paths. This means they have many nice properties (e.g., scalability, latency), but configuration is hard because it involves distributed programming (a configuration for each device). SDN simplifies configuration but this comes at the cost of many such properties [6,7].

Distributed programming: a config for each device
Centralized programming: a config for the network

Distributed mechanism: devices talk to each other
Centralized mechanism: a controller decides

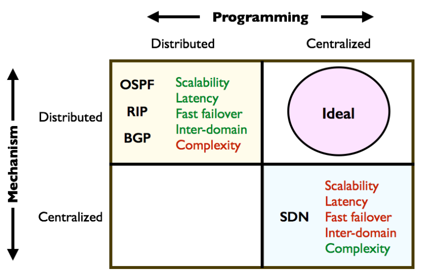
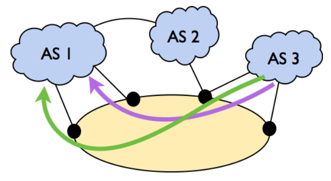


Figure 2. Tradeoffs between tradition networks and SDN

Propane Language

Programming Model:

- A fully centralized view of the network
- Including other autonomous system
- High-level constraints and preferences on paths
- Can describe both intra- and inter-domain routes



Guarantees:

- Complies to pure ***distributed BGP*** configurations
- Policy-compliant under ***all possible failures***
- That is, preserves the centralized abstraction

Example: Configuring a datacenter with BGP

- BGP running on every router
- Keep local services internal to the datacenter
- Aggregate global prefixes externally
- Prefer leaving through Peer1 over Peer2
- Don't let the datacenter become a transit point

```
define Ownership =
 {PG1  => end(A)
  PG2  => end(B)
  PL1  => end(E)
  PL2  => end(F)
  true => exit(Peer1 >> Peer2)}

define Locality =
 {PL1 | PL2 => internal}

define NoTransit =
 {true =>!transit(Peer1,Peer2)}

define Main =
   Ownership & Locality &
   NoTransit & agg(PG, in -> out)
```

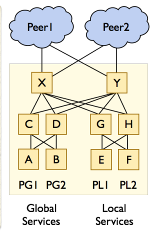


Figure 3. Configuring a datacenter with Propane

Compilation

Compilation Stages:

1. Compile Propane to per-destination state machines that associate paths with ranks. A lower rank means the path is preferred.
2. Combine this with the topology to get a graph representing all policy-compliant paths
3. After checking the graph for failure safety, we generate per-device BGP configs from this graph.

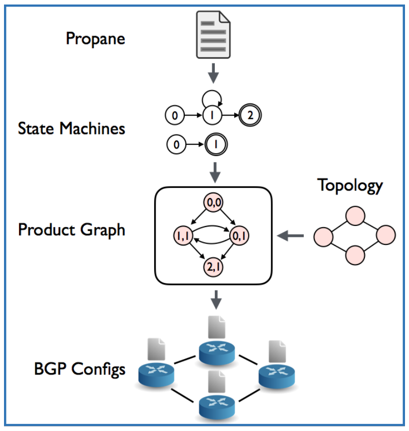


Figure 4. Compilation pipeline for the Propane language

Product Graph:

- Intermediate representation for policies
- Combines topology and policy
- Graph that captures all policy-compliant paths
- Associates ranks with paths to a destination
- Used for safety analysis / failure analysis of

Example: Creating a product graph

Add an edge in the Product Graph if
- The edge exists in the topology
- The state machines transition on the edge target

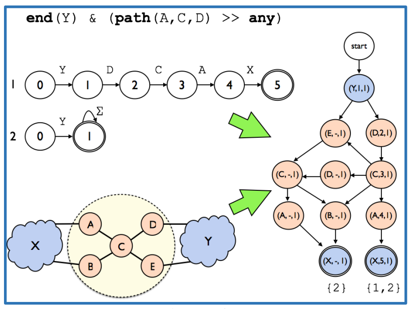


Figure 5. Product Graph construction

Translation to BGP:

- Tag Product Graph state with BGP communities
- Incoming edges → import filters
- Outgoing edges → export filters
- BGP local preferences inferred from analysis

Ensures that BGP will always dynamically find **some best path** to each destination.

Implementation

Compiler:

- Compiler is 6700 lines of F# code
- Flags to configure using: no-export, MEDS, Prepending, anycast, etc.
- Compilation parallelized across prefixes

Benchmarks:

- Policy translated to Propane for datacenter and transit networks for a large cloud provider
- Topology scaled to determine the effect of topology size on compilation time

Policy Size:
Lines of Propane: **31** for the datacenter, **43** for the transit network – orders of magnitude reduction.

Compilation Time:

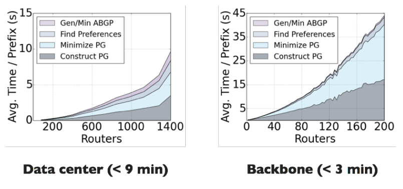


Figure 6. Compilation time for datacenter and backbone networks

Contact

Ryan Beckett
Princeton University
rbeckett@princeton.edu
(303) 956-6712

References

1. M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable commodity datacenter network architecture. In SIGCOMM, 2008.
2. N. Feamster and H. Balakrishnan. Detecting BGP configuration faults with static analysis. In NSDI, 2005
3. R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfiguration. In SIGCOMM, 2002.
4. Juniper. What's behind network downtime? https://www-935.ibm.com/services/au/gts/pdf/200249.pdf.
5. Yankee Group. As the value of enterprise networks escalates, so does the need for configuration management. https://www.cs.princeton.edu/courses/archive/fall10/cos561/papers/Yankee04.pdf.
6. J. McCauley, A. Panda, M. Casado, T. Koponen, and S. Shenker. Extending SDN to large-scale networks. In Open Networking Summit, 2013
7. P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar. ONOS: towards an open, distributed SDN OS. In HotSDN, 2014.