TOWARDS CORRECT-BY-CONSTRUCTION SDN

Leonid Ryzhyk    Nikolaj Bjorner    Marco Canini    Jean-Baptiste Jeannin    Nina Narodytska
Cole Schlesinger    Douglas Terry    George Varghese

22 August 2016
INTRODUCTION

- Cocoon is a
  - high-level SDN programming language
  - verification tool
  - SDN programming methodology
- ... based on the principle of correctness by construction
COCOON IN A NUTSHELL

Traditional SDN Verification Workflow

Design → Implement → Run on the controller → Verify generated network configuration

SDN verification with Cocoon

spec → refine → refine → refine → implement → Correct by construction
EXAMPLE: CAMPUS NETWORK

[Sung et al. Towards Systematic Design of Enterprise Networks]
INTER-VLAN ROUTING: HOP1
INTER-VLAN ROUTING: HOP 2

ACL

vlan-1  vlan-1 gw router
vlan-2  vlan-2 gw router
vlan-3  vlan-3 gw router
switch  router
This is messy:
- Large, ad hoc topology
- L2/L3 routing are mixed up
- Complex distributed security policies

Typical bugs:
- ACL distribution
- Routing loops
- Black holes

Let’s try to untangle this design …

vlant-1  vlant-1 gw router
vlant-2  vlant-2 gw router
vlant-3  vlant-3 gw router
□ switch  ○ router
STEP 1: HIGH-LEVEL SPECIFICATION

```
acl()

193.62.*.*
193.63.*.*
193.64.1.*
```
STEP 2: DISTRIBUTED ACLs

acl()

193.62.*.*
193.63.*.*
193.64.1.*
STEP 2: DISTRIBUTED ACLs

Assumption: acl() ≡ aclOut() ∧ aclIn()
STEP 3: L3 ROUTING

193.62.*.*  193.63.*.*  193.64.1.*
STEP 3: L3 ROUTING

Assumption: L3NextHop(pkt)* = Gateway(pkt)
STEP 4: L2 SWITCHING
STEP 4: L2 SWITCHING
STEP 4: L2 SWITCHING

Assumption: \( L2\text{NextHop}(pkt)^* = L3\text{NextHop}(pkt) \)
STEP 4: L2 SWITCHING

L2NextHop(pkt.vlan=0)

L2NextHop(pkt.vlan=green)

L2NextHop(pkt.vlan=red)

L2NextHop(pkt.vlan=blue)
SEPARATION OF CONCERNS

- High-level spec
- Distributed access control
- L3 routing
- L2 switching
WOULDN’T IT BE GREAT TO BUILD SDNs THIS WAY?

With Cocoon, you can!

- Language support for refinement-based programming
- Automatic compositional verification
- P4 and OpenFlow (via NetKAT) backends
Parameterized specifications

- Spec may contain *undefined functions*, e.g., acl(), l2NextHop(), l3NextHop()
- Verification relies on *assumptions*
- These functions are defined when the network design is instantiated
- They can also change at runtime, e.g., in response to link failures
- Assumptions are validated when concrete definitions are provided (statically or at runtime)
CASE STUDIES

- **Campus network**
  [Sung et al. Towards Systematic Design of Enterprise Networks]

- **F10**

- **B4-style WAN**
  [Jain et al. B4: Experience with a Globally-Deployed Software Defined WAN]

- **iSDX** (Software-defined Internet Exchange)
  [Gupta et al. An Industrial-Scale Software Defined Internet Exchange Point]

- **NSX-style network virtualization framework**
  [Koponen et al. Network Virtualization in Multi-tenant Datacenters]

- **Stag** (source-based routing + security labels for fat-tree topology)
PERFORMANCE

- All case studies verified in ~10 sec,
  - Compositional verification (1 refinement at a time)
  - Parameterized verification amplifies the power of symbolic reasoning
- No direct comparison (yet) with existing tools like NetKAT, HSA, but expect them to slow down for larger topologies
COCOON VS TRADITIONAL NETWORK VERIFICATION
CONCLUSIONS

- Correct-by-construction SDN via iterative refinement:
  - Enforces modular design
  - Enables strong static correctness guarantees
  - Scales to complex realistic networks
assume (hid_t sid, vid_t vid, MAC dst)
  (l2distance(sid, vid, dst) > 1)  =>
  l2distance(link(sid, l2NextHop(sid, vid, dst)), dst)
  == (l2distance(sid, vid, dst) - 1)