Network verification.

Building block

Narrow spec, subtle.

Impls are complicated

Protocols

Distributed

⋮

SDN

Controller, central

Reduces complexity

Verif. target.

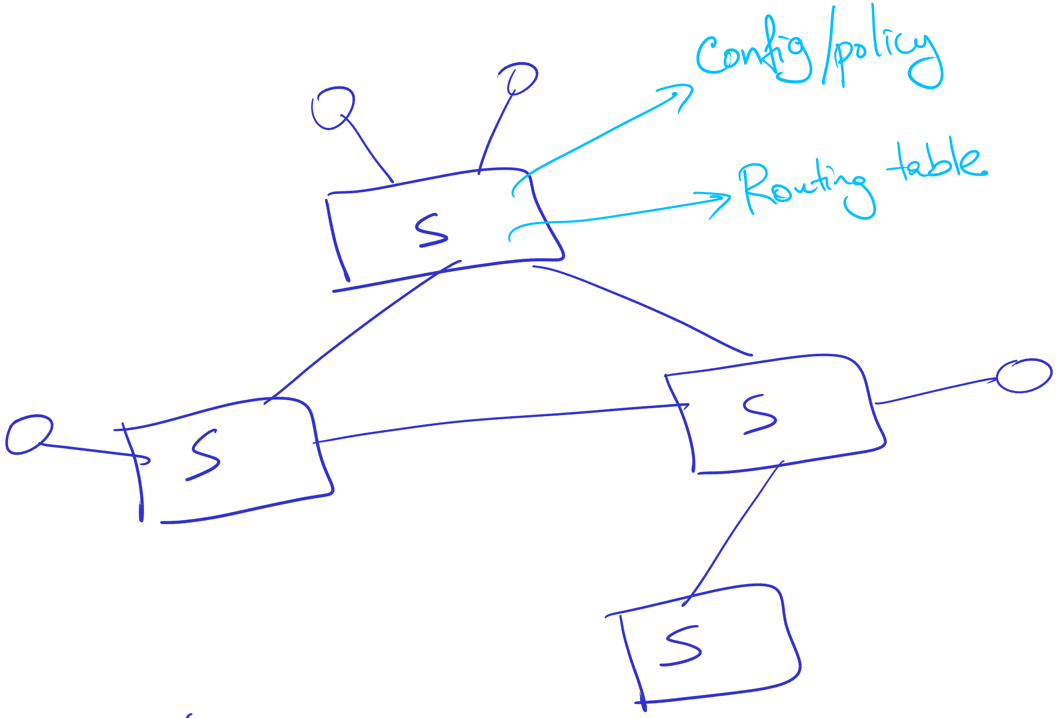
This paper:

Coq framework for SDN verification

Model for SDN.

Simple controller.

Network.



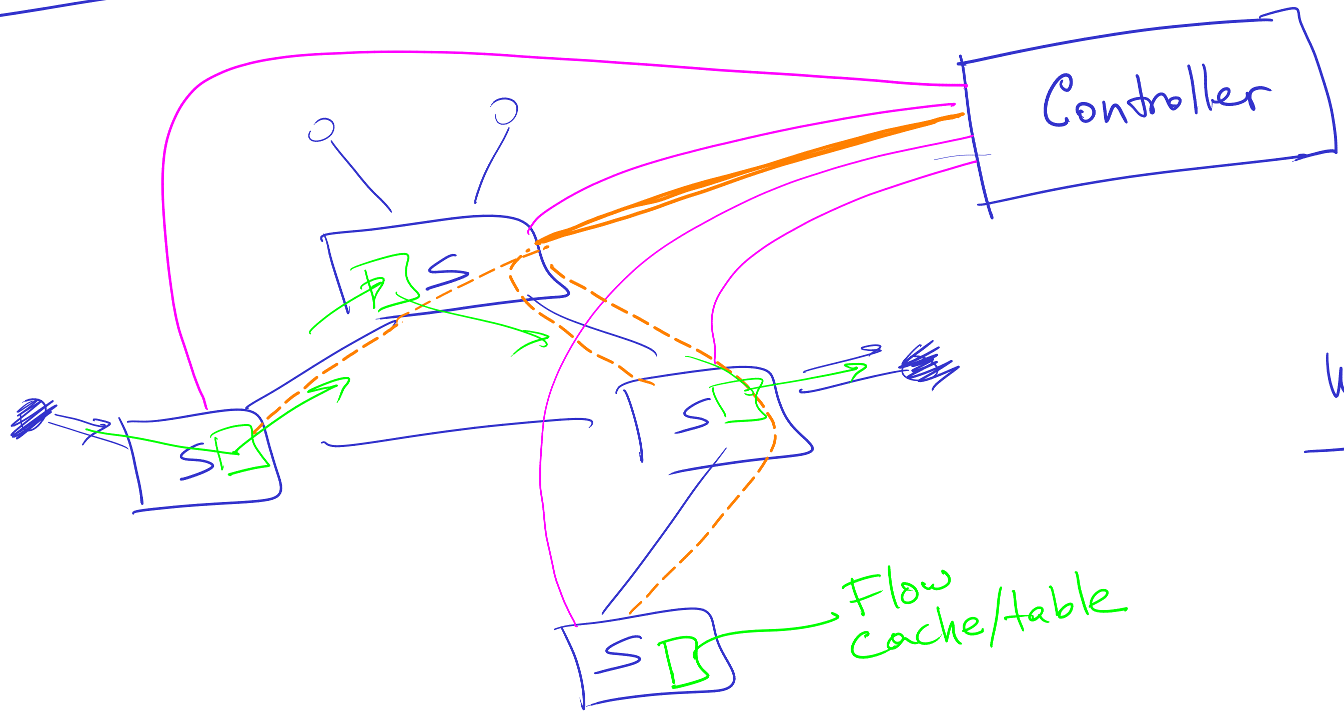
Traditional impl.

Per-switch config.
Protocol between switches.
Agreement on protocol,
not on config.

Goals/specs

Reachability/access control, filter.
Fault tolerance: link, switch
Resources
Middleboxes

SDN view.



Global view of network
Fault tolerance
Access control
⋮

Why SDN?

+ Central ⇒ simpler complexity control/config.
+ Simple switches ⇒ easy to deploy.
– Controller: reliable, performant.
– Single org → single controller.

Formal verif for networks

Unique aspects:

~ No state abstraction. → No existential quantifiers.

Few properties: reach, filter;
resources;
middleboxes.

→ Exploring viable.

State space small:
stateless
pkt headers: IP/port, src/dst...

Alt plan: model checking

Th3 paper: PL view

Operator / policy

↓ reconfigure

Config: program NetCore

- → Predicate
- → Transform
- → Action: output ports.
- → Union, restrict.

↓

Flow table

Δ: priorities

Switch ↔ Flow table

Switch — unmatched pkt ↑ / add/del flow ↓

Featherweight OpenFlow

Exec. model

Controller = compiler + runtime

NetCore —compiler→ Controller executable

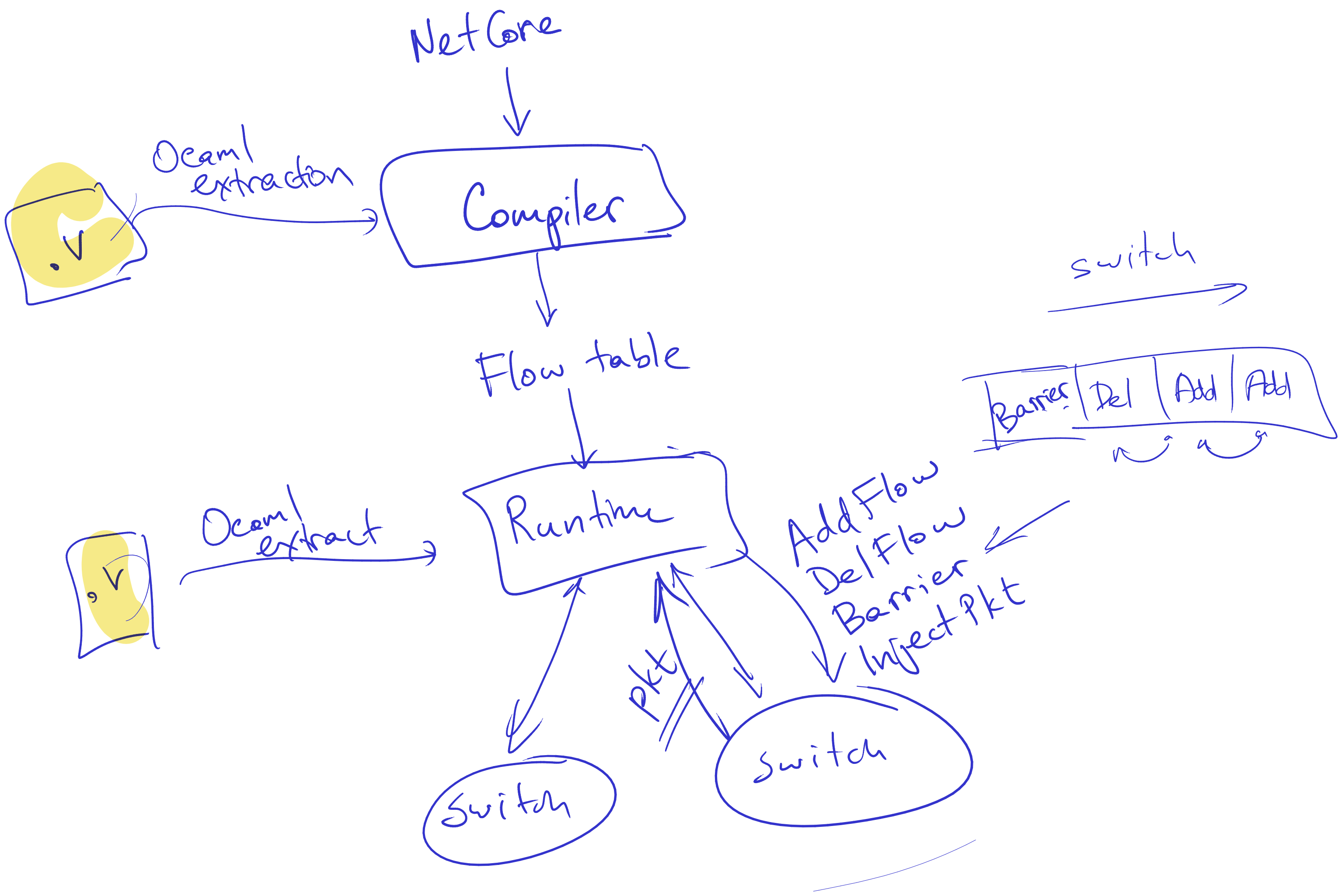
↓ run, interact w/ switches

Desired network.

Δ over CompCert:

- DSL
- Async API for switches.
- Prog. changes over time.

Q: Network, 3 services, each service logs to audit servers.



NetCore
Ocaml extraction
.v
Compiler
Flow table
Ocaml extract
.v
Runtime
pkt
Switch
Switch
AddFlow
DelFlow
Barrier
InjectPkt
switch
Barrier
Del
Add
Add

Bisimulation

spec: NetCore ⟶ exec. traces.

impl: FOF switches ⟶ exec. traces.

spec. traces ⊆ impl. traces

spec traces ⊆ impl. traces

Bi-simulation : trace equality.

Why?

Progress.

Why not?

Restricts hw.