Network verification.

Building block
Narrow spec, subtle.
Imples are complicated
Protocols
Distributed

SDN
Controller, central
Reduces complexity
Verif. target.

This paper:
Coq framework for SDN verification
Model for SDN.
Simple controller.
Network.

Goals/specs
- Reachability/access control, filter
- Fault tolerance: link, switch
- Resources
- Middleboxes

Traditional impl.
- Per-switch config
- Protocol between switches
- Agreement on protocol, not on config.
SDN views

Global view of network
Fault tolerance
Access control

Why SDN?
+ Central → simpler complexity
+ Simple switches → easy to deploy
- Controller: reliable, performant.
- Single org → single controller.
Formal verif for networks

Unique aspects:
- No state abstraction.
- Few properties: reach, filter, resources, middleboxes.
- State space small:
  - Stateless
  - pkt headers: IP/port, src/dst...

Alt plan: model checking

No existential quantifiers.

Exploring viable.
The paper: PL view

Operator/policy
reconfigure

Config: program NetCore

Predicate
Transform
Action: output ports.
Union, restrict.

Flow table

Switch

Controller = compiler + runtime

NetCore compiler
Controller executable

run, interact w/ switches
Desired network.

\[\Delta\text{ over Compiler:}\]

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

\[\Delta\text{ over Config:}\]

Fi + add/del flow Featherweight
OpenFlow Exec. model.

\[\Delta: \text{priorities}\]

Network, 3 services,
each service logs to audit server.
Bisimulation

spec: NetCore \xrightarrow{\text{exec. traces.}}

\text{spec. traces} \subseteq \text{impl. traces} \Rightarrow \subseteq \text{spec traces}

\subseteq \text{impl. traces}

impl: FOF switches \xrightarrow{\text{exec. traces}}

Bisimulation: trace equality.

Why?
- Progress.

Why not?
- Restricts hw.