Adding reliable data-plane acknowledgements to OpenFlow

(Project #4)

Peter Peresini
peter.peresini@epfl.ch

OpenFlow vision

"OpenFlow hides switch diversity behind a common configuration interface"

OpenFlow reality

Not really unifying behavior:

- different capabilities (e.g., flow table size)
- different rule installation speed
- different fidelity to the specification

Barrier messages

OpenFlow Barrier messages:

- enforce rule installation ordering
- inform when a rule was installed

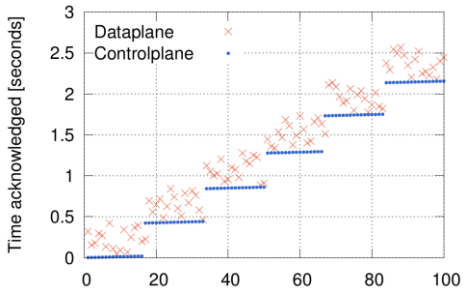
Used by many solutions!

- Consistent Updates, etc.

Barriers - reality

Despite barriers, rules are installed:

- up to 400 ms after barrier confirmation
- out of order



Barriers - fix

- switches are difficult to modify
- users need to wait for vendors to apply fixes
- what about older, existing models?

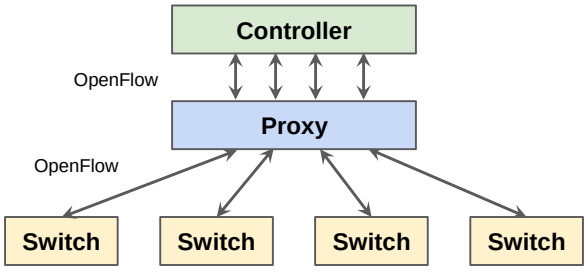
Solution:
Build a software layer below the controller that hides incorrect switch behavior

High-level project overview

1. develop layer fixing “bad” switch behavior
2. test your solution in a network emulator
3. evaluate & profile performance

Layer fixing bad switch behavior

- transparent proxy
  - between SDN controller and SDN switches
  - holds/modifies/creates OF messages



Proxy implementation details

- speaks OpenFlow 1.0 (and 1.3?)
  - reuse existing OF serialization libraries
    - investigate Flowgrammable vs OfSoftSwitch or others
    - need to create own “proxy-sdk” kit above that
- write own “proxy-SDK” framework (add timers, potential message queues, etc.)

Different proxy functions

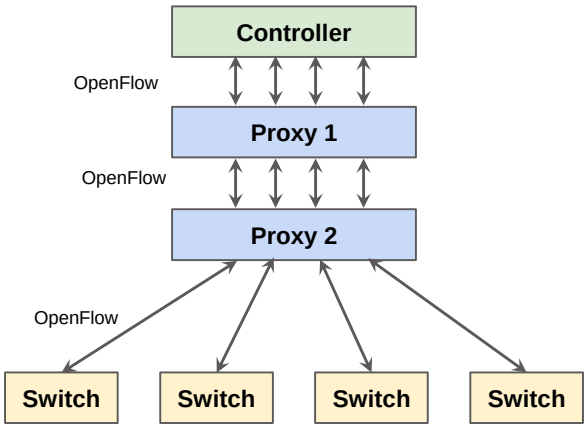
- pass-through (to test if things work)
- rate-limiting, keep-alive, etc.
- message id rewriting
- ensuring correct barriers
  - based on rule-is-in-dataplane acks
- rule-is-in-dataplane ack for different types of switches
- delayed-drop
  - enables dataplane ack for drop rules

Multi-functional proxy

Idea = chain several proxies

- decouples functionality
- easily deploy complex combinations of functionality
  - e.g. different proxy for each switch

SDN multi-function proxy



Chaining proxies

Potential performance hog

- repeated serialization-deserialization of OF messages
- one slow proxy delays whole path

Chaining proxies

If performance is a problem

a. bypass connection to next proxy which really needs it
   - complex configuration :-(

b. create framework for creating complex proxies
   - each sub-proxy running in its own thread
   - connect through message queues

Proxy functions - impl. details

- most of described functions semi-trivial
- interesting = rule-is-in-dataplane acknowledgement
  - quite technical, especially for some switches

Rule-is-in-dataplane ack proxy

- installs several new rules on the switch
- send periodic probe packets using neighbouring switches
  - need to compute these probes
- receive probes and decide whether flow is installed in dataplane

Computing probe packets

- highly technical!
- requires SAT/SMT solving
  - reuse existing SMT solvers such as Z3 / STP
  - reuse existing SAT solvers (e.g. MiniSat, Picosat, etc.) and write own conversion from SMT to SAT
    - start with existing Python implementation
- expected to be the performance bottleneck
  - experiment with various optimizations and alternatives

Testing & evaluating

Unittesting

- write unit-tests and microbenchmarks
- re-use existing frameworks such as GoogleTest or CppUnit, etc.

Testing & evaluating

Integration testing

- write proxy simulating bad behavior of a real hardware switch
- use Mininet to emulate whole network
- run different workloads and SDN controllers
  - (details not known yet)

Documentation & code

- setup/readme guides
  - must be easy to deploy
- extensive in-code documentation of API
- expected high quality of code
  - clean design
  - code reviews
  - working tests
  - adherence to code style

Additional work

- research prototype is incomplete
- need to do a lot of things properly

Additional work

- missing features
  - multi-rule modifications/deletions, action group modifications, etc.
- handle switch/controller re-connect
- correct XID proxying
  - with XID expiration, handling of all messages, etc.
- better OF 1.0 and 1.3 support
  - lots of devilish details there! (including probe generation, etc.)

Expected output

We really want project to be

- easily deployable
  - we want others to try and use it!
- reliable
  - no memory leaks, no crashes, no race conditions

Expected performance

We hope that proxy will be fast

- >40k OF messages proxied/sec
- >5k generated probes/sec

Coaching

- project will have PhD student as a tech lead
  - knows all relevant details
  - can give advice on design
  - will ensure code quality!
- however
  - you should be proactive
  - you need to manage yourselves