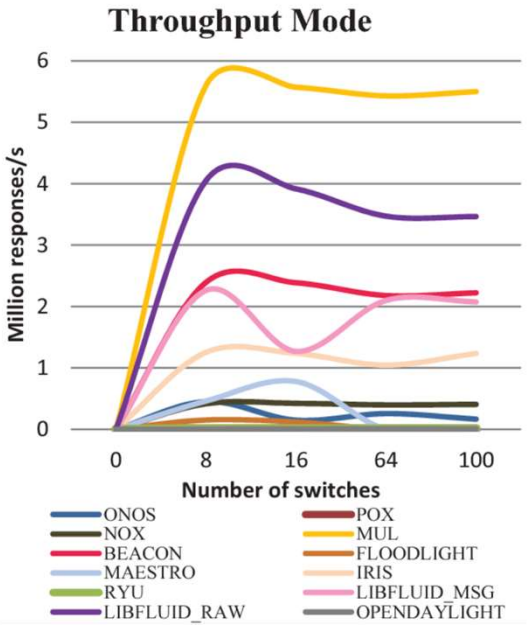
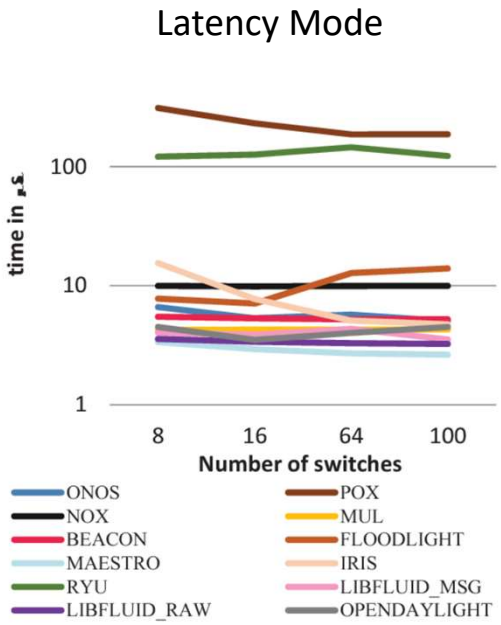
ACA ovs driver refactoring

Futurewei Cloud Lab

August 2021

OpenFlow controller performance evaluation

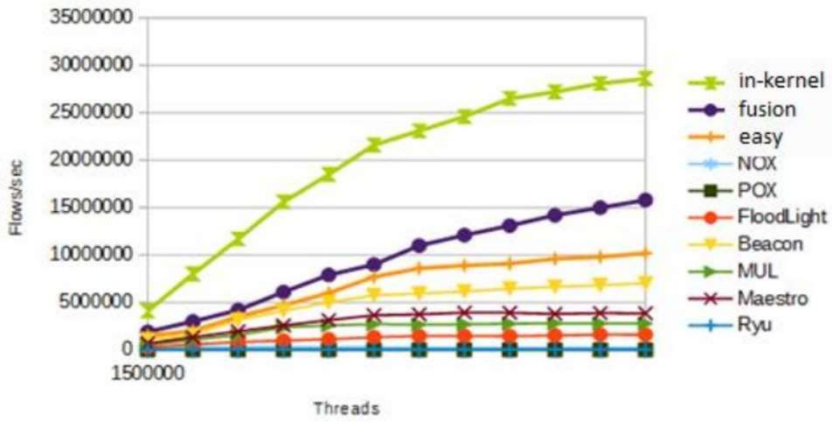




- From multiple horizontal comparison reports/papers of popular OpenFlow controllers like left and more, we can summarize that the names with relatively good performance are:
  - MUL (C)
  - Libfluid_raw / Libfluid_msg (C++)
  - Beacon (java)
  - Maestro (java)
  - NOX_MultiThreading (C++)
- From past performance test, python ovs agent learning switch throughput is ~1/70 of libfluid, meaning
  - 0.03 million fps
  - Similar to RYU in left charts

OpenFlow controller performance evaluation

I/O throughput (cbench + l2learning), fps



- Performance is **10M fps** (based on libfluid).
- Latency is **55us**.

- From this report of an easy wrapper of Libfluid-msg ("easy" in left diagram), it beats Beacon, Maestro and others in
  - packet-in -> local L2 discovery lookup -> add-flow of neighbor
  - "Threads" axis is from 1 – 12
  - could reach 10M fps with 12 threads

OpenFlow controller performance evaluation

Switches: 16

Controllers: Beacon, NOX MT, Floodlight, raw, msg

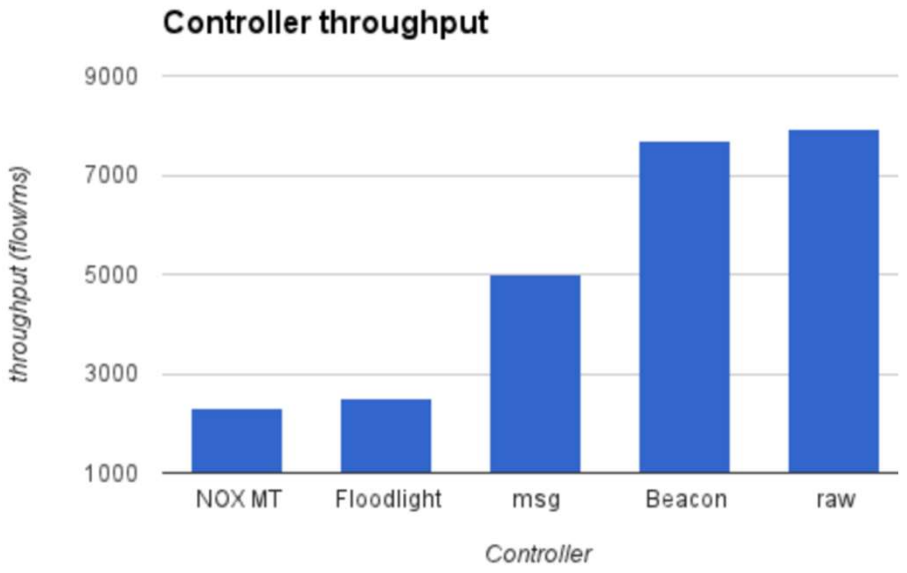
Threads: 8

Application: learning switch

Command: `cbench -c localhost -p 6653 -m 10000 -l 16 -s 16 -M 1000000 -t`

Results (higher is better):

Controller	flows/ms
raw	7929.39
Beacon	7682.54
msg	5013.35
Floodlight	2490.61



OpenFlow controller performance evaluation

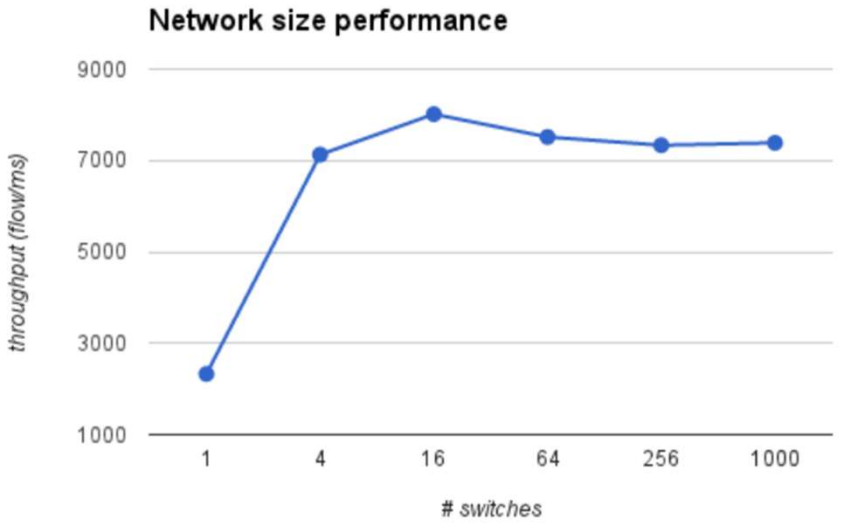
Threads: 8

Application: learning switch

Command: `cbench -c localhost -p 6653 -m 10000 -l 16 -s 16 -M 100000 -t`

Results (higher is better):

Switches	flows/ms
1	2327.88
4	7128.02
16	8013.98
64	7513.87
256	7334.20
1000	7386.28



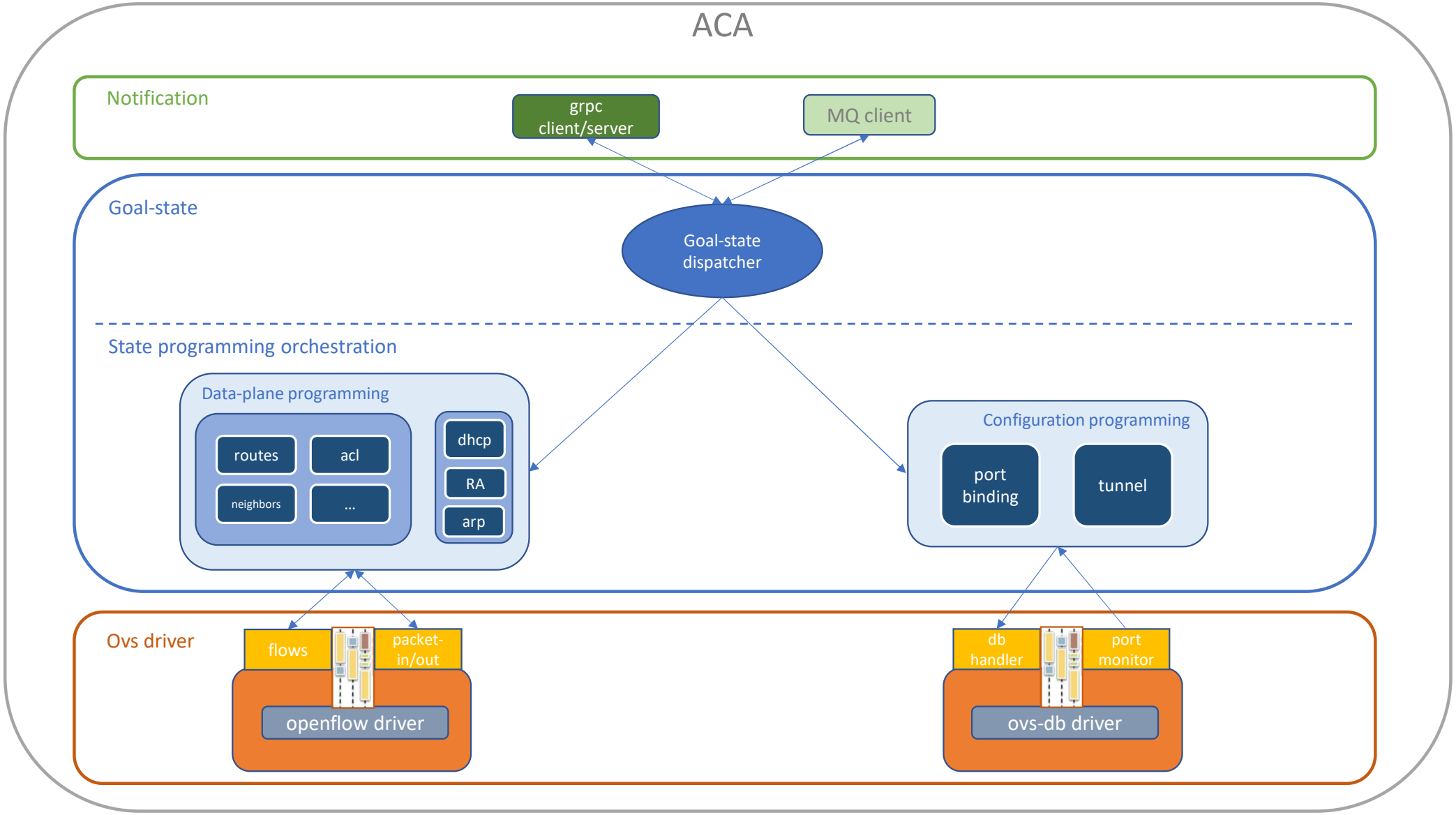
OpenFlow controller design evaluation

- From these OpenFlow controller projects, we can learn from them in order to achieve higher performance
  - Efficient multi-threading non-blocking I/O model
  - Lower-level programming language

- Unlike them
  - We only focus on one type of OpenFlow switch – ovs currently, instead of raw OF protocol we need to understand and tune ovs OF behavior
  - We don't care much about number of switches (since we are not building a central controller), the maximum for us will be 4 (br-tun + br-int + ovsdb + aux connection for further enhancement later)
  - Thus, we value making the best use of single ovs connection the most

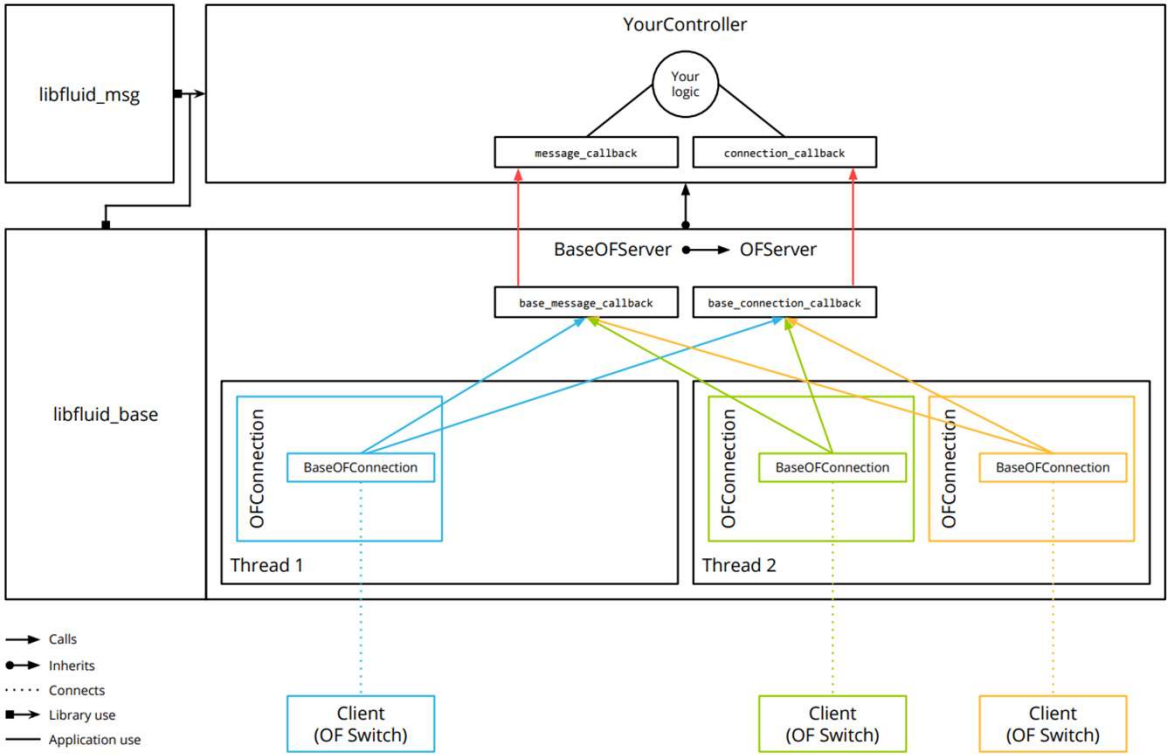
ACA V2 design
(including ovs-driver V2 refactoring)

- Layers
  - Upper
    - User update notification
    - Depending on upstream stack, can be grpc c/s or message queue library
  - Mid
    - Abstract goal states -> specific forwarding plane programming
    - Where ACA orchestrates and maintains states
  - Lower
    - Communicate with forwarding module, in our case is ovs (including ovs-db)
    - Where ovs-driver V2 refactoring takes place


ACA
Notification
grpc client/server
MQ client
Goal-state
Goal-state dispatcher
State programming orchestration
Data-plane programming
routes
acl
neighbors
...
dhcp
RA
arp
Configuration programming
port binding
tunnel
Ovs driver
flows
packet-in/out
openflow driver
db handler
port monitor
ovs-db driver

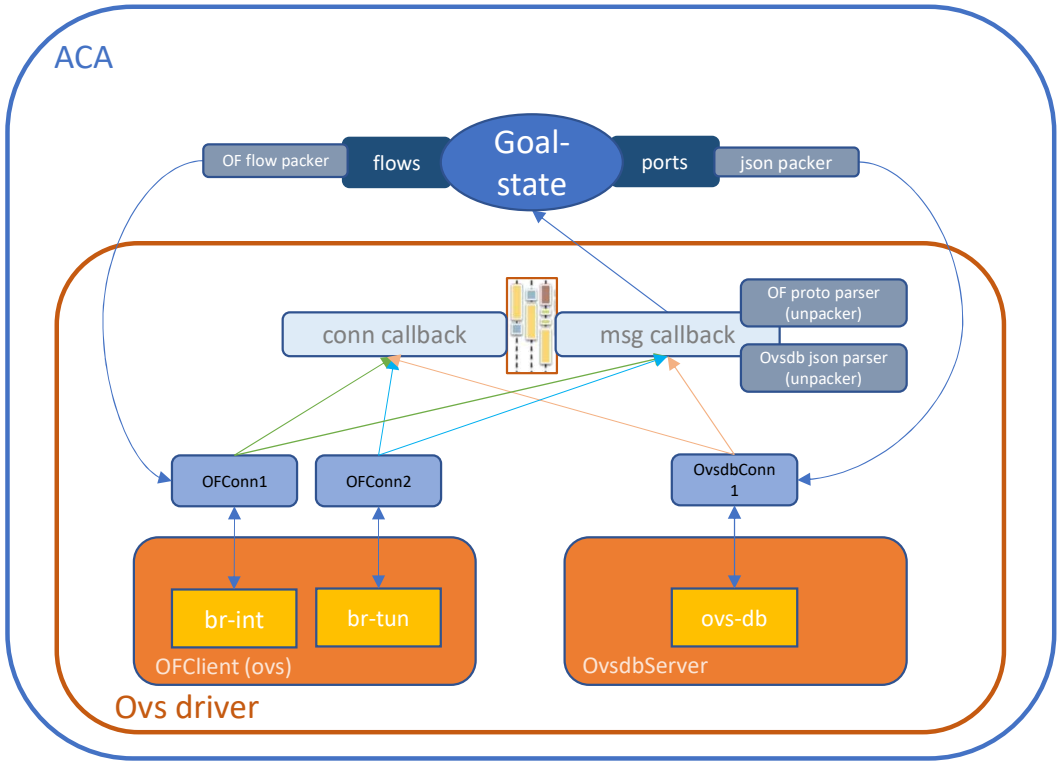
ACA ovs-driver V2 design

Libfluid fundamental



- Functions
  - Connection management
  - Event driven callbacks (non-blocking pipeline)
- Support numbers of connections (switches)
- High-performance server level event handling
- Based on raw OpenFlow protocol, needs some customization to support Ovs
- Does not include Ovsdb protocol/handler

ACA ovs-driver V2 design
break-down

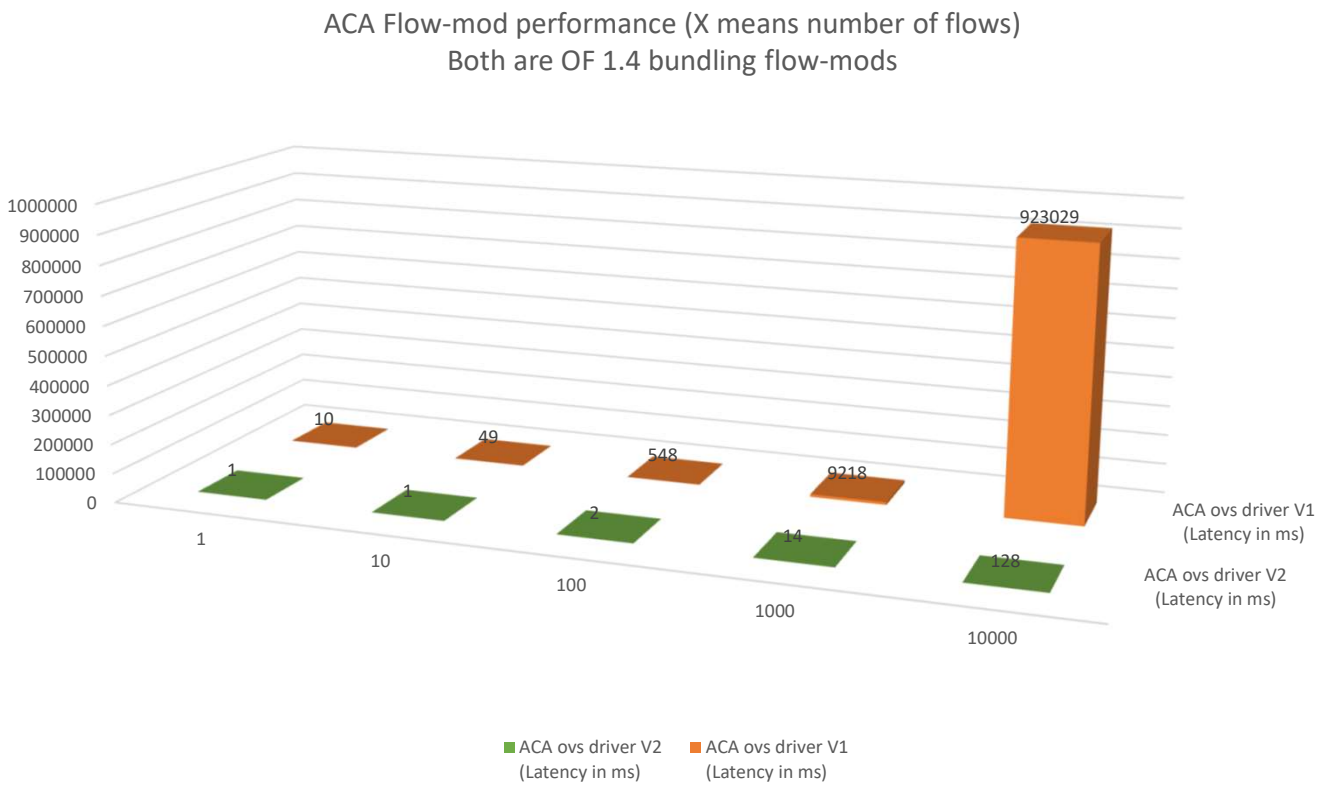


- Connections
  - br-int
  - br-tun
  - ovs-db
- Connection callbacks
  - connection up
  - connection down
- Message callbacks (receiver/unpacker)
  - openflow parser (from OFConn)
    - Ovs reply (dump-flows, call reply with xid, bundle transaction reply etc.)
    - Packet-in
  - json parser (from OvsdbConn)
    - db record change notifications (for example ports)
    - db query
    - transaction reply
- Programmer (sender/packer)
  - OFConnection
    - Construct flows -> add/mod/del flows (OF 1.3 and below)
    - Bundling flow-mods (OF 1.4 and above)
    - Packet out
  - OvsdbConnection
    - Construct json rpc params -> ovsdb transactions

ACA ovs-driver V2

Performance improvement

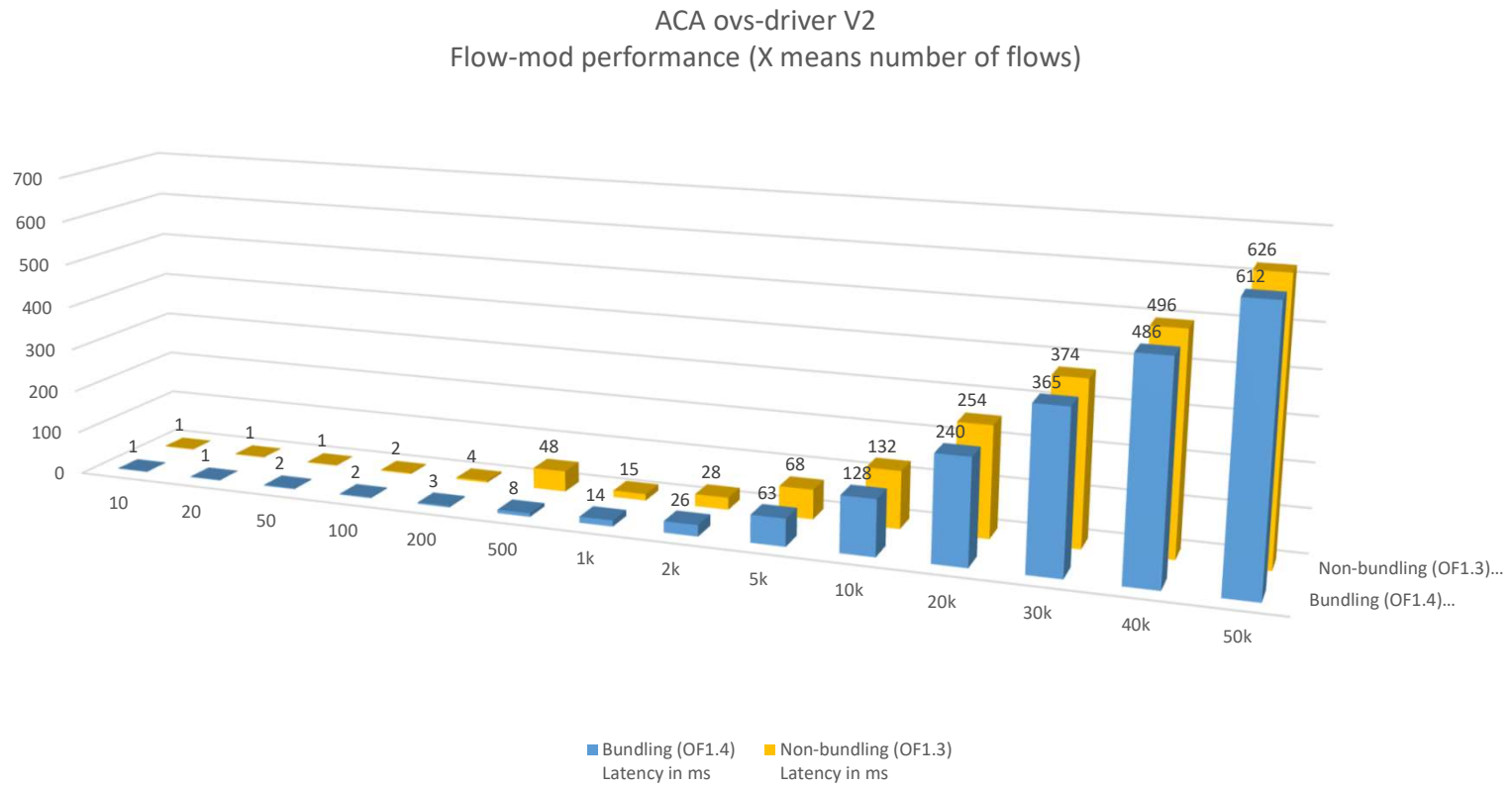
Flows	ACA ovs driver V2 (Latency in ms)	ACA ovs driver V1 (Latency in ms)
1	1	10
10	1	49
100	2	548
1000	14	9,218
10000	128	923,029



ACA ovs-driver V2

Performance expectation

Flows	Bundling (OF1.4) Latency in ms	Non-bundling (OF1.3) Latency in ms
10	1	1
20	1	1
50	2	1
100	2	2
200	3	4
500	8	48
1k	14	15
2k	26	28
5k	63	68
10k	128	132
20k	240	254
30k	365	374
40k	486	496
50k	612	626



ACA ovs-driver V2

Conclusions of performance improvements

Methods	**Effect (measured by percentage of latency it saved)**	**Comments**
Multiplexing non-blocking I/O model between ACA and ovs	99 – 99.8%	Instead of using V1 ofputil vconn which opens and closes each time and update flows sequentially, leverage non-blocking I/O model. Reduced add flow latency • For 100 rules from 548ms to 2ms which is 99.7% • For 1k rules from 9218ms to 14ms which is 99.8%
Use OF 1.4 bundling flow-mod transaction	14 - 42% (if upgrade from OF 1.0) 5 – 8% (if upgrade from OF 1.3)	From OF 1.0, reduced add flow latency • For 5k rules, from 110ms to 63ms which is 42% • For 15k rules, from 227ms to 195ms which is 14% From OF 1.3, reduced add flow latency • For 5k rules, from 68ms to 63ms which is 8% • For 20k rules, from 254ms to 240ms which is 5.8%