SDN Control Plane Approaches: A DDlog Retrospective

Ben Pfaff (VMware)

A packet shows up:
What do we do with it?

Network Policy Arithmetic

Suppose we have N VMs:	O(N) policy data.
Pack VMs several to a node:	O(N) nodes.
Distribute policy to all nodes:	$O(N^2)$ policy data to distribute.

This is true regardless of how we distribute policy data*

This is too much. How do we compromise?

*Multicast

If we can multicast the policy data to the nodes, we use only O(N) bandwidth.

I don't know an SDN system that does this.

Research (or references) needed.

1. Keep N small

Small N makes $O(N^2)$ practical.

- Early versions of OVN were OK for N = 2000.
- Most enterprises have 7 or fewer racks.
- The definition of “large” might be larger than one expects.

To some extent this is just “hope it works.”

2. Chew away at constant factors

- Uniformity.
- Subsetting.
- Compression.
- Simplicity.

3. Reactive control

Early SDN controllers set up one microflow at a time *reactively*, but:

- Latency
- Load
- Failure

Newer controllers are *proactive*.

OVS internals were once microflow-based; we invented megaflows.

Can we invent megaflows for controllers?

4. Federation

Divide the network into smaller networks.

Use a hierarchy of control.

Networks must be independent or mostly so.

5. Don’t change

If the network is static, or only changes rarely, it might not matter that it’s expensive to change.

6. Don’t centralize

Do we need centralization to accomplish our goals?

- Can a node do what we want with less than O(N) communication?
- Is network virtualization really needed?

7. Predictability

Eliminate the need to distribute per-VM data.

For example, encode VM MAC and IP addresses to imply the security policy and their node of residence.

8. Incremental Control

Can we just compute and transmit changes?

Incremental Control

	Full	Incremental	
A.	**N** VMs	**O(1)** VMs change	controller
B.	↓ **O(N)** computation	↓ **O(1)** computation	controller
C.	**O(N)** logical flows	**O(1)** logical flows changes	controller
D.	↓ **O(N^2)** bandwidth	↓ **O(N)** bandwidth	communication
E.	**O(N)** logical flows	**O(1)** logical flows change	per node
F.	↓ **O(N)** computation	↓ **O(1)** computation	per node
G.	**O(N)** OpenFlow flows	**O(1)** OpenFlow flows change	per node

Incremental Control: Assumptions

"Cold start" is fast enough.

A. Changes are small.

B. **Efficient delta computation.**

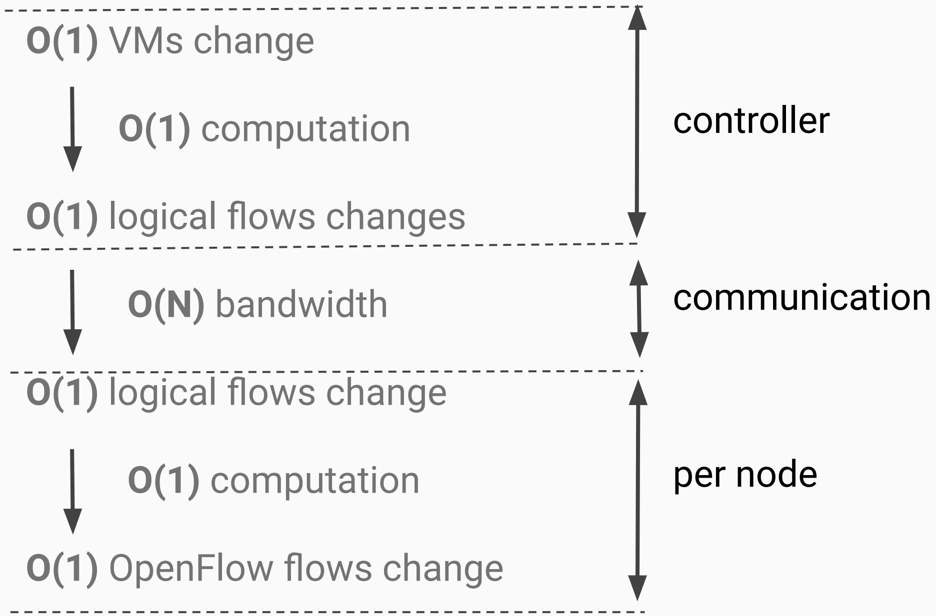
C. **|ΔOutput| = O(|ΔInput|).**

D. Efficient distribution of incremental changes.

E. (Ditto)

F. **Efficient generation of OpenFlow deltas.**

G. OVS handles OpenFlow deltas efficiently.



Incremental Control assumption C: $|\Delta \text{Output}| = O(|\Delta \text{Input}|)$

If a small input change can yield a much bigger output change, then incremental computation will not be effective.

If such changes happen only rarely, it might still be OK in practice.

OVN load balancers had such a problem: in important cases, changing one in a simple way could affect a hugely disproportionate number of logical flows.

("Load balancer groups" should help.)

Incremental Computation assumptions B+F: Efficient delta computation

ovn-northd and ovn-controller are complicated and hard to make incremental. We've tried three approaches:

- Ad hoc in C: in ovn-controller (in 2016 by Ryan Moats). This proved too hard to make reliable and was reverted.
- Disciplined in C: in ovn-controller (by Han Zhou). Uses an engine of C callbacks. Still working! Some known issues (based on the tests).
- **Automatic in DDlog: in ovn-northd (by Leonid Ryzhyk and others).**

Incremental ovn-northd with DDlog: Best case

From empty, add another router 250 times:

	step 1	step 250	total runtime
C:	.14 s	1.04 s	107 s
DDlog:	.13 s	.15 s	35 s

[*] https://mail.openvswitch.org/pipermail/ovs-dev/2021-April/381745.html

Incremental ovn-northd with DDlog: Worst case

Cold start with huge load balancers, then delete each of them:

	wall time	CPU time	RAM
C:	1:20	~87 s	3.8 GB
DDlog:	3:08	187 s	14.2 GB

- DDlog processes each change “twice”.
- DDlog can’t as easily parallelize processing.
- DDlog indexes data to enable incrementality.

Other DDlog issues

- New language:
  - Learning
  - Linking
  - Bugs
  - Tools
- New language paradigm
- Double work for programmers
- Slow builds

Questions?