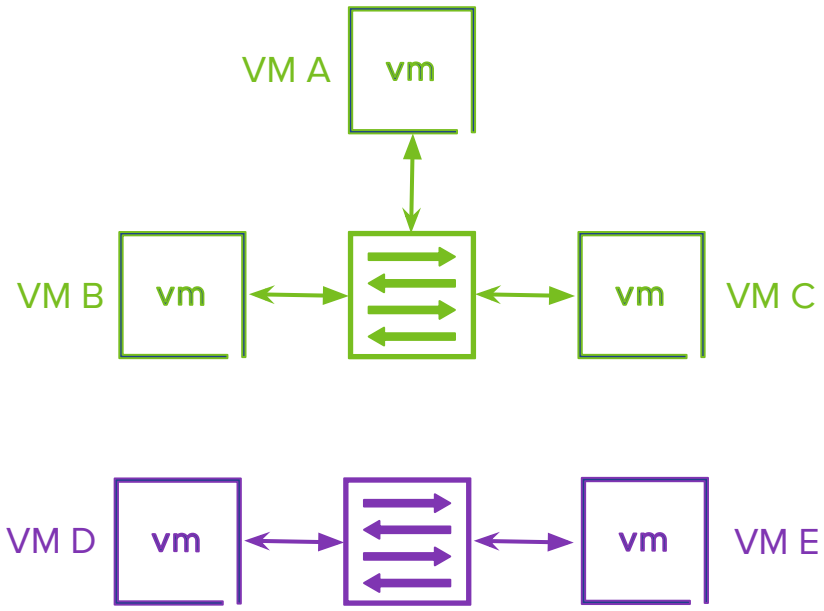
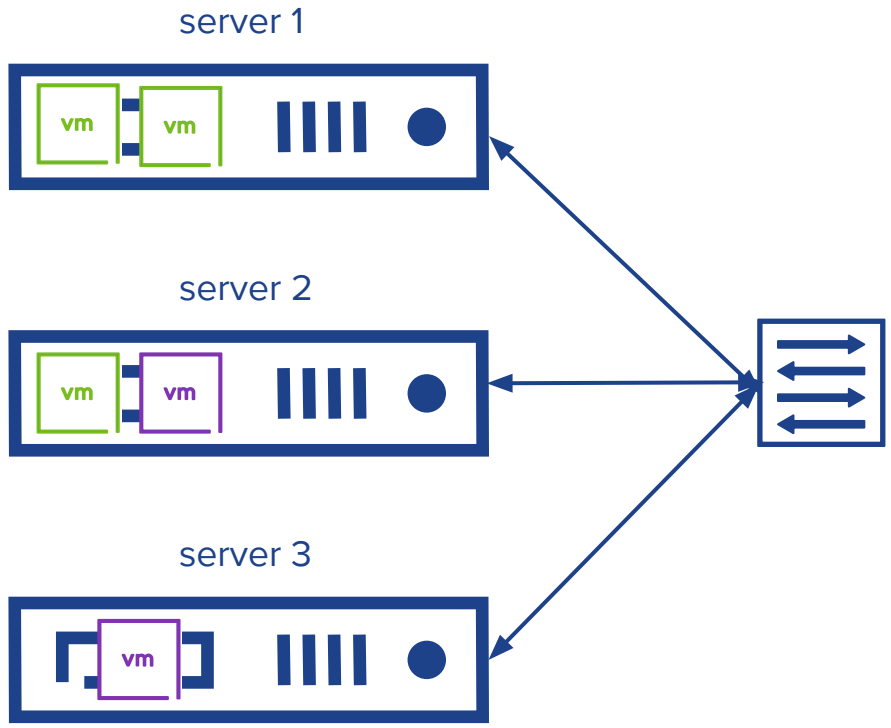
Scaling SDN Policy Distribution

Ben Pfaff

Network Virtualization Background





A packet shows up.

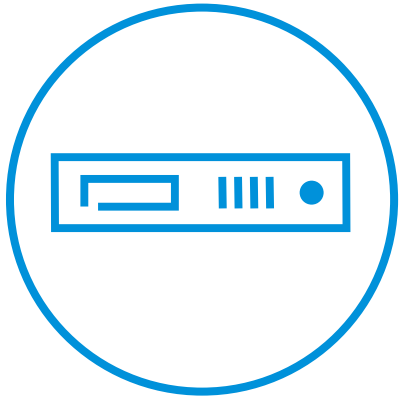
What do we do with it?

Network Policy Arithmetic

N VMs

Therefore we have:

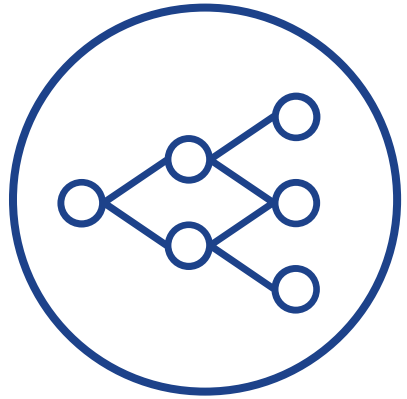
O(N) policy data



Pack VMs into nodes

Therefore we have:

O(N) nodes



Distribute O(N) policy to O(N) nodes

Therefore we distribute:

$O(N^2)$ policy data

How do we distribute $O(N^2)$ policy data?

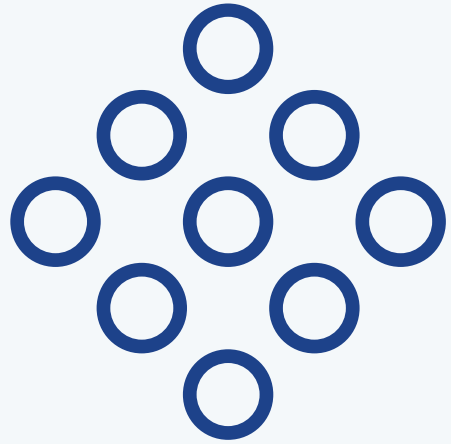
(without multicast)

1: Keep N small

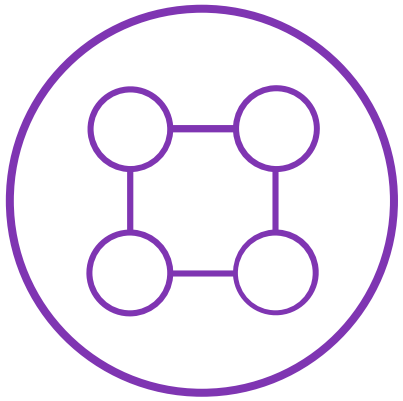
Small N makes $O(N^2)$ practical.

- 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.”

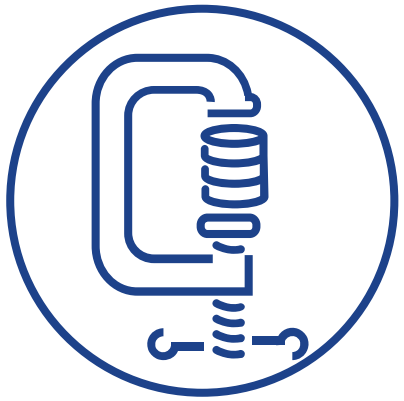


2: Chew away at constant factors



Uniformity

Subsetting



Compression

Simplicity

3: Reactive Control

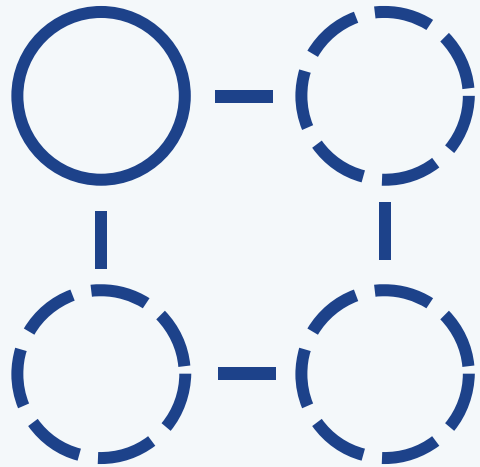
Early SDN controllers set up one microflow at a time reactively, but:

- Latency
- Load
- Failure

Newer controllers are proactive.

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

Can we invent megaflows for controllers?

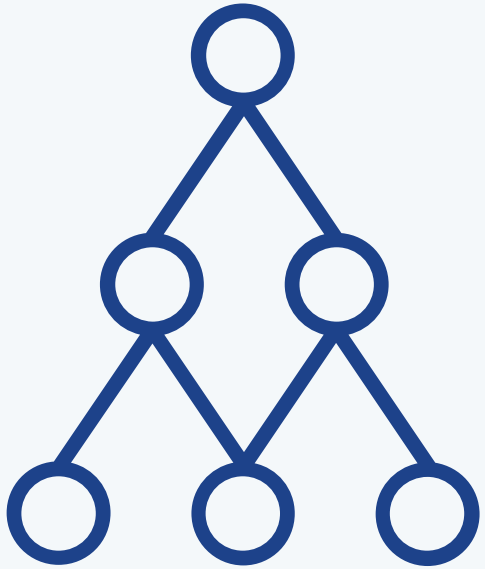


4: Federation

Divide the network into smaller networks.

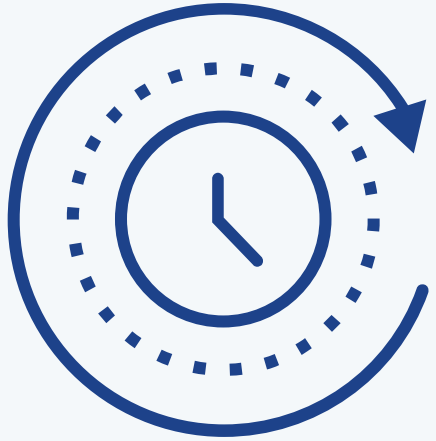
Use a hierarchy of control.

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

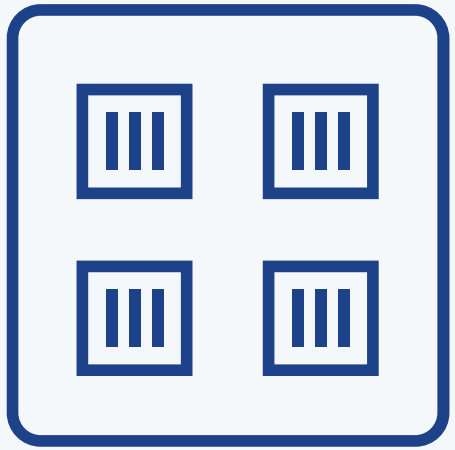
Do we need centralization to accomplish our goals?

- 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.

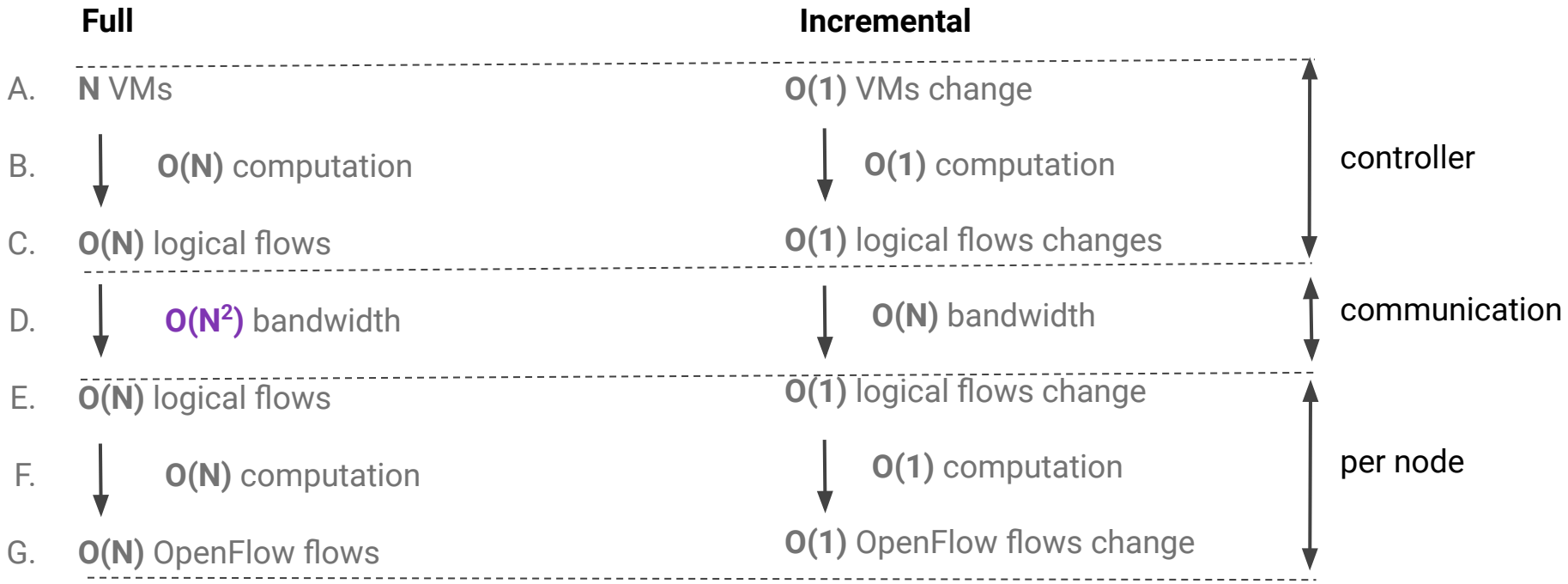
For example, encode VM MAC and IP addresses to imply the security policy and their node of residence.



8: Incremental Control

Can we just compute and transmit changes?

Incremental Control: Basics



Incremental Control: Assumptions

"Cold start" is fast enough.

A. Changes are small.

B. **Efficient delta computation.**

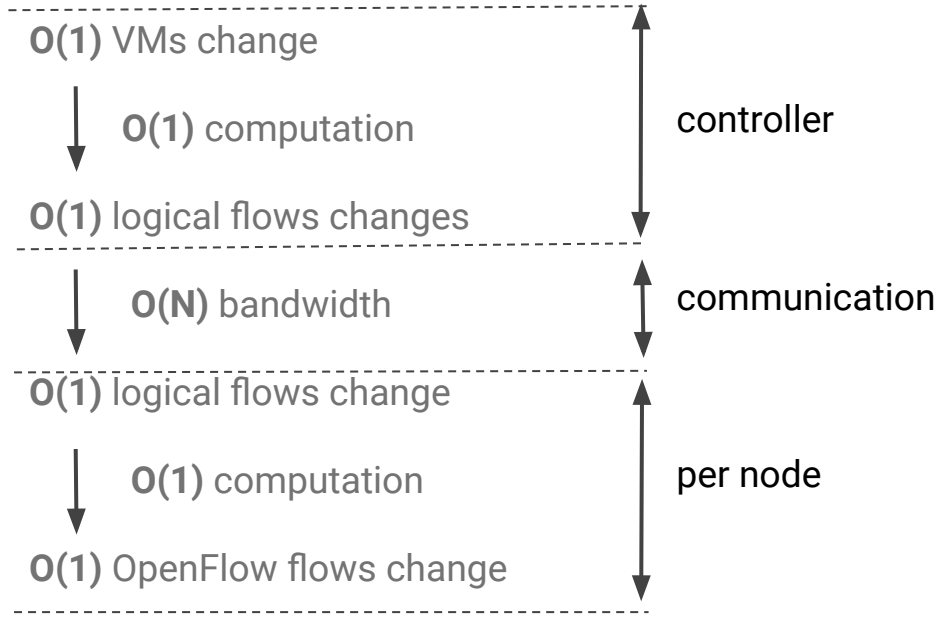
C. **|ΔOutput| = O(|ΔInput|).**

D. Efficient distribution of incremental changes.

E. (Ditto)

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

G. OVS handles OpenFlow deltas efficiently.



Assumption C: |ΔOutput| = O(|ΔInput|)

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.)

Assumptions B+F: **Efficient delta computation**

The two computations in our system are complicated and hard to make incremental. We tried three approaches:

- Ad hoc in C: in the per-node computation (in 2016). This proved too hard to make reliable and was reverted.
- Disciplined in C: in the per-node computation. Uses an engine of C callbacks. Still working! Some known issues (based on the tests).
- **Automatic in DDlog: in the controller computation.**

Incremental controller 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 controller 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.

Thank You