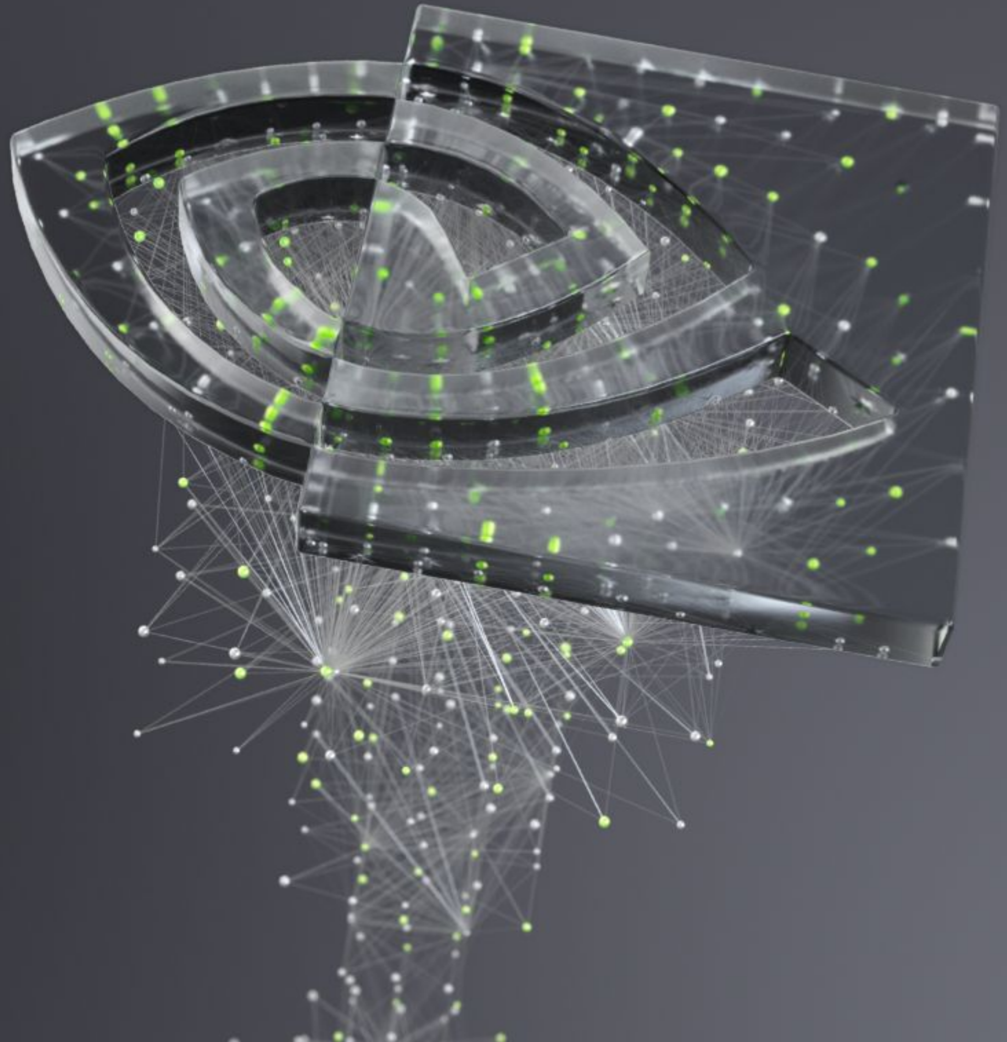


NVIDIA

Scale OVN
To The Next Level

Han Zhou
NVIDIA
OVSCON 2021

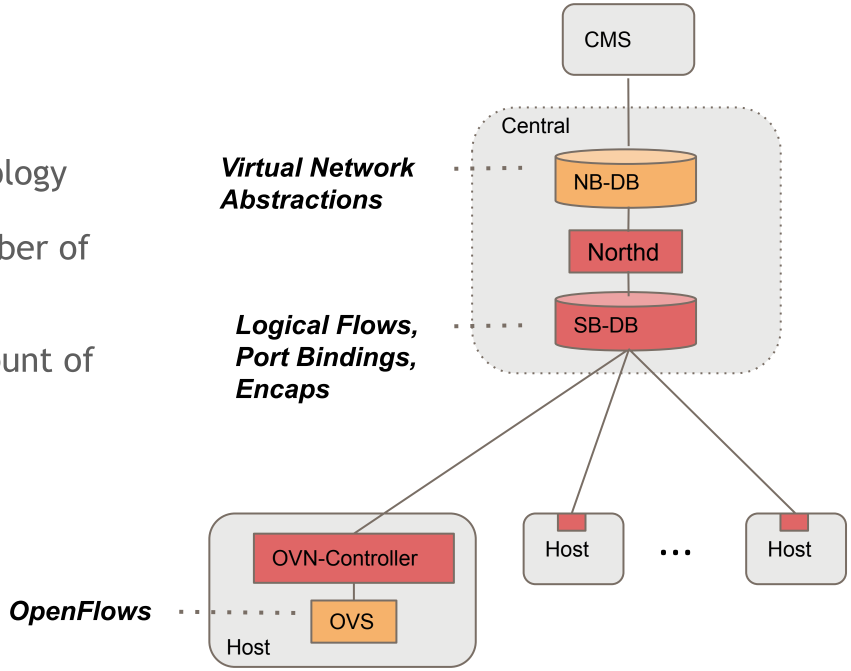
Agenda

- Scale challenges overview
- Trade IP mobility for scalability
- Logical flow tagging
- ACL optimizations
- Thoughts on incremental processing

OVN Control Plane

Scale Challenges Overview

- Bottlenecks
  - Northd
    - Processes large size of logical topology
  - SB-DB
    - JSON-RPC sessions for a large number of hosts
  - OVN-Controller
    - Processes and generates huge amount of flows
- Metrics
  - Latency
  - Throughput
  - CPU/Memory
- Factors
  - Data size
  - Number of nodes
  - Change rate



Trade IP Mobility for Scalability

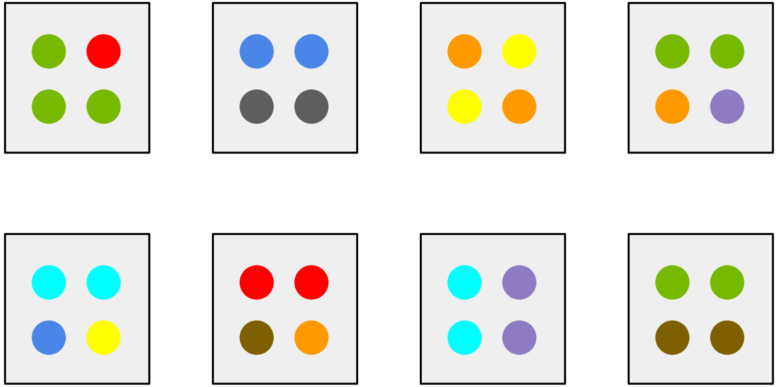
Pin Logical Switches to Nodes

Best Scenario

Lots of small isolated tenants

- Full-mesh connectivity within tenant.
- No connectivity between tenants.
- Each node cares about a small portion of the whole logical topology.
  - => A small portion of SB DB data need to be processed by each ovn-controller.

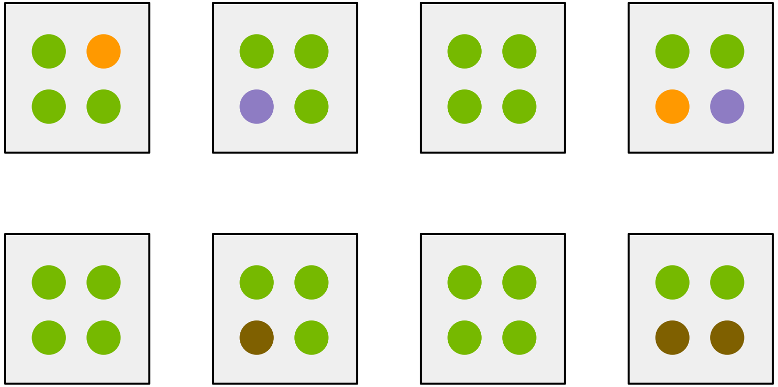
HOWEVER ...



Reality

When there are very big tenants

- A big tenant's workloads present on most nodes.
- Each node need to know almost the whole topology.
  - => Each ovn-controller processing almost the whole SB DB data.

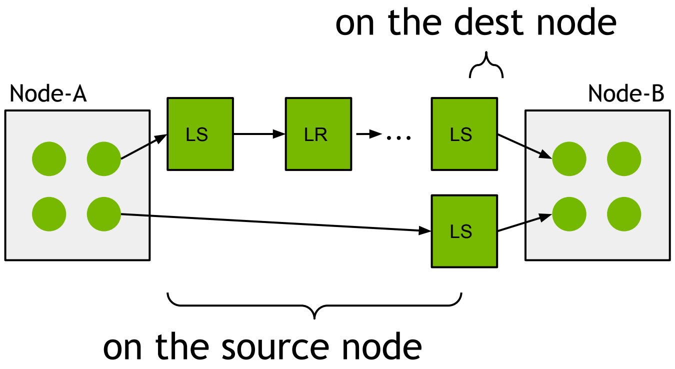


Data required by each node

IP-location decoupled (any-ip-anywhere)

- On the source node - O(p), p = #LSPs
  - Logical flows that find the dest LSP
  - Port binding that tells the physical location of the dest LSP (the dest node)

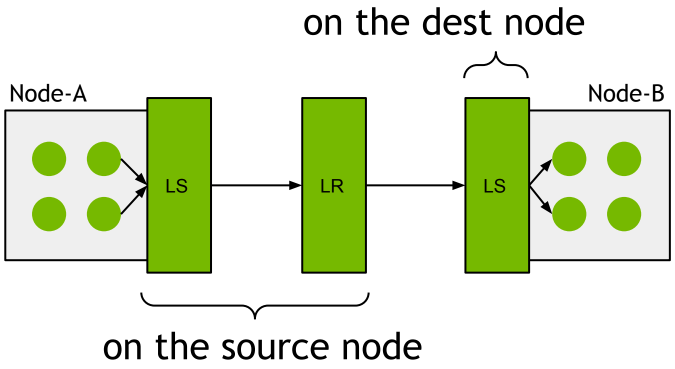
- On the destination node
  - Egress logical flows of the last hop LS
  - Port binding of the local VIFs



Trade IP mobility for scalability

Pin logical switches to nodes

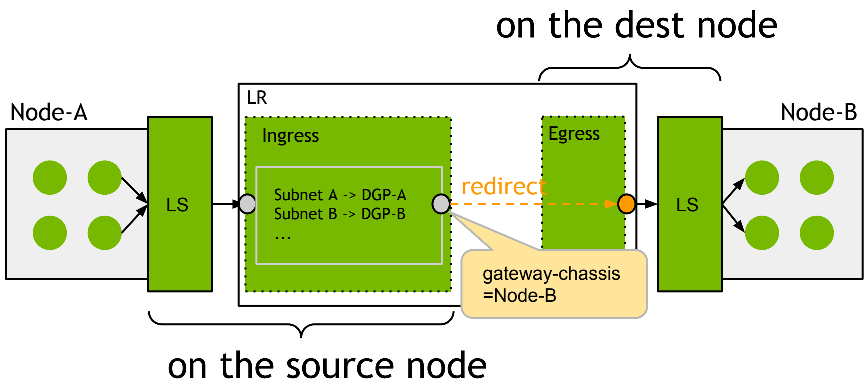
- Node-based subnet allocation
  - Contained in node-level logical switches
  - IP mobility at IP-block level only
- Between the nodes: routing - O(n), n = #nodes
  - Subnet A -> Node A (LS-A)
  - Subnet B -> Node B (LS-B)
  - ...
- Within a node: switching - O(v), v = #VIFs
  - IP1 -> MAC1 -> LSP1 -> VIF1
  - IP2 -> MAC2 -> LSP2 -> VIF2
  - ...



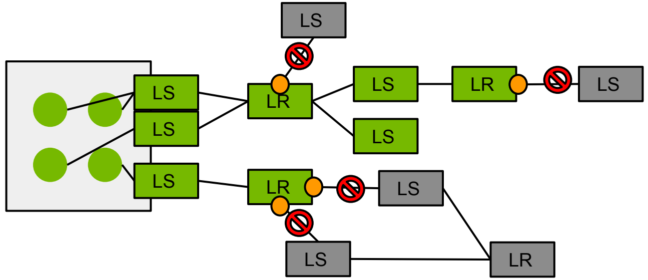
Distributed Gateway Port

Use DGPs to pin logical switches to nodes

- Distributed Gateway Port
  - A LRP with gateway-chassis set
  - Originally implemented for L3 Gateway
  - Non-distributed part: chassis-redirect-port
    - Redirected packets to a node for further pipelines



- Enhancements
  - Don't flood-fill local-DPs across DGP boundary (*when distributed NAT is not used*)
  - Support multiple DGPs per LR

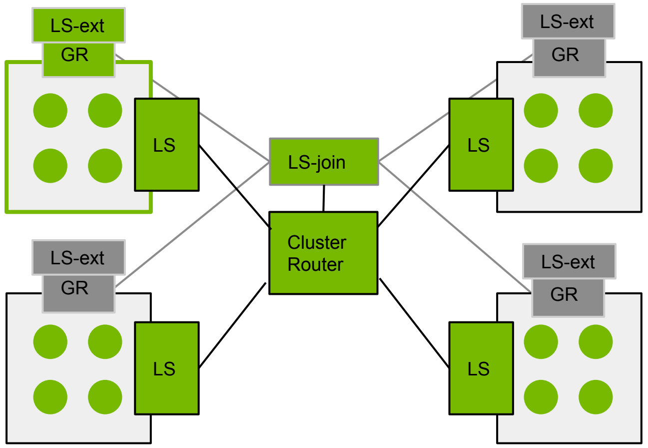


May need a better name for DGP:
Distributed Chassis-redirect Port

OVN-Kubernetes (before)

Full-mesh cluster pod network

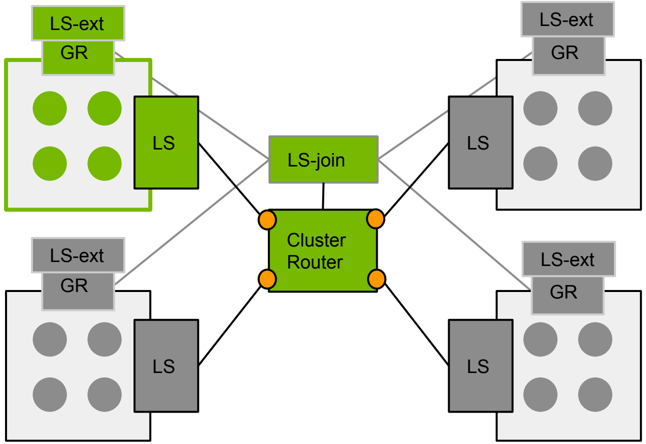
- Node-level subnets and LSes connected by a single shared cluster-level LR
- Data required by each node:
  - Datapaths
    - Node LSes x N
    - Node local GR and LS-ext
    - Cluster router, LS-join
  - Port-bindings
    - All LSPs
- “ovn-monitor-all” always set to true
  - Otherwise SB CPU too high, because the monitor condition is too big.



OVN-Kubernetes (now)

Use DGPs to pin logical switches to nodes

- Node-level subnets and LSes connected by a single shared cluster-level LR
- Data required by each node:
  - Datapaths
    - Node LSes x 1
    - Node local GR and LS-ext
    - Cluster router, LS-join
  - Port-bindings
    - ~~All LSPs~~ Node local LSPs
- "ovn-monitor-all" can be set to false
  - Each node only cares about a small portion of the SB data.

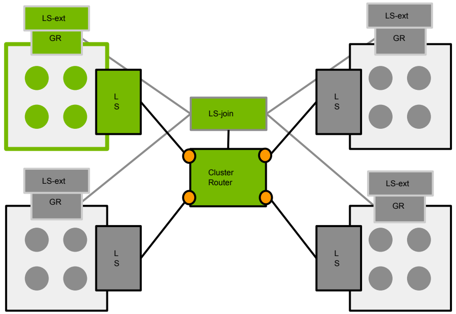


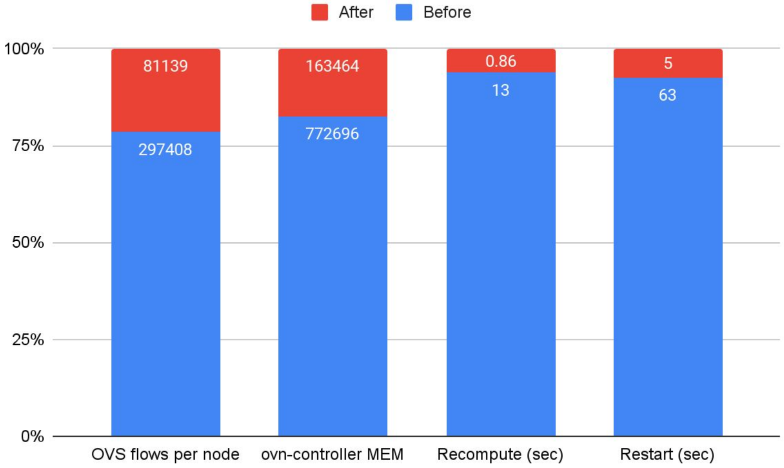
Benefits

- Faster ovn-controller recompute
  - Restarts (ovn-controller, OVS)
  - Node add/deletions
  - Other none I-P changes
- Faster I-P
  - Less DPs and PBs to process
  - Only one local DP for each DP group
- Conditional monitoring
  - SB server: higher cost for filtering but lower cost for data transferring
  - ovn-controller: lower IDL cost
- Memory savings on nodes
  - Less OVS flows to maintain in both ovn-controller and OVS
  - Less SB IDL data with conditional monitoring

Scale Test Result

- Environment:
  - CPU: Intel i9-7920X@2.90GHz
  - OVN Commit ID: *22298fd37908*
- Scale:
  - 1000 nodes, 10 LSPs per node
  - 2 PGs, each with 2000 LSPs
  - 5 pair of stateful ACLs: PG1 ⇔ PG2
- Result:
  - > 10x faster
  - 80% less memory





Further Improvement

Remove all non-local LSP related flows

- Still one flow per-LSP left on every node:
  - ARP resolving for LSPs happens at LR pipeline
- Goal:
  - O(n), n = # nodes
- Solution:
  - Move ARP resolving for LSPs to LS pipeline

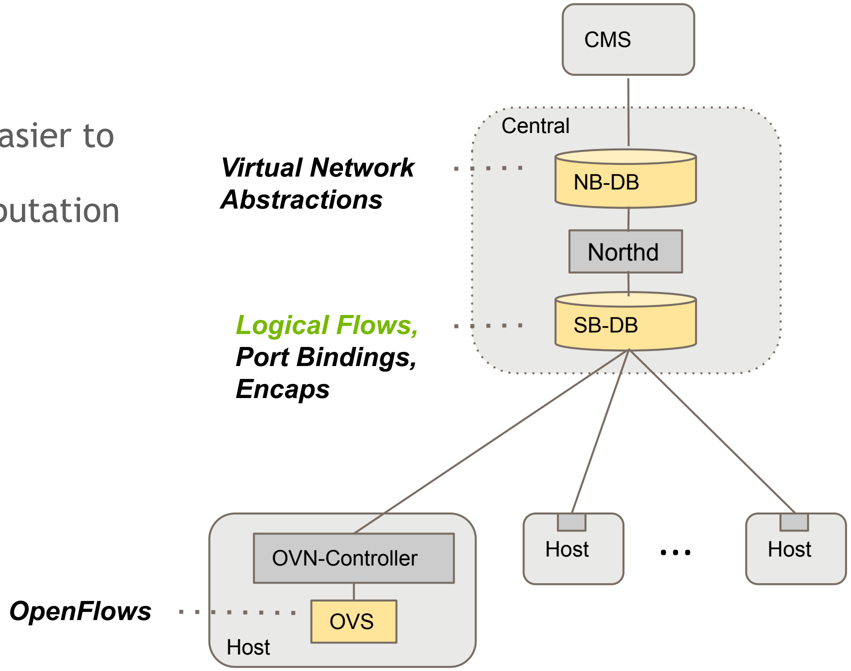
Logical Flow Tagging

Provide metadata for processing

Logical Flows Revisit

Pros & Cons

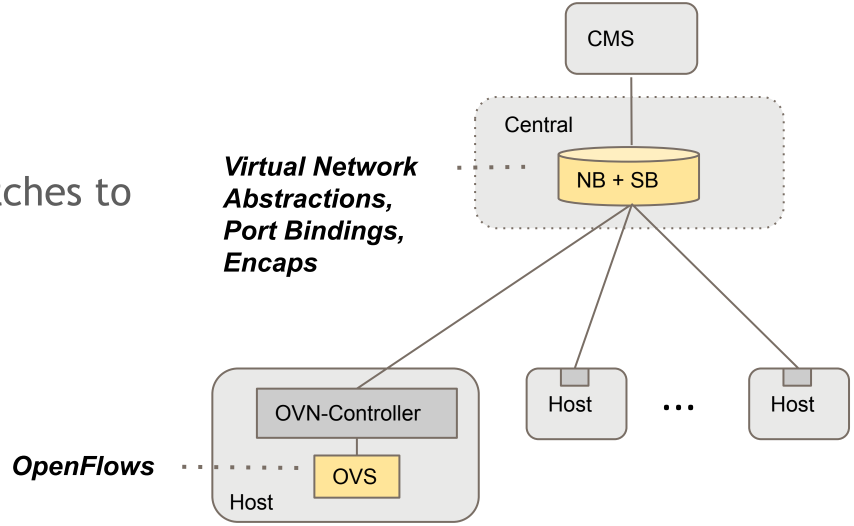
- Pros:
  - An intermediate representation that is easier to understand and debug
  - Centralized processing for common computation
- Cons:
  - An extra layer of processing cost
  - Strings (unstructured) - metadata lost



Remove the Logical Flow Layer

Needs more evaluation ...

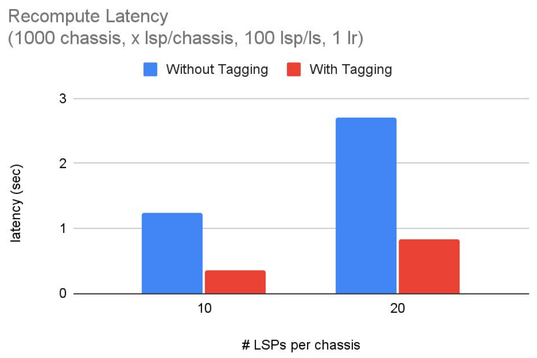
- Moving northd functions to every ovn-controller
- Almost rewriting OVN
- String parsing still needed for
  - ACL
  - QoS
  - Logical_Router_Policy
- No obvious benefit with “pin logical switches to nodes”



Logical Flow Tagging

Provide metadata for processing

- A new column in Logical_Flow table: tags
  - Key-value pairs providing help for ovn-controller to process logical flows more efficiently.
- The first use-case: in_out_port
  - key=*in_out_port*
    - For ingress pipeline, value=inport
    - For egress pipeline, value=outport
  - Filter out non-local logical flows before parsing.
  - Test result with full-mesh topology =>
- Limitation
  - Useful only if northd can provide the information.
  - E.g. doesn't help for ACL flows - northd doesn't parse the match string in ACLs.



ACL Optimizations

For an efficient distributed firewall

ACL Scaling Problem

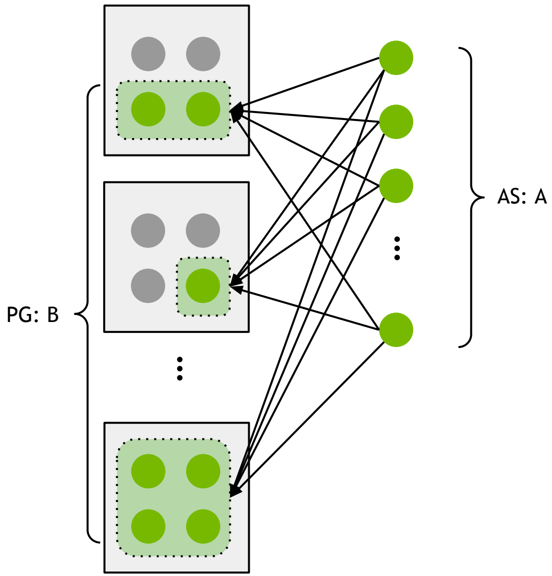
ACLs with Address-sets and Port-groups

- An ingress policy:
  - *Allow IPs in address-set A to access LSPs in port-group B*
- Direction: to-lport
- Match: outport == @B && ip4 && ip4.src == $A
- Action: allow-related
- M + N + 1 OpenFlow rules (M = # local VIFs of PG_B, N = # IPs in AS_A)

  - ip, reg<outport>=p1: actions=conjunction(<id>, 1/2)
  - ip, reg<outport>=p2: actions=conjunction(<id>, 1/2)
  - …
  - ip, reg<outport>=pM: actions=conjunction(<id>, 1/2)

  - ip, nw_src=ip1: actions=conjunction(<id>, 2/2)
  - ip, nw_src=ip2: actions=conjunction(<id>, 2/2)
  - …
  - ip, nw_src=ipN: actions=conjunction(<id>, 2/2)

  - ip, conj_id=<id>: actions=<the real action>



- Scale problem
  - N can be huge, but the Address-set change handling is naive.
  - VMs/Containers come & go => AS_A changes =>
  - Regenerate all the M + N + 1 OVS flows.

Consistent Conjunction ID Generation

Avoid unnecessary OVS flow-mod

- Before
  - Reprocessing a logical-flow uses a new conjunction ID (unless logical-flow cache is enabled)
  - => All the M + N + 1 flows are changed
  - => all deleted and reinstalled to OVS
    - Control plane latency
    - Dataplane impact - megaflow cache churns
- Now
  - Logical-flow uuid based consistent conjunction ID allocation algorithm
  - => Conjunction ID doesn’t change in 99.999…% cases
  - => Only the flows corresponding to the added/deleted IPs of the address-set are updated to OVS

- ip, nw_src=ip1: actions=conjunction(<id>, 2/2)
- ip, nw_src=ip2: actions=conjunction(<id>, 2/2)
- …
- ~~ip, nw_src=**ipOld**: actions=conjunction(<id>, 2/2)~~
- …
- ip, nw_src=**ipNew**: actions=conjunction(<id>, 2/2)
- …
- ip, nw_src=ipN: actions=conjunction(<id>, 2/2)

Fine-grained Address-set I-P (WIP)

Avoid unnecessary flow regeneration

- Why
  - Cost of reprocessing a single ACL logical flow can be high, when AS size is big
  - When churn rate is high, ovn-controller will be busy processing AS changes
- Goal
  - Only OpenFlow rules related to the changed AS members are computed

    - ip, nw_src=ip1: actions=conjunction(<id>, 2/2)
    - ip, nw_src=ip2: actions=conjunction(<id>, 2/2)
    - …
    - ~~ip, nw_src=**ipOld**: actions=conjunction(<id>, 2/2)~~
    - …
    - ip, nw_src=**ipNew**: actions=conjunction(<id>, 2/2)
    - …
    - ip, nw_src=ipN: actions=conjunction(<id>, 2/2)

- How
  - Track address-set information throughout the logical flow compiling
  - Maintain the mapping between each IP of address-sets to the desired OpenFlow rule(s) generated
- Challenges
  - Logical flow match format is flexible (unstructured)
  - Expression parsing is complex
    - Initial string parse -> annotate with symbol table -> simplify -> normalize -> generate OpenFlow matches
  - With v.s. Without conjunction
  - Shared conjunction flows between logical flows

NVIDIA

Incremental Processing

v.s. Recompute

Incremental Processing v.s. Recompute

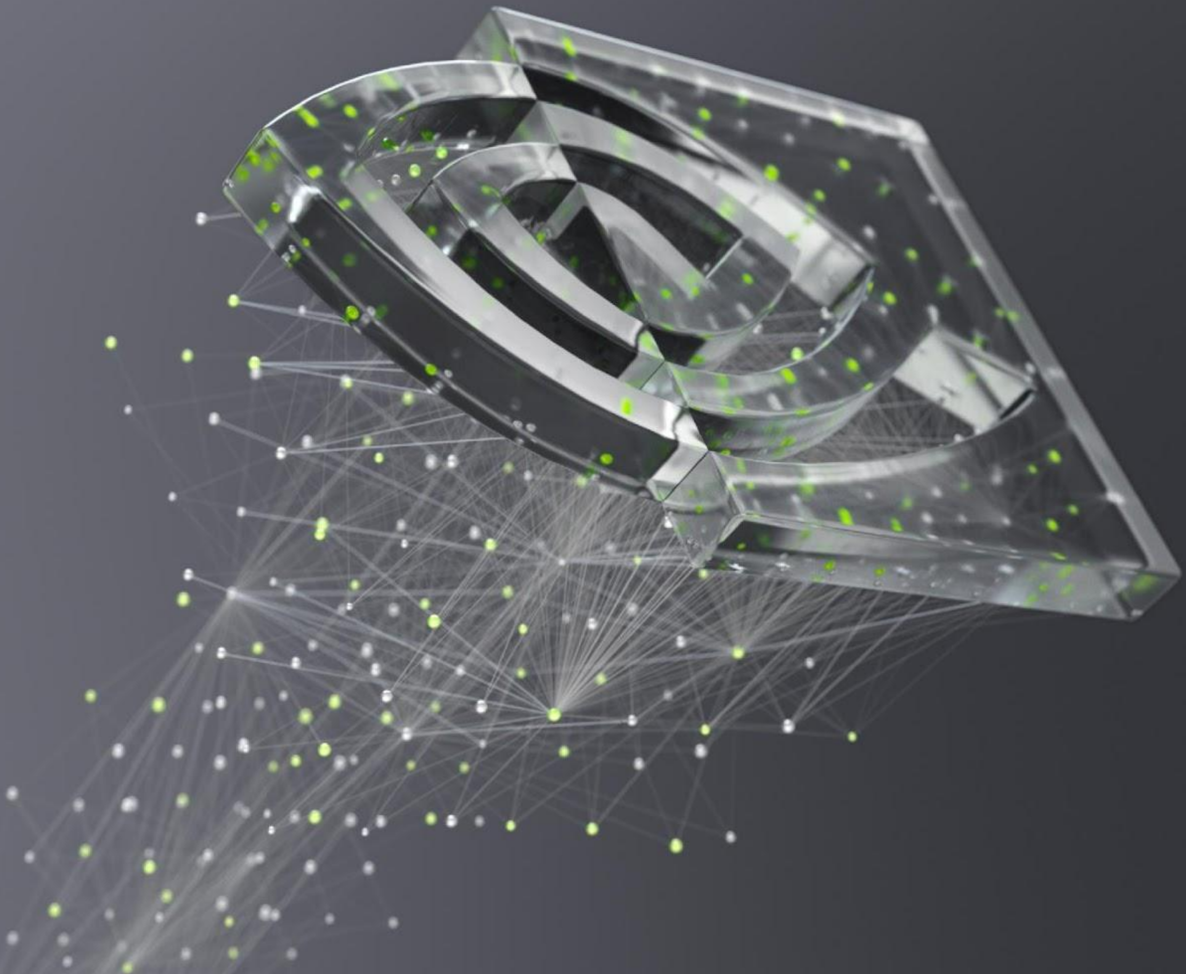
	Incremental Processing	Recompute
Latency - small change		
Latency - medium change (e.g. ~50% of the total data)	?	?
Latency - big change (e.g. ~90% of the total data)		
Throughput (req/s) - batch processing *		

* Keep pushing changes to the system without waiting for completion of earlier changes, until a large batch of changes has been pushed.

Incremental Processing

Some thoughts

- Great for latency sensitive system with small changes
  - Not necessarily good for systems that tolerates high latency but requires high throughput with batch jobs
    - E.g. “Must finish 10k jobs within 1 minute.”
- The efficiency of a single change processing in I-P is critical for throughput when change rate is high
- It is valuable to have the capability to fall-back to recompute for very big changes
  - Examples:
    - Flow computing: when most part of the input (logical topology) has changed
    - Flow installation: when tracked flow-changes are close to the total number of flows
- Rather doing less than doing it wrong



NVIDIA