Multi-tenant Inter-DC tunneling with OVN

Han Zhou (hzhou8@ebay.com)
OVSCON, 2019
Agenda

- Use case
- Logical topology
- Physical datapath
- Control plane implementation
- Gateway HA
- Gateway load balancing - OVN ECMP
Multiple OVN Deployments

- Single control plane?
  - Scale limit
  - Single point of failure

- Multiple Availability Zones (AZ)
  - Each AZ has an independent OVN deployment
  - How to interconnect between AZs?
Traditional Solutions

- OVN gateway exit to provider network + Firewall + Route/VPN
OVN Interconnection (new feature)

- OVN native solution - no external firewall/VPN configurations.
- Routed through transit logic switches.
- Reuse existed tunneling mechanism.
Logical Topology - Multi-tenancy
Physical Datapath

- LRP1@TS and LRP2@TS are **chassis-redirect** ports located on GW nodes.
Control Plane

- Global DBs
  - IC-NorthBound
    - Transit Switches
  - IC-SouthBound
    - Gateways & encaps
    - Port-bindings & tunnel keys
    - TS tunnel keys
    - Routes

- Interconnection Controller (ovn-ic)
  - Generate globally unique tunnel keys
  - Exchange data between AZ and global DBs

OVSDB (RFC 7047)
AZ Registration

- User configure a unique name in NB
- ovn-ic register to IC-SB
Gateway Sync

- User specify a chassis as interconnection gateway:
  - # ovs-vsctl set open_vswitch . external_ids:ovn-is-interconn=true
- ovn-controller sync the chassis to SB with is_interconn=true
- Local ovn-ic sync the chassis and its encaps to IC-SB
- Remote ovn-ic sync the chassis to remote SB with is_remote=true
Transit Switch Sync

- User creates a Transit Switch in IC-NB (# ovn-ic-nbctl ts-add <name>)
- `ovn-ic` in any AZ create a datapath and tunnel key in IC-SB
  - Avoid race by IC-SB transaction
  - Separate tunnel key space for global datapaths:
    - highest $2^{16}$ (65536) of the $2^{24}$ space.
- `ovn-ic` sync data to local NB
- `ovn-northd` sync to SB with specified tunnel key
Port-binding Sync

- User creates a LRP in NB connecting LR to TS
- Local ovn-ic:
  - Generate tunnel key, create port-binding to IC-SB
  - Sync back tunnel key to NB (updated to SB by northd)
- Remote ovn-ic:
  - Create port in NB, synced by remote northd to SB

- User specify gateway-chassis for the LRP
- ovn-controller updates port-binding to SB
- Local ovn-ic sync port-binding's chassis to IC-SB
- Remote ovn-ic sync port-binding's chassis to SB
Route Advertisement

- DON’T: manual config - tedious and error prone
- Interconnection route advertisement
  - Edge router: routers connected to transit switches
  - ovn-ic populate local routes to IC-SB for each edge router
    - Directly connected subnets
    - Static routes
    - Exclude internal transit routes and learned routes
Gateway HA

- Reuse existing Gateway HA mechanism
- GW failure detected by BFD
- LRP Port-binding updated in SB
- Local ovn-ic sync the port-binding update to IC-SB
- Remote ovn-ic sync the port-binding update to SB
- Remote GW OVS flow changed by ovn-controller on GW
Gateway HA (before failover)
Gateway HA (after failover)
Gateway Load-balancing - Problem

- Problem of unbalanced gateway load

```
10.2.1.0/24 => 169.254.0.1
10.2.2.0/24 => 169.254.0.2
```
Gateway Load-balancing with ECMP

- Solution: 2-tiers of routers, with ECMP routes (new feature).

![Diagram of Gateway Load-balancing with ECMP]

- AZ1
  - LR1
  - LR2
  - 0.0.0.0 => 169.254.128.1
  - 0.0.0.0 => 169.254.128.2

- AZ2
  - LR1
  - LR2
  - 10.2.0.0/16

- TS
  - GW1
    - LRP1
    - 169.254.0.1
  - GW2
    - LRP2
    - 169.254.0.2

- 10.2.0.0/16 => 169.254.0.1
- 10.2.0.0/16 => 169.254.0.2
- 0.0.0.0 => 169.254.128.1
- 0.0.0.0 => 169.254.128.2
OVN ECMP Routing (new feature)

- A new logical flow action “select”
  - Syntax: “select(<result field>, <id1>[=<weight>], <id2>[=<weight>], …)”
  - Example: select(reg0[0..15], 1, 2, 3)
  - Implemented using OpenFlow action “group”
  - Select an “ID” based on 5-tuple hash and save in the result field.

- A new stage IP_ROUTING_ECMP in Logical Router ingress pipeline
  - Example (simplified)
    - Static routes in NB:
      - Prefix = 10.2.0.0/16, Nexthop = 169.254.0.1
      - Prefix = 10.2.0.0/16, Nexthop = 169.254.0.2
      - Prefix = 192.168.1.0/24, Nexthop = 10.0.0.1
      - Prefix = 192.168.1.0/24, Nexthop = 10.0.0.2
      - Prefix = 192.168.1.0/24, Nexthop = 10.0.0.3
    - In IP_ROUTING stage, assign group id and select nexthop id:
      - ip4.dst == 10.2.0.0/16, reg8[0..15] = 1; select(reg8[16..31], 1, 2) /* ECMP route */
      - ip4.dst == 192.168.1.0/24, reg8[0..15] = 2; select(reg8[16..31], 1, 2, 3) /* ECMP route */
    - In IP_ROUTING_ECMP stage, match group id and nexthop id:
      - reg8[0..15] == 1 & reg8[16..31] == 1, reg0 = 169.254.0.1; eth.src = ...; outport = ...
      - reg8[0..15] == 1 & reg8[16..31] == 2, reg0 = 169.254.0.2; eth.src = ...; outport = ...
      - reg8[0..15] == 2 & reg8[16..31] == 1, reg0 = 10.0.0.1; eth.src = ...; outport = ...
      - reg8[0..15] == 2 & reg8[16..31] == 2, reg0 = 10.0.0.2; eth.src = ...; outport = ...
      - reg8[0..15] == 2 & reg8[16..31] == 3, reg0 = 10.0.0.3; eth.src = ...; outport = ...
      - reg8[0..15] == 0, next /* For other regular routes */
Gateway Failover with ECMP

• Before failover
Gateway Failover with ECMP

- After failover - hash buckets do NOT change - zero impact to existing flows
Q & A

Thank you!