MidoNet
and the
Open vSwitch Datapath

Duarte Nunes
duarte@midokura.com
@duarte_nunes
Agenda

● MidoNet
  ○ Architecture
  ○ Agent

● Distributed state
  ○ Device state
  ○ Flow state

● Relationship with datapath
  ○ Netlink library
  ○ Performance
  ○ Flow bookkeeping
MidoNet transform this...
MidoNet

- Fully distributed architecture
- All traffic processed at the edges, i.e., where it ingresses the physical network
  - virtual devices become distributed
  - a packet can traverse a particular virtual device at any host in the cloud
  - distributed virtual bridges, routers, NATs, FWs, LBs, etc.
- No SPOF
- No middle boxes
- Horizontally scalable L2 and L3 Gateways
MidoNet Hosts

Gateway 1

MidoNet Agent (Java Daemon)

Internet/WAN

IP Fabric

Compute 1

VM

VM

VM

VM

OVS kmod

VXLAN Tunnel Port

eth0

port5, tap12345

VM

VM

VM

VM

MidoNet Agent (Java Daemon)

Internet/WAN

IP Fabric

Gateway 1

MidoNet Agent (Java Daemon)

VXLAN Tunnel Port

eth0

eth1

port1

port2

port3, veth0

veth1

Quagga, bgpd

OVS kmod

IP1

eth0

IP3
Flow computation and tunneling

- Flows are computed at the ingress host
  - by simulating a packet’s path through the virtual topology
  - without fetching any information off-box (~99% of the time)

- Just-in-time flow computation

- If the egress port is on a different host, then the packet is tunneled
  - the tunnel key encodes the egress port
  - no computation is needed at the egress
Inside the Agent

Flow table
Flow state
ARP broker
CPU

Simulation

Backchannel
Virtual Topology

Upcall
Output

User
Kernel

Datapath
queue userspace packet
packet execution, flow create and delete

User
Kernel

queue userspace packet
packet execution, flow create and delete
Device state

- ZooKeeper serves the virtual network topology
  - reliable subscription to topology changes
- Agents fetch, cache, and “watch” virtual devices on-demand to process packets
- Packets naturally traverse the same virtual device at *different* hosts
- This affects device state:
  - a virtual bridge learns a MAC-port mapping a host and needs to read it in other hosts
  - a virtual router emits an ARP request out of one host and receives the reply on another host
- Store device state tables (ARP, MAC-learning, routes) in ZooKeeper
  - interested agents subscribe to tables to get updates
  - the owner of an entry manages its lifecycle
  - use ZK Ephemeral nodes so entries go away if a host fails
ARP Table
ARP Table
ARP Table
ARP Table

VM

ARP Table

ZK notification

ARP reply handled locally and written to ZK

VM

IP Fabric

VM

VM
ARP Table
Flow state

- Per-flow L4 state, e.g. connection tracking or NAT
- Forward and return flows are typically handled by different hosts
  - thus, they need to share state
- Tricky to leverage megaflows
  - agent needs to generate this state, replicate it
Sharing state - Peer-to-peer handoff

1. New flow arrives
2. Check or create local state
3. Replicate the flow state to interested set
4. Tunnel the packet
5. Deliver the packet
Sharing state - Peer-to-peer handoff

1. Return flow arrives
2. Lookup local state
3. Tunnel the packet
4. Deliver the packet

Node 1

Node 2

Node 3 (possible asym. ret. path)

Node 4 (possible asym. fwd. path)
Sharing state - Peer-to-peer handoff

1. Exiting flow arrives at different node

2. Lookup local state

3. Tunnel the packet

Node 1

Node 2

Node 4 (possible asym. fwd. path)

Node 3 (possible asym. ret. path)

4. Deliver the packet
Netlink requests

- JVM netlink library, implements rtnetlink and odp
- Replies and notifications are modeled as asynchronous, observable streams
- A simulation entails packet execution, and flow create and delete operations
- Flow create
  - optimistic, not ack’ed or echo’ed
  - errors are ignored
  - may result in duplicates
- Flow delete
  - echo’d to get stats
NetlinkRequestBroker

Pre-allocated buffer split into fixed size chunks

Publisher

Writer

NL Socket

Reader

Array of Observers indexed by seq
Performance

- **Packet Execution**
  - $2.747 \pm 0.241$ us/op

- **Flow creation**
  - $5.476 \pm 0.356$ us/op

- **Concurrent flow creation (2 threads)**
  - $24.960 \pm 2.138$ us/op
  - *ouch*

- **Flow creation + deletion**
  - $11.873 \pm 1.321$ us/op
  - 88k ops/s

- **Flow creation + deletion through broker**
  - $12.380 \pm 1.449$ us/op

---

**CPU:** Intel(R) Xeon(R) @ 2.40GHz

- **Number of CPUs:** 16
- **Threads per core:** 2
- **Cores per socket:** 4
- **Sockets:** 2
- **NUMA node(s):** 2
- **L1 cache:** 128K
- **L2 cache:** 1MB
- **L3 cache:** 12MB
- **System memory:** 24GB
Flow bookkeeping

- All flows have a hard time expiration
  - also important for the distributed flow state mechanism
- No idle expiration
  - flow gets would be too costly
- Invalidations
  - all flows are indexed by the set of tags applied during their simulation
  - e.g., the ID of each traversed device is a tag
  - this allows flows to be removed upon virtual topology changes
Some tricks

- Megaflow bypass by setting a bit in the tunnel key
  - Force packet into userspace for flow tracing
- Double encapsulation for overlay tunnels
Conntrack?

- **Synchronize conntrack state**
  - How? How often?
  - Will the state be available to the egress host when simulating the return flow?

- **Confine flow state to the compute host**
  - Same host must process forward and return flows
  - This means doing a simulation in the gateway and re-doing it in the compute
  - More load on computes
  - SPoF
Questions?
Thank you!