AWIC Spring 2014

“A Walk in the Clouds”

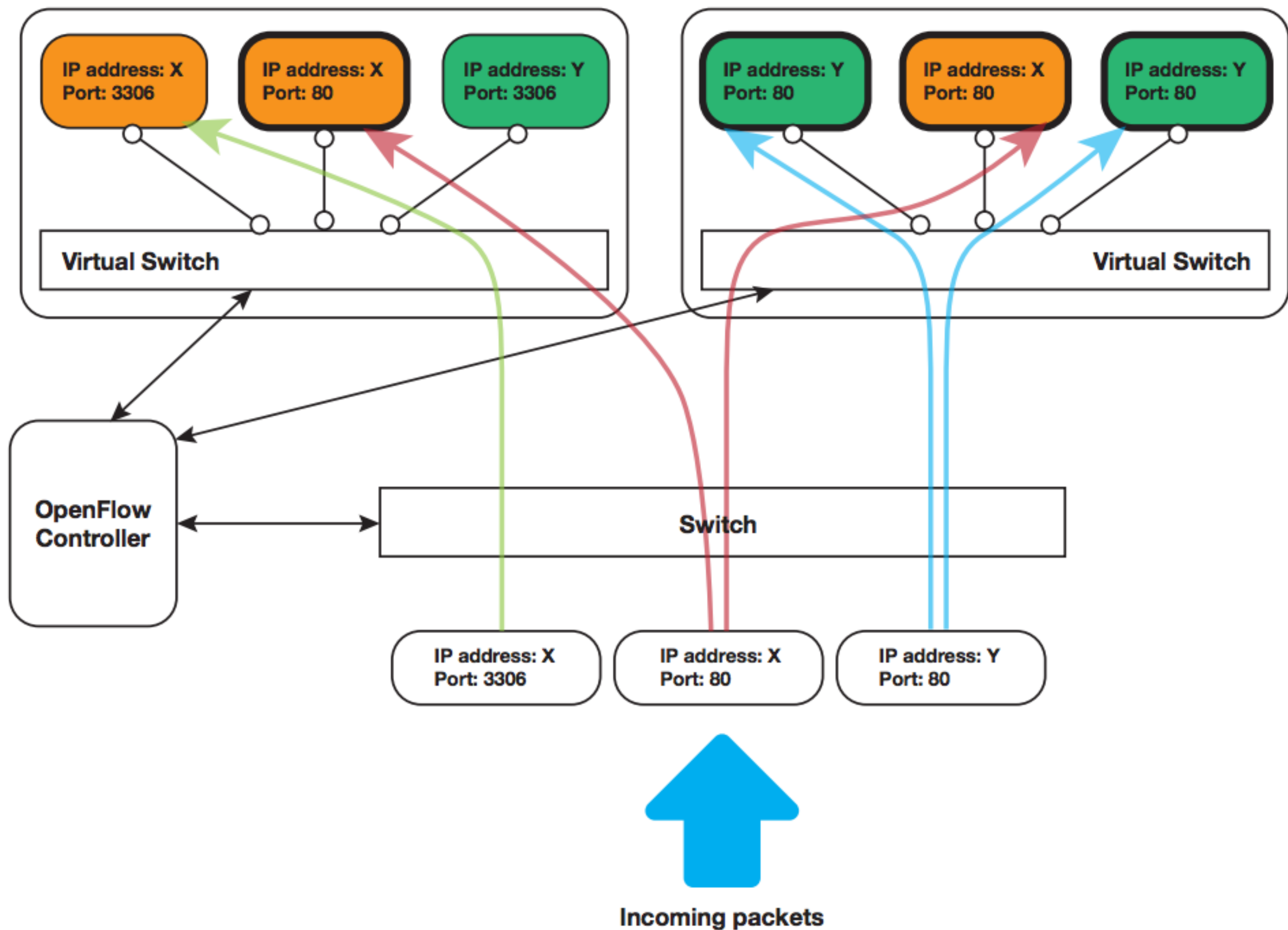
Michael Ben-Ami - Graduate Researcher
Kyung-Hwa Kim - Mentor
Jae Woo Lee - Mentor

Background Information

- Started as undergraduate project from the previous semester (Fall 2013)
    - Etan Zapinsky and Alex Merkulov
- Examining container-based clouds
- M-to-N mapping of network addresses and containers
- Container Management System

Original Goals

- Simplify the interface between cloud PaaS providers and customers (developers).
  - Reduce dependency on provider-generated static configuration files or environment variables
  - Allow applications to use their standard port numbers
- Simplify policy configuration in provider infrastructure
  - load balancers, firewalls, NAT gateways, etc.


IP address: X
Port: 3306
IP address: X
Port: 80
IP address: Y
Port: 3306
IP address: Y
Port: 80
IP address: X
Port: 80
IP address: Y
Port: 80
Virtual Switch
Virtual Switch
OpenFlow
Controller
Switch
IP address: X
Port: 3306
IP address: X
Port: 80
IP address: Y
Port: 80
Incoming packets

Original Implementation

- Mapping: (public_ip, port) ⇒ private_ip
  - Single VM with multiple Docker containers
    - Containers attached to Open vSwitch
  - Pyretic OpenFlow controller does translation
  - Containers can't act as clients, but can listen and respond as servers to the outside world
- Container Lifecycle Management
  - HTTP API, CLI utility

My Tasks

- Implement forwarding based on mapping: (public_ip, port) ⇒ destination MAC
    - No Layer 3 NAT
- Implement container-originated communication
- Expand from single to multiple VMs

My Tasks (continued)

- Adapt to run on top of public cloud IaaS
- Contribute to HotCloud workshop paper
- Focus more on network, less on container lifecycle management

New Mapping

- Implemented as Python dictionary at controller

```
self.arpTable = {}
```

- Keys are tuple (public_ip, port)
- Values are correct destination MAC

```
self.arpTable[(IPAddr('172.16.56.101'), 8000)] = EthAddr('32:af:a4:6d:58:db')
self.arpTable[(IPAddr('172.16.56.101'), 9000)] = EthAddr('86:23:1e:40:95:26')
```

New Mapping (continued)

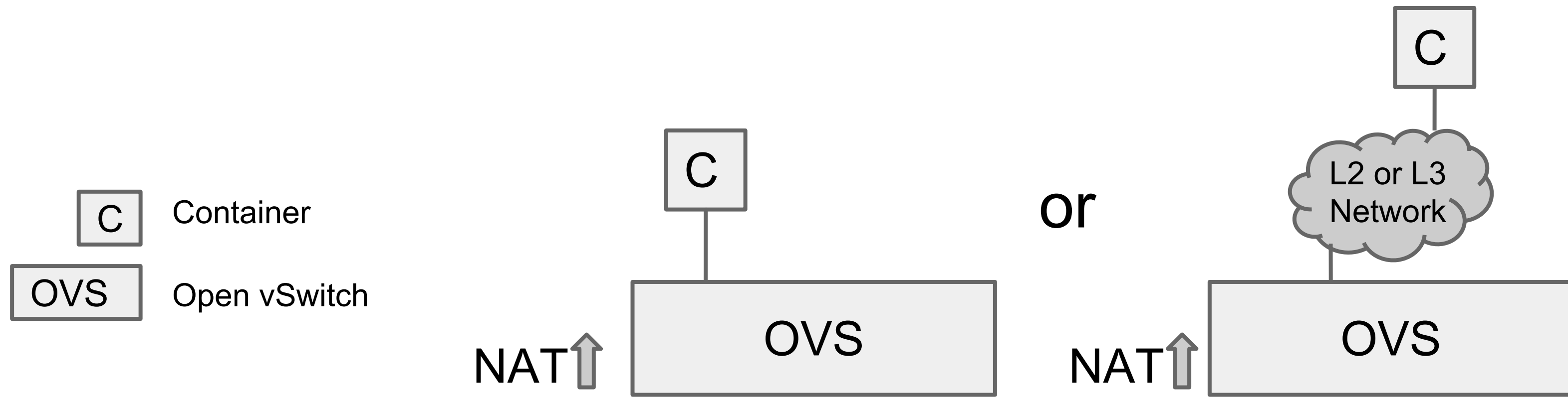
- Standard CAM table for layer 2 forwarding
  - Map MAC address to outgoing interface by learning
  - Checked after MAC address rewrite
  - Updated on received packet at the controller

```
# update the layer 2 learning table
self.macTable[packet.src] = event.port
```

  - Consulted before packet egress

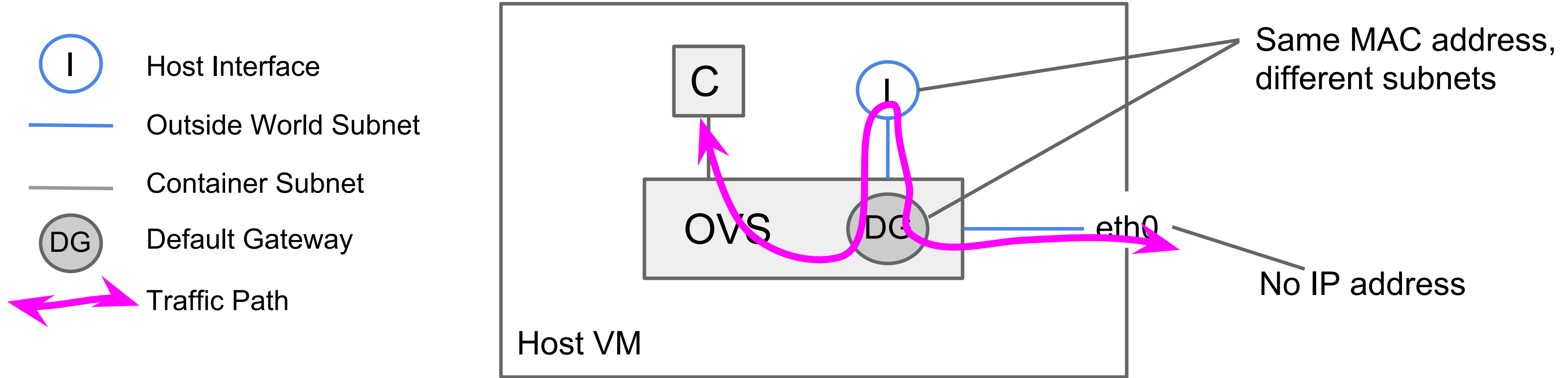
```
if dst_mac in self.macTable:
    port = self.macTable[dst_mac]
else:
    port = of.OFPP_FLOOD
```

Design Evolution - Phase 0



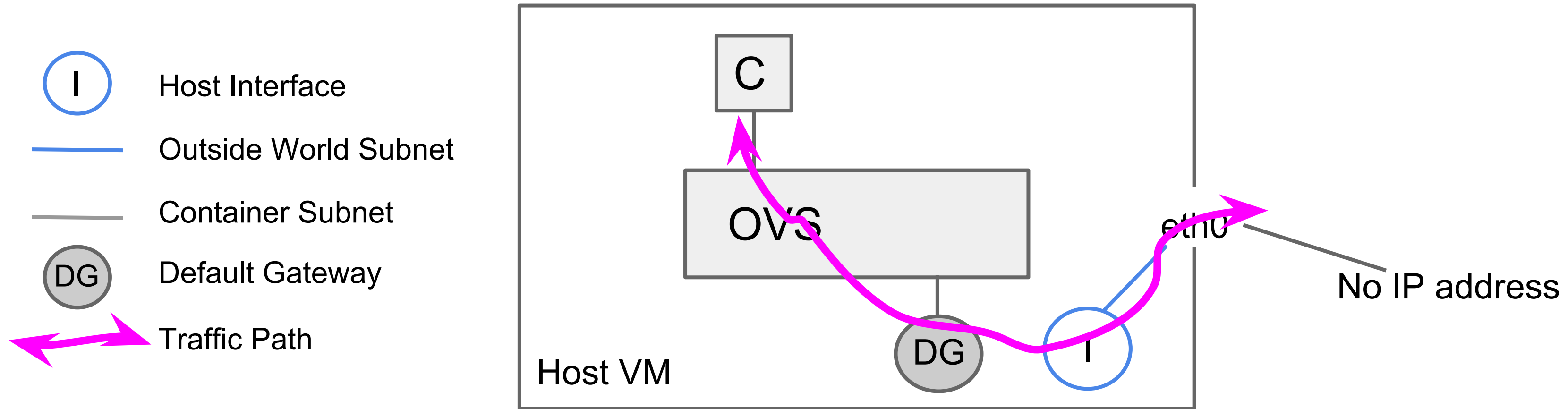
- Open vSwitch (OVS) is more like a layer 3 router or multilayer switch
- Performs special layer 3 rewrite (NAT)
- Performs normal layer 2 rewrite for next hop or final destination
- Default Gateway for containers provided by normal network infrastructure
  - Pyretic default controller code gives normal routing capabilities

Design Evolution - Phase 1



- Open vSwitch (OVS) is more like a layer 2 switch
  - Containers are in same broadcast domain as outside world
- Performs no layer 3 rewrite
- Performs special layer 2 rewrite for final destination
- Default Gateway for containers provided by special controller code
  - POX controller programmed with “fake” default gateway and dummy ARP responses on behalf of “fake” default gateway

Design Evolution - Phase 2



- Open vSwitch (OVS) is more like a layer 2 switch
    - Containers are in different broadcast domain than outside world
- Performs no layer 3 rewrite
- Performs special layer 2 rewrite for final destination
- Default Gateway for containers provided by container host machine (VM)
    - “ip_forward” turned on in host
    - OVS gives dummy ARP responses to gateway on behalf of containers

Container-originated traffic

- Mapping table only contains static entries (so far)
  - What happens when container wants to act as a client using random source port?
    - e.g. software updates, container-to-container traffic
  - If there is no entry in the table for the random source port, traffic will be dropped!

Container-originated traffic (contd)

- Solution: dynamically update mapping table on received packet at controller

```
self.arpTable[(src_ip, packet.payload.next.srcport)] = packet.src
```

Source IP — Source TCP/UDP Port — Source MAC

Container-originated traffic (contd)

- How do two containers with the same IP address communicate?
  - Must force traffic out of the container even if destination is self address
  - Manipulate policy routing database in container
    - ```
      root@009f0b236cce:/# ip rule show
      0:      from all lookup local
      1:      from all iif eth0 lookup local_copy
      32766:  from all lookup main
      32767:  from all lookup default
      ```

Only use “local” routing table when packet originates from outside (eth0)
      ```

Mutliple VMs, Multiple vSwitches

- Each VM is running one instance of OVS
  - Containers are directly attached to OVS
- All OVS instances talk to a centralized OpenFlow controller (POX)
- Must add switch identifier (dpid) to centralized CAM table

```
# update the layer 2 learning table
self.macTable[(event.dpid, packet.src)] = event.port
```

Switch ID

Source MAC

Ingress Interface

Multiple VMs (continued)

- What is the “fabric” that connects the “access layer” OVS instances?
  - Must be a **layer 2 network**, to preserve destination MAC address in transit (after rewrite)
  - Ideally all hops should be **SDN-enabled** and share the same OpenFlow control plane
  - Traditional layer 2 learning switches may work, but some may drop traffic because of advanced security features (e.g. ARP inspection)

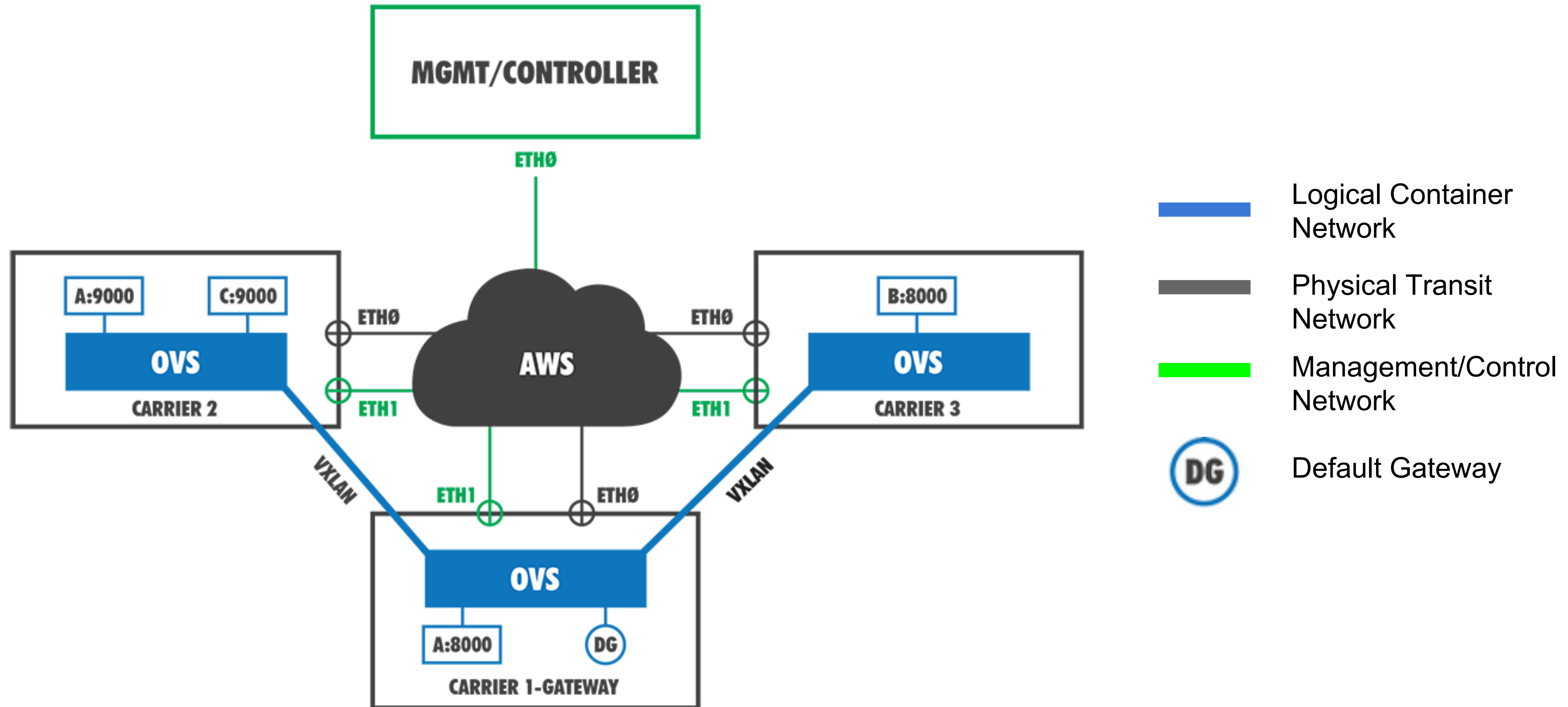
Multiple VMs (continued)

- Solution: VXLAN Tunnels
  - Layer 2 encapsulation preserves rewritten MAC address
  - Intermediate network need not be SDN-enabled
    - More suitable for deploying on top of public Iaas

Deploying on Public IaaS

- Amazon AWS was the obvious choice
- Amazon-supplied public IP addresses for prototype, but not optimal
  - Limited number of addresses
  - Forced NAT to private address
  - No IPv6
- VXLAN Tunnels across multiple VMs (EC2 instances)

Amazon Deployment



Amazon Demo

Paper Contributions

- General editing
  - Kyung-Hwa did the bulk of initial writing
- Wrote section “Provider-independent architectures”
  - Blueprint for running on top of Public IaaS like Amazon

Time also spent on…

- Troubleshooting bugs
  - Version dependencies
  - 3rd party patches
  - Side-effects of cloning VMs on Amazon
- Learning Python and POX controller
  - I chose POX over Pyretic
    - Pyretic is for high-level policy
    - POX is for direct interaction with OpenFlow

Future Work

- Flesh out use cases
  - More generally contributions of SDN
  - Target specific applications
- Integrate container lifecycle management
  - Integrate with OpenStack
- Implement on public IaaS with distributed highly-available gateways