Lecture 7 Software Defined Network - Control Plane

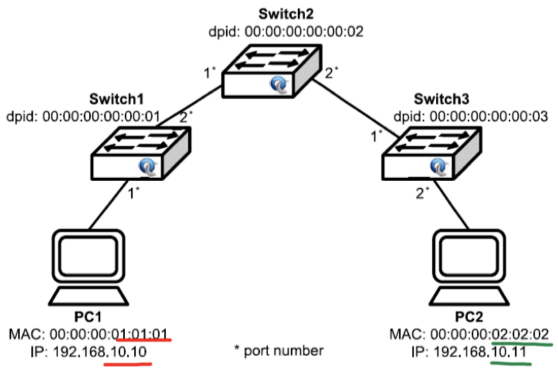
Programing Levels

- Level 1: South-Bound Interface
  - Program switches directly through OpenFlow
- Level 2: SDN Controller
  - Program with general-purpose language like C, Java, Python
- Level 3: Network Programming Languages
  - Program with domain-specific languages for networks
  - A Domain-Specific Language (DSL) is a programming language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain

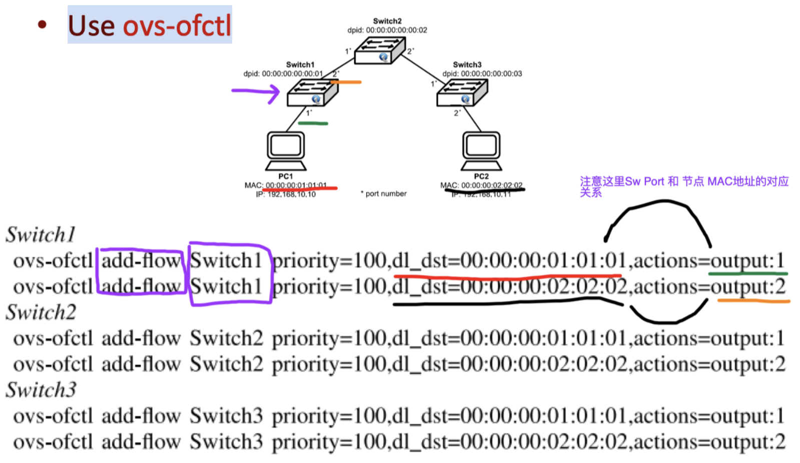
Level 1: South-Bound Interface

Task:

> make PC1 and PC2 reachable on layer 2



Use ovs-ofctl



- Use ovs-ofctl

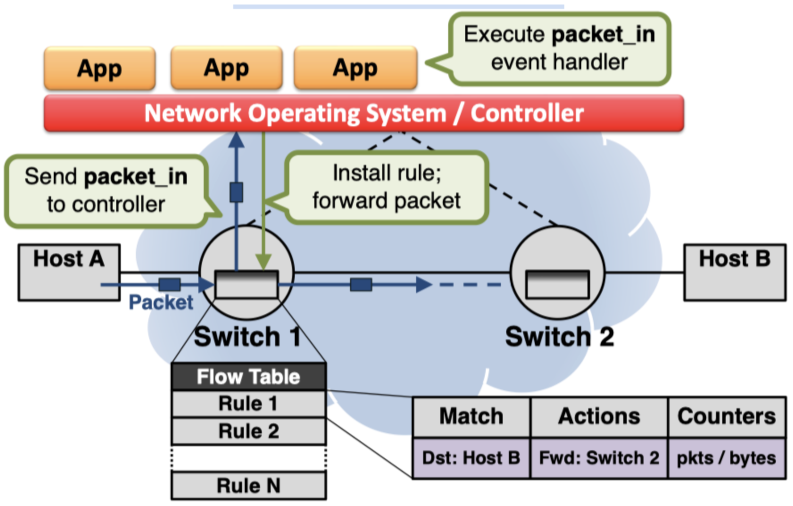
Switch1
```
ovs-ofctl add-flow Switch1 priority=100,dl_dst=00:00:00:01:01:01,actions=output:1
ovs-ofctl add-flow Switch1 priority=100,dl_dst=00:00:00:02:02:02,actions=output:2
```
Switch2
```
ovs-ofctl add-flow Switch2 priority=100,dl_dst=00:00:00:01:01:01,actions=output:1
ovs-ofctl add-flow Switch2 priority=100,dl_dst=00:00:00:02:02:02,actions=output:2
```
Switch3
```
ovs-ofctl add-flow Switch3 priority=100,dl_dst=00:00:00:01:01:01,actions=output:1
ovs-ofctl add-flow Switch3 priority=100,dl_dst=00:00:00:02:02:02,actions=output:2
```

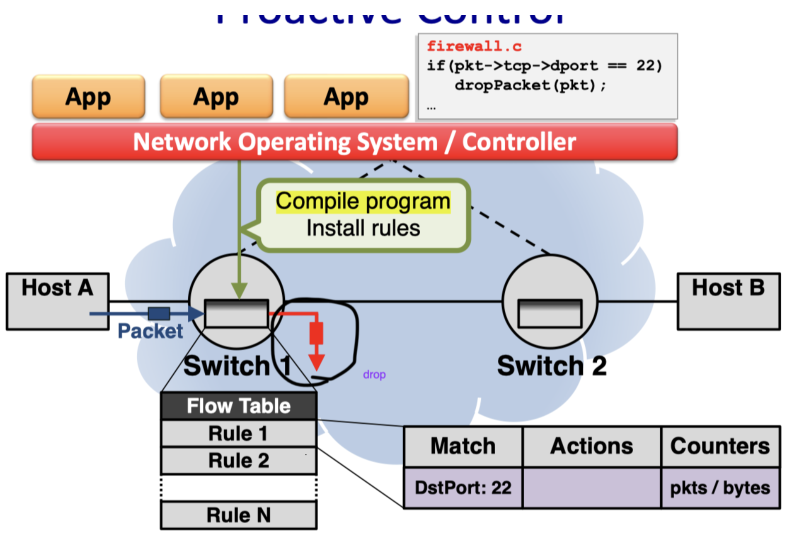
Level 2: Controller

Modes

- Reactive vs. Proactive
- Centralized vs. Distributed

Reactive Control vs. Proactive





```
firewall.c
if(pkt->tcp->dport == 22)
   dropPacket(pkt);
…
```

Reactive

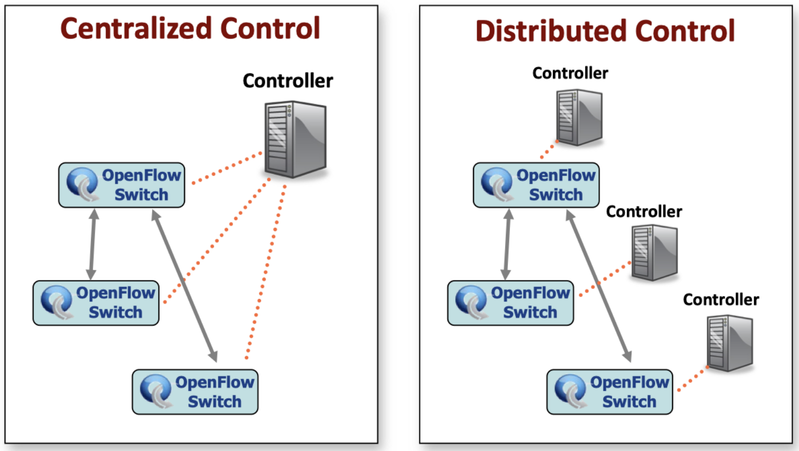
- First packet of flow triggers controller to insert flow entries
- Efficient use of flow table
- Every flow incurs an additional flow setup time
- If control connection lost, switch has limited utility

Proactive

- Controller pre-populates flow table in switch (预处理)
- Requires aggregated rules
- Zero additional flow setup time

- Loss of control connection does not disrupt traffic

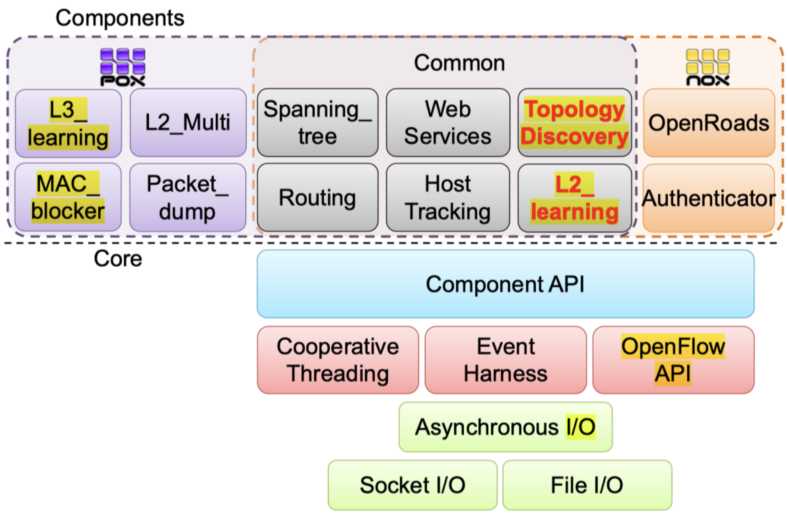
Centralized vs. Distributed Control



分布式系统的意义是将“压力”分担给多个controller，以防某一个controller出问题导致系统整体出问题

分布式系统并不意味着“隔离”，事实上，每个“区域级”的Controller包含“所有区域整体”的信息，因此不会存在“区域”之间的壁垒

NOX / POX Architecture

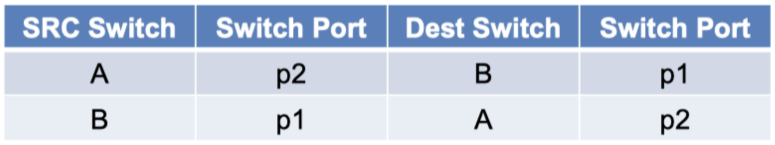


Topology Discovery

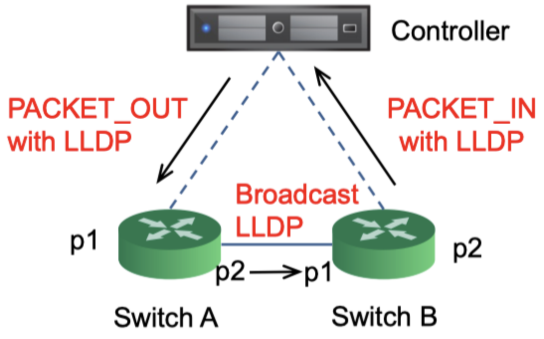
How to know the correct "path (topo)"? ⇒ use LLDP

def: Link Layer Discovery Protocol (LLDP)

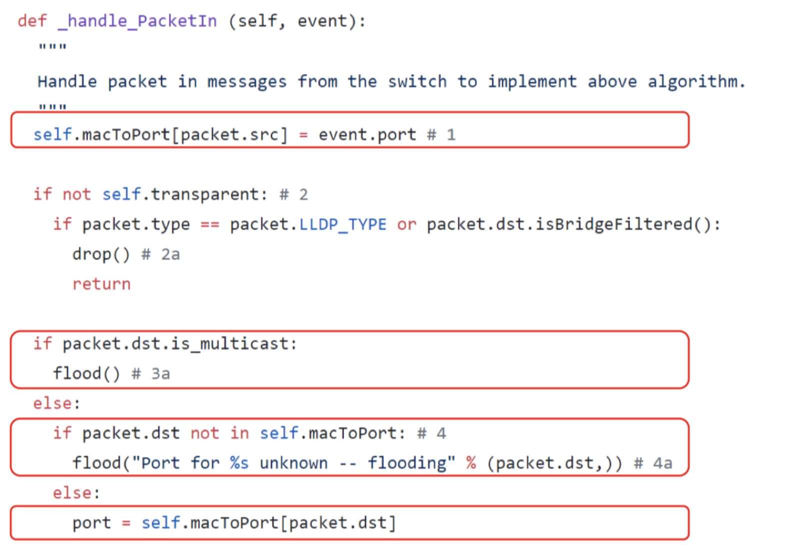
> A vendor neutral link layer protocol in the Internet Protocol Suite used by network devices for advertising(公布) their identity, capabilities and neighbors on an IEEE 802 LAN.



SRC Switch	Switch Port	Dest Switch	Switch Port
A	p2	B	p1
B	p1	A	p2



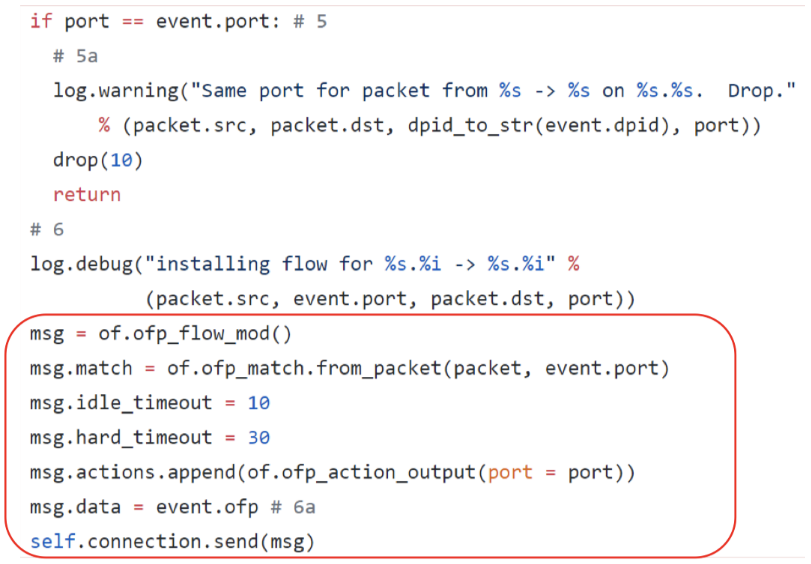
Learning Switch



```
def _handle_PacketIn (self, event):
  """
  Handle packet in messages from the switch to implement above algorithm.
  """
  self.macToPort[packet.src] = event.port # 1

  if not self.transparent: # 2
    if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
      drop() # 2a
      return

  if packet.dst.is_multicast:
    flood() # 3a
  else:
    if packet.dst not in self.macToPort: # 4
      flood("Port for %s unknown -- flooding" % (packet.dst,)) # 4a
    else:
      port = self.macToPort[packet.dst]
```

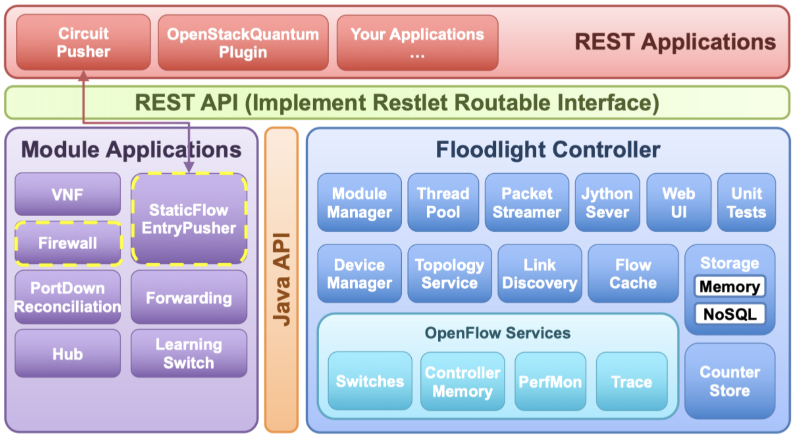


```
if port == event.port: # 5
  # 5a
  log.warning("Same port for packet from %s -> %s on %s.%s.  Drop."
      % (packet.src, packet.dst, dpid_to_str(event.dpid), port))
  drop(10)
  return
# 6
log.debug("installing flow for %s.%i -> %s.%i" %
          (packet.src, event.port, packet.dst, port))
msg = of.ofp_flow_mod()
msg.match = of.ofp_match.from_packet(packet, event.port)
msg.idle_timeout = 10
msg.hard_timeout = 30
msg.actions.append(of.ofp_action_output(port = port))
msg.data = event.ofp # 6a
self.connection.send(msg)
```

Floodlight

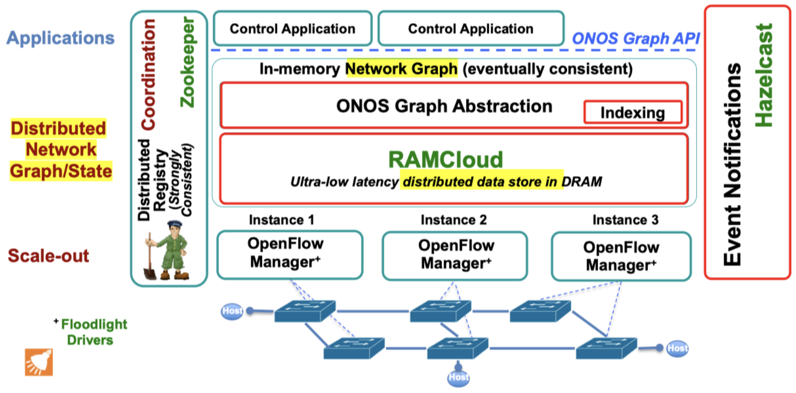
- An open, free, OpenFlow controller in Java

- Slowly supporting OpenFlow v1.3



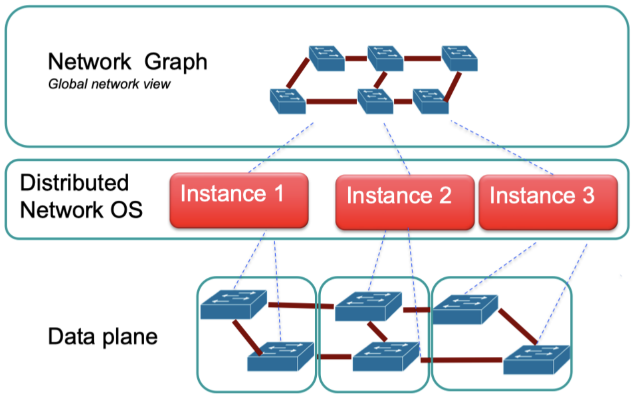
ONOS: Open Network Operating System

Architecture



Scale - Out

- An instance is responsible for maintaining a part of network graph.
- Control capacity can grow with network size or application need

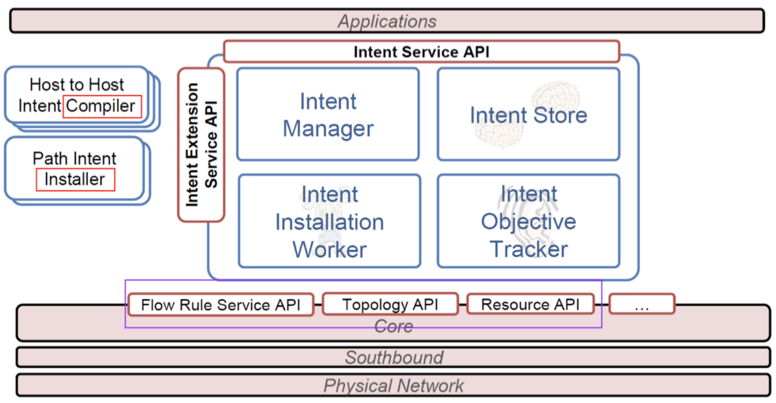


An instance is responsible for maintaining a part of network graph

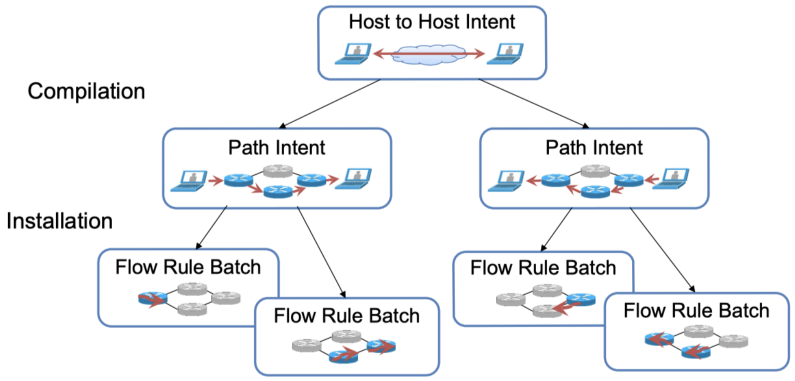
Control capacity can grow with network size or application need

Intent Framework

Translates intents into device instructions



- Compiler: produce more specific Intents given the environment
- Installer: transform Intents into device commands



OpenDaylight (ODL)

- Open-source project hosted by the Linux Foundation
  - Language: Java
  - License: Eclipse Public License 1.0
- Releases are named with chemical elements
  - Stable release: Chlorine (17) October 2022

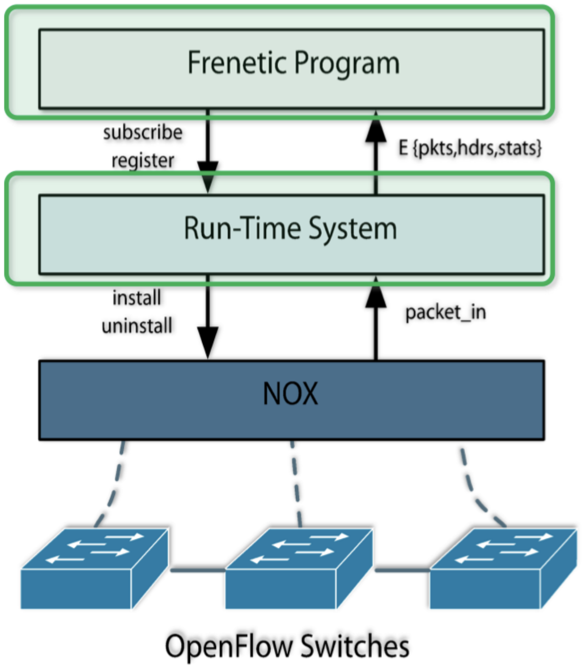
Ryu

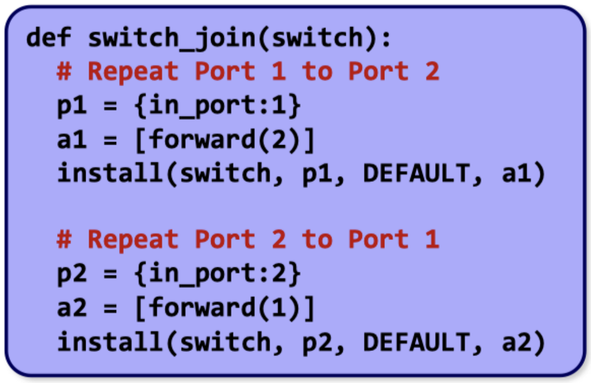
- Implemented with Python
- Open source, Apache 2.0 license
- Support various protocols for managing network devices, such as OpenFlow, Netconf, OF-config, etc.
  - For OpenFlow, Ryu supports fully 1.0, 1.2, 1.3, 1.4, 1.5 and Nicira Extensions

Network Programming Languages

Frenetic

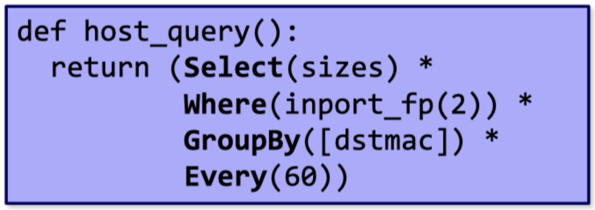
- High-level language
  - On top of NOX
  - Query (询问式) language
  - Composition of forwarding policies
- Program snippet: simple repeater
  - When a switch joins the network, install two forwarding rules.
- Query language for traffic monitoring
  - Provide a declarative SQL-like query language for classifying and aggregating network traffic
- Program snippet（代码片段）: summarize the total volume of traffic arriving on physical port 2, grouped by destination host, every 60 seconds.





```
def switch_join(switch):
  # Repeat Port 1 to Port 2
  p1 = {in_port:1}
  a1 = [forward(2)]
  install(switch, p1, DEFAULT, a1)

  # Repeat Port 2 to Port 1
  p2 = {in_port:2}
  a2 = [forward(1)]
  install(switch, p2, DEFAULT, a2)
```



```
def host_query():
  return (Select(sizes) *
          Where(inport_fp(2)) *
          GroupBy([dstmac]) *
          Every(60))
```

Other

- Pyretic
- Merlin
- ……