NOX

an OpenFlow controller

Role of Controller in OpenFlow Environments

- Push forwarding logic to switches
- Give developers a high-level API to develop advanced network applications

Features provided by NOX

- Learn network topologies
- Detect hosts
- C++/Python API bindings

InCNTRE

GlobalNOC
Global Research Network Operations Center

Bundled NOX Applications

- Core Apps
  - Messenger
    - TCP/SSL server sockets for communications with other devices, such as hosts
  - SNMP
    - handles snmptrap using a Python script as trap handler through NetSNMP

InCNTRE

GlobalNOC
Global Research Network Operations Center

Bundled NOX Applications

- Network Apps
  - Discovery
    - keeps track of links between controlled switches
  - Topology
    - provides an in-memory record of all links currently up in the network
  - Authenticator
    - keeps track of the location of hosts and switches on the network
  - Routing
    - component responsible for path calculation
  - Monitoring
    - periodically queries switches for statistics

InCNTRE

GlobalNOC
Global Research Network Operations Center

Bundled NOX Applications

- Web apps
  - Webservice
    - provides the web services interface for NOX applications
  - Webserver
    - the app hosting the control interface
  - Webserviceclient

Other NOX Components

- Switch.cc
  - C++ implementation of Layer2 learning switch
- Pyswitch.py
  - Python implementation of Layer2 learning switch
- Packetdump.py
  - Prints OpenFlow control messages

InCNTRE

GlobalNOC
Global Research Network Operations Center

pyhub.py

- Modified version of pytutorial
- NOX controller runs the pyhub.py component
  - Pyhub receives an OpenFlow packet-in message with associated dpid and inport
  - Pyhub sends an OpenFlow packet-out reply to forward to ALL ports in VLAN

InCNTRE

GlobalNOC
Global Research Network Operations Center

pyhub.py code

- NOX libraries are imported
- Python component models are instantiated
- NOX keeps state of OpenFlow ports per DPID in an UP state

InCNTRE

GlobalNOC
Global Research Network Operations Center

pyhub.py code

```
class pyhub(Component):
    def __init__(self, ctxt):
        Component.__init__(self, ctxt)

    def forward_packet(self, dpid, inport, packet, buf, bufid):
        flow = extract_flow(packet)
        flow[core.IN_PORT] = inport
        actions = [[openflow.OFPAT_OUTPUT, [0, openflow.OFPP_ALL]]]
        self.install_datapath_flow(dpid, flow, 5,
                                   openflow.OFP_FLOW_PERMANENT, actions,
                                   bufid, openflow.OFP_DEFAULT_PRIORITY,
                                   inport, buf)
```

InCNTRE

GlobalNOC
Global Research Network Operations Center

pyhub.py code

```
def packet_in_callback(self, dpid, inport, reason, _len, bufid, packet):
    if not packet.parsed:
        log.msg('Ignoring incomplete packet',system='pyswitch')

    if packet.type != ethernet.LLDP_TYPE:
        self.forward_packet(dpid, inport, packet, packet.arr, bufid)

    return CONTINUE

def install(self):
    self.register_for_packet_in(self.packet_in_callback)

def getInterface(self):
    return str(pyhub)
```

InCNTRE

GlobalNOC
Global Research Network Operations Center

pyswitch.py

```
# Timeout for cached MAC entries
CACHE_TIMEOUT = 5

# --
# Given a packet, learn the source and peg to a switch/inport
# --
def do_l2_learning(dpid, inport, packet):
    global inst

    # learn MAC on incoming port
    srcaddr = packet.src.tostring()
    if ord(srcaddr[0]) & 1:
        return
    if inst.st[dpid].has_key(srcaddr):
        dst = inst.st[dpid][srcaddr]
        if dst[0] != inport:
            log.msg('MAC has moved from '+str(dst)+'to'+str(inport), system='pyswitch')
        else:
            return
    else:
        log.msg('learned MAC '+mac_to_str(packet.src)+' on %d %d'% (dpid,inport), system="pyswitch")

    # learn or update timestamp of entry
    inst.st[dpid][srcaddr] = (inport, time(), packet)

    # Replace any old entry for (switch,mac).
    mac = mac_to_int(packet.src)
```

InCNTRE

GlobalNOC
Global Research Network Operations Center

pyswitch.py

```
# --
# If we've learned the destination MAC set up a flow and
# send only out of its inport.  Else, flood.
# --
def forward_l2_packet(dpid, inport, packet, buf, bufid):
    dstaddr = packet.dst.tostring()
    if not ord(dstaddr[0]) & 1 and inst.st[dpid].has_key(dstaddr):
        prt = inst.st[dpid][dstaddr]
        if  prt[0] == inport:
            log.err('**warning** learned port = inport', system="pyswitch")
            inst.send_openflow(dpid, bufid, buf, openflow.OFPP_FLOOD, inport)
        else:
            # We know the outport, set up a flow
            log.msg('installing flow for ' + str(packet), system="pyswitch")
            flow = extract_flow(packet)
            flow[core.IN_PORT] = inport
            actions = [[openflow.OFPAT_OUTPUT, [0, prt[0]]]]
            inst.install_datapath_flow(dpid, flow, CACHE_TIMEOUT,
                                       openflow.OFP_FLOW_PERMANENT, actions,
                                       bufid, openflow.OFP_DEFAULT_PRIORITY,
                                       inport, buf)
    else:
        # haven't learned destination MAC. Flood
        inst.send_openflow(dpid, bufid, buf, openflow.OFPP_FLOOD, inport)
```

InCNTRE

GlobalNOC
Global Research Network Operations Center

Understanding NOX Interactions

- Use Wireshark with OpenFlow plugin
  - already provided git://openflow.org/openflow.git
- Start SSH session with Xwindow Forwarding
  - ssh openflow@10.101.1.5X –X
  - sudo wireshark

InCNTRE

GlobalNOC
Global Research Network Operations Center

Wireshark OpenFlow Plugin

- In **Filter:** field type **of**
- OF Hello
- OF Echo
- OF FlowMod
- OF FlowRemoved

InCNTRE

GlobalNOC
Global Research Network Operations Center