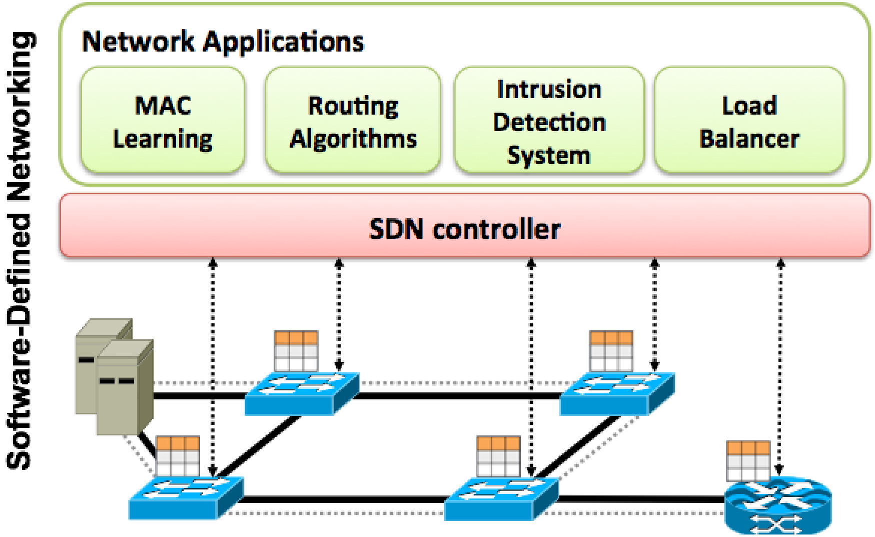
Hálózatok építése és üzemeltetése

SDN a gyakorlatban -- kontrolleralkalmazások

OF kontrollerek

- Mára számos kontroller platform alakult ki
- programozás
  - különböző szoftver környezetben
  - különböző programozási nyelveken
- különböző célok
- különböző teljesítmény



POX

- Elavult: csak OpenFlow 1.0-t támogat
  - Konkurensek: OF 1.4+, OF-config, netconf, snmp
- Fejlesztése gyakorlatilag leállt
  - Hibajavítások kivételével
- Ipari igényeket nem elégít ki

- Minimális függőségi lista (python 2.7)
- Könnyen installálható
- Szkript nyelvet használ:
  - gyors edit/(compile)/debug ciklus
  - gyors prototípus implementálás
- Rendkívül elegáns eseménykezelő keretrendszer
- Single threaded: így is gyors, de nehezebb hibázni
- Keretrendszer, amiben alkalmazások írhatók

http://www.noxrepo.org

http://github.com/noxrepo/pox/

https://openflow.stanford.edu/display/ONL/POX+Wiki

Installáció, futtatás

```
~$ git clone http://github.com/noxrepo/pox
~$ cd pox
~/pox$ git checkout eel

~/pox$ ./pox.py samples.pretty_log forwarding.l2_learning

$ mn --topo=linear --mac --controller=remote
$ mn --topo=linear --mac --controller=remote,ip=192.168.56.1
```

- Controller argumentum: az OVS switch-ek hol találják az OF kontrollert.

Pox komponensek (amik azért nem NOS alkalmazások)

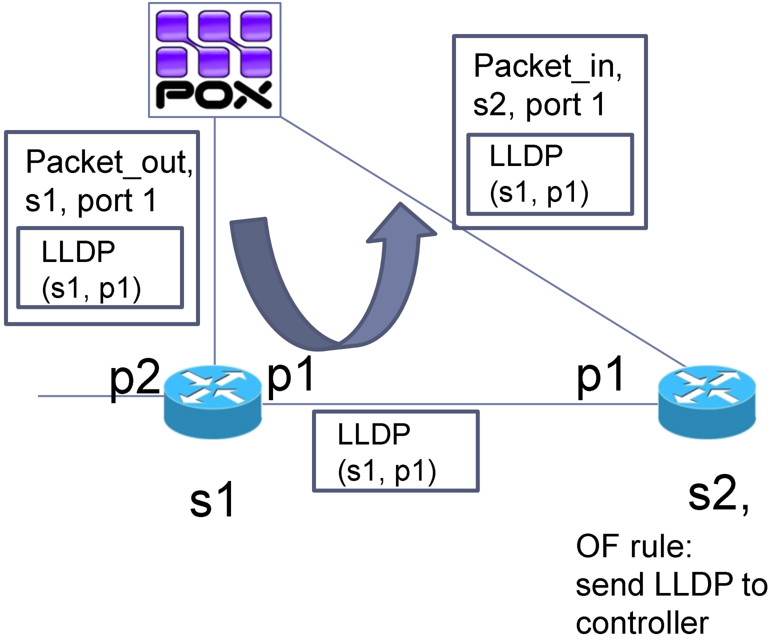
Pox komponensek

- forwarding.hub
- forwarding.l2_learning
- forwarding.l3_learning
- forwarding.topo_proactive
- Az összes komponens: find ~/pox/pox

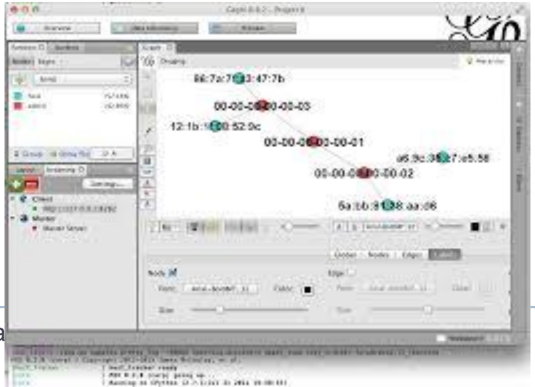
- openflow.of_01
  - --port=<X>
  - OF üzenetek küldése, fogadása, POX eseménnyé alakítása
- openflow.discovery
  - Topólogia feltérképezése LLDP üzenetekkel
- openflow.webservice
  - Északi interfész alacsonyszintű OF protokollhoz

openflow.discovery

LLDP: Link Layer Discovery Protocol
(EtherType == 0x88cc)

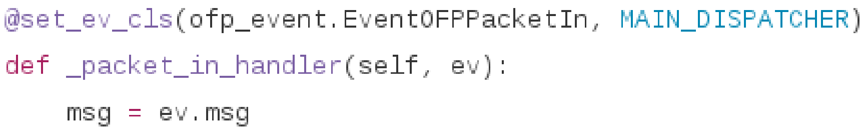


- A packet_in vétele után
  - a kontroller megtanulja, hogy van egy **s1.p1-s2.p1** link
  - Az openflow.discovery küld egy **LinkEvent** üzenetet.
- LinkEventet az kapja meg, aki feliratkozik rá, pl:
- `./pox.py openflow.discovery` **`misc.gephi_topo`** `\`
  `host_tracker forwarding.l2_learning`

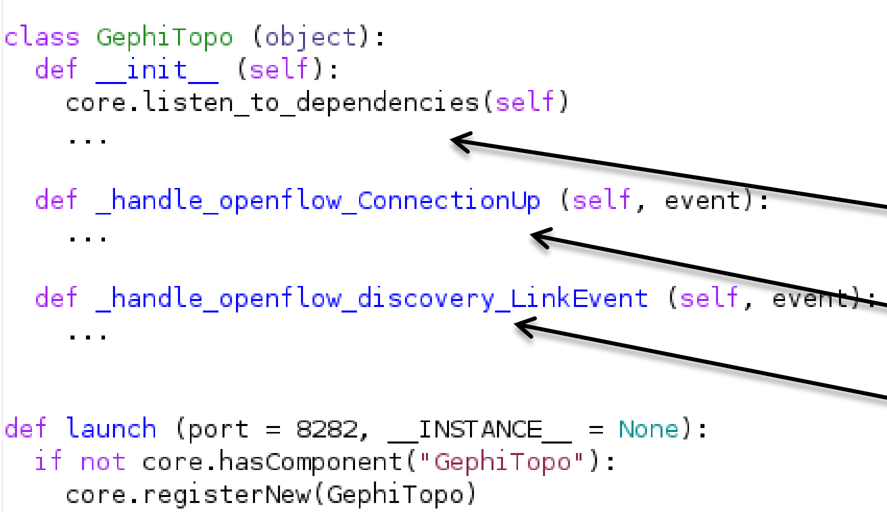


POX: eseménykezelés

- Ryu: dekorátorokkal hasonló eredmény:



```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    msg = ev.msg
```



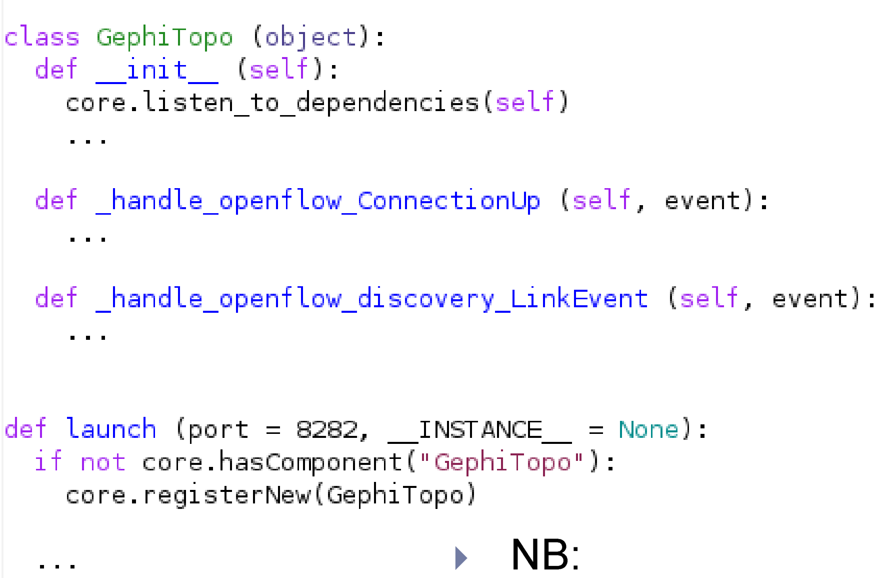
```
class GephiTopo (object):
  def __init__ (self):
    core.listen_to_dependencies(self)
    ...

  def _handle_openflow_ConnectionUp (self, event):
    ...

  def _handle_openflow_discovery_LinkEvent (self, event):
    ...


def launch (port = 8282, __INSTANCE__ = None):
  if not core.hasComponent("GephiTopo"):
    core.registerNew(GephiTopo)

  ...
```

az osztály metódusnevei alapján fog az eseményekre reagálni

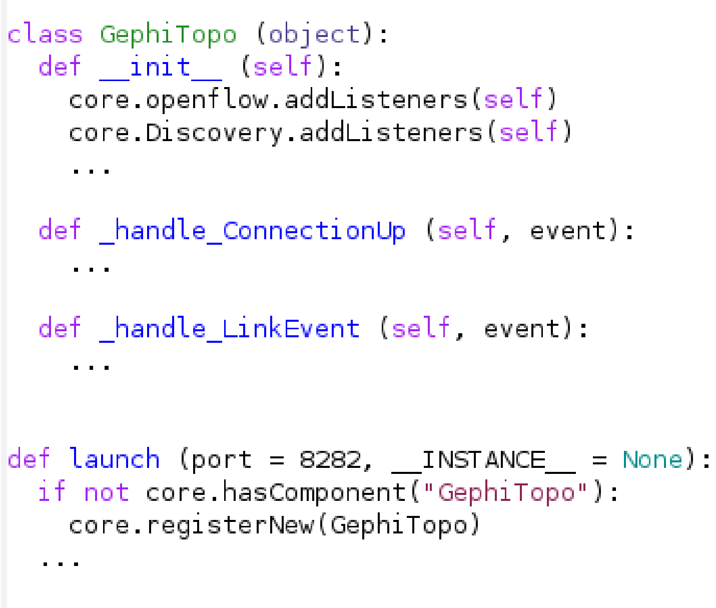
- Autómatikusan meghívódik az openflow osztály ConnectionUp küldésekor
- Illetve a openflow.discovery LinkEvent esemény küldésekor

Regisztálja a GephiTopo osztályt, aminek egy példánya ezután a globális core.GephiTopo változóként elérhető.

Eseménykezelés



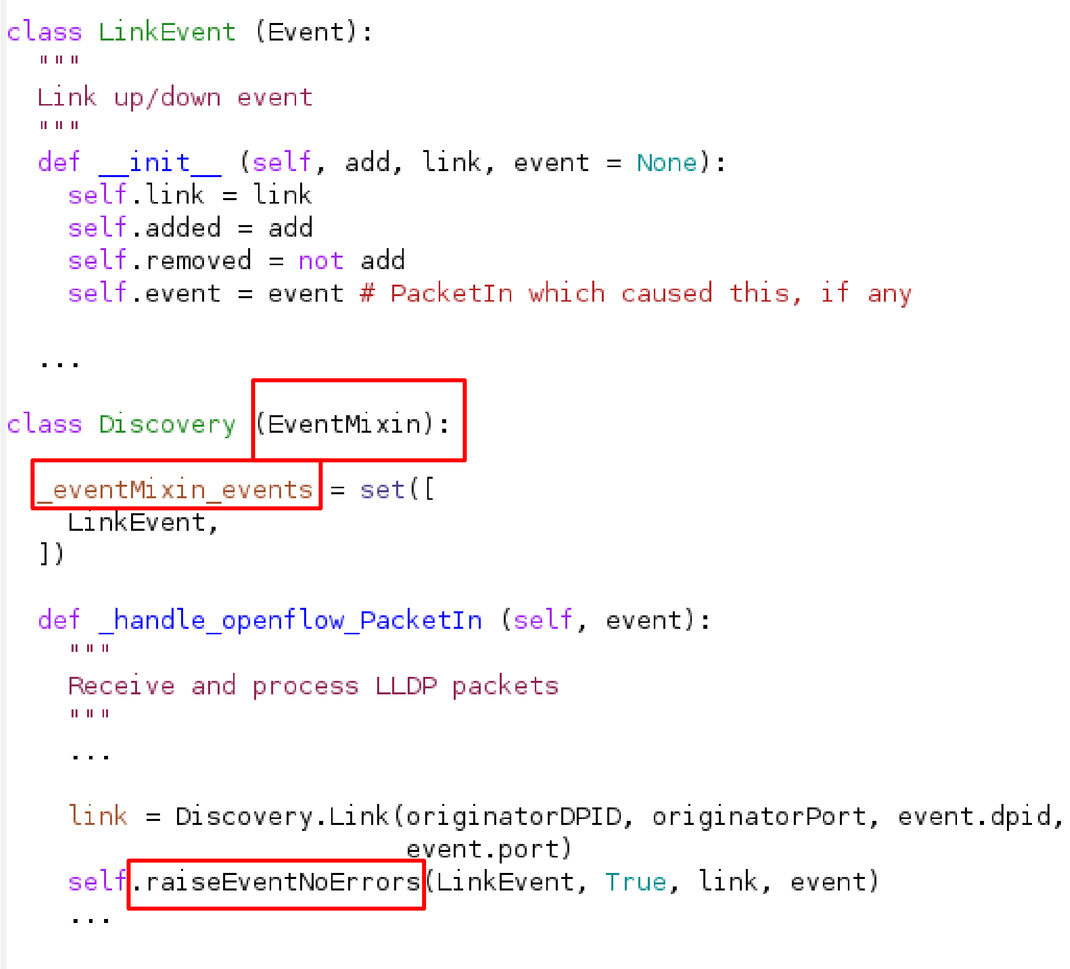
```
class GephiTopo (object):
  def __init__ (self):
    core.listen_to_dependencies(self)
    ...

  def _handle_openflow_ConnectionUp (self, event):
    ...

  def _handle_openflow_discovery_LinkEvent (self, event):
    ...


def launch (port = 8282, __INSTANCE__ = None):
  if not core.hasComponent("GephiTopo"):
    core.registerNew(GephiTopo)

  ...
```



```
class GephiTopo (object):
  def __init__ (self):
    core.openflow.addListeners(self)
    core.Discovery.addListeners(self)
    ...

  def _handle_ConnectionUp (self, event):
    ...

  def _handle_LinkEvent (self, event):
    ...


def launch (port = 8282, __INSTANCE__ = None):
  if not core.hasComponent("GephiTopo"):
    core.registerNew(GephiTopo)
  ...
```

- NB:
  - core.openflow.addListeners()
  - core.registerNew(Class, “name”)

Események küldése

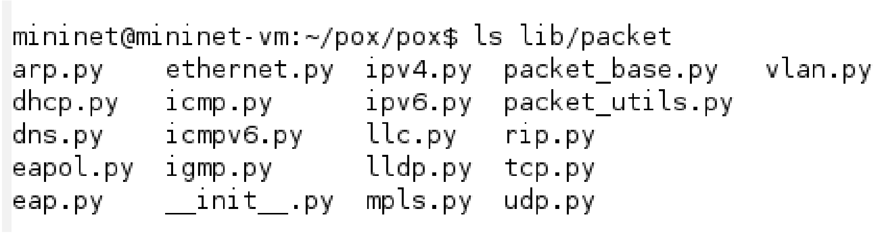


```
class LinkEvent (Event):
  """
  Link up/down event
  """
  def __init__ (self, add, link, event = None):
    self.link = link
    self.added = add
    self.removed = not add
    self.event = event # PacketIn which caused this, if any

  ...

class Discovery (EventMixin):

  _eventMixin_events = set([
    LinkEvent,
  ])

  def _handle_openflow_PacketIn (self, event):
    """
    Receive and process LLDP packets
    """
    ...

    link = Discovery.Link(originatorDPID, originatorPort, event.dpid,
                          event.port)
    self.raiseEventNoErrors(LinkEvent, True, link, event)
    ...
```

- raiseEvent vs raiseEventNoErrors
  - Utóbbi esetben a kivételeket automatikusan elkapja a keretrendszer (hiba esetén a program működése nem áll meg)

Lazán csatolt komponensek

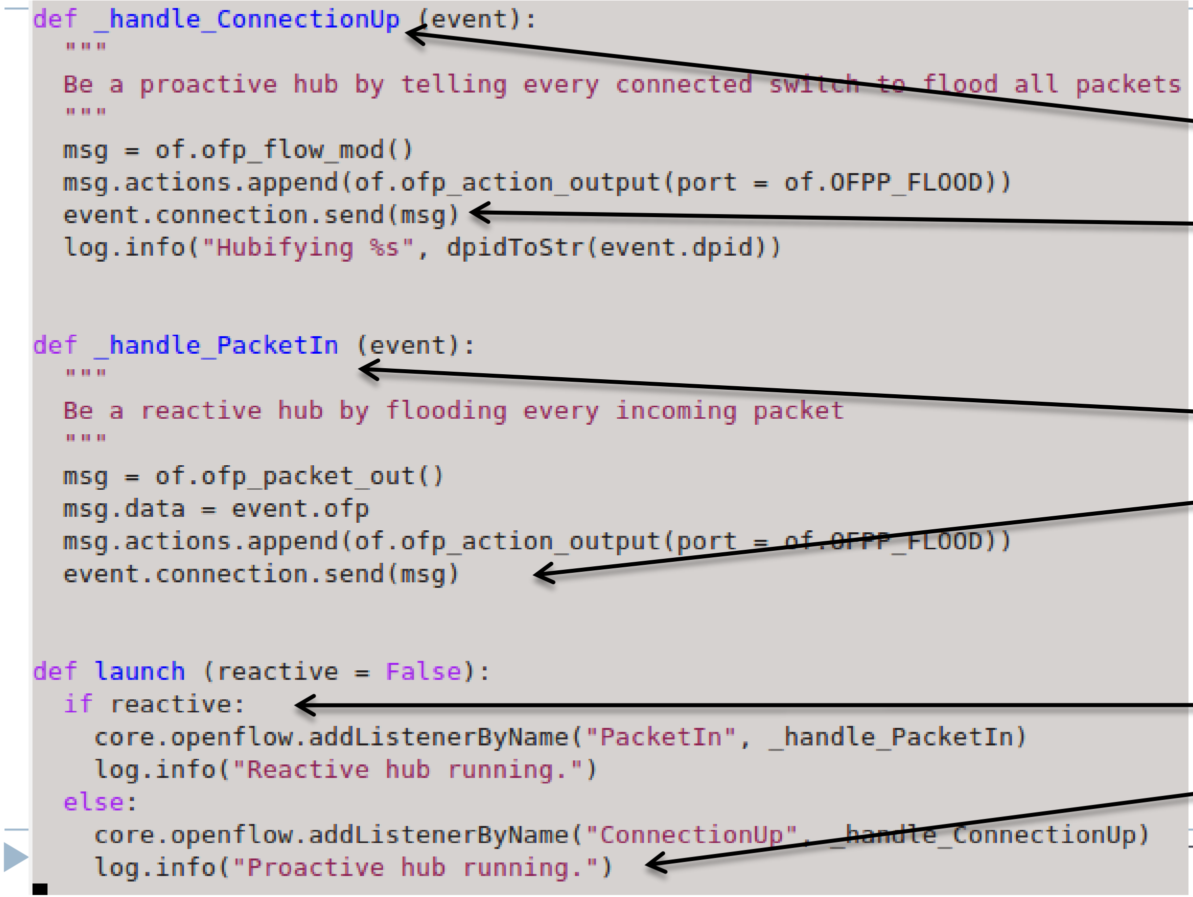
- GephiTopo LinkEventre vár; nem számít ki küldi
- Discovery LLDP alapján térképezi fel a hálózati topológiát
- De lehetne írni egy komponenst, ami pl. OSPF hello üzeneteket használna, de ugyanúgy LinkEventeket küldene



```
mininet@mininet-vm:~/pox/pox$ ls lib/packet
arp.py     ethernet.py  ipv4.py  packet_base.py   vlan.py
dhcp.py    icmp.py      ipv6.py  packet_utils.py
dns.py     icmpv6.py    llc.py   rip.py
eapol.py   igmp.py      lldp.py  tcp.py
eap.py     __init__.py  mpls.py  udp.py
```

- libopenflow_01
  - Python objektumok és a bináris hálózati formátum között végez átalakításokat
- openflow.of_01
  - Python objektumok segítségével valósítja meg az OF protokollt
  - Egyes protokolleseményekhez eseményeket küld (pl.: PacketIn)
- Discovery
  - OF eseményeket lekezeli, absztrakt eseményeket (LinkEvent) küld
  - Csomaggenerálásra, -feldolgozásra a lib.packet.* osztályokat használja
- GephiTopo
  - Az absztrakt eseményeket dolgozza fel.
  - RPC adatforrást nyújt a Gephi programnak.

Hub alkalmazás POX kontrollerben



```
def _handle_ConnectionUp (event):
  """
  Be a proactive hub by telling every connected switch to flood all packets
  """
  msg = of.ofp_flow_mod()
  msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
  event.connection.send(msg)
  log.info("Hubifying %s", dpidToStr(event.dpid))


def _handle_PacketIn (event):
  """
  Be a reactive hub by flooding every incoming packet
  """
  msg = of.ofp_packet_out()
  msg.data = event.ofp
  msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
  event.connection.send(msg)


def launch (reactive = False):
  if reactive:
    core.openflow.addListenerByName("PacketIn", _handle_PacketIn)
    log.info("Reactive hub running.")
  else:
    core.openflow.addListenerByName("ConnectionUp", _handle_ConnectionUp)
    log.info("Proactive hub running.")
```

proaktív hub:

- ConnectionUp esemény kezelése
- (switch csatlakozása)
- flow bejegyzés összeállítása & leküldése

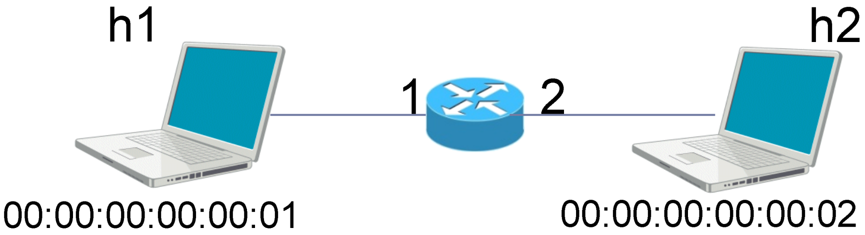
reaktív hub:

- PacketIn esemény kezelése
- (csomag sw→ ctrl)
- flow bejegyzés

kétféle üzemmód:

- reaktív
- proaktív

Learning switch



- https://github.com/noxrepo/pox/blob/eel/pox/forwarding/l2_learning.py

- h1 $: ping –c 1 h2

source address	source port

- Első csomag: who-has 10.0.0.2
  - Broadcast -> flood, de új sor a táblázatban 00:01-1

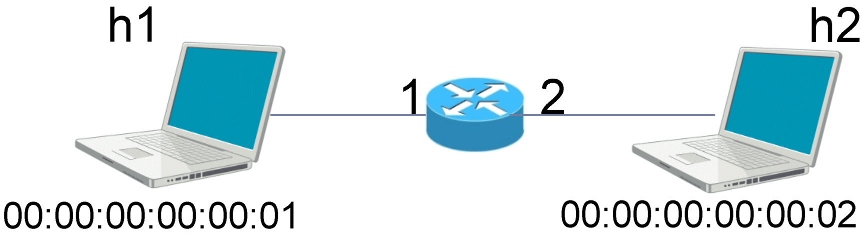
source address	source port
00:00:00:00:00:01	1

- Második csomag: arp-reply
  - Cél már ismert, de új sor a táblázatban: 00:02-2

source address	source port
00:00:00:00:00:01	1
00:00:00:00:00:02	2

- Harmadik csomag: ping 00:01→00:02
  - A kontrollernek nem kell felküldeni semmit sem

Learning switch



For each packet from the switch:

1) Use source address and switch port to update address/port table

2) Is transparent = False and either Ethertype is LLDP or the packet's destination address is a Bridge Filtered address?

Yes:

2a) Drop packet -- don't forward link-local traffic (LLDP, 802.1x) -- DONE

3) Is destination multicast?

Yes:

3a) Flood the packet -- DONE

4) Port for destination address in our address/port table?

No:

4a) Flood the packet -- DONE

5) Is output port the same as input port?

Yes:

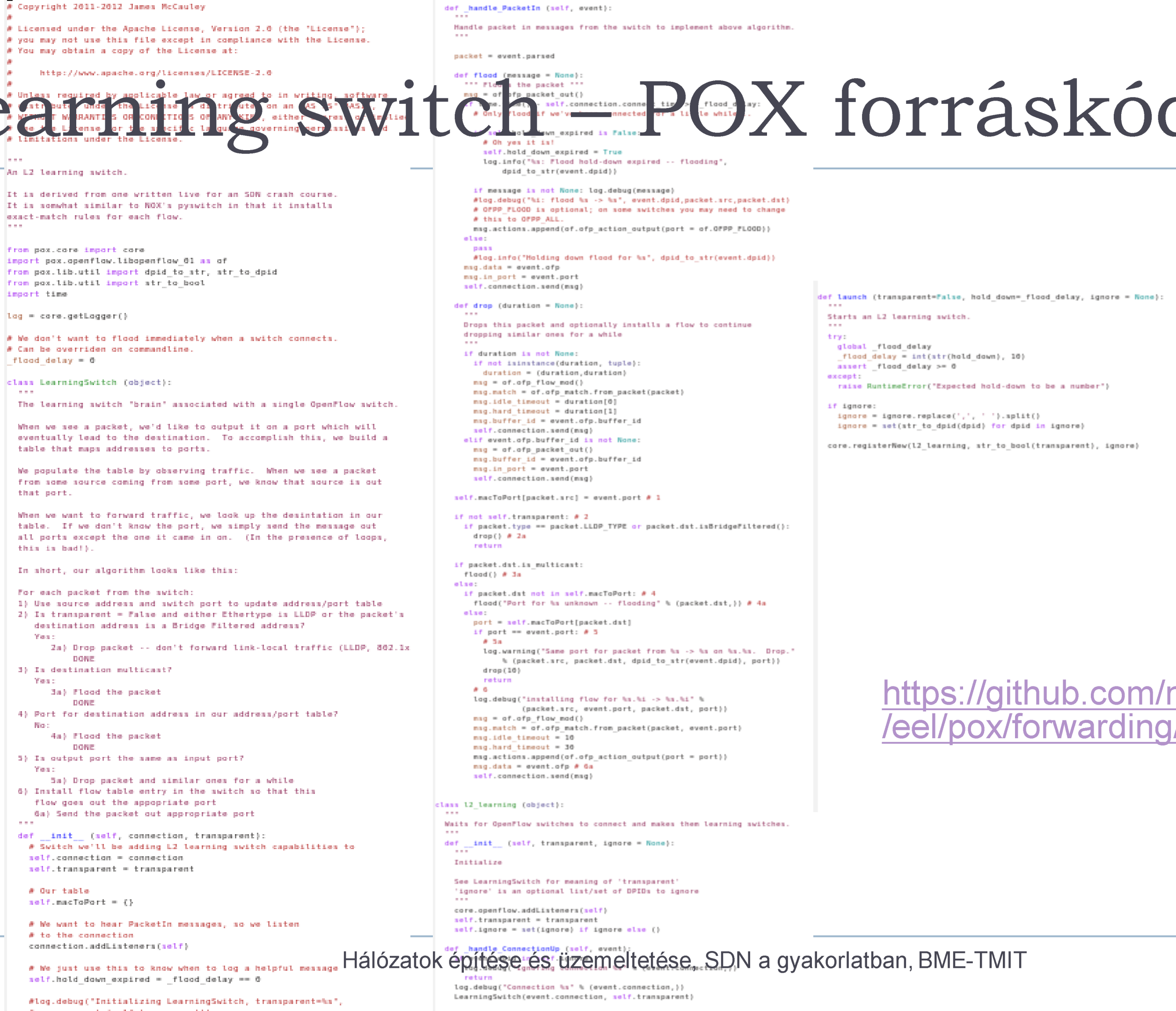
5a) Drop packet and similar ones for a while

6) Install flow table entry in the switch so that this flow goes out the appropriate port

6a) Send the packet out appropriate port

source address	source port
00:00:00:00:00:01	1
00:00:00:00:00:02	2

Learning switch – POX forráskód



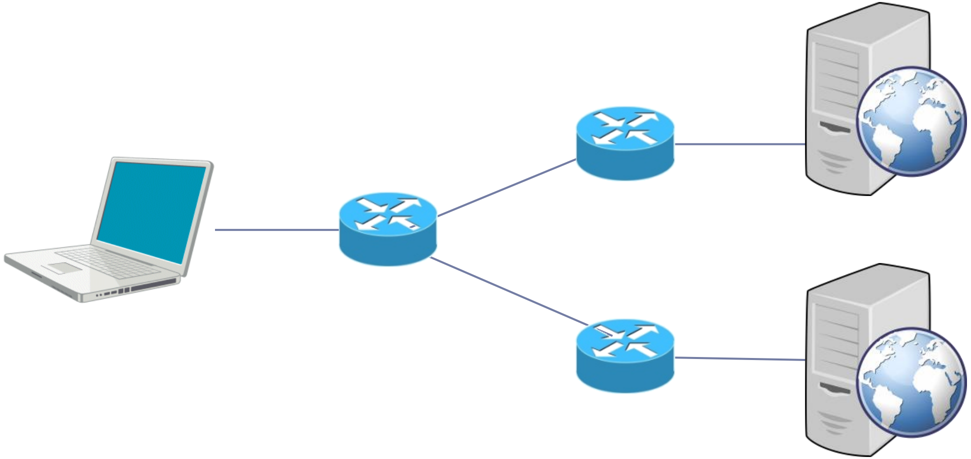
```
# Copyright 2011-2012 James McCauley
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at:
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""
An L2 learning switch.

It is derived from one written live for an SDN crash course.
It is somwhat similar to NOX's pyswitch in that it installs
exact-match rules for each flow.
"""

from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str, str_to_dpid
from pox.lib.util import str_to_bool
import time

log = core.getLogger()

# We don't want to flood immediately when a switch connects.
# Can be overriden on commandline.
_flood_delay = 0

class LearningSwitch (object):
  """
  The learning switch "brain" associated with a single OpenFlow switch.

  When we see a packet, we'd like to output it on a port which will
  eventually lead to the destination.  To accomplish this, we build a
  table that maps addresses to ports.

  We populate the table by observing traffic.  When we see a packet
  from some source coming from some port, we know that source is out
  that port.

  When we want to forward traffic, we look up the desintation in our
  table.  If we don't know the port, we simply send the message out
  all ports except the one it came in on.  (In the presence of loops,
  this is bad!).

  In short, our algorithm looks like this:

  For each packet from the switch:
  1) Use source address and switch port to update address/port table
  2) Is transparent = False and either Ethertype is LLDP or the packet's
     destination address is a Bridge Filtered address?
     Yes:
        2a) Drop packet -- don't forward link-local traffic (LLDP, 802.1x
            DONE
  3) Is destination multicast?
     Yes:
        3a) Flood the packet
            DONE
  4) Port for destination address in our address/port table?
     No:
        4a) Flood the packet
            DONE
  5) Is output port the same as input port?
     Yes:
        5a) Drop packet and similar ones for a while
  6) Install flow table entry in the switch so that this
     flow goes out the appopriate port
     6a) Send the packet out appropriate port
  """
  def __init__ (self, connection, transparent):
    # Switch we'll be adding L2 learning switch capabilities to
    self.connection = connection
    self.transparent = transparent

    # Our table
    self.macToPort = {}

    # We want to hear PacketIn messages, so we listen
    # to the connection
    connection.addListeners(self)

    # We just use this to know when to log a helpful message
    self.hold_down_expired = _flood_delay == 0

    #log.debug("Initializing LearningSwitch, transparent=%s",
```

```
  def _handle_PacketIn (self, event):
    """
    Handle packet in messages from the switch to implement above algorithm.
    """

    packet = event.parsed

    def flood (message = None):
      """ Floods the packet """
      msg = of.ofp_packet_out()
      if time.time() - self.connection.connect_time >= _flood_delay:
        # Only flood if we've been connected for a little while...

        if self.hold_down_expired is False:
          # Oh yes it is!
          self.hold_down_expired = True
          log.info("%s: Flood hold-down expired -- flooding",
              dpid_to_str(event.dpid))

        if message is not None: log.debug(message)
        #log.debug("%i: flood %s -> %s", event.dpid,packet.src,packet.dst)
        # OFPP_FLOOD is optional; on some switches you may need to change
        # this to OFPP_ALL.
        msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
      else:
        pass
        #log.info("Holding down flood for %s", dpid_to_str(event.dpid))
      msg.data = event.ofp
      msg.in_port = event.port
      self.connection.send(msg)

    def drop (duration = None):
      """
      Drops this packet and optionally installs a flow to continue
      dropping similar ones for a while
      """
      if duration is not None:
        if not isinstance(duration, tuple):
          duration = (duration,duration)
        msg = of.ofp_flow_mod()
        msg.match = of.ofp_match.from_packet(packet)
        msg.idle_timeout = duration[0]
        msg.hard_timeout = duration[1]
        msg.buffer_id = event.ofp.buffer_id
        self.connection.send(msg)
      elif event.ofp.buffer_id is not None:
        msg = of.ofp_packet_out()
        msg.buffer_id = event.ofp.buffer_id
        msg.in_port = event.port
        self.connection.send(msg)

    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)


class l2_learning (object):
  """
  Waits for OpenFlow switches to connect and makes them learning switches.
  """
  def __init__ (self, transparent, ignore = None):
    """
    Initialize

    See LearningSwitch for meaning of 'transparent'
    'ignore' is an optional list/set of DPIDs to ignore
    """
    core.openflow.addListeners(self)
    self.transparent = transparent
    self.ignore = set(ignore) if ignore else ()

  def _handle_ConnectionUp (self, event):
    if event.dpid in self.ignore:
      log.debug("Ignoring connection %s" % (event.connection,))
      return
    log.debug("Connection %s" % (event.connection,))
    LearningSwitch(event.connection, self.transparent)
```

```
def launch (transparent=False, hold_down=_flood_delay, ignore = None):
  """
  Starts an L2 learning switch.
  """
  try:
    global _flood_delay
    _flood_delay = int(str(hold_down), 10)
    assert _flood_delay >= 0
  except:
    raise RuntimeError("Expected hold-down to be a number")

  if ignore:
    ignore = ignore.replace(',', ' ').split()
    ignore = set(str_to_dpid(dpid) for dpid in ignore)

  core.registerNew(l2_learning, str_to_bool(transparent), ignore)
```

https://github.com/noxrepo/pox/blob/eel/pox/forwarding/l2_learning.py

Példa: teheléselosztó (gyakorlaton lesz)

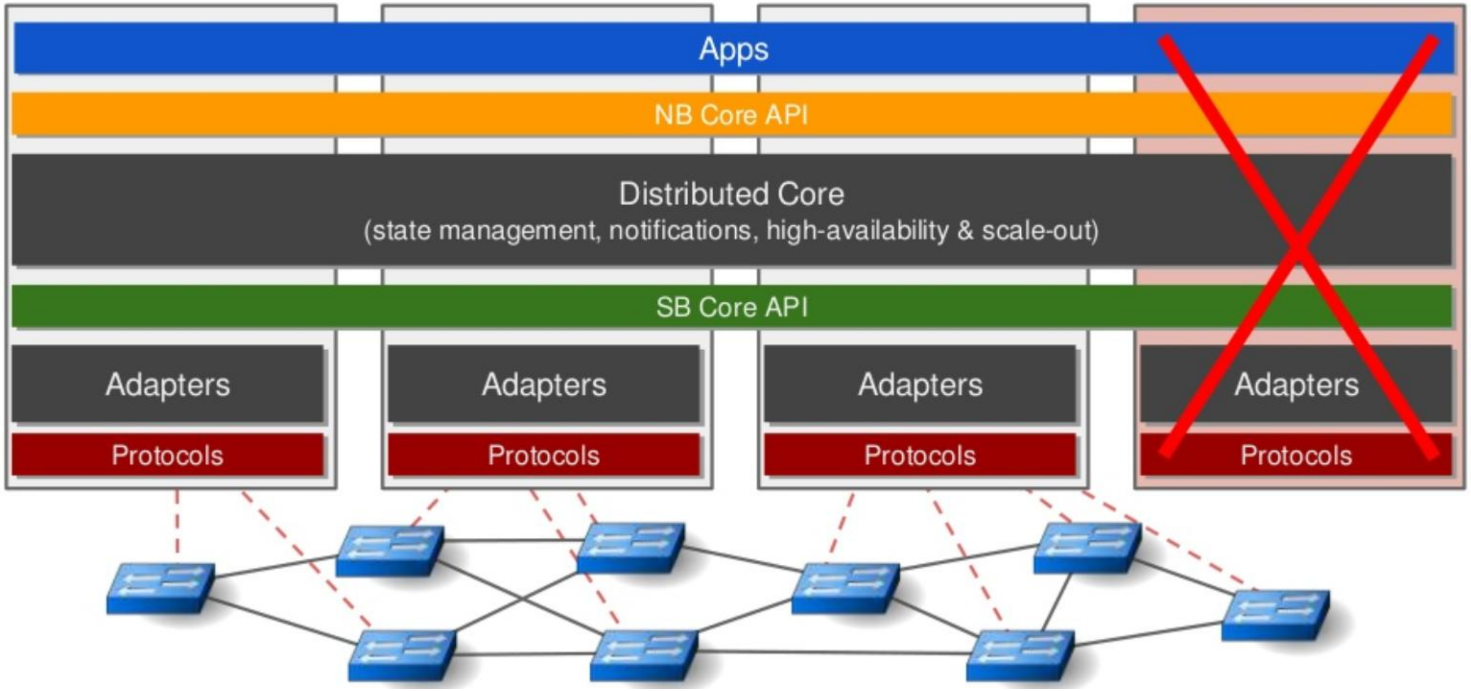


- Terheléselosztás alapja?
  - random
  - adatforgalom a linkeken
  - terhelés a szervereken
- Megvalósítási kérdések
  - milyen infót gyűjtsön a kontroller
  - hogyan gyűjtse
  - hogyan határozza meg az utakat

Kontrolleralkalmazások

- Mit lehet megvalósítani?
  - Mindent, de igazából sem sok újat
  - Amit eddig elosztott protokoll valósított meg, most egyszerűbben, központosítva, globális nézeten lehet megírni
- A jó keretrendszer
  - karbantartja a globális nézetet
  - hibatűrő
  - logikai centralizációt biztosít (ONOS)

Distributed Architecture ONOS



- logikai centralizációt biztosít
  - a hálózat fennakadás nélkül üzemel, még ha egy kontrollerpéldányt futtató hardver le is hal
  - az adatok sokszorozását a keretrendszer végzi, a programozónak nem kell ezzel foglalkoznia
- OSGi komponensek (OpenDaylighthoz hasonlóan)

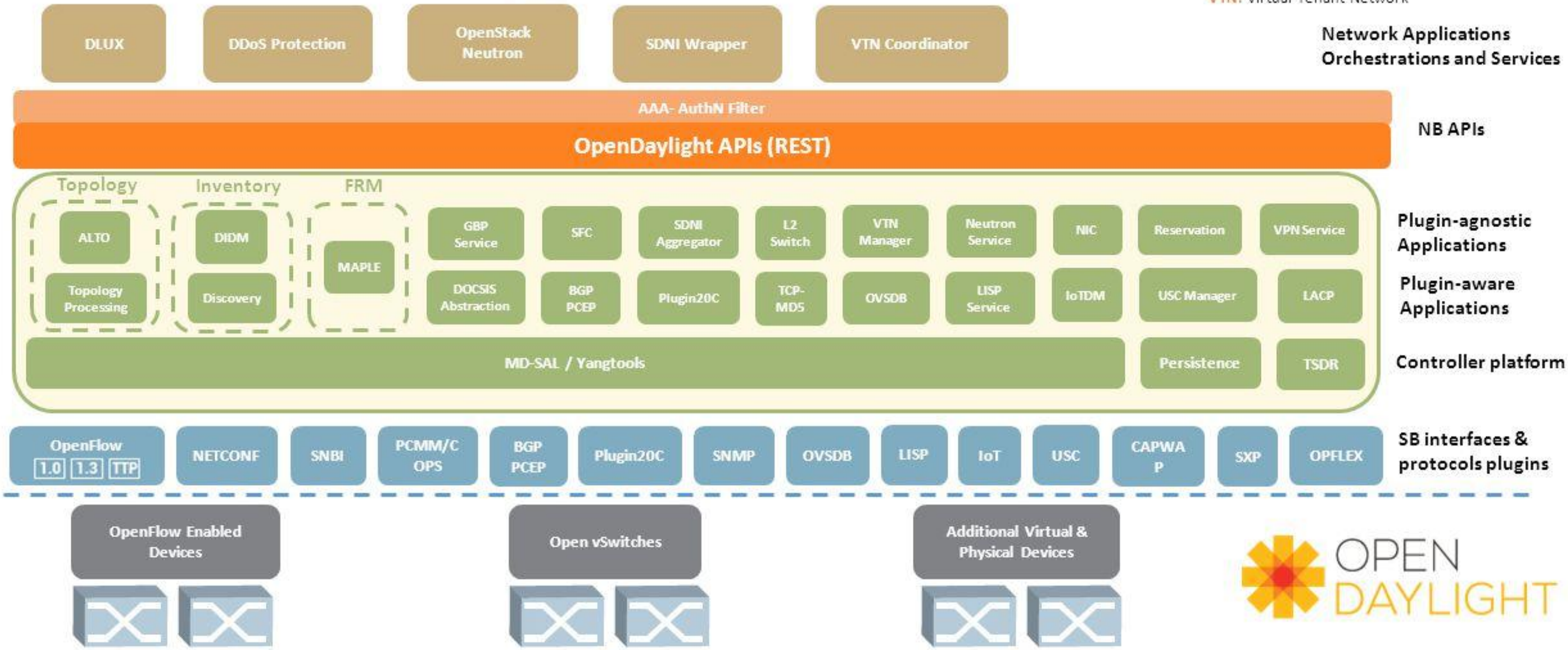
ONOS – intent
Példa egy északi interfészre

- Leírja, hogy minek kéne lenne ahelyett, hogy leírná, hogy hogyan kéne elérni a célt
- Pl.: “Host A és Host B között legyen összeköttetés”
- ONOS a topológia függvényében ezt egy úttá alakítja és beállítja a kapcsolókban a megfelelő szabályokat
- Ha topológia változik, akkor az út is változhat, de az északi interfészen nincs forgalom

OPEN DAYLIGHT "LITHIUM"

Legend

AAA: Authentication, Authorization & Accounting
ALTO: Application Layer Traffic Optimization
AuthN: Authentication
BGP: Border Gateway Protocol
CAPWAP: Control and Provisioning of Wireless Access Points
COPS: Common Open Policy Service
DIDM: Device Identification and Driver management
DLUX: OpenDaylight User Experience
DDoS: Distributed Denial Of Service
DOCSIS: Data Over Cable Service Interface Specification
FRM: Forwarding Rules Manager
GBP: Group Based Policy
IoTDM: Internet of Things Data Broker
LACP: Link Aggregation Control Protocol
LISP: Locator/Identifier Separation Protocol
MAPLE: Maple Programming
NIC: Network Intent Proposal
OVSDB: Open vSwitch DataBase Protocol
OPFLEX: Extensible Policy Protocol
PCEP: Path Computation Element Protocol
PCMM: Packet Cable MultiMedia
Plugin2OC: Plugin To OpenContrail
SDNI: SDN Interface (Cross-Controller Federation)
SFC: Service Function Chaining
SNBI: Secure Network Bootstrapping Infrastructure
SNMP: Simple Network Management Protocol
SXP: Source-Group Tag eXchange Protocol
TSDR: Time Series Data Repository
TTP: Table Type Patterns
USC: Unified Secure Channel
VTN: Virtual Tenant Network



Network Applications Orchestrations and Services: DLUX, DDoS Protection, OpenStack Neutron, SDNI Wrapper, VTN Coordinator

NB APIs: AAA- AuthN Filter; OpenDaylight APIs (REST)

Plugin-agnostic Applications / Plugin-aware Applications:
Topology: ALTO, Topology Processing
Inventory: DIDM, Discovery
FRM: MAPLE
GBP Service, SFC, SDNI Aggregator, L2 Switch, VTN Manager, Neutron Service, NIC, Reservation, VPN Service
DOCSIS Abstraction, BGP PCEP, Plugin20C, TCP-MD5, OVSDB, LISP Service, IoTDM, USC Manager, LACP

Controller platform: MD-SAL / Yangtools, Persistence, TSDR

SB interfaces & protocols plugins: OpenFlow 1.0 1.3 TTP, NETCONF, SNBI, PCMM/COPS, BGP PCEP, Plugin20C, SNMP, OVSDB, LISP, IoT, USC, CAPWAP, SXP, OPFLEX

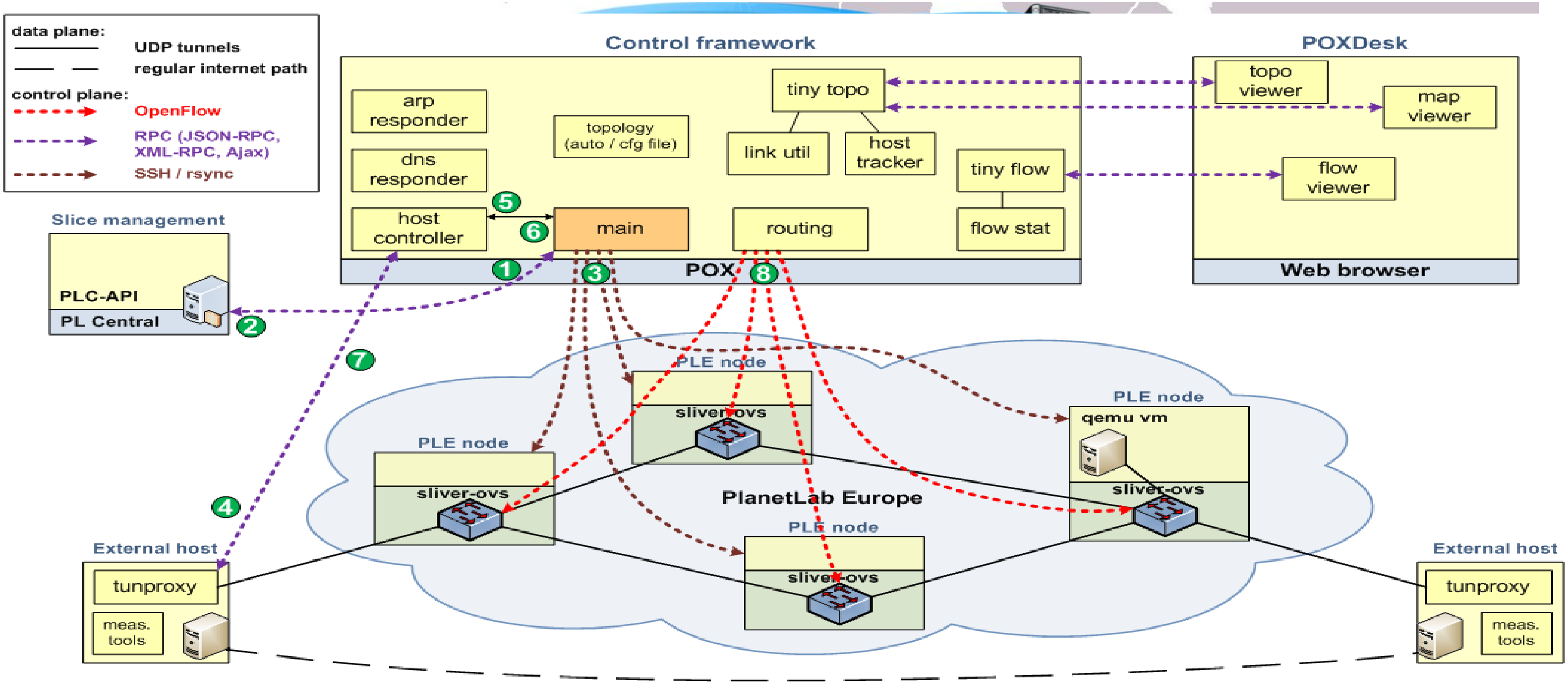
OpenFlow Enabled Devices, Open vSwitches, Additional Virtual & Physical Devices

OPEN DAYLIGHT

Azért POX-ban is lehet komplex kontrollert írni

Példa: Multipath TCP + SDN

MultiPath TCP + SDN



data plane:
UDP tunnels
regular internet path
control plane:
OpenFlow
RPC (JSON-RPC, XML-RPC, Ajax)
SSH / rsync

Control framework
arp responder
dns responder
host controller
topology (auto / cfg file)
main
tiny topo
link util
host tracker
routing
tiny flow
flow stat
POX

POXDesk
topo viewer
map viewer
flow viewer
Web browser

Slice management
PLC-API
PL Central

PlanetLab Europe
PLE node
sliver-ovs
PLE node
sliver-ovs
PLE node
qemu vm
sliver-ovs
PLE node
sliver-ovs

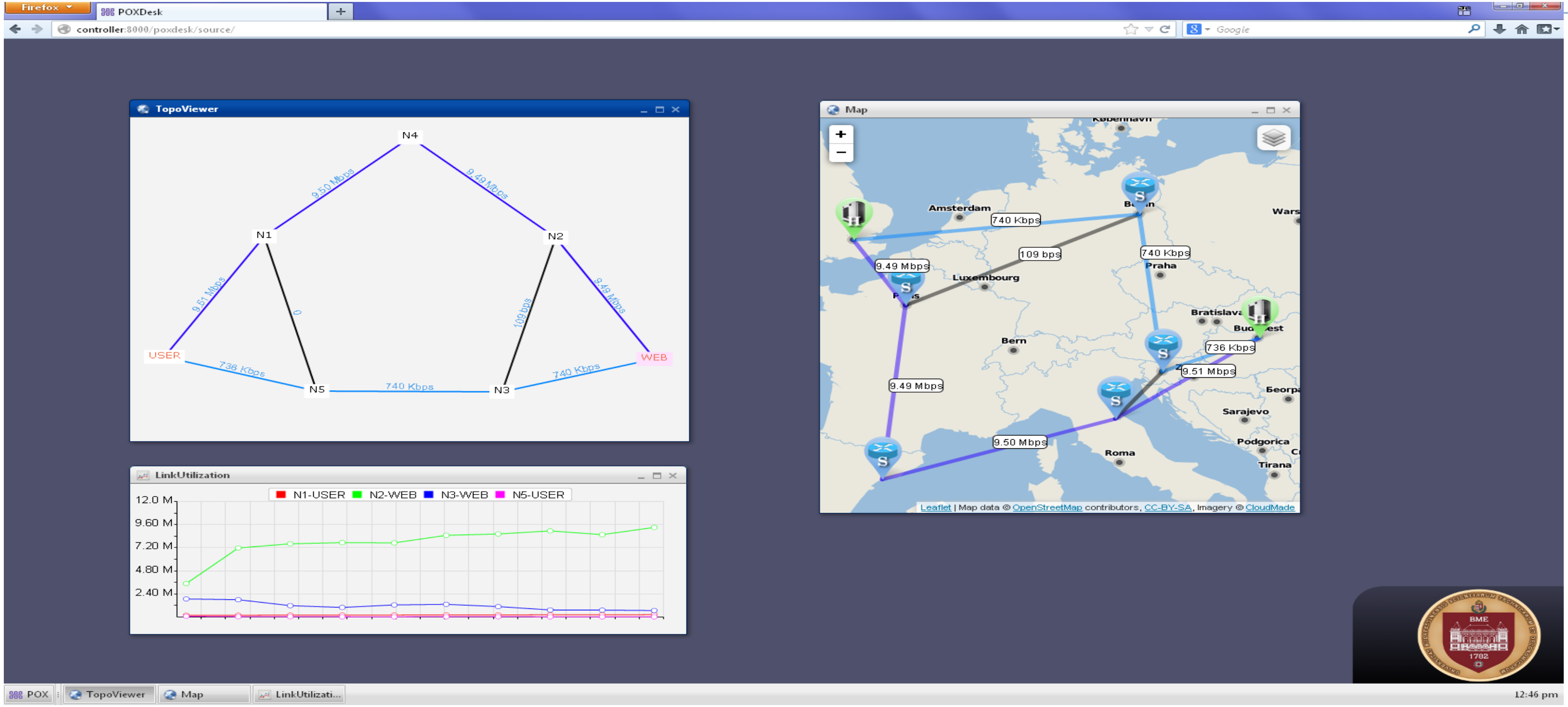
External host
tunproxy
meas. tools

External host
tunproxy
meas. tools

1
2
3
4
5
6
7
8

PLANETLAB
An open platform for developing, deploying, and accessing planetary-scale services

MultiPath TCP + SDN


Firefox
POXDesk
controller:8000/poxdesk/source/
Google
TopoViewer
N4
N1
N2
USER
WEB
N5
N3
9.50 Mbps
9.49 Mbps
9.51 Mbps
0
109 bps
9.49 Mbps
736 Kbps
740 Kbps
740 Kbps
LinkUtilization
N1-USER
N2-WEB
N3-WEB
N5-USER
12.0 M
9.60 M
7.20 M
4.80 M
2.40 M
Map
København
Amsterdam
Berlin
Warszawa
Praha
Luxembourg
Paris
Bern
Bratislava
Budapest
Beograd
Sarajevo
Podgorica
Tirana
Roma
740 Kbps
109 bps
740 Kbps
9.49 Mbps
736 Kbps
9.51 Mbps
9.49 Mbps
9.50 Mbps
S
H
Leaflet | Map data © OpenStreetMap contributors, CC-BY-SA, Imagery © CloudMade
BME
1782
POX
TopoViewer
Map
LinkUtilizati...
12:46 pm