

# Hálózatok építése és üzemeltetése

OpenFlow / POX gyakorlat

---

▶ **Előző gyakorlat:**

- ▶ OSPF (routing protokoll)
- ▶ elosztott működés
- ▶ több-több (many-to-many) kommunikáció
- ▶ bonyolult!

▶ **Most:**

- ▶ más koncepció, SDN
- ▶ (bonyolult??)

# 1. feladat:

## térképezzük fel az emulált topológiát

---

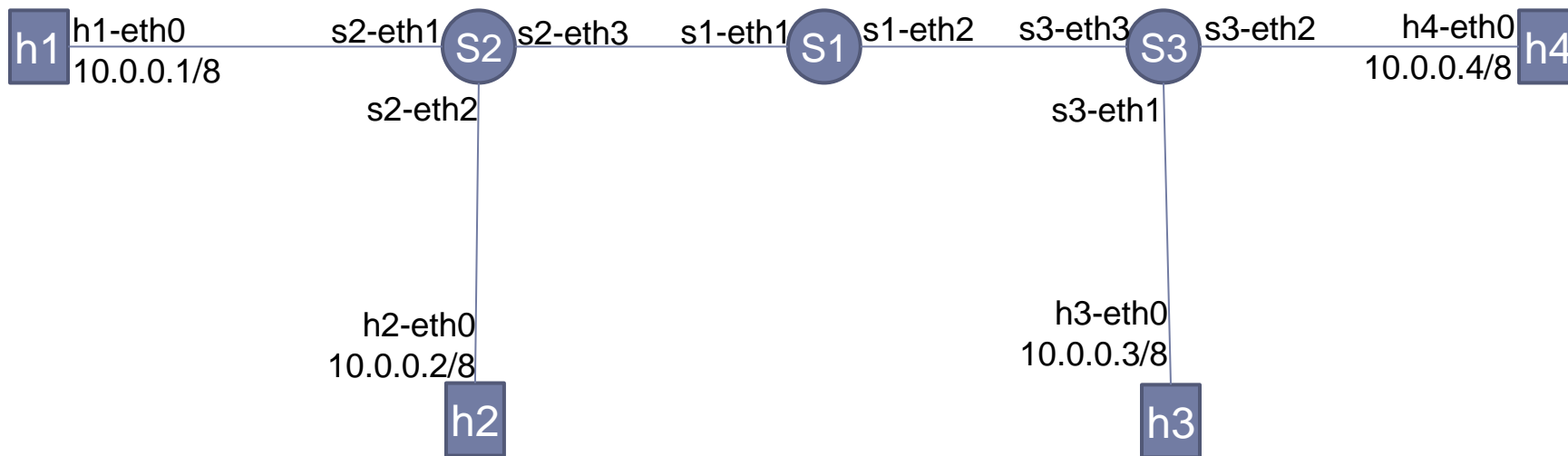
- ▶ `$ sudo -E mn --topo tree,depth=2 --controller=remote,port=6633`
- ▶ **Használható parancsok:**
  - ▶ `mininet> net`
  - ▶ `mininet> dump`
  - ▶ `mininet> h1 ifconfig`
  - ▶ `mininet> ...`

# 1. feladat:

térképezzük fel az emulált topológiát

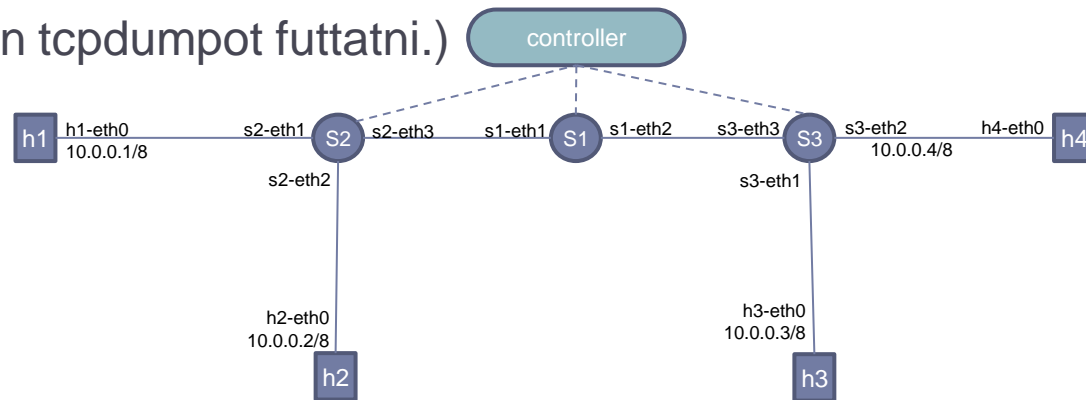


▶ `$ sudo -E mn --topo tree,depth=2 --controller=remote`



## 2. feladat: forgalom megfigyelése

- ▶ `mininet> xterm h1`
- ▶ `h1: ~# ping -c 1 10.0.0.4`
- ▶ Figyeljük meg az `s?-eth?` interfészeket. Meddig jutnak el az ARP kérések?
  - ▶ (érdeemes külön ablakokban `tcpdump`ot futtatni.)
  - ▶ vizsgálandó linkek:
    - ▶ `h1-s2`, `s1-s2`, `s2-h2`



## 2. feladat: forgalom megfigyelése



- ▶ h1: ~# ping -c 1 10.0.0.4
- ▶ Figyeljük meg az s?-eth? interfészt. Meddig jutnak el az ARP kérések?
  - ▶ ALT-F2 xterm, majd: ~\$ sudo wireshark &
  - ▶ Mindössze három ARP csomag lesz a h1-s2 linken.
- ▶ Hallgassunk bele a lo interfészbe is!
  - ▶ (itt a wiresharkot érdemes használni, de az installált verzió *már* nem támogatja megfelelően a OF1.0-ás csomagokat)

## 2. feladat: forgalom megfigyelése

---

- ▶ h1: ~# ping -c 1 10.0.0.4
- ▶ Figyeljük meg az s?-eth? interfészt. Meddig jutnak el az ARP kérések?
  - ▶ ALT-F2 xterm, majd: ~\$ sudo wireshark &
  - ▶ .
- ▶ Hallgassunk bele a lo interfészbe is!
  - ▶ (itt a wiresharkot érdemes használni, de az installált verzió *már* nem támogatja megfelelően a OF1.0-ás csomagokat)

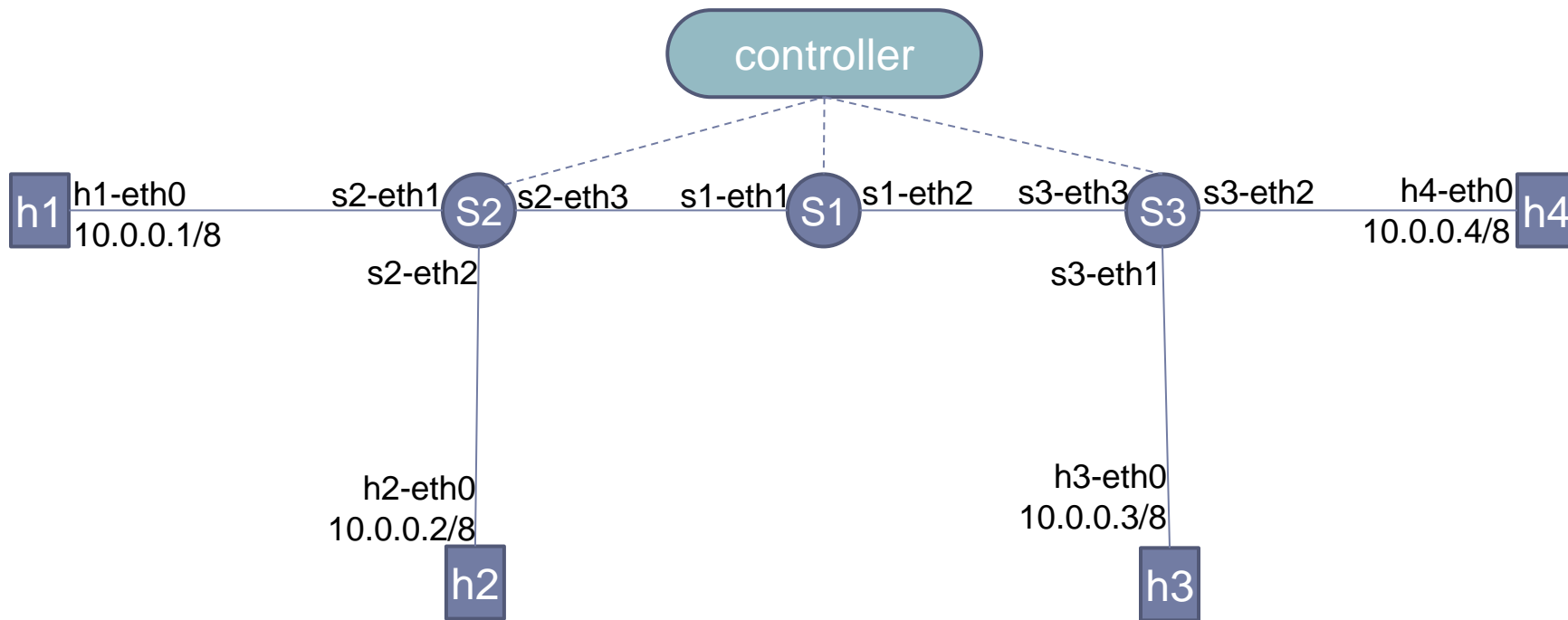
## 2. feladat: forgalom megfigyelése



- ▶ h1: ~# ping -c 1 10.0.0.4
- ▶ Figyeljük meg az s?-eth? interfészt. Meddig jutnak el az ARP kérések?
  - ▶ ALT-F2 xterm, majd: ~\$ sudo wireshark &
  - ▶ Mindössze három ARP csomag lesz a h1-s2 linken.
- ▶ Hallgassunk bele a lo interfészbe is!
  - ▶ A switchek próbálnak kapcsolódni a controllerhez.



# 1. feladat: térképezzük fel az emulált topológiát



# POX: OpenFlow controller-keretrendszer

- ▶ Elavult: csak OpenFlow 1.0-t támogat
  - ▶ Konkurensok: 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://noxrepo.github.io/pox-doc/html/>

# POX installálása, frissítése

---

A kiadott VM-en nincs rá szükség

```
git clone http://github.com/noxrepo/pox
```

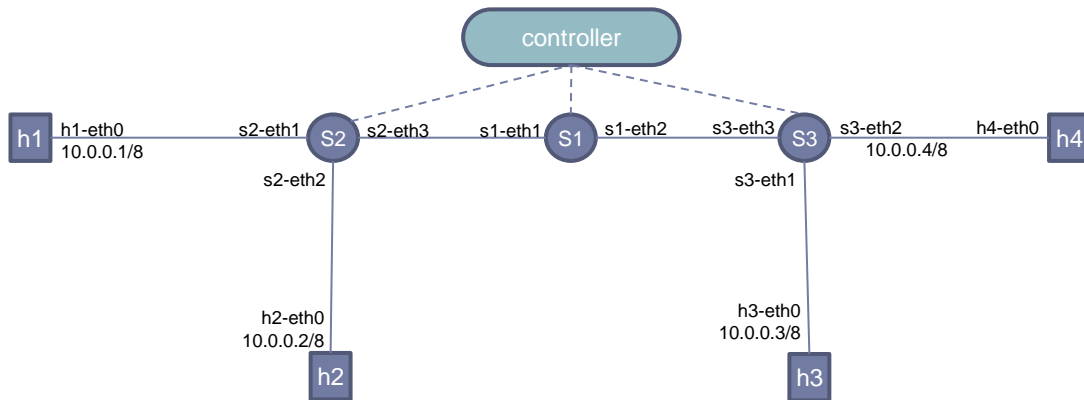
```
cd ~/pox
```

```
git pull
```

```
git checkout eel
```

# Egyszerű kontrolleralkalmazás

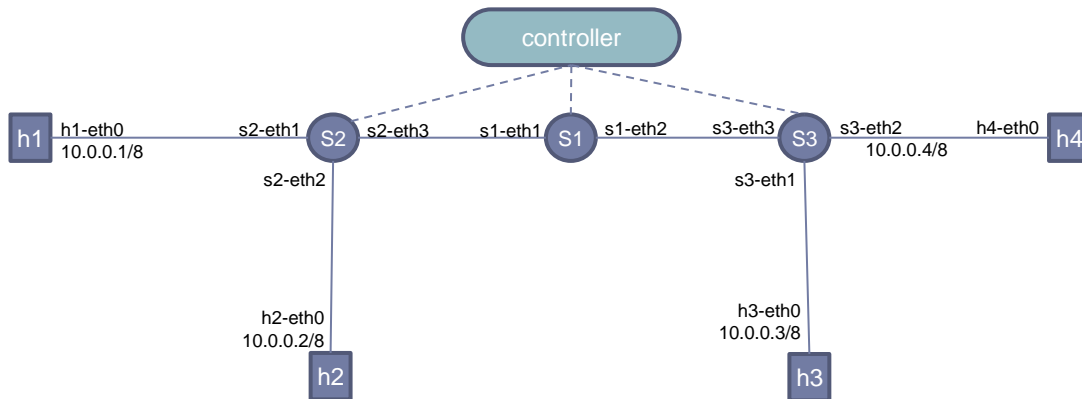
- ▶ `~/pox$ ./pox.py forwarding.hub`
- ▶ Most sikeres lesz az előző ping parancs, de hol lesz most adatforgalom?
  - ▶ (érdeemes külön ablakokban tcpdumpot futtatni.)
  - ▶ vizsgálandó linkek:
    - ▶ h1-s2, s1-s3, s2-h2
  - ▶ és interfész:
    - ▶ lo



# Egyszerű kontrolleralkalmazás



- ▶ `~/pox$ ./pox.py forwarding.hub`
- ▶ Most sikeres lesz az előző ping parancs, de hol lesz most adatforgalom?
  - ▶ minden linken
  - ▶ hub:
    - ▶ L1-es működés
    - ▶ buta csomagmásolás



# Mi történik a kontrollerrel?

---

- ▶ Vizsgáljuk meg a **lo** forgalmát a kontroller indításakor?
  - ▶ Értelmezzük az üzeneteket!

# Mi történik a kontrollerrel?



- ▶ Vizsgáljuk meg a **lo** forgalmát a kontroller indításakor?
  - ▶ kapcsolatfelépítés
  - ▶ FlowMod: szabályok installálása a switch-ekbe

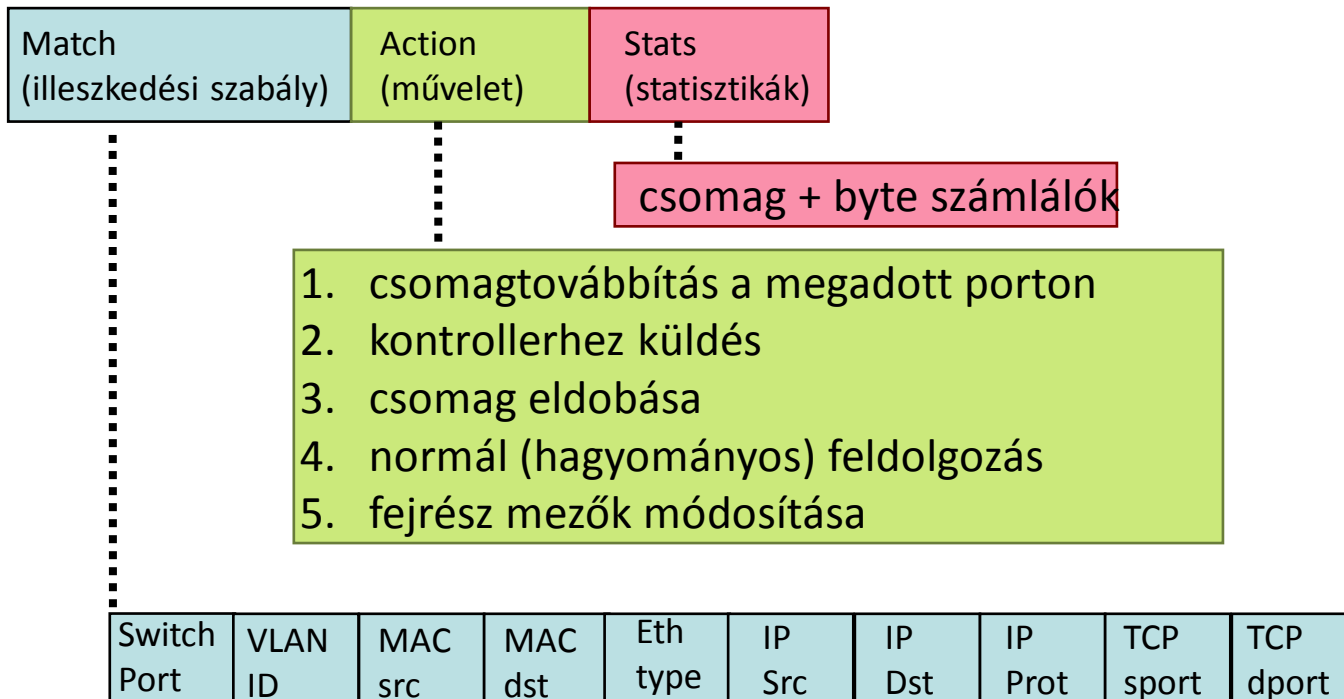
# Mi történik a kontrollerrel?

---

- ▶ Hogy nézhetnek ki a folyamtáblák a switch-ekben?
- ▶ Nézzük meg a folyamtáblákat!
  - ▶ `$ sudo ovs-ofctl show s2`
  - ▶ `$ sudo ovs-ofctl dump-flows s2`
  - ▶ `mininet> dpctl dump-flows`
  - ▶ `$ dpctl dump-flows tcp:localhost:6654 (vagy 55, 56)`



# Folyam tábla-bejegyzés



+ a nem szükséges mezők maszkolhatók (wildcard)

## Egyszerű kontrolleralkalmazás 2.

---

- ▶ `~/pox$ ./pox.py log.level --DEBUG forwarding.hub --reactive`
- ▶ Most sikeres is lesz az előző ping parancs, de hogyan alakul most a **kontroll**forgalom?
- ▶ Mi lesz a folyamtáblákban? Miért?

# Egyszerű kontrolleralkalmazás 2.



- ▶ `~/pox$ ./pox.py log.level --DEBUG forwarding.hub --reactive`
- ▶ Most sikeres is lesz az előző ping parancs, de hogyan alakul most a **kontroll**forgalom?
- ▶ Mi lesz a folyamatáblákban? Miért?
  - ▶ Semmi, mert a switchek minden csomagot felküldenek a kontrollernek, ami nem installál folyamatbejegyzéseket, hanem minden egyes csomagra megmondja, hogy merre kell továbbítania a switchnek.

# Érdekesebb folyamatábla bejegyzések

---

```
$ ./pox.py log.color log.level --DEBUG forwarding.l2_learning
```

```
h2:~$ nc -l 2222
```

```
h1:~$ date|nc 10.0.0.2 2222
```

```
mininet> dpctl dump-flows
```

# Érdekesebb folyamatábla bejegyzések

```
$ ./pox.py log.color log.level --DEBUG forwarding.l2_learning
```

```
h2:~$ nc -l 2222
```

```
h1:~$ date|nc 10.0.0.2 2222
```

Érdekesebb statisztikai mezők

```
cookie=0x0, duration=2.27s, table=0, n_packets=5, n_bytes=367,  
idle_timeout=10, hard_timeout=30, idle_age=2, priority=65535,  
tcp,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,  
nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,tp_src=46359,tp_dst=2222  
actions=output:2
```

```
cookie=0x0, duration=2.267s, table=0, n_packets=3, n_bytes=206,  
idle_timeout=10, hard_timeout=30, idle_age=2, priority=65535,  
tcp,in_port=2,vlan_tci=0x0000,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,  
nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,tp_src=2222,tp_dst=46359  
actions=output:1
```

# Érdekesebb folyamatábla bejegyzések

```
$ ./pox.py log.color log.level --DEBUG forwarding.l2_learning
```

```
h2:~$ nc -l 2222
```

```
h1:~$ date|nc 10.0.0.2 2222
```

Érdekesebb illeszkedési mezők

```
cookie=0x0, duration=2.27s, table=0, n_packets=5, n_bytes=367,  
idle_timeout=10, hard_timeout=30, idle_age=2, priority=65535,  
tcp,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,  
nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,tp_src=46359,tp_dst=2222  
actions=output:2
```

```
cookie=0x0, duration=2.267s, table=0, n_packets=3, n_bytes=206,  
idle_timeout=10, hard_timeout=30, idle_age=2, priority=65535,  
tcp,in_port=2,vlan_tci=0x0000,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,  
nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,tp_src=2222,tp_dst=46359  
actions=output:1
```

# Érdekesebb folyamatábla bejegyzések

```
$ ./pox.py log.color log.level --DEBUG forwarding.l2_learning
```

```
h2:~$ nc -l 2222
```

```
h1:~$ date|nc 10.0.0.2 2222
```

Egyszerű akciólista

```
cookie=0x0, duration=2.27s, table=0, n_packets=5, n_bytes=367,  
idle_timeout=10, hard_timeout=30, idle_age=2, priority=65535,  
tcp,in_port=1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,  
nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,tp_src=46359,tp_dst=2222  
actions=output:2
```

```
cookie=0x0, duration=2.267s, table=0, n_packets=3, n_bytes=206,  
idle_timeout=10, hard_timeout=30, idle_age=2, priority=65535,  
tcp,in_port=2,vlan_tci=0x0000,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,  
nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,tp_src=2222,tp_dst=46359  
actions=output:1
```

- 
- ▶ `sudo -E mn --mac --topo linear,k=5,n=1 --controller=remote,port=6633 --link=tc,delay=10ms`
  - ▶ `pox.py log.color log.level --DEBUG forwarding.l2_learning`
  - ▶ `mininet> h1 ping h5`



- 
- ▶ `sudo -E mn --mac --topo linear,k=5,n=1 --controller=remote,port=6633 --link=tc,delay=10ms`
  - ▶ `pox.py log.color log.level --DEBUG forwarding.l2_learning`

▶ `mininet> h1 ping h5`

PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.

64 bytes from 10.0.0.5: icmp\_seq=1 ttl=64 time=**459** ms

64 bytes from 10.0.0.5: icmp\_seq=2 ttl=64 time=**167** ms

64 bytes from 10.0.0.5: icmp\_seq=3 ttl=64 time=124 ms

64 bytes from 10.0.0.5: icmp\_seq=4 ttl=64 time=121 ms

^C

- ▶ Mitől nagyobb az első válaszidő?
  - ▶ szokásos tcpdump + wireshark + ... megfigyelések kellene



- ▶ sudo -E mn --mac --topo linear,k=5,n=1 --controller=remote,port=6633 --link=tc,delay=10ms
- ▶ pox.py log.color log.level --DEBUG forwarding.l2\_learning

▶ mininet> h1 ping h5

PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.

64 bytes from 10.0.0.5: icmp\_seq=1 ttl=64 time=459 ms

64 bytes from 10.0.0.5: icmp\_seq=2 ttl=64 time=167 ms

64 bytes from 10.0.0.5: icmp\_seq=3 ttl=64 time=124 ms

64 bytes from 10.0.0.5: icmp\_seq=4 ttl=64 time=121 ms

^C

- ▶ Mitől nagyobb az első válaszidő?
  - ▶ ARP + ping + kontrollerkommunikáció

---

```
pox.py log.color log.level --DEBUG forwarding.l2_learning proto.arp_responder --10.0.0.5=00:00:00:00:00:05 --  
10.0.0.1=00:00:00:00:00:01 py
```

▶ mininet> h1 ping h5

PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.

64 bytes from 10.0.0.5: icmp\_seq=1 ttl=64 time=232 ms

64 bytes from 10.0.0.5: icmp\_seq=2 ttl=64 time=186 ms

64 bytes from 10.0.0.5: icmp\_seq=3 ttl=64 time=121 ms

^C

▶ POX> arp



```
pox.py log.color log.level --DEBUG forwarding.l2_learning proto.arp_responder --10.0.0.5=00:00:00:00:00:05 --10.0.0.1=00:00:00:00:00:01 py
```

▶ mininet> h1 ping h5

PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.

64 bytes from 10.0.0.5: icmp\_seq=1 ttl=64 time=**232** ms

▶ gyorsabb az első PING, mert az első switch felküldi az ARP requestet a kontrollernek, ami rögtön megválaszolja; nem kell a csomagot 5 hopon keresztül küldeni

▶ **POX> arp**

▶ POX CLI, lekérdezi az arp modul egyik változóját: az ARP táblát.

# Szorgalmi (*i*MSc) feladat: POX módosítása

---

- ▶ Soronként megadni az ARP táblát macerás.  
Egyszerűbb is lehet az üzemeltetés, ha ismerjük a topológiát vagy algoritmikusan kikövetkeztethető a tábla.
- ▶ Jelenleg: 10.0.0.X → 00:00:00:00:00:X
- ▶ Írjuk át az arp\_responder modult, hogy automatikusan válaszoljon a 10.0.0.0/24-es kérésekre!

# POX, ethernet címek

---

- ▶ Írjuk át az `arp_responder` modult, hogy automatikusan válaszoljon a `10.0.0.0/24`-es kérésekre!
- ▶ `from pox.lib.addresses import EthAddr`
- ▶ **String** → **EthAddr**:  
`mac = EthAddr("aa:cc:dd:cc:AC:DC")`
- ▶ **EthAddr** → **string**:  
`s = str(mac)`

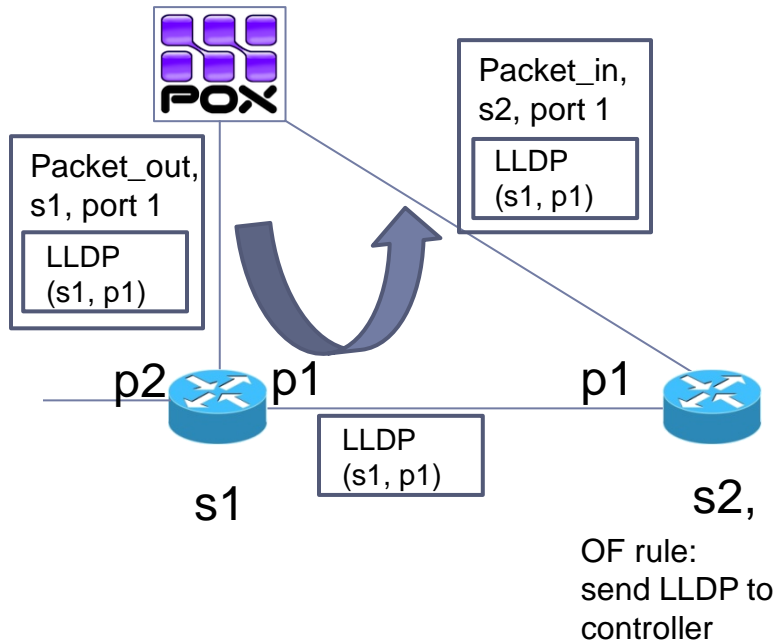
# forwarding.topo\_proactive

---

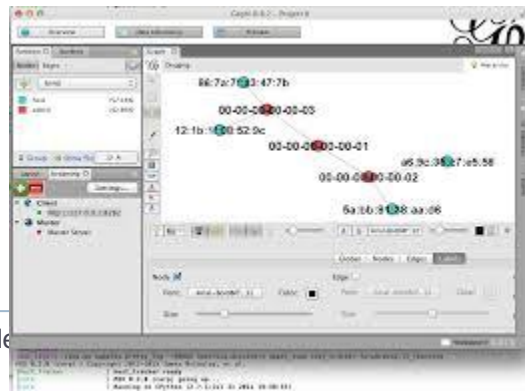
- ▶ A debuggolást megkönnyíti a mininet --mac argumentuma, de a valóságban az IP-mac cím hozzárendelés előre nem ismert.
- ▶ De ha az IP címeket a DHCP szerver szisztematikusan osztja ki, akkor azt figyelembe lehet venni a routingnál.
- ▶ címkiosztási séma: **10.switchID.portNumber.x**
- ▶ Szorgalmi (IMSC) feladat: az implementáció megértése és elmagyarázása

# 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`





# forwarding.topo\_proactive

- ▶ Kis segítség...
- ▶ POX indítása: kell a discovery modul is (ld. előadás 54. fólia)
  - ▶ `pox.py log.color log.level --DEBUG openflow.discovery forwarding.topo_proactive`
- ▶ Mininetben
  - ▶ IP címeket törölni kell a hosztokon, majd dhcp-vel kérni
  - ▶ amit a POX-on futó dhcp szerver oszt ki
  - ▶ `ip addr del 10.0.0.1/8 dev h1-eth0; dhclient -v h1-eth0`
- ▶ bug a POX használt verziójában
  - ▶ `~/pox/pox/proto/arp_helper.py`
  - ▶ egy változó nincs inicializálva
  - ▶ gyors fix (ld. git diff)

```
vagrant@vagrant:~/pox/pox/proto$ git diff
diff --git a/pox/proto/arp_helper.py b/pox/proto/arp_helper.py
index 983dc64..831cf53 100644
--- a/pox/proto/arp_helper.py
+++ b/pox/proto/arp_helper.py
@@ -180,6 +180,7 @@ class ARPHelper (EventMixin):
     self.eat_packets = eat_packets
     self.default_request_src_mac = default_request_src_mac
     self.default_reply_src_mac = default_reply_src_mac
+    self.use_port_mac = False

     def send_arp_request (self, connection, ip, port = of.OFPP_FLOOD,
                           src_mac = _default_mac, src_ip = None):
vagrant@vagrant:~/pox/pox/proto$ █
```