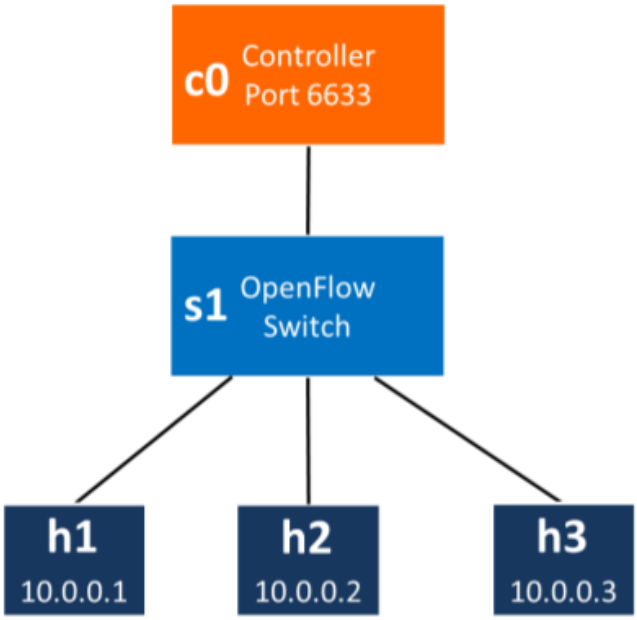
RAZVOJ APLIKACIJA NA POX KONTROLERU

CILJ VJEŽBE

Cilj vježbe je sticanje osnovnih programerskih vještina neophodnih za razvoj mrežnih aplikacija na POX OpenFlow kontroleru. U prvom dijelu vježbe razmatraćemo strukturu elementarnih POX aplikacija: *Hub* i *Layer 2 MAC learning* i testirati ih na virtuelnoj mreži u Mininetu. Drugi dio vježbe podrazumijeva samostalni rad na razvoju *Firewall* aplikacije koja bi trebalo da blokira određeni saobraćaj između dva hosta na osnovu MAC adresa.

PRIPREMA VJEŽBE

Mreža koju ćemo koristiti u vježbi sastoji se od tri hosta i jednog OpenFlow *switch*-a koji je kontrolisan od strane POX kontrolera (Slika 1).



Slika 1: Topologija virtuelne mreže.

POX je *open-source* SDN kontroler napisan u Python programskom jeziku. S obzirom da nećemo koristiti referentnu implementaciju SDN kontrolera koja je po *default*-u aktivna u Mininet emulacijama, potrebno je provjeriti da li je odgovarajući proces aktivan u pozadini:

```
$ ps -A | grep controller
```

Ukoliko je proces aktivan, potrebno ga je isključiti komandom:

```
$ sudo killall controller
```

Takođe, potrebno je pokrenuti:

```
$ sudo mn -c
```

i restartovati Mininet kako bi osigurali da su oslobođeni resursi korišćeni u prethodnim Mininet emulacijama. Zatim, emulacija je pokreće sa:

```
$ sudo mn --topo single,3 --mac --topo ovsk --controller remote
```

U prethodnj komandi opcija `--mac` se koristi za setovanje MAC i IP adresa u mreži na male, jedinstvene i lako-čitljive vrijednosti. Opcijom `--topo` se indicira korišćenje jedne od *default* Mininet topologija (*single* u ovom slučaju). Opcija `--switch` kao argument uzima tip OpenFlow *switch*-a koji će se koristiti u emulacijama, dok se opcijom `--controller remote` nagovještava korišćenje udaljenog kontrolera.

POX kontroler je instaliran na Mininet virtuelnoj mašini. Za pokretanje Hub aplikacije na kontroleru potrebno je ući u **pox** folder gdje je smještena *pox.py* skripta, i zatim ukucati u konzoli:

```
$ pox.py log.level --DEBUG forwarding.hub
```

Na ovaj način POX kontroler startuje *Hub* komponentu (koja je smještena u folderu **~/pox/pox/forwarding**) i komponentu za logovanje. Nakon nekog vremena OpenFlow *switch* se povezuje sa kontrolerom. Kada OpenFlow *switch* izgubi konekciju prema kontroleru pokušaji novog povezivanja vrše se u dužim vremenskim razmacima, maksimalno do 15 sekundi. Ukoliko je potrebno smanjiti ovo kašnjenje, tj. konfigurisati *switch* tako da kašnjenje bude manje od *N* sekundi, potrebno je koristiti `--max-backoff` parametar u gore navednoj komandi. Alternativno rešenje je isključiti Mininet, pokrenuti kontroler, a zatim ponovo pokrenuti virtuelnu topologiju u Mininetu. Kada se *switch* poveže sa kontrolerom, POX će prikazati log sledećeg oblika:

```
INFO:openflow.of_01:[Con 1/1] Connected to 00-00-00-00-00-01
DEBUG:samples.of_tutorial:Controlling [Con 1/1]
```

PROVJERA FUNKCIONALNOSTI HUB KOMPONENTE

Provjeriti da li hostovi mogu međusobno da komuniciraju i da li svi hostovi vide potpuno isti saobraćaj. Sa tim ciljem otvorićemo xterm prozore za svaki host i posmatrati dolazni saobraćaj. Dakle, u Mininet konzoli je potrebno je unijeti:

```
mininet> xterm h1 h2 h3
```

U xterm prozorima koji odgovaraju hostovima h2 I h3 pokrenuti `tcpdump` alat koji prikazuje sve pakete koji se primaju/šalju preko određenog mrežnog interfejsa (argument `tcpdump` komande):

```
# tcpdump -XX -n -i h2-eth0
# tcpdump -XX -n -i h3-eth0
```

U xterm prozoru hosta h1 pokrenuti ping:

```
# ping -c 3 10.0.0.2
```

Ping paketi se sada šalju kontroleru, koji ih dalje prosleđuje preko svih interfejsa *switch*-a izuzev onog koji je primio paket (tzv. Flood akcija). Identični ARP i ICMP paketi trebalo bi da se prikažu u oba xterm prozora gdje je pokrenuta `tcpdump` komanda. Ovo potvrđuje funkcionalnost *Hub* komponente.

Dalje, iz xterm prozora hosta h1 pokrenućemo ping prema nepostojećem hostu:

```
# ping -c 3 10.0.0.5
```

Tri neodgovorena ARP zahtjeva trebalo bi da se prikažu u `tcpdump` prozorima. Zatvoriti xterm prozore nakon ovog eksperimenta.

Kod *Hub* komponente dat je u nastavku:

```
import pox.openflow.libopenflow_01 as of
from pox.core mport core
from pox.lib.util import dpidToStr

log = core.getLogger()

def _handle_ConnectionUp (event):
        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 launch ():
        core.openflow.addListenerByName("ConnectionUp", _handle_ConnectionUp)
```

POX API

POX kontroler čine 3 osnovne komponente:

1. Osluškivači događaja
2. Kontrolna logika
3. Jedinica za razmjenu poruka

Prvo je potrebno definisati tipove događaja koje kontroler treba da osluškuje (npr. *ConnectionUp*, *PacketIn*, itd.). Zatim, koristeći odgovarajuću kontrolnu logiku moguće je diferencirati različite

saobraćajne tokove i sprovoditi različite akcije nad svakim od njih. Konačno, kontroler može da šalje poruke OpenFlow *switch*-evima i popunjava njihove Tabele tokova.

OSLUŠKIVANJE DOGAĐAJA

POX aplikacije se mogu registrovati da osluškuju događaje na dva načina:

- Registracijom odgovarajuće *callback* funkcije za specifične događaje koje oglašavaju različiti moduli kontrolera (npr. *Topology Discovery* ili *core* modul). U *launch* ili *init* dijelu klase potrebno je unijeti:

  core.openflow.addListenerByName("DOGAĐAJ", CALLBACK_FUNKCIJA, PRIORITET)

  Na primjer, ukoliko je već kreirana funkcija *switch()* koja treba da se poziva prilikom povezivanja *switch*-a na kontroler, u *launch* dijelu aplikacije dodaje se osluškivač na *ConnectionUp* događaj:

  core.openflow.addListenerByName("ConnectionUp", switch, 1)

2. U init bloku klase koja želi da osluškuje događaj pozvati *addListeners(self)* metod modula koji generiše događaje. Na primjer:

```
class l2_learning (object):
        def __init__ (self, transparent):
            core.openflow.addListeners(self)

        def _handle_ConnectionUp (self, event):
            log.debug("Connection %s" % (event.connection,))
```

   U ovom slučaju kontroler će automatski prepoznati svaku funkciju naziva *_handle_Događaj(self, event)* kao osluškivač navedenog događaja. Kontrolna logika je uglavnom smještena u ovim funkcijama.

Spisak događaja koje aplikacija može d osluškuje dat je u lijevom dijelu Tabele 1.

Tabela 1: Spisak događaja na koje se POX aplikacije mogu pretplatiti i paketi podržani u POX bibliotekama.

Događaji koji se generišu pilikom prijema poruka	Paketi koji se mogu parsirati
FlowRemoved, ConnectionUp, FeaturesReceived RawStatsReply, PortStatus PacketIn, BarrierIn, SwitchDescReceived, FlowStatsReceived, QueueStatsReceived, AggregateFlowStatsReceived, TableStatsReceived, PortStatsReceived	**arp,dhcp, dns eapol, eap ethernet, icmp igmp, ipv4 llc, lldp, mpls rip, tcp, udp, vlan**

PARSOVANJE PAKETA

POX nudi biblioteke za parsovanje dobro poznatih tipova paketa (Tabela 1). Informacije iz Ethernet paketa koji se primaju tokom *PacketIn* događaja mogu se ekstraktovati na sledeći način:

```
packet = event.parsed
src_mac = packet.src
dst_mac = packet.dst
if packet.type == ethernet.IP_TYPE:
    ipv4_packet = event.parsed.find("ipv4")
    # Dalja obrada IPv4 paketa
    src_ip = ipv4_packet.srcip
    dst_ip = ipv4_packet.dstip
```

POX nudi jednostavan mehanizam za kreiranje *match* dijela zapisa za Tabele tokova OpenFlow *switch*-a. Ukoliko *matching* treba da se odradi nad primljenim paketom, to se postiže sa:

```
match = of.ofp_match.from_packet(packet)
```

slanje poruka switch-evima

Kreiranje i slanje OpenFlow poruka se vrši na jednostavan način kod POX kontrolera. Koraci podrazumijevaju kreiranje odgovarajućeg objekta, popunjavanje njegovih parametara i slanje metodom *send()* objekta konekcije. U nastavku je iskomentarisan primjer iz Hub skripte:

```
msg = of.ofp_packet_out()            # Kreiranje OpenFlow Packet_Out poruke
msg.buffer_id = event.ofp.buffer_id  #Korišćenje primljenog paketa u Packet_Out poruci
msg.in_port = packet_in.in_port      # Setovanje in_port polja na osnovu Packet_In paketa
msg.match = of.ofp_match.from_packet(packet)  #Kreiranje match zapisa za Tabelu tokova

# Dodavanje akije kojom se paket šalje na određeni port ili grupu portova
action = of.ofp_action_output(port = of.OFPP_FLOOD)
msg.actions.append(action)

# Slanje poruke switch-u
self.connection.send(msg)
```

POX API – VIŠE DETALJA

- ***connection.send(...)*** funkcija šalje OpenFlow poruku *switch*-u.
- Kada kontroler uspostavi konekciju sa *switch*-em objavljuje se ***ConnectionUp*** događaj. Funkcija ***_handle_ConnectionUp ()*** obrađuje ovaj događaj.

- ***ofp_packet_out*** OpenFlow poruka se koristi kada kontroler šalje paket podataka *switch*-u, zajedno sa instrukcijom kako taj paket treba biti obrađen. Kontroler može sam kreirati paket ili iskoristiti postojeći koji mu *switch* prethodno proslijedi (npr. *Packet_In* događaj). Paketi koje *switch*-evi pošalju kontroleru se baferuju i mogu se u aplikaciji referencirati sa ***buffer_id*** identifikatorom. Značajna polja ***ofp_packet_out*** objekta su:

  - ***buffer_id*** – Identifikuje baferovani paket koji želimo poslati. Nije potrebno definisati ukoliko kontroler šalje paket koji je sam kreirao.
  - ***data*** – Bajti podataka koji se šalju. Nije potrebno definisati ukoliko se šalje baferovani paket.
  - ***actions*** – Lista akcija koje treba primijeniti nad saobraćajnim tokom.
  - ***in_port*** – U slučaju baferovanog paketa definiše broj porta *switch*-a na koji je paket inicijalno stigao, u suprotnom je *OFPP_NONE*.

  Primjer:

```
msg = of.ofp_packet_out()
msg.in_port = in_port
if buffer_id != -1 and buffer_id is not None:
        msg.buffer_id = buffer_id          #Ukoliko šaljemo baferovani paket
else:
        if raw_data is None:               #Ukoliko paket nije kreiran ne šalji ništa
                return
        msg.data = raw_data
        action = of.ofp_action_output(port = out_port)
        msg.actions.append(action)         #Dodaj akcije i pošalji poruku
        self.connection.send(msg)
```

- ***ofp_flow_mod*** je OpenFlow poruka kojom kontroler instalira novi zapis u Tabeli tokova. Sastoji se od *match*, *actions* i *priority* atributa. Ukoliko saobraćajni tok zadovoljava više zapisa u Tabeli tokova, primijeniće se zapis najvećeg prioriteta.

```
fm = of.ofp_flow_mod()
match=of.ofp_match()
match.in_port=3
fm.match=match
fm.actions.append(of.ofp_action_output(port = 4))
```

- ***ofp_action_output*** klasa se koristi prilikom slanja ***ofp_packet_out*** i ***ofp_flow_mod*** poruka. Definiše broj porta *switch*-a na koji treba poslati paket. Takođe, konstruktor ove klase može uzimati razne "specijalne" brojeve porta kao argument. Na primjer, *OFPP_FLOOD* se koristi kao indikator da paket treba poslati preko svih portova izuzev onog preko kojeg je paket inicijalno primljen:

```
out_action = of.ofp_action_output(port = of.OFPP_FLOOD)
```

- **ofp_match** klasa opisuje polja u zaglavlju paketa i ulazni port koje će *switch* analizirati u cilju identifikovanja saobraćajnog toka. Sva polja su opciona – ona koja se eksplicitno ne specificiraju su "*wildecard*", što znači da prihvataju bilo koju vrijednost. Neka od značajnih *match* polja su:

  dl_src – Izvorišna MAC adresa
  dl_dst – Destinaciona MAC adresa
  dl_type – Tip Ethernet paketa (npr. 0x800 za IP, 0x806 za ARP)
  nw_src – Izvorišna IP adresa
  nw_dst – Destinaciona IP adresa
  in_port – Ulazni port switch-a

  Primjer: Kreirati *match* zapis koji će se odnosti na sve pakete koji pristižu na portu 3:

```
match = of.ofp_match()
match.in_port = 3
```

L2 LEARNING SWITCH APLIKACIJA

Privremeno isključite POX kontroler sa CTRL-C. Zatim pokrenite *l2_learning* aplikaciju sa:

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

Kao i prije, kreiraćemo xterm prozor za svaki host kako bismo posmatrali saobraćaj:

```
mininet> xterm h1 h2 h3
```

U xterm prozorima koji odgovaraju hostovima h2 i h3 pokrenite `tcpdump` komandu:

```
# tcpdump -XX -n -i h2-eth0
```

```
# tcpdump -XX -n -i h3-eth0
```

U xterm prozoru hosta h1 pokrenućemo ping prema h2:

```
# ping -c 1 10.0.0.2
```

Pokrenuta aplikacija ispituje pakete i uči na koje portove su povezane koje MAC adrese (tzv.port-adresa mapiranje). Ukoliko je destinaciona adresa paketa već pridružena nekom portu *switch*-a, paket

će se proslijediti na taj port. U suprotnom, paket će se poslati na sve portove *switch*-a izuzev na onaj preko kojeg je primljen.

ZADATAK ZA SAMOSTALNI RAD

Kreirati *layer-2 firewall* aplikaciju na POX kontroleru. Mrežna topologija ne bi trebalo da ima uticaj na funkcionisanje aplikacije. Neka aplikacija uzima kao ulazni argument parove MAC adresa između kojih treba blokirati saobraćaj (ACL – Access Control List). Uraditi jednostavno, iako manje efikasno, rešenje koja podrazumijeva instalaciju novih zapisa u Tabelama tokova svih *switch*-eva u mreži.

Napomena: POX dozvoljava paralelno izvršavanje većeg broja aplikacija. Stoga, moguće je da različite aplikacije pošalju konfliktna pravila *switch*-evima. Npr. dvije aplikacije mogu pokušati da instaliraju pravila sa istim src/dst MAC adresama, istog prioriteta, ali različitih akcija. Najjednostavniji način da se izbjegne ova situacija je dodijeliti aplikacijama različite prioritete.

Koristiti fajlove sa sledećeg linka kao osnovu za izradu zadatka:
https://drive.google.com/file/d/149_SFBWILOtjRMr18Hw91L_QxyiEaKn4/view?usp=sharing

- Fajl *firewall-policies.csv* sadrži listu parova MAC adresa između kojih treba blokirati saobraćaj (čitati ovaj fajl iz aplikacije).

- Fajl *firewall.py* iskorisiti kao templejt koji je potrebno nadograditi odgovarajućom kontrolnom logikom. Ovaj fajl sadrži *firewall* klasu sa *_handle_ConnectionUp* funkcijom. Takođe, sadži i globalnu varijablu policyFile koja referencira fajl Funkcija *_handle_ConnectionUp* poziva se svaki put kada se neki *switch* poveže na kontroler. Vaš zadatak je da dopunite ovu funkciju.

TESTIRANJE RAZVIJENE APLIKACIJE

Kada kreirate potrebni kod, prekopirajte fajl *firewall.py* u folder **~/pox/pox/misc**. U istom folderu prekopirati *firewall-policies.csv* fajl. U *firewall-policies.csv* upišite:

```
id,mac_0,mac_1
1,00:00:00:00:00:01,00:00:00:00:00:02
```

Ovaj sadržaj ukazuje da je potrebno blokirati komunikaciju između h1 i h2. Pokrenite POX kontroler:

```
$ cd ~
$ pox.py forwarding.l2_learning misc.firewall &
```

Pokrenuće se i *l2_learning* i *firewall* aplikacija. Sada, pokrenite Mininet:

```
$ sudo mn --topo single,3 --controller remote --mac
```

U Mininetu pokušajte da pingujete host h2 sa hosta h1.

```
mininet> h1 ping -c 1 h2
```

Ukoliko ste zadatak ispravno uradili ping neće raditi. Na kraju, pingujte h3 sa h1:

```
mininet> h1 ping -c 1 h3
```

Ovoga puta ping bi trebao da radi.

LITERATURA

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