Napotkane problemy

Początkowo instalacja wykonywana była na Ubuntu 22.04, które domyślnie ma Python w wersji 3.10. Ryu jest już w stanie depracated i taka wersja była dla niego zbyt duża. Instalacje powtórzono na Ubuntu 20.04, na którym domyślny Python to już 3.8.10. Instalacja przebiegła bez problemu.

Wykonane czynności

1. Setup

Instalacja ryu

Instalacja wg https://github.com/faucetsdn/ryu?tab=readme-ov-file w opcji "z kodu źródłowego". Test instalacji za pomocą komedny:

```
ryu-manager --version
```

oraz napisanie prostej apki

```
from ryu.base import app_manager

class L2Switch(app_manager.RyuApp):
    def __init__(self, *args, **kwargs):
        super(L2Switch, self).__init__(*args, **kwargs)
```

I uruchomienie jej:

```
ryu-manager yourapp.py
```

Instalacja mininet

```
sudo apt update
sudo apt upgrade -y
sudo apt install mininet -y
sudo mn --test pingall
```

2. Testowe połączenie mininet i Ryu

Na początek uruchomiono najprostszą topologie sieci mininet podłączając do niej kontroler ryu:

```
sudo mn --controller=remote,ip=127.0.0.1,port=6633 --switch
ovsk,protocols=OpenFlow13
```

Spośród aplikacji Ryu wybrano taki kontroler SDN, który będzie sterował węzły sieci jako zwykłe switche L3.

```
ryu-manager ryu.app.simple_switch_13
```

13 - ta aplikacja jest dostosowana do OpenFlow13

W konsoli mininet wykonałem test:

```
mininet> pingall
```

który zakończył się sukcesem (co ważne wykonanie takigo testu przed włączeniem Ryu kończy się niepowodzeniem).

3. Stworzenie Topologii

Stworzono skrypt w Python, który tworzy w mininet sieć o zadanej topologii. Skrypt w pliku topo.py.

Uruchomienie jej wraz z kontrolerem bez zmian kończy się tym, że nie jesteśmy w stanie spingować hostów. Ryu w logach dostaje non stop jakieś pakiety co może wskazywać na to, iż kontroler simple_switch_13 jest zbyt prosty i powstają pętle rutingowe. W tym celu apkę kontrolera podmienimy na `simple_switch_stp_13.py`.

```
ryu-manager ryu.app.simple_switch_stp_13
```

W tym przypadku pingi między hostami już działają (należy odczekać aż kontroler wykona algorytm Spanning Tree Protocol).

4. Host H1 i H2 w różnych podsieciach

Plik z nową topologią (taką gdzie h1 i h2 są w różnych sieciach) jest tutaj.

Zmiany dotknęły linijki numer 12 i 13, które teraz wyglądają tak:

```
        h1 = self.addHost('h1', ip='192.168.1.1/24')
        h2 = self.addHost('h2', ip='192.168.2.1/24')
```

Jednakże tym razem ping pomiędzy hostami nie działa.

```
mininet> h1 ping h2
ping: connect: Network is unreachable
```

Nic dziwnego, inne podsieci więc urządzenia SDN, muszą być czymś co pracuje na L3. Szukam więc innej aplikacji ryu. Niestety nie udało się takowej znaleźć.

Odpowiedzi na pytania

Jaką aplikację należy uruchomić na sterwniku Ryu? Czy wymaga ona jakiś zmian w topologii sieci? Aplikacja jakiej z powodzeniem używałem bylo simple_switch_stp_13, czyli Switch obsługujący OpenFlow w wersji 1.3 oraz potrafiący "ogarnąć" nieco bardziej skomplikowane sieci dzięki protokołowi STP.

Jaka jest używana wersja protokołu OpenFlow? 1.3

Kod źródłowy

Topologia 1 (h1 i h2 w tej samej sieci)

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
from mininet.node import Controller, RemoteController
from mininet.cli import CLI
from mininet.link import TCLink

class CustomTopology(Topo):
    def build(self):
        # Add hosts and switches
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')
        s5 = self.addSwitch('s5')

        # Add links with the respective delays
        self.addLink(h1, s1, delay='5ms')
        self.addLink(s1, s2, delay='2ms')
        self.addLink(s1, s3, delay='5ms')
        self.addLink(s1, s4, delay='5ms')
        self.addLink(s2, s3, delay='10ms')
        self.addLink(s3, s4, delay='5ms')
        self.addLink(s4, s5, delay='5ms')
        self.addLink(s2, s5, delay='3ms')
        self.addLink(s3, s5, delay='5ms')
        self.addLink(h2, s5, delay='5ms')
```

```
def runCustomTopo():
    # Create and start the network
    topo = CustomTopology()
    net = Mininet(topo=topo, link=TCLink, controller=lambda name:
RemoteController(name, ip='127.0.0.1'))
    net.start()

    # Dump host connections
    dumpNodeConnections(net.hosts)

    # Start CLI
    CLI(net)

    # Stop the network
    net.stop()

# This is to avoid running the topology if this script is imported elsewhere
if __name__ == '__main__':
    # Tell mininet to print useful information
    setLogLevel('info')
    runCustomTopo()
```

Topologia 2 (h1 i h2 w różnych sieciach):

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel
from mininet.node import Controller, RemoteController
from mininet.cli import CLI
from mininet.link import TCLink

class CustomTopology(Topo):
    def build(self):
        # Add hosts and switches
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')
        s5 = self.addSwitch('s5')

        # Add links with the respective delays
        self.addLink(h1, s1, delay='5ms')
        self.addLink(s1, s2, delay='2ms')
        self.addLink(s1, s3, delay='5ms')
        self.addLink(s1, s4, delay='5ms')
        self.addLink(s2, s3, delay='10ms')
        self.addLink(s3, s4, delay='5ms')
        self.addLink(s4, s5, delay='5ms')
```

```
        self.addLink(s2, s5, delay='3ms')
        self.addLink(s3, s5, delay='5ms')
        self.addLink(h2, s5, delay='5ms')

def runCustomTopo():
    # Create and start the network
    topo = CustomTopology()
    net = Mininet(topo=topo, link=TCLink, controller=lambda name:
RemoteController(name, ip='127.0.0.1'))
    net.start()

    # Dump host connections
    dumpNodeConnections(net.hosts)

    # Start CLI
    CLI(net)

    # Stop the network
    net.stop()

# This is to avoid running the topology if this script is imported elsewhere
if __name__ == '__main__':
    # Tell mininet to print useful information
    setLogLevel('info')
    runCustomTopo()
```