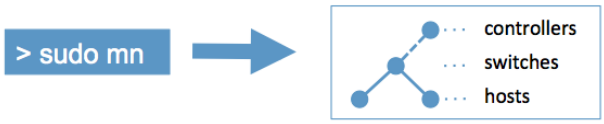
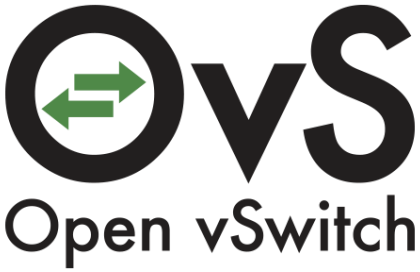
SDN Mininet Topologie





Wolfgang Hurst

1 Einleitung

1.1 Dieses Dokument und das Ziel

In diesem Dokument erstellen wir mit Mininet Netzwerk Topologien und benutzen ONOS als Controller.

1.2 Voraussetzungen

Wir benötigen eine

- Mininet Umgebung => „SDN Mininet Installation“
- Laufende ONOS Umgebung => „SDN ONOS Developer Installation“

2 Einleitung Mininet

2.1 Grundlagen

Mininet ist eine Software mit der man Netzwerke simulieren kann. Dabei kommen virtuelle Switche und virtuelle Hosts ins Spiel. Mininet verwaltet diese und gibt uns eine CLI um das Ganze zu kontrollieren.

Mininet benutzt unter anderem eine OpenVSwitch Implementation. Das OpenVSwitch kann Ports eines Switches als Interfaces im Linux Kernel ablegen. Dadurch kann man dann später auf diesen Interfaces dann zum Beispiel `tcpdump`, oder `dhcp` oder `ppp` machen. Das ist eine ziemlich coole Geschichte, weil man alles an einem Ort hat.

Wenn man bei Mininet einen Controller angibt, schaltet es automatisch die ganzen Switch mit diesem Controller zusammen und muss sich darüber keine Sorgen mehr machen. Mininet ist im Vergleich zu einem echten Real Netzwerk ein echter Gewinn und man kann damit extrem schnell Sachen testen und ausprobieren.

Mininet selbst at bereits ein paar Topologien mit ein paar Switchen und Hosts dabei. Wenn man jedoch was Eigenes bauen will, bietet Mininet eine Python Schnittstelle an, mit der man das machen kann was man will.

Und genau das werden wir tun.

2.2 Der erste Start

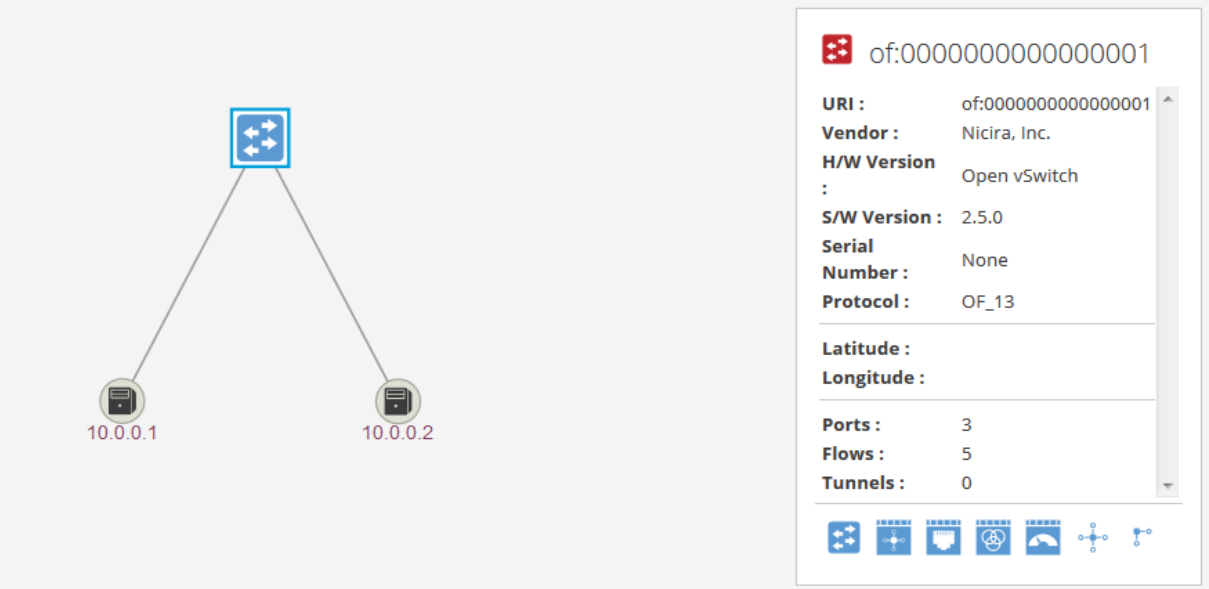
Wir loggen uns auf der Mininet Maschine ein und starten Mininet und übergeben ihn die IP Adresse der ONOS Maschine.

```
sudo mn --controller=remote,ip=10.0.0.194
```

Als Ergebnis sieht man:

```
whurst@sdn-mininet-01:~$ sudo mn --controller=remote,ip=10.0.0.194
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=16935>
<Host h2: h2-eth0:10.0.0.2 pid=16938>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=16944>
<RemoteController{'ip': '10.0.0.194'} c0: 10.0.0.194:6633 pid=16929>
mininet>
```

Man sieht er hat zwei Hosts (h1 und h2) erstellt, ein Switch (s1). Das ganze kann man sich jetzt bereits im ONOS Weg-UI anschauen



Das war einfach …

3 Topologien via Python

3.1 Erste Schritte – Rumpf Datei

Man kann eine Topologie mit einem Python Script erstellen. Dazu erstellt man eine Python Datei, importiert die Klassen die wir brauchen. Erstellen eine Klasse die Mininet benutzen kann und geben am Ende die Topologie dem Mininet bekannt, damit er das Ding findet.

Hier ein Rumpf den wir brauchen, der zwar keine Hosts, Switche, Links erstellt, aber den Rumpf können wir immer weiter benutzen. Hier die Datei `rumpf.py`

```
# Rumpf Template fuer Mininet Topologien

from mininet.topo import Topo

# die Klasse mit unserer topologie
class MyRumpf (Topo):

        # der contructor der klasse
        def __init__ (self):

                # Topologie Initialisieren
                Topo.__init__ (self)

# hinzufuegen der Topologie mit Namen und Klasse damit Mininet es findet
topos = { 'rumpf': ( lambda: MyRumpf() ) }
```

Damit Mininet die Datei und Topologie auch findet, muss man es ihm sagen.

Mit der Option `--custom` gibt man dem Mininet eine Datei mit die es als Konfiguration benutzt.

Mit der Option `--topo` gibt man den Namen der Topologie an die man benutzten will.

```
sudo mn --custom=rumpf.py --topo=rumpf --controller=remote,ip=10.0.0.194
```

Man sieht das Mininet nun keinen Switch und Host erstellt hat. So wie wir es gesagt haben.

Weiterhin kopieren wir diese Datei jetzt unter dem Namen `babystep` und nennen unsere Topologie in `BabyStep` um.

3.2 Grundelemente erstellen

3.2.1 Switch

Um einen Switch zu erstellen brauchen wir eine Variable für das Objekt, einen Namen und die Switch ID. Bei der Switch ID ist Vorsicht angesagt, Mininet vergibt automatische Switch IDs, wenn man keine Angibt, anhand des Namens. Das geht dann in die Hose wenn man ein Switch „frankfurt1“ und ein anderen „berlin1“ nennt. Mininet vergibt beiden die Switch ID 1, was nicht so gut ist.

Zum Hinzufügen eines Switches wird die Funktion `addSwitch` verwendet.

```
from mininet.topo import Topo
class classBabyStep (Topo):
        def __init__ (self):
                Topo.__init__ (self)

                MeinErsterSwitch = self.addSwitch ('switch1', dpid="880011");

topos = { 'BabyStep': ( lambda: classBabyStep() ) }
```

Danach starten wir Mininet und sehen auch im ONOS unseren Switch

```
sudo mn --custom=babystep.py --topo=BabyStep --controller=remote,ip=10.0.0.194
```

3.2.2 Hosts

Ein Host zu erstellen ist ähnlich simpel. Da wir jedoch die Funktion `addHost` benutzt.

```
                MeinErsterHost = self.addHost ('host1');
                MeinZweiterHost = self.addHost ('host2');
```

3.2.3 Link

Jetzt will man den Host mit einem Switch verbinden. Oder ein Switch mit einem Switch. Dazu wird die Funktion `addLink` benötigt. Mit dieser Funktion legt man virtuell ein Kabel zwischen den beiden.

Die Funktion braucht nur zwei Parameter, Mininet nimmt dann einfach den nächsten freien Port, oder erfindet kurzerhand einen neuen Port, um die beiden Geräte zu verbinden. Das ist zwar nett, aber wenn man das Debuggen will, bekommt man ein Problem, weil man kaum noch weiß was wo angeschlossen ist. Meine Empfehlung daher ist immer die Ports mit anzugeben, wie an einem richtigen Switch auch.

```
                self.addLink (MeinErsterSwitch, MeinErsterHost, 1, 1);
                self.addLink (MeinErsterSwitch, MeinZweiterHost, 2, 1);
```

Wenn die Hosts im ONOS nicht angezeigt werden, liegt das daran das ONOS die Hosts noch nicht kennt. Wenn man im Mininet den Befehl `pingall` absetzt versucht Mininet alles Hosts gegenseitig zu Pingen, was dazu führt das ONOS alle Hosts kennenlernt.

3.2.4 Übersicht

Jetzt hat man die babystep.py Datei

```
from mininet.topo import Topo
class classBabyStep (Topo):
        def __init__ (self):
                Topo.__init__ (self)

                # ein Switch
                MeinErsterSwitch = self.addSwitch ('switch1', dpid="880011");

                # zwei Hosts
                MeinErsterHost = self.addHost ('host1');
                MeinZweiterHost = self.addHost ('host2');

                # Verbindung
                self.addLink (MeinErsterSwitch, MeinErsterHost, 1, 1);
                self.addLink (MeinErsterSwitch, MeinZweiterHost, 2, 1);

topos = { 'BabyStep': ( lambda: classBabyStep() ) }
```

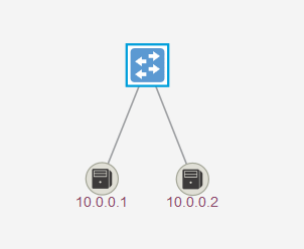
Einen gestarteten Mininet

```
sudo mn --custom=babystep.py --topo=BabyStep --controller=remote,ip=10.0.0.194
```

Das Ergebnis von Mininet

```
*** Creating network
*** Adding controller
*** Adding hosts:
host1 host2
*** Adding switches:
switch1
*** Adding links:
(switch1, host1) (switch1, host2)
*** Configuring hosts
host1 host2
*** Starting controller
c0
*** Starting 1 switches
switch1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
host1 -> host2
host2 -> host1
*** Results: 0% dropped (2/2 received)
```

Und im ONOS sieht man



Alles perfekt!

3.3 Routing

Mit Mininet kann man auch Routing simulieren. Zum Einsatz kommt dabei ein Host der dann entsprechend als Router konfiguriert werden kann. Wir wollen ein kleines Beispiel machen.

Es gibt zwei Netzwerke. Links Netzwerk mit der 172.16.64.0/24 und ein rechtes Netzwerk mit der 192.168.128.0/24. In beiden Netzwerken ist ein Switch der jeweils am Port 1 mit dem Router verbunden ist. Ziel ist es nun das die Hosts sich über den Router hinweg pingen können.

3.3.1 Basis Script

Wir erstellen erst einmal grob alle Geräte und Links die wir brauchen.

```
# -*- coding: utf-8 -*-
from mininet.topo import Topo
class classSimpleRouter (Topo):
        def __init__ (self):
                Topo.__init__ (self)

                # zwei linke hosts
                lh1 = self.addHost ('lh1', ip='172.16.64.11/24');
                lh2 = self.addHost ('lh2', ip='172.16.64.12/24');

                # der linke switche
                lsw = self.addSwitch ('lsw1', dpid="172001");

                # die verbindungen auf der linken seite
                self.addLink (lsw, lh1, 11, 1);
                self.addLink (lsw, lh2, 12, 1);

                # das gleiche für die rechte seite ... die zwei hosts
                rh1 = self.addHost ('rh1', ip='192.168.128.21/24');
                rh2 = self.addHost ('rh2', ip='192.168.128.22/24');

                # der rechte switche
                rsw = self.addSwitch ('rsw1', dpid="192001");

                # die verbindungen auf der rechten seite
                self.addLink (rsw, rh1, 11, 1);
                self.addLink (rsw, rh2, 12, 1);

                # nun ein Host als Router erstellen
                router = self.addHost ('router');

                # den Router mit beiden Switchen verbinden
                self.addLink (router, lsw, 1, 1);
                self.addLink (router, rsw, 2, 1);

topos = { 'SimpleRouter': ( lambda: classSimpleRouter() ) }
```

Man sieht jetzt zwei Hosts auf beiden Seiten mit einer IP konfiguriert und ein Router zwischen den beiden Switchen. Startet man das wird das natürlich nicht funktionieren.

3.3.2 IP Adressen pro Link

Der Router braucht auf den beiden Links die Information welche IP Adresse er verwenden soll. Das fehlt noch. Wir ändern also unseren Router so dass er auf beiden Seiten eine IP bekommt

```
router = self.addHost ('router', ip='172.16.64.1/24');

# den Router mit beiden Switchen verbinden
self.addLink (router, lsw, 1, 1, intfName1='router-eth1', params1={'ip':'172.16.64.1/24'});
self.addLink (router, rsw, 2, 1, intfName1='router-eth2', params1={'ip':'192.168.128.1/24'});
```

Die IP Adresse der erste Anweisung von `addLink` wird leider von Mininet ignoriert und durch die Default IP ersetzt. Deswegen ist im Host die IP Adresse noch einmal definiert. Ansonsten bekommt man auf dem Link eine 10'er IP.

Wenn man nun Mininet startet und ein Ping auf alle Kisten absetzt, sieht man dass das vorerst gut aussieht.

```
mininet> pingall
*** Ping: testing ping reachability
lh1 -> lh2 X X router
lh2 -> lh1 X X router
rh1 -> X X rh2 X
rh2 -> X X rh1 X
router -> lh1 lh2 rh1 rh2
*** Results: 50% dropped (10/20 received)
mininet> dump
<Host lh1: lh1-eth1:172.16.64.11 pid=20213>
<Host lh2: lh2-eth1:172.16.64.12 pid=20216>
<Host rh1: rh1-eth1:192.168.128.21 pid=20219>
<Host rh2: rh2-eth1:192.168.128.22 pid=20222>
<Host router: router-eth1:172.16.64.1,router-eth2:192.168.128.1 pid=20225>
<OVSSwitch lsw1: lo:127.0.0.1,lsw1-eth1:None,lsw1-eth11:None,lsw1-eth12:None pid=20231>
<OVSSwitch rsw1: lo:127.0.0.1,rsw1-eth1:None,rsw1-eth11:None,rsw1-eth12:None pid=20234>
<RemoteController{'ip': '10.0.0.194'} c0: 10.0.0.194:6633 pid=20207>
```

3.3.3 Default Route auf den Hosts

Nun schauen wir uns einen Host an. Dort stellen wir fest, dass er, selbst wenn, der Router gehen würde, nicht in das andere Netz käme. So ohne Route ist das schlecht. Man kann dem Host im Mininet lokal eine Route verpassen, so wie man das auch auf einem echten Host machen würde.

```
mininet> lh1 ip r
172.16.64.0/24 dev lh1-eth1  proto kernel  scope link  src 172.16.64.11
mininet> lh2 ip route add default via 172.16.64.1
mininet> lh2 ip route list
default via 172.16.64.1 dev lh2-eth1
172.16.64.0/24 dev lh2-eth1  proto kernel  scope link  src 172.16.64.12
```

Das ist zwar nett, aber wenn wir Mininet neu starten ist wieder alles weg. Der zweite Weg ist das direkt in unsere Topologie zu schreiben. Und genau das tun wir auch

```
lh1 = self.addHost ('lh1', ip='172.16.64.11/24', defaultRoute='via 172.16.64.1');
lh2 = self.addHost ('lh2', ip='172.16.64.12/24', defaultRoute='via 172.16.64.1');

rh1 = self.addHost ('rh1', ip='192.168.128.21/24', defaultRoute='via 192.168.128.1');
rh2 = self.addHost ('rh2', ip='192.168.128.22/24', defaultRoute='via 192.168.128.1');
```

3.3.4 Routing auf dem Router

Das was jetzt noch fehlt ist, das der Router auch das tut was er soll, nämlich Routen. Da gibt es jetzt zwei Wege. Zum einen kann man im Mininet auf dem Host das Routing einschalten, oder man definiert das im Topologie Script.

Damit der Host Routen kann muss man das IP Forwarding einschalten. Das geht mit

```
sysctl net.ipv4.ip_forward=1
```

Nimmt man den manuellen Weg, dann sieht das in etwa so aus

```
mininet> router sysctl net.ipv4.ip_forward=1
```

```
net.ipv4.ip_forward = 1
mininet> pingall
*** Ping: testing ping reachability
lh1 -> lh2 rh1 rh2 router
lh2 -> lh1 rh1 rh2 router
rh1 -> lh1 lh2 rh2 router
rh2 -> lh1 lh2 rh1 router
router -> lh1 lh2 rh1 rh2
*** Results: 0% dropped (20/20 received)
```

Wie man sieht funktioniert das. Der Router leitet die Pakete weiter. Alle Hosts haben den Router als Gateway. Alles Perfekt und Super. Aber, wenn man alles neu startet ist dieses Forwarding weg.

Es gibt jetzt keinen Weg bei zum Beispiel dem `addHost` im Script ein solchen Parameter mit zu geben. Aber es gibt eine Möglichkeit bei einem `addHost` eine Klasse anzugeben, die benutzt werden soll, wenn der Host erstellt wird. Die Idee ist nun einfach die, dass man das Forwarding in dieser Klasse definiert, wenn der Host hochfährt. Genau das tun wir auch

Wir bauen unsere eigene Host Klasse

```
from mininet.node import Node

class classRouter (Node):
    def config (self, **params):
        super (classRouter, self).config (**params)

        # Routing einschalten nachdem der Host hochgefahren ist
        self.cmd ('sysctl net.ipv4.ip_forward=1')

    def terminate (self):

        # Routing wieder abschalten bevor der Host runterfährt
        self.cmd ('sysctl net.ipv4.ip_forward=0')

        super (classRouter, self).terminate()
```

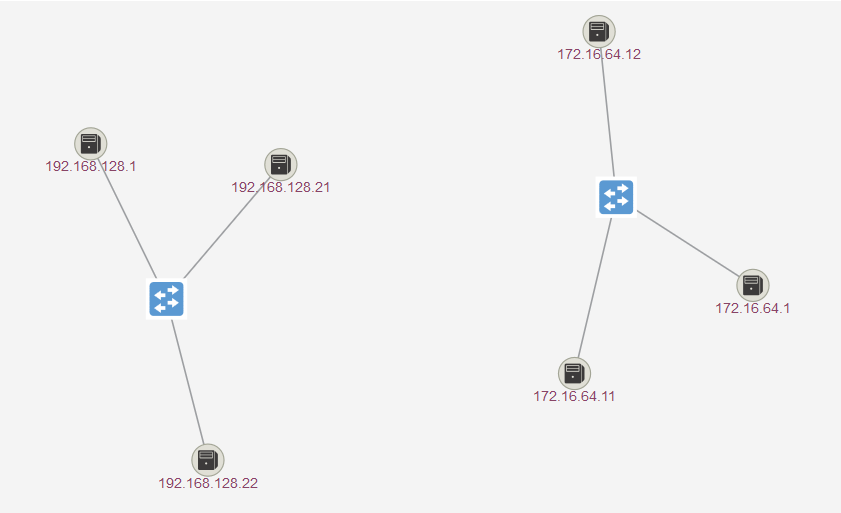
Wir ändern das hinzufügen des Hosts indem wir dem Mininet mitteilen, dass es jetzt unsere Klasse verwenden soll.

```
                router = self.addHost ('router', cls=classRouter, ip='172.16.64.1/24');
```

Und wenn man jetzt Mininet startet, ist das gesamte Netzwerk inklusive Routen einsatzfähig.

3.3.5 ONOS Darstellung

Ein Problem gibt es noch. Schaut man sich das im ONOS an, könnte man meinen dass da was nicht stimmt.



Wie man sieht erkennt er den Router als zwei verschiedene Hosts. Das ist zum einen auch nicht ganz falsch. ONOS betrachtet dabei Broadcast Domains und keine Routing Domains. Die Darstellung von ONOS ist zwar für uns Menschen falsch, aber für SDN ist es korrekt. Man kann nämlich unmöglich zwischen diesen beiden Netzwerken irgendeine SDN Technologie verwenden. Es ist ein klassischer Router, der kennt SDN nicht …

Jetzt denk der Leser … toll … habe jetzt 231 Seiten gelesen für eine Technik die nicht funktioniert und vollkommen nutzlos ist? … Ähm … Ja … genau das haben Sie gerade getan …

Keine Sorge … wir werden diese Technik mit den Host Klassen und dem Routing später brauchen, wenn man zum Beispiel oben ein BGP Router hat und unten ein BGP Router und dazwischen SDN. Spätestens dann wenn das ONOS in das Routing einsteigt um im SDN Pfade auf Grundlage von Routing Informationen zu erstellen, spätestens dann braucht man die Mininet Host Klassen und die Grundlagen.

3.3.6 Zusammenfassung

Mininet Topologie Script mit dem Namen `simpleRouter.py`

```
# -*- coding: utf-8 -*-
from mininet.topo import Topo
from mininet.node import Node

class classRouter (Node):
    def config (self, **params):
        super (classRouter, self).config (**params)

      # Routing einschalten nachdem der Host hochgefahren ist
        self.cmd ('sysctl net.ipv4.ip_forward=1')

    def terminate (self):

      # Routing wieder abschalten bevor der Host runterfährt
        self.cmd ('sysctl net.ipv4.ip_forward=0')

        super (classRouter, self).terminate()

class classSimpleRouter (Topo):
     def __init__ (self):
              Topo.__init__ (self)

              # zwei linke hosts
              lh1 = self.addHost ('lh1', ip='172.16.64.11/24', defaultRoute='via 172.16.64.1');
              lh2 = self.addHost ('lh2', ip='172.16.64.12/24', defaultRoute='via 172.16.64.1');

              # der linke switche
              lsw = self.addSwitch ('lsw1', dpid="172001");

              # die verbindungen auf der linken seite
              self.addLink (lsw, lh1, 11, 1);
              self.addLink (lsw, lh2, 12, 1);

              # das gleiche für die rechte seite ... die zwei hosts
              rh1 = self.addHost ('rh1', ip='192.168.128.21/24', defaultRoute='via 192.168.128.1');
              rh2 = self.addHost ('rh2', ip='192.168.128.22/24', defaultRoute='via 192.168.128.1');

              # der rechte switche
              rsw = self.addSwitch ('rsw1', dpid="192001");

              # die verbindungen auf der rechten seite
              self.addLink (rsw, rh1, 11, 1);
              self.addLink (rsw, rh2, 12, 1);

              # nun ein Host als Router erstellen
              router = self.addHost ('router', cls=classRouter, ip='172.16.64.1/24');

              # den Router mit beiden Switchen verbinden
              self.addLink (router, lsw, 1, 1, intfName1='router-eth1', params1={'ip':'172.16.64.1/24'});
              self.addLink (router, rsw, 2, 1, intfName1='router-eth2', params1={'ip':'192.168.128.1/24'});

topos = { 'SimpleRouter': ( lambda: classSimpleRouter() ) }
```

Starten und Pingen im Mininet

```
$ sudo mn --custom=simpleRouter.py --topo=SimpleRouter --controller=remote,ip=10.0.0.194
*** Creating network
*** Adding controller
*** Adding hosts:
lh1 lh2 rh1 rh2 router
*** Adding switches:
lsw1 rsw1
*** Adding links:
(lsw1, lh1) (lsw1, lh2) (router, lsw1) (router, rsw1) (rsw1, rh1) (rsw1, rh2)
*** Configuring hosts
lh1 lh2 rh1 rh2 router
*** Starting controller
c0
*** Starting 2 switches
lsw1 rsw1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
lh1 -> lh2 rh1 rh2 router
lh2 -> lh1 rh1 rh2 router
rh1 -> lh1 lh2 rh2 router
rh2 -> lh1 lh2 rh1 router
router -> lh1 lh2 rh1 rh2
*** Results: 0% dropped (20/20 received)
mininet>
```

Ansicht aus dem ONOS