

# SDN Yourself!

**Grijp nu je kans om hands-on ervaring op te doen met SDN! Hier volgt een tutorial die je kennis laat maken met SDN, de ONOS-controller, Mininet en Intent Based Networking, zodat je beter kunt voorstellen waar we nu eigenlijk allemaal mee bezig zijn.**

## Voordat we beginnen

In de Technology Radar, gepresenteerd in de deze [blog](#), komen veel nieuwe technologieën langs die veel mensen nog moeilijk voor zich kunnen zien. SDN, ONOS, Intent Based Networking, P4 en Whiteboxing zijn allemaal termen die ik tegenkom binnen ons huidige project. Omdat ik weet hoe vervelend het is als je iets niet goed voor je kunt zien, deel ik graag deze tutorial voor wat hands-on ervaring. Na het volgen van de tutorial zou je in staat moeten zijn met Mininet een gevirtualiseerd netwerk op te zetten dat aangestuurd gaat worden door de ONOS (SDN)-controller. Op deze controller kan je een specifieke applicatie installeren die door middel van 'Intents' het verkeer door het netwerk stuurt. Als laatst plaatsen we alles nog in de juiste context.

Je hoeft geen netwerk expert te zijn om dit te kunnen volgen. Wel ga ik ervan uit dat het je lukt om een VM te importeren in VirtualBox en dat je enig gevoel hebt voor terminal commando's - wees vooral niet bang om random dingen te proberen en dingen op te zoeken. Het Internet is je vriend! Als je de tutorial echt wil uitvoeren en nog meer wil spelen met SDN, neem daar dan vooral *ruim* de tijd voor - het is ook mogelijk de tutorial in etappes te doen. Mocht je er (nu) geen tijd voor hebben, dan zou alleen het lezen je ook al enig beeld moeten kunnen geven.

## Kort over Software Defined Networking (SDN)

SDN heeft binnen verschillende bedrijfstakken een verschillende definitie gekregen. Hier wordt met SDN bedoeld op het principe dat binnen computernetwerken de *controle laag* en *forwarding laag* fysiek van elkaar gescheiden zijn. In het kort betekent dit dat switches heel 'domme' apparaten worden die alleen maar pakketten kunnen doorsturen. Hoe ze dat moeten doen, lezen ze af in een zogenaamde *flow table*. Deze tabel wordt gevuld door de SDN-controller, die al het rekenwerk op zich neemt en daardoor het 'brein' van het netwerk vormt. Het protocol OpenFlow wordt gebruikt voor alle communicatie tussen de switches en controller die te maken heeft met het forwarding gedrag van de switches.

## De Voorbereiding

Deze tutorial is het beste uit te voeren met gebruik van de VM die te downloaden is op deze [site](#). Pak de 64-bit versie en zorg ervoor dat als je deze VM importeert in VirtualBox, dat je hem (minimaal) 2GB geheugen geeft. Op deze VM staat Mininet en de packet-sniffer Wireshark geïnstalleerd, en ook de basis voor verschillende SDN-controllers (zoals Ryu en OpenDaylight). Dus, als je na deze tutorial nog meer wil uitproberen, dan is deze VM uitermate geschikt!

**Let op:** Als je niet direct het bureaublad te zien krijgt, de inloggegevens zijn `'ubuntu'/'ubuntu'`.

**Belangrijk:** Voor deze tutorial heb je ook twee extra bestanden nodig (*triangle.py* en *onos-app-ifwd-1.9.0-SNAPSHOT.oar*), te downloaden via de blogpost of via deze [pagina](#). Zorg dat deze bestanden in de home folder van de VM terecht komen (dit is `/home/ubuntu`).

## Ontdek ONOS (Open Network Operating System)

ONOS is een *open source* controller, geschreven in Java. Je mag de code gratis gebruiken, inzien en naar eigen believen aanpassen (op je eigen systeem). Programmeurs over heel de wereld hebben hieraan meegewerkt. Alle documentatie (en nog meer tutorials/tips) is te vinden op hun [wiki](#).

Ondanks dat er een ONOS-versie op onze VM staat, raad ik je aan deze te verwijderen (verwijder de map 'onos') en een nieuwe versie te downloaden. Op de VM, ga naar de volgende [internetpagina](#) en download de 'tar.gz' file van de versie 1.10.4 (Kingfisher).

Open nu een terminal (in de VM is de sneltoets Ctrl+Alt+T) en voer de volgende commando's uit:

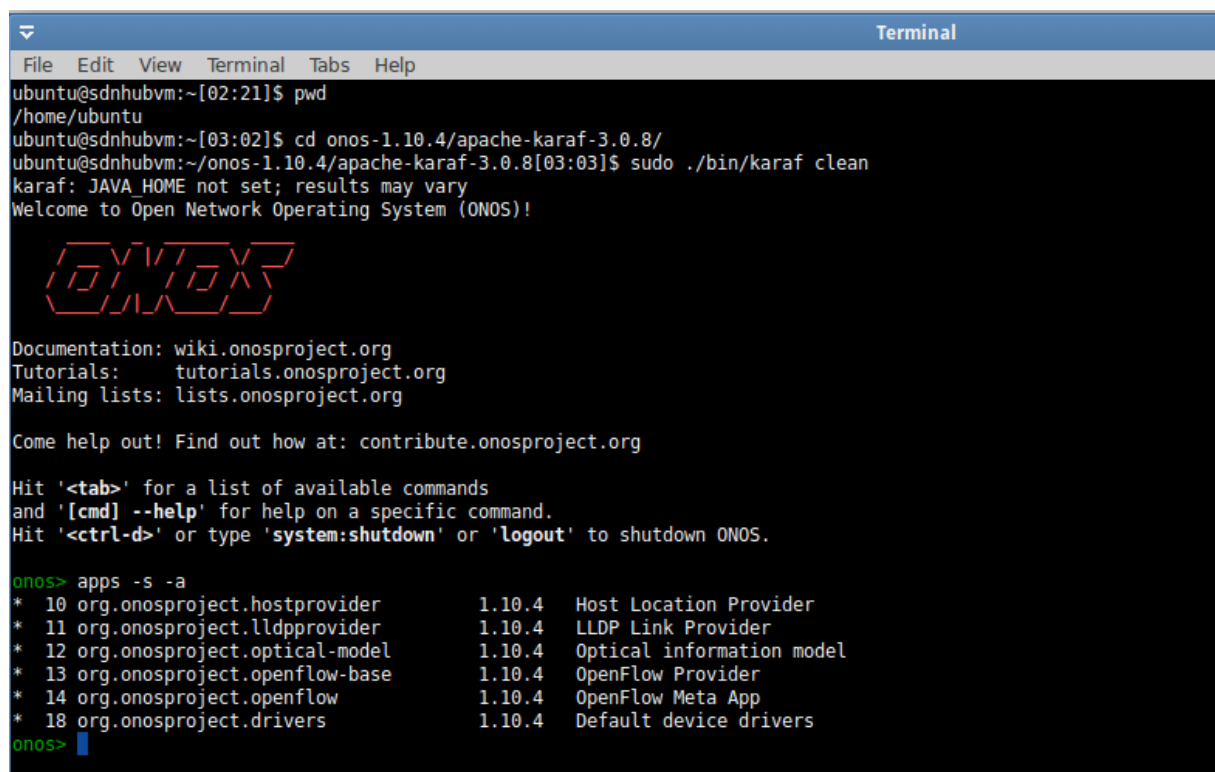
```
cd Downloads/  
tar -zxvf onos-1.10.4.tar.gz
```

Zorg vervolgens dat de uitgekakte map in de home-folder terecht komt (/home/ubuntu/). Dit kan via de terminal of via de Verkenner, wat je zelf fijn vindt.


Laten we gelijk de SDN-controller opstarten! Dit kan door de volgende commando's uit te voeren in een terminal (gegeven dat je start in /home/ubuntu/):

```
cd onos-1.10.4/apache-karaf-3.0.8  
sudo ./bin/karaf clean
```

Als het goed is, start nu de Command Line Interface (CLI) van de controller op (*heb geduld*). ONOS gebruikt hiervoor de tool 'Karaf' - maar daar gaan we in deze tutorial niet verder op in.



```
ubuntu@sdnhubvm:~[02:21]$ pwd  
/home/ubuntu  
ubuntu@sdnhubvm:~[03:02]$ cd onos-1.10.4/apache-karaf-3.0.8/  
ubuntu@sdnhubvm:~/onos-1.10.4/apache-karaf-3.0.8[03:03]$ sudo ./bin/karaf clean  
karaf: JAVA HOME not set; results may vary  
Welcome to Open Network Operating System (ONOS)!
```



```
Documentation: wiki.onosproject.org  
Tutorials:    tutorials.onosproject.org  
Mailing lists: lists.onosproject.org  
  
Come help out! Find out how at: contribute.onosproject.org  
  
Hit '<tab>' for a list of available commands  
and '[cmd] --help' for help on a specific command.  
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown ONOS.  
  
onos> apps -s -a  
* 10 org.onosproject.hostprovider      1.10.4  Host Location Provider  
* 11 org.onosproject.lldpprovider      1.10.4  LLDP Link Provider  
* 12 org.onosproject.optical-model     1.10.4  Optical information model  
* 13 org.onosproject.openflow-base    1.10.4  OpenFlow Provider  
* 14 org.onosproject.openflow         1.10.4  OpenFlow Meta App  
* 18 org.onosproject.drivers           1.10.4  Default device drivers  
onos>
```

Vanaf de CLI kun je verschillende commando's uitvoeren en bijvoorbeeld applicaties activeren. Om de controller ook daadwerkelijk nuttige dingen te laten doen, is dit ook nodig - ONOS heeft een collectie aan functionaliteiten in apps gegoten die je aan/uit kunt zetten. Voor de basis-applicaties, voer de volgende commando's uit:

```
app activate org.onosproject.openflow  
app activate org.onosproject.drivers  
apps -s -a
```

Als het goed is, wanneer je het laatste commando uitvoert, krijg je dezelfde lijst met geactiveerde applicaties te zien als op de afbeelding hierboven. (Zo niet, dan kan je ze los activeren).

**Extra:** in het commando `apps -s -a`, betekent de optie `-s` dat je een samenvattend resultaat wil zien en `-a` dat je alleen de geactiveerde applicaties wil zien. Wanneer je `apps --help` doet, krijg je een lijst van alle opties voor het commando `apps` te zien. In de ONOS CLI kan je `--help` gebruiken achter elk commando als er meer over wilt weten.

## Je eigen netwerk met Mininet

Mininet is een tool waarmee je gemakkelijk een virtueel netwerk op kan zetten. Deze tool wordt voornamelijk gebruikt om netwerkjes te creëren dat Open vSwitches bevat. Open vSwitches zijn van die ‘domme’ switches die een controller nodig hebben om te functioneren. Om Mininet helemaal te ontdekken raad ik je aan een van de beschikbare tutorials te volgen die online te vinden zijn.

Mininet kan je starten met het volgende commando (plaats deze op één regel):

```
sudo mn --topo tree,2,3 --controller remote,ip=127.0.0.1,port=6633
--switch ovsk,protocols=OpenFlow13 --mac
```

Kortweg zijn er in dit commando vier dingen die je doet:

- Je kiest de topologie van het netwerk. In dit geval bouwen we een ‘boom’ met diepte 2 en 3 hosts per ‘leaf’ switch.
- We verbinden ons netwerk aan een controller die ‘remote’ (op een andere locatie) is, en geven daarbij een IP-adres en port mee waarop we de controller denken te kunnen bereiken. Het OpenFlow protocol gebruikt standaard port 6633 of 6653, en in dit geval draait de controller op dezelfde computer dus kunnen we het *localhost*-adres gebruiken.
- We definiëren de switches, in dit geval zijn dat Open vSwitches met OpenFlow 1.3.
- We willen simpele MAC-adressen (IP-adres 10.0.0.1 krijgt MAC 00:00:00:00:00:01)

```
ubuntu@sdnhubvm:~[03:24]$ sudo mn --topo tree,2,3 --controller remote,ip=127.0.0.1,port=6633 --switch ovsk,protocols=OpenFlow13
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(s1, s2) (s1, s3) (s1, s4) (s2, h1) (s2, h2) (s2, h3) (s3, h4) (s3, h5) (s3, h6) (s4, h7) (s4, h8) (s4, h9)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ...
*** Starting CLI:
mininet>
```

Via Mininet heb je toegang tot een aparte CLI waar je verschillende commando's kan uitvoeren, zoals `h1 ping h2` of `xterm h1` (opent een apart terminal voor host 1) of `exit` (sluit af). Wanneer je alle informatie van het netwerk wil hebben, is het commando `dump` interessant.

```
mininet> dump
<Host h1: h1-eth0:10.0.0.1 pid=3379>
<Host h2: h2-eth0:10.0.0.2 pid=3388>
<Host h3: h3-eth0:10.0.0.3 pid=3393>
<Host h4: h4-eth0:10.0.0.4 pid=3398>
<Host h5: h5-eth0:10.0.0.5 pid=3403>
<Host h6: h6-eth0:10.0.0.6 pid=3408>
<Host h7: h7-eth0:10.0.0.7 pid=3413>
<Host h8: h8-eth0:10.0.0.8 pid=3418>
<Host h9: h9-eth0:10.0.0.9 pid=3423>
<OVSSwitch('protocols': 'OpenFlow13') s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=3431>
<OVSSwitch('protocols': 'OpenFlow13') s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None,s2-eth4:None pid=3434>
<OVSSwitch('protocols': 'OpenFlow13') s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None,s3-eth4:None pid=3437>
<OVSSwitch('protocols': 'OpenFlow13') s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None,s4-eth4:None pid=3440>
<RemoteController('ip': '127.0.0.1', 'port': 6633) c0: 127.0.0.1:6633 pid=3373>
mininet>
```

**Extra:** Als je wilt dat verkeer het doet op deze topologie, activeer dan de `org.onosproject.fwd` applicatie op ONOS. Initieer vervolgens een ping tussen twee hosts via Mininet en voilà!

## SDN in detail: Stap 1

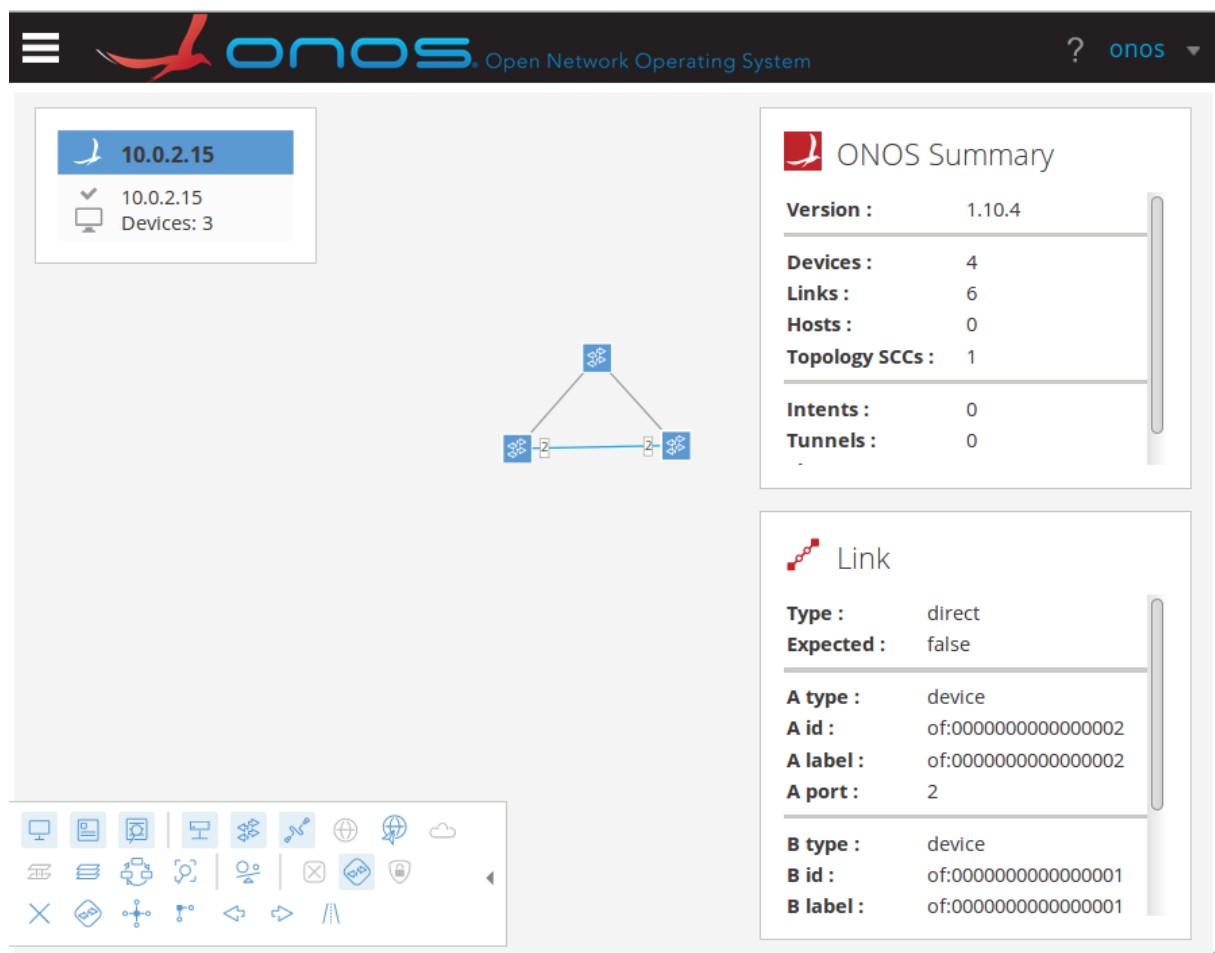
**Belangrijk:** Vanaf nu heb je de extra bestanden nodig op de VM (in de home-folder). Mocht downloaden via de blogpost moeilijk zijn, dan kun je ze ook downloaden via deze [pagina](#).

Nu gaan we wat gedetailleerder kijken naar hoe SDN werkt. Daarnaast gaan we gebruik maken van de GUI van ONOS. Allereerst, zorg dat ONOS opgestart is in een terminal en dat de basis applicaties zijn geactiveerd. Start vervolgens vanuit de home-folder een andere terminal Mininet met onze eigen topologie (commando moet op één regel):

```
sudo mn --mac --topo mytopo --custom triangle.py --controller remote,ip=127.0.0.1,port=6633 --switch ovsk,protocols=OpenFlow13
```

**Extra:** Triangle.py definieert een netwerk met drie switches in een driehoek met aan elke switch één host. Nieuwsgierig hoe? Open dan een het bestand met een tekst-editor (bv. 'gedit' op de VM). Je kunt zelf ook zo'n soort script kunnen schrijven en daarmee je eigen topologie definiëren.

De GUI van ONOS vinden we in een browser via de URL: <http://localhost:8181/onos/ui/login.html>, en login 'onos' / 'rocks'. Vervolgens zien we iets wat hierop lijkt:



The screenshot displays the ONOS GUI interface. At the top, there is a header with the ONOS logo and the text "Open Network Operating System". Below the header, the main area is divided into several sections:

- Top Left:** A box showing the IP address "10.0.2.15" and a checkmark, indicating a successful connection or status.
- Center:** A network topology diagram showing three switches connected in a triangle, with hosts connected to each switch.
- Right Panel:** An "ONOS Summary" section with a table of network statistics:

Version :	1.10.4
Devices :	4
Links :	6
Hosts :	0
Topology SCCs :	1
Intents :	0
Tunnels :	0
- Bottom Right Panel:** A "Link" details section showing information for a specific link:

Type :	direct
Expected :	false
A type :	device
A id :	of:000000000000000002
A label :	of:000000000000000002
A port :	2
B type :	device
B id :	of:000000000000000001
B label :	of:000000000000000001

**Extra:** Neem de tijd om op deze pagina wat te experimenteren. Je kan op verbindingen en op switches klikken (ook op hosts zodra ze verschenen zijn), en portnummers verschijnen. Druk op '/' om het menu met sneltoetsen te openen om nog meer opties te ontdekken!

Je kunt nu de app `org.onosproject.fwd` activeren op de controller en in Mininet een `pingall` commando uitvoeren. Terwijl je dit doet, let op de GUI. Hoewel de pings *niet* succesvol zullen zijn, zal je wel de hosts zien verschijnen in de topologie (zorg dat 'Host Visibility' aanstaat (sneltoets H)).

*Ehm, pardon?! De pings doen het niet, leg eens uit!*

Zoals gezegd, switches zijn ‘domme’ apparaten die alleen maar hun flow table kunnen raadplegen. Handig genoeg kan je via Mininet (en ONOS) de inhoud van de flow table opvragen. Via Mininet gaat dat met het commando:

```
sh ovs-ofctl -O OpenFlow dump-flows s1
```

```
mininet> sh ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST FLOW reply (OFL3) (xid=0x2):
cookie=0x100001729c813, duration=1127.709s, table=0, n_packets=727, n_bytes=58887, send_flow_rem priority=40000,dl_type=0x88cc actions=CONTROLLER:65535,clear_actions
cookie=0x100001f5248c4, duration=1127.707s, table=0, n_packets=9, n_bytes=378, send_flow_rem priority=40000,arp actions=CONTROLLER:65535,clear_actions
cookie=0x10000e50eea1d, duration=1127.621s, table=0, n_packets=727, n_bytes=58887, send_flow_rem priority=40000,dl_type=0x8942 actions=CONTROLLER:65535,clear_actions
mininet>
```

**Extra:** In de ONOS CLI zijn commando's als `flows`, `devices`, `hosts` en `links` interessant.

Hoewel in zeer kleine lettertjes op dit plaatje, te zien is in de tweede regel dat een deel zegt:

`priority=40000, arp actions=CONTROLLER:65535.`

Kort gezegd houdt deze regel in dat als de switch een ARP-pakket tegenkomt, hij het gehele pakket moet doorsturen naar de controller, zodat die een beslissing kan nemen. De controller probeert met behulp van alle ontvangen pakketten de topologie te ontdekken - dit is geïmplementeerd in de forwarding-app die we hebben geïnstalleerd. In topologieën zonder loops (cirkels) werkt deze app prima en worden er als resultaat door de controller nieuwe regels in de flow tables geïnstalleerd zodat de switches voortaan weten wat ze moeten doen. Maar in dit geval kan de controller geen beslissing maken en loopt het verkeer dus spaak.

**Extra:** Probeer de `org.onosproject.fwd` app nog eens met een andere Mininet topologie (zoals de `--tree,2,3` of `--single,6`) en kijk vóór en nádat je gaat pingen naar de inhoud van de flow table(s)!

## SDN in detail 2: Intent Based Networking

Nu gaan we een applicatie installeren op de controller die werkt op basis van *Intents*. Intents kunnen gezien worden als policy regels die wensen en eisen specificeren. De controller berekent hoe het verkeer moet lopen om aan deze wensen en eisen te voldoen.

Voor de zekerheid, verschoon Mininet, nadat je het hebt afgesloten met `exit`, met het commando:

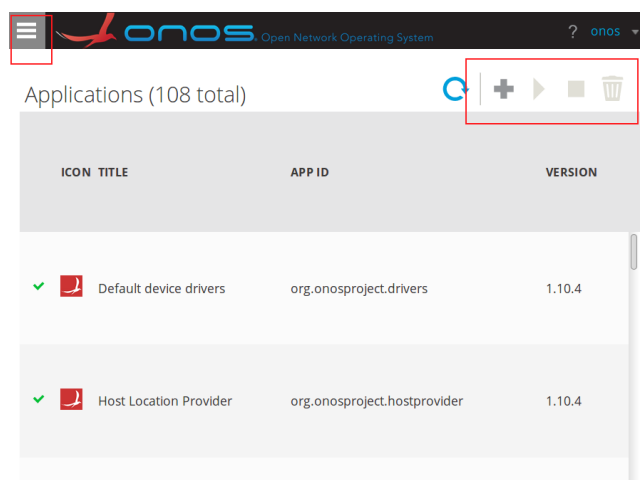
```
sudo mn -c
```

En verschoon de controller (opnieuw opstarten mag ook), met het commando's:

```
app deactivate org.onosproject.fwd
wipe-out please
```

Wanneer de controller en Mininet weer zijn opgestart, ga in de ONOS GUI via het menu naar 'Applications'. Hier kunnen we het `.oar` bestand toevoegen. OAR staat voor *ONOS Application aRchive*; in principe kun je elk zelf-geschreven applicatie ombouwen tot zo'n bestand zodat je deze kan toevoegen aan de controller. Dat is nu waar de term *software defined* vandaan komt!

In de GUI, gebruik de plus-knop om het `.oar` bestand te uploaden en uiteindelijk de play-knop om de applicatie te activeren.



**Extra:** Voer in de ONOS CLI `apps -s -a` uit om daar te zien dat de applicatie het doet.

Probeer nu in Mininet weer een `pingall`. Heb geduld, want de controller moet het een en al berekenen, maar als je het commando een tweede keer uitvoert hoort het allemaal te werken.

### *Wat is er nu gebeurd?*

Om te begrijpen wat er is gebeurd, bekijk in Mininet de flow table van een switch met:

```
sh ovs-ofctl -O OpenFlow13 dump-flows s1
```

en voer in de ONOS CLI het volgende commando in:

```
intents
```

De uitkomst voor de Intent voor verkeer tussen Host 1 en Host 2 zie je hieronder:

```
onos> intents
Id: 0x86
State: INSTALLED
Key: 00:00:00:00:00:01/None00:00:00:00:00:02/None
Intent type: HostToHostIntent
Application Id: org.onosproject.ifwd
Resources: [00:00:00:00:00:02/None, 00:00:00:00:00:01/None]
Treatment: [NOACTION]
Constraints: [LinkTypeConstraint{inclusive=false, types=[OPTICAL]}]
Source host: 00:00:00:00:00:02/None
Destination host: 00:00:00:00:00:01/None
```

Zoals gezegd heeft de controller op basis van de Intent besloten hoe het verkeer moet lopen. De Intents (die je ziet in de ONOS CLI) zijn vertaald naar flow rules (die je ziet m.b.v. Mininet). Zoals je hierboven kan aflezen heeft deze Intent weinig constraints: we willen verkeer tussen twee hosts en hebben geen verdere eisen. Maar Intents kunnen heel veel constraints met zich meegeven, zoals je kan zien als je in de ONOS CLI het volgende commando uitvoert:

```
add-host-intent --help
```

De uitkomst is een hele lange lijst met opties die je kunt meegeven aan zelf gedefinieerde Intents. En voor deze Intents hoeft je helemaal niet na te denken over de route die het verkeer zal afleggen - dat rekenwerk zal de controller uitvoeren en vertalen naar flow rules voor de betrokken switches. En dit is geïmplementeerd in de ifwd applicatie.

### *Dynamische netwerken*

De controller houdt te allen tijde het netwerk en de eisen (ofwel Intents) in de gaten. Wanneer je een Intent voor verkeer tussen Host 1 en Host 2 weg zou gooien met het commando:

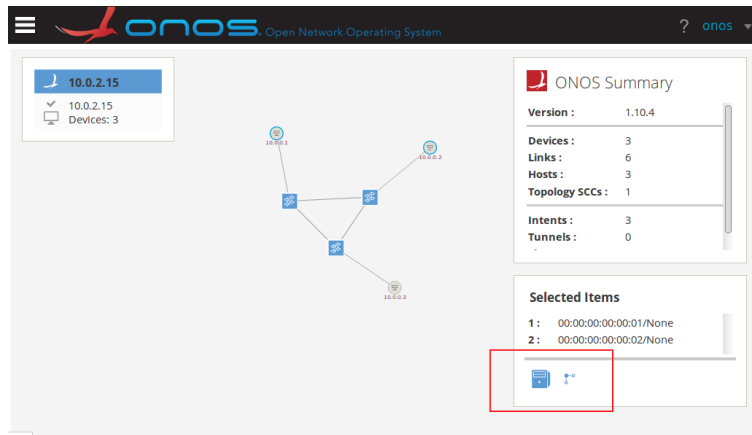
```
remove-intent org.onosproject.ifwd 00:00:00:00:00:01/None00:00:00:00:00:02/None
```

dan is het resultaat dat de controller de flow rules die bij deze Intent horen ook verwijderd. En dat kan je zien in de flow table van de betrokken switches:

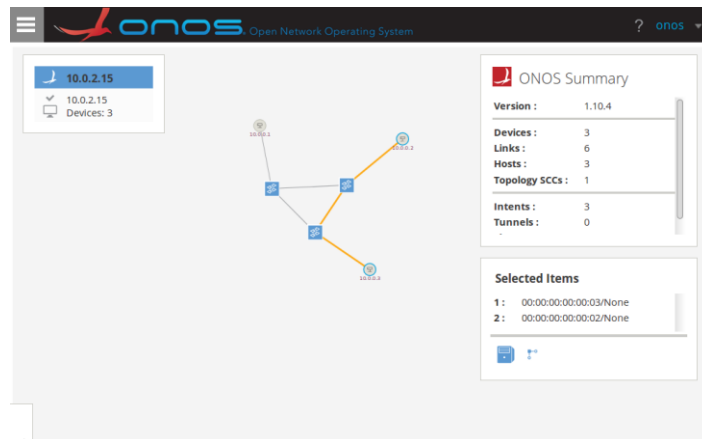
```
mininet> sh ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x1000033229ece, duration=750.462s, table=0, em priority=5, ip actions=CONTROLLER:65535, clear_actions
cookie=0x10000e50eeaid, duration=750.459s, table=0, ow rem priority=40000, dl type=0x8942 actions=CONTROLLER:65535, clear_actions
cookie=0x100001f5248c4, duration=750.447s, table=0, em priority=40000, arp actions=CONTROLLER:65535, clear_actions
cookie=0x100001729c813, duration=750.447s, table=0, ow rem priority=40000, dl type=0x88cc actions=CONTROLLER:65535, clear_actions
cookie=0x7300004d0f346e, duration=748.028s, table=0, sm priority=100, in_port=1, dl_src=00:00:00:00:00:01, dl_dst=00:00:00:00:00:03 actions=output:3
cookie=0x730000452e6807, duration=748.028s, table=0, sm priority=100, in_port=3, dl_src=00:00:00:00:00:03, dl_dst=00:00:00:00:00:01 actions=output:1
cookie=0x730000a96e1653, duration=747.478s, table=0, n priority=100, in_port=1, dl_src=00:00:00:00:00:01, dl_dst=00:00:00:00:00:02 actions=output:2
cookie=0x73000060db5e8, duration=747.478s, table=0, n priority=100, in_port=2, dl_src=00:00:00:00:00:02, dl_dst=00:00:00:00:00:01 actions=output:1
mininet> sh ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x1000033229ece, duration=780.350s, table=0, em priority=5, ip actions=CONTROLLER:65535, clear_actions
cookie=0x10000e50eeaid, duration=780.347s, table=0, ow rem priority=40000, dl type=0x8942 actions=CONTROLLER:65535, clear_actions
cookie=0x100001f5248c4, duration=780.335s, table=0, em priority=40000, arp actions=CONTROLLER:65535, clear_actions
cookie=0x100001729c813, duration=780.335s, table=0, ow rem priority=40000, dl type=0x88cc actions=CONTROLLER:65535, clear_actions
cookie=0x7300004d0f346e, duration=777.916s, table=0, sm priority=100, in_port=1, dl_src=00:00:00:00:00:01, dl_dst=00:00:00:00:00:03 actions=output:3
cookie=0x730000452e6807, duration=777.916s, table=0, sm priority=100, in_port=3, dl_src=00:00:00:00:00:03, dl_dst=00:00:00:00:00:01 actions=output:1
mininet>
```

De verwijderde Intent kan je op twee manieren weer terugkrijgen. Allereerst door een ping te initiëren (maar dat is de saaie oplossing), ten tweede via de GUI. In pagina waar je de topologie ziet, kun je m.b.v. de Shift-knop de twee betreffende Hosts aanklikken en dan verschijnt er de mogelijkheid om een *Host-to-Host Intent* te creëren:

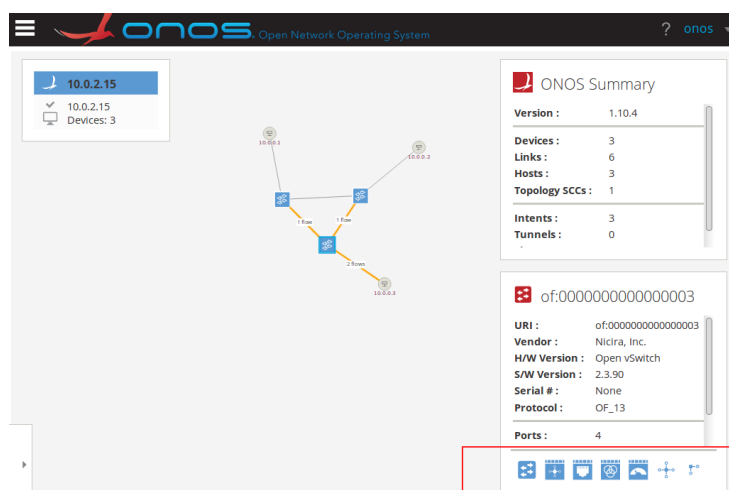




De rechter knop toont 'Related Traffic': Omdat de Intent net verwijderd is, is dit resultaat leeg - Hieronder zie je bijvoorbeeld wel related traffic tussen twee andere hosts. De nieuwe Intent kan toegevoegd met de linker knop.



**Extra:** Je kunt ook een switch aanklikken, waarna er getoond wordt hoeveel flows er zijn geïnstalleerd voor de verschillende verbindingen. Gebruik de icoontjes rechtsonder om nog meer te ontdekken!



**Extra:** Intents houden ook rekening met de status van het netwerk. Kijk maar eens in de flow tables en topologie wat er gebeurt als je een verbinding tussen twee switches (tijdelijk) uitzet in Mininet met:  
link s2 s3 down

## Wat kunnen we nog meer?!

Ik zou nog eindeloos door kunnen praten en voorbeelden kunnen noemen van wat we nog kunnen doen, maar ik zie dat het aantal pagina's van deze tutorial aan het groeien is, dus vanaf nu is het aan jezelf om dingen te proberen. (En mocht het niet lukken, dan mag je altijd even langslopen). Er zijn namelijk nog twee concepten die interessant zijn om in te duiken:

### *Point Intents*

De Host-to-Host Intents laten het rekenwerk aan de controller over en de route van het verkeer bepalen. Je kunt ook zogenaamde *Point Intents* gebruiken om specifiek door te voeren wat je wil dat er gebeurt als verkeer op een bepaalde port in een switch binnenkomt. Zulke Intents voeg je toe met het commando `add-point-intent`.

Met behulp van deze Intents kan je zelf routes bepalen en er dus voorzorgen dat verkeer (bijvoorbeeld) op de heenweg linksom loopt en op de terugweg rechtsom. Probeer zelf eens hoe ver je komt - meer informatie is te vinden in de volgende [YouTube-video](#).

### *Wat houdt OpenFlow verkeer in?*

Het enige wat ik heb verteld is dat ONOS het OpenFlow protocol gebruikt om flow rules te installeren op de switches, maar in feite hebben we nog heel weinig van het protocol gezien. Mocht je dit nu interessant vinden, dan kun je hiervoor de packet sniffer *Wireshark* gebruiken. Wireshark is in staat om OpenFlow verkeer te onderscheppen en identificeren, zodat jij er dieper op in kan gaan.

Wil je het protocol en Wireshark induiken, dan raadt ik je aan om dit met een simpele netwerk topologie te doen (--single,3 is al voldoende). Voordat je de topologie opstart (en daarmee verbindt met de controller), start Wireshark op. Dit kan het handigste via een terminal met het commando:

```
sudo wireshark &
```

In Wireshark, begin een Capture op de `Loopback:lo` interface. Voer vervolgens een filter in zodat je alleen het OpenFlow verkeer te zien krijgt:

```
openflow_v4
```

**Alternatief:** Nadat de switches zijn verbonden met de controller is er veel onderling verkeer om op te hoogte te blijven van de status van het netwerk. Dit is (voor nu) niet interessant, en filter je via:

```
openflow_v4 and openflow_v4.type != ofpt_multipart_request and  
openflow_v4.type != ofpt_multipart_reply
```

Wanneer je nu Mininet opstart, zie je in Wireshark verschillende OpenFlow berichten langskomen, zoals een HELLO, FEATURE\_REQUEST en FEATURE\_REPLY en uiteindelijk ook FLOW\_MODs. De eerste drie type berichten horen bij de handshake tussen switches en de controller om verbinding te maken. De FLOW\_MODs zijn de berichten die de flow rules bevatten.

Kijk vervolgens wat voor communicatie tussen de switches en controllers is wanneer je een ping of pingall initieert in Mininet. Je kan dit bijvoorbeeld doen terwijl de `org.onosproject.fwd` of `org.onosproject.ifwd` applicatie is geactiveerd op de controller.

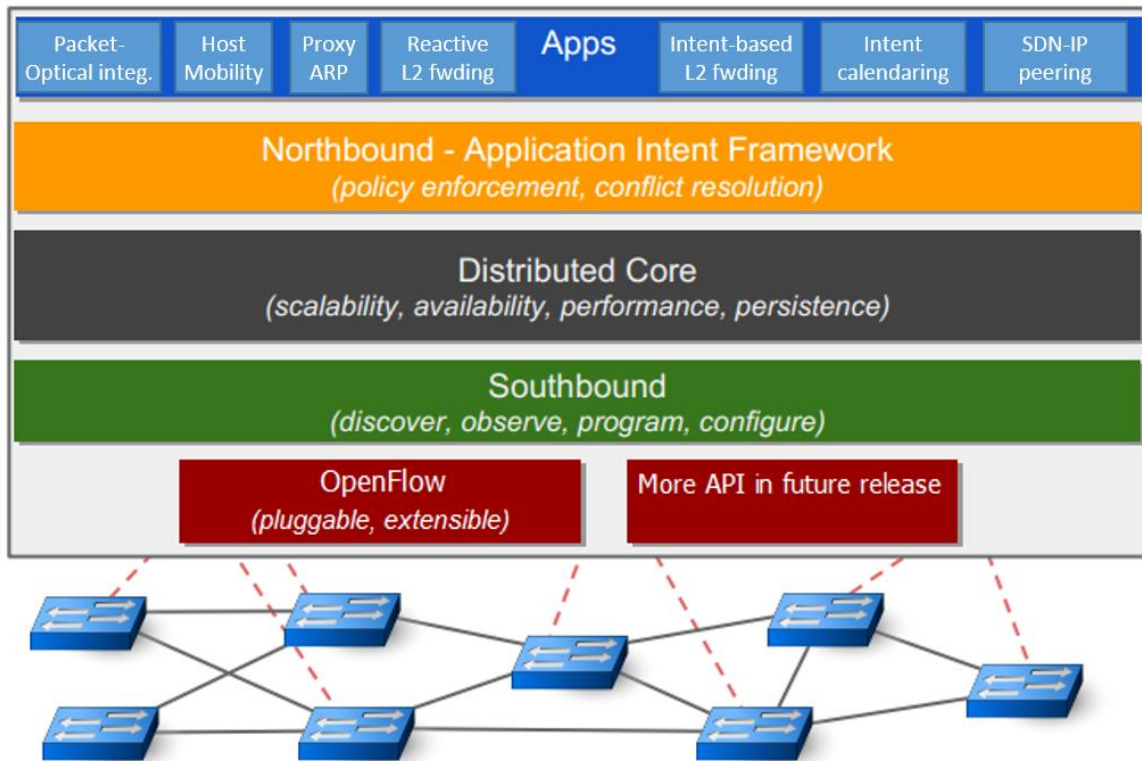
**Extra:** Je kan de inhoud uitpluizen via de Packet Details - de Packet Bytes zijn minder interessant.

**Extra:** Je kunt ook een capture uitvoeren op een interface/port van een switch. Zo kan je zien of verkeer echt de route neemt die je bijvoorbeeld hebt afgedwongen met de Point Intents.



## Alles in context

Nu we van alles hebben gezien van de ONOS-controller en hoe dingen z'n gang gaan in een Software-Defined Network, is het goed te herhalen wat we hebben gezien en hoe dat nu allemaal in elkaar past. Dit kunnen we doen door te kijken naar de architectuur ONOS:



Eigenlijk is de ONOS-implementatie de vormgeving van de gele, zwarte en groene laag: De *core* van de controller ondersteunt aan de boven- en onderkant verschillende verbindingen met externe 'dingen'. Aan de onderkant is dat verbinding met netwerkapparatuur. In deze tutorial is er vooral gefocust op het OpenFlow protocol, dat gebruikt wordt om de flow tables van SDN-enabled switches te vullen. Deze SDN-enabled switches hebben we in deze tutorial opgezet met behulp van Mininet.

De core implementeert de basis functionaliteit van de controller, maar uiteindelijk zullen de applicaties bovenop de controller bepalen hoe het netwerk functioneert. Wij hebben twee applicaties bekeken, namelijk Reactive L2 Forwarding en van Intent-based L2 Forwarding. Er zijn er nog veel meer, want een netwerk moet veel meer kunnen doen dan alleen pakketten doorsturen. De ONOS GUI is ook een voorbeeld van een applicatie!

Applicaties gebruiken de northbound interface (ofwel API's) om hun eisen en wensen in het juiste format te gieten zodat de controller er iets mee kan. Tegelijkertijd heeft de controller een 'policy enforcement' implementatie die er bijvoorbeeld voor zorgt dat de eisen die applicatie A geeft tegelijkertijd doorgevoerd kunnen worden als de eisen van applicatie B. Conflicten tussen verschillende applicaties kunnen door de implementatie voor 'conflict resolution' opgelost worden.

*En alle applicaties zijn door middel van software geschreven - en dat maakt het software-defined networking!*