LAB 3/2 & LAB 3/3 – Construindo um *Switch* L2

Objetivo

O objetivo destas práticas é construir um *Switch* L2 e aprimorá-lo sucessivamente até se tornar uma aplicação de encaminhamento, que se utiliza das vantagens de uma rede SDN, como a visão global da rede, e das abstrações de *Intents*, provendo inteligência aos *switches* da rede, tornando-o capaz de reagir às mudanças na topologia imediatamente.

Em um *switch c*om aprendizado, examinamos o conteúdo de cada pacote, mantemos um registro da porta associada a cada MAC *address* e encaminhamos os pacotes em direção ao destino. Para destinos desconhecidos, utiliza-se o recurso de *flood* do pacote, como em um HUB.

Esperamos que o aluno aprenda alguns conceitos novos da programação no ONOS, como a criação de regras de fluxo e *intents*.

LAB 3/2 - Construindo um *Switch* L2 básico

Introdução

Para construir um *switch* com aprendizado simples, utilizaremos uma tabela para registrar o par MAC↔SW/Porta, referentes ao *host* de origem do pacote e ao ponto de conexão onde o pacote foi recebido.

Aproveitando novamente das primitivas do ONOS, utilizaremos o serviço de *HOSTS* para recuperar as informações de um *host*, no caso o *switch*/porta ao qual ele está conectado na topologia, à partir do endereço MAC do pacote recebido.

Inicialmente a aplicação não criará as regras de fluxo nos *switches* e todos os pacotes subirão para o controlador tratá-los. Há diversos problemas com essa abordagem ingênua, como:

- O controlador fica sobrecarregado tratando todos os pacotes de todos os fluxos;
- Maior latência no enlace, uma vez que todos os pacotes irão ao controlador antes de encaminhados ao destino;
- Se houver uma falha no controlador a comunicação na rede irá parar de ocorrer pois não há regras instaladas nos *switches.*
- A tabela de encaminhamento do *switch*, onde parte das otimizações de desempenho são realizadas, tornando o *switch* eficiente, fica inutilizada, uma vez que nada mais é armazenado nela

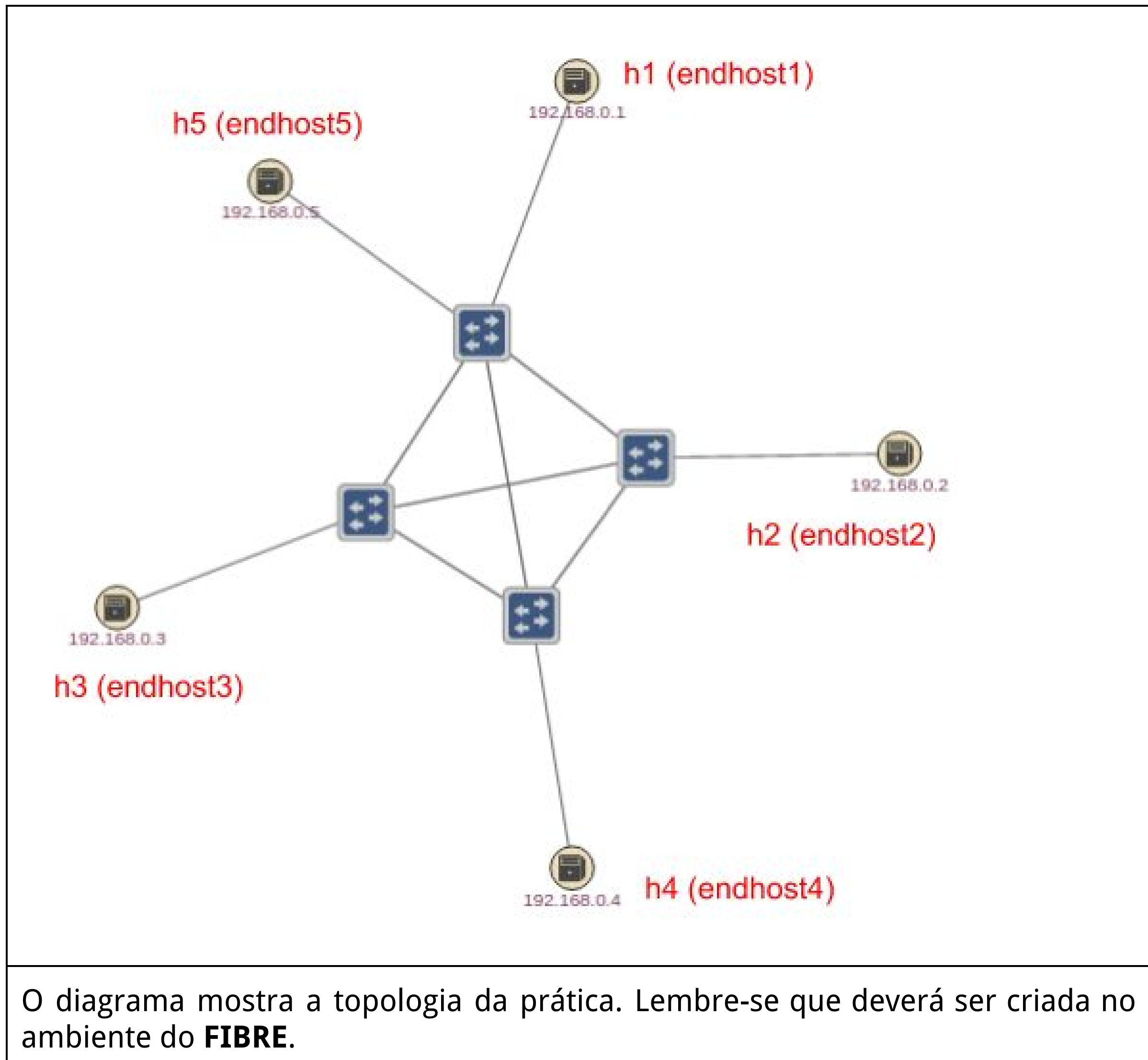
Posteriormente iremos alterar a aplicação para um *switch* baseado em fluxos, de forma que a aplicação instale as regras de fluxo nos *switches*.

Princípios de um *Switch* de aprendizado

- Ao receber um pacote e lê seu endereço MAC de destino;
- Busca em uma base de dados a qual porta o pacote deve ser encaminhado;
- Para destinos conhecidos o pacote é encaminhado e se cria uma regra de fluxo;
- Para destinos desconhecidos, ocorre um "*flood*" na rede;

Topologia no FIBRE

Nesse ponto já devemos ter a topologia construída no FIBRE, conforme ensinado no tutorial do FIBRE, realizado na prática anterior**.** A topologia da prática consiste de uma rede *full-mesh* de 4 *switches*, com 5 *hosts* conectados, conforme figura abaixo:



O diagrama mostra a topologia da prática. Lembre-se que deverá ser criada no ambiente do **FIBRE**.

Código relevante para o aprendizado

Código do Simple Switch L2

```
@Reference(cardinality = ReferenceCardinality.MANDATORY_UNARY)
protected FlowRuleService flowRuleService;
```

Importa o serviço de *Flows* do ONOS, responsável por criar e gerenciar as regras de fluxos. Ele será usado para criarmos e instalarmos as regras nos *switches*.

```
if ( pkt.receivedFrom().deviceId().equals(
        dst.location().deviceId())) {
    if(!context.inPacket().receivedFrom().port().
            equals(dst.location().port() )) {
        installRule(context, dst.location().port());
    }
    return;
}
```

Se estamos no mesmo *switch* de borda que o destino (***pkt.receivedFrom().deviceId*** e ***dst.location().deviceId()***), devemos criar a regra de fluxo (***installRule***) para o destino

```
Set<Path> paths = topologyService.getPaths(
    topologyService.currentTopology(),
    pkt.receivedFrom().deviceId(),
    dst.location().deviceId());
```

Utilizamos o serviço de topologia do ONOS para solicitar todos os caminhos que levam da origem (**pkt.receivedFrom().deviceId()**) para o *switch* onde o host de destino se encontra (**dst.location().deviceId()**)

```
selectorBuilder.matchInPort(
    context.inPacket().receivedFrom().port())
    .matchEthSrc(inPkt.getSourceMAC())
    .matchEthDst(inPkt.getDestinationMAC())
    .matchEthType(Ethernet.TYPE_IPV4);
```

Define um objeto do tipo seletor utilizado para realizar o match da regra com o fluxo. No caso, cria-se um match para fluxos que coincidam com os parâmetros especificados abaixo:

- Porta de entrada do pacote (***.matchInPort***),
- Endereço mac de origem e destino (***.matchEthSrc* e *.matchEthDst***)
- Tipo IPv4 (***.matchEthType***).

`TrafficTreatment treatment = DefaultTrafficTreatment.`
`    builder().`
`    setOutput(portNumber).`
`    build();`
 	
Define a ação a ser executada para os fluxos que coincidirem com o *match* na regra especificada. No caso, alteramos a porta de saida do pacote para a porta especificada em *portNumber*

`ForwardingObjective forwardingObjective =`
`   DefaultForwardingObjective.builder()`
`      .fromApp(appId)`
`      .withSelector(selectorBuilder.build())`
`      .withTreatment(treatment)`
`      .withPriority(flowPriority)`
`      .withFlag(ForwardingObjective.Flag.VERSATILE)`
`      .makeTemporary(flowTimeout)`
`      .add();`
 	
A regra de fluxo conterá o *match* e ação definidos, além de outros parâmetros como a prioridade da regra em caso de conflitos (***flowPriority***).
 	
O parâmetro ***makeTemporary***, torna a regra temporária, expirando-a após o tempo definido no argumento "***flowTimeout***". Pode-se utilizar o parâmetro ***.makePermanent*()** para tornar a regra permanente.

Importando o código no IntelliJ

A prática se encontra no diretório **~/sci2015/aplicacoes/simple-switchl2-app/**

Compilando e carregando a aplicação no ONOS

Para compilar o código e carregá-lo no controlador, execute os comandos a seguir:

```
#> cd ~/sci-2015/aplicacoes/simple-switchl2-app
#> mvn clean install && onos-app $(ipdocker onos1) install \
target/switchl2-app-1.0-SNAPSHOT.oar
```

Ative a aplicação e verifique que ela foi carregada corretamente. Foram criadas 20 regras nos *switches* da topologia (comando **summary**, campo *flows***)**, requisitando que encaminhem os pacotes IPV4, ARP e de controle para o controlador.

```
onos> app activate simple.switchl2.app

onos> summary
node=10.0.3.11, version=1.2.2.docker nodes=1, devices=4, links=12,
hosts=3, SCC(s)=1, flows=20, intents=0
```

```
onos> log:tail
… … …
| 93 - org.onosproject.onos-core-net - 1.2.3.SNAPSHOT | Application
simple.switchl2.app has been installed
| 20 - org.apache.karaf.features.core - 3.0.3 | Installing feature
switchl2-app 1.0-SNAPSHOT
```

Deixar o **log:tail** em execução para acompanhar os eventos dentro do controlador, durante a prática.

Testando a aplicação

Verifique o funcionamento da aplicação. Para tanto, iremos executar um ping de h1 para h3 para ver se o encaminhamento foi realizado conforme o esperado.

************************** **IMPORTANTE** **********************************

Lembre-se que estamos no ambiente do FIBRE e não no Mininet. Substitua qualquer ocorrência de h1 a h5 pelos respectivos *hosts* na topologia do FIBRE. Consulte a imagem da topologia no inicio da prática sempre que precisar verificar o mapeamento dos *hosts* para os IP's.

Para simplificar a prática, abra um terminal em cada uma das 5 VMs (*hosts*) no FIBRE e deixe-os abertos para utilizá-los durante a prática. Consulte a última página do laboratório 3/1 (FIBRE) para verificar como acessar as VMs novamente. Veja um exemplo abaixo, relativo ao acesso à VM endhost1 (192.168.0.1)

```
#> ssh 10.136.12.22 -l meulogin@rnp
Password for meulogin@rnp@10.136.12.22:
root@endhost1:~#
```

Chame os instrutores, caso tenha dificuldade em abrir os terminais nos 5 *hosts* (VMs) da prática.

****

Quando o ONOS não conhece o *host* de destino, a aplicação realiza o *flood* do pacote. No caso do ping entre h1 e h3, como h1 não sabe o endereço MAC de h3, um pacote de *ARP-REQUEST* é gerado antes do ping ser enviado.

Apenas para demonstrar o fato de que, quando o destino é desconhecido, a aplicação realiza o *flood* do pacote, cadastre o endereço MAC de h3 na tabela ARP de h1 manualmente (caso contrário, h3 seria descoberto antes do pacote ICMP ser enviado, devido ao *ARP_REQUEST*). Como o endereço MAC dos *hosts* do FIBRE não são fixos (VM's), é necessário verificar o MAC de h3, conforme abaixo:

```
h3> ip addr |grep 192.168 -B1
link/ether 00:00:00:00:00:03 brd ff:ff:ff:ff:ff:ff
inet 192.168.0.3/24 brd 192.168.0.255 scope global eth1.2134
```

Dado o endereço MAC de h3, cadastre-o na tabela ARP de h1:

```
h1> arp -s 192.168.0.3 00:00:00:00:00:03
```

Para verificar o *flood* durante o primeiro ping (descoberta do *host*), utilize o comando **tcpdump** em outro *host* (h4), enquanto o ping de h1 para h3 está em execução.

Atenção: Como as interfaces das VMs foram configuradas com Vlans, é necessário verificar o *vlanid* correto para utilizar no **tcpdump.**

```
h1> ip addr |grep 192.168
inet 192.168.0.1/24 brd 192.168.0.255 scope global eth1.2134

h4> sudo tcpdump -i eth1.2134 -n not ether proto 0x88cc and not ether
proto 0x8942
tcpdump: verbose output suppressed, use -v or -vv for full protocol
listening on h4-eth0, link-type EN10MB (Ethernet), capture size
18:53:41.343412 IP 192.168.0.1 > 192.168.0.3: ICMP echo request, id
18:53:41.629291 ARP, Request who-has 192.168.0.1 tell 192.168.0.3,

h1> ping 192.168.0.3 -c1
PING 10.0.0.3 (192.168.0.3) 56(84) bytes of data.
64 bytes from 192.168.0.3: icmp_seq=1 ttl=64 time=874 ms
```

Note que o ping funcionou normalmente mas que o pacote de *ICMP* foi recebido pelo *host* h4 (devido ao *flood*), como esperado.

Vamos realizar o *ping* novamente, dessa vez de h2 para h3, mantendo o **tcpdump,** já em execução, no *host* h4. Como o destino, h3, é conhecido pelo controlador, o pacote será entregue diretamente em h3 e nenhum outro dispositivo da rede (*switch* ou *host*) verá o pacote.

```
h4> sudo tcpdump -i eth1.2134 -n not ether proto 0x88cc and not ether
proto 0x8942
tcpdump: verbose output suppressed, use -v or -vv for full
listening on h4-eth0, link-type EN10MB (Ethernet), capture size
19:01:01.813510 ARP, Request who-has 192.168.0.3 tell 192.168.0.2,

h2> ping 192.168.0.3 -c1
PING 10.0.0.3 (192.168.0.3) 56(84) bytes of data.
64 bytes from 192.168.0.3: icmp_seq=1 ttl=64 time=874 ms
```

O *ping* foi respondido corretamente mas, dessa vez, h4 não recebeu o pacote de *ICMP,* recebendo apenas o pacote de ARP vindo de h3 e solicitando o endereço MAC de h2 (como a aplicação de ARP_RESPONDER não está ativada, o ARP sempre incorre em um *flood* na rede).

No entanto, a aplicação não está completa, uma vez que não cria as regras de fluxo nos *switches* e todo pacote sempre é tratado pelo controlador, o que pode sobrecarregá-lo rapidamente. Podemos ver esse comportamento, verificando o log do ONOS (CLI) e o número de regras criadas na topologia.

```
h2> ping 192.168.0.3 -c 5
PING 192.168.0.3 (192.168.0.3) 56(84) bytes of data.
64 bytes from 192.168.0.3: icmp_seq=1 ttl=64 time=4.97 ms
… … … … …
64 bytes from 192.168.0.3: icmp_seq=5 ttl=64 time=9.75 ms

onos> log:tail
2015-09-10 22:05:41,929 | WARN  | w I/O worker #13 | SimpleSwitch
| 175 - simple.switchl2.app - 1.0.0.SNAPSHOT | Trafego do mac
00:00:00:00:00:02 para o mac_dst 00:00:00:00:00:03 via Switch/porta
ConnectPoint{elementId=of:00000000000000a3, portNumber=1}
2015-09-10 22:05:41,931 | WARN  | w I/O worker #11 | SimpleSwitch
| 175 - simple.switchl2.app - 1.0.0.SNAPSHOT | Trafego do mac
00:00:00:00:00:03 para o mac_dst 00:00:00:00:00:02 via Switch/porta
ConnectPoint{elementId=of:00000000000000a2, portNumber=1}
… … … … … …
CTRL+c (para sair do log:tail)

onos> summary
node=10.0.3.11, version=1.3.0.mininet nodes=1, devices=4, links=12,
hosts=3, SCC(s)=1, flows=20, intents=0
```

Desde a ativação da aplicação havia 20 fluxos instalados nos *switches*, indicando que nenhum fluxo foi instalado no sistema pela aplicação. Também podemos notar que, a cada *ping* realizado, há uma entrada no log indicando que a aplicação lidou diretamente com o encaminhamento de todos os pacote do fluxo.

Aprimorando o código - Criando as regras de fluxo

Vamos melhorar o código da aplicação, instruindo-a para que crie as regras OF correspondentes nos *switches*, evitando que novos pacotes do mesmo fluxo sejam encaminhados para o controlador.

No código existem algumas linhas escritas **"TODO Lab 1: Criando regras de Fluxo"**, onde você deverá fazer as adições necessárias para calcular o caminho entre o *host* de origem e destino e adicionar a regra de fluxo em cada elemento desse caminho (para facilitar, procure nas **linhas 200 a 266 do código**).

Testando novamente

Feitas as adições necessárias e substituindo a função *sendTo()* pela função *installRules()*, responsável por criar as regras de fluxos no caminho, precisamos recompilar e instalar a aplicação novamente.

```
#> mvn clean install && onos-app $(ipdocker onos1) reinstall! \
target/switchl2-app-1.0-SNAPSHOT.oar
```

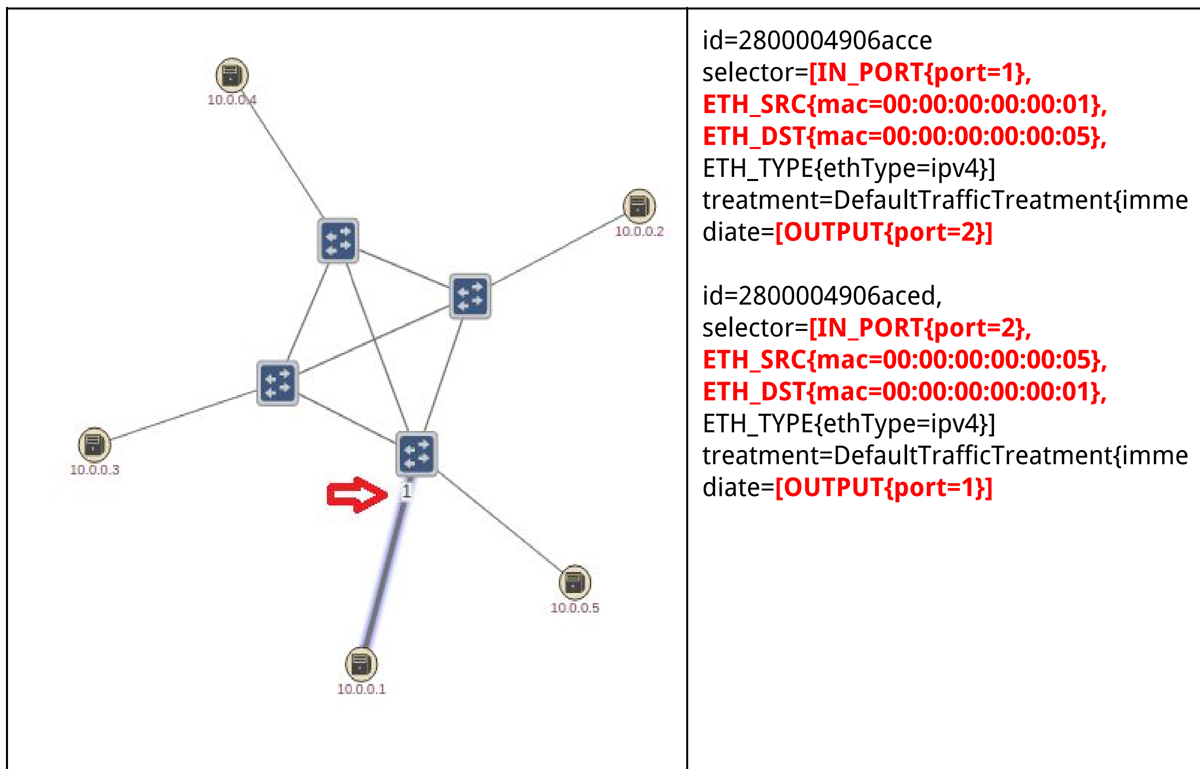
Agora vamos realizar o *ping* novamente e verificar a criação das regras de fluxo. Repare que há apenas 20 regras de fluxos instaladas no momento (**summary)**

```
onos> summary
node=10.0.3.11, version=1.3.0.mininet nodes=1, devices=4, links=12,
hosts=4, SCC(s)=1, flows=20, intents=0

onos> log:clear

h1> ping 192.168.0.5 -c5
PING 192.168.0.5 (192.168.0.5) 56(84) bytes of data.
64 bytes from 192.168.0.5: icmp_seq=1 ttl=64 time=6.90 ms

onos> log:display | grep -i instalando
2015-09-11 01:33:21,451 | WARN  | w I/O worker #13 | SimpleSwitch
| 177 - simple.switchl2.app - 1.0.0.SNAPSHOT | Instalando regra de
fluxo no switch of:00000000000000a1
2015-09-11 01:33:21,460 | WARN  | w I/O worker #13 | SimpleSwitch
| 177 - simple.switchl2.app - 1.0.0.SNAPSHOT | Instalando regra de
fluxo no switch of:00000000000000a1

onos> summary
node=10.0.3.11, version=1.3.0.mininet nodes=1, devices=4, links=12,
hosts=4, SCC(s)=1, flows=22 intents=0

onos> flows |grep -A 2 switchl2.app
  id=20000049063c11, state=ADDED, bytes=392, packets=4, duration=8, priority=100,
tableId=0 appId=simple.switchl2.app, payLoad=null
      selector=[ETH_SRC{mac=00:00:00:00:00:01}, ETH_DST{mac=00:00:00:00:00:05},
ETH_TYPE{ethType=800}, IN_PORT{port=1}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=2}], deferred=[],
transition=None, cleared=false, metadata=null}
   id=20000049063c30, state=ADDED, bytes=392, packets=4, duration=8,
priority=100, tableId=0 appId=simple.switchl2.app, payLoad=null
      selector=[ETH_SRC{mac=00:00:00:00:00:05}, ETH_DST{mac=00:00:00:00:00:01},
ETH_TYPE{ethType=800}, IN_PORT{port=2}]
      treatment=DefaultTrafficTreatment{immediate=[OUTPUT{port=1}], deferred=[],
transition=None, cleared=false, metadata=null}
```

Verifique que o *ping* funcionou normalmente e que o número de regras de fluxo mudou de 20 para 22, indicando que duas regras foram criadas. No caso, como ambos os *hosts* (h1 e h5) estão no mesmo *switch*, foram criadas apenas duas regras (h1 → h2 e h2 → h1), em apenas 1 *switch*, resultando em duas regras de fluxo novas.

No conteúdo das regras, observe que a primeira regra possui um *match* do endereço MAC de origem h1 (00:00:00:00:00:01) para o endereço MAC de destino h5 (00:00:00:00:00:05) e a ação (*treatment*) a ser executada é encaminhar pela porta de saída 2, onde h5 está localizado. Tente acompanhar o *match* e o *treatment* da segunda regra e verifique que ela trata sobre a conexão de h5 para h1 (realçado em vermelho)

Verifique pela interface GUI (WEB) que as portas de entrada e saida em cada *switch* do caminho escolhido e na conexão dos *hosts* de origem e destino com a rede, estão de acordo com as regras de fluxo criadas. Para tanto, basta colocar o foco do *mouse* ("passar em cima") no enlace de h1 com o *switch* para verificar que o *host* se encontra na porta 1 do *switch* (por isso, ***IN_PORT=1 no seletor da regra***) e que para chegarmos ao *host* h5, o pacote deverá sair pela interface 2 do *switch* (**OUTPUT{port=2},** no tratamento da regra). Se houvessem mais *switches* no caminho, iriamos ver na variável *OUTPUT* a porta de conexão do *host* h1 com o próximo *switch* do caminho escolhido.



Vamos realizar um teste de comunicação entre dois *hosts* mais distantes na topologia (h1 e h3) para verificar a instalação da regra em múltiplos *switches*.

Atenção: Confirme os ID's dos *switches* onde os *hosts* h1 e h3 estão conectados utilizando o comando *hosts*, uma vez que o ID do *switch* não é fixo no FIBRE e as análises a seguir serão diferentes em seu terminal.

```
onos> hosts
id=00:00:00:00:00:01/-1, mac=00:00:00:00:00:01, location=of:00000000000000a1/1,
vlan=-1, ip(s)=[192.168.0.1]
id=00:00:00:00:00:02/-1, mac=00:00:00:00:00:02, location=of:00000000000000a2/1,
vlan=-1, ip(s)=[192.168.0.2]
id=00:00:00:00:00:03/-1, mac=00:00:00:00:00:03, location=of:00000000000000a3/1,
vlan=-1, ip(s)=[192.168.0.3]
id=00:00:00:00:00:05/-1, mac=00:00:00:00:00:05, location=of:00000000000000a1/2,
vlan=-1, ip(s)=[192.168.0.5]
```

No caso, os *hosts* h1 e h3 estão localizados nos *switches* com ID of:00000000000a1 e of:00000000000a3, respectivamente.

```
onos> summary
node=10.0.3.11, version=1.3.0.mininet nodes=1, devices=4, links=12, hosts=4,
SCC(s)=1, flows=20, intents=0

onos> log:clear

h1> ping 192.168.0.3 -c1
PING 192.168.0.3 (192.168.0.3) 56(84) bytes of data.
64 bytes from 192.168.0.3: icmp_seq=1 ttl=64 time=6.90 ms

onos> summary
node=10.0.3.11, version=1.3.0.mininet nodes=1, devices=4, links=12, hosts=4,
SCC(s)=1, flows=24 intents=0

onos> log:tail | grep SimpleSwitch
2015-09-11 01:55:44,456 | WARN  | w I/O worker #10 | SimpleSwitch
| 178 - simple.switchl2.app - 1.0.0.SNAPSHOT | Calculo do caminho de
of:00000000000000a1 para of:00000000000000a3
2015-09-11 01:55:44,468 | WARN  | w I/O worker #10 | SimpleSwitch
| 178 - simple.switchl2.app - 1.0.0.SNAPSHOT | Instalando regra de fluxo no
switch of:00000000000000a1
2015-09-11 01:55:44,473 | WARN  | w I/O worker #11 | SimpleSwitch
| 178 - simple.switchl2.app - 1.0.0.SNAPSHOT | Calculo do caminho de
of:00000000000000a3 para of:00000000000000a3
2015-09-11 01:55:44,473 | WARN  | w I/O worker #11 | SimpleSwitch
| 178 - simple.switchl2.app - 1.0.0.SNAPSHOT | Instalando regra de fluxo no
switch of:00000000000000a3
2015-09-11 01:55:44,478 | WARN  | w I/O worker #11 | SimpleSwitch
| 178 - simple.switchl2.app - 1.0.0.SNAPSHOT | Calculo do caminho de
of:00000000000000a3 para of:00000000000000a1
2015-09-11 01:55:44,481 | WARN  | w I/O worker #11 | SimpleSwitch
| 178 - simple.switchl2.app - 1.0.0.SNAPSHOT | Instalando regra de fluxo no
switch of:00000000000000a3
```

```
2015-09-11 01:44:52,095 | WARN  | w I/O worker #10 | SimpleSwitch
| 177 - simple.switchl2.app - 1.0.0.SNAPSHOT | Trafego do mac 00:00:00:00:00:01
para o mac_dst 00:00:00:00:00:03 via Switch/porta
ConnectPoint{elementId=of:00000000000000a1, portNumber=3}
2015-09-11 01:44:52,099 | WARN  | w I/O worker #11 | SimpleSwitch
| 177 - simple.switchl2.app - 1.0.0.SNAPSHOT | Trafego do mac 00:00:00:00:00:03
para o mac_dst 00:00:00:00:00:01 via Switch/porta
ConnectPoint{elementId=of:00000000000000a3, portNumber=4}
```

Observe que a aplicação calcula o caminho do *switch* **of:00000000000000a1** para o **of:00000000000000a3**, instala a regra no primeiro *switch* e encaminha o pacote para o segundo. Como não há regra no segundo *switch*, a aplicação recebe o pacote novamente, calcula o caminho do *switch* **of:00000000000000a3** a ele mesmo (afinal h3 está conectado nele) e instala a regra em **of:00000000000000a3**.

Além disso, para se chegar de h1 (00:00:00:00:00:01) em h3 (00:00:00:00:00:00:03), o tráfego flui pela porta 3 do *switch* **of:000000000000a1** e retorna pela porta 4 do *switch* **of:00000000000a3.**

Por fim, verifique que estamos calculando o caminho a ser percorrido uma vez a cada *switch* no caminho, tratando o pacote novamente e definindo a mesma regra de fluxo. Podemos melhorar a aplicação novamente, utilizando da abstração de *Intents,* uma vez que conhecemos os *hosts* de origem e destino. Dessa forma, deixamos a responsabilidade de calcular o caminho e instalar as regras nos dispositivos da topologia para o *framework* de *Intents*, reduzindo a complexidade da programação da aplicação - o código foi reduzido em 120 linhas (30%) - e ainda ganhamos a flexibilidade das *Intents,* que passarão a monitorar o caminho utilizado, sendo capazes de reagir à qualquer mudança da topologia, recursos - como banda disponível - ou evento de falha.

Para validar o funcionamento da aplicação e mostrar todos os *hosts* da topologia na interface WEB, execute um último ping de h1 para h4

```
h1> ping 192.168.0.4 -c1
```

Desativando a aplicação

Por fim, devemos desativar a aplicação utilizada para que ela não interfira com o *switch* baseado em *Intents*, que será utilizado a seguir:

```
onos> app deactivate simple.switchl2.app
onos> log:clear
```

LAB 3/3 - Aprimorando o Switch L2

Introdução

Agora vamos alterar a aplicação de encaminhamento para uma que utilize as abstrações de *Intents*.

Há vários benefícios em se utilizar os *Intents* no encaminhamento, como:

- Abstrair a complexidade da rede da camada da aplicação e do usuário.
  - ONOS calcula o melhor caminho, considerando toda a topologia e a reserva de recursos como banda e latência.
  - Quando há uma falha na rede, o ONOS automaticamente calcula um novo caminho e instala os fluxos correspondentes.
  - Em caso de um novo caminho não poder ser encontrado, a *intent* fica em um estado de 'falha' e, assim que possível - na recuperação do enlace por exemplo, a intent é instalada novamente.
- Permite combinar diversas *intents* diferentes, referentes ao mesmo fluxo, para criar funcionalidades mais complexas
- Se houver uma falha na rede e existir um outro caminho para atender a comunicação, o mesmo será utilizado de forma transparente para os usuários.

Repare nas facilidades introduzidas pelas *Intents*. Para introduzir resiliência na rede, podemos utilizar uma rede L2 com *loops* na topologia, sem nos preocuparmos com tal fato. As *Intents* utilizam ora um caminho, ora outro e não precisamos configurar proteções como *STP* ou *EAPS*.

Criar topologia no FIBRE

Iremos utilizar a mesma topologia no FIBRE e, portanto, não precisamos nos preocupar com essa parte. Apenas lembrar que a topologia consiste em uma rede *full-mesh* de 4 *switches*, com 5 *hosts* conectados.

Código Relevante para o aprendizado

A grande mudança no código do *switch* ingênuo para o *switch* baseado em *Intents* é a remoção de toda a funcionalidade implementada para cálculo e escolha do caminho e para a instalação das regras nos dispositivos do caminho, uma vez que essa complexidade será tratada pelo *framework* de *Intents*. Além disso, precisamos trocar a função de instalação de fluxos (*installRules*) pela função de criação e submissão de Intents ao framework (*installIntent*) que, inclusive, é bem mais simples do que a de criação de fluxos diretamente.

Código do Intent Switch

`@Reference(cardinality = ReferenceCardinality.MANDATORY_UNARY)`
`protected IntentService intentService;`
 	
Importa o serviço de *Intents* do ONOS, que iremos utilizar para criar e submeter as *Intents* para o *framework* de *Intents*

```
private installIntent(PacketContext context, HostId srcId,HostId dstId) {
   HostToHostIntent intent = HostToHostIntent.builder()
      .appId(appId)
      .one(srcId)
      .two(dstId)
      .selector(selector)
      .treatment(treatment)
      .build();
      // Envia a Intent criada para o framework de intents.
      // O framework ira compilar a intent instala-la e
      // monitorar os eventos da rede que possam afetar
      // a intent.
      intentService.submit(intent);
```

Nessa parte do código criamos a *intent* do tipo *HostToHost*. No caso mais simples, precisamos apenas repassar os *hosts* de origem (***srcId***) e destino (***dstId***) para criar a *intent*, mas poderia ser repassado restrições de utilização como banda, latência e outras ações.

Por fim. devemos submeter a *intent* para o *framework* de *Intents* através da função ***submit*()**.

Importando o código no Intellij

A prática se encontra no diretório **~/sci2015/aplicacoes/intentSwitch/.** Importe a aplicação na IDE, como feito nas práticas anteriores.

Compilando e carregando a aplicação no ONOS

Para evitarmos tratar o ARP, iremos utilizar da aplicação de tratamento de ARP, incluída junto com o CORE do ONOS. Para tanto, ative a aplicação de *proxyarp*

```
onos> app activate org.onosproject.proxyarp
```

Compile e ative a aplicação de *intentSwitch* desenvolvida. Nessa altura, já estamos acostumados a instalar as aplicações no ONOS.

```
#> cd ~/sci-2015/aplicacoes/intentSwitch/
#> mvn clean install && onos-app $(ipdocker onos1) install! \
target/intent-switch-app-1.3.0-SNAPSHOT.oar
```

Repare que colocamos uma exclamação no comando "**onos-app**". Como já explicamos, seu efeito é que a aplicação será automaticamente ativada no ONOS após a sua compilação. No lugar do log do ONOS, vamos utilizar o comando "*apps*" para confirmar que a aplicação foi instalada e ativada com sucesso

```
onos> summary
node=172.17.0.3, version=1.2.3.docker nodes=1, devices=4, links=12,
hosts=5, SCC(s)=1, flows=20, intents=0

onos> apps -s -a
* 3 org.onosproject.drivers  1.3.0.SNAPSHOT Builtin device drivers
* 6 org.onosproject.proxyarp 1.2.2.SNAPSHOT Proxy ARP/NDP application
* 32 org.onosproject.openflow 1.3.0.SNAPSHOT OpenFlow protocol
* 41 intent.switch.app       1.3.0.SNAPSHOT Reactive forwarding
application using intent service (experimental)
```

Os parâmetros "-s" significa que queremos uma versão resumida da aplicação (*short*) e o "-a" que queremos listar apenas as aplicações ativadas.

Testando a aplicação

Vamos verificar o funcionamento da aplicação de *switch*. Para tanto, iremos realizar um *ping* de h1 para h3, para verificarmos se o encaminhamento foi realizado e as *intents* criadas conforme esperado.

```
h1> ping 192.168.0.3 -c 5
PING 192.168.0.3 (192.168.0.3) 56(84) bytes of data.
64 bytes from 192.168.0.3: icmp_seq=2 ttl=64 time=1.21 ms
64 bytes from 192.168.0.3: icmp_seq=3 ttl=64 time=0.142 ms
64 bytes from 192.168.0.3: icmp_seq=4 ttl=64 time=0.202 ms
64 bytes from 192.168.0.3: icmp_seq=5 ttl=64 time=0.144 ms
--- 192.168.0.3 ping statistics ---
5 packets transmitted, 4 received, 20% packet loss, time 4006ms

onos> summary
node=10.0.3.11, version=1.3.0.mininet nodes=1, devices=4, links=12,
hosts=4, SCC(s)=1, flows=24, intents=1

onos> intents
id=0x0, state=INSTALLED, key=0x0, type=HostToHostIntent,
appId=org.onosproject.ifwd
    resources=[00:00:00:00:00:01/-1, 00:00:00:00:00:03/-1]
    host1=00:00:00:00:00:01/-1, host2=00:00:00:00:00:03/-1
```

```
**          Caso tenha interesse, execute o  comando flows          **
**          para verificar os fluxos criados pelas intents          **
onos> flows |grep -A 2 intent
```

Sobre os testes acima, é importante ressaltar alguns detalhes:

Verifique que, após o ping de h1 para h3, houve a criação de uma *intent* no sistema (comando **summary**), com a criação de correspondentes 4 regras de fluxos, devido à *intent* recém criada, uma vez que há dois *switches* no caminho da comunicação e criamos duas regras (bidirecional) em ambos.

Podemos ver que o estado das intents é "*INSTALLED*", que são do tipo *HostToHost* e que os critérios (*resources*) são apenas o MAC de origem e destino.

Por fim, note que perdemos o primeiro pacote do ping. Isso só está ocorrendo em novos fluxos e se deve ao fato de que o primeiro pacote está subindo para o controlador, sendo encaminhando para a aplicação que, por sua vez, está criando as *intents* necessárias a partir dele, descartando-o após o processamento (realize um ping entre h1 e h3 novamente e veja que nenhum pacote será perdido).

Para evitar a perda do primeiro pacote, podemos modificar a aplicação para encaminhar o primeiro pacote diretamente para a interface onde o destino se encontra, após o tratamento dele e a criação da *intent*.

Corrigindo a aplicação para evitar a perda de pacotes

No código existe uma linha comentada com "**LAB 1 - Encaminhando o primeiro pacote para o destino**". Siga a instrução do comentário para configurar o encaminhamento do primeiro pacote para o destino e evitar a sua perda.

Após corrigir a aplicação, é necessário compilar o código novamente e carregá-lo no controlador:

```
#> cd ~/sci-2015/aplicacoes/intentSwitch/

#> mvn clean install && onos-app $(ipdocker onos1) reinstall! target/intent-switch-app-1.3.0-SNAPSHOT.oar
```

A seguir, teste o ping novamente entre dois *hosts* quaisquer:

```
h1> ping 192.168.0.4 -c 5
PING 192.168.0.4 (192.168.0.4) 56(84) bytes of data.
64 bytes from 192.168.0.4: icmp_seq=1 ttl=64 time=0.612 ms
64 bytes from 192.168.0.4: icmp_seq=2 ttl=64 time=0.096 ms
… … … …
64 bytes from 192.168.0.4: icmp_seq=5 ttl=64 time=0.079 ms
--- 192.168.0.4 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 986ms
```

Repare que agora o primeiro pacote do ping não foi perdido, conforme o esperado.

Desativando a aplicação

Iremos utilizar a mesma topologia no FIBRE na próxima prática, onde será mostrada uma aplicação simples de Firewall.

Desative as aplicações utilizadas para que elas não interfiram na próxima prática:

```
onos> app deactivate intent.switch.app
onos> app deactivate org.onosproject.proxyarp

onos> purge-intents
```