

Relatório de E.T.

**Relatório nº3 - *Software-Defined Networking and OpenFlow***

**Curso:** METI

**Turno:** 3ª feira 08:00 > 09:30

**Grupo:** Bancada 8

|  |
| --- |
| **Trabalho realizado por:** |
|  |
| Luís Pereira, nº77984 |
| Ruben Condesso, nº 81969 |

**Índice**

**1 -** [**Introdução**](#_tyjcwt) **2**

**2 -** **Laboratório 4 3**

**2.1 - Ponto 3: *Testing VM* 3**

**2.2 - Ponto 4: Testing Mininet 3, 4**

**2.3 - Ponto 5: Test the network 5, 6, 7**

**3 - Laboratório 5 7**

**3.1 - Ponto 4: *Testing Mininet 8***

**3.2 - Ponto 5: *Analysing hub behavior 8, 9***

**3.3 - Ponto 7: *Pox of\_tutorial code for simple hub 9, 10***

**3.4 - Ponto 8*: Implementing a learning switch 10, 11***

# 

# 

# **Introdução**

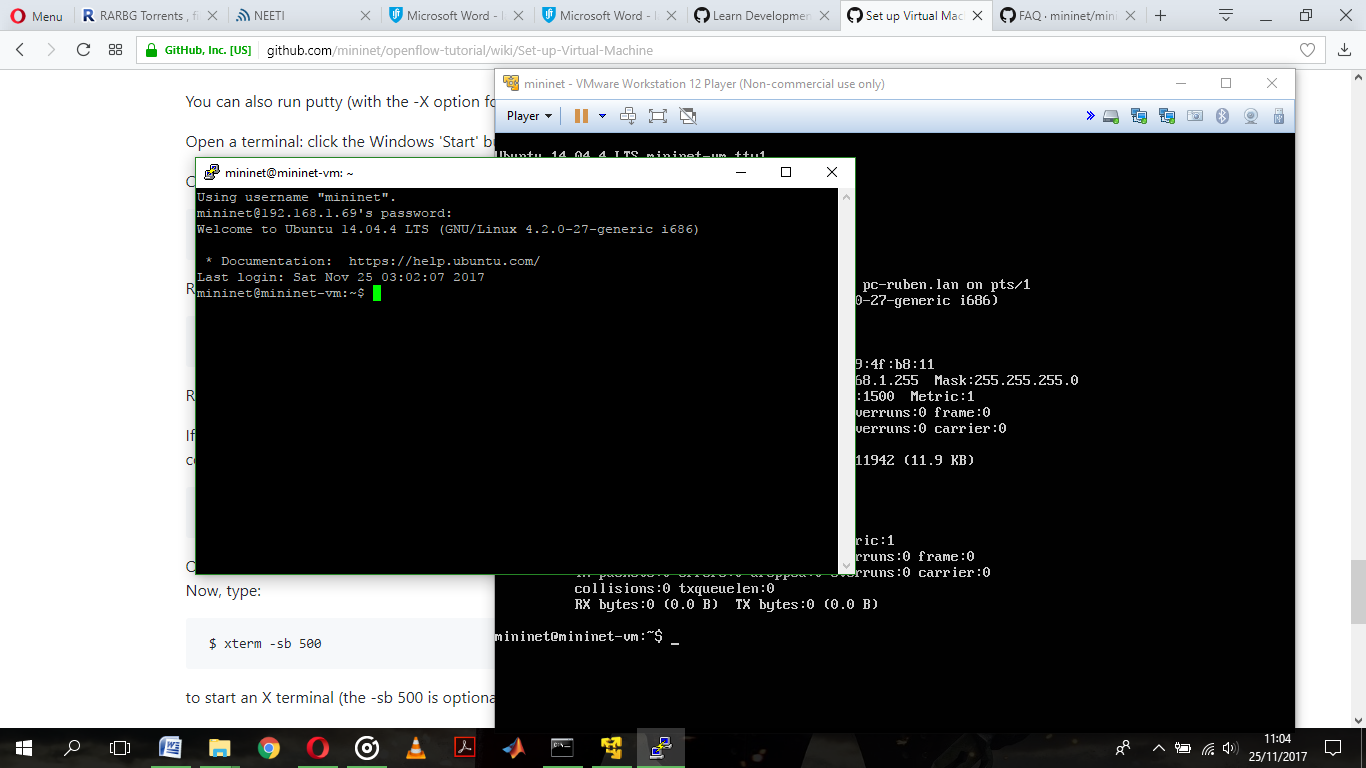
Este relatório aborda o tema "*Software-Defined Networking and OpenFlow"*, e é referente aos laboratórios nº4 e nº5, *Introduction to Mininet* e *Introduction to SDN controllers*, respetivamente. Desta forma, o relatório está dividido em duas partes: a primeira diz respeito ao laboratório nº4 e a segunda ao laboratório nº5.

A junção dos dois guias tem como intuito criar um ambiente de teste para implementar e testar *SDN network*, juntamente com o protocolo *OpenFlow*. *SDN* é uma tecnologia que aborda a computação em *cloud*, facilitando a gestão da rede, e permite a configuração da rede com uma programação mais eficiente, melhorando a sua performance e monotorização. Um dos princípios da *SDN* é que a programação das tabelas de encaminhamento são feitas por um controlador (servidor centralizado), onde concretamente, neste relatório será demonstrado o uso e desenvolvimento de um simples módulo para um POX *controller*. O *OpenFlow* não é mais que um protocolo de comunicação emergente que permite a um servidor de *software* determinar o caminho do encaminhamento de pacotes, que deverá seguir uma rede de *switches*.

# **Laboratório nº4**

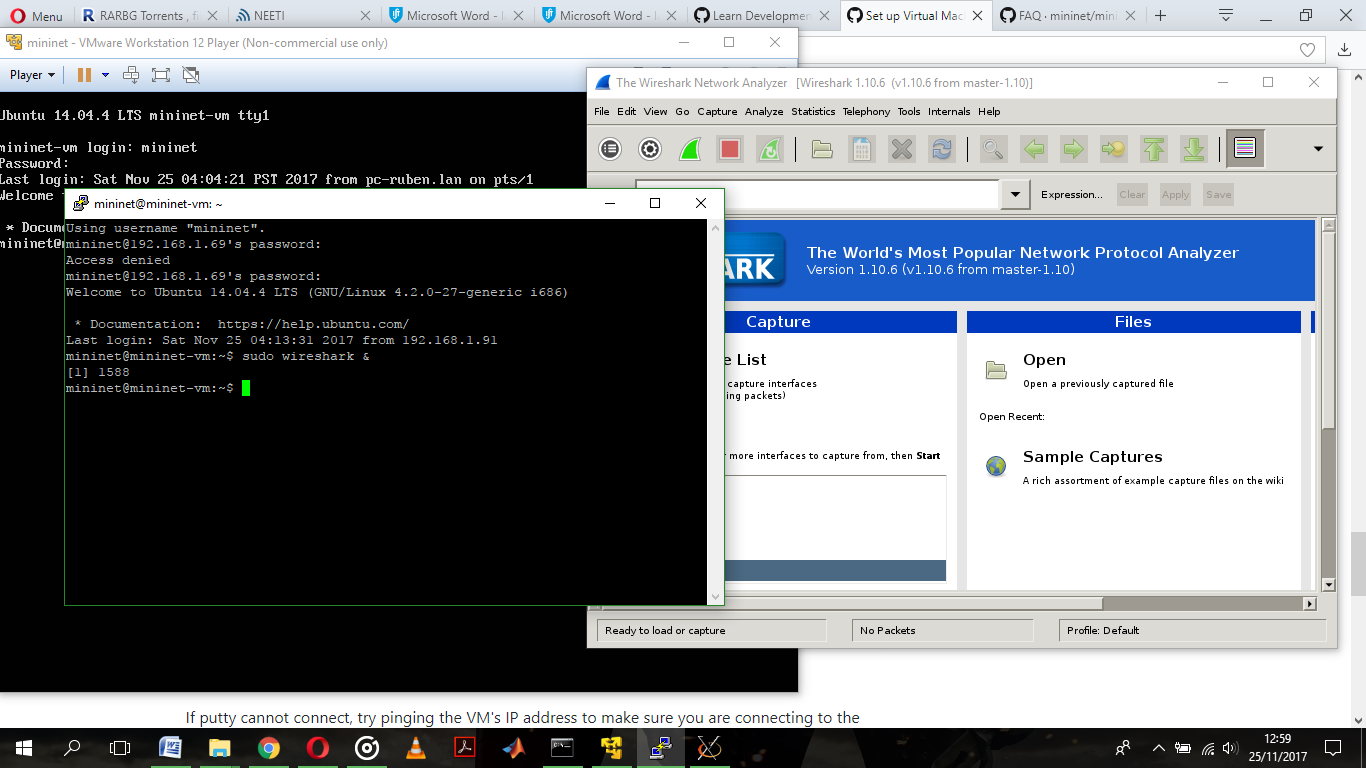
Este laboratório tem como objetivo fazer uma introdução ao *mininet*. Este permite-nos criar uma rede virtual realística, onde é possível correr um código com um núcleo real, numa única máquina virtual, com um único comando, em poucos segundos, daí a sua grande utilidade. O ponto 3 e o ponto 4 do guia de laboratório têm apenas a finalidade de testar o funcionamento do sistema *mininet* e da VM, com o auxílio de alguns comandos.

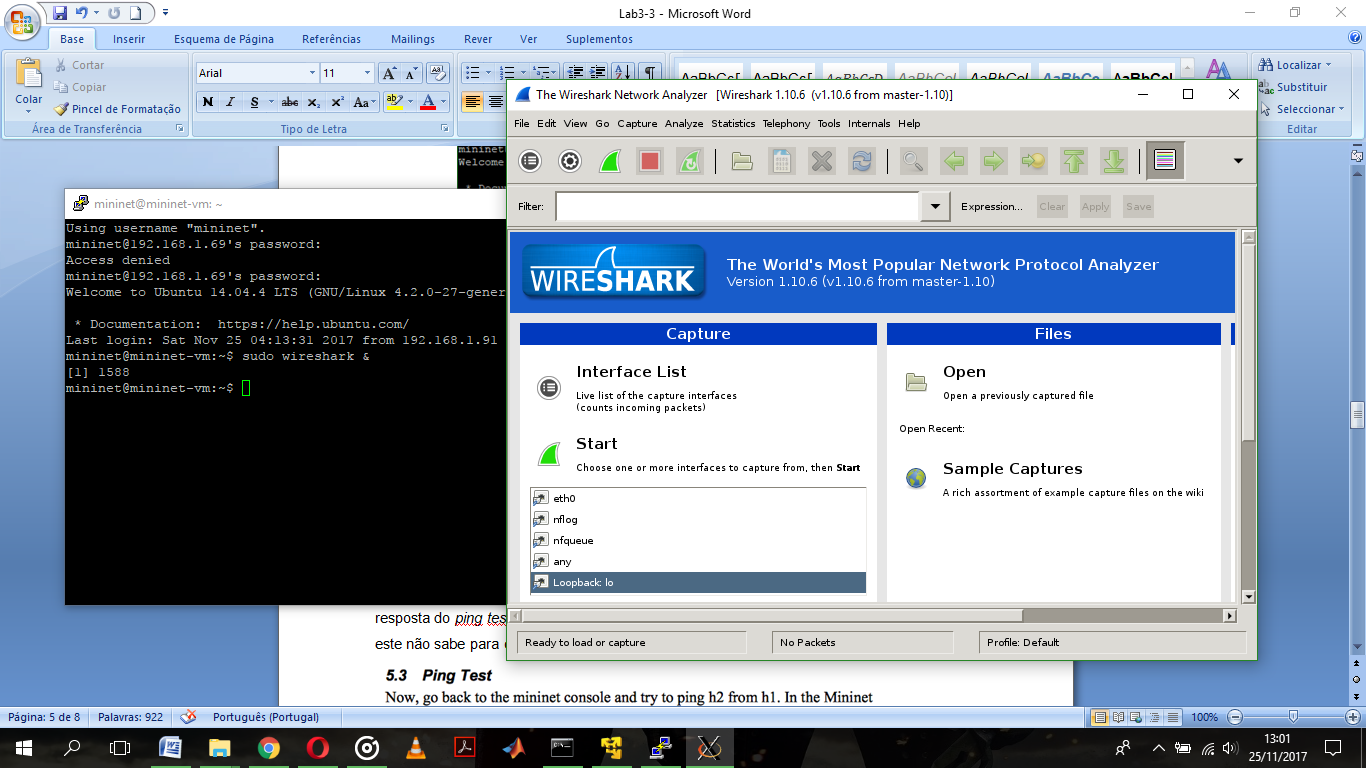
**Ponto 3** - *Testing VM*:

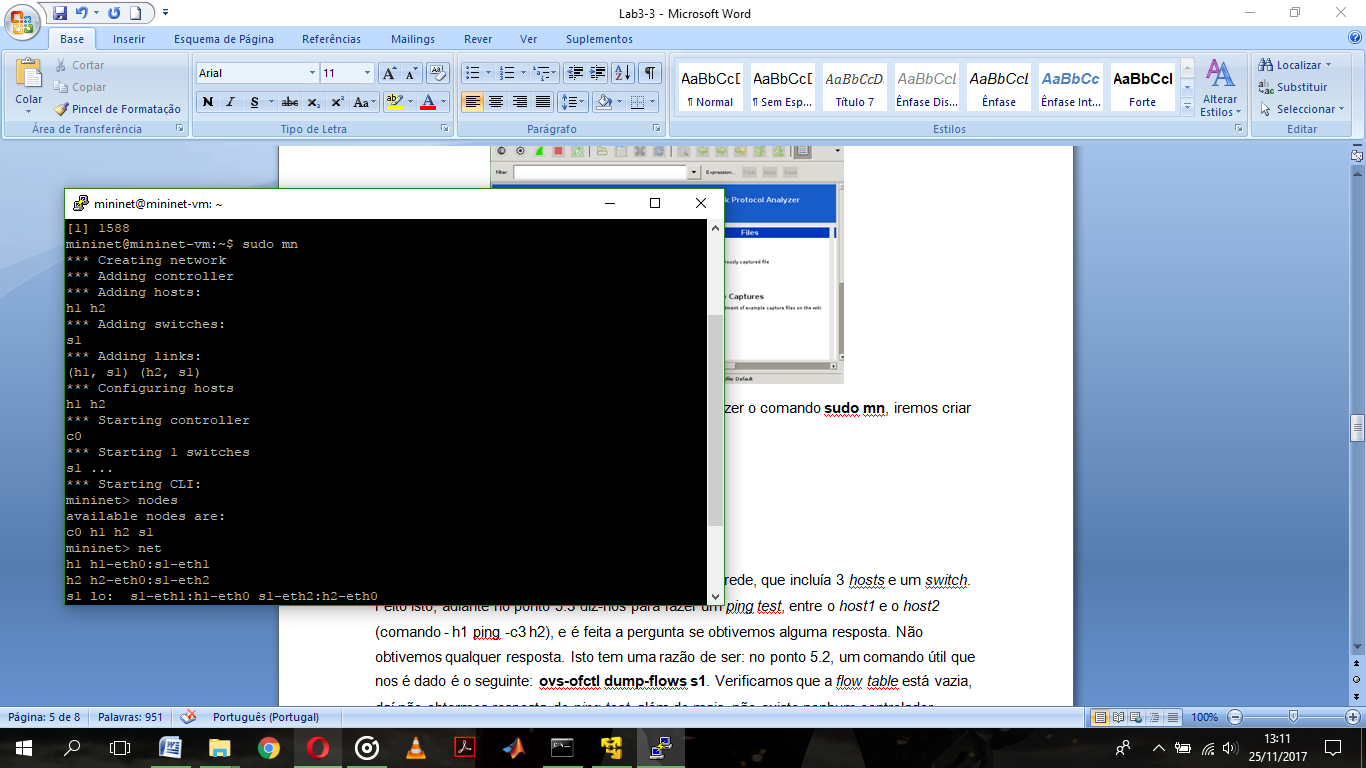
 A partir do *print*, ilustrado em baixo, podemos ver que conseguimos fazer *login* no sistema *mininet,* pelo *X terminal,* a partir do *host system*:

**Ponto 4** - *Testing mininet*:

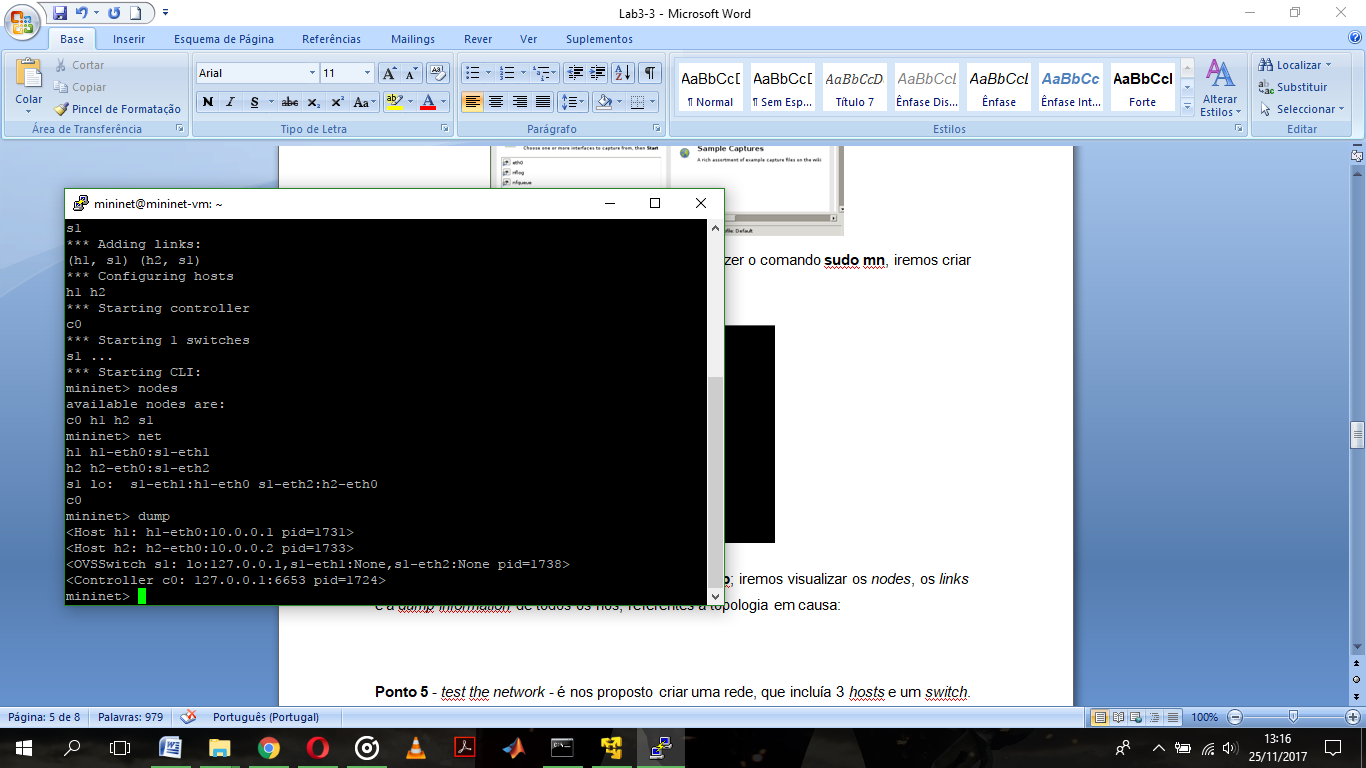
No ponto 4.1 - *start wireshark* - o objetivo é abrir o *wireshark* a partir do *host system,* pelo *X terminal*:



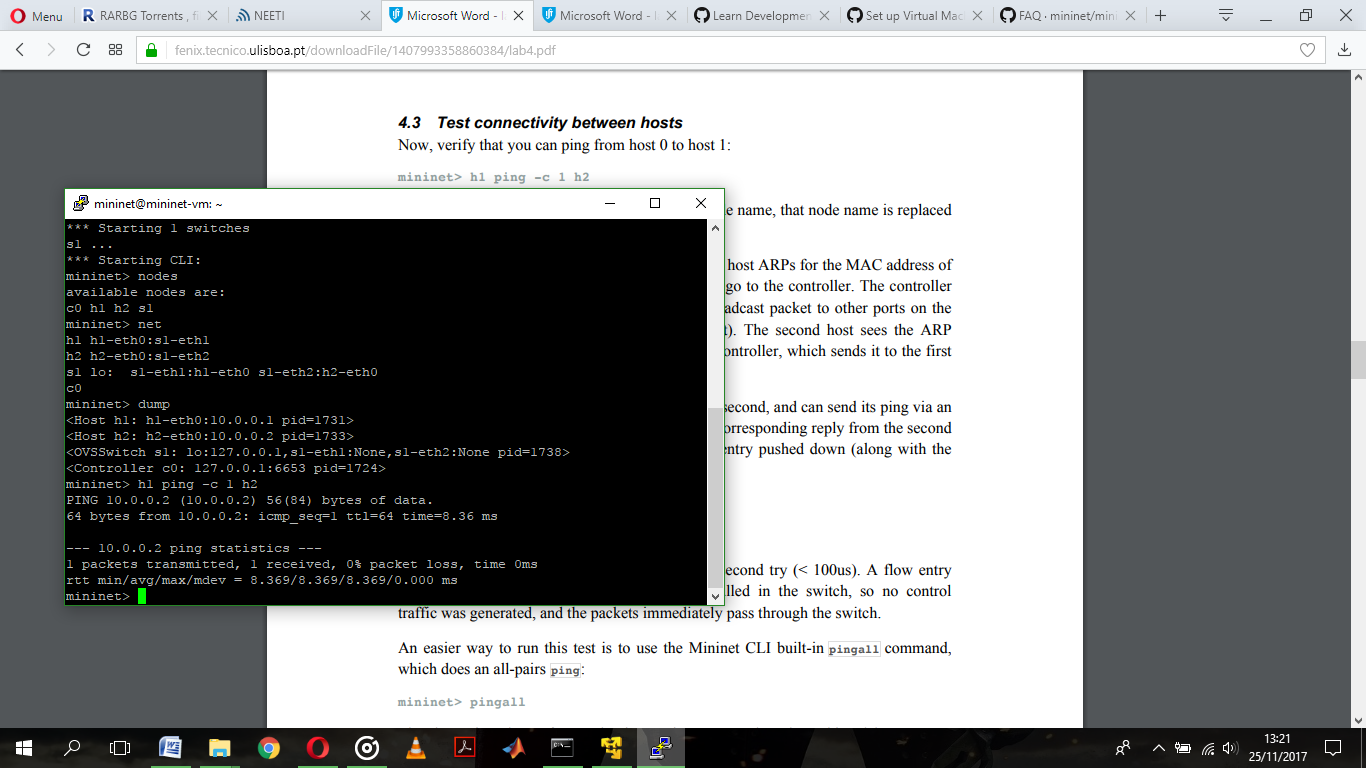


 No ponto 4.2 - *interact with hosts and switches* - ao fazer o comando **sudo mn**, iremos criar uma topologia simples e entrar na CLI:

De seguida, ao fazer os comandos: **nodes**; **net**; **dump**; iremos visualizar os *nodes*, os *links* e a *dump information* de todos os nós, referentes à topologia em causa:



Por fim, no ponto 4.3 - *test conectivity between hosts* - fizemos um *test ping*, entre o *host0* e o *host1*, com sucesso:



**Ponto 5** - *test the network:*

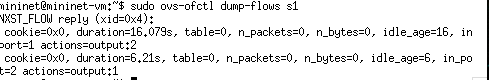
No ponto 5.1 - *network instantiation* - é nos proposto criar uma rede, que inclui 3 *hosts* e um *switch* (comando - **sudo mn --topo single,3 --mac --switch ovsk -- controller remote**). Feito isto, adiante no ponto 5.3 - *ping test* - é nos dito para fazer um *ping test*, entre o *host1* e o *host2* (comando **- h1 ping -c3 h2**), onde não obtivemos qualquer resposta.

Isto tem uma razão de ser: no ponto 5.2 - *ovs-ofctl example usage -* um comando útil que nos é dado é o seguinte: **ovs-ofctl dump-flows s1**. Verificamos que a *flow table* está vazia, daí não obtermos resposta do *ping test*, além do mais, não existe nenhum controlador ligado ao s*witch*, então este não sabe para onde enviar o tráfego recebido. Fica assim respondido à primeira pergunta do ponto 5.3, onde nos é perguntado a razão pela qual não obtivemos resposta no *ping*.

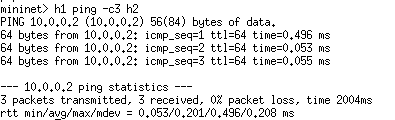
A seguir, executamos os seguintes comandos:

* **ovs-ofctl add-flow s1 in\_port=1,actions=output:2 ;**
* **ovs-ofctl add-flow s1 in\_port=2,actions=output:1;**

Tal, irá gerar *flows* que entram na porta1 e saem na porta2 e vice-versa.

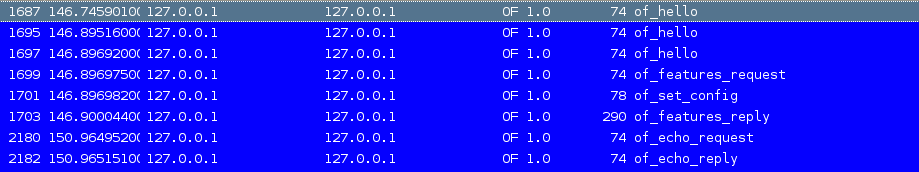
Para verificar a *flow table*, fizemos o seguinte comando: **sudo ovs-ofct1 dump-flows s1**; originando o seguinte *output*:

Com o auxílio do *print* ilustrado acima, podemos ver a tabela completa do *flow*. Fazendo novamente o *ping*, entre o *host1* e o *host2*, podemos verificar que já temos conectividade entre ambos, pelo que o *ping* ocorre com sucesso como podemos concluir pela imagem abaixo:



Existe um aumento de 3 pacotes no *n\_packets*, mas por outro lado o *n\_bytes* não corresponde aos 192bytes supostamente enviados. Fazendo *test* *pings*, entre os outros *hosts*, verificamos que só temos conexão entre o *host1* e *host2*, sendo que nenhum dos outros *hosts* se encontra *reachable*. Isto deve-se à falta dos dois comandos ilustrados acima entre os *hosts* em causa, que causou depois a conectividade entre eles.

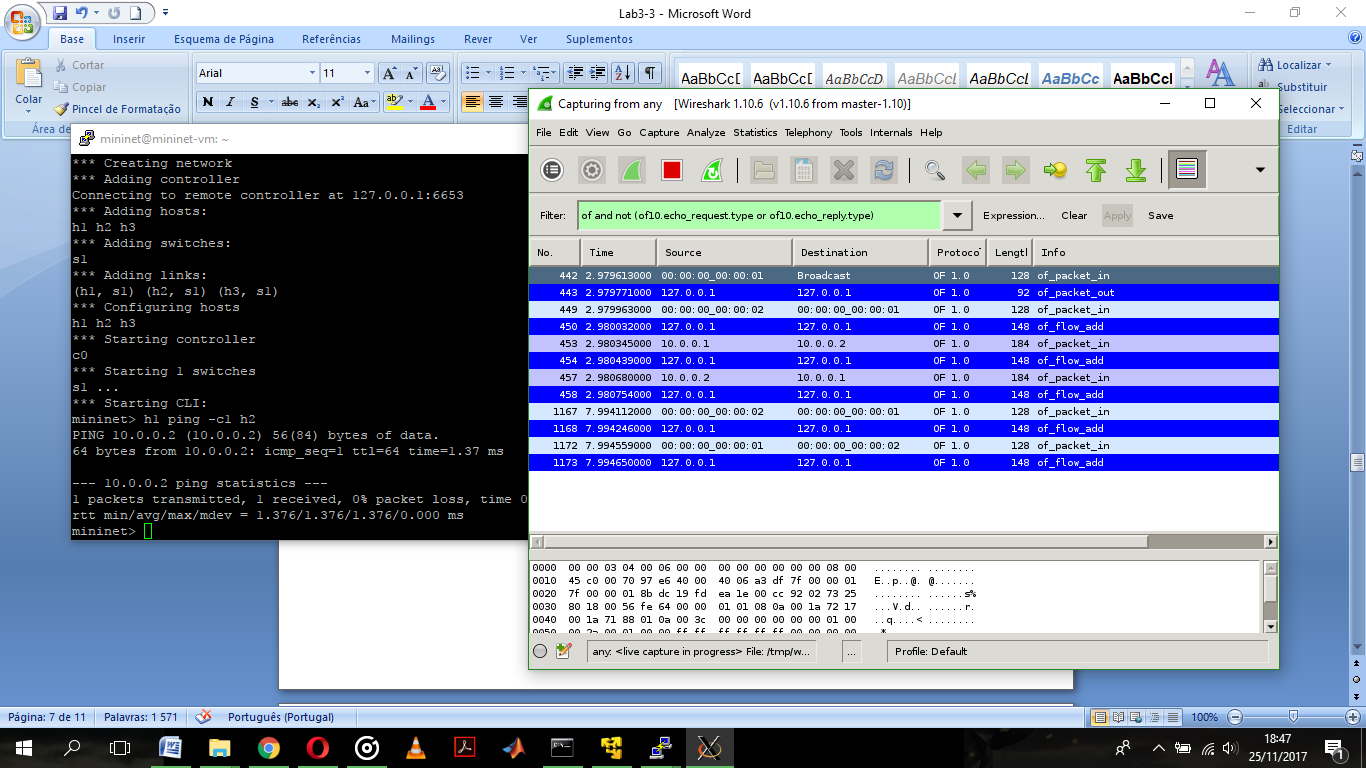
Fazendo o comando sudo **ovs-ofctl del-flows s1**, verificamos que deixamos de ter conetividade entre o *host1* e *host2*, visto que terminamos o *flow* em *s1*.

 No ponto 5.4 - *start controller and view startup messages in Wireshark* - fizemos o comando **controller ptcp:6653**, que irá começar um simples *controller*, no porto 6653, que irá agir como um *learning switch* sem que tenha que ter qualquer entrada de *flows*. As mensagens que obtivemos foram as seguintes:

Podemos visualizar diferentes tipos de mensagens: primeiro temos a troca de mensagens *hello* entre o *controller* e o *switch*, sendo que a primeira é do tipo *controller-switch* e representa o envio do número da versão por parte do *controller* para o *switch*; e de forma análoga, a segunda mensagem é do tipo *switch-controller*, e representa a resposta do *switch* para o controller*,* contendo o número da sua versão.

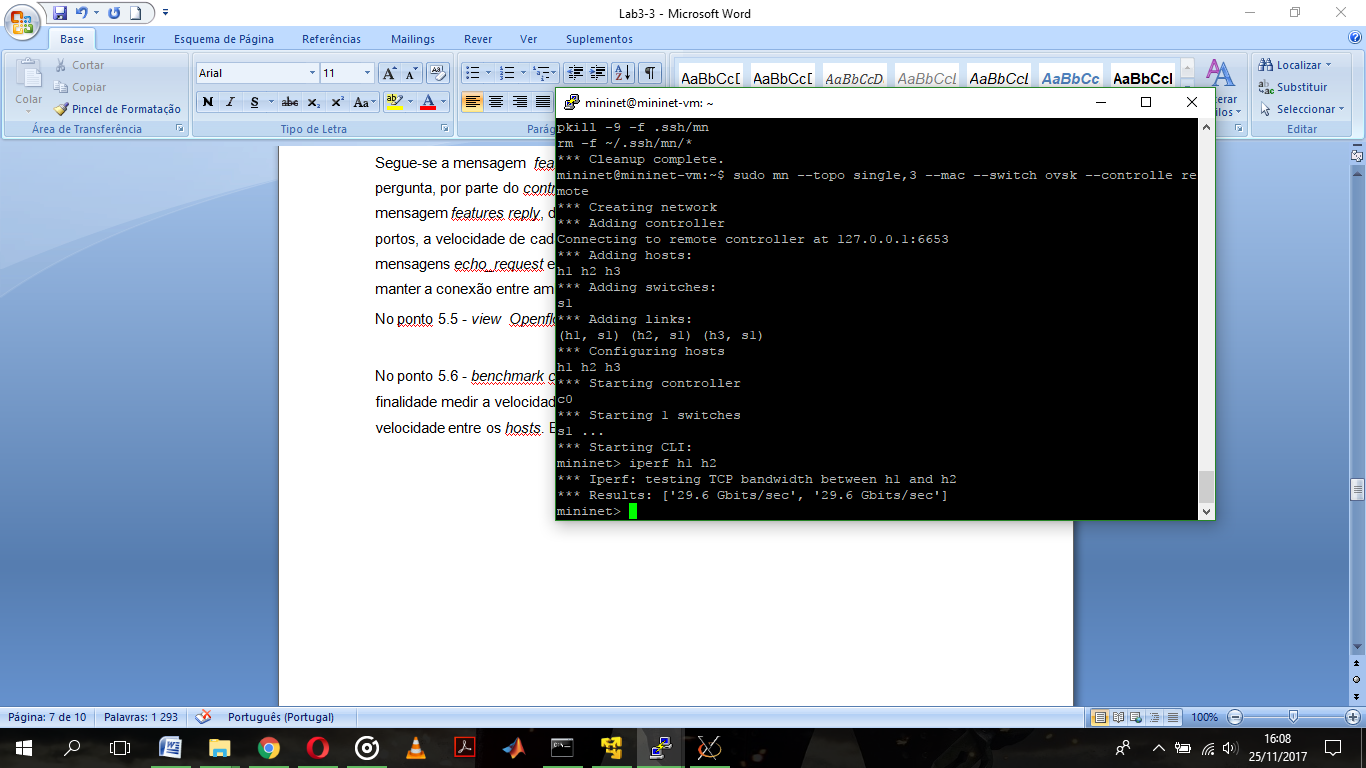
Segue-se a mensagem *features request*, que é do tipo *controller-switch*, onde é feita a pergunta, por parte do *controller*, que portos se encontram disponíveis. Depois vem a mensagem *features reply*, do tipo *switch-controller*, onde o *switch* responde com a lista de portos, a velocidade de cada porto, e as tabelas e ações que suporta. Por fim, vem as normais mensagens *echo\_request* e *echo\_reply* trocadas entre os dois, que tem como finalidade manter a conexão entre ambos "viva".

O ponto 5.5 - *view Openflow messages for Ping* - tem como objetivo ver as mensagens geradas, nas respostas dos pacotes. Usando o seguinte filtro no *wireshark*: **of and not (of10.echo\_request.type or of10.echo\_reply.type)**, seguido de um *ping* entre o *host1* e o *host2*, obtivemos as seguintes mensagens:

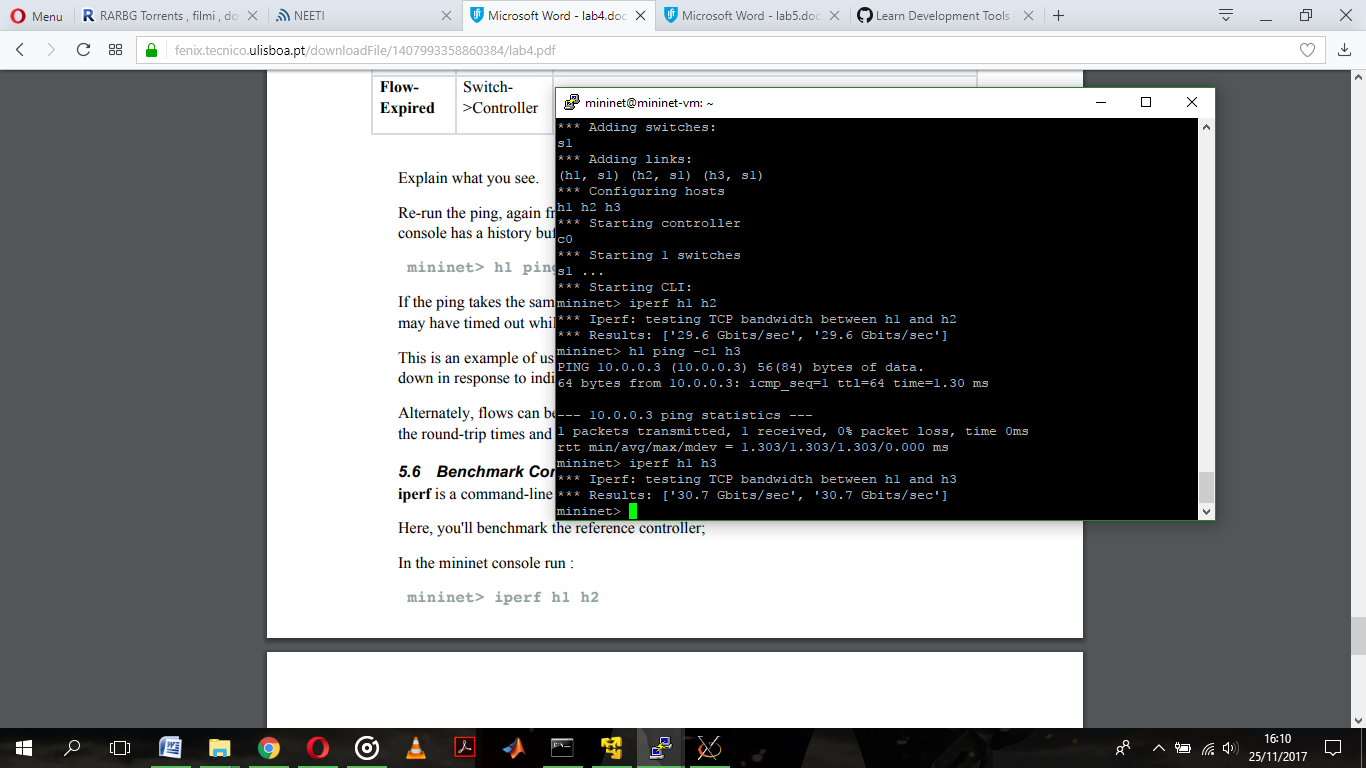


Podemos verificar as mensagens *packet\_in*, do tipo *switch-controller*, que significam que foi recebido um pacote que não constava ainda na *flow table* do *switch*, razão pela qual foi enviado para o *controller* (tem como origem *host1* e como destino o *host2*, ou vice-versa sendo que há uma resposta dos dois lados no *ping*). Pode-se ver também a mensagem *packet\_out* (tendo como origem o *host1*), do tipo *controller-switch*, que representa o envio de pacote para um ou mais portos. Finalmente, podemos enviar mensagens *flow\_add*, do tipo *switch-controller*, que significa que foi adicionado um *flow* específico à *flow table* do *switch*.

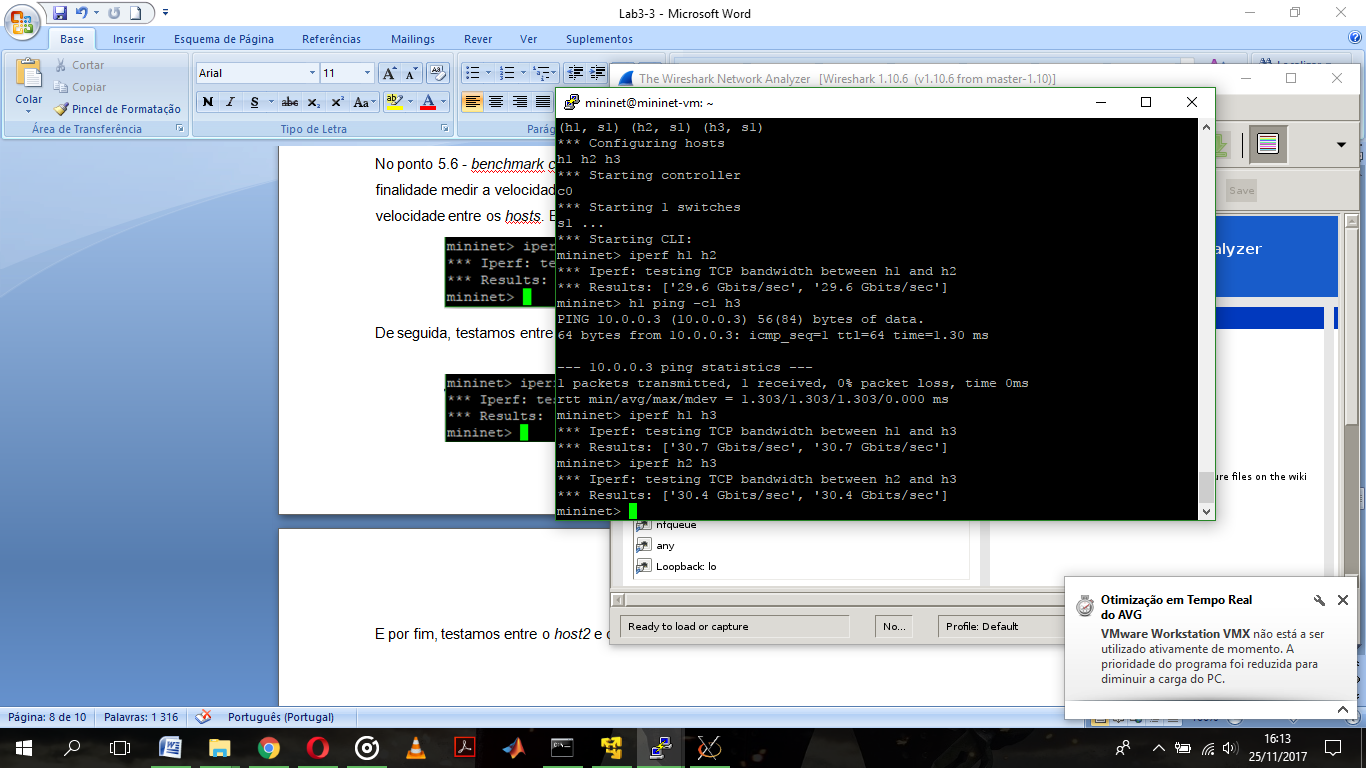
No ponto 5.6 - *benchmark controller w/iperf* - é utilizado o comando **iperf** que tem como finalidade medir a velocidade entre computadores, neste será caso será feito a medição da velocidade entre os *hosts.* Entre o *host1* e o *host2*, obtivemos a seguinte velocidade:



De seguida, testamos entre o *host1* e o *host3*, obtivemos a seguinte velocidade:



E por fim, testamos entre o *host2* e o *host3*:

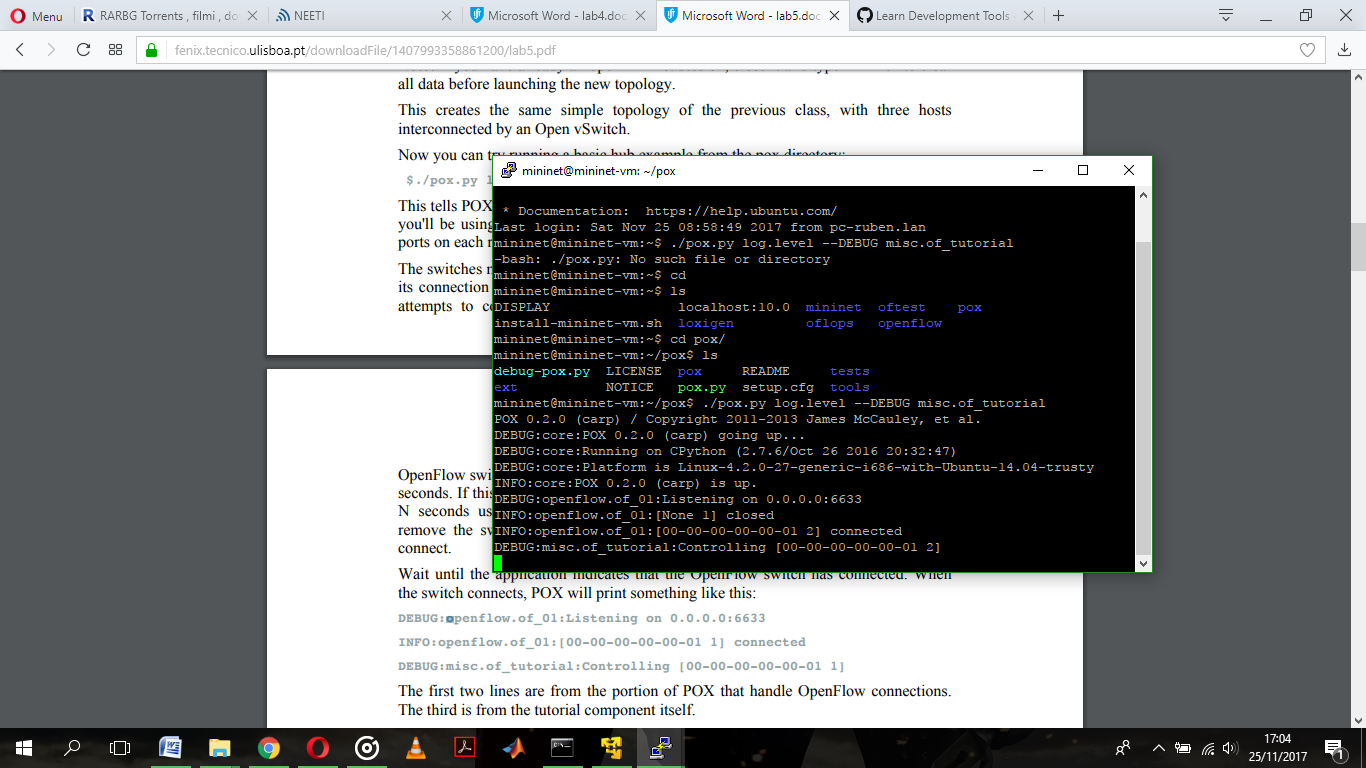


Podemos verificar que as três velocidades se aproximam bastante umas das outras, isto deve-se ao facto de estarmos a testar numa topologia bastante simples, o que leva a estas aproximações de resultados.

# **Laboratório 5**

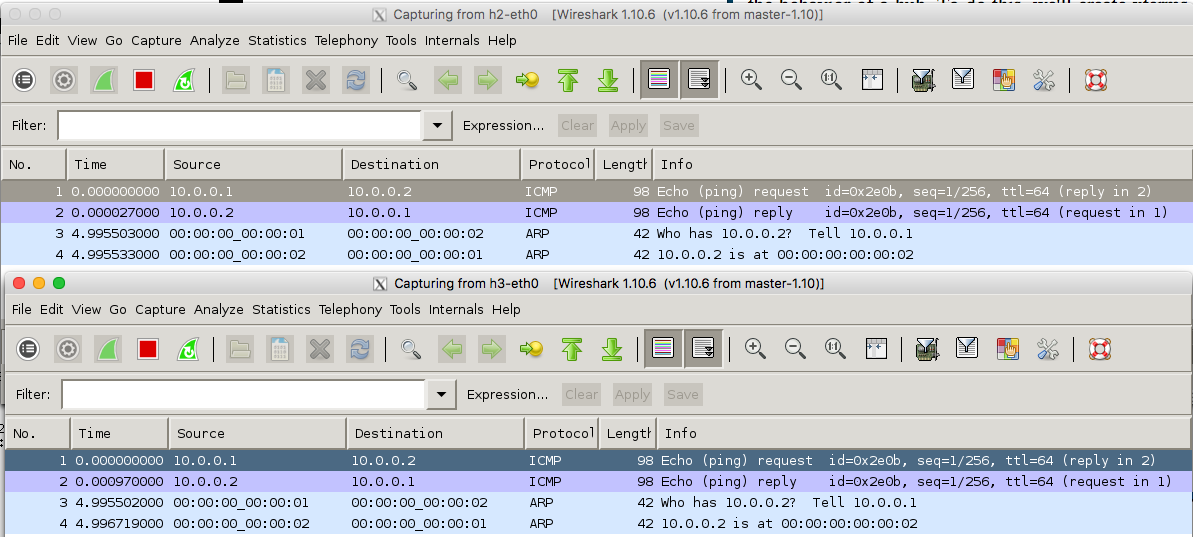
Este laboratório tem como finalidade complementar o anterior, com uma introdução aos *SDN controllers*, continuando a usar o *mininet.* Vai ser desenvolvido ao longo do laboratório, um simples módulo para o controlador POX, onde este oferece uma *framework* para comunicar com *SDN switches*, usando o *OpenFlow*.

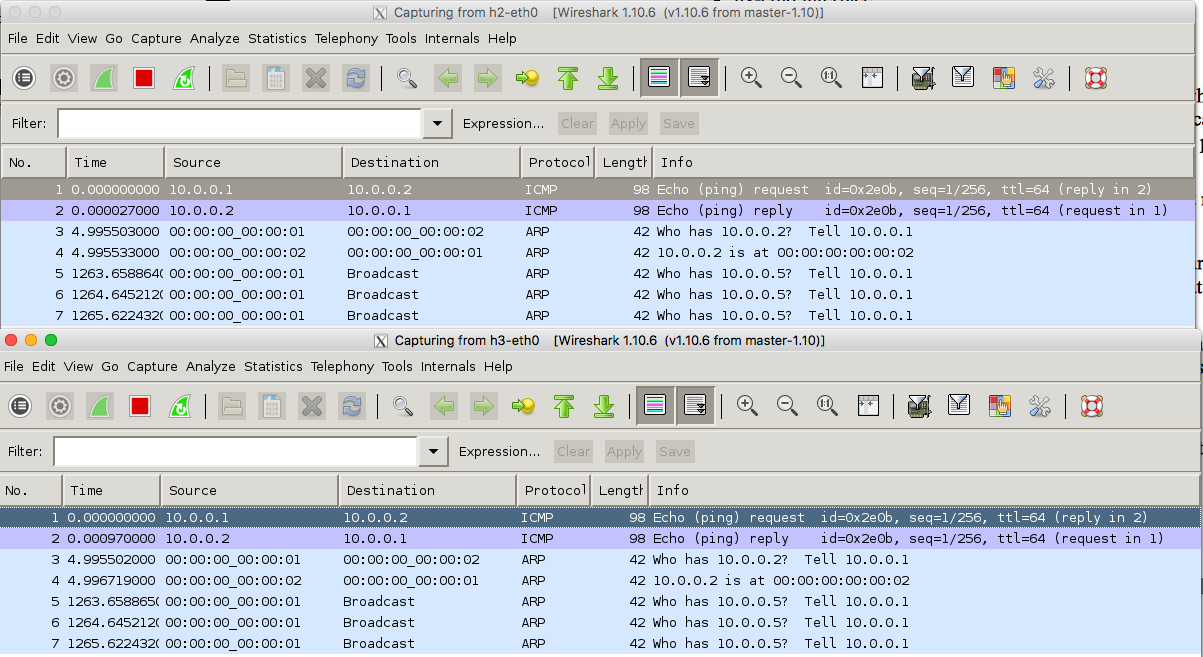
**Ponto 4** - *starting mininet* - criamos a mesma topologia usada no laboratório anterior. De seguida, lançamos o exemplo de um *hub* simples, a partir do diretório do POX, com o comando: **./pox.py log.level --DEBUG misc.of\_tutorial**. Obtivemos as seguintes respostas do POX:



**Ponto 5** - *Analysing Hub Behavior*:

Vamos verificar que cada *host* consegue fazer *ping* para os restantes, e cada um deles irá visualizar o mesmo tráfego (o comportamento do *hub*).

 Realizado o comando **ping -c1 10.0.0.2**, podemos verificar, que no *host2* e no *host3,* são recebidas exatamente as mesmas mensagens, como podemos verificar na figura abaixo:

 Da mesma forma, podemos verificar que acontece o mesmo fazendo *ping* para um *host* que não existe como é o caso do 10.0.0.5:

Enviando 10 pacotes do *host1* para o *host3* apresentamos os resultados:

* Max RTT: 44.542 ms;
* Min RTT: 2.888 ms;
* Avg RTT: 21.187 ms

Enviando 10 pacotes do *host1* para o *host2* apresentamos os resultados:

* Max RTT: 45.115 ms;
* Min RTT: 4.366ms;
* Avg RTT: 22.527 ms

Os resultados são praticamente os mesmos uma vez que o *hub* envia sempre para todos os *hosts* a perguntar quem é o destinatário do pacote e só depois o entrega.

**Ponto 7** - *POX of\_tutorial code for simple* hub:

Iremos explicar o código sobre o método *act\_like\_hub()* descrito no ficheiro *of\_tutorial.py* que se localiza na pasta pox/pox/misc.

|  |
| --- |
|  |
| def act\_like\_hub(self,packet,packet\_in):  self.resend\_packet(packet\_in,of.OFPP\_ALL); |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |  |
|  |
|  |
|  |
|  |  |

Ao usarmos esta função, estaremos a reenviar (chama a função *resend\_packet*) os pacotes para todas as portas do *hub* (*of.OFPP\_ALL*), excepto a de *input* (de onde vem o pacote), uma vez que não sabemos quem irá ser o recetor. Iremos criar então uma *action*, para cada porto específico passado como argumento na função, para que estes pacotes sejam enviados para as respetivas portas. É também emitida uma mensagem para o *switch*, em como fez esse reenvio.

**Ponto 8** - *implementing a learning switch*:

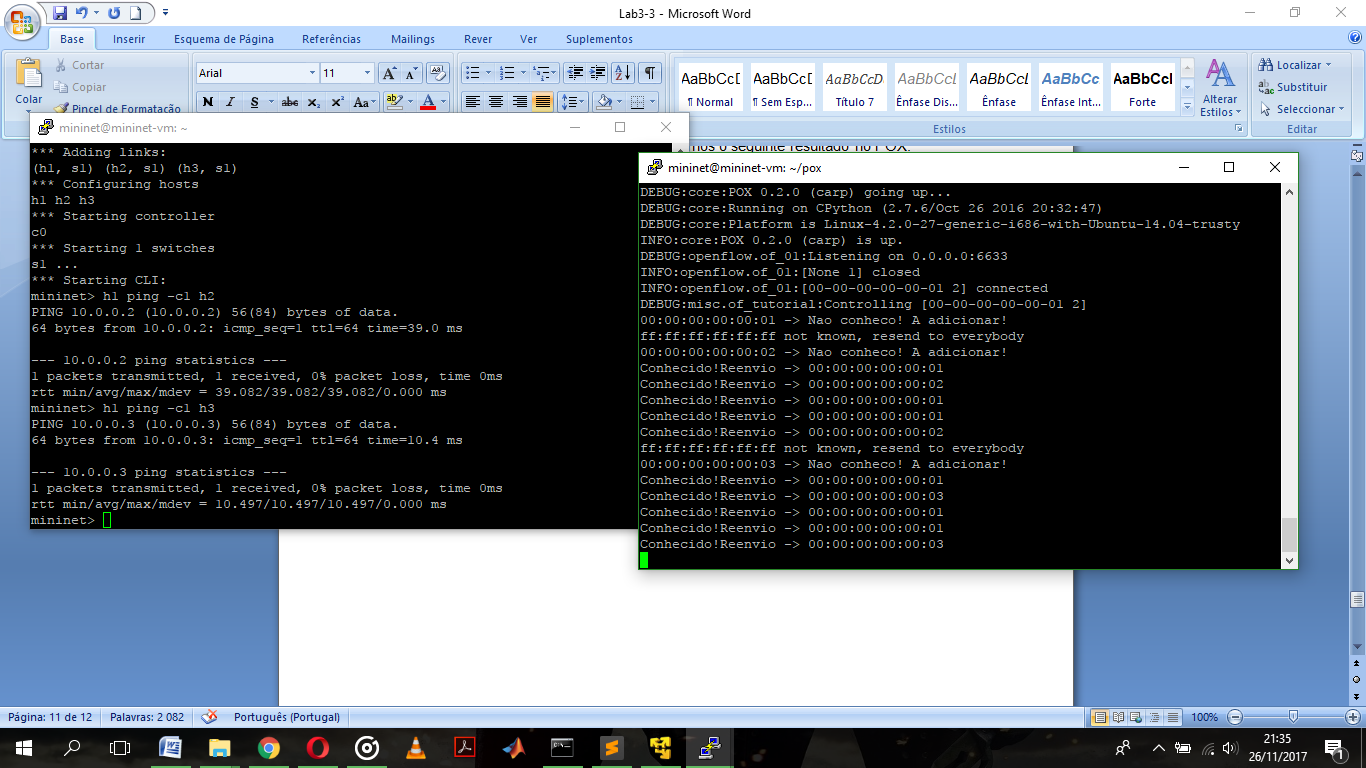
No ponto 8.1 - *implementing a learning switch without installing table flows entries* - tivemos que editar o ficheiro *of\_tutorial*, mais concretamente tivemos que completar a função *act\_like\_switch()*, que se encontrava nesse ficheiro de modo a respeitar as seguintes regras: para cada pacote que chegasse ao controlador se o endereço de origem do *mac* não fosse conhecido antes, teria de se registado a associação entre esse endereço e o  *input port*, no dicionário; e para a mesma situação, teríamos de enviar o pacote para todos os *output ports*; caso o destino do *mac* estivesse já associada a um porto específico, teríamos de enviar apenas para o porto indicado. Posto isto, alteramos o ficheiro *of\_tutorial* de modo a seguir as regras impostas, dando origem ao ficheiro: *of\_tutorial\_8.1*.

Começamos por comentar a chamada da função *act\_like\_hub()*, na função *handlepacketIn()* e retiramos os comentários na mesma função na chamada de *act\_like\_hub()*, para quando chegasse um pacote fosse esta a função a ser chamada e não a anterior.

Depois criamos dois *if's*, um para caso o *destination mac* tivesse já associado a um porto conhecido e outro para caso o *source mac* fosse conhecido. Se entrasse no primeiro *if*: iríamos chamar a função *resend\_packet()*, para reenviar o pacote em causa mas apenas para o porto apropriado (*self.mac\_to\_port[packet.dst])*. Caso entrasse no segundo *if:* iríamos primeiro associar o *source mac* ao *input port* no dicionário (*mac\_to\_port*), e depois iríamos reenviar o pacote para todos os *output ports (self.resend\_packet(packet\_in, of.OFPP\_ALL))*, tal como tínhamos visto na função *act\_like\_hub()*. Se não entrasse em nenhum dos dois *if´s*, iria para um *else* onde iria reenviar o pacote para todo o lado, excepto o *input port*.

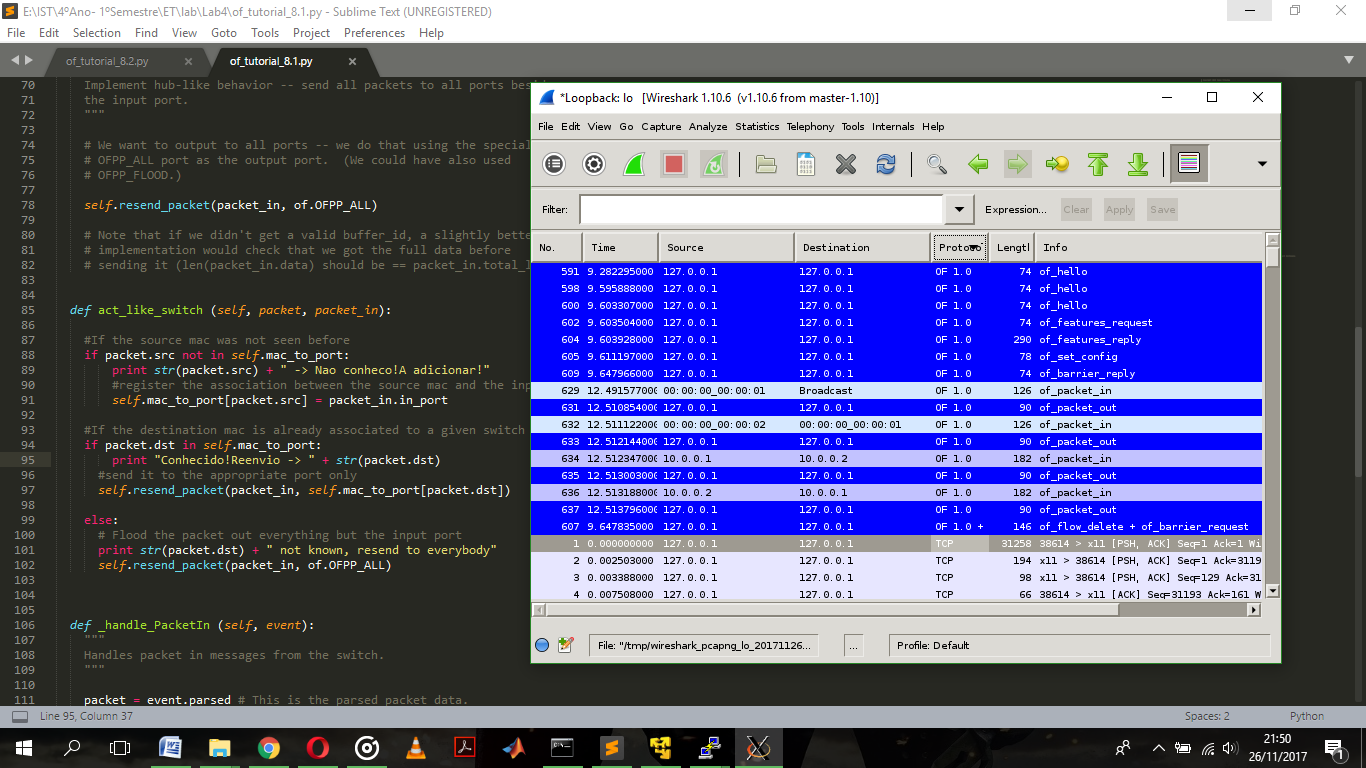
Para verificar o funcionamento da função fizemos os requeridos testes que se encontram no enunciado.

Fazendo um *ping* entre o *host1* e o *host*2, e entre o *host1* e o *host3*, obtivemos os seguintes resultados no POX:



Pode se verificar os *prints* colocados na função *act\_like\_switch*, primeiro quando recebe o *ping*, de um destino que não conhece e depois adiciona ao dicionário, e posteriormente quando o *destionation mac* já está associado a um porto específico e é reenviado para o porto em causa.

No *wireshark*, também foi possível visualizar a troca de mensagens aquando o processo de ligação do POX e *mininet*, e posteriormente quando foi feito um *ping* entre o *host1* o *host2*:



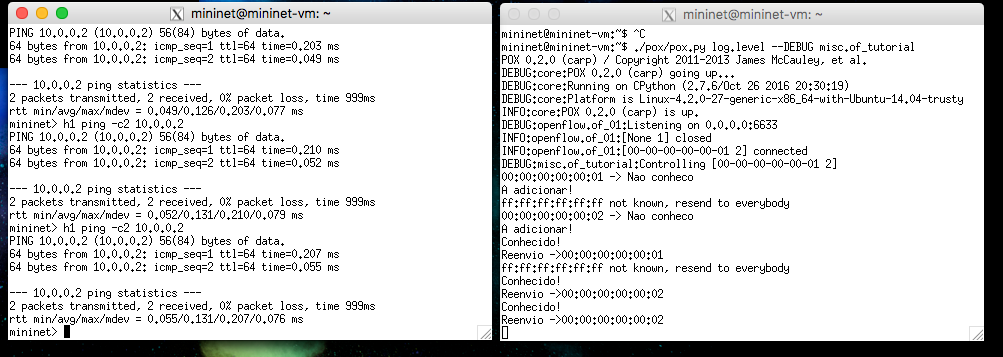
Podemos ver as mensagens trocadas entre o *controller* e *switch*, quando foi iniciada a rede. Ou seja, podemos visualizar as mensagens *hello*, *features\_request*, *features\_reply*, tal como foram descritas no ponto 5.4, do laboratório 4.

E podemos visualizar, igualmente, as mensagens, aquando foi feito o *ping*, concretamente as mensagens *packet\_in* e *packet\_out*, que também foram referidas no laboratório 4, mas agora no ponto 5.5. Podemos assim verificar as mensagens enviadas para o *controller*, quando é feito o *ping*, e não são visualizadas mensagens *flow\_add* porque neste exercício não há a adição de *flows* á *flow table*.

O ponto 8.2 - *implementing a learning switch with table flows entries* - tem como diferenças do exercício anterior, que desta vez não iremos enviar os pacotes todos para o controlador. Iremos criar *action rules,* cada vez que um novo mapeamento é aprendido.

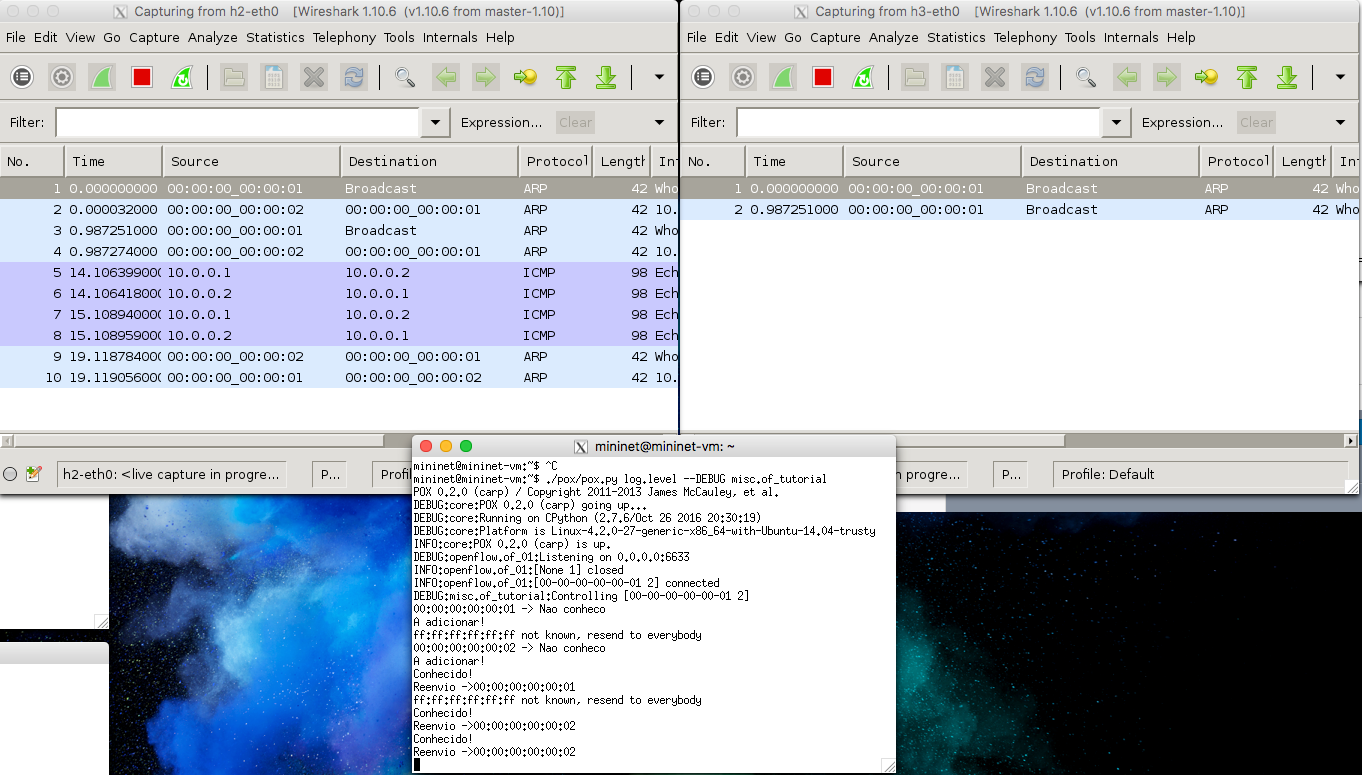
Mantivemos a estrutura de código, na função, relativamente ao exercício anterior, com a diferença no primeiro *if*. Se entrasse nesse *if*, iríamos criar uma *action*, tal como na função *resend\_packet()*, só passaríamos o porto apropriado, dando origem à linha de código: *action = of.ofp\_action\_output(port = self.mac\_to\_port[packet.dst] );* e posteriormente iremos associar essa *action* a uma mensagem, e enviar a mesma ao *switch*. O ficheiro relativo a este exercício está titulado como *of\_tutorial\_8.2*.

Mais uma vez, seguimos os testes referidos no enunciado, de modo a verificar o funcionamento da função.

Fazendo um *ping*, o *host1* e o *host2*, obtivemos o seguinte resultado:

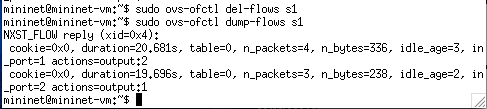
Pode-se verificar pela figura do lado esquerdo que foi feito dois *pings*, entre o *host1* e o *host2*, sendo que paralelamente, pela figura do lado direito, só houve um pedido ao *controller*. Desta vez, os pacotes não foram todos enviados para o *controller*, como acontecia no exercício anterior, uma vez que da segunda vez que fazemos o ping ele tem uma action que determina qual é a porta para onde é encaminhado o trafego, não sendo necessário o envio em broadcast para todas as portas.

Com o auxílio do *wireshark*, fizemos a mesma ação: um *ping* com dois pacotes, do *host1* para o *host2*, obtendo assim o seguinte resultado:



Como se pode verificar pela figura acima, ao fazer o *ping*, entre os dois *hosts*, houve o envio do primeiro pacote seguido do envio do segundo pacote, onde desta vez o *controller* já sabia para onde enviar (dado o adicionamento à *flow table* na primeira vez), e desta forma enviou logo o pacote para o porto correto.

Para verificar as flow tables executamos o seguinte:



Após executar o comando sudo *ovs-ofctl del-flows s1* percebemos que não temos conectividade entre *host1-host2*, *host1-host3* e *host2-host3*. De seguida ao fazermos os *pings*, tal como pedido no exercício, percebemos que a *flow table* fica com *actions* novas criadas pelo código do exercício 8.2. Assim, ao fazermos um *ping* entre o *host1* e o *host2*, iremos inserir *flows* na tabela, e dessa forma o controlador da segunda vez já não irá receber os pacotes.