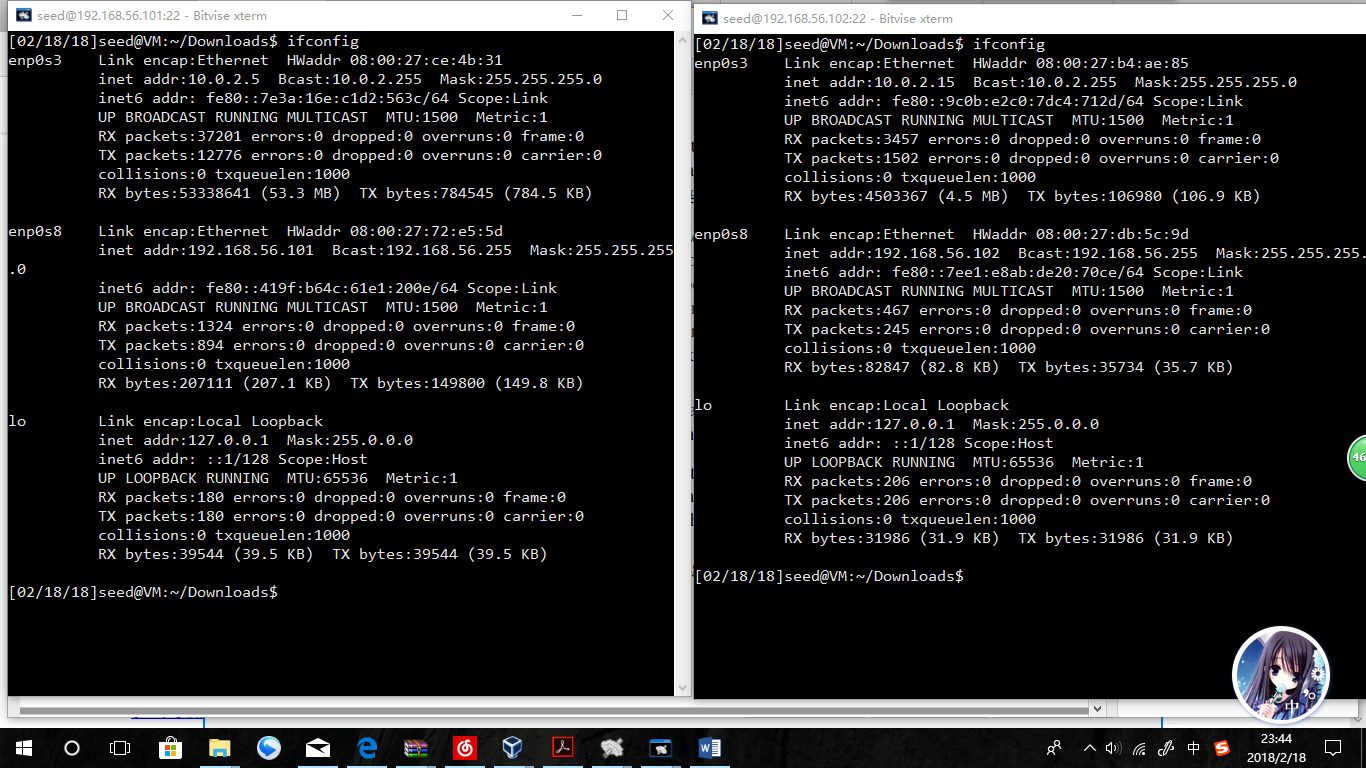
**Report for Lab3**

**3.1 Task 1: Create a Host-to-Host Tunnel using TUN/TAP**

**1. Launch two virtual machines.**



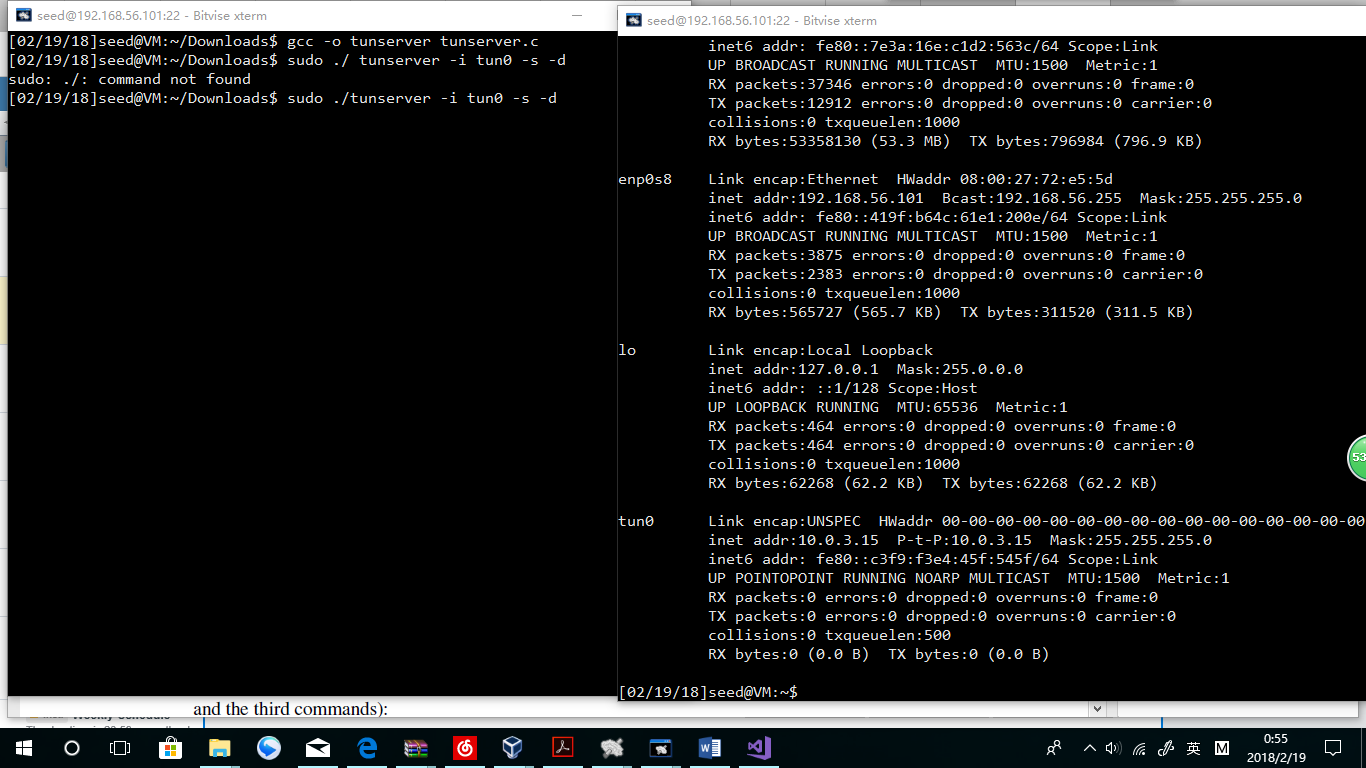
Observation:

2 VMs are running. They both have the NAT network interface cards and Host-only(mainly for SSH ) NICs. Machine A is 10.0.2.5, B is 10.0.2.15

**2. Tunnel Point A:**

**Running the tun0:**

**sudo ./tunserver -i tun0 -s -d**

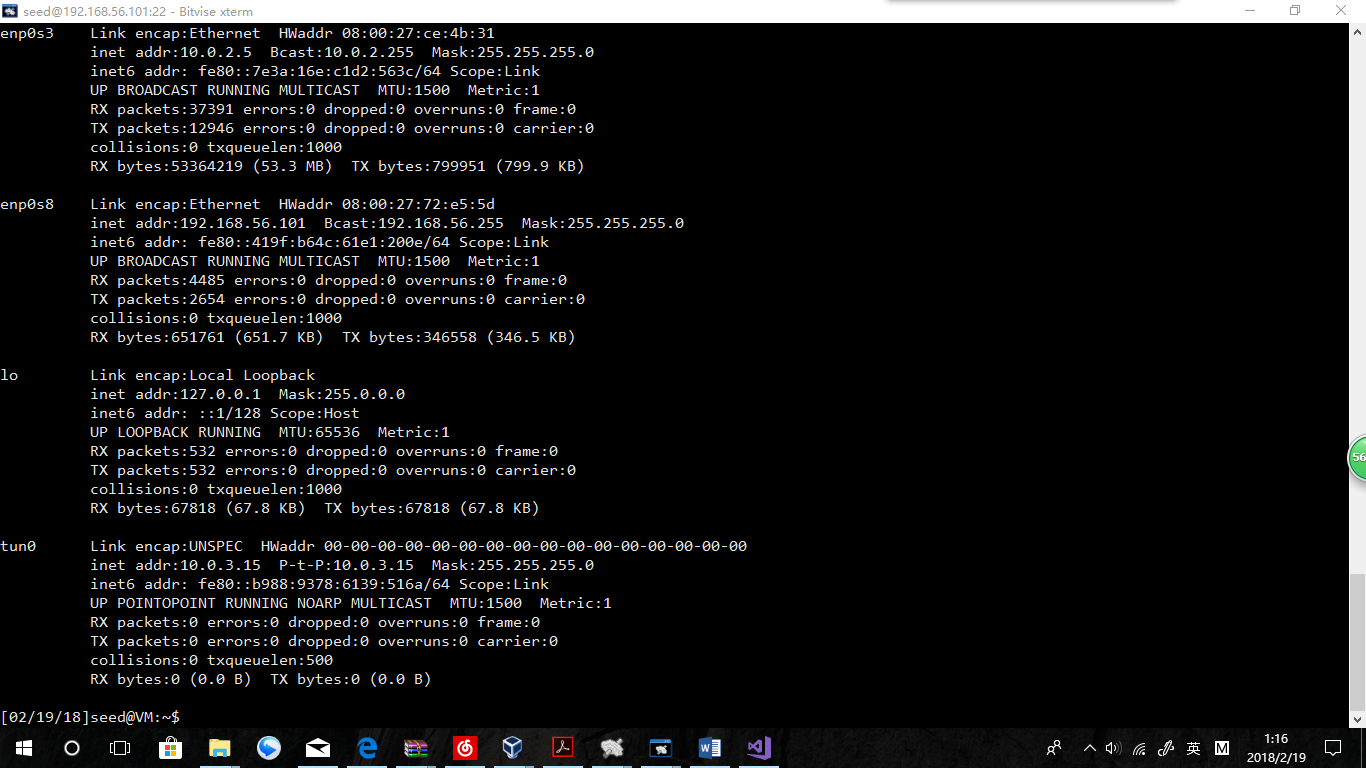


Observation:

Obviously those are 2 SSH terminal windows of Machine A. The left window shows that tun 0 is set up by the c file I just compiled. The right one shows that tun 0 is shown In ifconfig and has a right configure setting.

# ip addr add 10.0.3.15/24 dev tun0

# ifconfig tun0 up

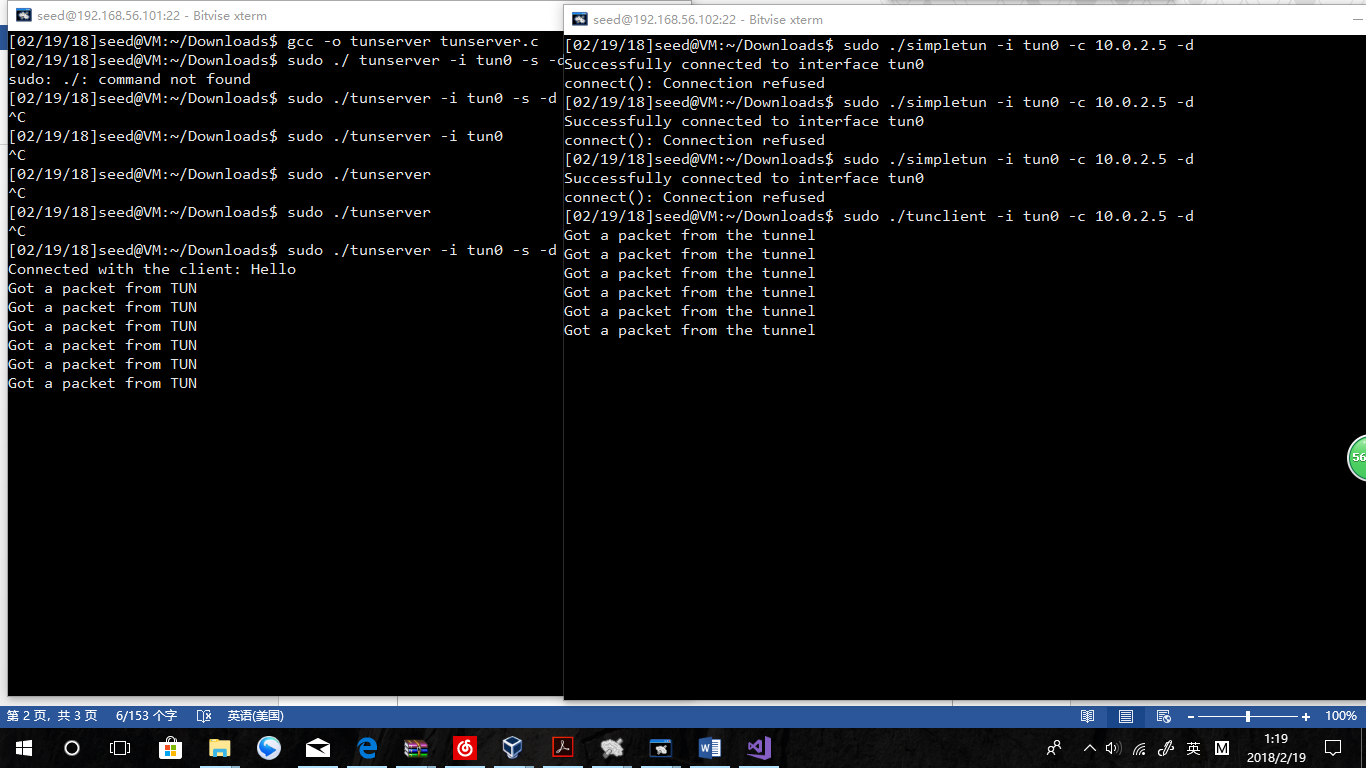


Observation:

The settings are correct! A is successfully settled.

**3. Tunnel Point B:**

**./tunclient -i tun0 -c 10.0.2.5 -d**



**Observation:**

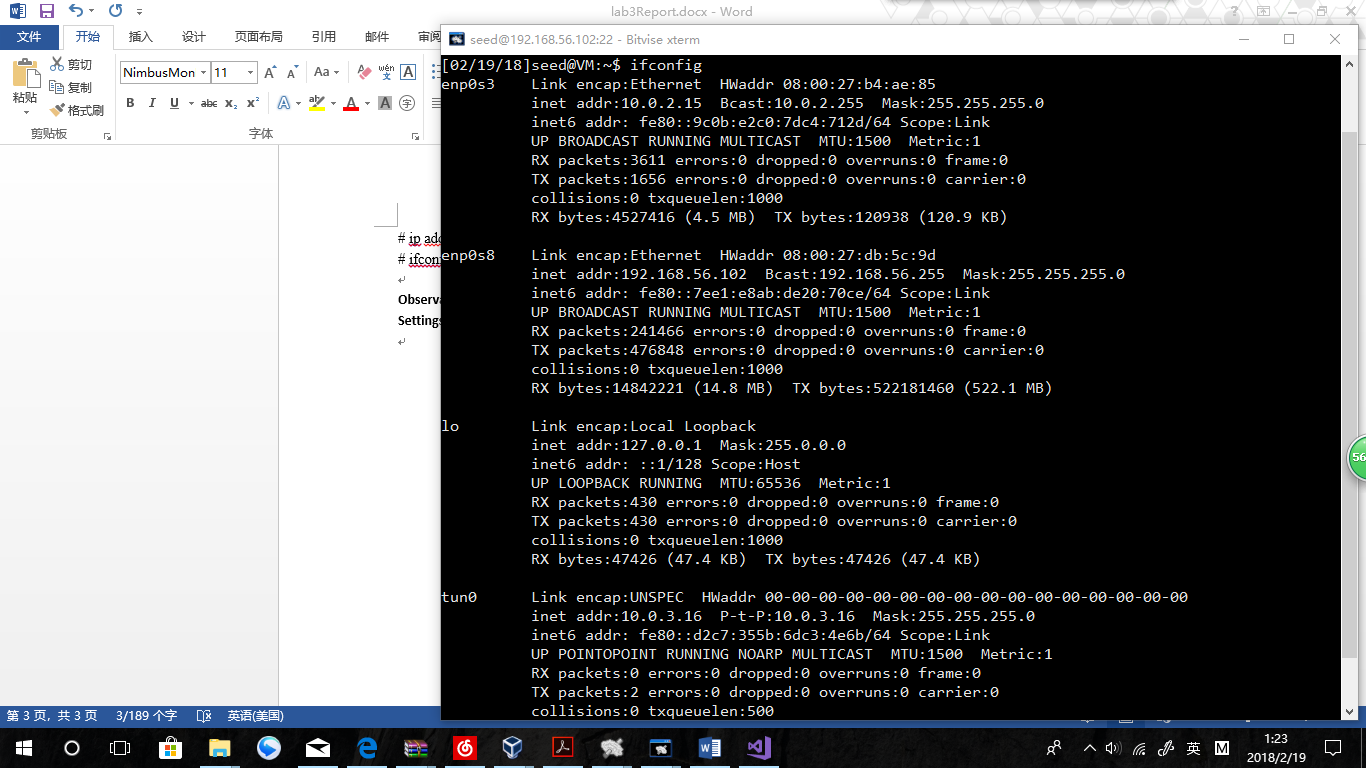
**A is receiving packets from TUN, while B is receiving packets from tunnel.**

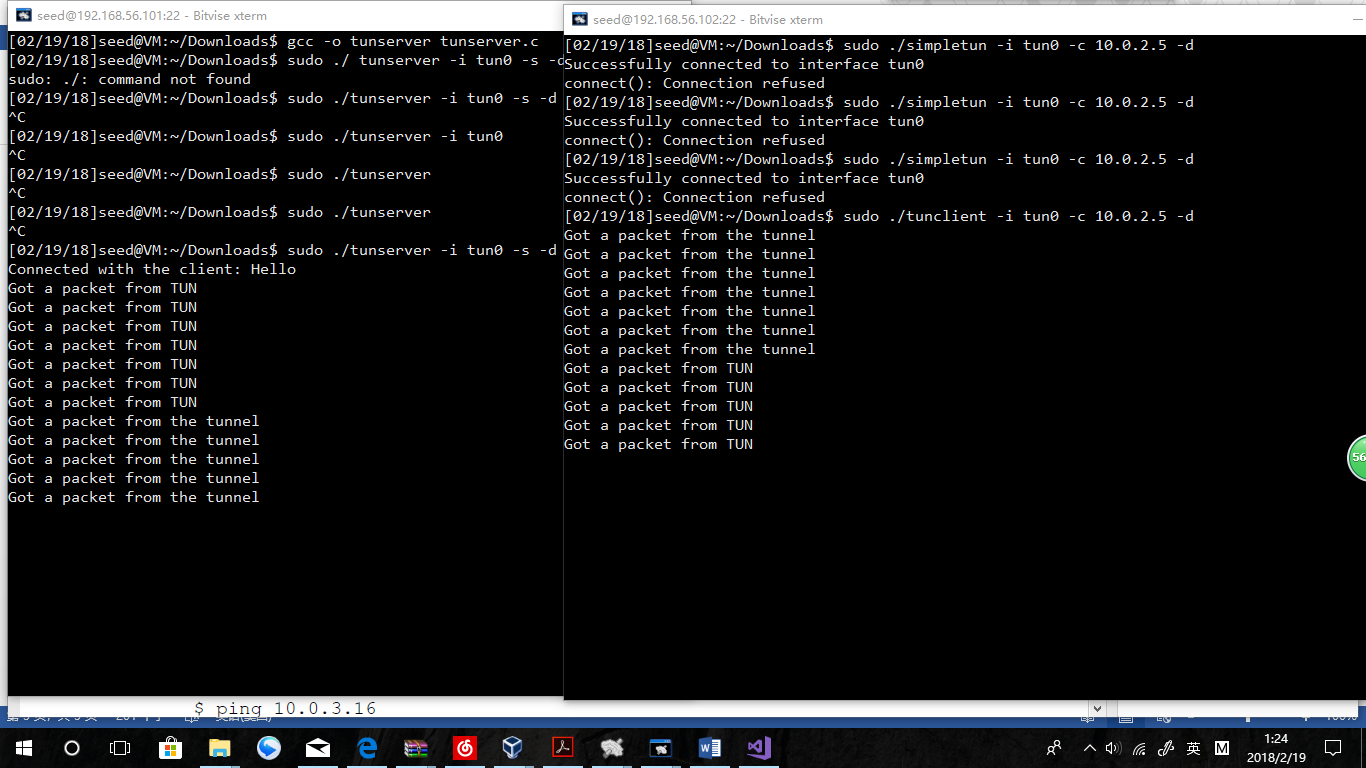
**Explanation:**

**Via tun0, the connection is established. The connection is checked only by packets exchange and Machine B still needs other operations to built itself a virtual address.**

# ip addr add 10.0.3.16/24 dev tun0

# ifconfig tun0 up





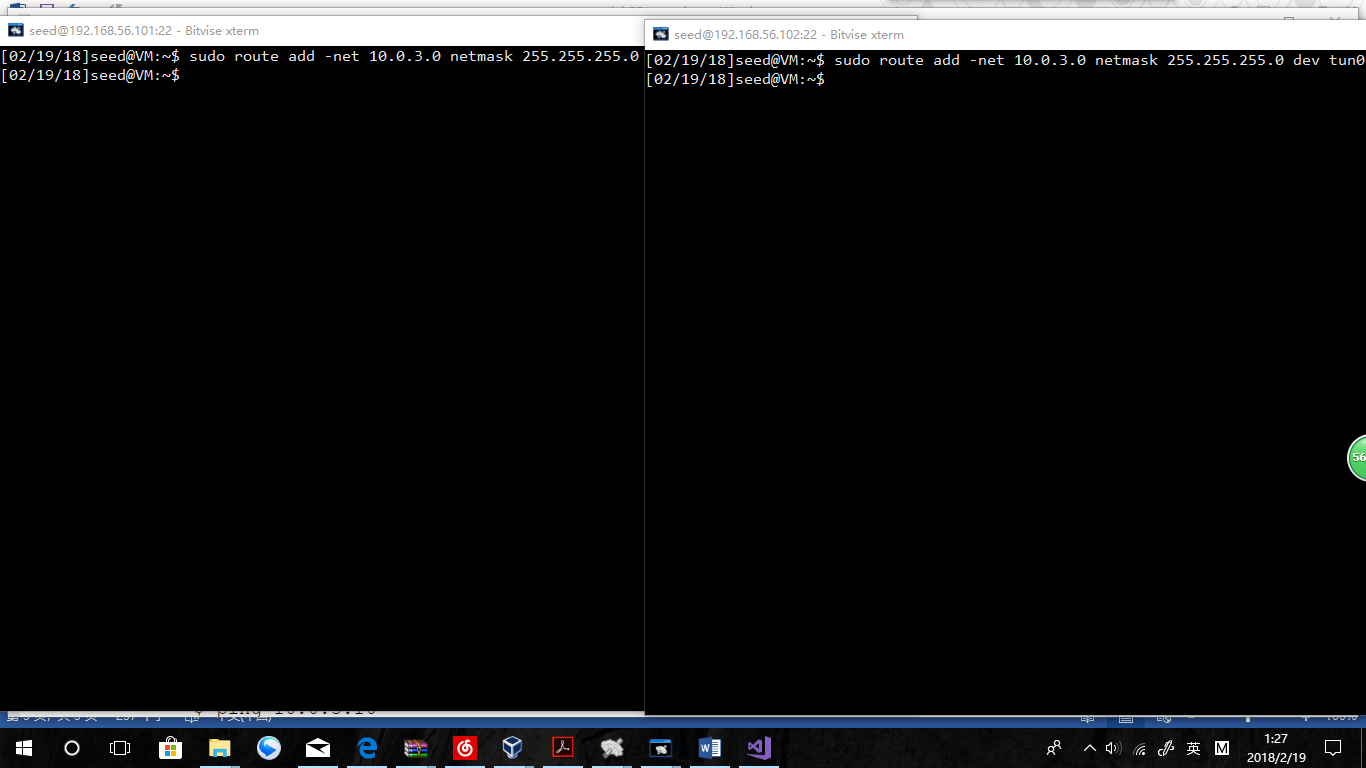
**Observation:**

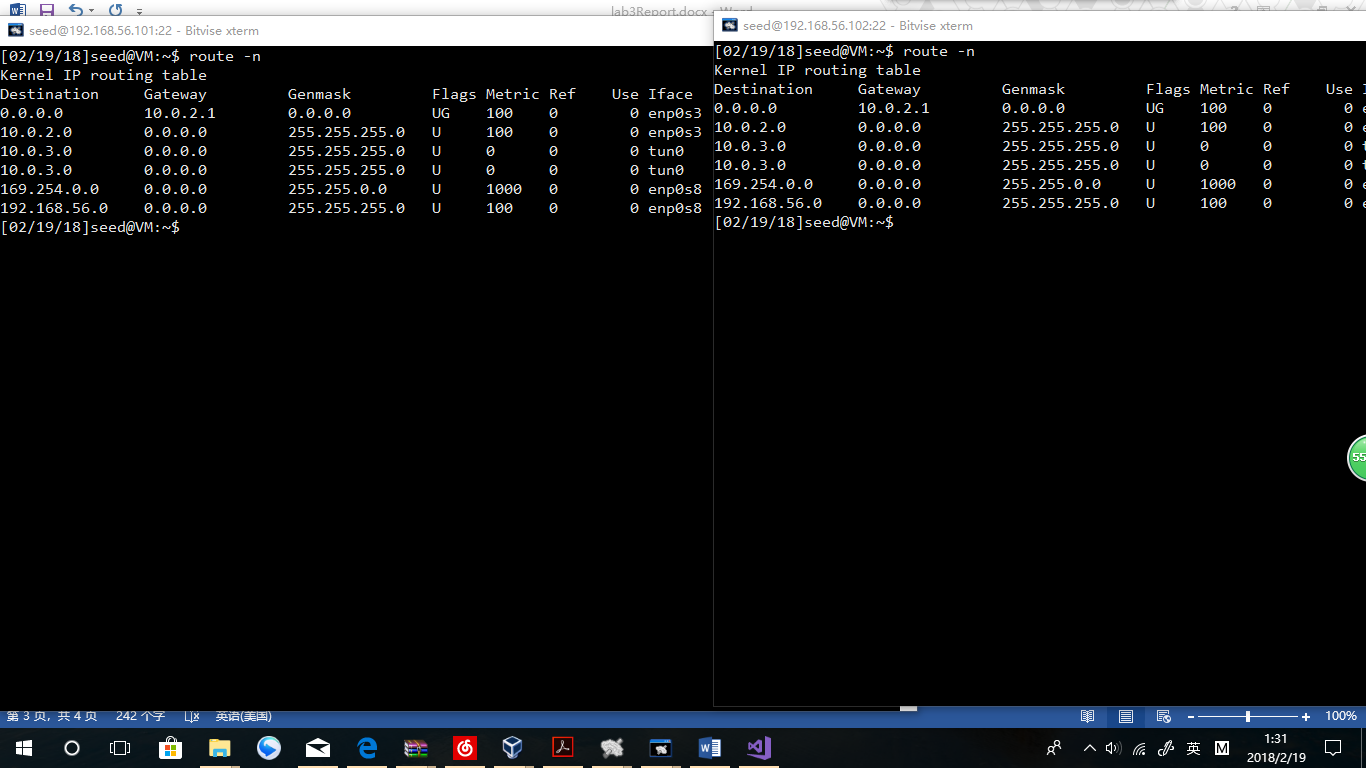
**From the ifconfig information, we find that Machine B is now having a tun0 with an address of 10.0.3.16. From now on, the connection is established with correct configuration of tun0 settings.**

**4. Routing Path:**

**Machine A: route add -net 10.0.3.0 netmask 255.255.255.0 dev tun0**

**Machine B：route add -net 10.0.3.0 netmask 255.255.255.0 dev tun0**





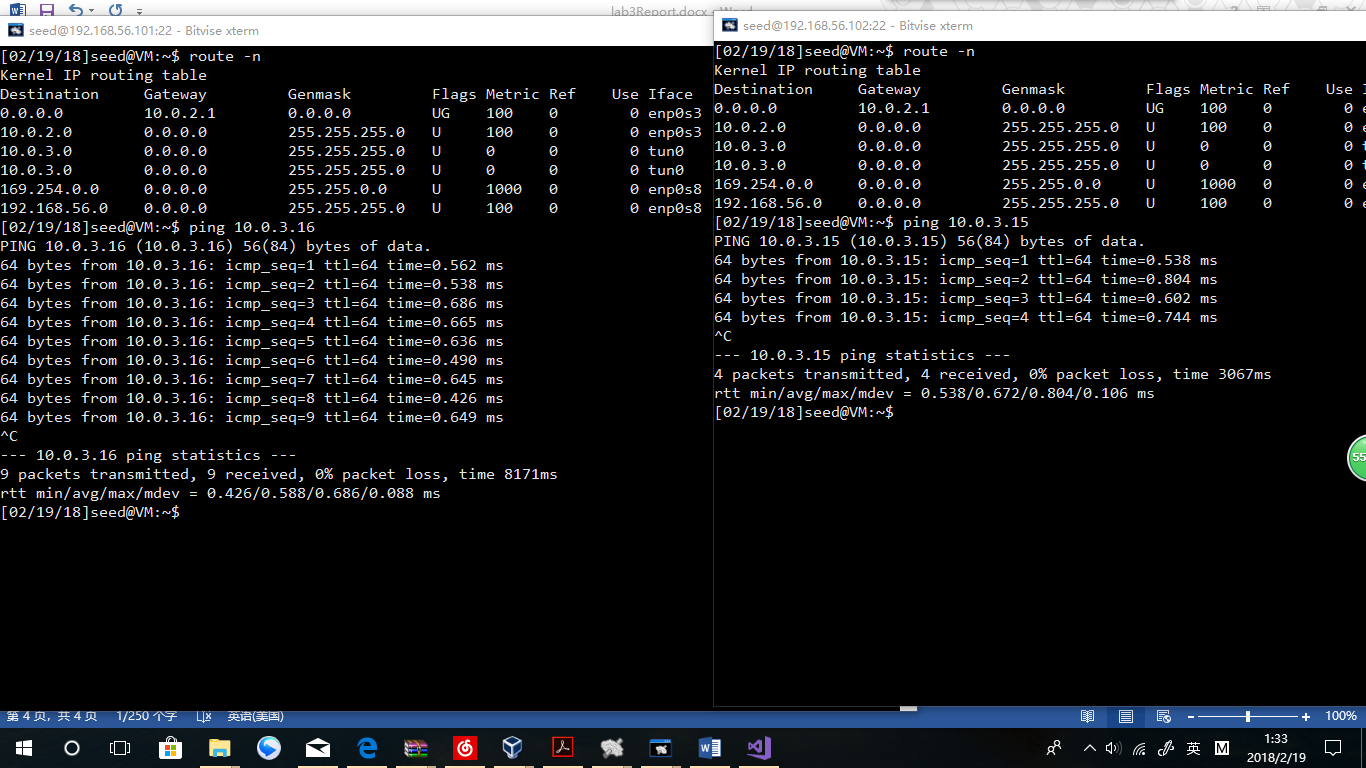
**Observation:**

**The Route of 10.0.3.0 is well settled.**

**5. Using Tunnel:**

ping 10.0.3.16

ping 10.0.3.15



**Observation:**

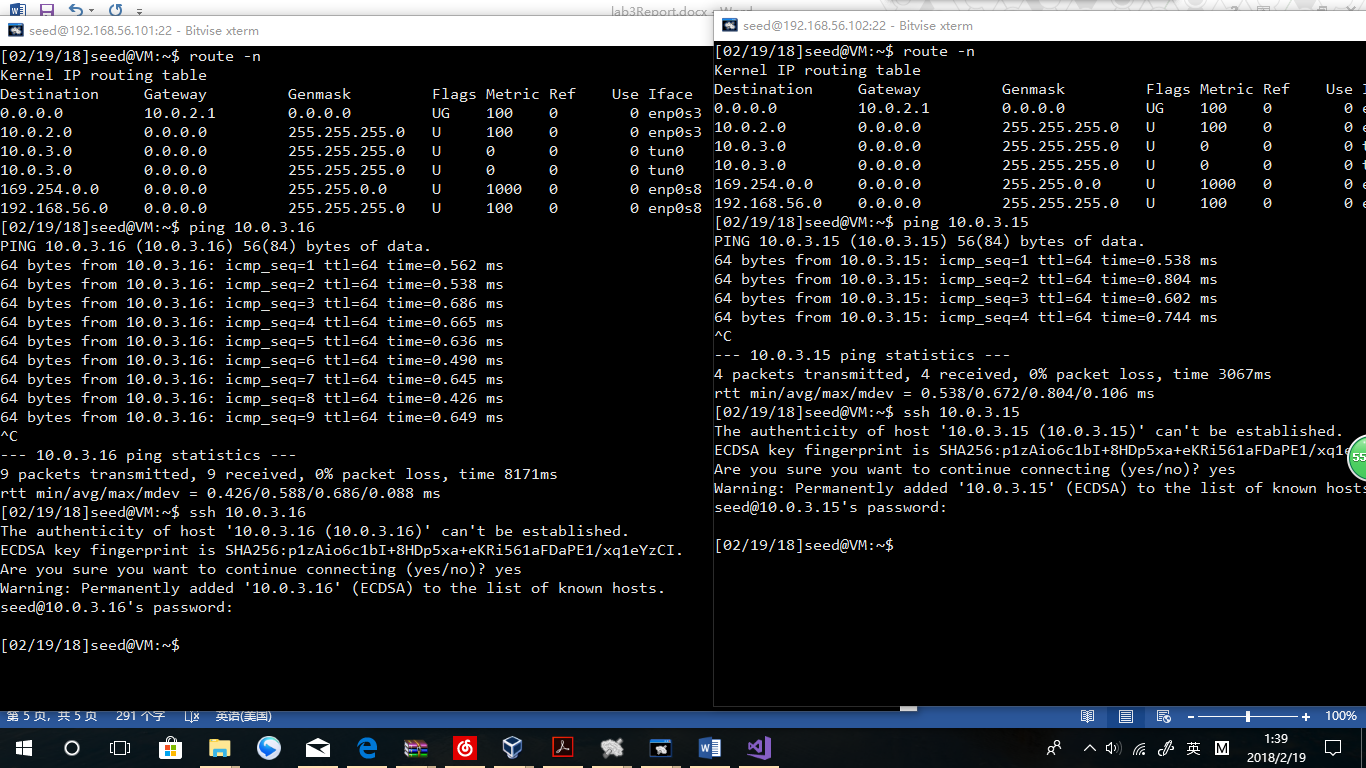
**A can ping B and B can ping A, and both ping operations are using the new address of tun0.**

**Explanation:**

**With the TUN connection and Route established, A and B can find each other in 10.0.3.0 network.**

SSH 10.0.3.16

SSH 10.0.3.15



**Observation:**

**A and B can use SSH between each other, using the new addresses of tun0.**

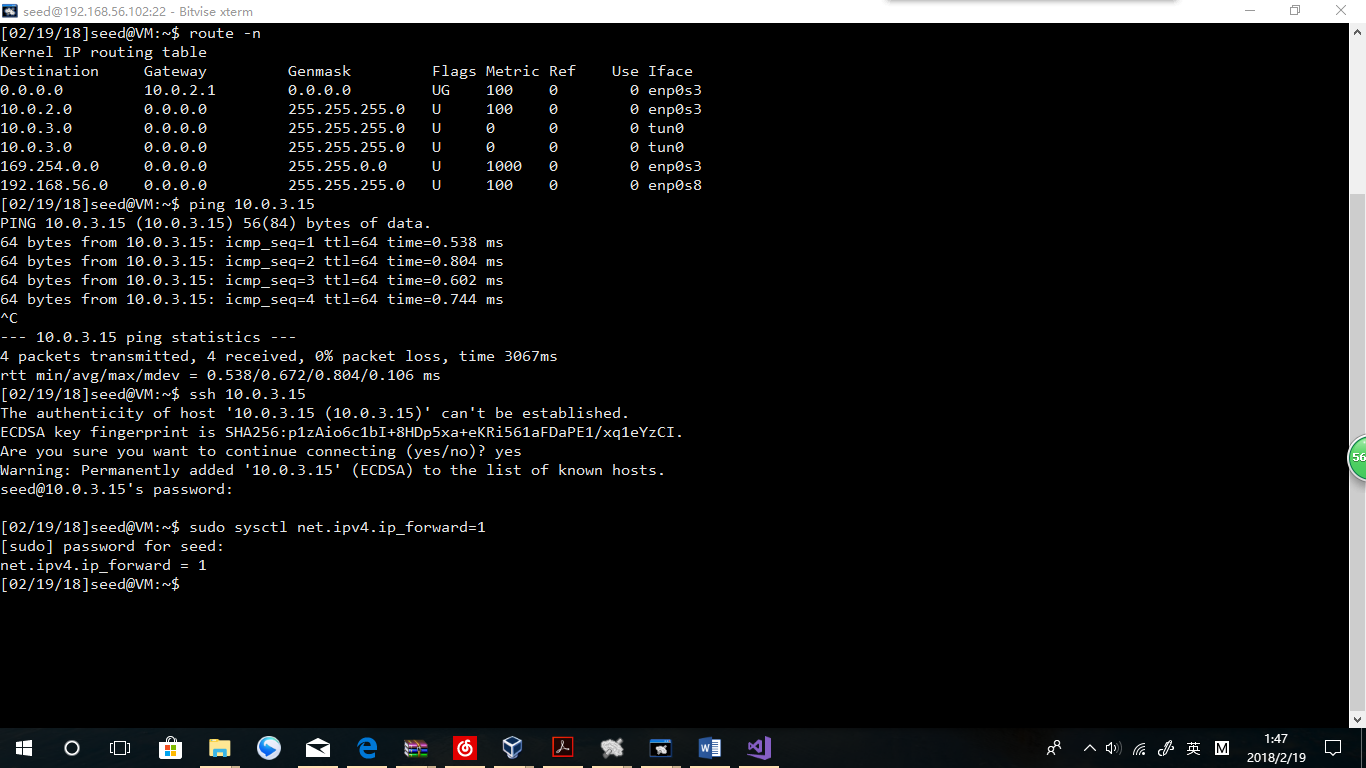
**Explanation:**

**With the TUN connection and Route established, A and B can find each other and exchange packets in 10.0.3.0 network.**

**3.2 Task 2: Set up Host-to-Gateway Tunnel**

**1. Set up IP forwarding:**

**sudo sysctl net.ipv4.ip\_forward=1**



**Observation:**

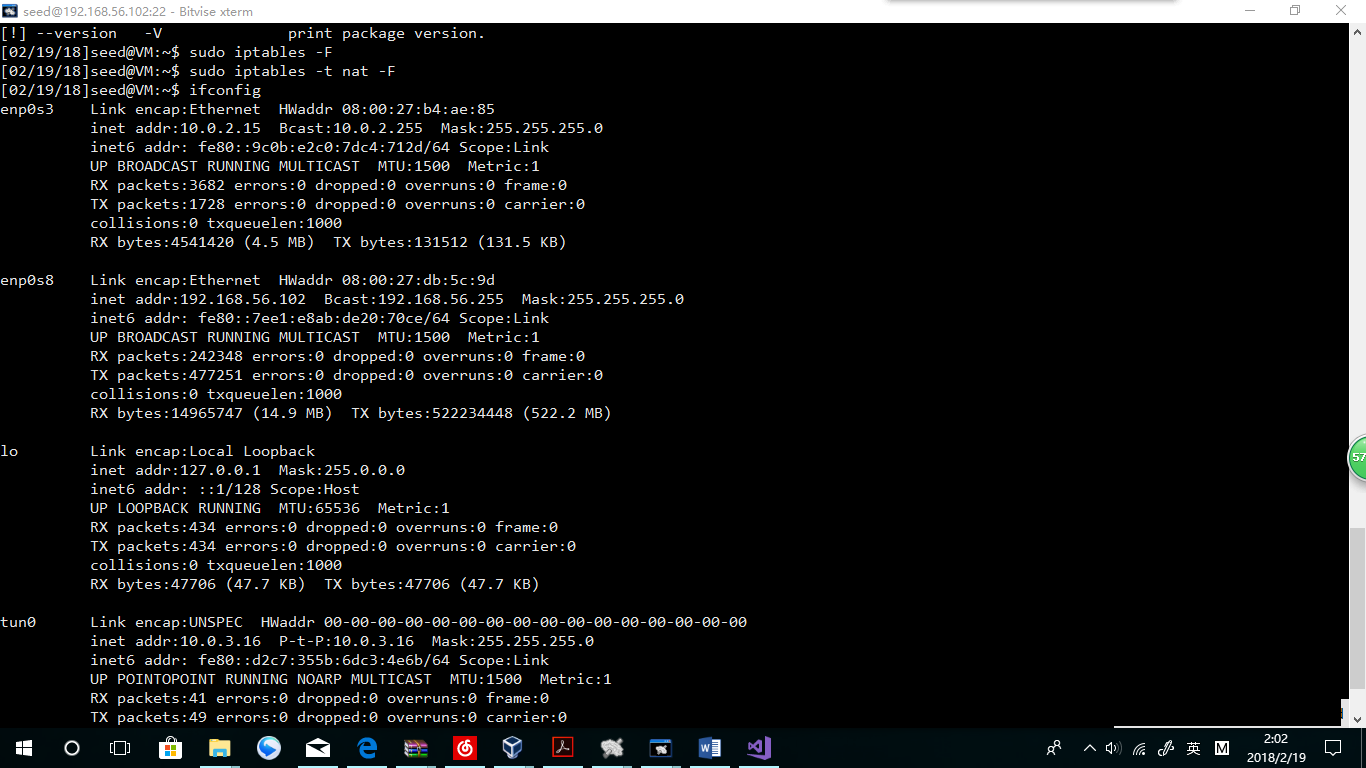
**The net.ipv4.ip\_forward has been set to 1.**

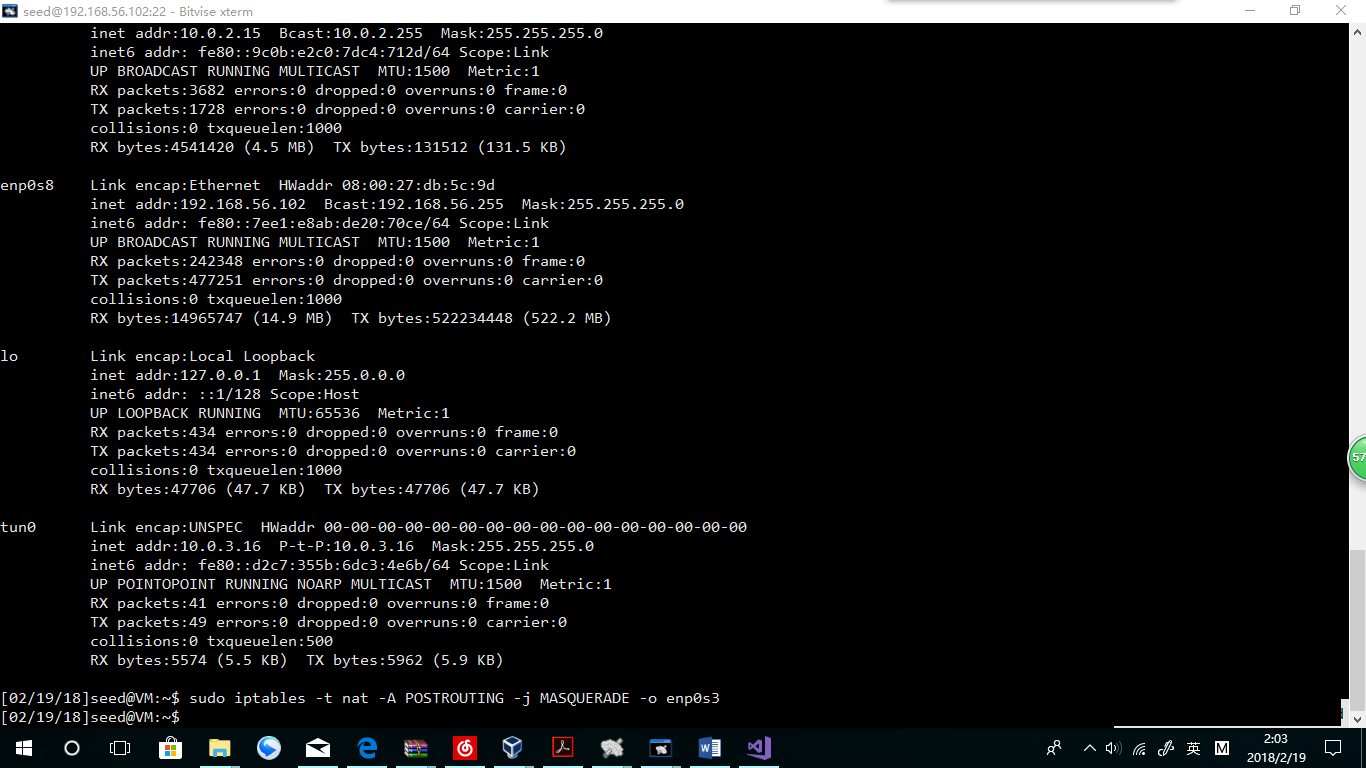
**2. Get around the limitation of NAT:**

**3. Another way to get around the limitation of NAT:**

**$ sudo iptables -F**

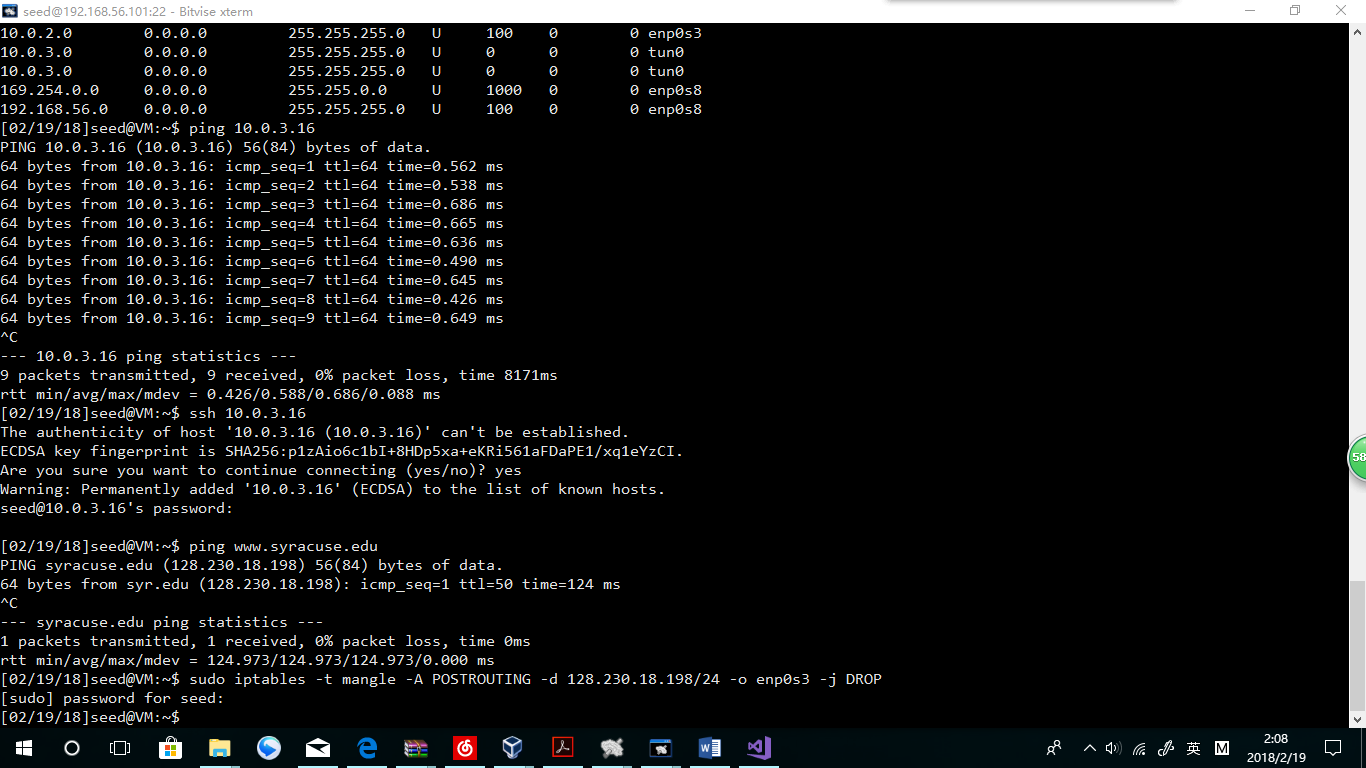
**$ sudo iptables -t nat -F**



**Sudo iptables -t nat -A POSTROUTING -j MASQUERADE -o enp0s3**

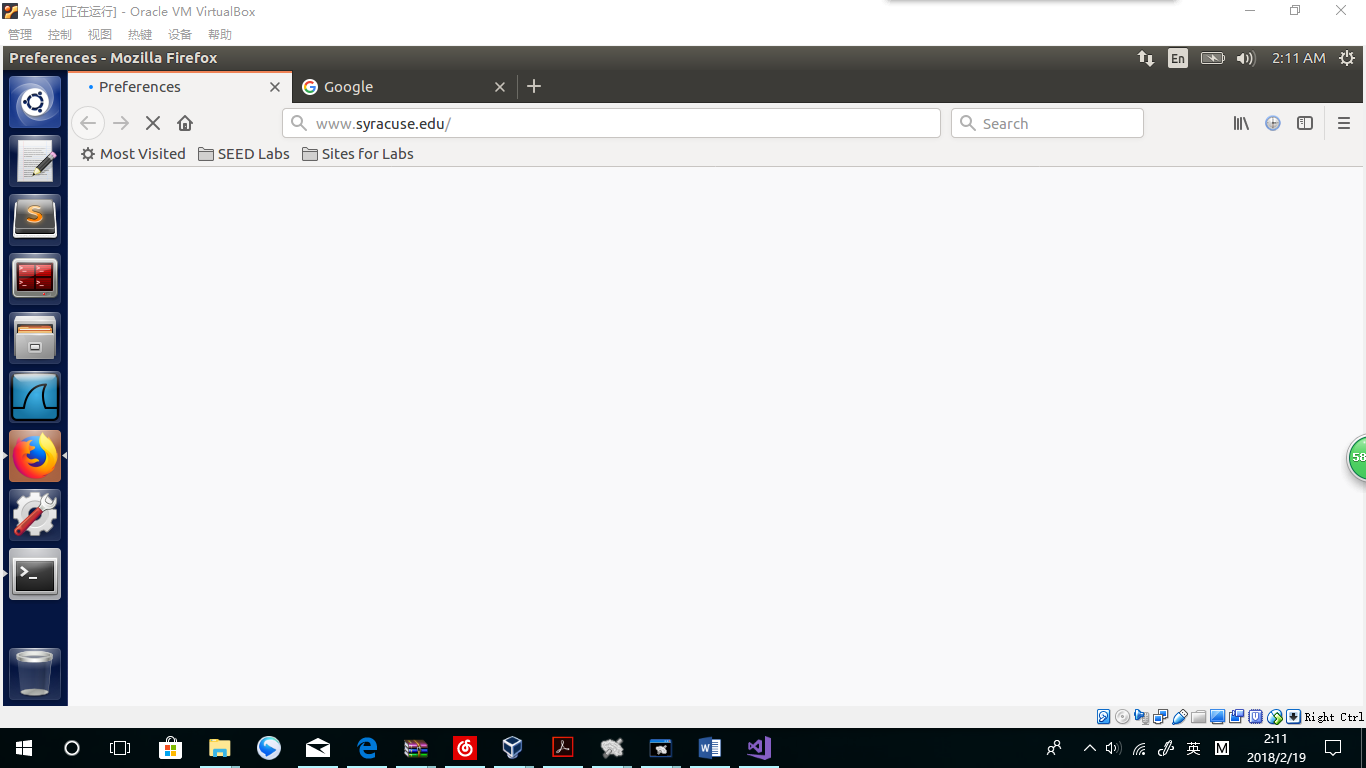
**3.3 Task 3: Set up Firewall**

**Machine A:sudo iptables -t mangle -A POSTROUTING -d 128.230.18.198/24 -o enp0s3 -j DROP**

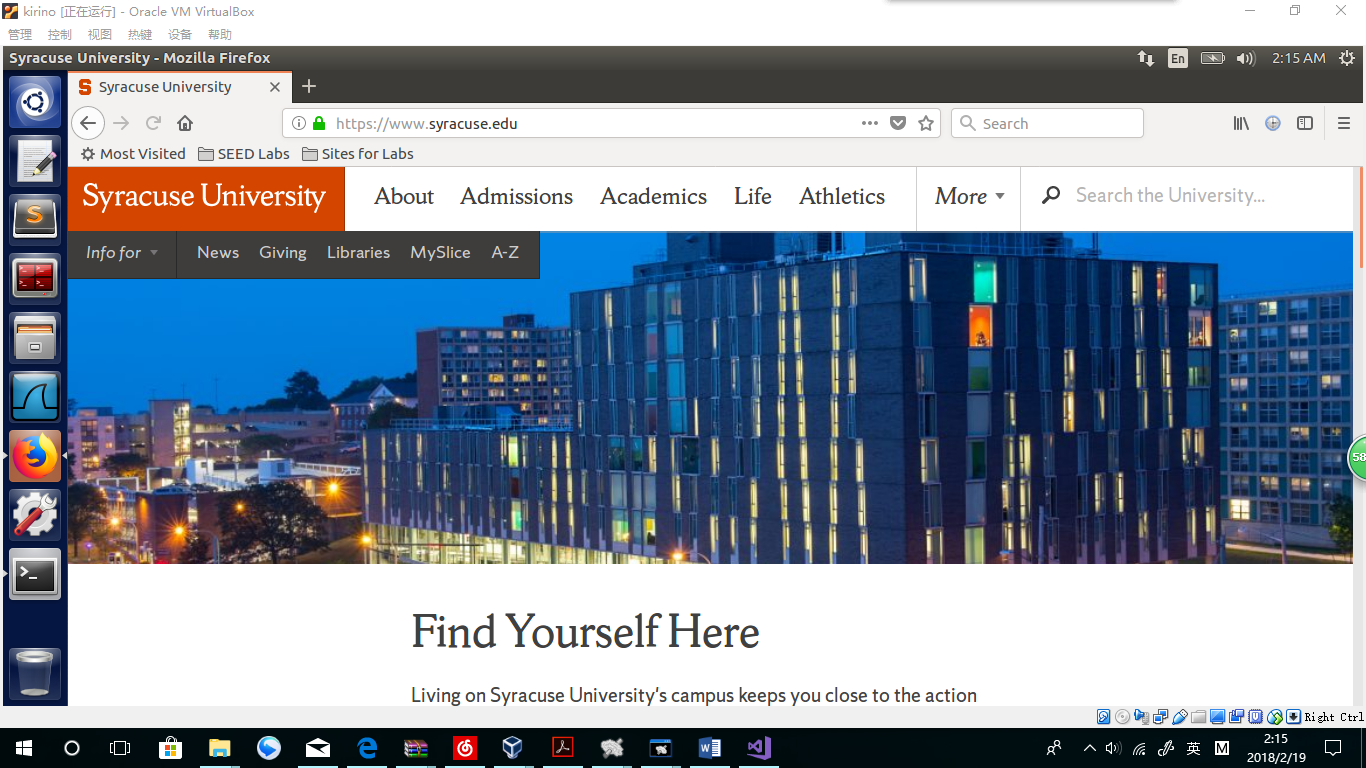


**Explanation:**

**By typing in the command,** [**www.syracuse.edu**](http://www.syracuse.edu) **(128.230.18.198) should be blocked from enp0s3, which means, without the TUN the website should not be available.**







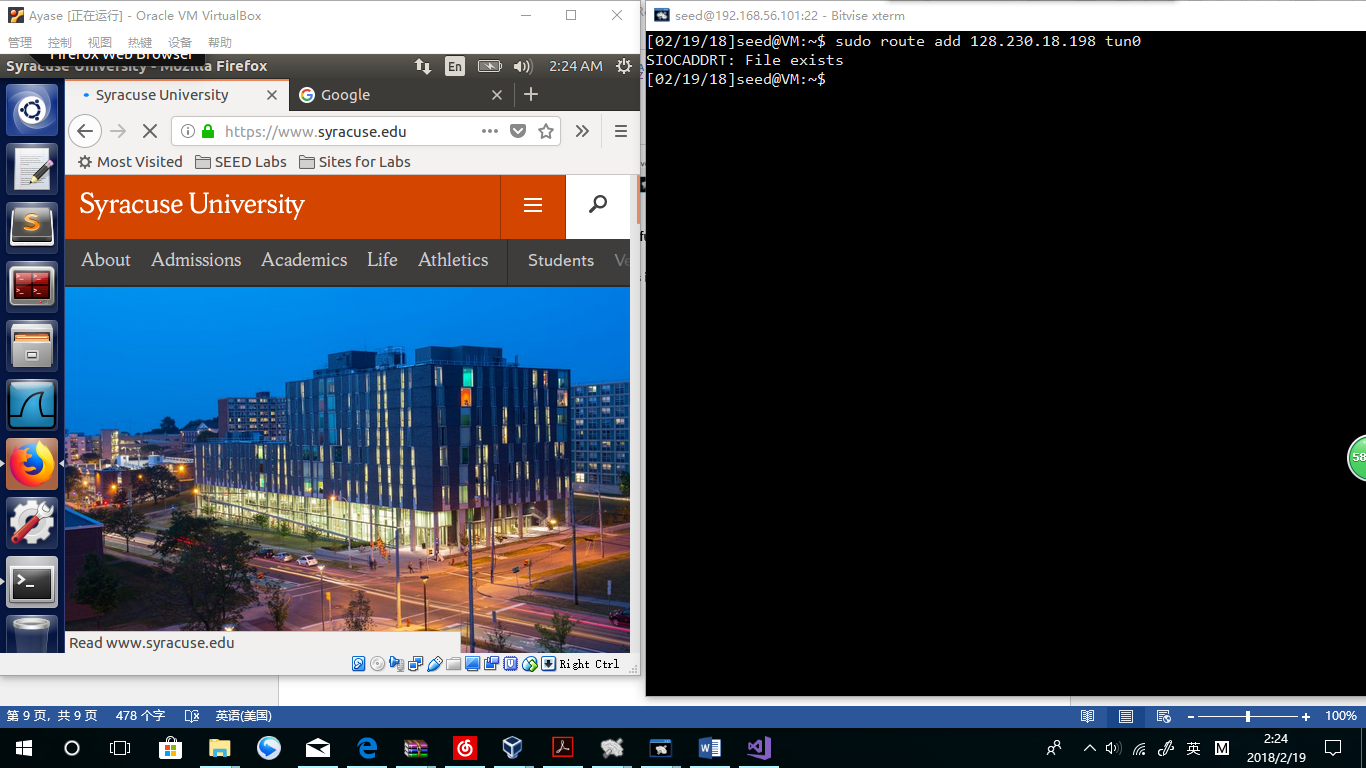
**Observation:**

**The** [**www.syracuse.edu**](http://www.syracuse.edu) **is blocked successfully. B can visit the website while A cannot.**

**Explanation:**

**This is because I have blocked the address in my NIC “enp0s3” , which is the way how Machine A can visit network.**

**sudo route add 128.230.18.198 tun0**



**Observation:**

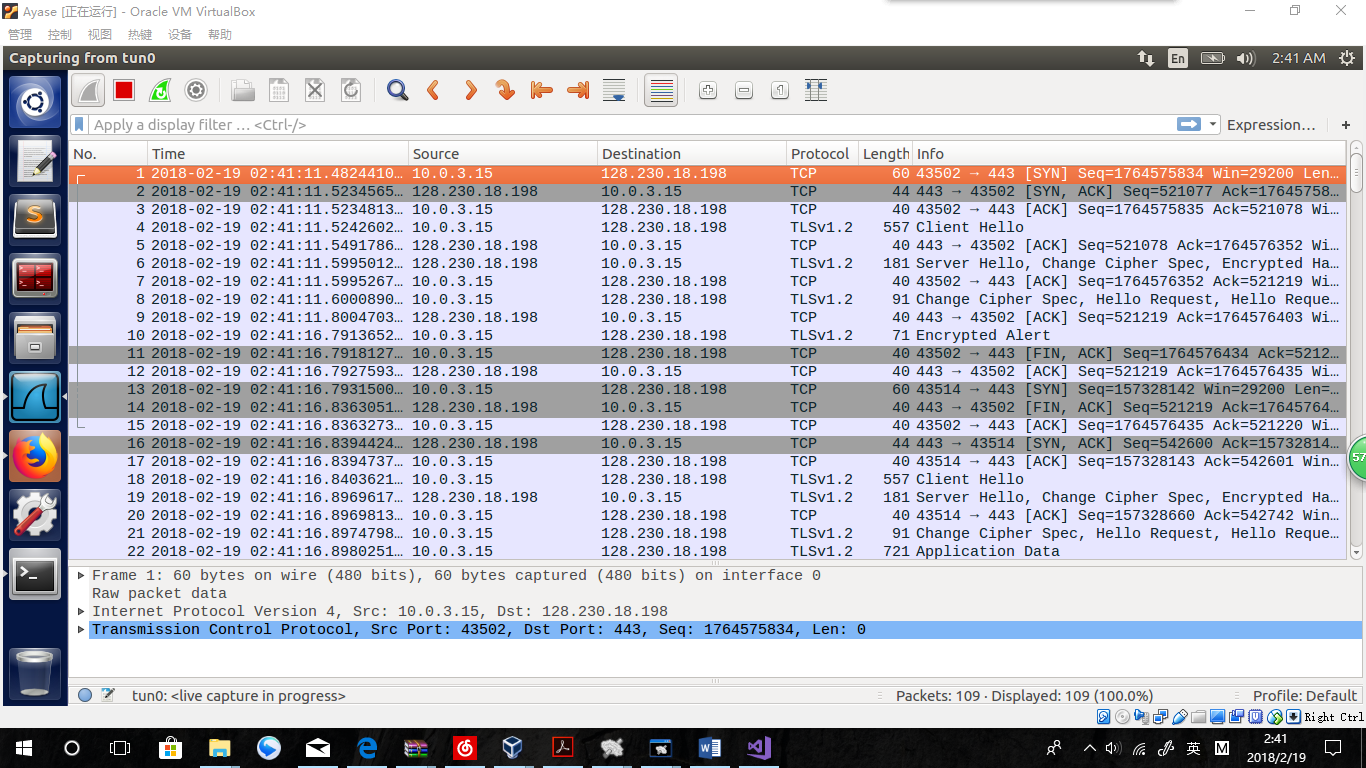
**Machine A can visit** [**www.syracuse.edu**](http://www.syracuse.edu) **again!**

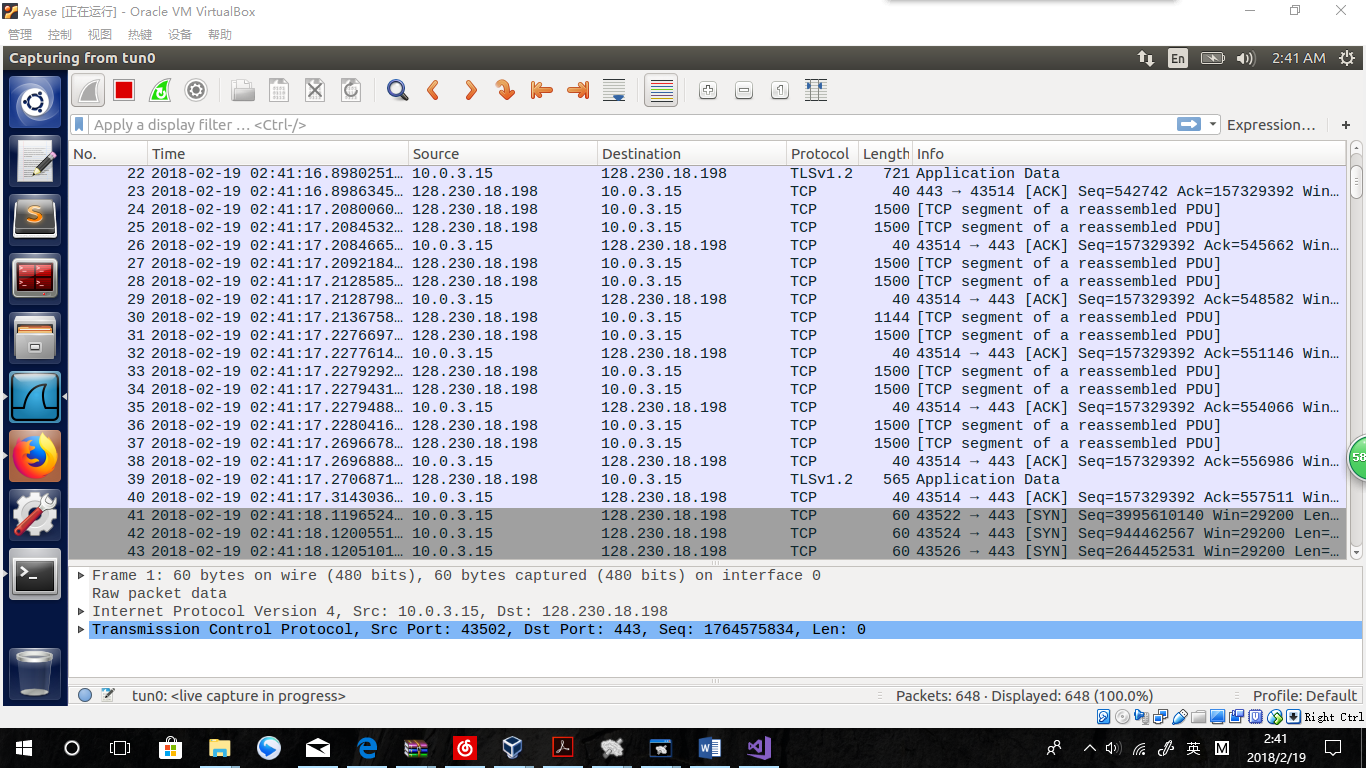
**Explanation:**

**Every time Machine A wants to visit** [**www.syracuse.edu**](http://www.syracuse.edu) **(128.230.18.198), it uses “tun0” instead of “enp0s3”. “Tun0” is the TUN between A and B, while B can visit the website. As a result, A can visit the website.**

**Wireshark:**

**WireShark in Machine A (tun0):**



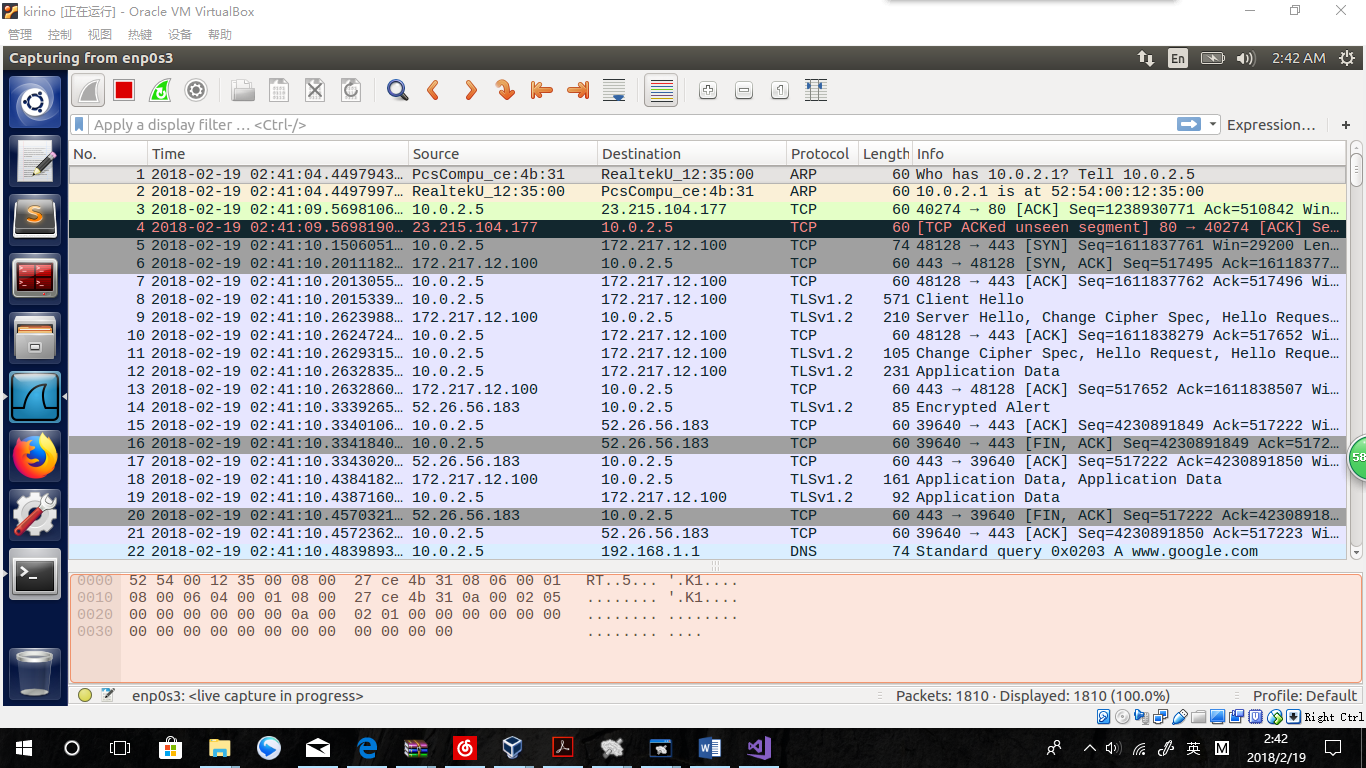


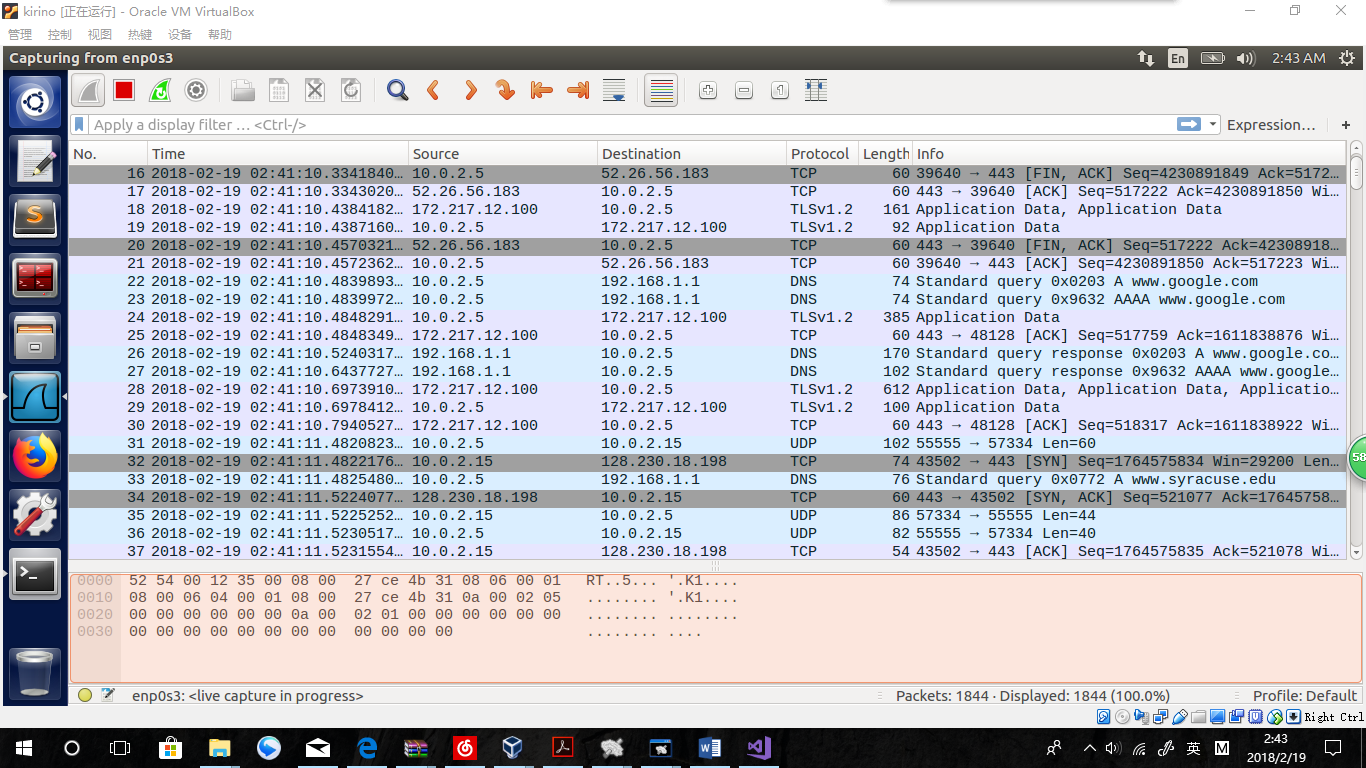
**Explanation: From information in Machine A’s tun0, we can easily find out how the website is visited. This is what machine A is doing:**

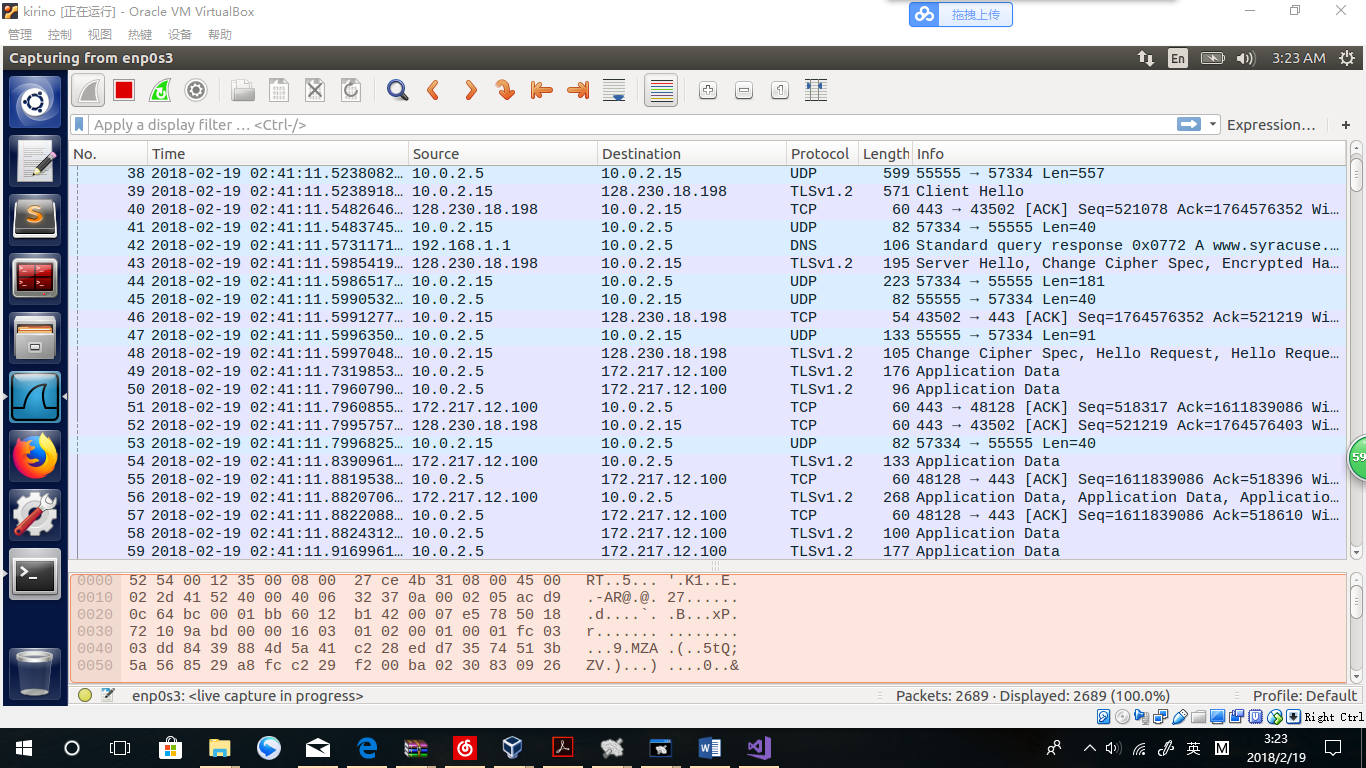
**When machine A needs to visit 128.230.18.198, the address has been routed to the tun0 interface using the virtual address “10.0.3.15”. At this time, the tun0 is more like another NIC that A is using. Via this virtual address, machine A sends and receives packets between the website, including TCP ACK messages and webpage contents.**

**As we can see, there are only packets with addresses of “10.0.3.15” and “128.230.18.198” shown in the tun0. The TUN has been used as a bridge for A to visit** [**www.syracuse.edu**](http://www.syracuse.edu) **.**

**Machine B(enp0s3):**







**Explanation:**

**In machine B, we can find more details how this VPN works.**

**Because of the test codes in tunclient.c, many “Hello” message are captured (like 8 and 9) . 172.217.12.100 stands for the local server runs on Machine B for the TUN connection. Which means, in the tun0, it equals to “10.0.3.16”. The IP address changes mainly because I changed the ip-tables to use the default NIC as the post routing position for A.**

**From No.31, the TUN started to work for A. An UDP packet was sent to B, telling the requests of visiting the website. Then, Machine B visited the Syracuse homepage in the line 32. While receiving back massages from Syracuse homepage, B packed the massages as UDP packets and sent to A. While A received the packet, A has to deal with the packet and sent another UDP packet to B as an ACK answer. After B get it, B sent the contents of the UDP packet, which is a TCP ACK packet, to the webpage.**

**Like the process above, the process of all communications between A and the website can be explained as follow:**

1. **Machine A built a packet which was wanted by the target website.**
2. **Machine A regard the whole packet as data and pack it in a UDP packet.**
3. **Machine A sends the UDP packets to Machine B via the TUN connection.**
4. **Machine B receives the UDP packets and unpacked it to an original version.**
5. **Machine B sends the original packet to the target website.**
6. **Machine B receives the answer and return it to A via the same steps of 2-5.**

\*



**Explanation:**

**That is why we captured many “Hello” messages in Machine B’s Wireshark.**