Johns Hopkins University
EN.601.444/644 Network Security
Fall 2017
Seth James Nielson
Lab #1d
Revision 1.0 (9/12/2017:1650)

ASSIGNED: 9/12/2017
DUE: Midnight 9/13/2017

Introduction

This should be a quick and easy lab to convert your protocols over from TCP to Playground.

As a quick reminder, here are the mini-labs we are working on in phase 1:

- ~~1[a] – Design a simple protocol with at least three packet types~~
- ~~1[b] – Install the Playground framework and implement the packets from 1[a]~~
- ~~1[c] – Create a TCP/IP client server implementation of your protocol~~
- 1[d] (this lab) – Convert your protocol from TCP/IP to Playground Network
- 1[e] – Install a simple "pass-through" network layer for the Playground Network.

And the due dates are

- ~~1[a] is due 9/4~~
- ~~1[b] is due 9/6~~
- ~~1[c] is due 9/13~~
- 1[d] is due 9/13
- 1[e] is due 9/18

This will be a programming assignment in python using the Playground framework.

An Overview of Playground

If you've successfully completed lab 3, you've been able to connect a client and server to each other using asyncio and TCP. Now, you're going to connect the exact same protocols over the playground network instead!

The Playground Overlay Network

Playground is an "overlay" network. That means that it runs on top of another network. From a certain point of view, most networks are overlay networks. For example, you could argue that the IP network is overlaid on top of Ethernet, etc. But typically we don't use that term for IP because it adds functionality to the network beneath it. That is, it provides inter-network connectivity, which Ethernet does not.

You use overlay networks all the time, though. For example, the entire email system is an example of an overlay network. You have an address (e.g., sethjn@cs.jhu.edu), and you have mechanisms for sending messages over this network. The fact that it runs over TCP is, for the most part, invisible to you.

Playground needs to run as an overlay network because we want the network to behave a little differently than TCP/IP. At the same time, I want the concepts in Playground to be as familiar as possible, so it uses concepts and names from a typical local area network.

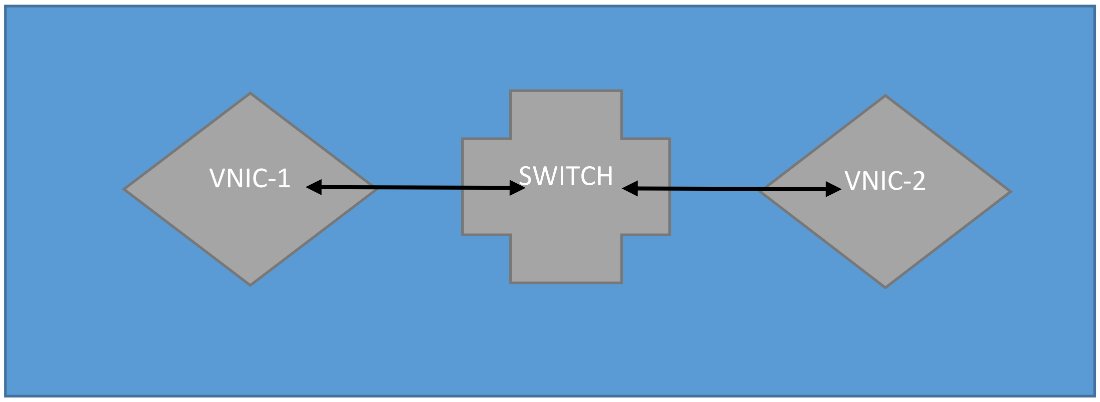
First of all, to access an Ethernet or WiFi network, you need to have a Network Interface Card (NIC). Up until about 10 or 15 years ago, it was still common for people to install an Ethernet NIC into their computers as an "add on" card. But it started becoming so common that now almost all motherboards/devices have Ethernet built-in.

Similarly, to access the Playground network, you'll need a Virtual Network Interface Card (VNIC). It's a virtual NIC because it isn't a piece of hardware. It's a piece of software that's going to provide you connectivity to the network.

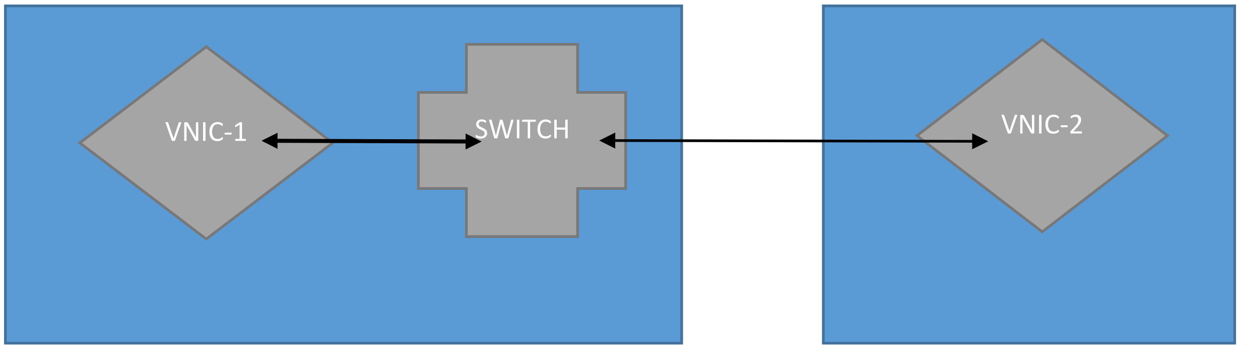
In a real local area network, once you have a NIC, it won't do you much good unless you can connect it to some kind of "switch." With Ethernet, you connect a physical wire from your NIC to a switch. With Wifi, you connect to the switch (access point) via Radio waves.

The playground network also uses switches and, conceptually, plugs in like an Ethernet cable. Again, there's no physical cable. The underlying connection isn't important; it could be TCP/IP, a named pipe, or even shared memory. But from your perspective, it is just a "wire."

Here's a picture to help visualize what's going on here. For the first picture, there are two Playground VNIC's and one Switch on the same physical computer:



The black lines represent the “wires” between the VNIC’s and the switch. But remember, these aren’t real wires. For most of your labs, they will be TCP/IP connections. But that doesn’t matter to playground. Nor does it matter to playground that these virtual pieces of hardware are on the same computer. Consider the following example:



Here, the two blue boxes represent two physical machines. But from the perspective of playground, the above configuration is the same as the single-machine example above.

In terms of functionality, the purpose of a switch is to connect one VNIC to another. This is *not* routing. Routing is a process by which you get a packet to its destination without, possibly, knowing where that destination is. Switches know that all their destinations are directly connected to them and simply serves as a “switchboard” between the two endpoints.

When a real ethernet-enabled device connects to an Ethernet switch, it must have a self-assigned Ethernet address and it is up to the NIC, not the switch, to ensure that the address is unique.

Playground addresses are a little different than Ethernet addresses because they are “smooshed” together with the equivalent of an IP address. To be more clear, in the real Internet, your computer has both an Ethernet address AND an IP address. But in Playground, your VNIC has just one address that will be used for both local and wide-area routing (if we get that far). A playground address, like an IPv4 address, has four parts (a.b.c.d) but each part can be an arbitrarily large number (e.g., 50000.200.66666.91919191)

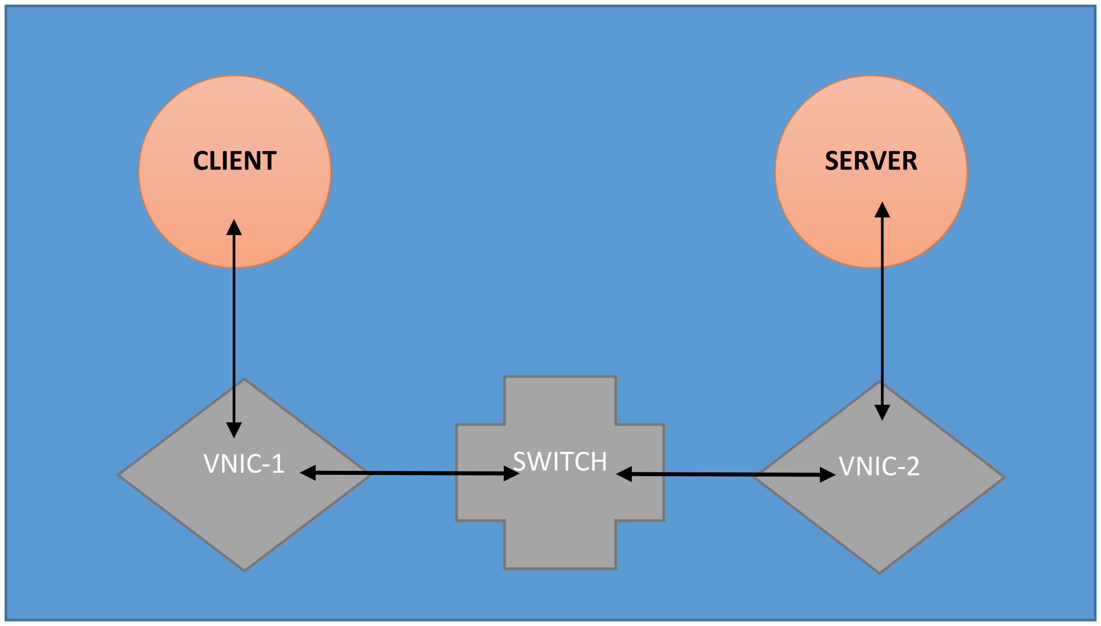
But like a real NIC, your VNIC needs to be assigned an address. And like a real NIC, the switch will do nothing to ensure that you have a unique address. If two VNIC’s use the same address, the switch will send messages destined to that address *to both of the VNIC’s*.

HINT: This will be an interesting security problem. Can you see how easy eavesdropping is?

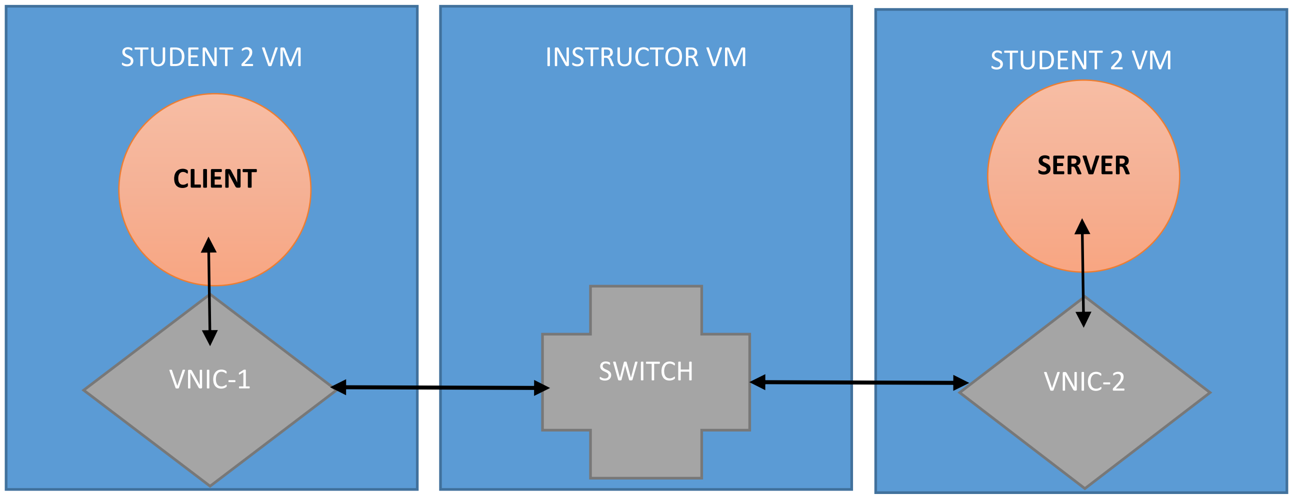
Once you have a VNIC, and once you have it connected to a switch, you can send messages to any other playground device connected to the same switch. However, unlike Ethernet, which has no ports or data fragmentation, Playground provides a basic UDP-like level of connectivity. You can set up servers and create connections, but there is no reliable delivery or messages and no session control.

Still, for simple protocols, such as your lab 1 protocols, this is more than enough.

Visually, your program will communicate over playground like this:



In the above visualization, there is only one physical computer (blue box). For lab 1d, that is how you will most likely test. You will run a switch, VNIC (maybe just one instead of 2), and client/server all on the same machine. Later in class, when everyone is communicating with each other, there will be a class switch and the network architecture will probably look something like this:



Setting Up Playground Networking

In previous years, students had to manually configure all of their virtual hardware themselves. It was a *pain*. You typically had to open three or four terminals: one for the switch, one for the VNIC, one for the client, and one for the server.

YUCK.

This year, I've made things very easy to setup and run VNIC's and Switches. All you have to use is the pnetworking utility. This utility is new, relatively untested, and completely undocumented. I'm working on that. But it should be so much easier than manual configuration that it's worth the "alpha" status.

pnetworking allows you to create, configure, and use any playground virtual hardware (currently just VNICs and Switches). The following instructions should get you setup with a single VNIC and single Switch. This is all you should need for this lab:

1. Do all the lab 1[b] stuff (set up a venv environment, etc and pip install Playground 3)
2. Use the pnetworking script to install one VNIC and one (virutal) Switch
   1. pnetworking add switch1 switch managed
   2. pnetworking add v_eth0 vnic 20174.1.1.1
   3. pnetworking config v_eth0 connect switch1
   4. pnetworking config v_eth0 route add default
3. You can check the status with: pnetworking status
4. Enable the devices:
   1. pnetworking enable switch1
   2. pnetworking enable v_eth0
5. If everything worked correctly, "pnetworking status" will show them both enabled with pid's.

Here's what's going on.

- pnetworking add switch1 switch managed

This command adds a new switch named "switch1." Switches receive connections over TCP and require a port. The "managed" option lets the playground networking automatically choose a port.

- pnetworking add v_eth0 vnic 20174.1.1.1

This command adds a new vnic named "v_eth0." It is assigned the playground address 20174.1.1.1. You can obviously use a different address if you choose.

- pnetworking config v_eth0 connect switch1

This connects the v_eth0 VNIC to the switch named switch1. This is equivalent to plugging a wire in from your ethernet card to a hub. You don't have to worry about any tcp ports, etc. Later in the semester when you need to connect to a class switch, it will be a little different, but we'll worry about that later.

- pnetworking config v_eth0 route add default

You may not know it, but you can configure your real network to route different destinations on different NIC's. If, for example, you have two ethernet cards or, more likely, one ethernet and one wireless, you can configure it such that certain destination addresses are routed on one and other addresses on the other. The pnetworking utiliy provides the same feature. This will come in handy later in the semester if you want to have a local switch that you and your teammates use for local addresses but still be able to automatically connect to the class switch. However, for now, we just need one route and so we set v_eth0 as the default (route everything).

Once all this configuration is done, you can enable both devices. You'll need to enable the switch first because the VNIC is dependent on it. However, I believe I have auto-enable working, so try enabling the VNIC and see if it automatically enables your switch as well.

Once this is done, you can quickly test if the playground network is working using the following commands:

6. Run the echo test server: python -m test.echotest server
7. Run the echo test client: python -m test.echotest 20174.1.1.1
8. The test client should allow you to enter messages and see them echo'd.

Now, for some reason, this doesn't work in all installations. If you get a message about test.echotest not being installed, download echotest directly from the Playground3 github and run it in a local directory.

Your Assignment for 1[d]

Your assignment for 1[d] is to take your protocols, which should already be working on TCP/IP and get them working on the Playground network. Once you have networking set up, all you need to do is convert any references to "loop.create_server" and "loop.create_connection" with the playground equivalents.

For create_server, use the following command instead:

```
playground.getConnector().create_playground_server(factory, port)
```

For create_connection, use the following command instead:

```
playground.getConnector().create_playground_connection (factory, playgroundAddress, port)
```

Please note that it is the playground address you're connecting to, not the IP address. If you used 20174.1.1.1 for your VNIC, that is the address you would put in here.

Assignment Description

Your assignment for lab 1[d] is to connect the protocol you implemented in your lab 1[c] using the playground network. For this lab, there is no unit test that needs to be graded. Simply submit the code and we will review it manually.

Grading

Your assignment will be graded out of 10 points as pass fail (10/0).

Collaboration Policy

For this assignment, you ***MAY NOT*** discuss with anyone other than the professor or the TA, and my not use the Internet (e.g., Google) for any ideas. I am testing your ability to program and I do not want you using other students.

Due Date and Late Policy

The lab is due by midnight on Monday 9/13/2017. We will expect to pull the data out of your repository at that time. Your files should be in the following directory:

/netsec_fall2017/lab_1d/