GENI Developer Docs

Release 1.0.0

Kenny Katzgrau

October 13, 2015

Contents

These documents are intended to help software developers become familiar with GENI, a project sponsored by the National Science Foundation, to help experiment with *future networks*. The "what and why" of GENI is discussed in the next section, but overall, GENI is a platform for:

- Exploring gigabit networks
- Implementing new routing rules in those networks
- Writing applications which leverage these networks

A major benefit to GENI is that experiments can run "at scale", meaning they just aren't simulated in a lab, but can be run on machines that you provision across the world.

Read on to find out why GENI exists.

CHAPTER 1

Introduction

Let's talk about what you can do with GENI, how GENI is architected, and how you can get started.

Contents

1.1 What Is GENI?

GENI is both a project sponsored by the NSF and commonly refers to a number of applications and tools used to build and experiment with high-speed programmable networks. These tools are being developed by various institutions, and the efforts surrounding GENI are still very much **in progress**.

1.1.1 A Short Background (Or, What Are the Limitations of the Internet Today?)

The landscape of the internet has changed dramatically since its beginnings as ARPANET[1] in 1969. The devices using it have grown to include everything from PCs, laptops, and mobile phones to TVs, cars, and refrigerators. In addition to the range of devices, growing advanced in video and voice technology require greater bandwidth.

Although some of these applications were foreseeable, this is not what the current architecture of the internet was intended for.

1.1.2 A Global Environment for Network Innovations (GENI)

Experimentation and research into highspeed networks, new protocols, and distributed applications was previously limited to isolated labs. The GENI project was started to facilitate wide-scale

[1] http://en.wikipedia.org/wiki/ARPANET

experimentation — creating a *virtual laboratory* where engineers, researchers, and developers have access to:

- **Programmable Networks**: Routers can be programmed to handle packets in new ways[2]; opening the door to experimenting with alternatives to the core protocols of the internet.
- **Gigabit Networks**: High speed networks that are 250 times faster than what is available today
- **Virtual Machines**: Provisionable environments for running applications that leverage these programmable, high-speed networks.

GENI refers to both the overall project and the suite of tools and services that have been developed around the 3 items above. This documentation exists to show you how, as a developer, you can get set up and write your own next-generation applications.

1.1.3 It's the Side of Hacking You Haven't Been Exposed To

How long have you taken TCP/IP for granted? How long have you build your web applications upon the decades-old work of others? With GENI, you are exposed to a new frontier of hacking.

Read more about what you can do with GENI.

1.2 What Can I do With GENI?

The GENI platform is very open-ended, but it's intended to be an environment for:

- Exploring gigabit networks
- Implementing new routing rules in those networks
- Writing applications which leverage these networks

1.2.1 Examples

Here are a few project ideas that are possible with GENI tools and resources.

Write an application that can utilize gigabit connections

Gigabit internet connections are the future. So what kind of network-reliant applications could you build that aren't possible with the typical consumer internet speeds of today?

Test how different protocols are affected by network bottlenecks

If you wanted to compare, for instance, how TCP and UDP perform under different levels of network congestion, you could easily simulate that with GENI

[2] http://www.openflow.org/

Build distributed applications

You have considerable computing and network resources available via GENI, so you can develop and test your next-generation distributed applications over gigabit links.

Design an alternative to the IP protocol

Is IP holding you back? Define the rules that a router follows for forwarding packets, and experiment with what *could be* the internet's future backbone protocol.

1.2.2 Going Forward

Are you ready to dig into GENI? Check out the prerequisites to make sure you can hit the ground running.

1.3 What You Need to Know Before Starting

Before you dive into using GENI, there are a few concepts that you should be familiar with beforehand. A good grip on these concepts will ensure that you can follow along with various GENI tutorials, examples, and videos.

1.3.1 GENI Credentials

You'll need GENI credentials before you can start experimenting. Click here to find out how to get set up with those.

1.3.2 Virtual Machines

We've mentioned that GENI is a virtual laboratory for running network experiments. As a GENI user, you are allowed to *request* and allocate resources around the world, such as virtual and physical machines.

It's a good idea that you understand the basic difference between the different machine types. Sometimes it's helpful to work with specific machine types in your experiments.

Additional Resources

1. Wikipedia - Virtual Machine[3]
2. Discussions on ServerFault[4]

[3] http://en.wikipedia.org/wiki/Virtual_machine
[4] http://serverfault.com/questions/135431/is-virtual-machine-slower-than-the-underlying-physical-machine

1.3.3 Linux and Relevant Utilities

It's important that you are comfortable with using the Linux/UNIX command line. You will be using it heavily in the process of running your experiments, and it's the only method you'll have for controlling your resources.

In particular, make sure you're well versed with:

1. A unix shell[5], such as bash[6]
2. The ssh utility (used for connecting to your machines)[7]
3. The ping utility[8]
4. Setting up and using SSH keys (your only login method)[9]
5. A scripting language, like ruby[10] or python[11]

1.3.4 Networking

What good would access to computing resources around the world be without a way to connect them? A basic understanding of computer networking concepts is important to have.

More specifically:

1. The layers of the OSI model[12]
2. Internet Protocol, or IP[13], the backbone protocol of the internet
3. TCP[14] and UDP[15]
4. Connections over the mentioned protocols using socket APIs[16]
5. How data packets are routed[17] around the internet

When you feel comfortable with the topics above, you can move on to read about the terms and concepts used when discussing GENI.

[5] http://en.wikipedia.org/wiki/Unix_shell
[6] http://en.wikipedia.org/wiki/Bash_(Unix_shell)
[7] http://support.suso.com/supki/SSH_Tutorial_for_Linux
[8] http://en.wikipedia.org/wiki/Ping_(networking_utility)
[9] http://www.ece.uci.edu/ chou/ssh-key.html
[10] http://www.ruby-lang.org/en/
[11] http://www.python.org/
[12] http://en.wikipedia.org/wiki/OSI_model
[13] http://en.wikipedia.org/wiki/Internet_Protocol
[14] http://en.wikipedia.org/wiki/Transmission_Control_Protocol
[15] http://en.wikipedia.org/wiki/User_Datagram_Protocol
[16] http://en.wikipedia.org/wiki/Network_socket
[17] http://en.wikipedia.org/wiki/Internet_layer

1.4 How Is GENI Set Up?

For the purposes of this section, the lower-level details of how GENI is set up are spared. The "Additional Resources" section, although, contains a link to a more extensive document authored by the GENI Project Office.

1.4.1 GENI Setup: A Developer's View

The basic work-flow of a developer looking to start building on GENI offers a nice view on the structure of the GENI project. So let's consider Alice, a fictional GENI wannabe-experimenter.

From the top: Clearinghouse

In order to start interacting with GENI, Alice need to become a GENI user. Alice can obtain credentials from a GENI *Clearinghouse*. A GENI Clearing house can issue credentials that are *federated*, meaning that they are recognized by all GENI *Management Authorities*.

What's in a Management Authority?

A Management Authority, or Manager, is an organization or institution that makes resources available to experimenters like Alice. These resources are made available via APIs. As a developer, Alice does not have to deal directly with these APIs because of GENI tools that are available to her.

What Tools?

Once Alice has GENI credentials, she can use tools like OMNI or Flack. Because Alice wants to dive in quickly, she opts for Flack, the visual web-based option.

Alice wants to start an experiment, so she needs to create an environment for her experiment, called a *slice*. The slice Alice creates via Flack will be recognized throughout by all of the GENI Management Authorities, which supply the resources and *slivers* that she needs.

What's a Sliver?

Once Alice has her slice set up, she'll want to request virtual machines from different Managers. The virtual machines, and *any* other resources that Alice might request are collectively referred to as slivers. They are called 'slivers' because they are often virtualized and shared components of a physical machine.

Whereas a slice is a global container for Alice's experiments, the collection of resources she is using on any given site is referred to as a *sliver*. So if she has 2 nodes running in Utah, that's one *sliver*, and her nodes running in Washington DC are in another *sliver*.

And that's it

Now that you have a basic idea of how GENI is set up from a (very) high level, it's best to move on the the more extensive GENI Terms and Concepts section.

1.4.2 Additional Resources

A glossary of the terms italicized in the document are available in the GENI Terms and Concepts section.

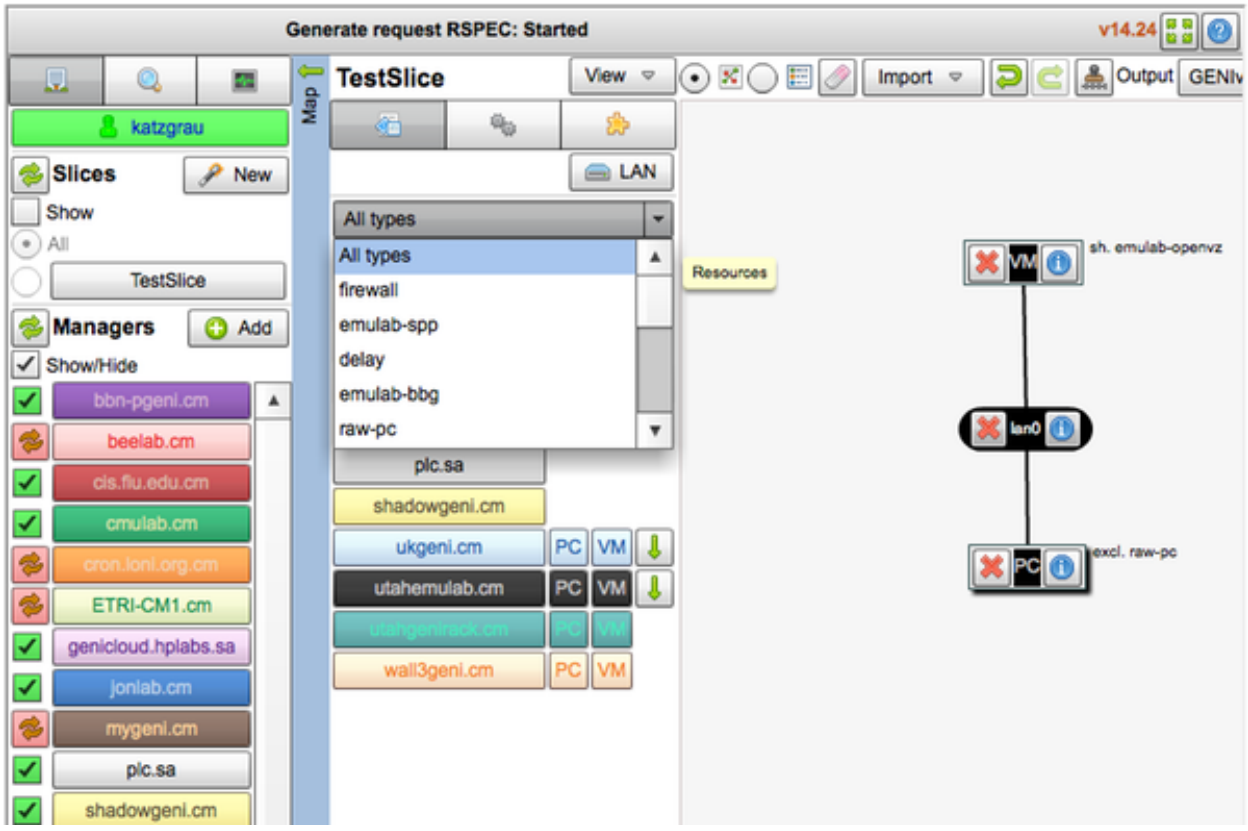
A more encompassing document with lower-level details on GENI architecture is available here[18].

1.5 Key Terms and Concepts

Before you jump into reading the core GENI documentation, you should become familiar with the terminology used when discussing the project. Because GENI is a recently-developed, global platform, an evolving set of terms is used to described the details.

Throughout the definitions, we'll refer the figure below for reference. This is *Flack*, the visual interface used to interact with GENI. Flack is described in further detail below and in the Flack guide.

Figure 1: Flack



[18] http://groups.geni.net/geni/attachment/wiki/GeniSysOvrvw/GENI-AnIntroduction-28Feb2012.pdf

1.5.1 Terms List

The list of terms below is organized in a way that the items further up on the list will help build your understanding of the terms toward the end of the list. Terms mentioned within definitions are in *italics*.

Resources

A *resource* is a generic term used to describe anything you can request and use on the GENi platform. It may include physical machine, virtual machines, network hardware and more. in the image of *Flack* above, the types of *resources* you can request from GENI are in the drop-down menu in the center of the screen (All types, firewall, etc).

Credentials

Your *credentials* are your keys to using the GENI environment. Once you have GENI *credentials*, you may begin setting up environments, requesting resources, and using the various tools that have been developed for GENI. Your credentials come in the form of a username, SSH key, and a password.

Flack

Flack is pictured at the top of this document. It's is a visual tool for setting up experiment environments (known as a *slice*), requesting resources (such as a *sliver*), and setting up network paths between them. **Much of your interfacing with GENI in experiments and examples will be through Flack**. You can learn more about using Flack here.

Omni

Omni is a command-line tool for interacting with GENI. You can allocate resources like you can do with Flack, but without the visual interface. Omni is a powerful utility, which you can learn more about here[19].

Slice

A *slice* is an environment and conceptual container that your resources reside in. A single *slice* is recognized globally throughout the GENI environment. Within each slice, you can request resources from any GENI authority. Note that in the image of *Flack* above, A slice named "TestSlice" has been created.

Sliver

a sliver is a set of resources that are reserved in an AM for a slice, i.e. slivers are what a user has in specific AMs

Up until now, we have referred to the different computing and network components that you allocate within GENI as *resources*. On GENI, you may have resources in many different places. For example, maybe you have resources allocated in Utah, and you have other resources allocated in Washington DC.

[19] http://trac.gpolab.bbn.com/gcf/wiki/Omni

In GENI terminology, a *sliver* refers to the set of resources in a single location. So that group of resources in Utah is one sliver, and the grouping of resources in Washington DC is another sliver.

Your slivers are all part of the same *Slice*.

In the Image of *Flack* above, you can see the sliver that is going to be allocated for the *slice* named 'TestSlice': 1 Virtual Machine (VM), 1 Physical Machine (PC), and network interfaces to link them (lan0).

Clearinghouse

A *Clearinghouse* is a GENI authority that is capable of signing users up to experiment with GENI. In the GENI architecture, there are multiple clearinghouses. If you are gaining credentials through the The Mozilla Ignite App Challenge[20], your clearinghouse will be located at `pgeni.gpolab.bbn.com`.

Federation

This is the general idea that once you receive your GENi credentials from a *Clearinghouse*, you can use them to request resources and slivers anywhere. In visual tools such as *Flack*, this is largely transparent.

Management Authority (Authority)

Resources around the world are managed by what is called a *Management Authority*. This is typically an organization or institution that manages the servers and virtual machines on their premises, and makes them available to GENI as a whole. In the image of *Flack* above, the "Managers" list on the far left are the *Management Authorities*.

Resource Specification (RSpec)

Also largely transparent to the end user when using tools such as *Flack*, a *Resource Specification*, more commonly called an *RSpec*, is an XML document that lists resources. It is used in two primary cases:

1. When querying a *Management Authority* on what types of resources it has available. Before you ask for a few machines for your experiment, you need to know what is available to you, right? This is called an *Advertisement* RSpec.
2. Once you get a list of resources available to you, and you decide what you need, you send an RSpec that describes the resources you are requesting to the appropriate *Management Authorities*.

Again, with a visual tool such as Flack, you don't have a reason to build or read RSpecs manually. Flack does the work for you. If you are curious although, Flack allows you to both read the RSpecs it has created for you, and import RSpecs that you have written yourself or saved prior.

Ticket

[20] http://mozillaignite.com

When a request is made for resources (And an RSpec is sent out), the *Management Authorities* typically issues *Tickets*, which is essentially a promise to fulfill the request. Sometimes, requests cannot be fulfilled for various reasons.

Seattle

Seattle is another project under the GENI umbrella which allows you allocate virtual machines around the globe and run experiments. Some of these virtual machines reside on physical machine, host virtual machines, and even mobile devices. Because of the transient connectivity of some devices, it is **very helpful** for simulating what the internet is really like for experiments. You do not need GENI *credentials* to use Seattle. Anyone can sign up.

1.5.2 Additional Resources

This list is a modification of the original GENI glossary available at GENI.net. It has been modified to be of use specifically to developers. You are encouraged, however, to read as deeply into GENI as you see fit.

1. Geni.net GENI Glossary[21]

[21] http://groups.geni.net/geni/wiki/GeniGlossary

CHAPTER 2

Setup

You're now ready to sign up and start experimenting with GENI. The first thing you need to do is acquire GENI credentials. When that's finished, you're encouraged to try out the GENI Quick Start in order to see the fundamentals of working with GENI.

Contents

2.1 Get Your GENI Credentials

Before you can jump into using GENI tools and services, you will need to obtain an account with a GENI clearinghouse.

Mozilla offers GENI access to developers and experimenters participating in the Mozilla Ignite App Challenge[1]. If you're interested in hopping on GENI and getting access to gigabit speed and OpenFlow testbeds, send a note to kenny@mozillafoundation.org[2].

2.2 GENI Quick Start

Getting started with GENI and building a solid understanding of its capability is as easy as:

- Obtaining GENI credentials
- Reading up on Key GENI Concepts
- Working through Example 1

At that point, you may feel comfortable enough to start exploring what sort of tools and resources are available to you on GENI, or perhaps dive into gigabit networking and OpenFlow.

So after you've completed the 3 points above, branch out and explore:

[1] https://mozillaignite.org/
[2] kenny@mozillafoundation.org

- Running gigabit-speed experiments on GENI
- Running OpenFlow on GENI
- Finding out what features and resources are available
- Other tools that can be used to manipulate GENI resources

And lastly, if you're a hacker who wants to build webapps of the future, be sure to check out the list of tips and trick for hackers, which will come in handy as your apps and experiments become for sophisticated and user-oriented.

2.3 Tips and Tricks for Hackers

There are a few handy tricks that you can use to fully experience the apps you're building. This document contains a few tricks that may be useful.

2.3.1 See Your Gigabit App Through A Browser

If you have a box in Utah and another in DC, as we did in example 2, with a gigabit link in between, it may be nice to see your app through a browser.

For example, perhaps one box was intended to act as a client, and another as a web server, because you're building a next-generation webapp that relies on a high speed connection. Using you application from home doesn't take advantage of the high speed link the client box has, so you need to *see* your app through the eyes of the client, as if you were sitting in Utah.

`X`, commonly called `X11` is a tool that can use to gain a remote desktop capability with your provisioned client. This is possible if:

1. The system you are remoting **from** (your Mac, Thinkpad, windows box, etc) supports X11
2. The system you are remoting **to** supports X11, and has SSH installed

In terms of requirement number 1, this is easiest is your laptop or desktop has a recent version of OSX or Ubuntu installed. It's possible with a Windows computer too, with a little extra work.

Requirement #2 is easily satisfiable if the system you are remoting to has Ubuntu installed. From there, getting X11 and a Firefox browser installation is as simple as:

```
$ sudo apt-get update
$ sudo apt-get install xorg openbox ubuntu-desktop firefox
```

And then from your terminal:

```
$ ssh -X username@pcXXX.genihost.net
```

At that point, any windowed application you run in the terminal on the remote host will open a window on your local machine. Try something like:

```
$ firefox
```

And you should see a window appear, showing your Firefox browser running remotely.

2.3.2 Accessing Nodes Behind Firewalls

In a couple of the example projects, we set up nodes in Utah and Washington DC. In DC, the names of our nodes were prefixed with `pg`. For example, the name of the node might be something like `pg502`.

Nodes prefixed with `pg` are behind a firewall, and accessing them can only be done through another node networked with them. For that reason, it's easiest to set up an SSH tunnel to reach the `pg` node.

This is a necessary part of completing Example 2, so it would be good to know how this works ahead of time.

Setting Up an SSH Tunnel

Let's say that we have two GENI nodes set up: `pc100`, and `pg200`.

Locally, you would open up two SSH terminals. One will be for connecting to `pc100` at pc100.emulab:22.net, and the other will be for connecting to `pg200` at pg200.emulab.net:22.

We can easily connect to `pc100` directly in one terminal. But in the process, we're going to have a tunnel open on that same connection. For example, we'll execute something in the form of:

```
$ ssh -p [pcxxx port] [username]@[pcxxx host] -L 2222:[pgxxx host]:[pgxxx ssh port]
```

How generic! Let's put that into perspective for our example, with the username 'katzgrau'. We'll execute:

```
$ ssh -p 22 katzgrau@pc100.emulab.net -L 2222:pg200.emulab.net:22
```

At that point, you should have connected by SSH to `pc100`. Now, we're in the clear to connect to `pg200`. The generic for would look like:

```
$ ssh -p [local tunnel port][username]@127.0.0.1
```

But in this example, we'll execute:

```
$ ssh -p 2222 katzgrau@127.0.0.1
```

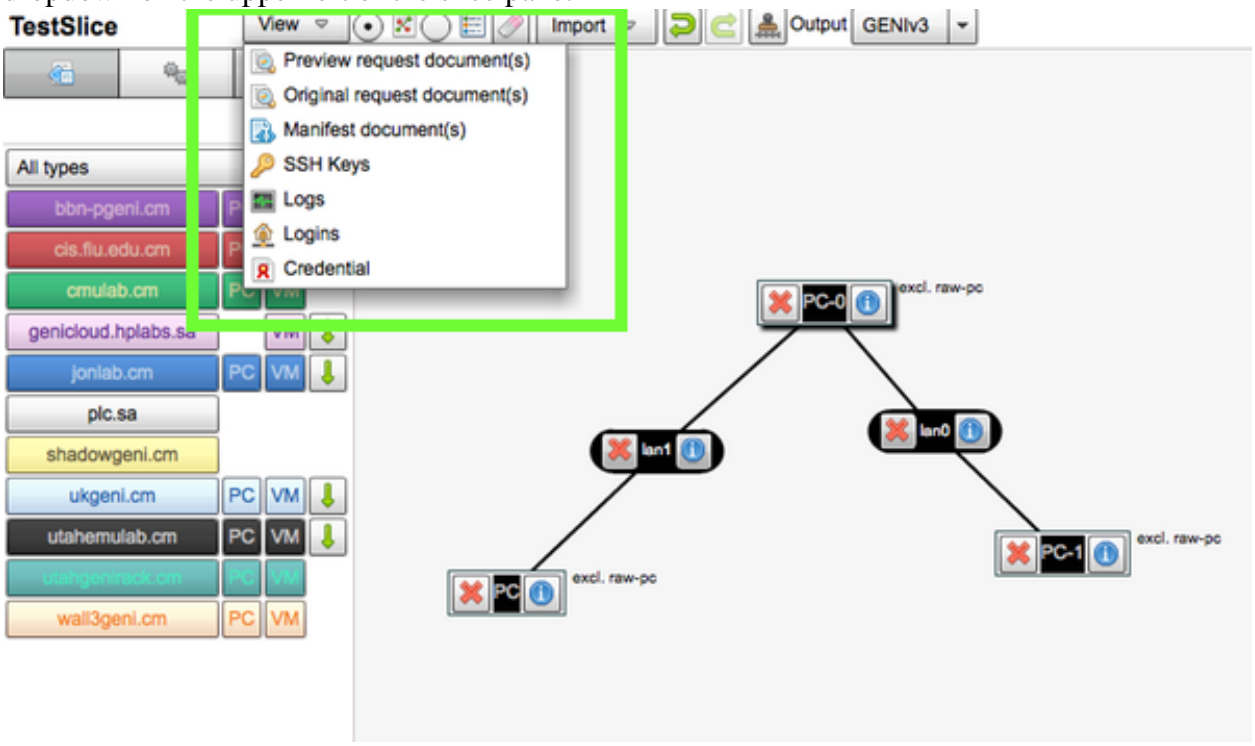
Our SSH connection will be routed through `pc100` over to `pg200`, and we should be good to go!

If you find yourself having trouble with this, it may be wise to read up on SSH tunnels in general[3]. They can be confusing to a newcomer, but that barrier is easily overcome. In addition, creating tunnels can be very handy in your experiments.

2.3.3 Saving Configurations/RSpecs

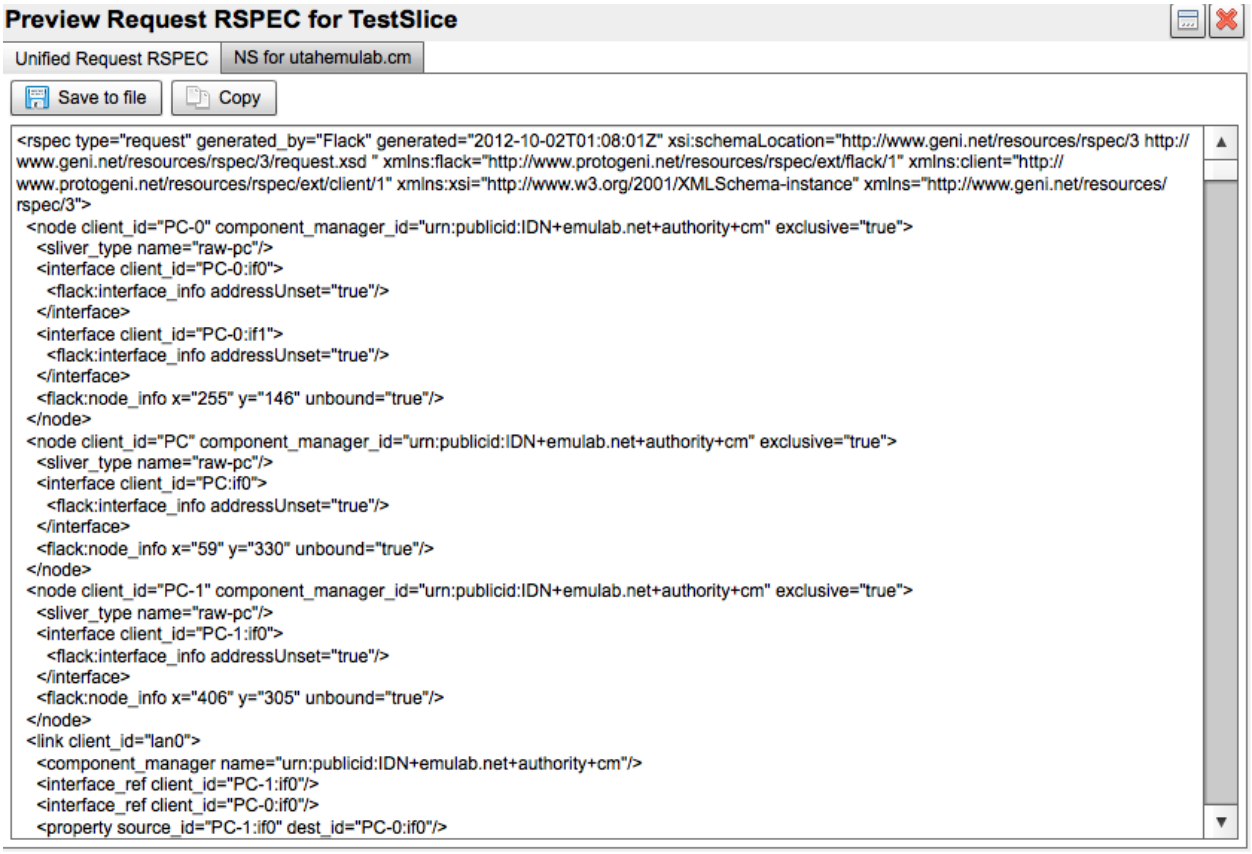
If you've gone through the trouble of setting up a complex topology with specially configured nodes, it can be extremely helpful to save configuration of your experiment into an RSpec/text file. Next time that you want to set up the same experiment, you can load in the RSpec.

To see the RSpec that was generated while you were configuring your project, click the "View" dropdown on the upper left of the slice pane:

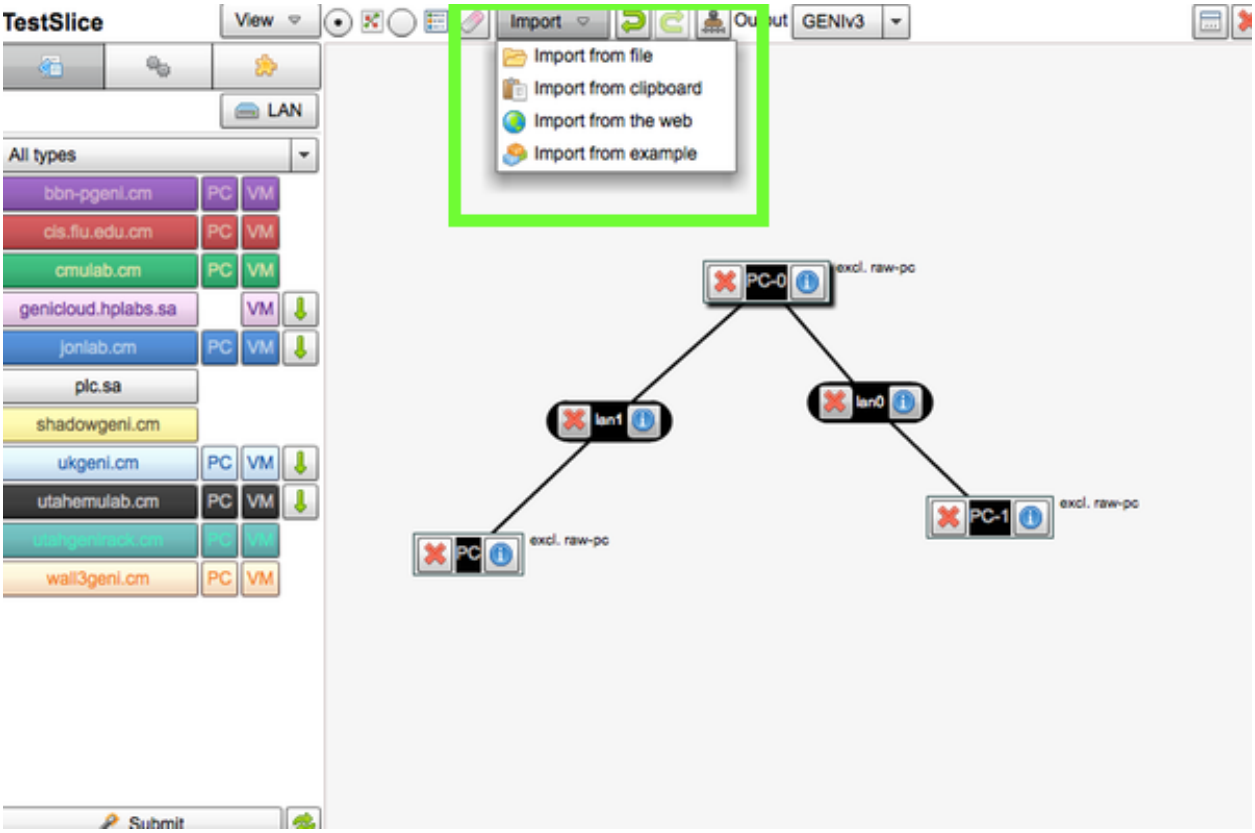


From there, click "Preview request documents", and you should see:

[3] http://www.revsys.com/writings/quicktips/ssh-tunnel.html



You have the copy to copy this RSpec or save it to a file. It's probably best to save it to a file so you can load it up later. When that time comes, click the "Import" dropdown.



When you import the RSpec, your experiment will be created with all the same settings and configurations it had before.

CHAPTER 3

Examples

Contents

3.1 Example 1 - Set Up 2 VMs and Ping

In this short experiment you'll learn how to set up 2 virtual machines that can ping each other on the GENI platform. You'll need to get GENI credentials before you can complete this example.

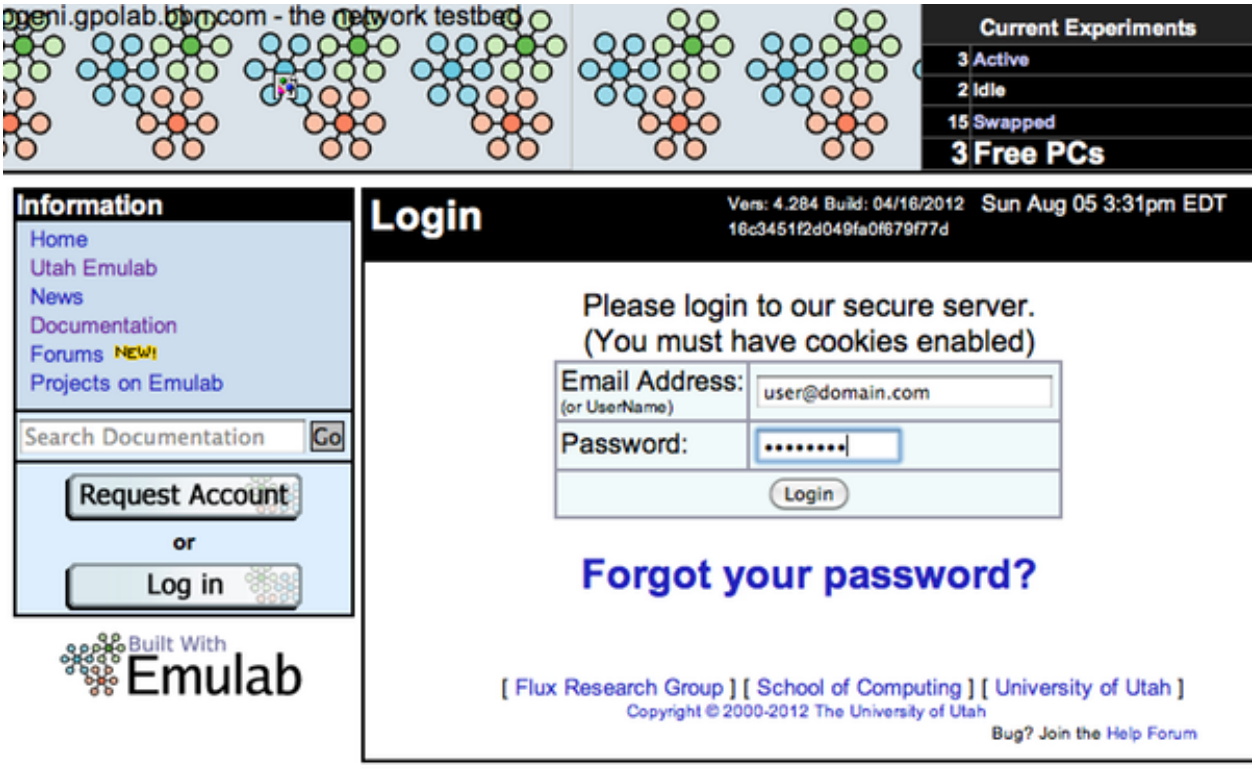
3.1.1 Tutorial

If you run into any problems with this tutorial, reach out to kenny at mozillafoundation dot org.

Step 1. Log In to Your Management Authority

For this tutorial we're going to use Flack, the browser-based visual tool for interacting with GENI. The easiest way to authenticate with Flack is to simply be logged in to your clearinghouse' website in one window or tab, and to have Flack open in another window or tab.

Figure 1: Logging in to a management authority (yours may differ)



Step 2. Open Flack In A New Tab

Flack is an in-browser tool written in Flash. It's likely you already have flash installed, but if you don't you can get it here:

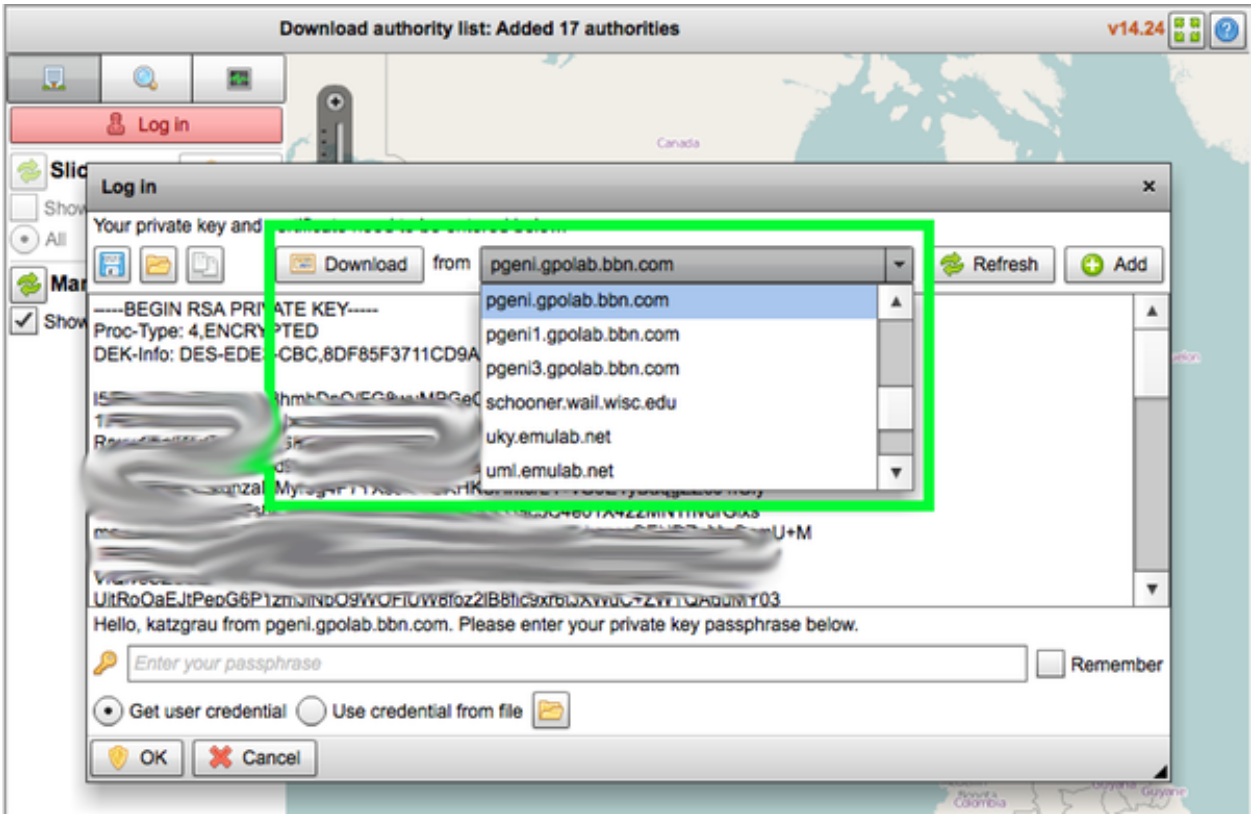
Open Flack in a new tab or browser window. Flack can be found here: www.protogeni.net/trac/protogeni/wiki/Flack . You should see Flack initialize in the window, as in the figure below:

Figure 2: Flack Initialized

Step 3. Log In With Flack

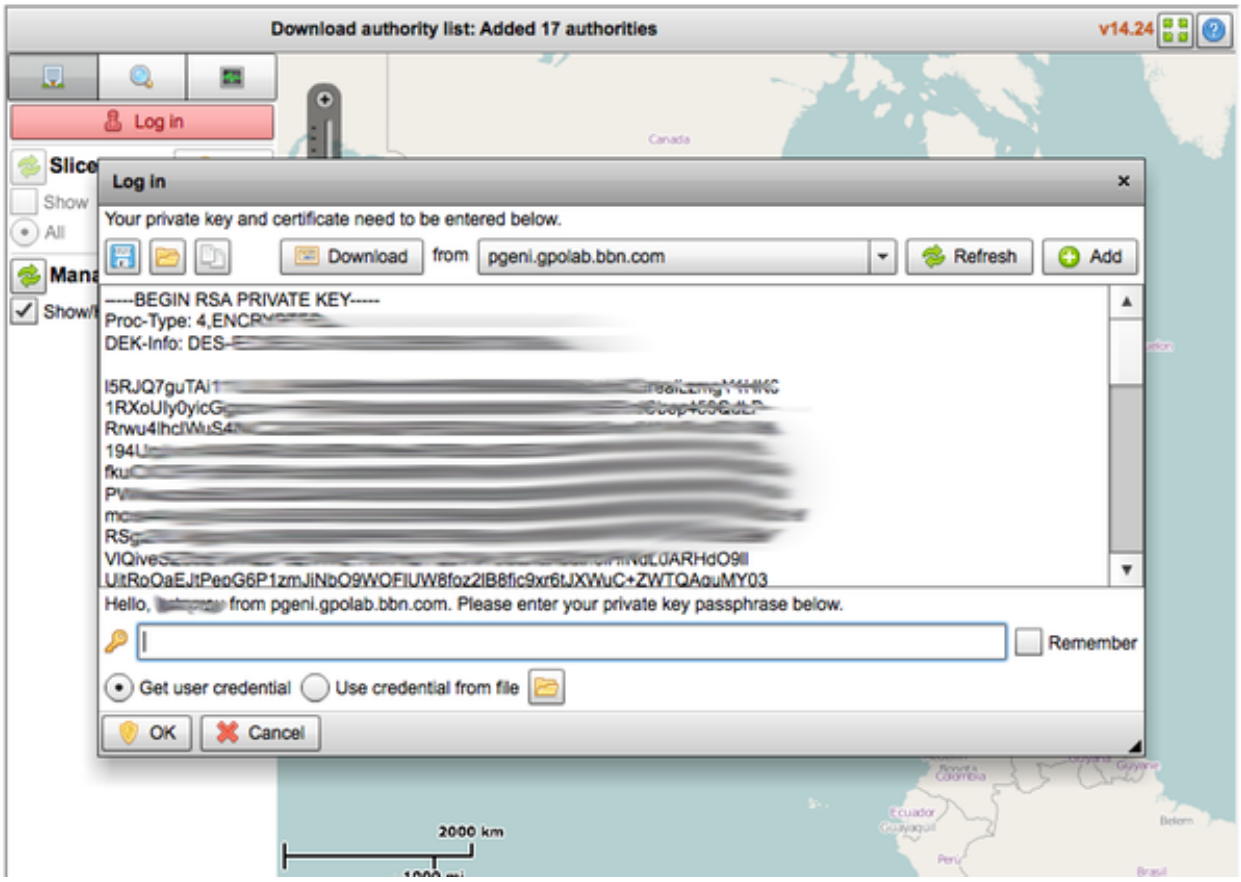
Click the green "Log In" button above. Here, you are expected to paste your GENI private key. But because we are already logged into our management authority via another tab, we can complete this step easily and have our private key information inserted automatically.

Along the top of the prompt, there should be a button and dropdown box that says "Download from [Select authority]" as in the figure below.



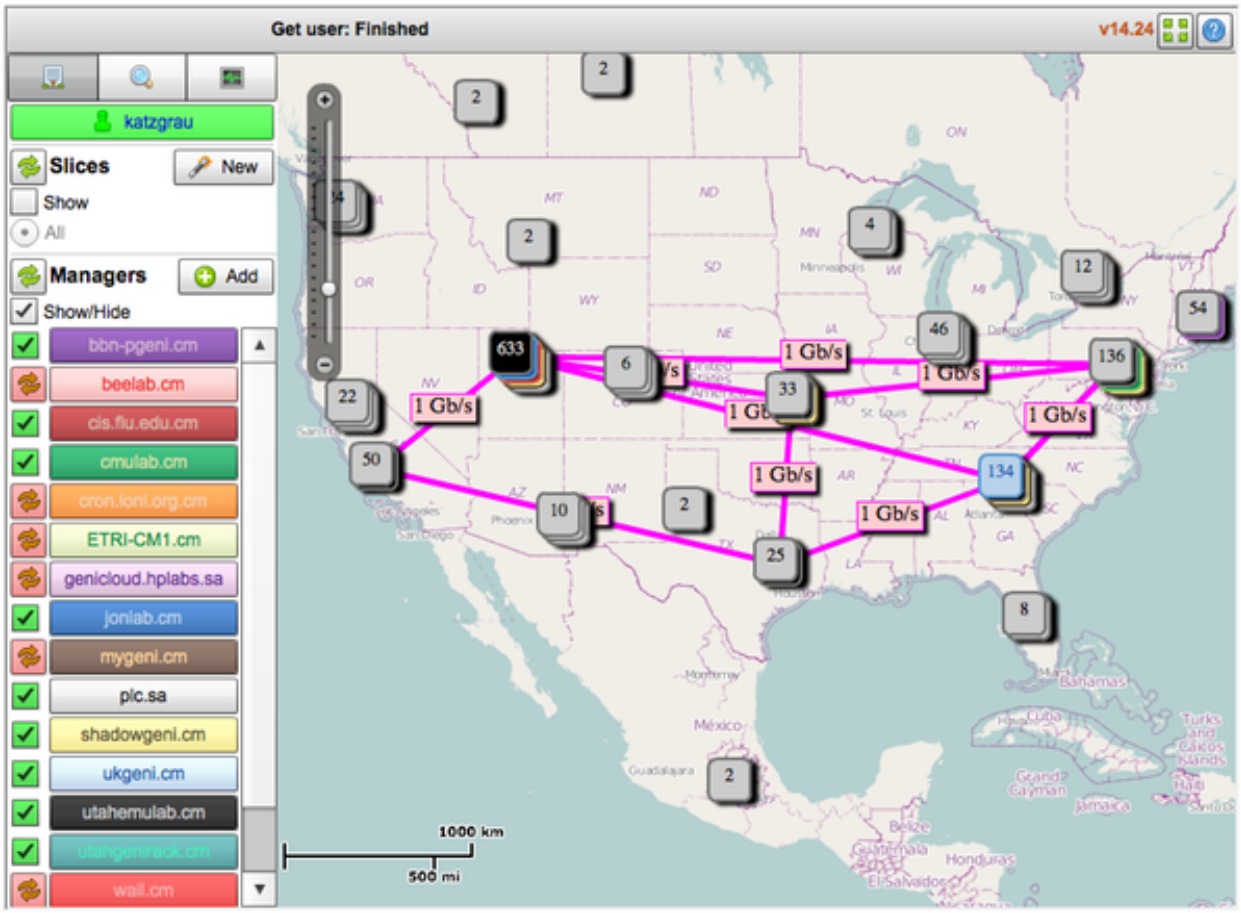
Select your management authority's domain from the dropdown. The correct domain to choose will match up with the domain where you logged in at Step 1.

Once the correct item is selected, click the "Download" button next to the option list. Your certificate (credentials) should appear, as in the figure below:



Finally, enter the private key passphrase that you used when you signed up at your management authority when you signed up. Click "OK" on the bottom left.

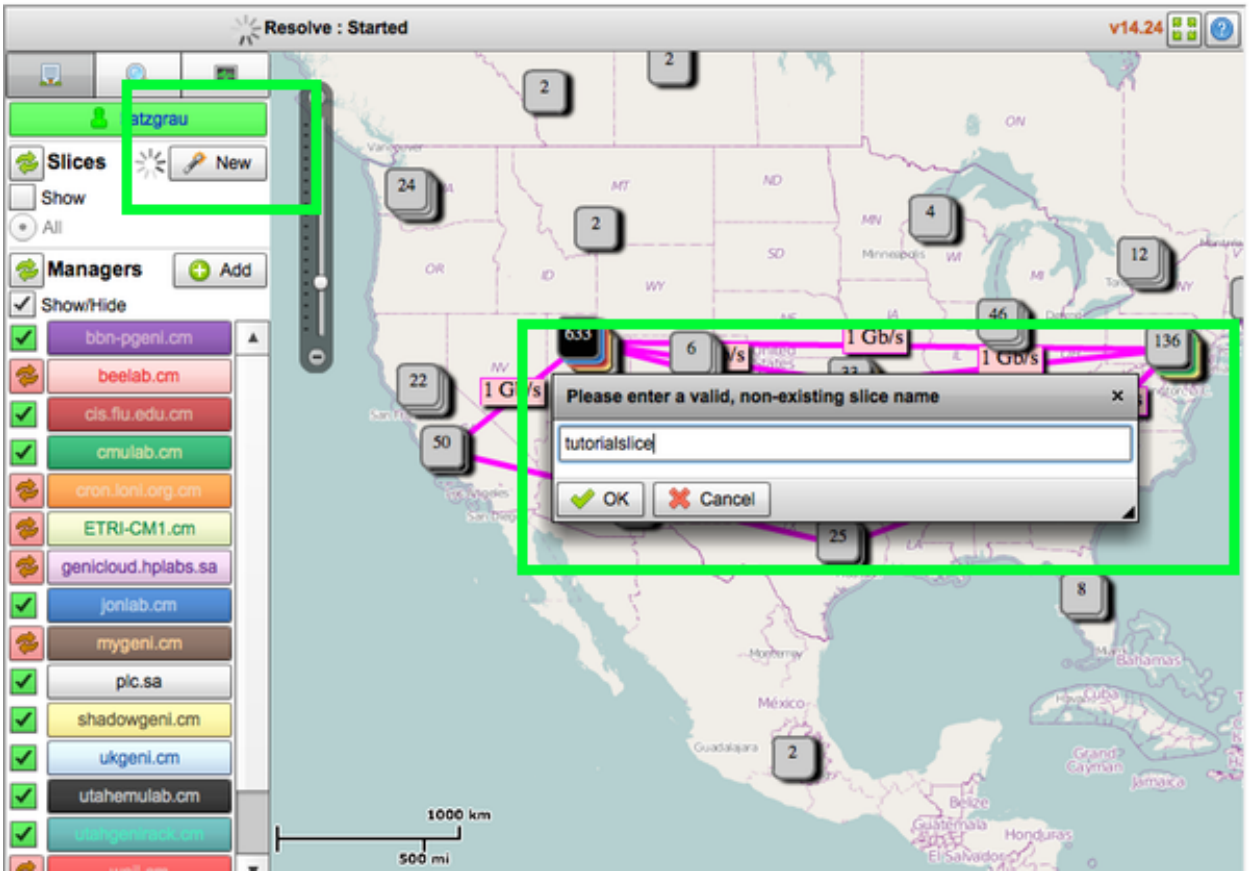
At this point, Flack will attempt to retrieve resources from the various authorities. When prompted to select where to list resources from, leave all of the options checked and click "Continue". Flack will gather the resources lists, and display a map of North America, as pictured below:



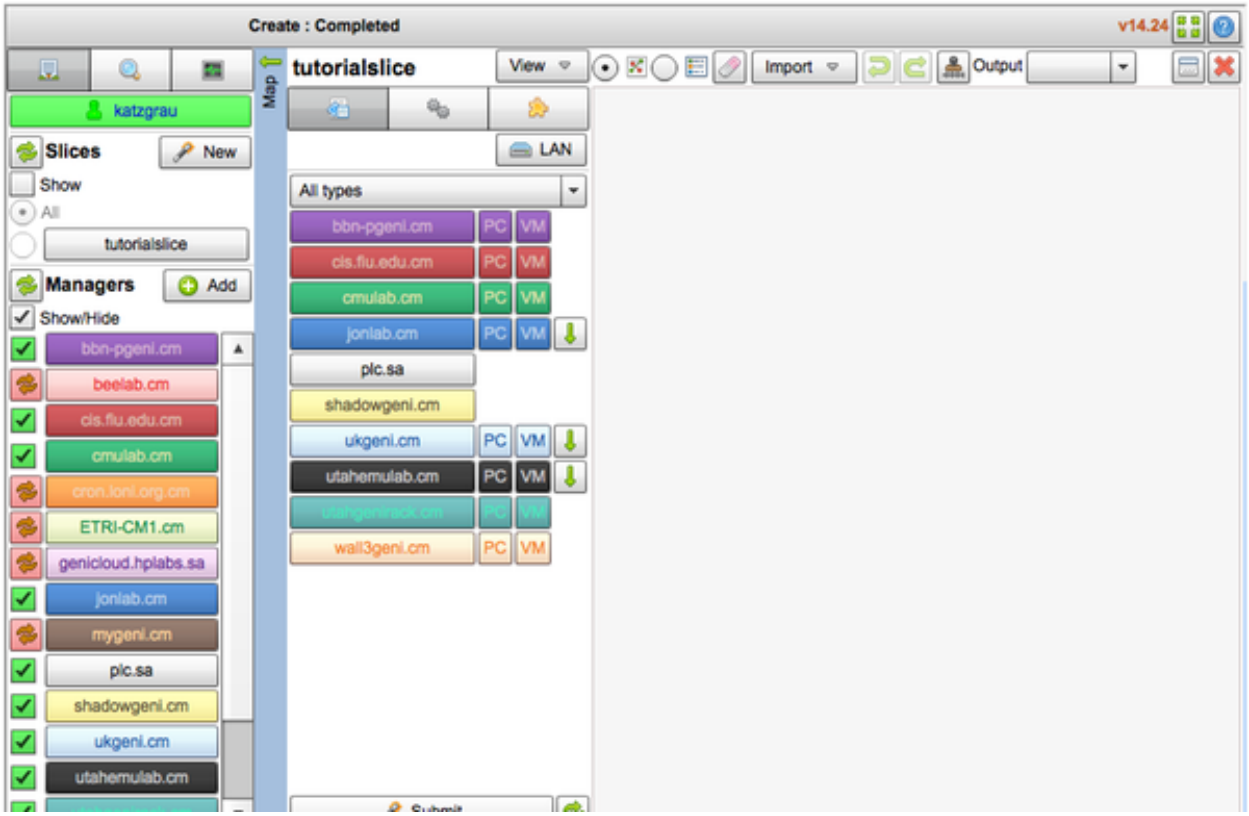
Step 4. Create a New Slice

You're now ready to create a GENI "slice." Your slice is a GENI-wide environment that you can add resources to, and run experiments in. For more on GENI terminology, see the glossary.

Click on the "New" button in the top-left corner of Flack, and enter a name for this slice. **Your slice name will have to be unique from all other GENI slices, so it cannot be the same name as in the example.**



When you're finished typing the name of your slice, click "OK". Once your slice has been created (it may take a moment), you'll see a blank pane with the name of your slice toward the top of your screen. The blank pane on the right is where we'll set up our experiment.

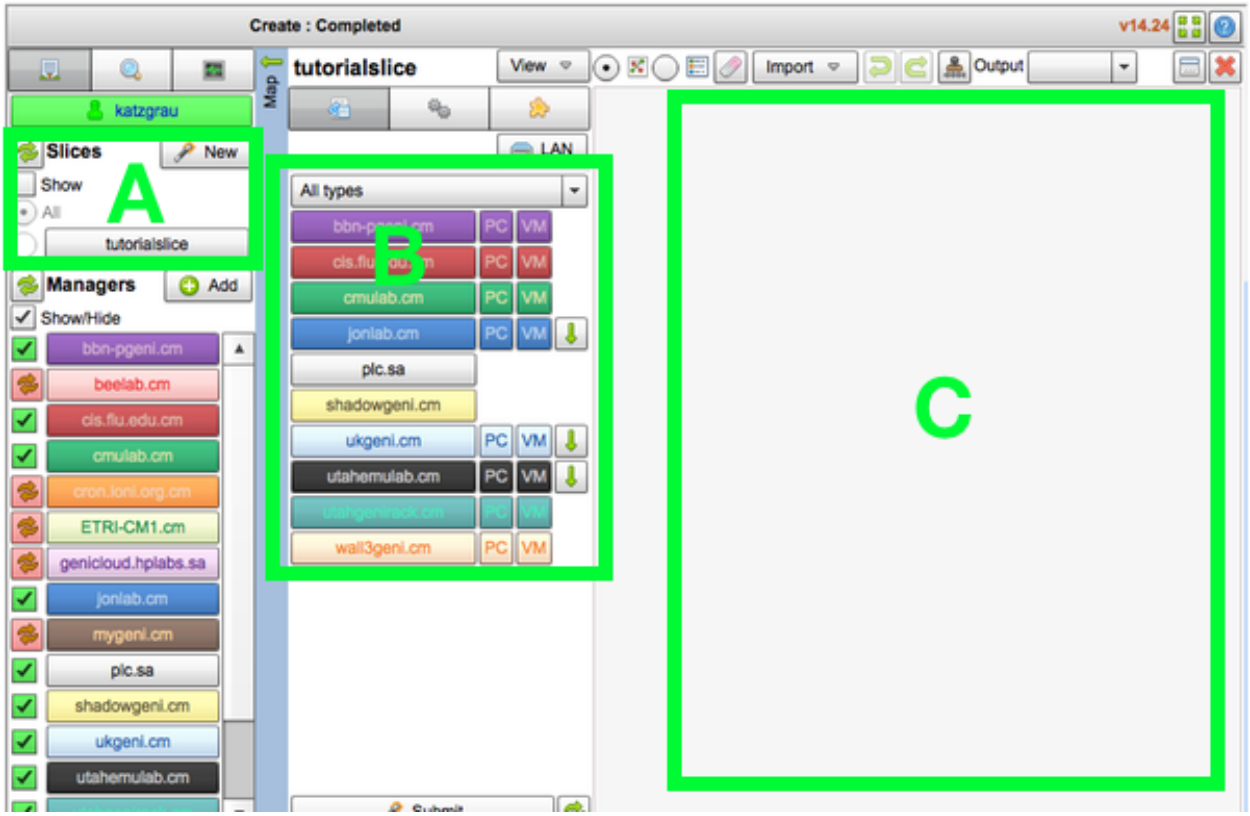


Step 5. Set Up Two Machines and A Link

Now for the fun part!

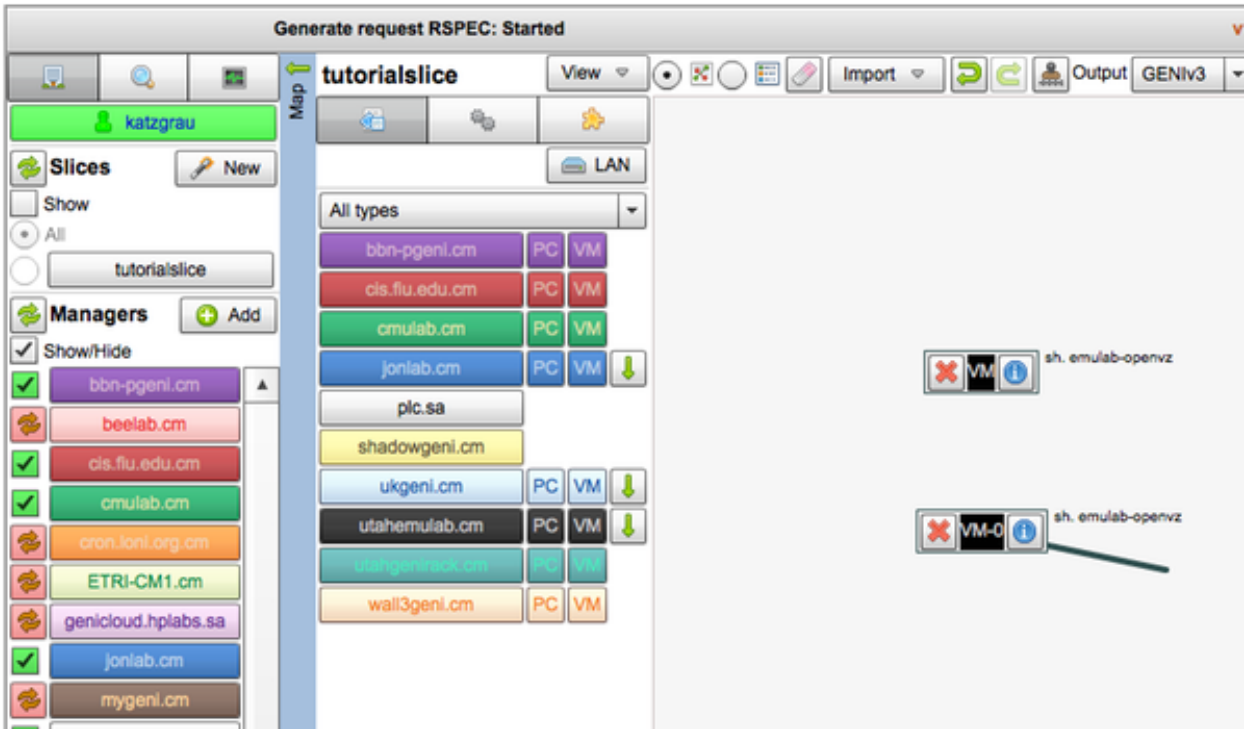
In the image below, you'll see:

- **A**: The area where new slices are created, and your existing slices are listed
- **B**: The area where the resources that you have available to you are listed (like VMs and PCs)
- **C**: The area where you can drag and drop resources

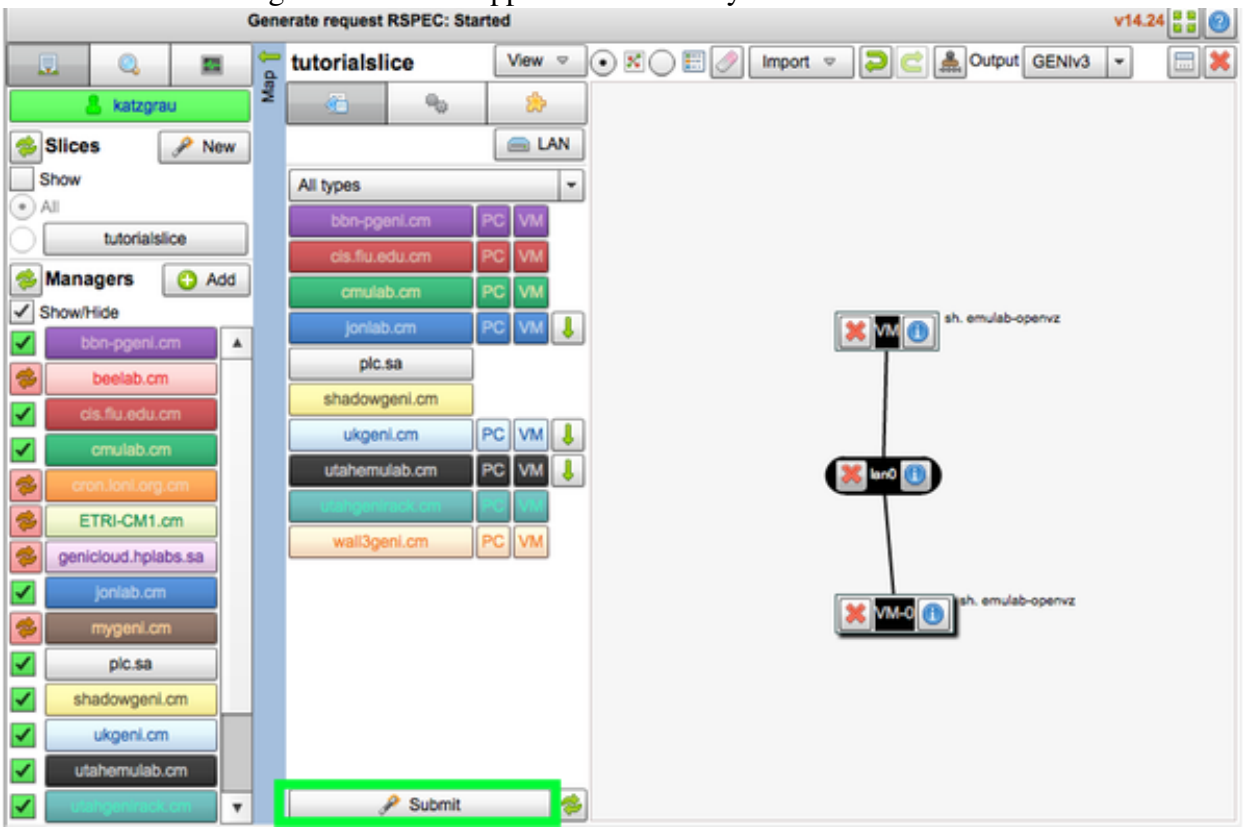


In area B, there will be an authority listed as 'utahemulab.cm'. Drag the "VM" component immediately next to it to the area on the right. You should see:

Now, drag another VM onto the pane:



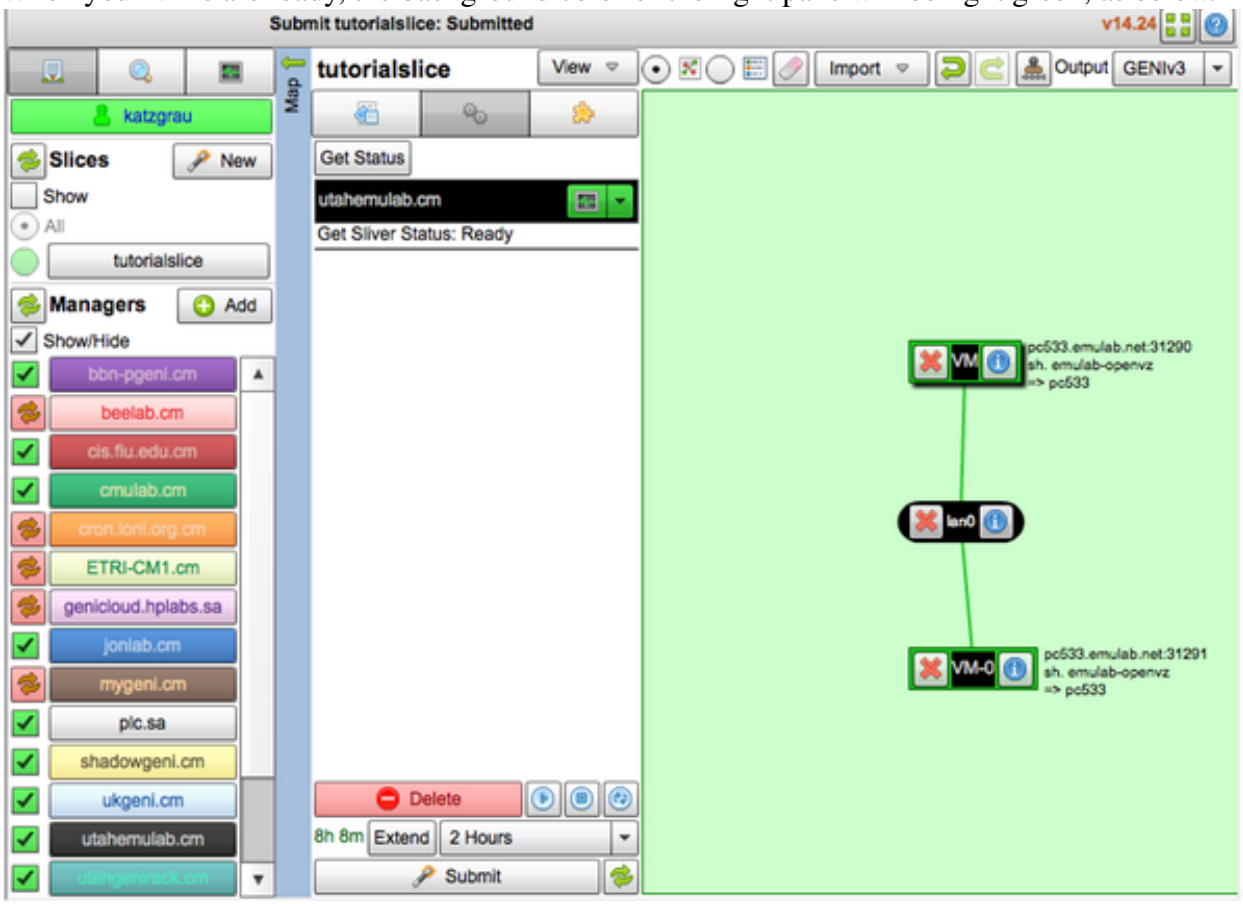
Lastly, drag a line between the two VMs, and click "Submit" on the bottom. Keep in mind, the 'lan0' device in the figure below will appear automatically.



At this point, you have two virtual machines on the GENI platform that are being provisioned.

These two VMs are networked, and will have entries in their hosts file, (`/etc/hosts`) automatically created which point to each other. **In other words, from this visual tool (Flack), you created two real virtual machines that are networked with each other on GENI.**

When your VMs are ready, the background color of the right pane will be light green, as below.



Step 6. Send A Ping From One Node to the Other

At the last step in Step 5, you should see the connection info for each node listed to the right of it. For example, the hostname and port of the nodes in the example are.

- **VM**: Host `pc533.emulab.net` port `31290`
- **VM-0**: Host `pc533.emulab.net` port `31291`

If I open a terminal window, I should be able to connect to both without using a password:

Logging into VM

```
magnesium:~ katzgrau$ ssh -p 31290 pc533.emulab.net
Last login: Mon Aug  6 12:55:53 2012 from pool-96-234-6
3-96.nwrknj.fios.verizon.net
[katzgrau@VM ~]$
```

Logging into VM-0

```
magnesium:~ katzgrau$ ssh -p 31290 pc533.emulab.net
Last login: Mon Aug  6 12:55:53 2012 from pool-96-234-6
3-96.nwrknj.fios.verizon.net
[katzgrau@VM ~]$
```

As mentioned before, the hosts are already networked, and conveniently, there are already hosts for the appropriate IPs for VM and VM-0 in each host file:

```
magnesium:~ katzgrau$ ssh -p 31290 pc533.emulab.net
Last login: Mon Aug  6 12:55:53 2012 from pool-96-234-6
3-96.nwrknj.fios.verizon.net
[katzgrau@VM ~]$ cat /etc/hosts
127.0.0.1       localhost loghost localhost.tutorialsli
ce.pgeni-gpolab-bbn-com.emulab.net
10.10.1.1       VM-0-lan0 VM-0-0 VM-0
10.10.1.2       VM-lan0 VM-0 VM
[katzgrau@VM ~]$
```

At this point, we're ready to `ping` one host from the other. Open the window for machine VM and type:

```
ping -c 2 VM-0
```

This will ping VM-0 twice. The `-c` flag allows us to limit the number of pings to 2. You should see something very similar to:

```
[katzgrau@VM ~]$ ping -c 2 VM-0
PING VM-0-lan0 (10.10.1.1) 56(84) bytes of data.
64 bytes from VM-0-lan0 (10.10.1.1): icmp_req=1 ttl=64
time=0.025 ms
64 bytes from VM-0-lan0 (10.10.1.1): icmp_req=2 ttl=64
time=0.027 ms

--- VM-0-lan0 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time
 999ms
rtt min/avg/max/mdev = 0.025/0.026/0.027/0.001 ms
[katzgrau@VM ~]$
```

If you did, congratulation! You've just confirmed your two new GENI VMs can talk to each other! If you're a bit handy with client/server apps, what might you be able to code up at this point?

If things didn't go so well, see the troubleshooting section below and see if you can address any issues.

3.1.2 Troubleshooting

Did this experiment fail to go as planned? Ping kenny at mozillafoundation dot org and share the details. Or, see if your problem can be addressed below.

I clicked 'Submit' to create my machines, but there was an error

Occasionally a request to provision machines cannot be fulfilled. You can always try:

- Deleting the failed resources if necessary (a red background means failure)
- Submitting the request again
- Trying to provision machines from a different management authority

I can't log into my machines via SSH

Double check that your public SSH key (usually on your computer at something similar to `~/.ssh/id_rsa.pub`) is registered with your Management Authority user account. When your machines are provisioned by GENI, your public key will be placed in the `~/.ssh/authorized_keys` file on each host so you will have a password-less login.

3.2 Example 2 - Set Up 2 Machines with a Gigabit Link

One of the most exciting parts of using the GENI platform is having access to a gigabit backbone. In this tutorial, we're going to set up two networked machines just like we did in Example 1, but we're going to do two things differently:

- Set up a gigabit link between the two machines
- Set up the machines in different parts of the country
  - One in Utah
  - One in Washington DC

3.2.1 Considerations Before Proceeding

Important: Because GENI is a shared platform, it's your responsibility as a developer to be mindful of the resources you are requesting. The resources for gigabit networks are limited, so you

are encouraged to release those resources as soon as you are finished using them. This will make them available to other developers and experimenters.

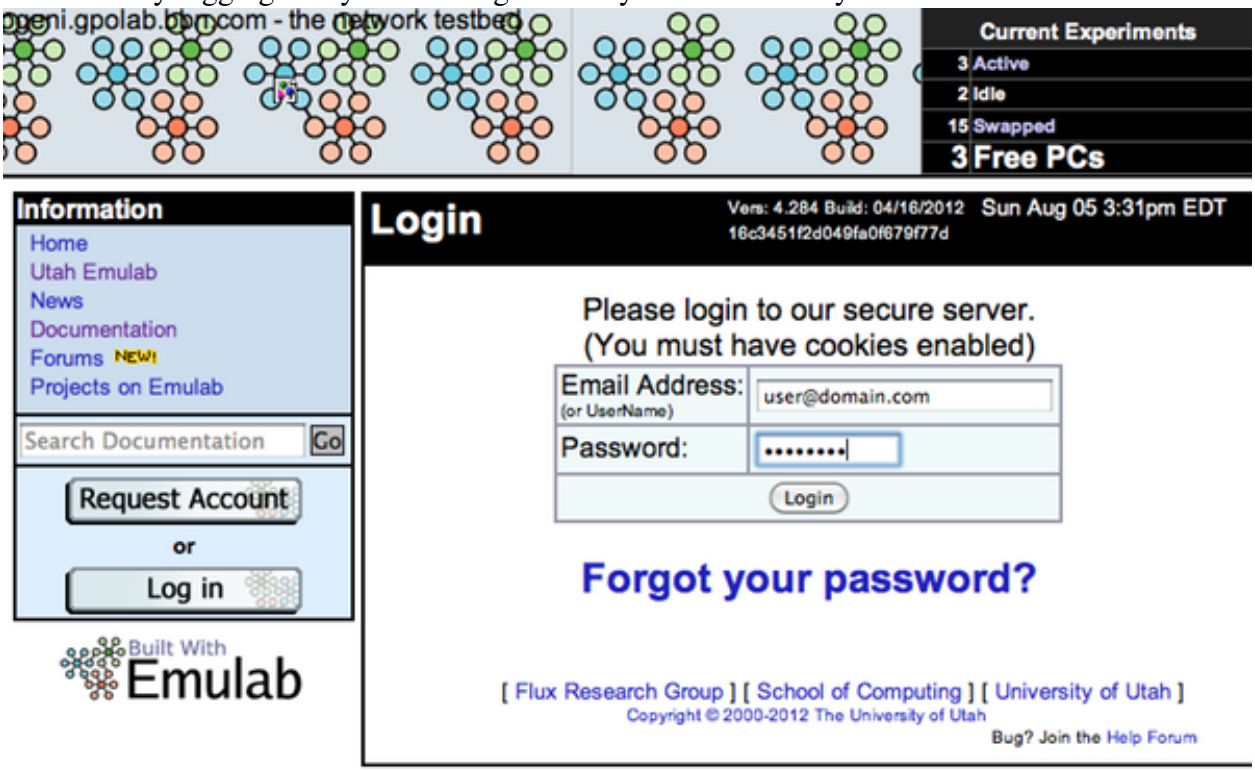
It's also highly recommended that you complete the first GENI example before beginning this one. This tutorial may reference introductory concepts that were explained previously.

3.2.2 Tutorial

The steps of this tutorial are very similar to the last tutorial, up until step number 4.

Step 1. Log In to Your Management Authority

Let's start by logging into your clearinghouse if you aren't already authenticated.



Step 2. Open Flack In A New Tab

Next, we'll open Flack in a new browser window or tab: http://www.protogeni.net/trac/protogeni/wiki/Flack .

Figure 2: Flack Initialized

Step 3. Log In With Flack

As we had done previously, choose the correct clearinghouse if it isn't already selected along the top of flack. Click the "download" button to have your keys loaded, and enter you password.

When prompted to select where to list resources from, leave all of the options checked and click "Continue".

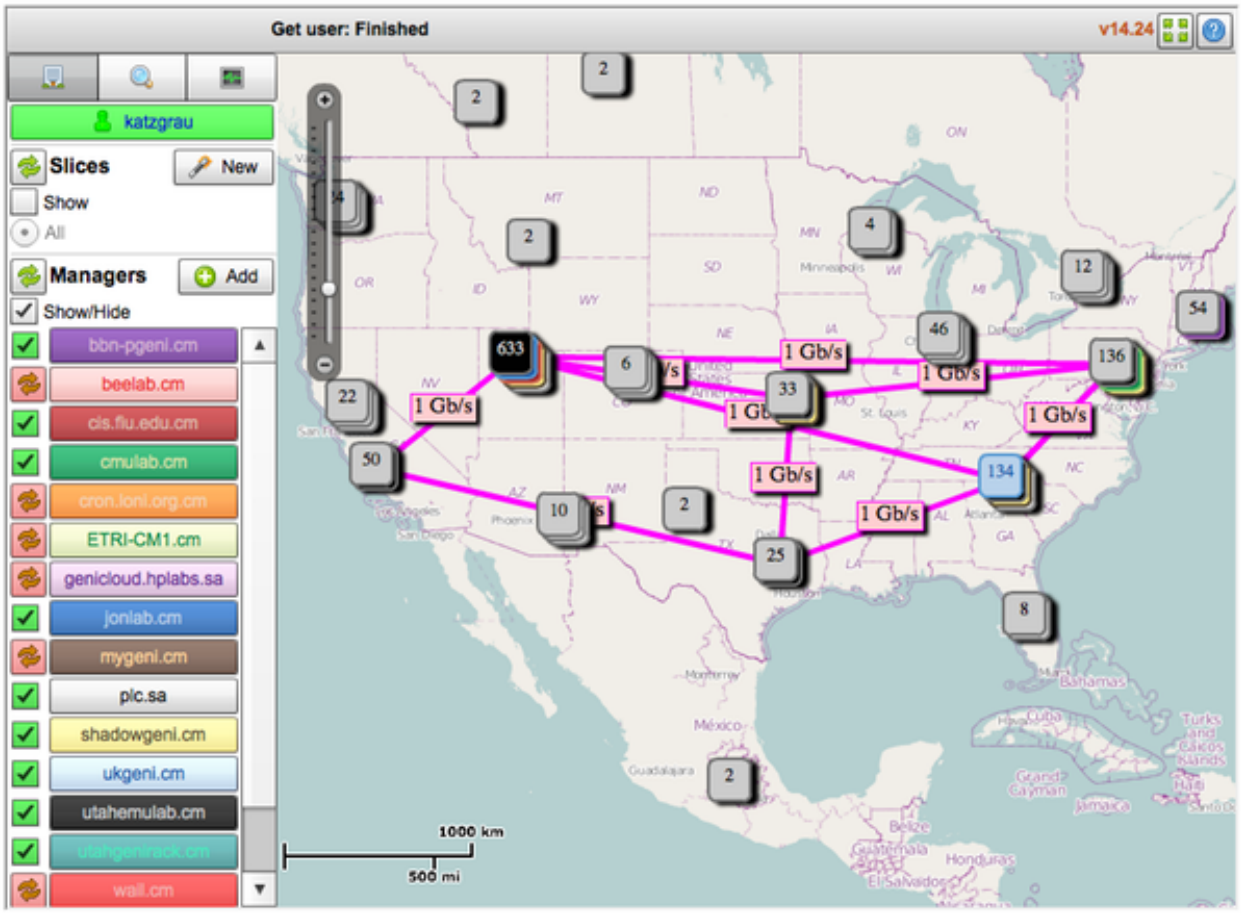
Step 4. Let's Look At Our Gigabit Backbones

Let's take a look at the map that is displayed when the authority list loading is complete. Do you see the pink lines running across the map? Those are **gigabit backbones** that are available to use.

In order to utilize a gigabit backbone, we'll provision a machine at both ends of it. In this example, one end will be in Utah and the other will be in Washington DC.

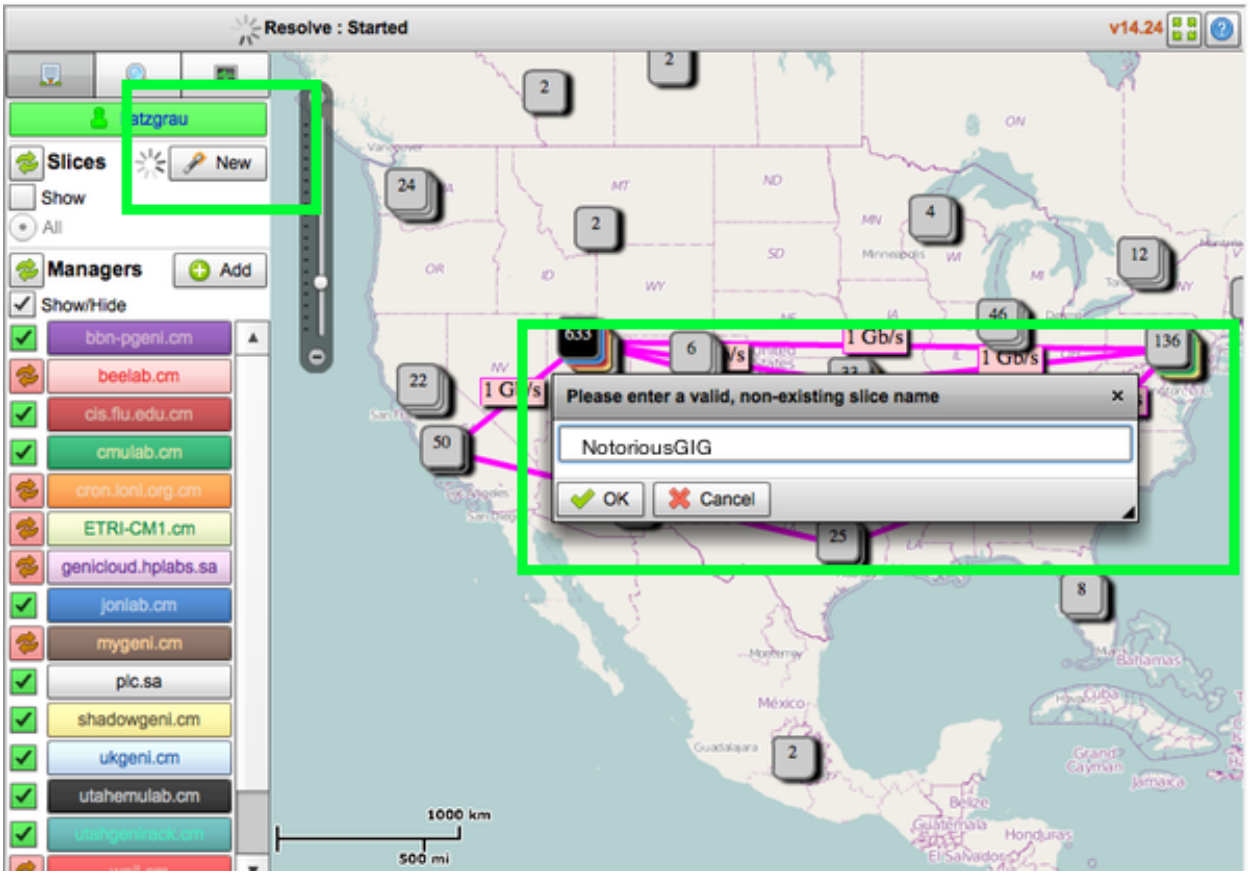
In the image below, the Utah aggregate is black and has the number "633" on it. Washington DC is grey and has "136" on top of it.

Note: Because those numbers denote the number of resources available in those aggregates, the number fluctuates, and is likely different at the time that you are reading this.

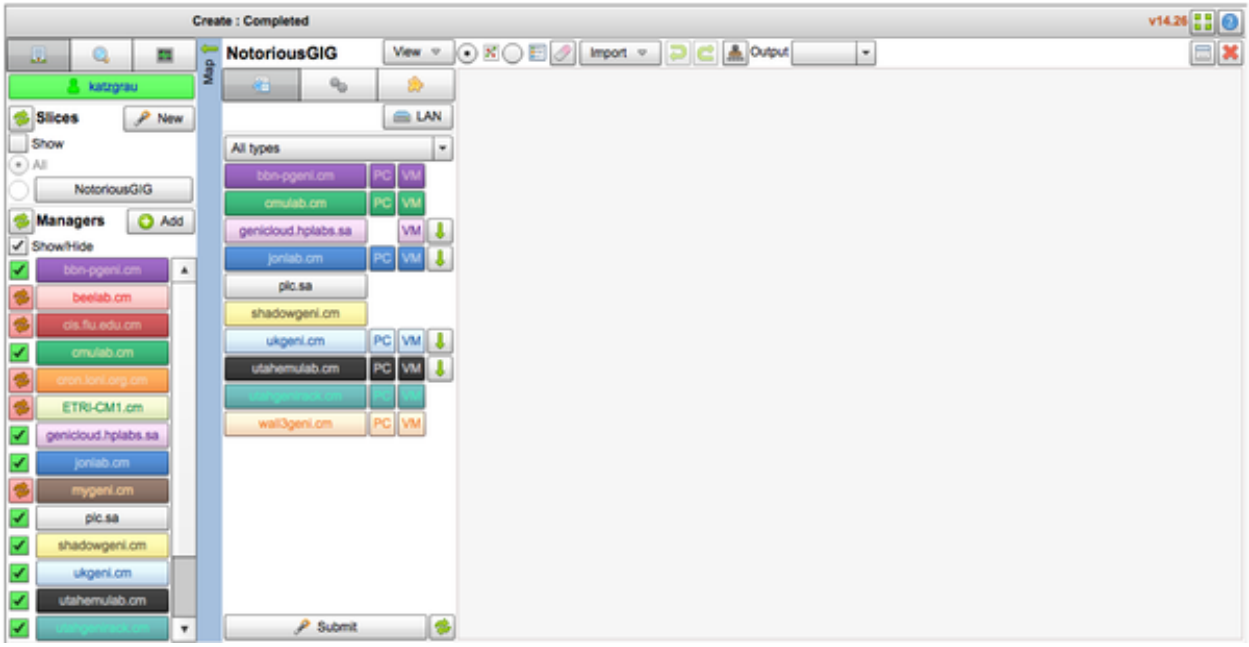


Step 5. Create a New Slice

Let's create a new slice for our gigabit experiment to live in. Give your slice a different name than the one in the figure below.

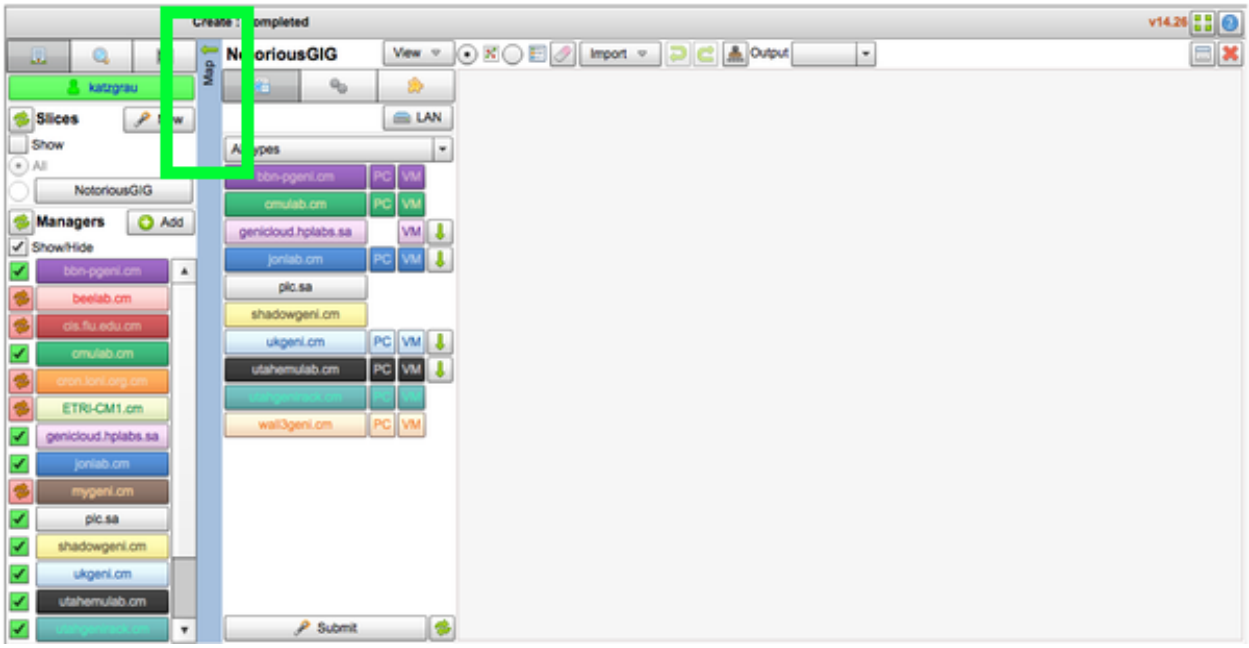


Click "Ok" and wait for your slice to initialize. When it's done, you will see:

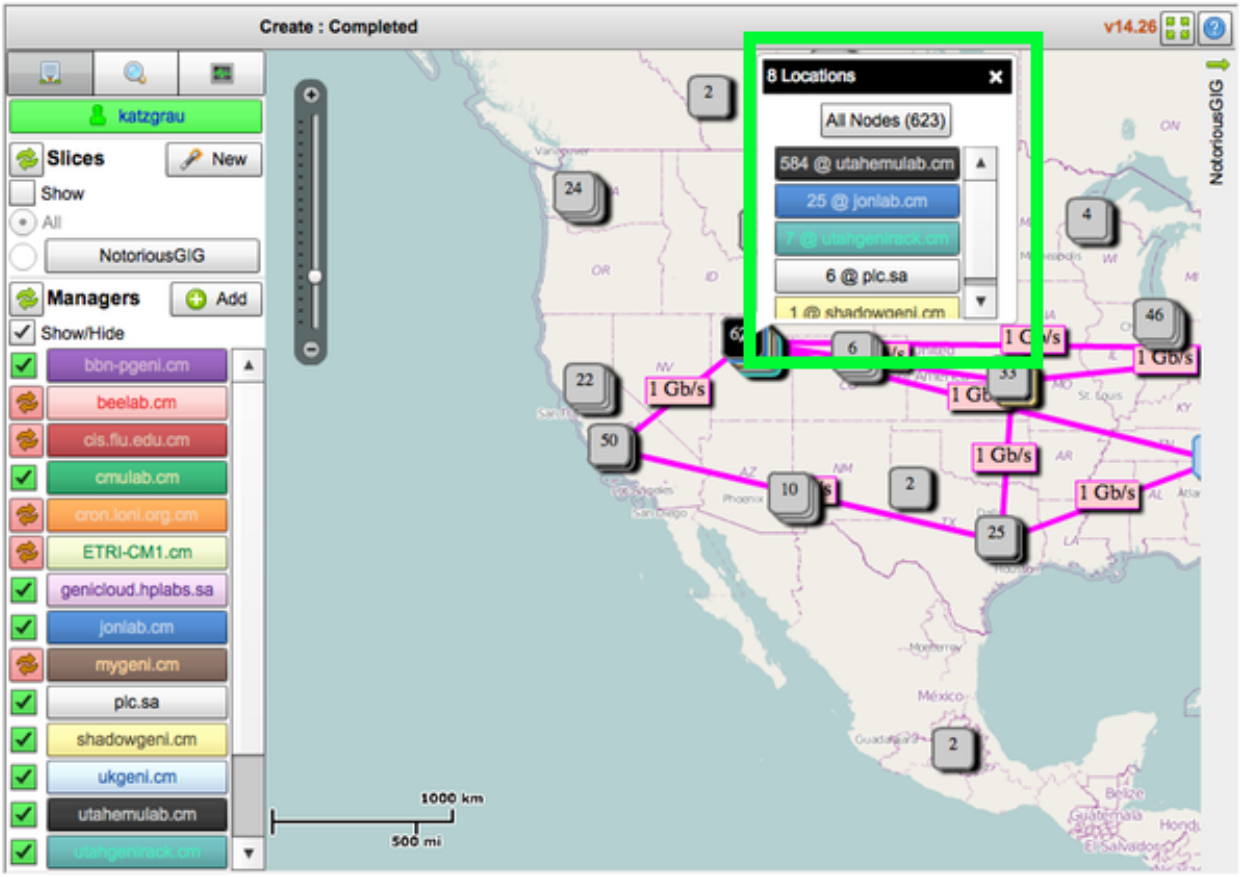


Step 6. Find A Machine With A Gigabit Link

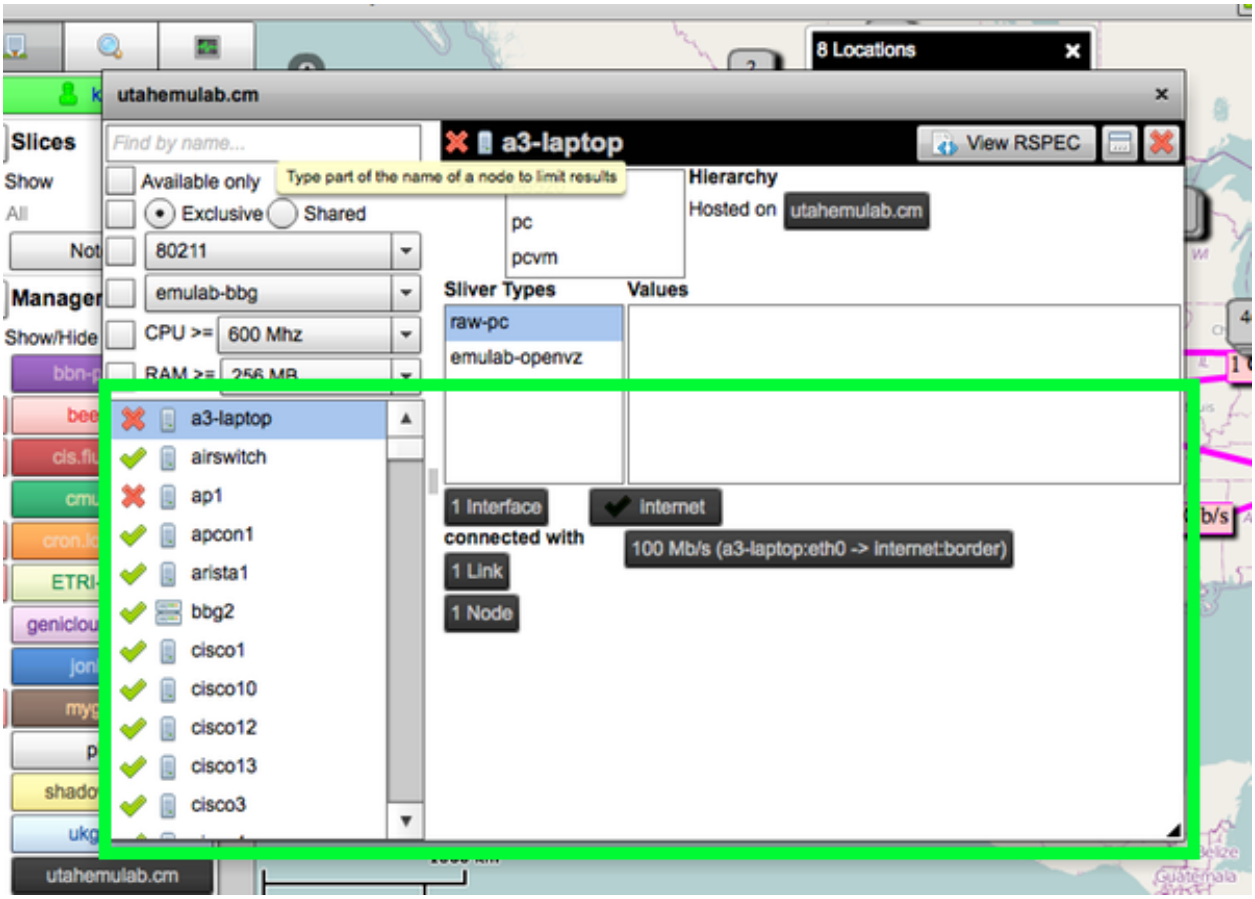
Just as before, we're at the fun part of the experiment. Let's go hunting for a machine that's hooked up to the Gigabit backbone in Utah. Click on the Map panel in the left part of the screen:



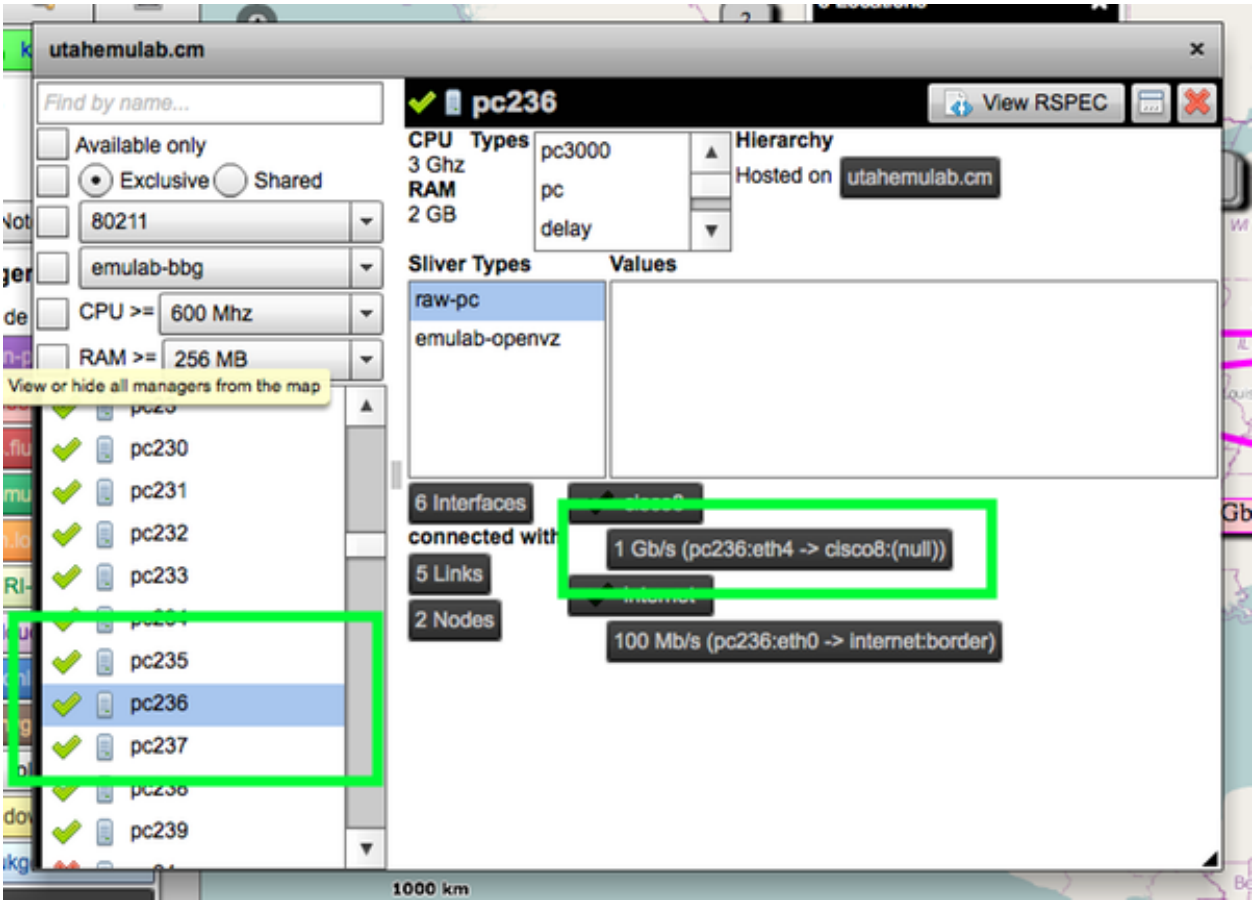
Click on the Utah aggregate. You'll see a box pop up with a breakdown of the different authorities in that grouping. For these tutorials, we'll stick with Utah/emulab, since they have the most resources available at the time of this writing.



Click on the box labeled “utahemulab.cm” to see the list of resources available. You’ll see:

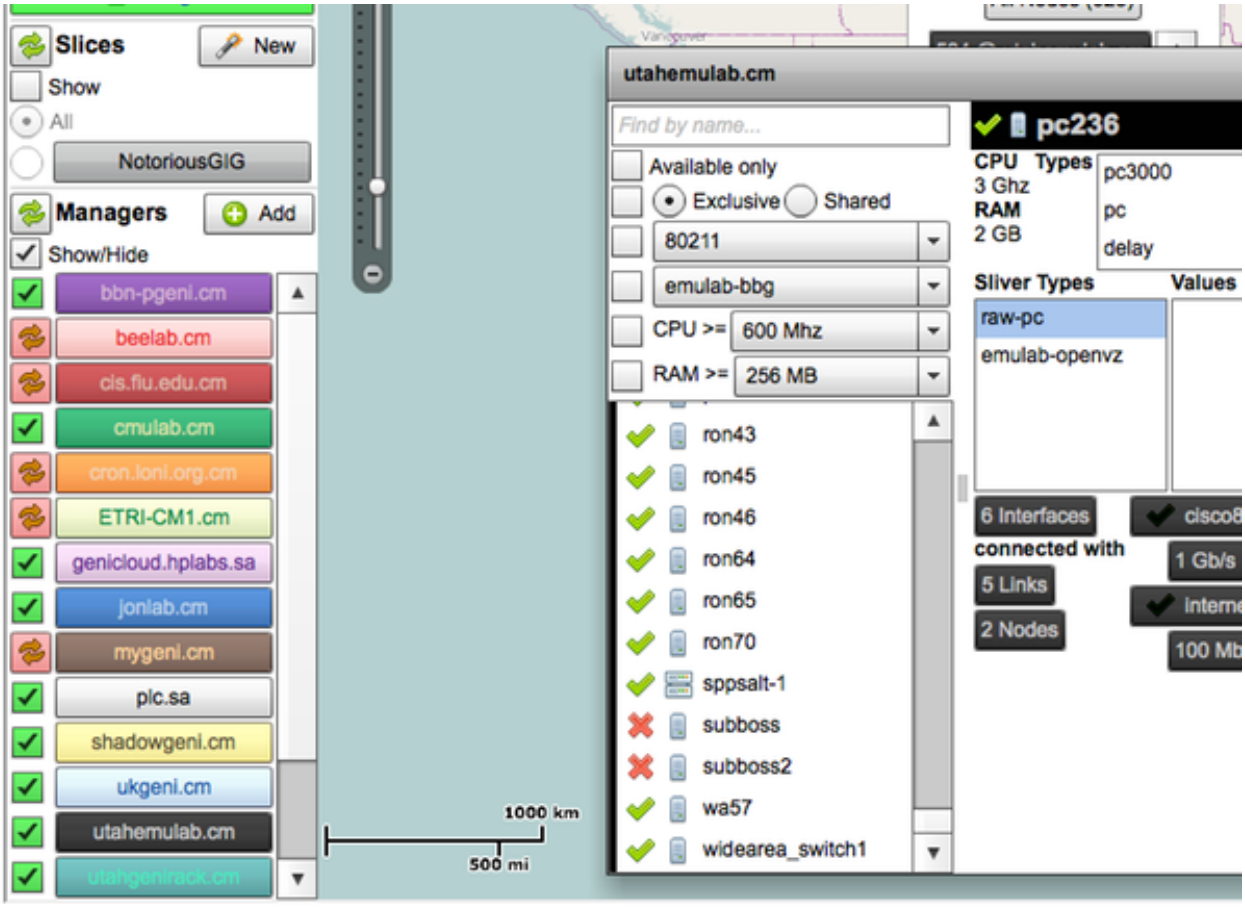


Our goal here is to find a PC hooked up to a gigabit interface, which will be labeled in the right pane as “1 Gb/s (...)”. Find one, like we did in the figure below.

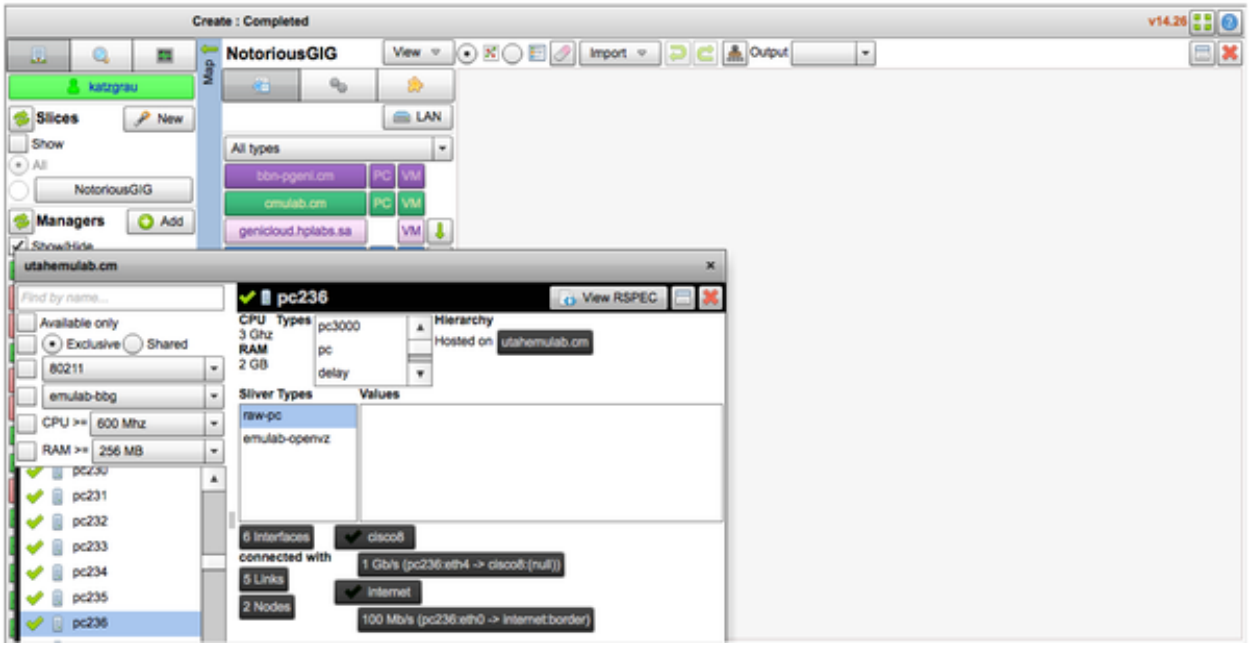


Note: It's possible, but not highly likely, that all gig resources are unavailable, in which case they'll have a red "x" in front of them instead of a checkmark. In this case, you may need to find another utahemulab.cm grouping on the map which has a gigabit-linked PC available.

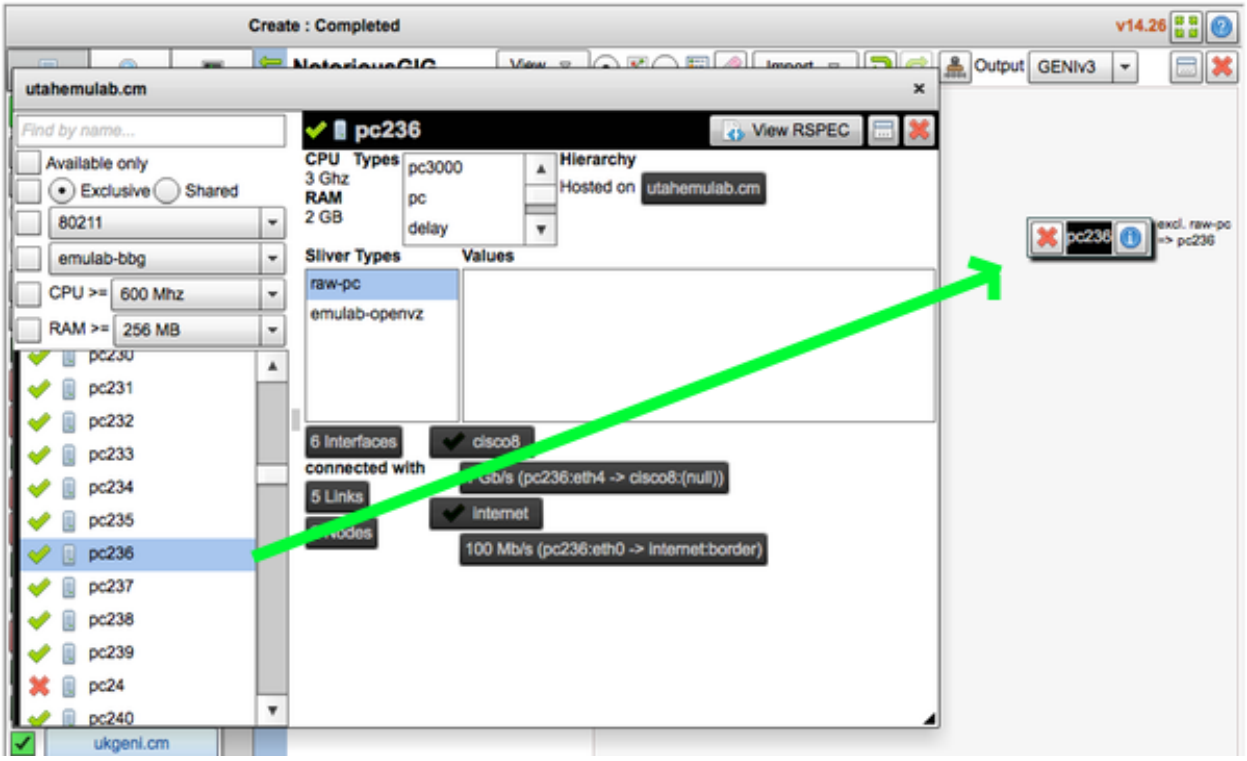
Now that you've found a machine with a gigabit link, drag that top-level window off to the side, and bring your slice pane back up by clicking on the name of your slice on the left side.



Click into your slice, and drag that resource window over to the left so you can see your pane:

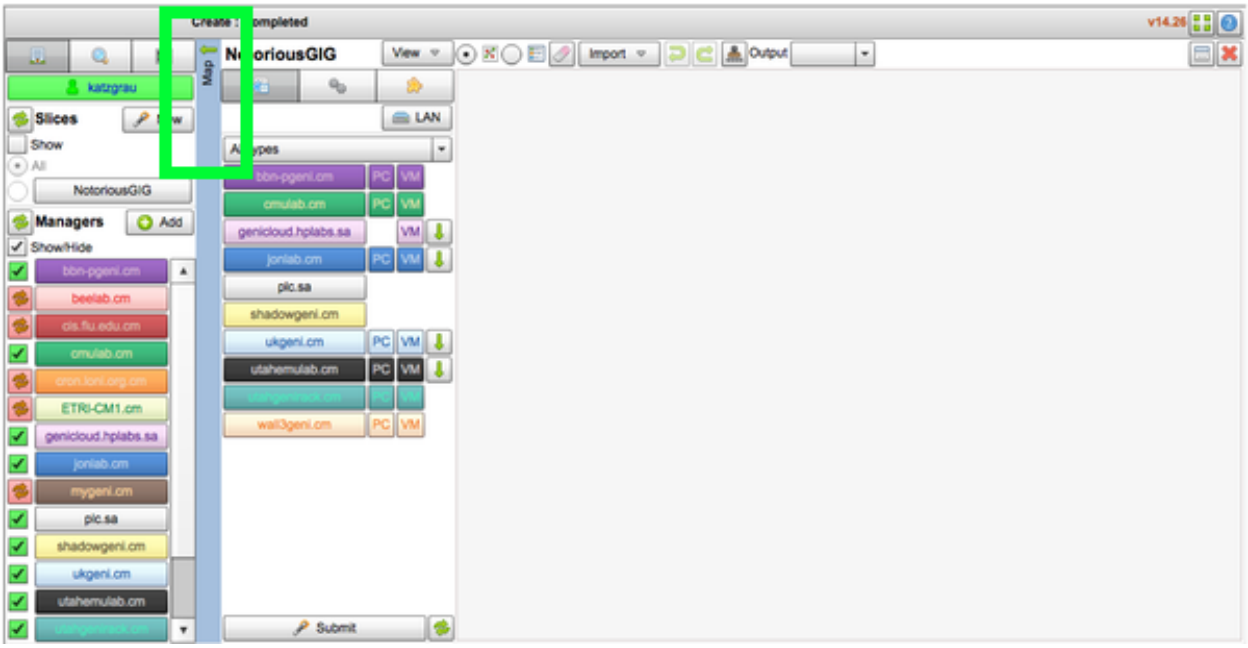


Drag the Gigabit PC we found over to the pane, like so:



Step 7. Find Another Machine With A Gigabit Link

We've now got a machine in Utah ready to go in our Gigabit setup. But now we need a machine on the *other* end of a network link. Back to the map!

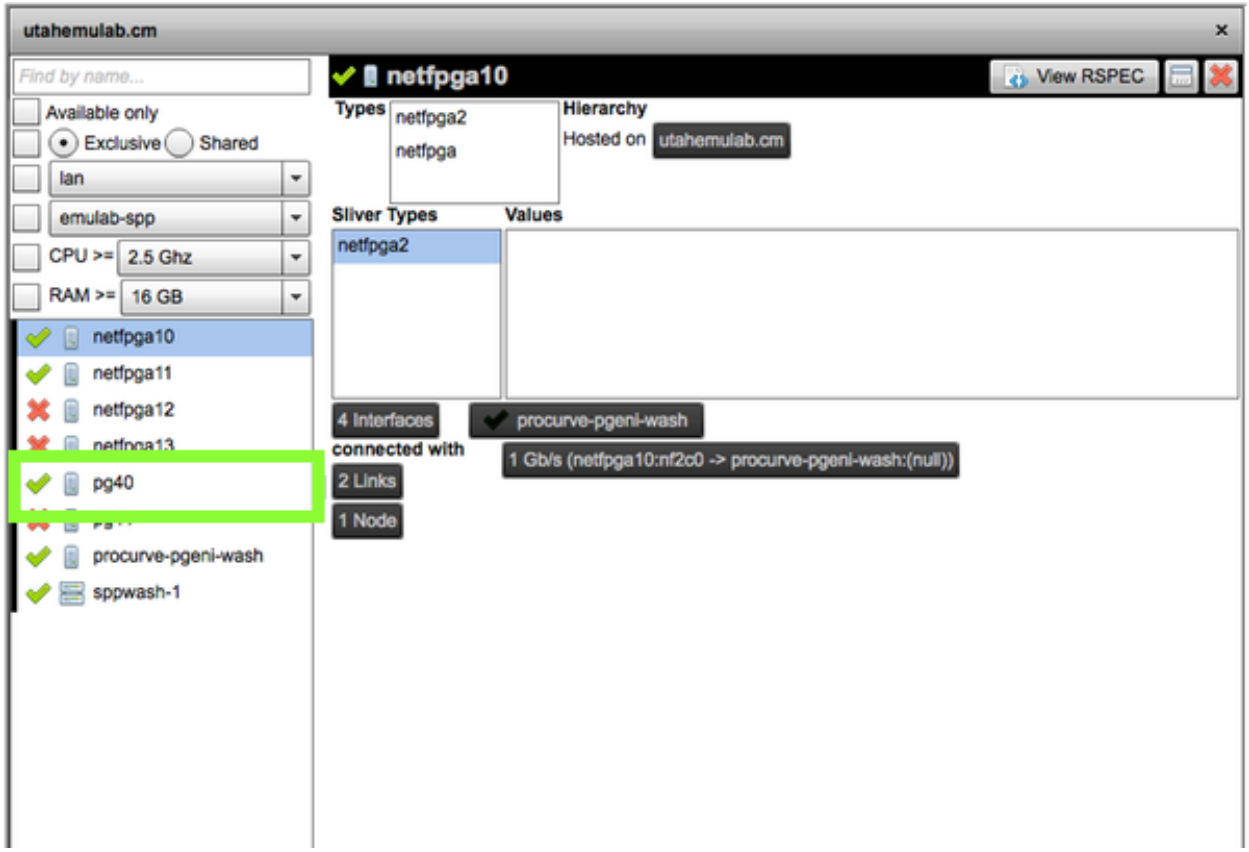


It was our stated intention at the beginning of this example to get a link going from Utah to Washington DC. As mentioned before, resources on GENI are limited, and it's possible that resources

aren't available in DC when you run this experiment. If that turns out to be the case, **pick another grouping of resources on the map that have a pink line running from Utah**.

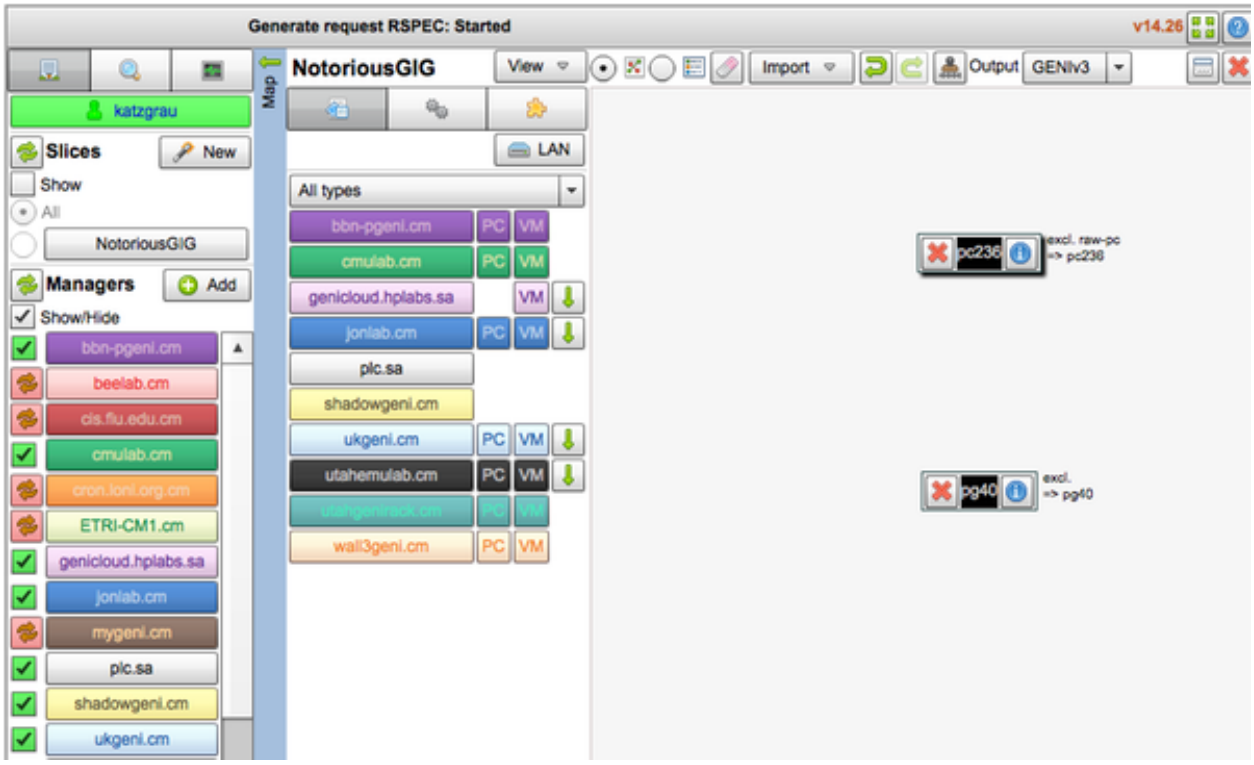
So let's look at Washington DC's grouping of resources, since the pink line representing a gigabit link runs directly from Utah.

We can see in the figure above that we have 8 resources for our use in utahemulab.cm. Click into that, and you'll likely see something similar to the figure below.



Remote computing resources are prefixed with "pg" as opposed to "pc" like we saw in Utah. pg40 in the figure above is hooked up to a gigabit link — just what we need! We'll take the one resource available in the window above, and drag it to our slice as before.

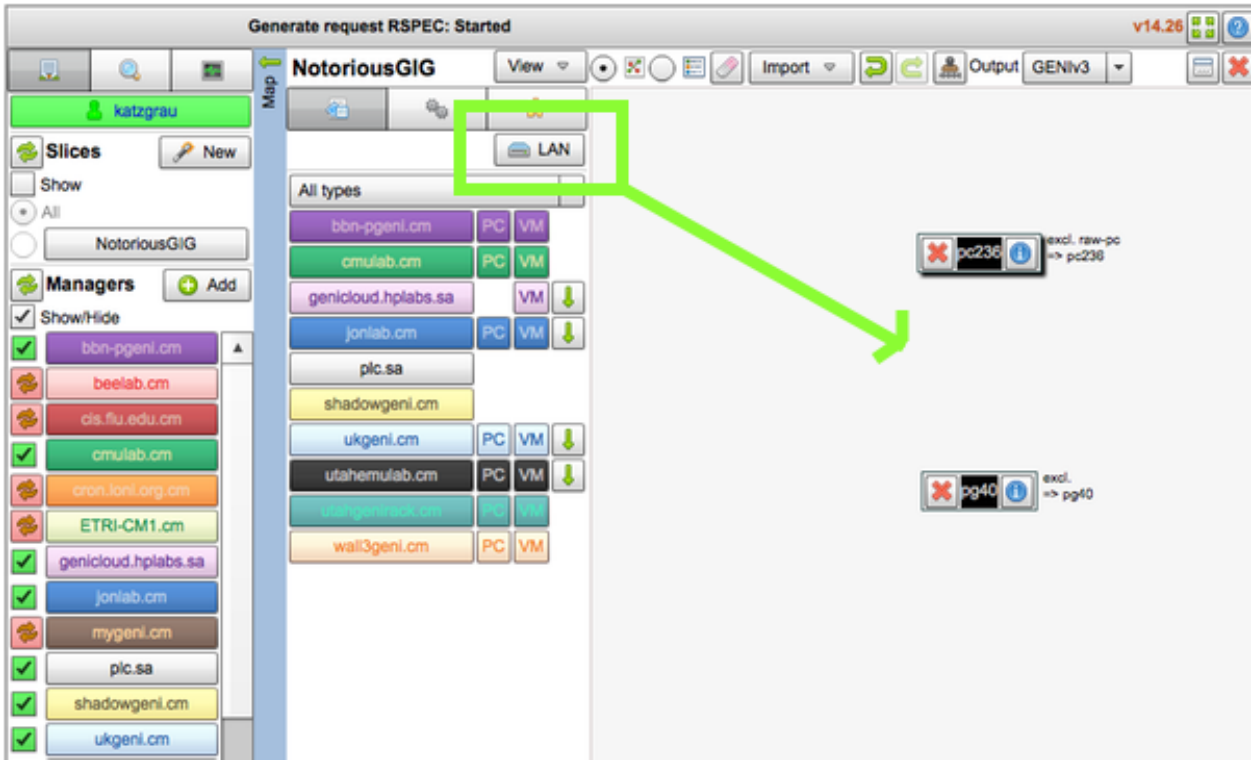
At this point, our Utah and DC machines should be sitting in our slice:



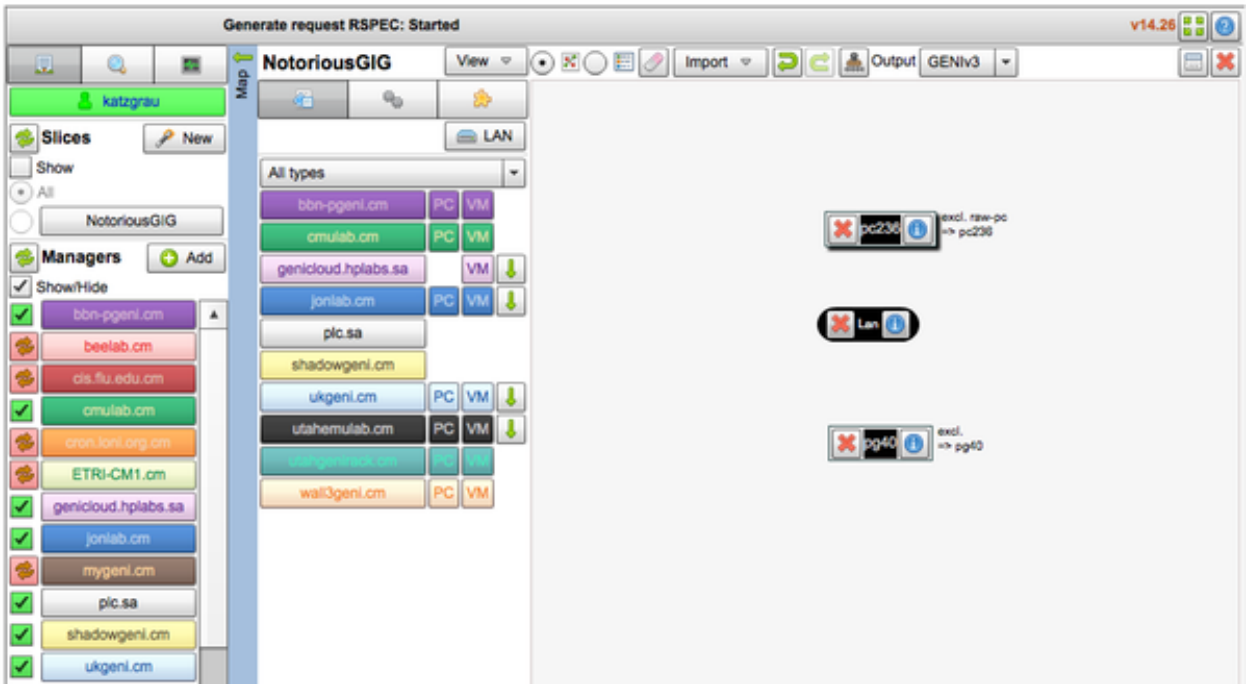
Step 8. Set Up a Gigabit Network!

We're now ready to hook up our network. Toward the upper left part of the slice pane, a LAN resource is available for us to drag into the slice.

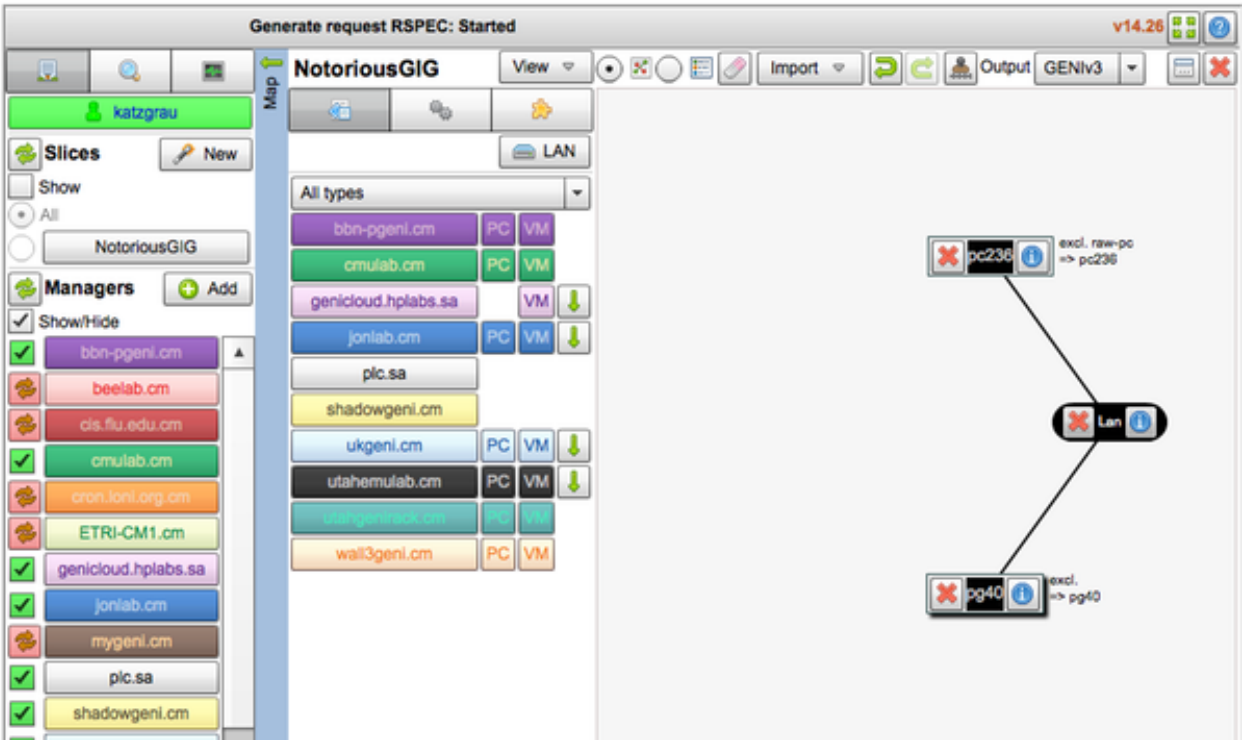
Before:



After:

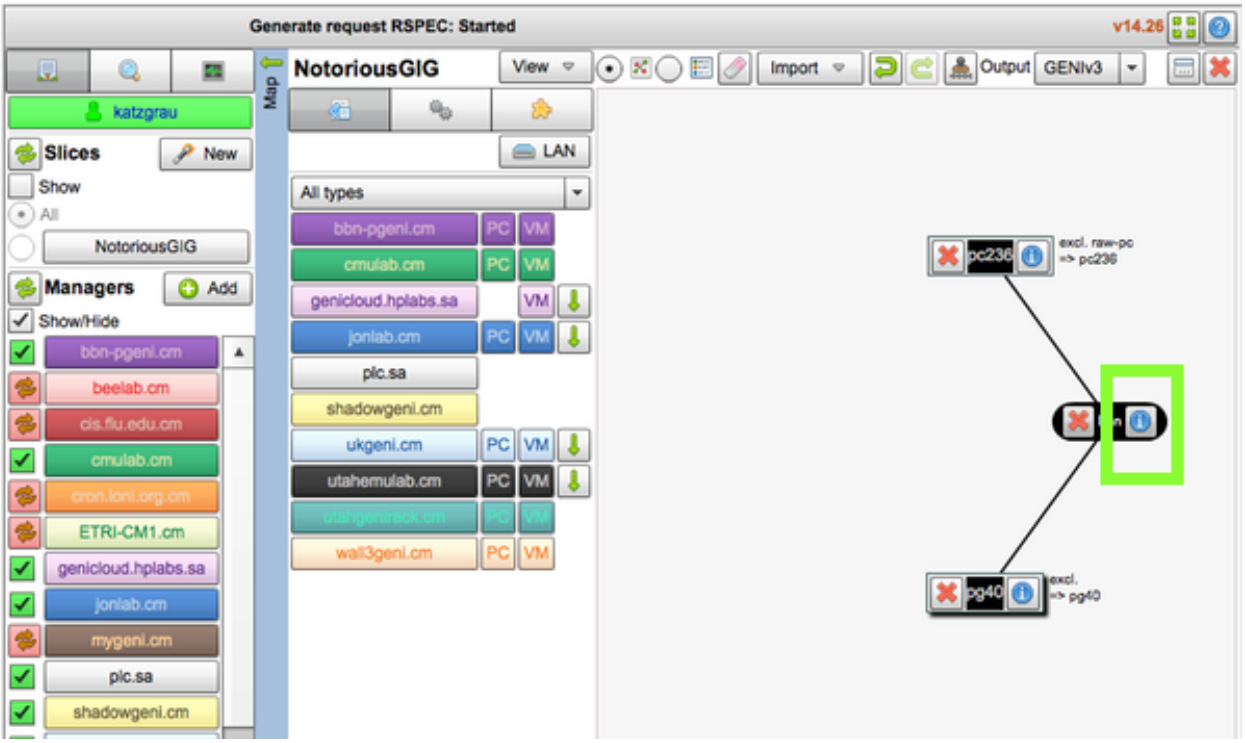


From here, drag a line **from** each machine to the lan. Your final lan should look like:

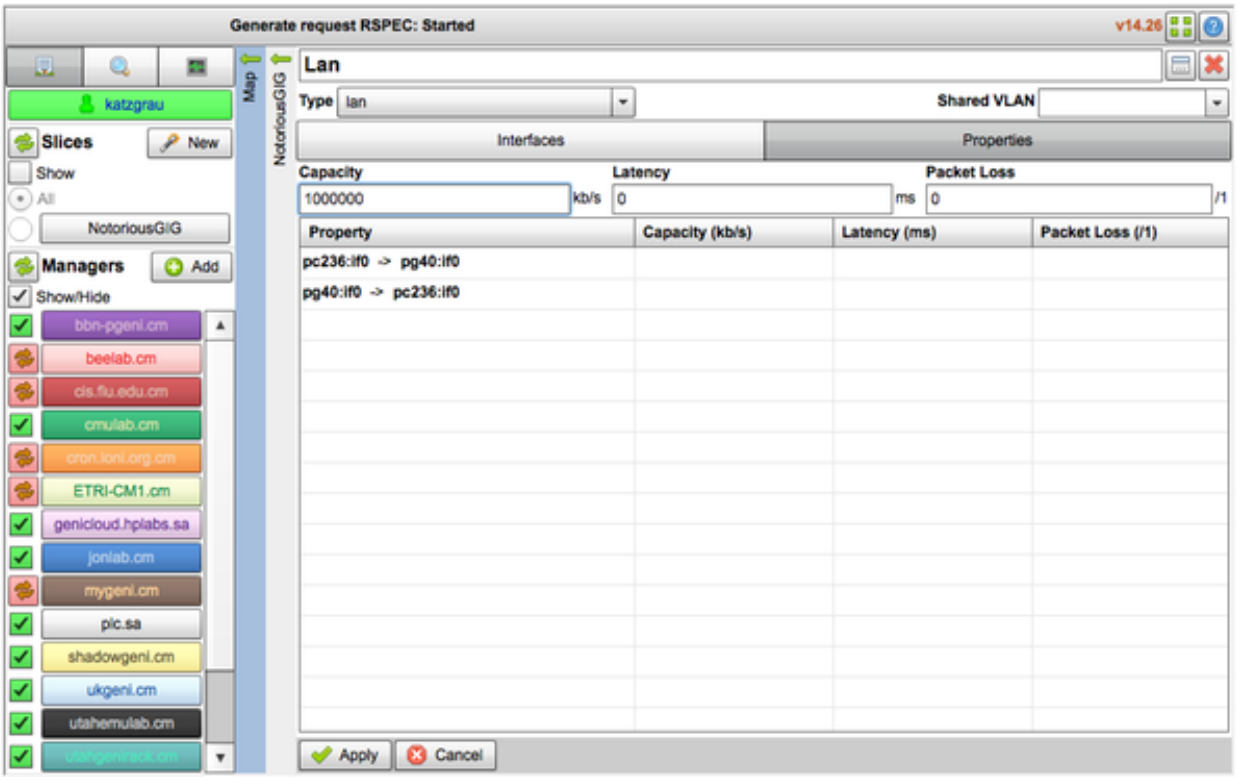


This is all very similar to how we set up our network in example 1. However, **this time we need to explicitly set the speed of the lan to be 1 Gb/s.**

Click the blue "i", or "information" icon on the lan. The information pane for the lan will appear. We want to change the *properties* of the lan, so we'll click on the "properties" tab:

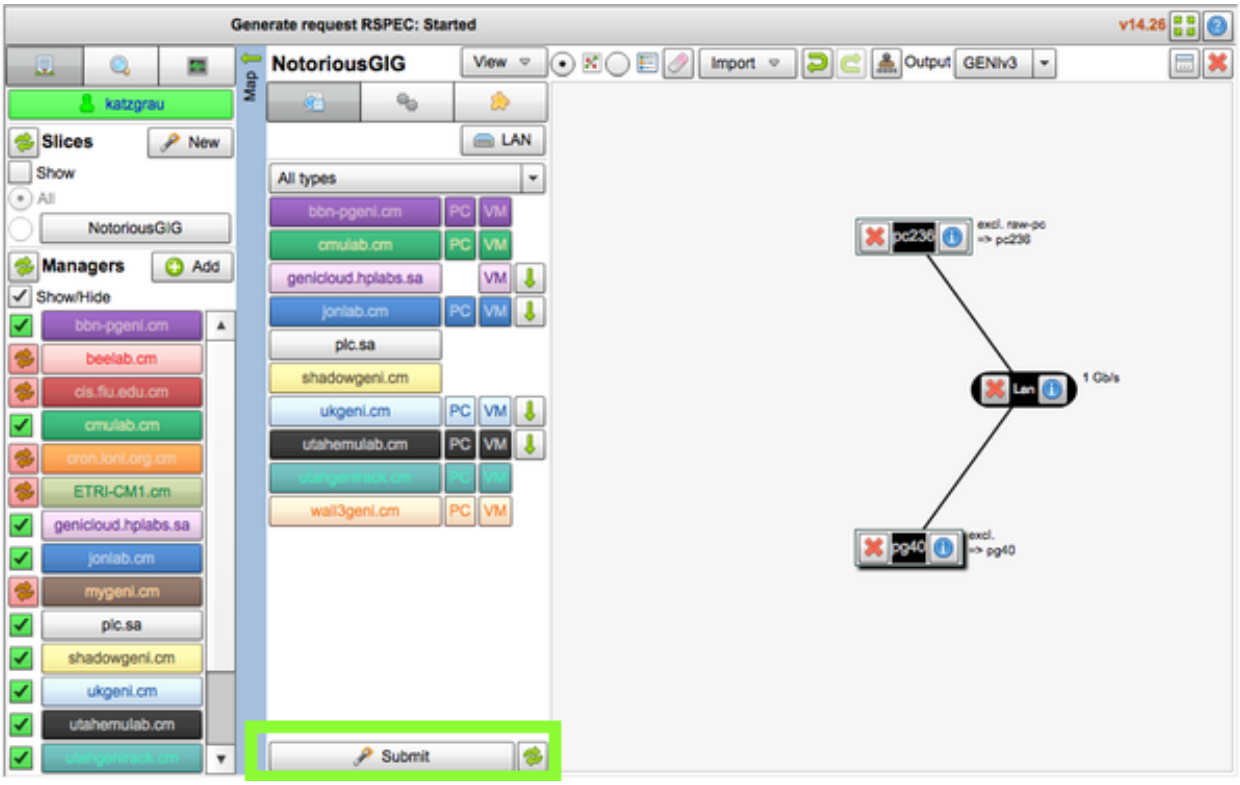


In the capacity field, enter "1000000". The unit of the capacity field is in kb/s, so we enter one million to specify 1 gb/s.

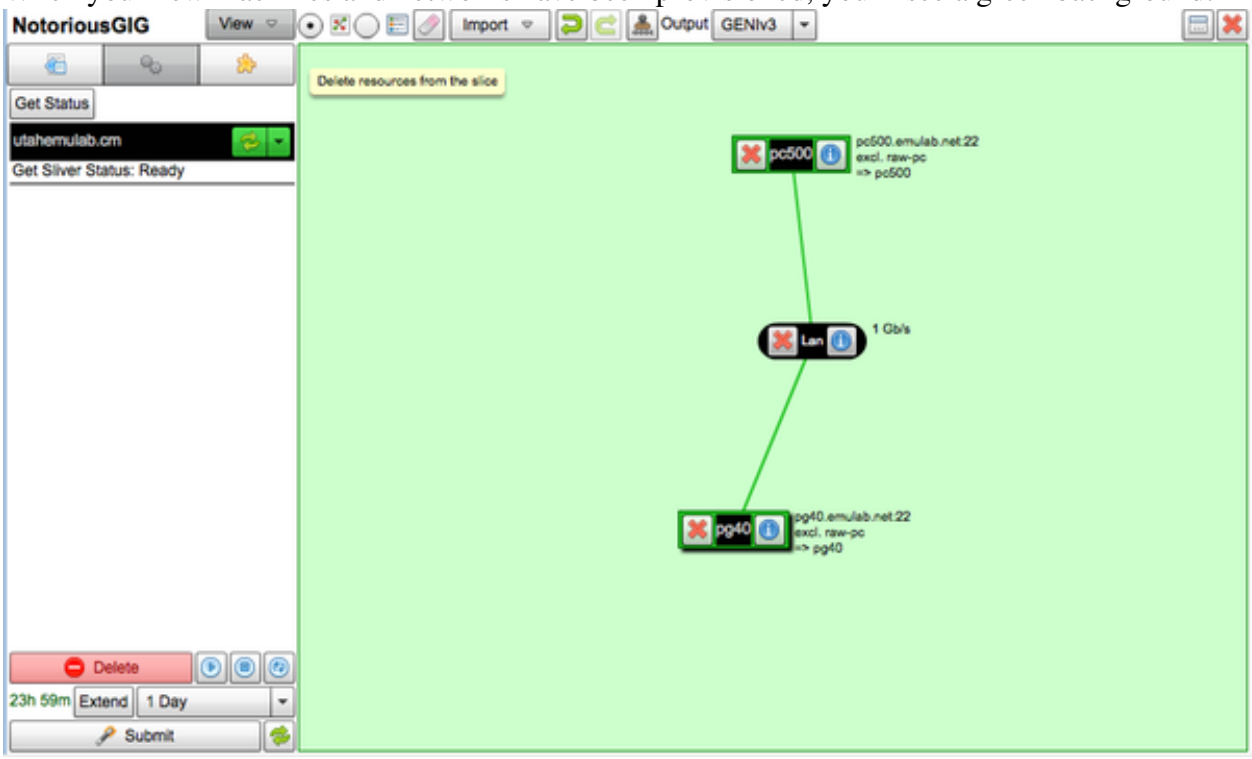


Click "Apply" at the bottom of the screen. If you've made it this far, congratulations! It's time to reserve your gigabit network! Click "Submit" at the bottom of the slice pane, and confirm.

Note: This can sometimes be a process that takes up to 15 minutes, and it occasionally fails or hangs. If it does fail, you may try again until it succeeds. If it continues to fail, it may be possible that your resource was granted to another experimenter while you were setting up your lan. You can click the "refresh" icon above the "Managers" list on the far left side of Flack to get the latest resource lists.



When your new machines and networks have been provisioned, you'll see a green background:



Author's Note: In the process of setting this up, I did in fact run into the provisioning failure I said might happen. You'll notice that I had to replace pc236, which was no longer available, with pc500 from the same Utah grouping.

You now have a full provisioned Gigabit lan!

Step 9. Log in to the machines

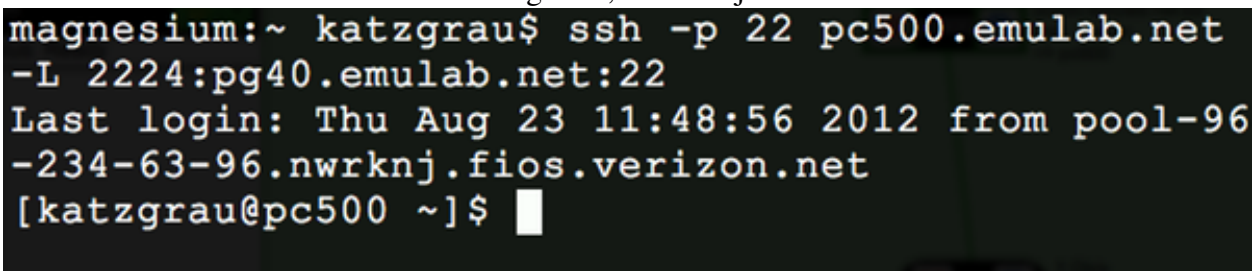
Let's now open two terminals windows - one for each box we've provisioned, and one for an SSH tunnel that we'll need. Let's start with logging into the box prefixed with **pc**. You should specify the address of the pc, and be sure to include the port number of needed with the `-p` flag.

Note: At the same time we're going to log into the first box, we'll set up an SSH tunnel that we can use to log into the other box. pg40 is behind a firewall, and we won't be able to access it directly.

Your SSH login command should look like:

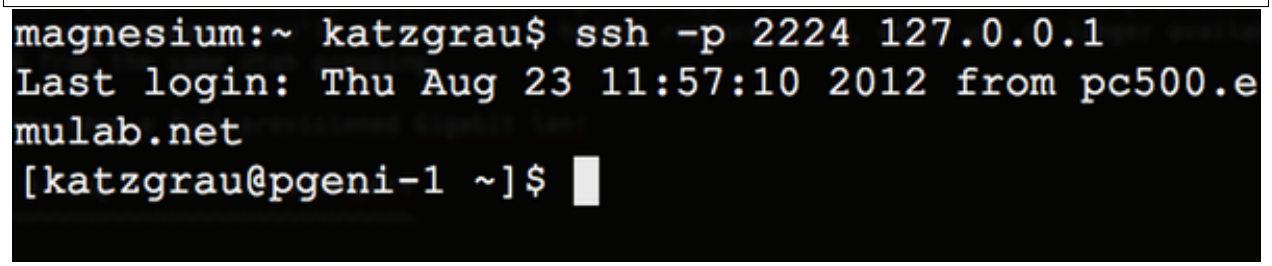
```
ssh -p [pc 1 port] [pc 1 host] -L 2224:[pc 2 host]:[pc 2 ssh port]
```

So with the machines we've been working with, we'll do just that:



Great. Now for logging into the second box (prefixed 'pg'). Since we have a tunnel open on port 2224 locally, we'll connect to that. You can type the same exact command, replacing [username] with your username:

```
$ ssh -p 2224 [username]@127.0.0.1
```

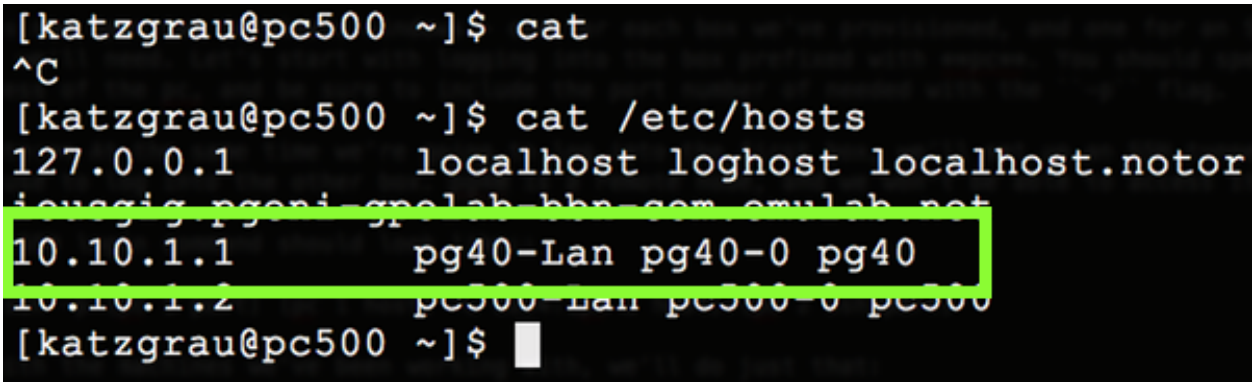


Step 10. Install iperf

Now, let's check our network connection. Run this in the terminal with the connection to the machine prefixed with **pc**:

```
$ cat /etc/hosts
```

You'll see something similar to:



The line highlighted in the image above is the alias to the remote host. It will be prefixed by **pg**. Let's keep that in mind.

Next, run this command in the window with the **pc** prefix. This will download and install iperf:

```
$ wget http://iperf.googlecode.com/files/iperf-3.0b4.tar.gz && tar -xf iperf-3.0b4.
```

You'll see something similar to:

```
[katzgrau@pc500 ~]$ wget http://iperf.googlecode.com/files/iperf-3.0b4.tar.gz && ta
--2012-08-23 14:31:23--  http://iperf.googlecode.com/files/iperf-3.0b4.tar.gz
Resolving iperf.googlecode.com... 173.194.79.82, 2607:f8b0:400e:c01::52
Connecting to iperf.googlecode.com|173.194.79.82|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 309547 (302K) [application/x-gzip]
Saving to: `iperf-3.0b4.tar.gz.1'

100%[========>] 309,547     1.31M/s   in 0.2s

2012-08-23 14:31:23 (1.31 MB/s) - `iperf-3.0b4.tar.gz.1' saved [309547/309547]

checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... gawk
[truncated for brevity]
make[2]: Leaving directory `/users/katzgrau/iperf-3.0b4'
make[1]: Leaving directory `/users/katzgrau/iperf-3.0b4'
```

Great! We've just installed iperf on our first box.

Open a new, temporary terminal window. Copy and paste this command:

```
wget http://iperf.googlecode.com/files/iperf-3.0b4.tar.gz && scp -P 2224 iperf-3.0b
```

This will download iperf, and send it to our box in DC, pg40.

Lastly, open the terminal tab of the box prefixed with **pg**. Run:

```
tar -xf iperf-3.0b4.tar.gz && cd iperf-3.0b4 && ./configure && make && sudo make ir
```

You now have iperf installed on both machines!

Step 11. Run iperf

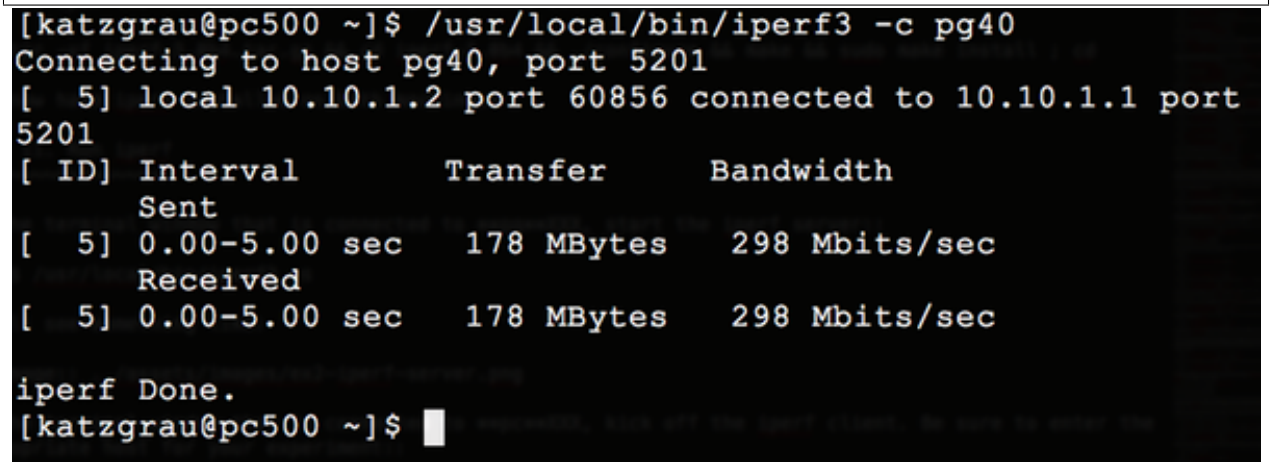
In the terminal window that is connected to pgXXX, start the iperf server:

```
$ /usr/local/bin/iperf3 -s
```

You'll see something similar to:

In the terminal window that is connected to pcXXX, kick off the iperf client. Be sure to enter the appropriate host for your experiment:

```
$ /usr/local/bin/iperf3 -c
```



Awesome! You've just gigabit data transfer from Utah to DC!

In the figure above, we can see that data was transferred at about 300 Mbits/s. That data transfer was over TCP, which is slower than UDP due to its mechanisms for avoiding data loss. We'd test with UDP, but at the time of this writing, the `-u` flag for iperf3 is broken!

Imagine the possibilities. You have a gigabit network to play on. What could you hack on, or create?

But remember: If you don't need the resources you provisioned right now, it's better that you delete your machines/slivers and release them into the pool of available resources for other hackers and experimenters.

Lastly, if you found yourself tripped up by any part of this tutorial, be sure to reach out to katzgrau@gmail.com[1] with feedback or questions.

3.3 Example 3 - Programming Networks with OpenFlow

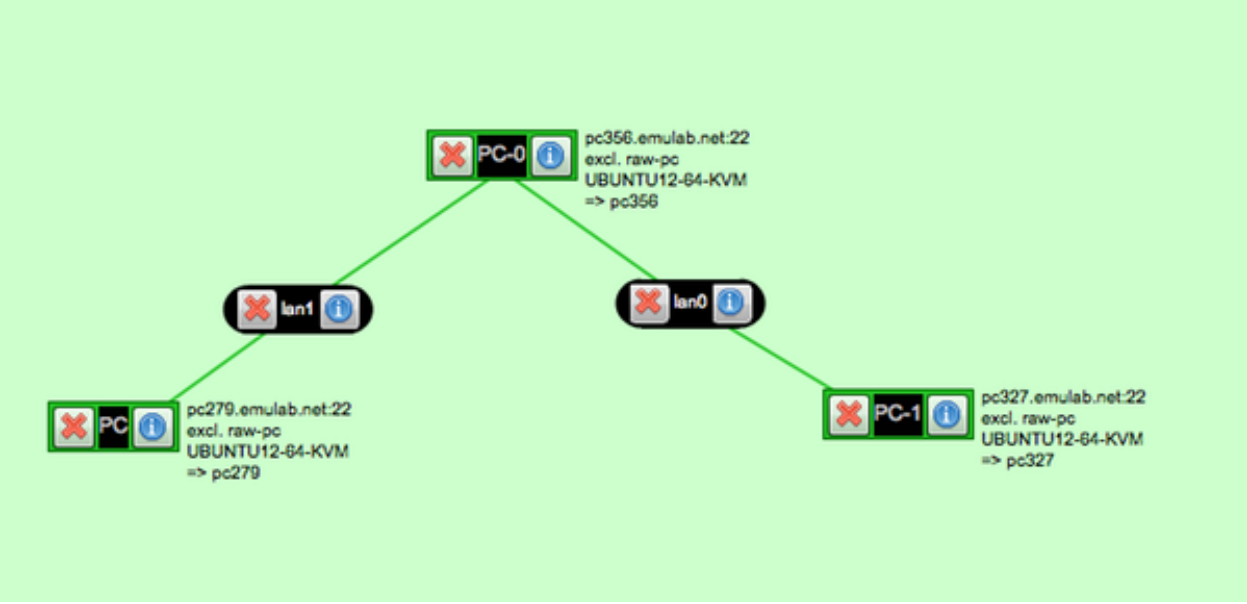
Note: This example references many topics which were discussed in example 1 and example 2, which act as sufficient prerequisites.

The full "what and why" of OpenFlow is discussed in the core documentation, but the reason you would want to use OpenFlow is to program how exactly your network behaves. For example, maybe you want to write a custom protocol and segment traffic in your network. Maybe you simply want to play around with a programmable switch.

OpenFlow is the leading architecture of Software Defined Networking (SDN), which is shaping up to be the future of deeply programmable networks.

In this experiment, we're going to get a software OpenFlow switch up and running on a single machine on GENI. Our node will act as an OpenFlow switch **and** it will be networked with two additional machines. We're going to send `ping` packets through our switch from one node to another, and then program the switch to drop any packets headed from one node to another.

Our virtual topology will look like this:



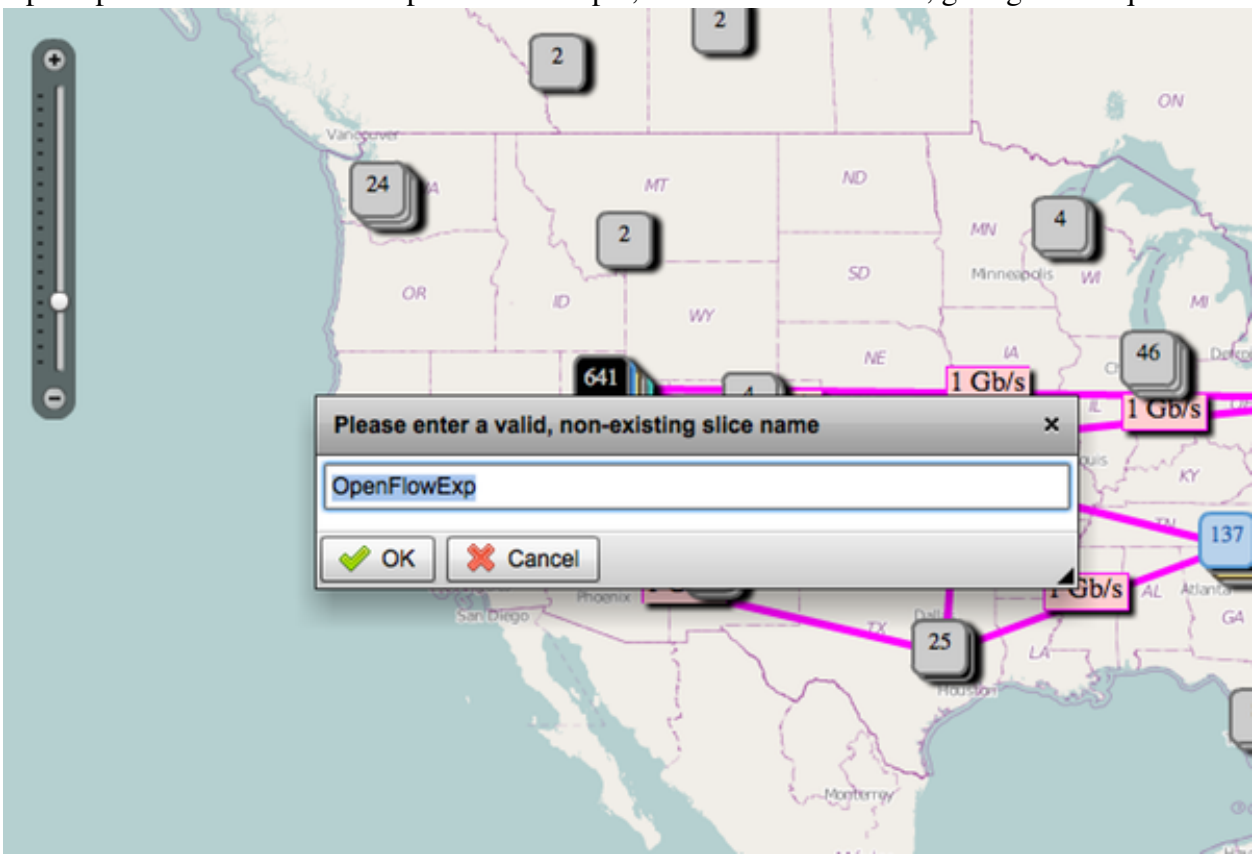
The node on the left will be the node we send packets to, and the node on the right will be the receiver of the packets. The node in the middle will be our software switch.

[1] katzgrau@gmail.com

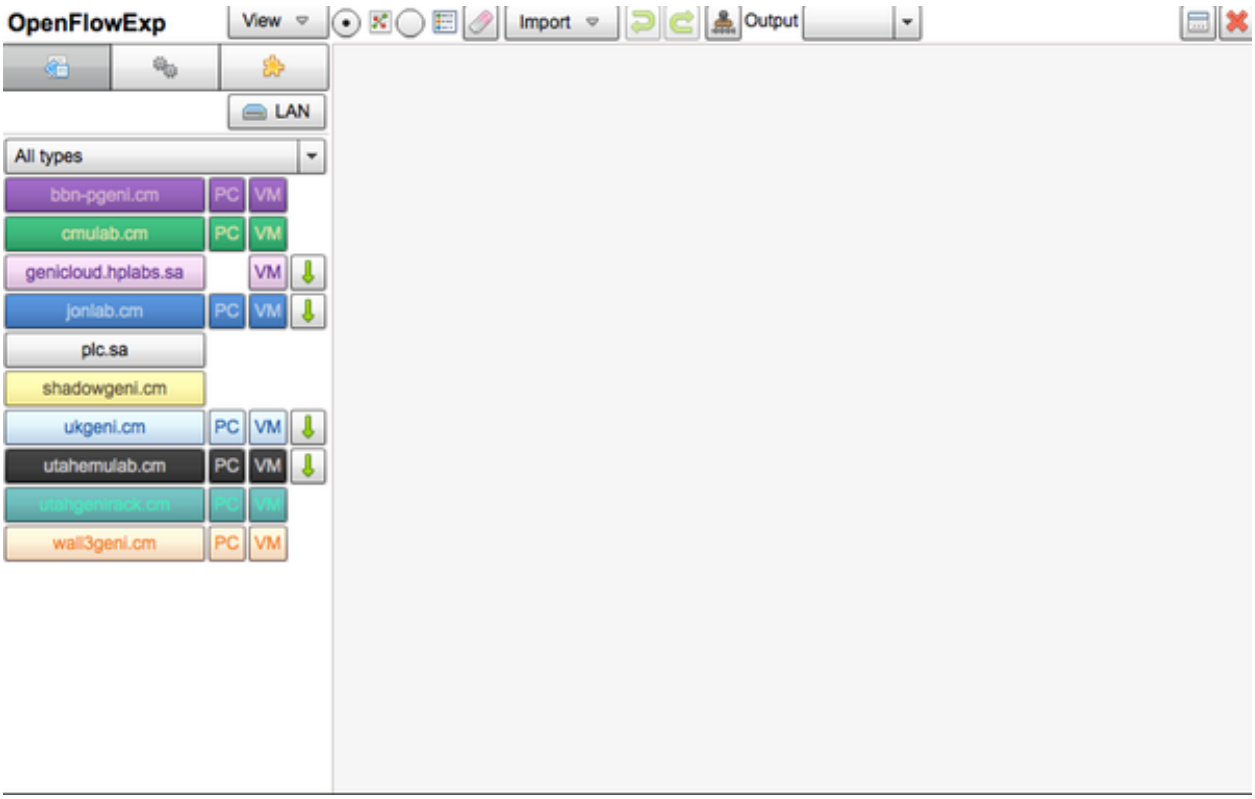
3.3.1 Tutorial

Step 1 - Set Up Our Network

Open up Flack as we did in the previous example, and create a new slice, giving it a unique name:

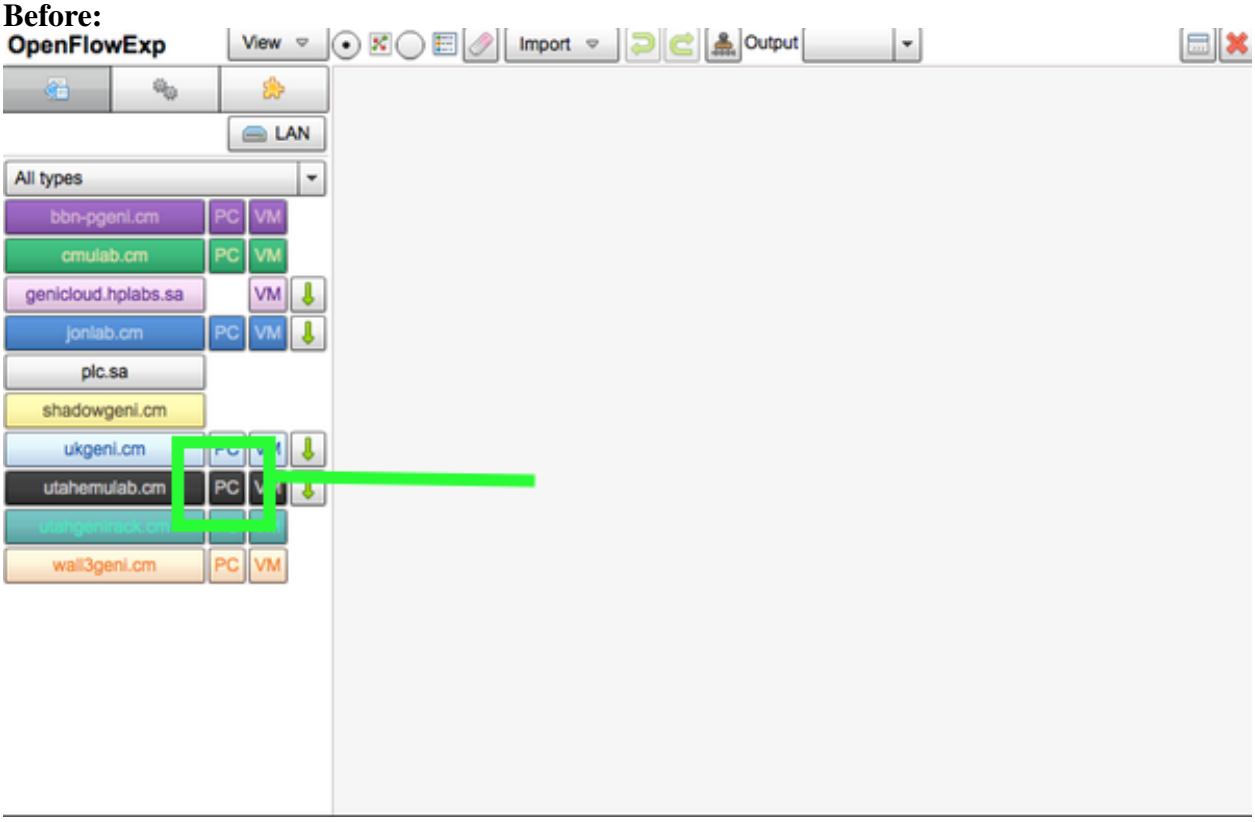


When you've submitted the new slice request, and the slice is created, click into it so you have a blank pane in front of you:

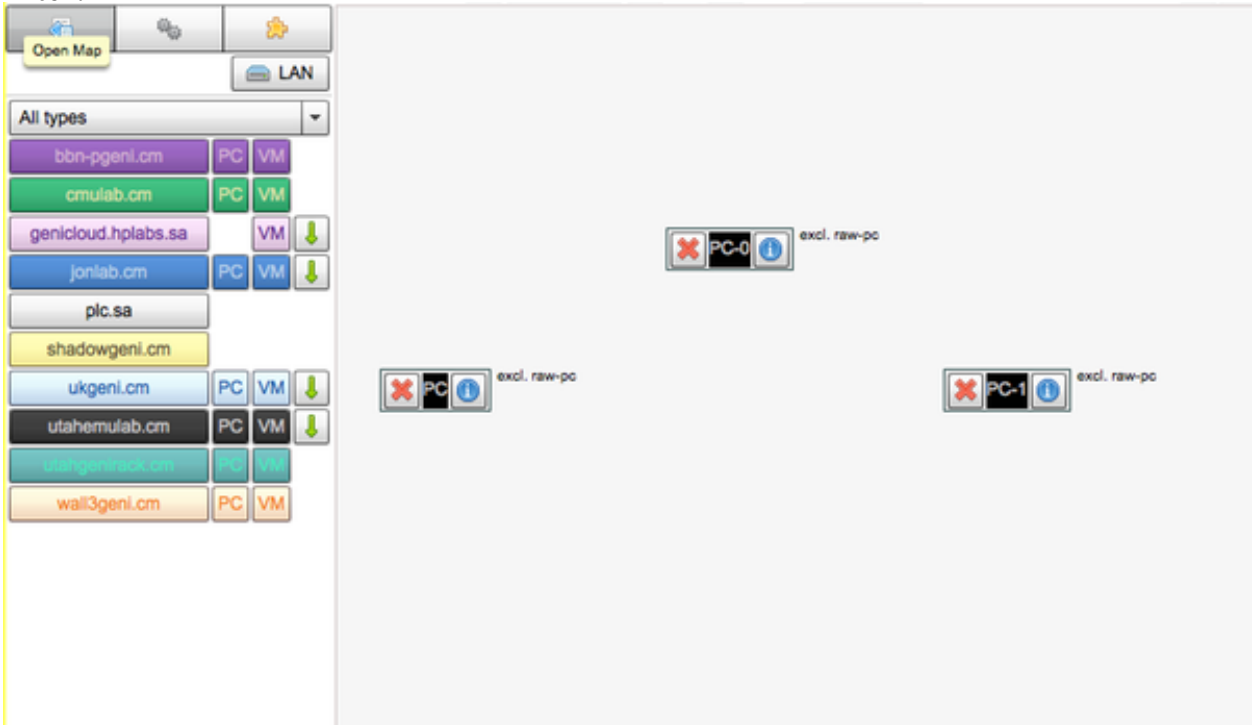


Drag three PC nodes from utahemulab.cm on the pane:

Before:



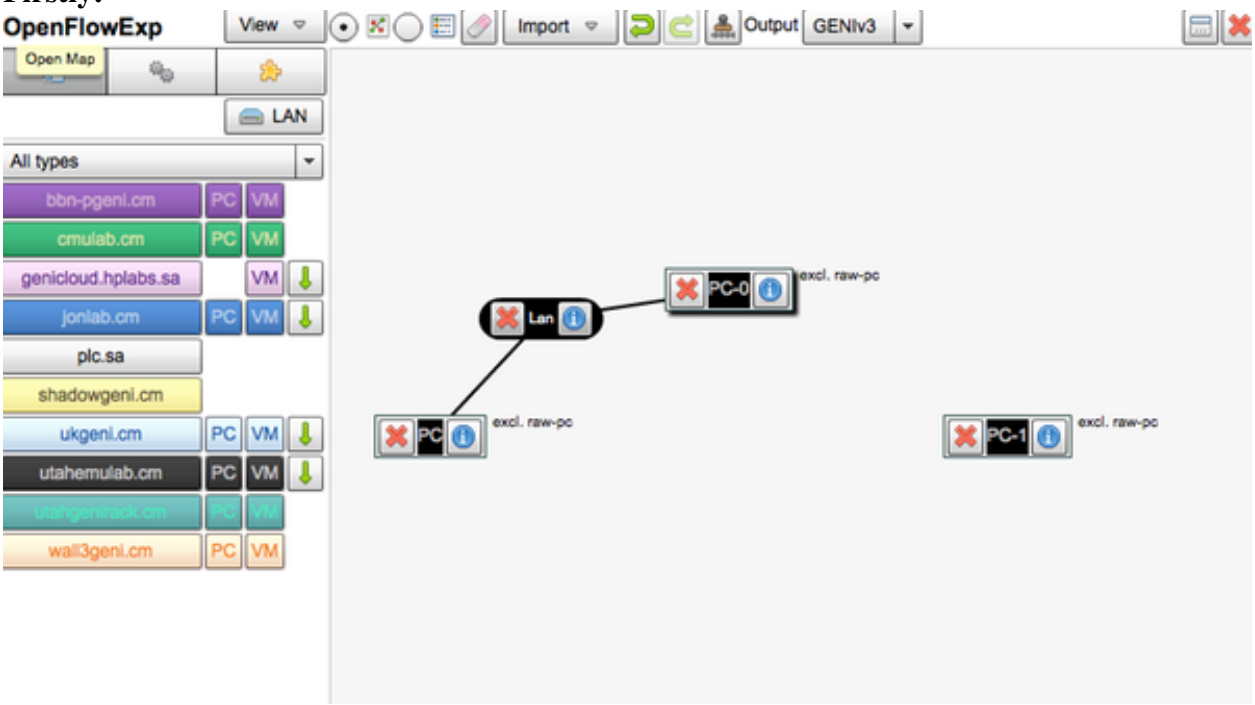
After:



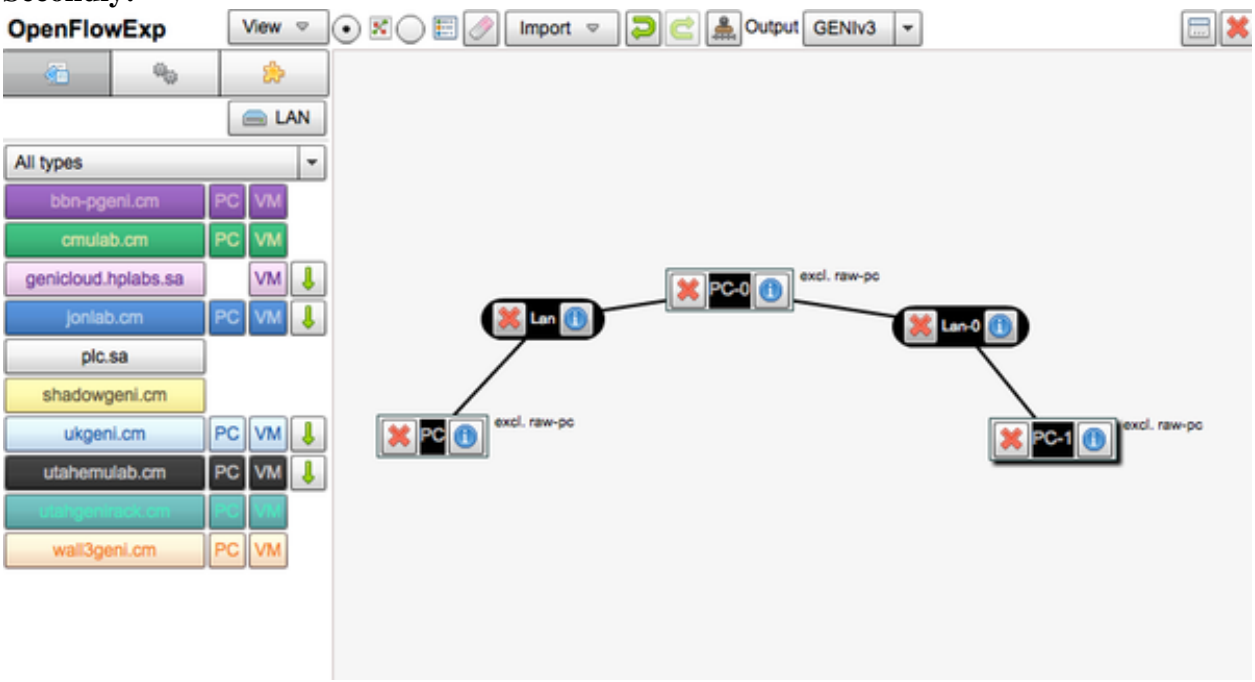
Important: The node in the middle will be our network switch.

The next thing we'll want to do is network our three machines together. For this, we'll drag two LAN components onto the pane. We want to have two distinct networks, so we'll network the switch and the node on the left first, and secondly, the node on the right and the switch.

Firstly:



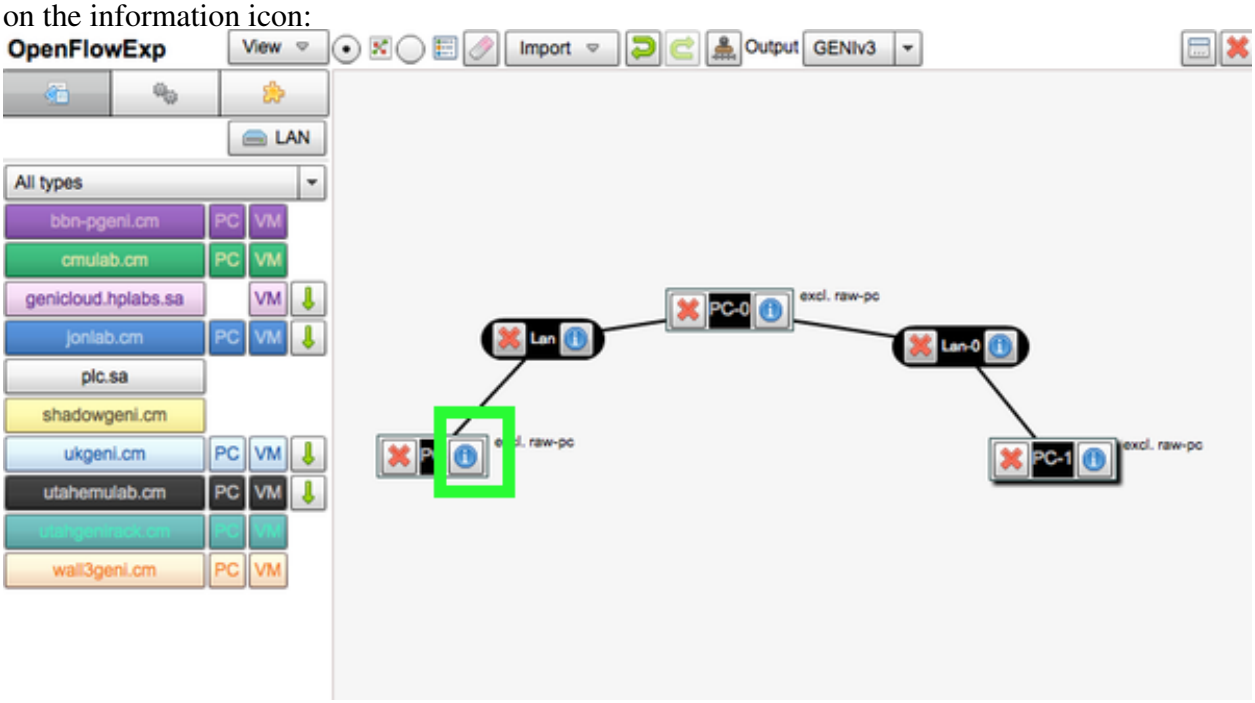
Secondly:



Now we're ready to configure out individual boxes.

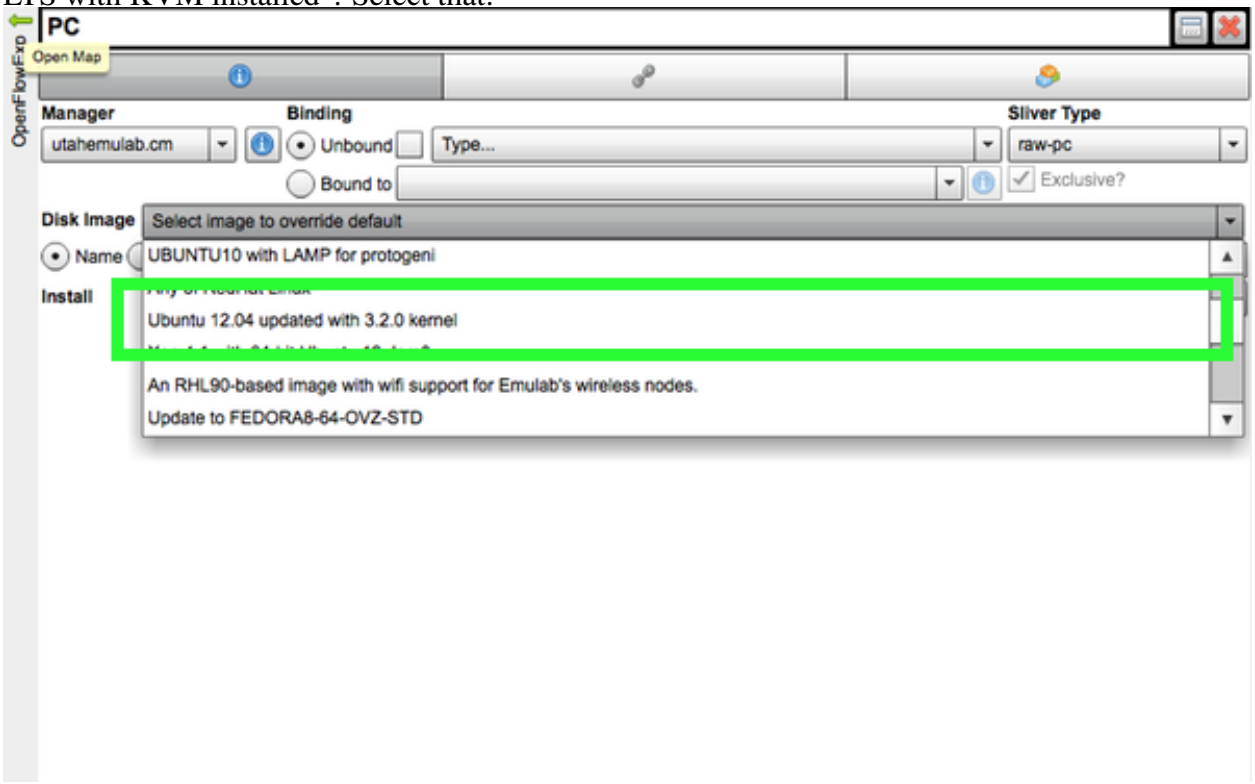
Step 2 - Specify Ubuntu As the Operating System and Submit

Let's configure all three nodes so that they have Ubuntu installed. For the node on the left, click on the information icon:

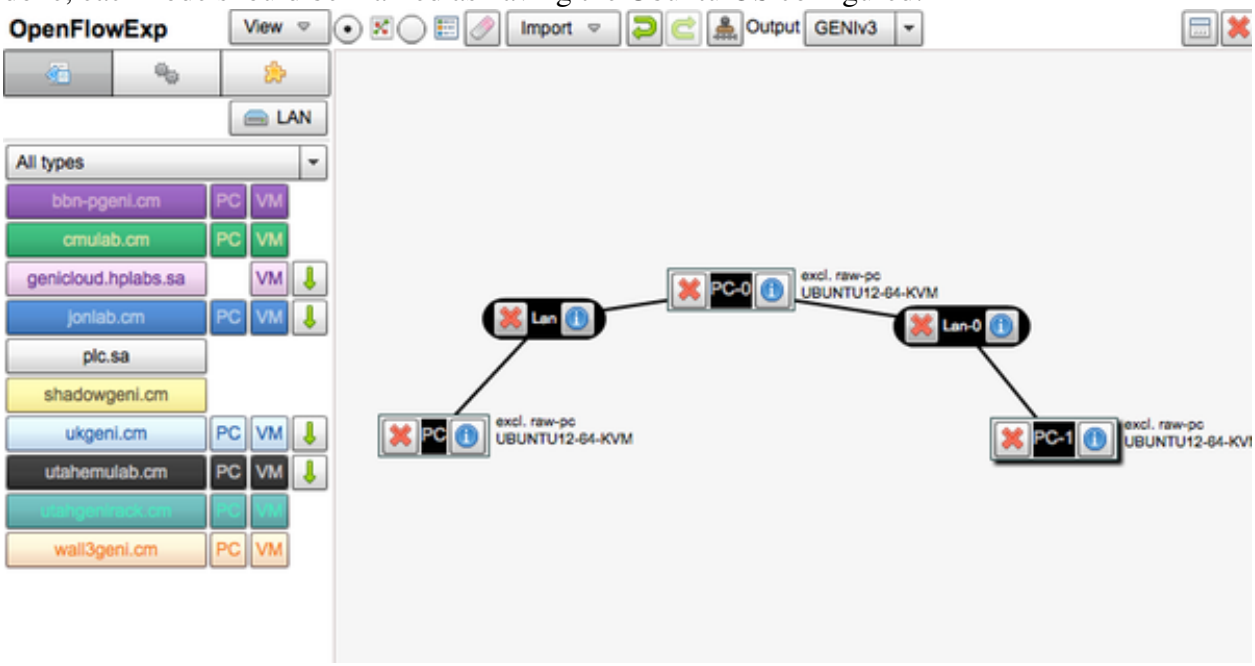


In the pane that appears, head to the "Disk Image" field, and find the option for "Ubuntu 12.04

LTS with KVM installed". Select that:



Click "Apply" at the bottom, and do the same for the other two nodes in our network. When you're done, each node should be marked as having the Ubuntu OS configured:



We've got our network set up! We're ready to submit our request to provision these nodes. Click "Submit" on the bottom of the screen. If you hit any errors while provisioning, you can submit again.

Step 3 - Install Open vSwitch

The node in the middle is going to be our switch. A this point, that node should have an address next to it, such as `pcXXX.emulab.net`, possibly with a port number. SSH into that back in a new terminal. In the example above, we'll do:

```
$ ssh pc209.emulab.net
$ sudo bash
```

We executed `sudo bash` since we're going to be doing a lot of heavy lifting that requires root. Now we need to install Open vSwitch. This is actually very easy, and only requires that we install a few packages with the `apt` package manager and do some housework. First, run these commands and confirm 'Yes' any prompts.:

```
$ apt-get update
$ apt-get install openvswitch-datapath-source bridge-utils
$ module-assistant auto-install openvswitch-datapath
$ apt-get install openvswitch-brcompat openvswitch-common
$ apt-get install curl traceroute
```

Now, we'll do a little editing. Execute:

```
$ nano /etc/default/openvswitch-switch
```

And change the line that says:

```
# BRCOMPAT=no
```

To:

```
# BRCOMPAT=yes
```

And finally, restart Open vSwitch:

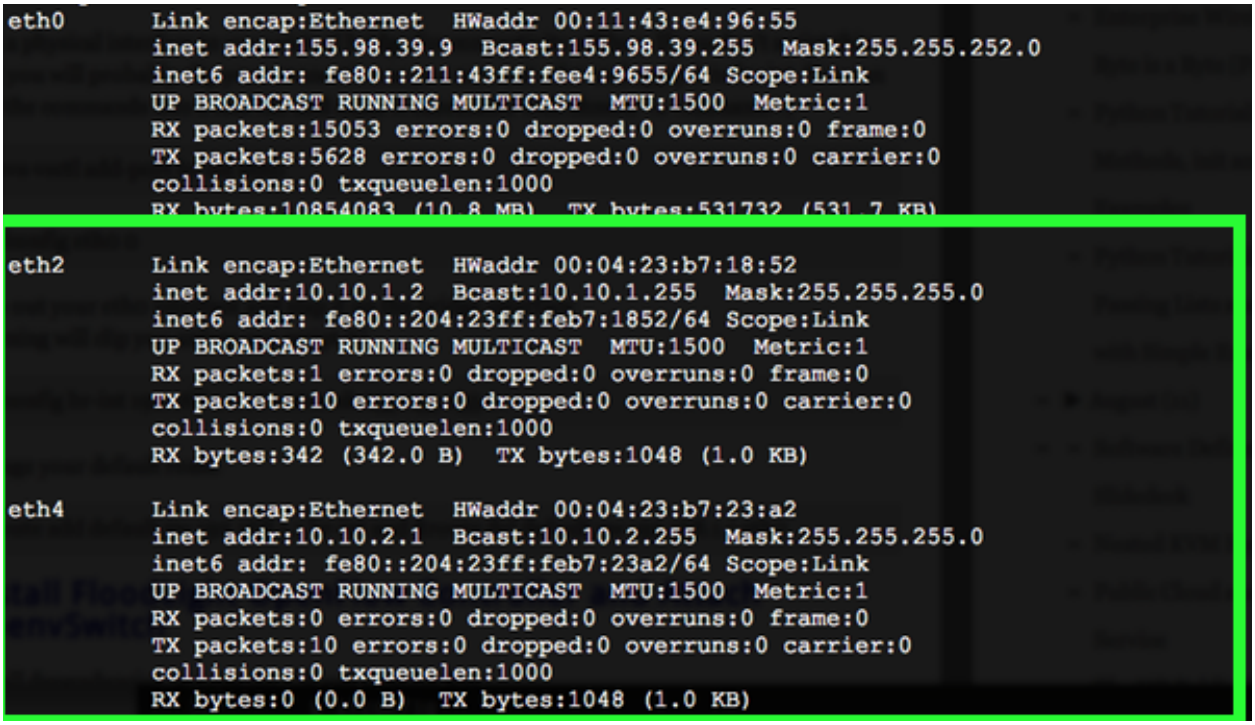
```
$ /etc/init.d/openvswitch-switch restart
```

Step 4 - Configure Open vSwitch to Use our Ethernet Interfaces

This next step truly depends on the box that you're working on. You need to know which interfaces you're going to be working with for the next steps. To do that, run:

```
$ ifconfig
```

In the output, you'll see a few different interfaces. The ones you need are prefixed by 'eth', and end with anything between 1 and 5. In the example, below, the two interfaces relevant to us are `eth2` and `eth4`. Take note of your interfaces, and apply them to the instructions below, along with the appropriate IP addresses.



So we have interfaces `eth2` and `eth4`. We want to configure Open vSwitch to use those interfaces as if they were the interfaces of a switch. We do this by setting up a virtual bridge interface. Type, or copy and paste these lines into your terminal:

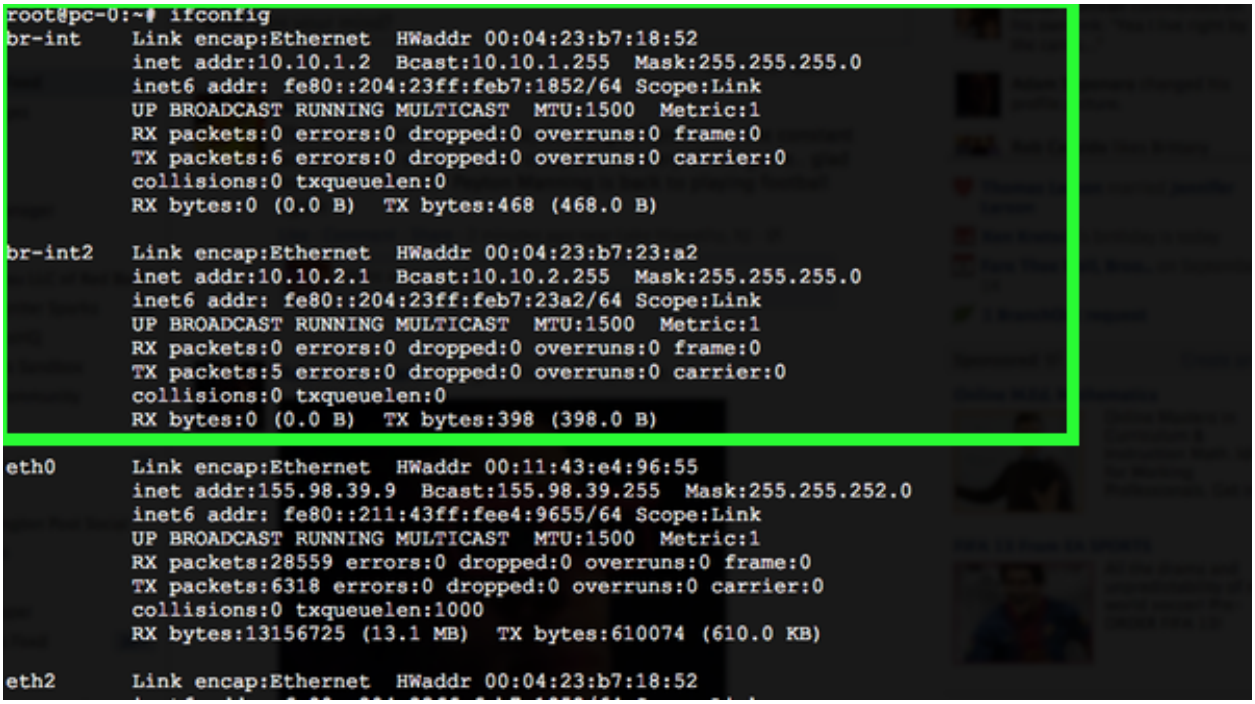
```
ovs-vsctl add-br br-int
ovs-vsctl add-port br-int eth2
ifconfig eth2 0
ifconfig br-int 10.10.1.2 netmask 255.255.255.0
route add -net 10.10.1.0 netmask 255.255.255.0 dev br-int

ovs-vsctl add-br br-int2
ovs-vsctl add-port br-int2 eth4
ifconfig eth4 0
ifconfig br-int2 10.10.2.1 netmask 255.255.255.0
route add -net 10.10.2.0 netmask 255.255.255.0 dev br-int2
```

At this point, running another:

```
$ ifconfig
```

Should show your new bridge interfaces:



Your switch is ready to go! Now you just need an OpenFlow controller.

Step 5 - Set Up Floodlight, an OpenFlow Controller

Your OpenFlow switch is controlled by an OpenFlow *controller*. An OpenFlow Controller is just a software package that interfaces with the switch via the OpenFlow API, and pushes routing rules.

In a situation where you had a hardware-based openflow switch, the controller would reside on a separate host. We're using a software based switch in the example, and for convenience, we're going to install the controller on the same box as our software switch.

Again, we'll install a package and run a series of commands:

```
$ apt-get install build-essential default-jdk ant python-dev uml-utilities git
$ git clone git://github.com/floodlight/floodlight.git
$ cd floodlight
$ make
$ java -jar target/floodlight.jar
```

We've just started our Floodlight controller. This controller will act as our interface to the switch. Since we just started the controller, we'll see log messages appearing on the screen:

```
20:51:40.748 [pool-3-thread-10] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:51:55.753 [pool-3-thread-14] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:52:10.759 [pool-3-thread-14] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:52:25.764 [pool-3-thread-4] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:52:40.770 [pool-3-thread-10] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:52:55.775 [pool-3-thread-14] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:53:10.781 [pool-3-thread-7] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:53:25.787 [pool-3-thread-3] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:53:40.793 [pool-3-thread-3] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:53:55.799 [pool-3-thread-14] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:54:10.804 [pool-3-thread-8] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:54:25.810 [pool-3-thread-11] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:54:40.816 [pool-3-thread-6] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:54:55.822 [pool-3-thread-1] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:55:10.827 [pool-3-thread-13] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:55:25.833 [pool-3-thread-1] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:55:40.839 [pool-3-thread-11] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:55:55.844 [pool-3-thread-14] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:56:10.850 [pool-3-thread-13] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:56:25.855 [pool-3-thread-11] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:56:40.861 [pool-3-thread-13] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:56:55.866 [pool-3-thread-13] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
20:57:10.872 [pool-3-thread-10] DEBUG n.f.l.internal.LinkDiscoveryManager - Sending LLDP out on all ports.
```

Step 6 - Test Our Switch

Because Floodlight is running in our terminal, we're going to open up another terminal window and ssh into our switch so we can keep working. We'll also tell the switch to listen to the Floodlight controller while we're at it:

```
$ ssh pc209.emulab.net
$ sudo bash
$ ovs-vsctl set-controller br-int tcp:127.0.0.1:6633
```

Additionally, we're going to open up a connection to the machine in our topology labeled **PC**. **Your address will be different. Check your topology pane for the correct address**:

```
$ ssh pc536.emulab.net
$ sudo bash
```

Now let's install a package of utilities we can use to test switch connectivity:

```
$ apt-get install uml-utilities traceroute
```

Next, From **PC**, let's ping PC-0 to make the boxes on both ends of the switch can talk to each other:

```
$ ping pc-1
```

This should yield something similar too:

```
root@pc:~# ping pc-0
PING PC-Lan (10.10.1.2) 56(84) bytes of data.
64 bytes from PC-Lan (10.10.1.2): icmp_req=1 ttl=64 time=0.012 ms
64 bytes from PC-Lan (10.10.1.2): icmp_req=2 ttl=64 time=0.011 ms
64 bytes from PC-Lan (10.10.1.2): icmp_req=3 ttl=64 time=0.014 ms
64 bytes from PC-Lan (10.10.1.2): icmp_req=4 ttl=64 time=0.014 ms
64 bytes from PC-Lan (10.10.1.2): icmp_req=5 ttl=64 time=0.014 ms
^C
--- PC-Lan ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3998ms
rtt min/avg/max/mdev = 0.011/0.013/0.014/0.001 ms
root@pc:~#
```

Great! Right now let's test the *route* our packets are taking from **PC** to **PC-0**. We want to make sure our packets are flowing through the switch we set up. We can double-check by running:

```
$ route -n
```

Which would yield:

```
root@pc:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.10.1.0       0.0.0.0         255.255.255.0   U     0      0        0 eth3
155.98.36.0     0.0.0.0         255.255.252.0   U     0      0        0 eth0
10.0.0.0        10.10.1.1       255.0.0.0       UG    0      0        0 eth3
0.0.0.0         155.98.36.1     0.0.0.0         UG    0      0        0 eth0
root@pc:~#
```

In the output above, you can see that our route from PC to PC-1 is indeed through the switch we set up. If you would like, you can also test with traceroute, in the manner of:

```
$ traceroute pc-1
```

Step 7 - See Floodlight's Web Interface

Floodlight has a simple web interface built in. It allows you to see which nodes are connected to the switch, and which rules are currently active. Access it by opening a browser window to:

```
http://pcXXX.emulab.net:8080/ui/index.html
```

Of course, replace your switch's hostname with the one above.

Additionally, take note of the DPID of the switch that you created:

```
.. image:: ../assets/images/ex3-floodlight-ui.png
```

Step 8 - Put Some Routing Rules!

Now for the *programmable networks* part of our openflow experiment. Let's get back in the terminal window that is connected to our switch. We'll now add a rule to our switch through Floodlight that will drop any packets coming from PC (10.10.2.2). Be sure to adjust your IPs below if they don't match up. Also, replace [DPID] with the DPID you found above:

```
$ curl -d '{"switch": "[DPID]", "name":"drop-flow", "src-ip": "10.10.2.2", "dst-ip'
```

Let's make sure that rule was pushed correctly, by issuing:

```
$ curl http://127.0.0.1:8080/wm/staticflowentrypusher/list/[DPID]/json | json_pp -t
```

And you'll see something similar to:

```
root@pc-1:~# curl http://127.0.0.1:8080/wm/staticflowentrypusher/list/00:00:00:24:e8:79:29:1a/json | json_pp -t dumper
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   670    0   670    0     0  69791      0 --:--:-- --:--:-- --:--:-- 74444
$VAR1 = {
          '00:00:00:24:e8:79:29:1a' => {
                                         'drop-flow' => {
                                                          'priority' => 32767,
                                                          'actions' => undef,
                                                          'flags' => 0,
                                                          'version' => 1,
                                                          'bufferId' => -1,
                                                          'match' => {
                                                                       'dataLayerVirtualLanPriorityCodePoint' => 0,
                                                                       'wildcards' => 3145983,
                                                                       'networkDestinationMaskLen' => 32,
                                                                       'networkProtocol' => 0,
                                                                       'transportSource' => 0,
                                                                       'networkSourceMaskLen' => 32,
                                                                       'dataLayerSource' => '00:00:00:00:00:00',
                                                                       'dataLayerType' => '0x0000',
                                                                       'networkTypeOfService' => 0,
                                                                       'dataLayerDestination' => '00:00:00:00:00:00',
                                                                       'inputPort' => 0,
                                                                       'networkDestination' => '10.10.2.2',
                                                                       'transportDestination' => 0,
                                                                       'networkSource' => '10.10.1.2',
                                                                       'dataLayerVirtualLan' => -1
                                                                     },
                                                          'cookie' => '45035997289868789',
                                                          'lengthU' => 72,
                                                          'length' => 72,
                                                          'outPort' => -1,
                                                          'xid' => 0,
                                                          'type' => 'FLOW_MOD',
                                                          'hardTimeout' => 0,
                                                          'idleTimeout' => 0,
                                                          'command' => 0
                                                        }
                                       }
        };
root@pc-1:~#
```

Step 9 - Retest That Switch

Go back to step 6 where we were on **PC**, and try to ping PC-1. How did that work? Did it fail?

Congratulations! You programmed your first switch with a simple JSON packet. If you want to remove the rule and try once more, you'll find that your packets are free to flow again:

```
$ curl -X DELETE -d '{"name":"drop-flow"}' http://127.0.0.1:8080/wm/staticflowentry
```

Step 10 - Think About The Applications

For the purposes of this example, we manually pushed all the required JSON rules to demonstrate the programmability of the switch.

But imagine that instead of pushing the rules manually, you wrote a web application to push those rules instead. Since the switch is now controllable with something like JSON, you can truly define the behavior of your network *with software*. That's the essence of software-defined networking!

3.3.2 Additional Resources

This tutorial was adapted from this blog post for GENI: http://networkstatic.net/2012/06/openflow-openvswitch-lab/

For more on the Floodlight REST API, and what methods are available: http://www.openflowhub.org/display/floodlightcontroller/REST+API

CHAPTER 4

GENI Tools and Services (A Rundown)

Contents

4.1 Flack

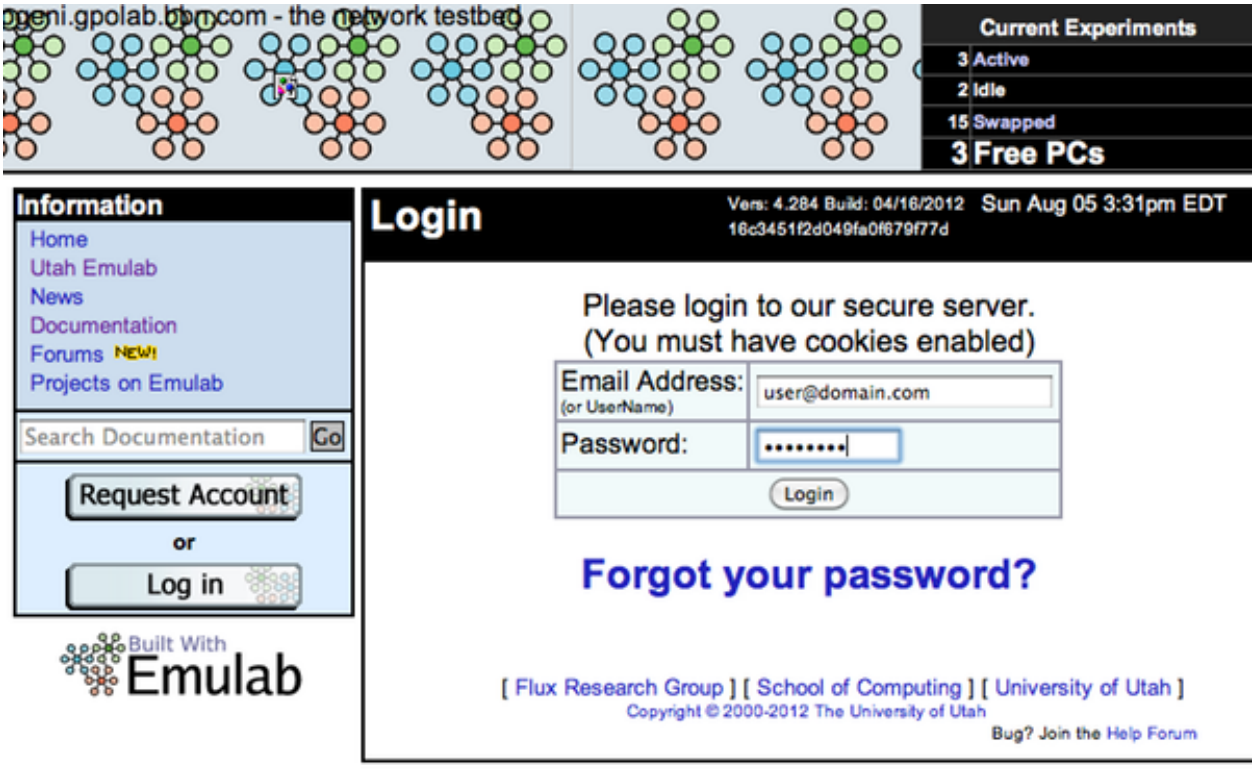
Flack is a visual tool that you can use to provision and network machines on GENI. It's similar to omni in terms of capability, however omni is a command-line tool that provides a wider range of abilities.

Because of it's low barrier to entry, Flack is the preferred tool for getting developers up and running quickly, and the example GENI projects use it exclusively.

This document will go over the different features of Flack that may not have been touched in the examples.

4.1.1 Logging In to Flack

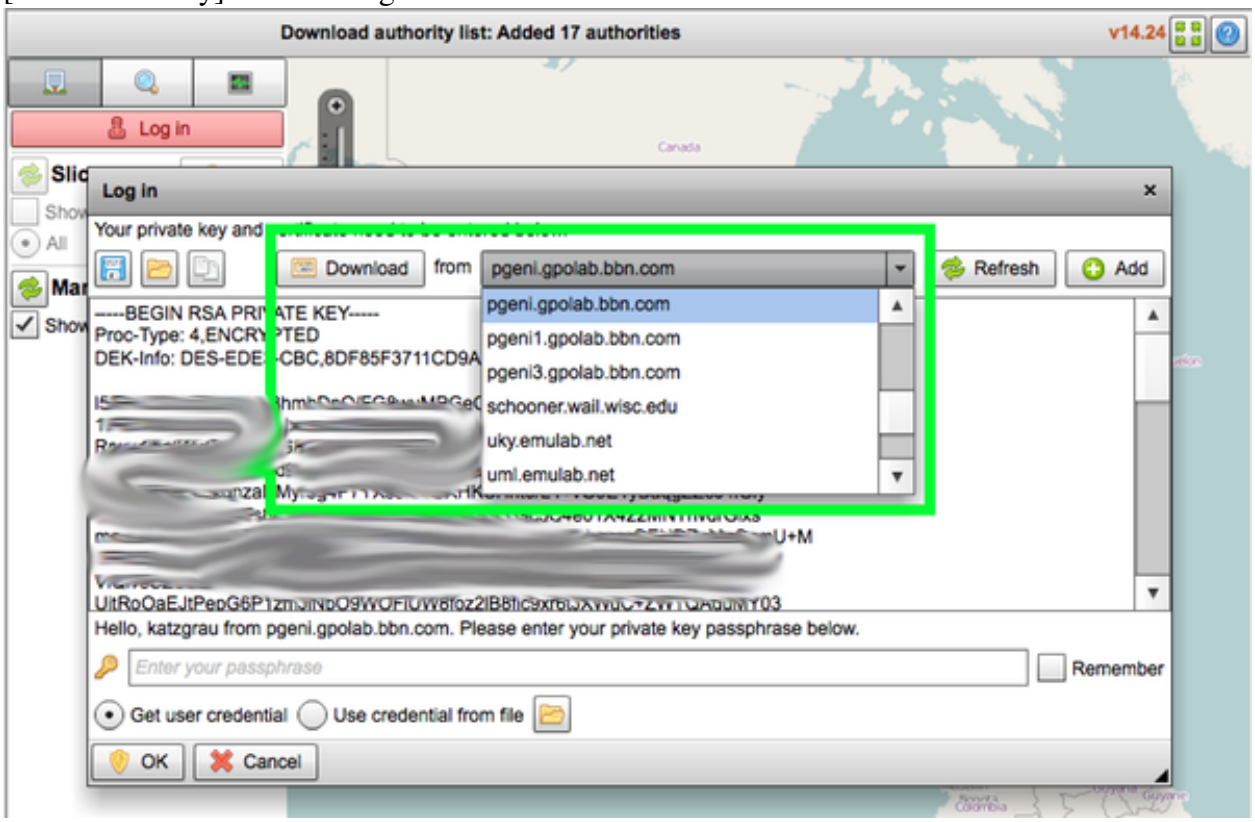
The easiest way to authenticate with Flack is to simply be logged in to your clearinghouse's website in one window or tab, and to have Flack open in another window or tab.



Open Flack in a new tab or browser window. You should see Flack initialize in the window, as in the figure below:

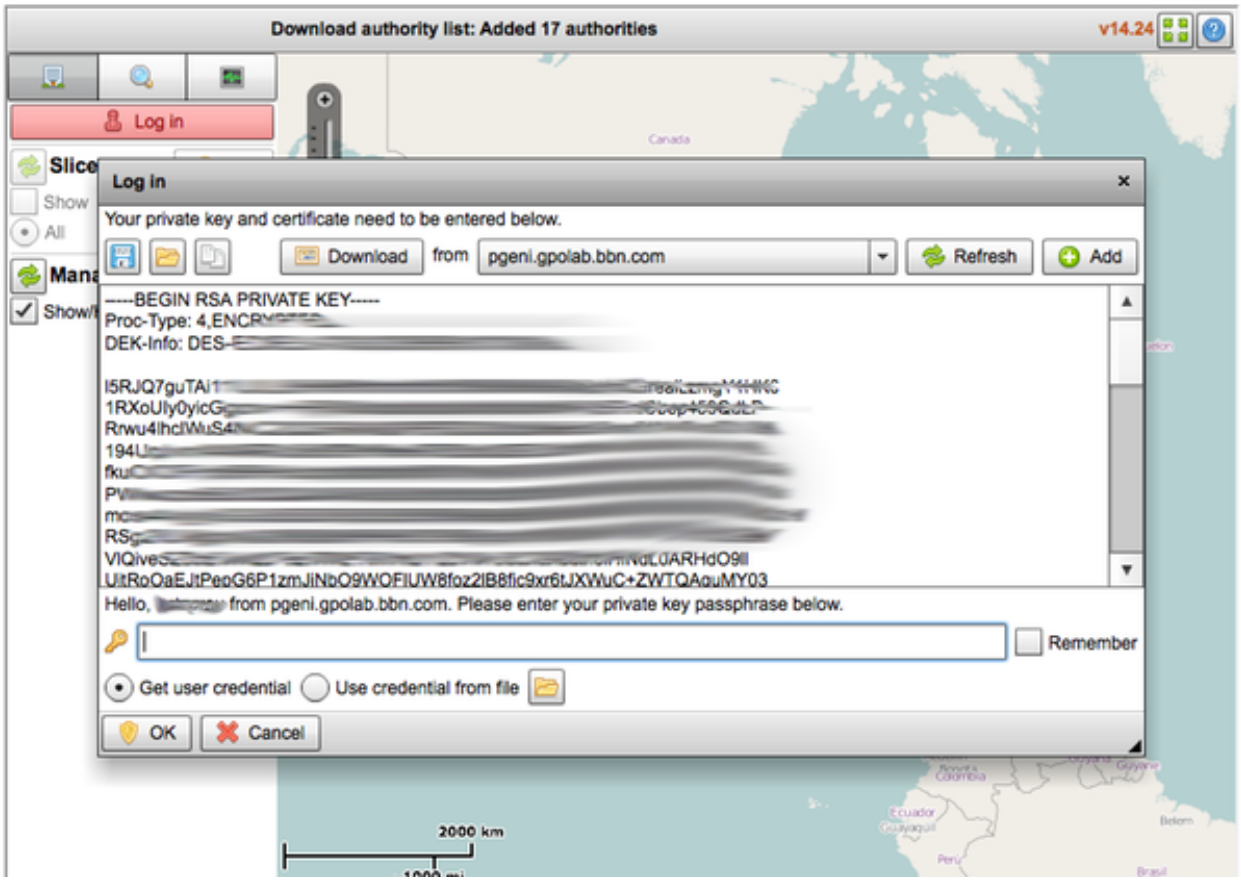
Click the green "Log In" button above. Because we are already logged into our management authority via another tab, we can complete this step easily and have our private key information inserted automatically.

Along the top of the prompt, there should be a button and dropdown box that says "Download from [Select authority]" as in the figure below.



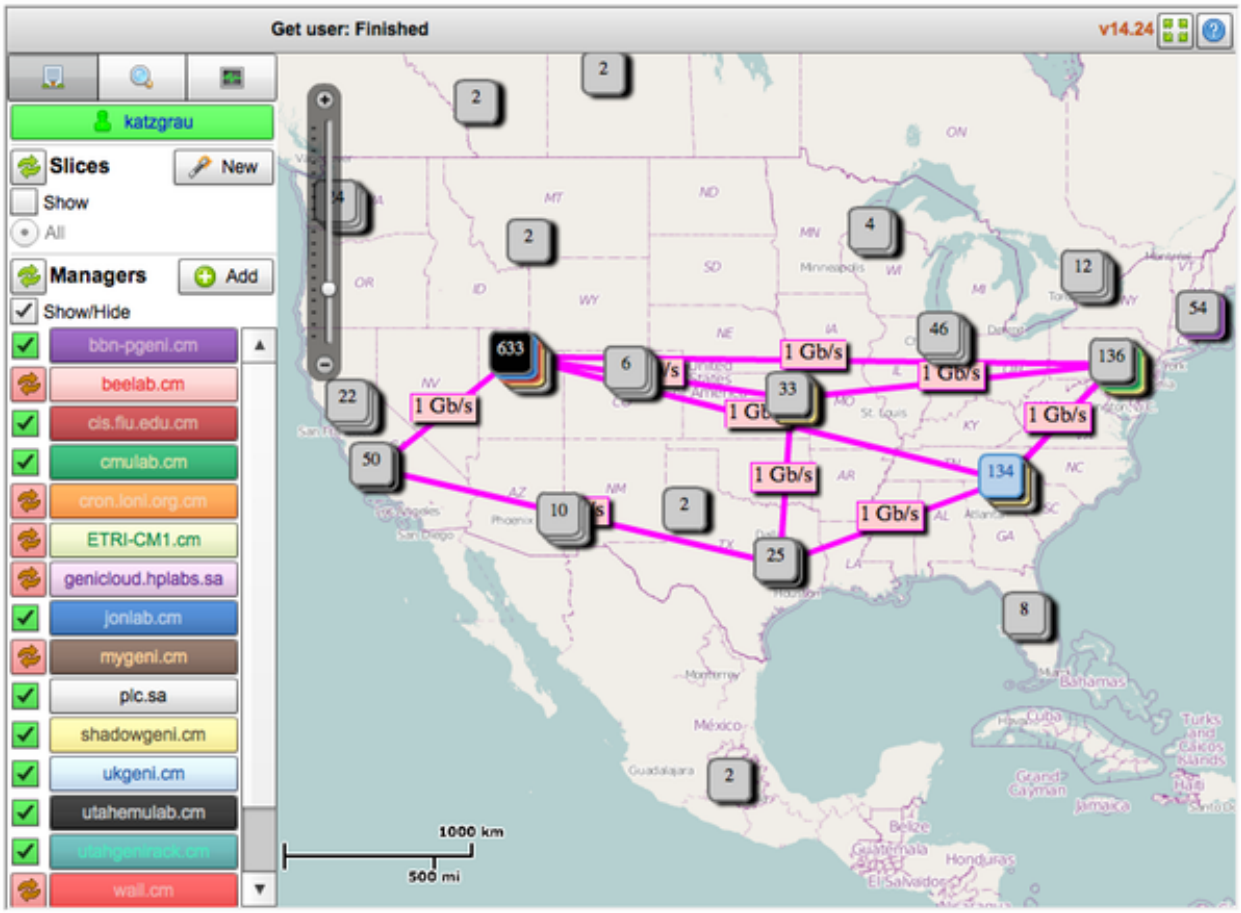
Select your management authority's domain from the dropdown. The correct domain to choose will match up with the domain where you logged in at Step 1.

Once the correct item is selected, click the "Download" button next to the option list. Your certificate (credentials) should appear, as in the figure below:



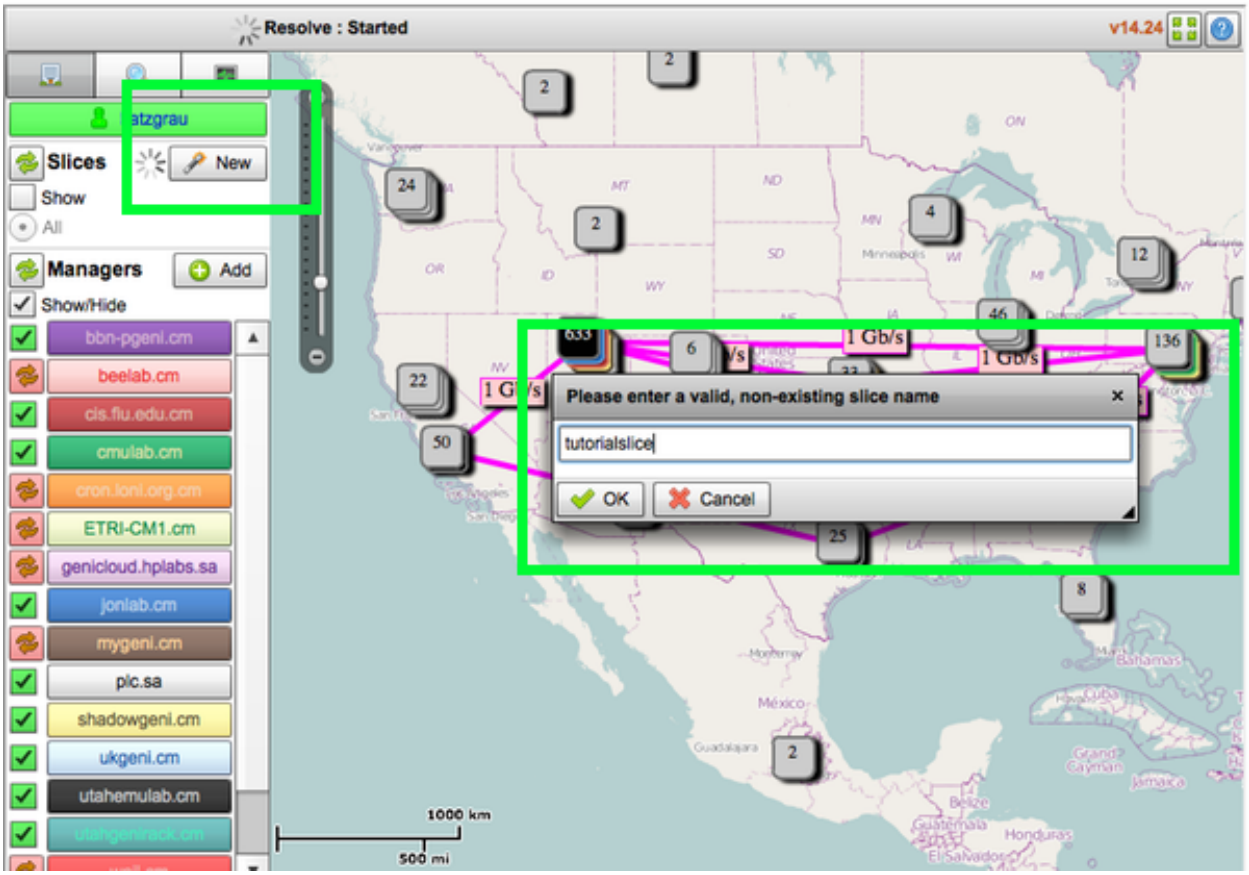
Finally, enter the private key passphrase that you used when you signed up at your management authority when you signed up. Click "OK" on the bottom left.

At this point, Flack will attempt to retrieve resources from the various authorities. When prompted to select where to list resources from, leave all of the options checked and click "Continue". Flack will gather the resources lists, and display a map of North America, as pictured below:

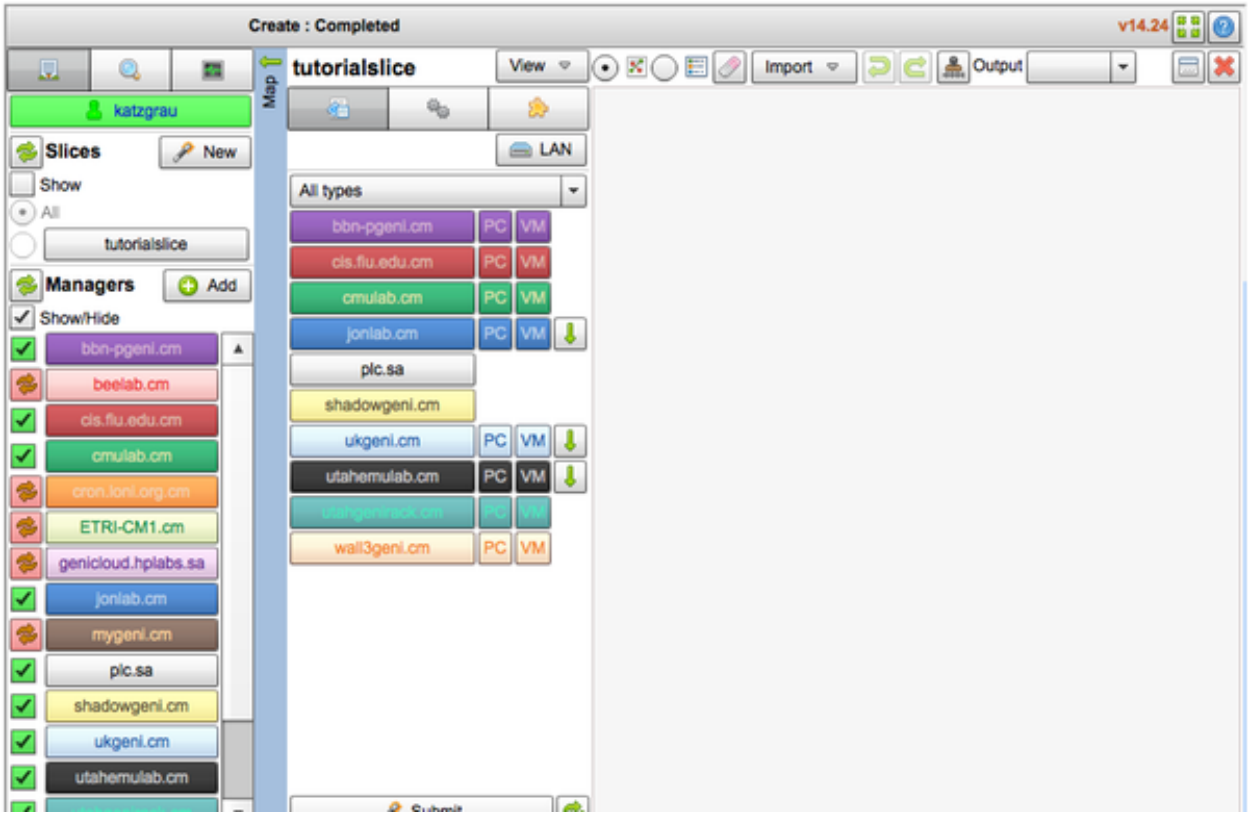


4.1.2 Creating a Slice

Click on the "New" button in the top-left corner of Flack, and enter a name for this slice. **Your slice name will have to be unique from all other GENI slices, so it cannot be the same name as in the example.**



When you're finished typing the name of your slice, click "OK". Once your slice has been created (it may take a moment), you'll see a blank pane with the name of your slice toward the top of your screen. The blank pane on the right is where we'll set up our experiment.

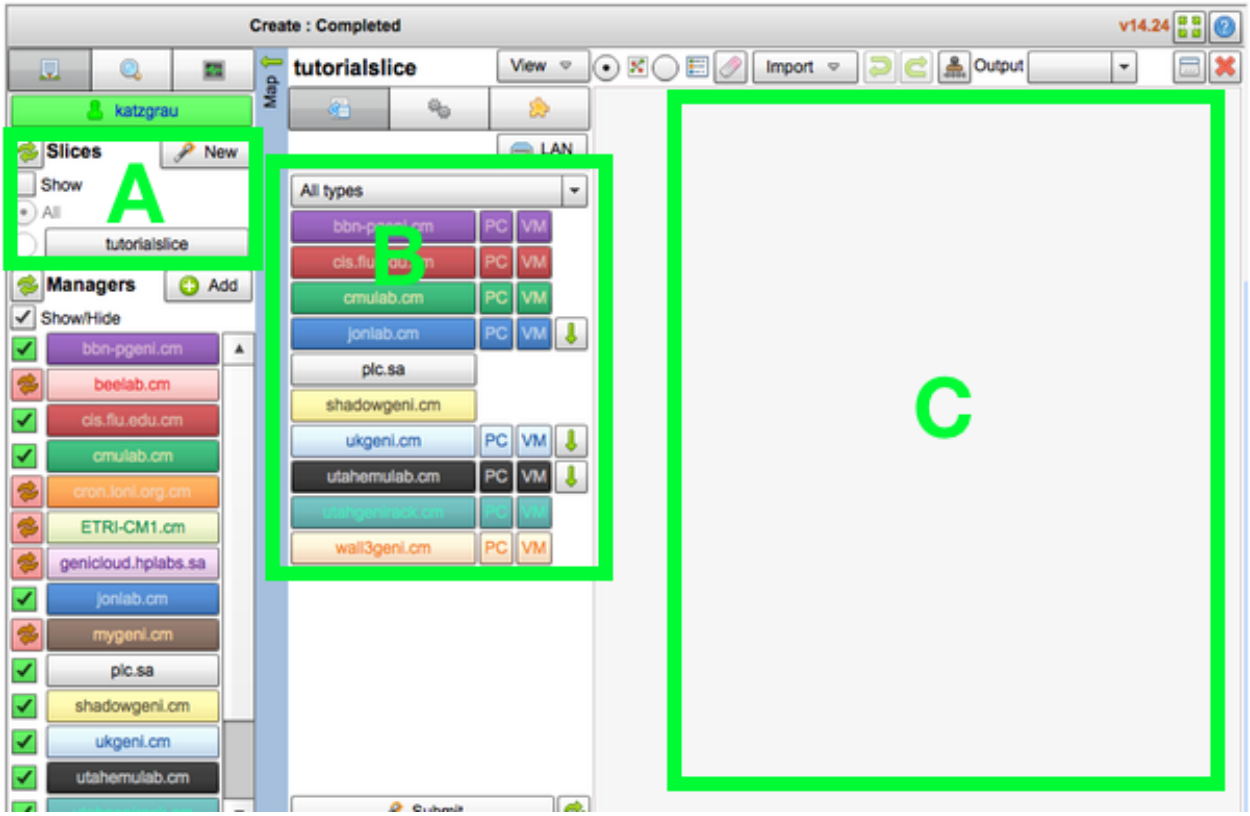


4.1.3 Looking For Resources

In the figure below, we have a new slice aptly named "tutorialslice", which is listed in box A. In box B, there is a list of the resource types that are available to our topology.

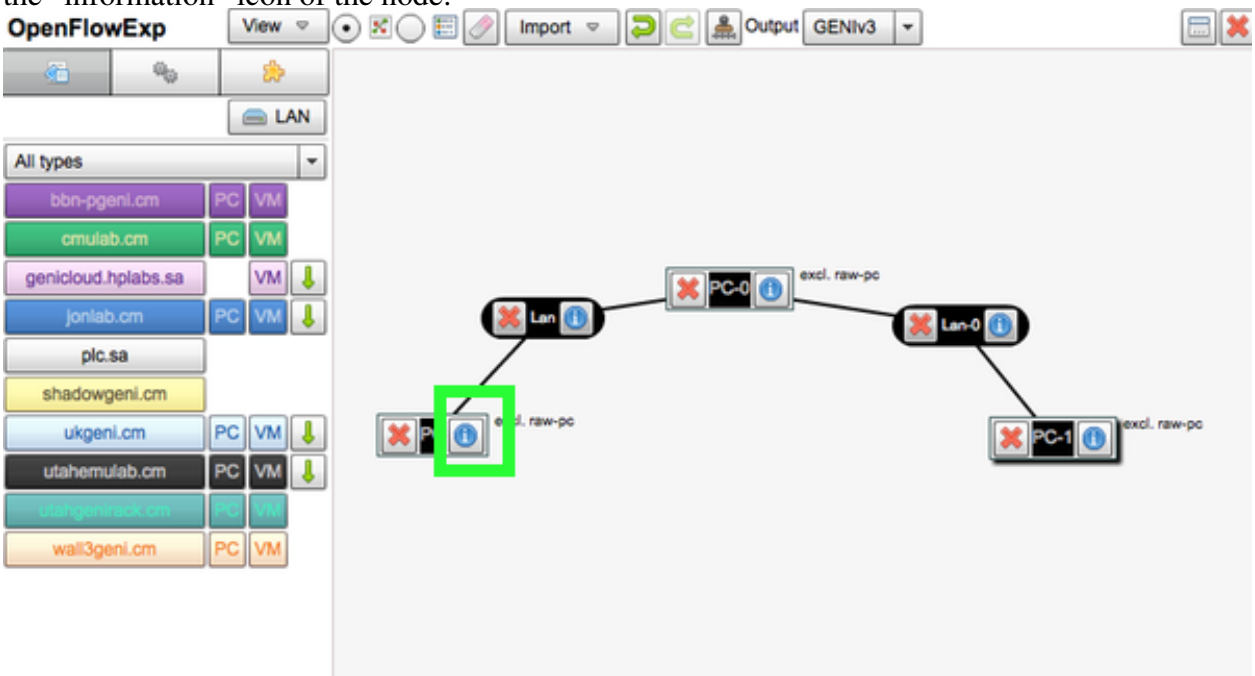
The dropdown control at the top of box B will filter the resource types shown. You can select "raw-pc", which will show only physical computers, emu-vmz, which will show only virtual machines, and other types of resources.

When you find a resource you'd like to use, drag it over to box C.

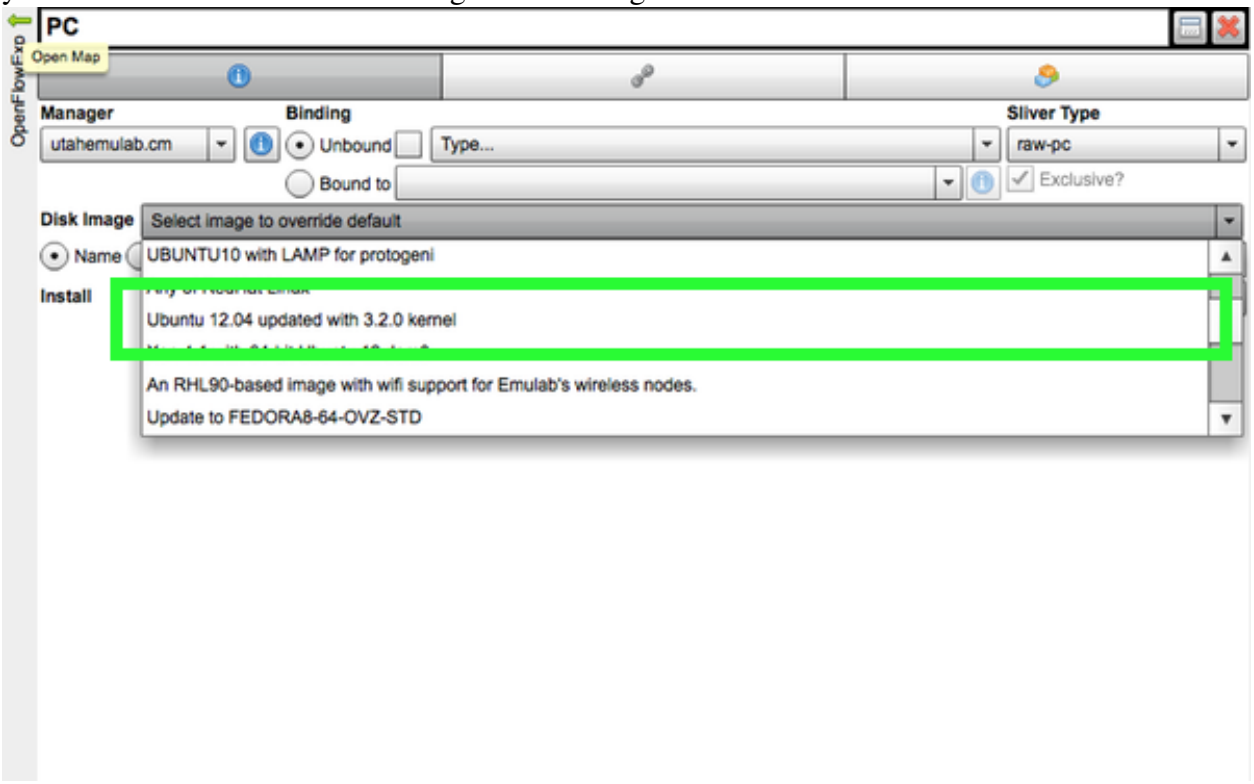


4.1.4 Configuring Compute Resources

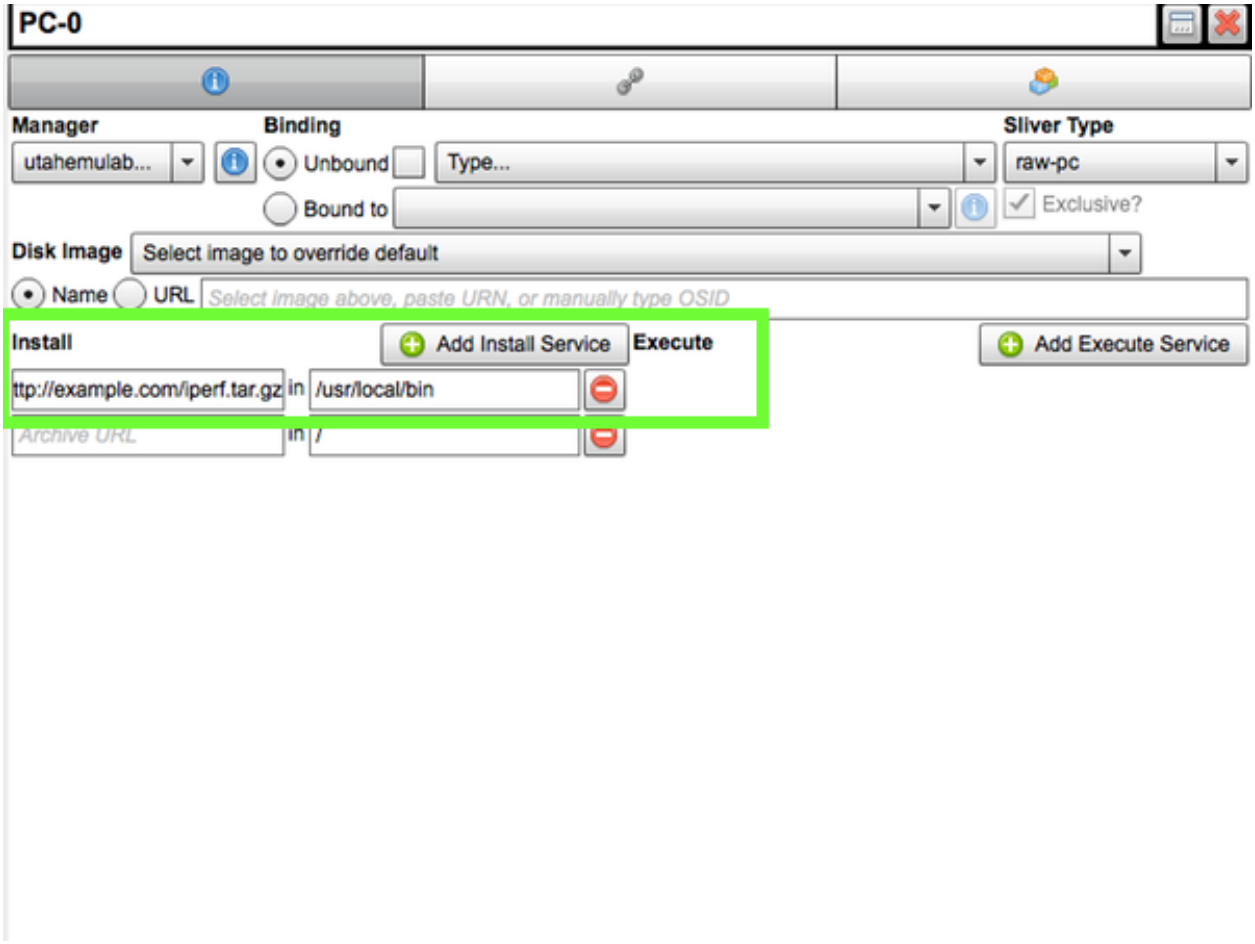
In order to configure a computer resource like a VM or PC that is already in your topology, click the "information" icon of the node:



You will see a configuration pane appear, with an option to select the "Disk Image". This is essentially the base image, and more importantly, the operating system that will be installed to your resource. At this time selecting the disk image is limited to PCs:

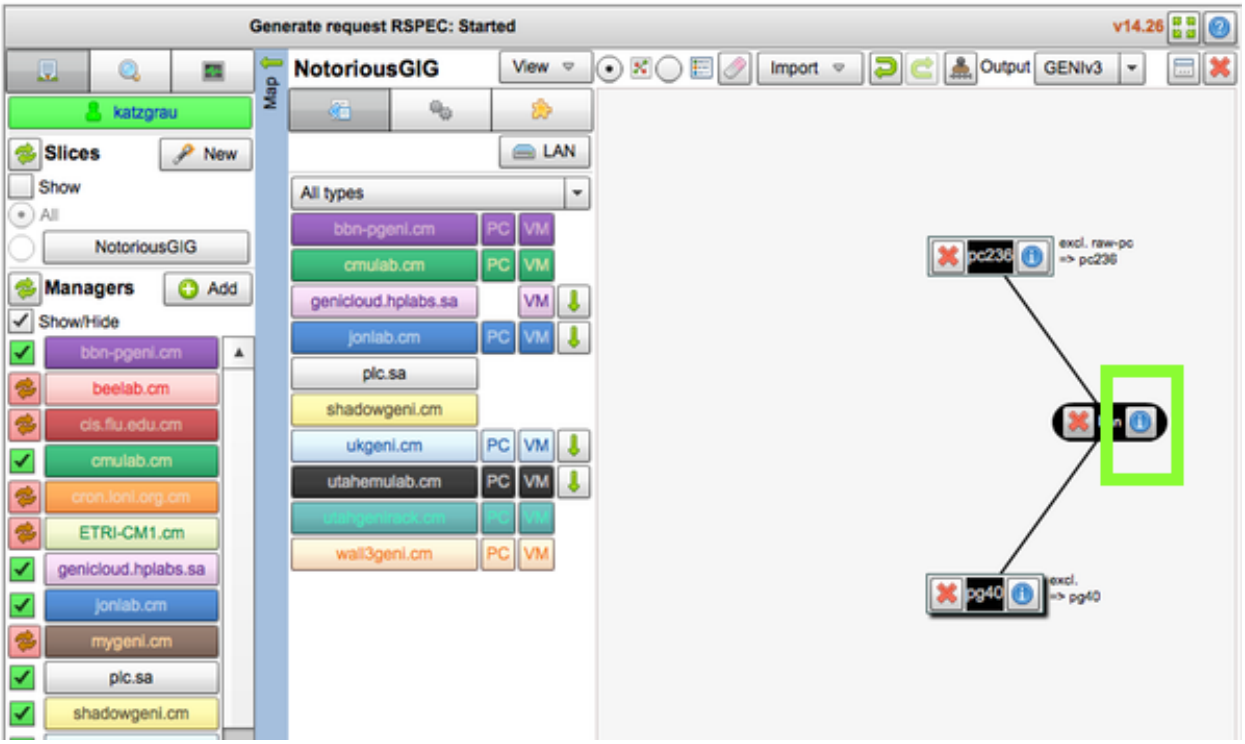


For both PCs and VMs, you are able to add "install services". If you have the URL of a tar'd and gzip'd file with package source code (as most packages are distributed), you can add it here so it will be installed automatically when the machine is brought up:

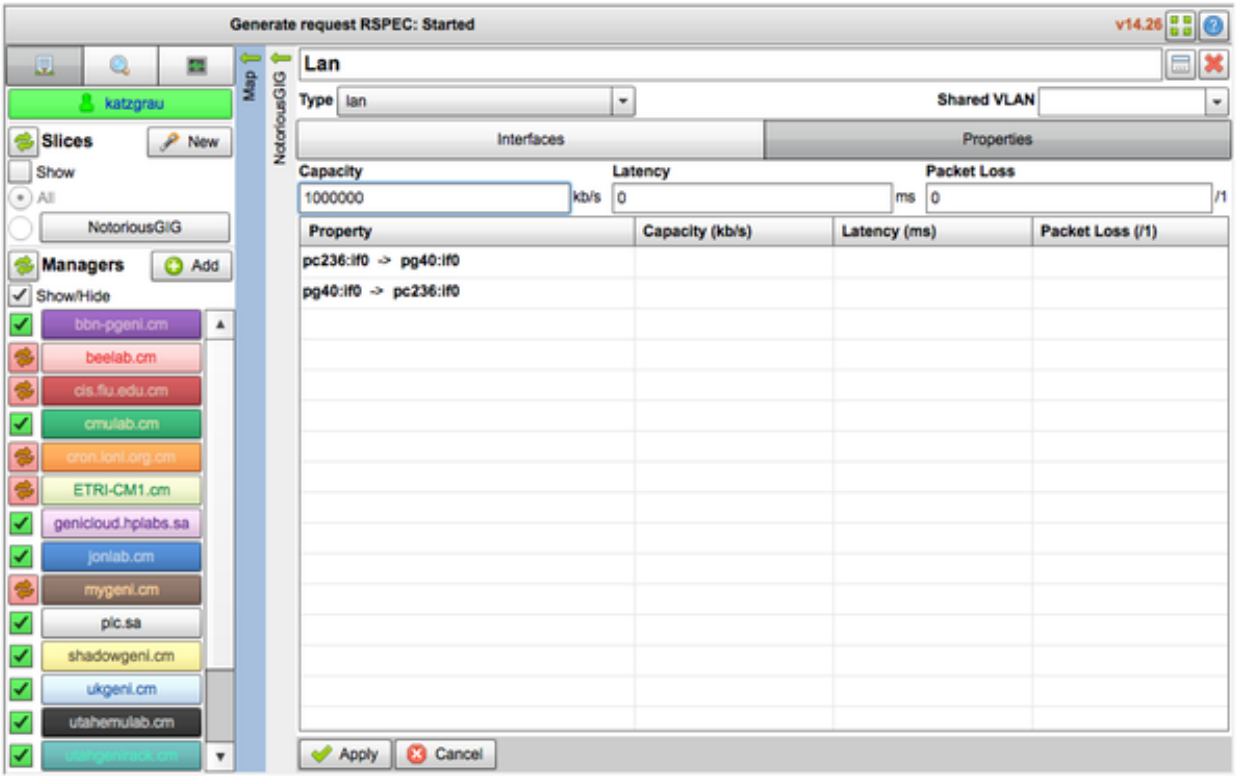


4.1.5 Configuring Network Resources

You can set certain properties of your lan resources, such as the requested link capacity, latency, and packet loss rate. Do that by clicking the lan's information icon:



And setting the appropriate values in the fields:



4.1.6 Additional Resources

Official Flack Manual: http://www.protogeni.net/trac/protogeni/wiki/FlackManual

CHAPTER 5

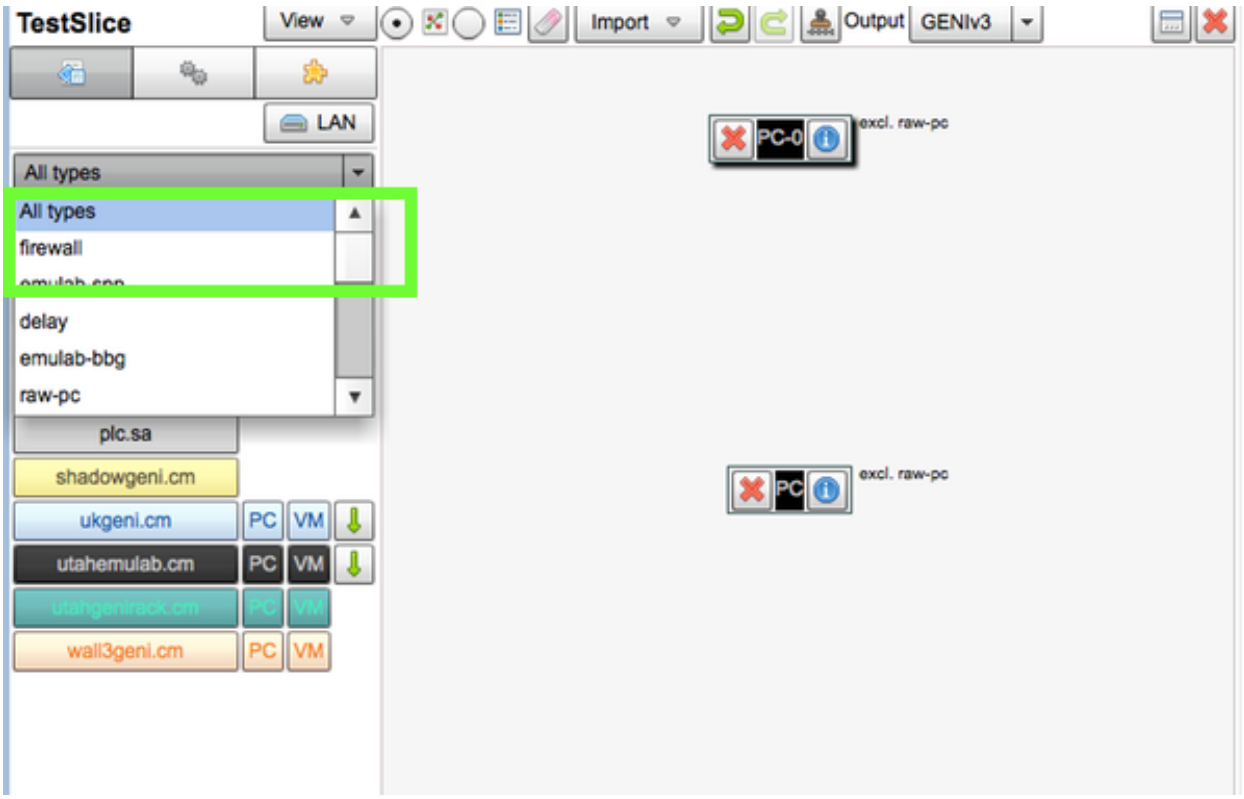
Resource Types

Throughout the examples, we've only seen the use of the standard computing resources available to us on GENI, like VMs and PCs. This section will go over some of the more commonly used resources that we haven't seen yet, in to aspects of PCs and VMS that you may not have been exposed to yet.

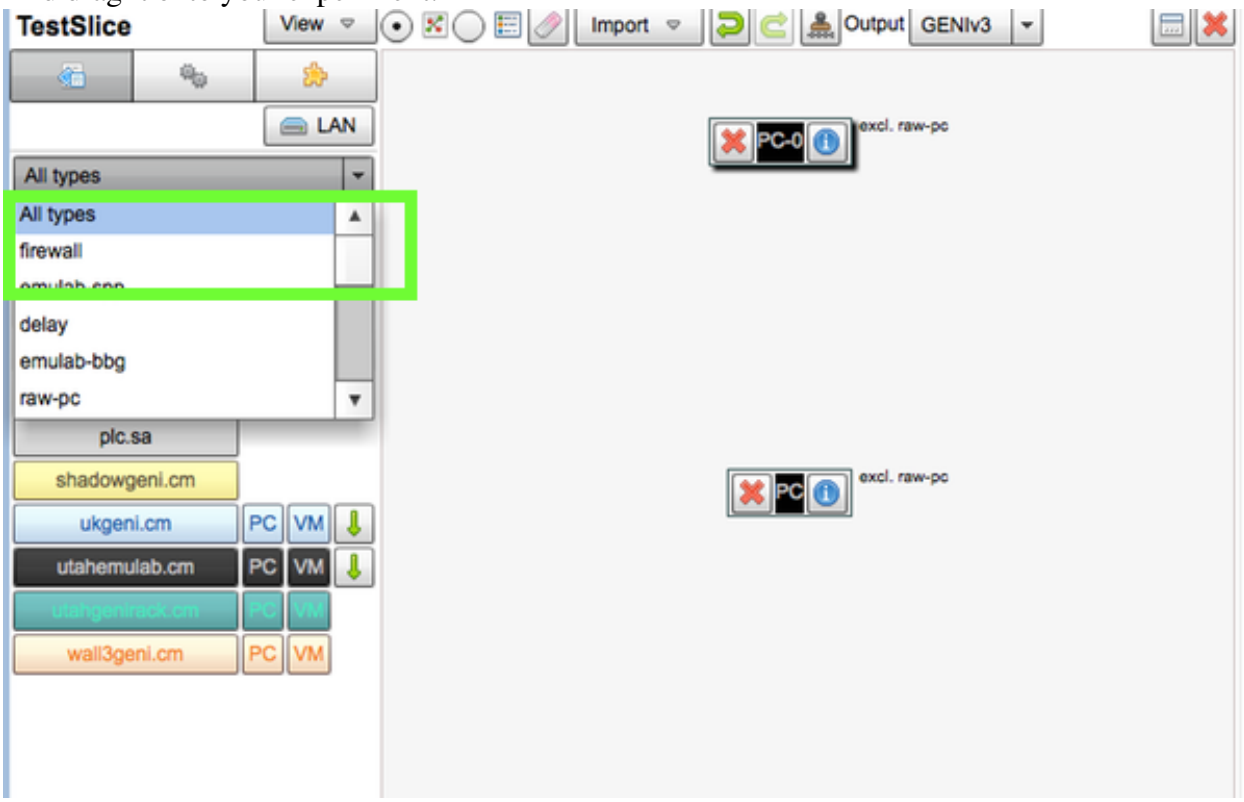
Contents

5.1 Firewalls

Setting up a firewall in your GENI topology is as simple as dragging a node onto the experiment pane. Simply filter your list of resource types using the dropdown menu on the left. Select 'firewall':

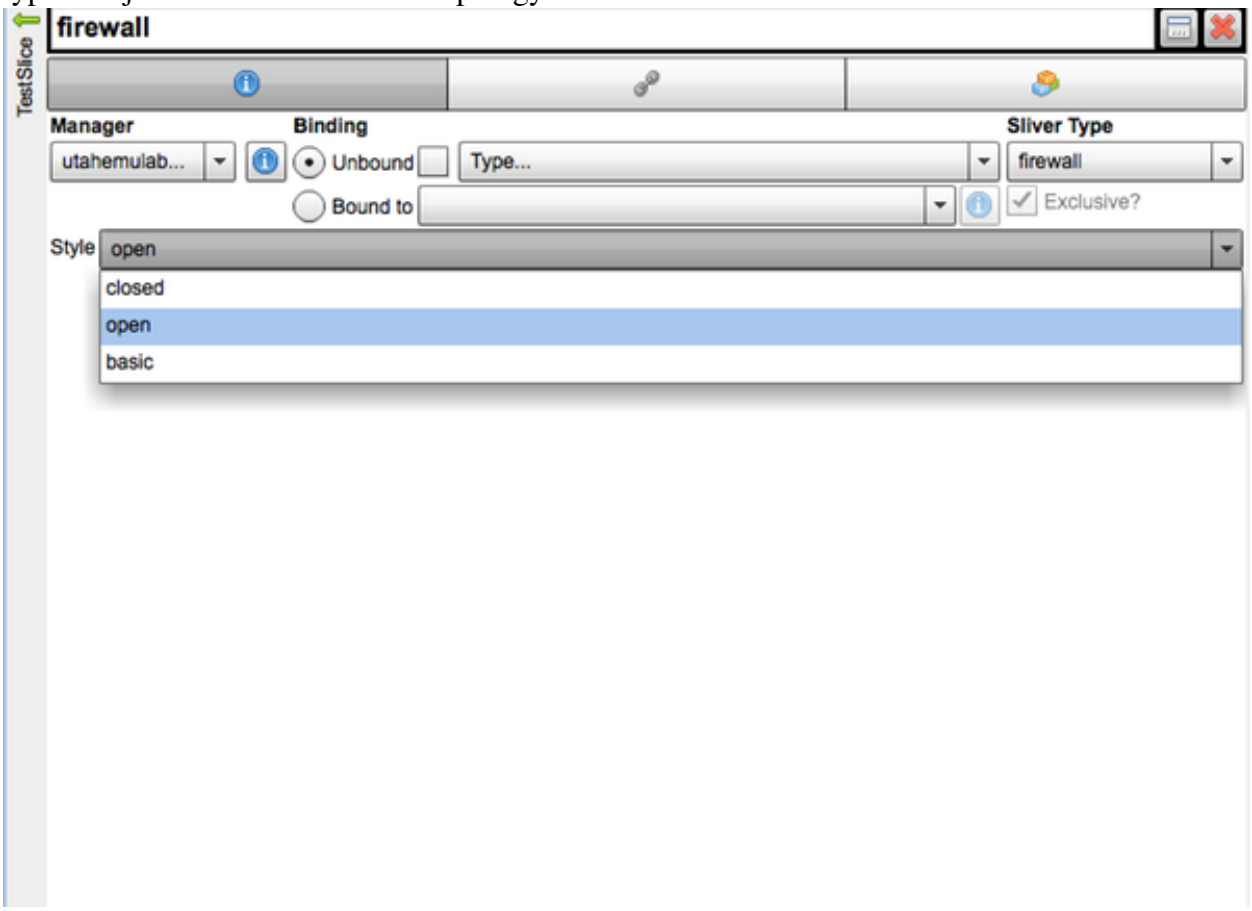


And drag it onto your experiment:



You can network and configure the properties of a firewall just like you can with any other resource

type. It's just another node in the topology:



In the properties window above, you can specify that the firewall is either open, closed, or basic.

- Open: All traffic may flow freely
- Closed: No traffic may flow
- Basic: SSH and HTTP/HTTPs are allowed

Using an "open" style, you will be able to access the node to define rules at a later time.

5.1.1 Additional Resources

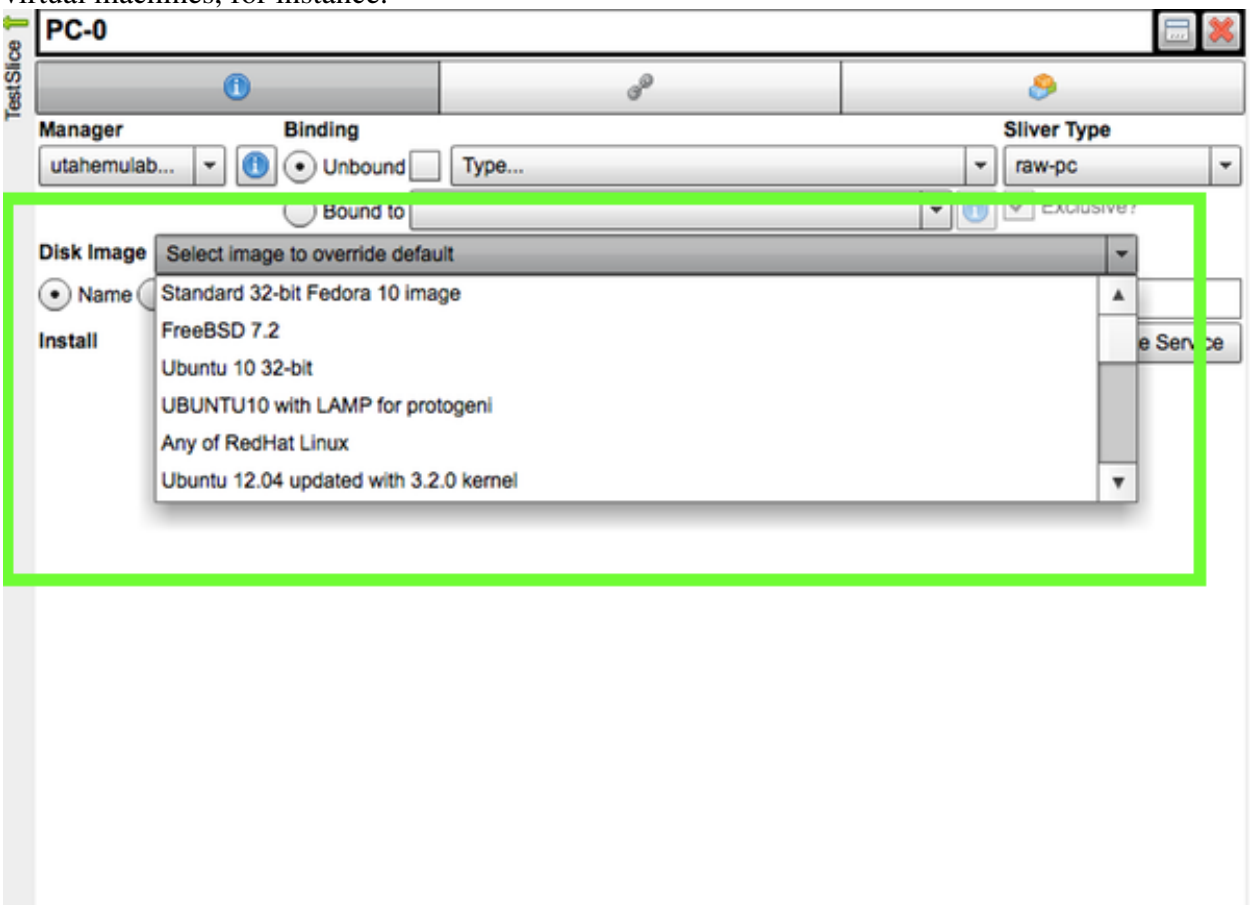
An emulab tutorial with more detail on firewalls: http://www.uky.emulab.net/tutorial/docwrapper.php3?docname=

5.2 Physical Machines

Of the two computing resources you'll have available to you on GENI, Physical Machines are one of them. They are most often referred to as "PC" or "PCs" in documentation and software tools.

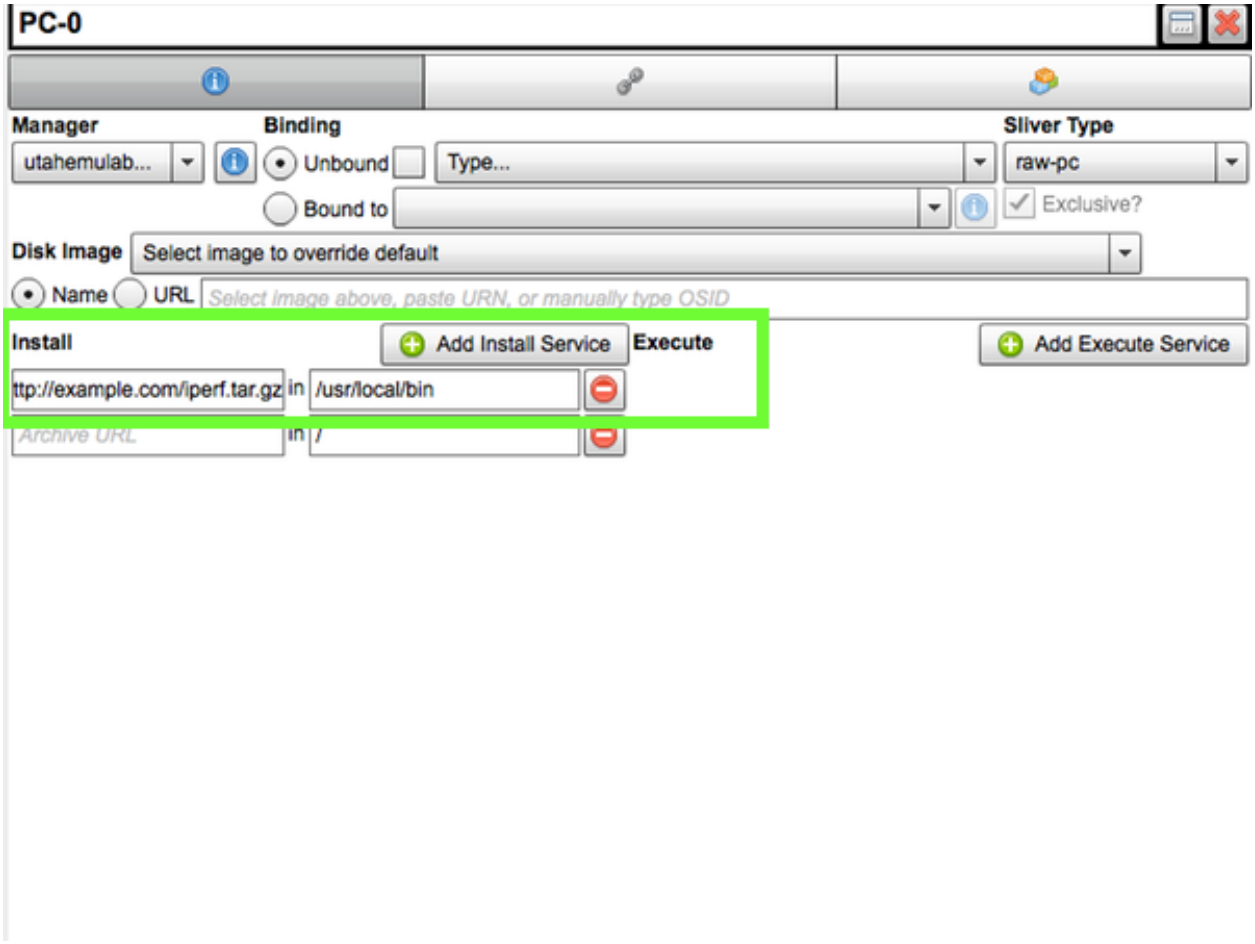
5.2.1 Install A Specific Operating System

By default, GENI nodes get Fedora Core installed on them when created. You can override this by clicking into the properties of a PC and setting its operating system. Your choices are fairly standard: Ubuntu, Redhat, FreeBSD, or Fedora. Additionally, there are distros that come pre-packages with tools like KVM (Kernel Virtual Machine), which might be handy if you plan to host virtual machines, for instance.



5.2.2 Install Packages Automatically

If you have the URL to a package you would like to have installed for you, you can add an "install service". When the machine is created, the package will be installed in the directory that you specify. This option is available in the properties pane of the PC.



5.3 Virtual Machines

Of the two computing resources you'll have available to you on GENI, Virtual Machines are one of them. They are most often referred to as "VM" or "VMs" in documentation and software tools.

Virtual Machines are almost the same as PCs, with the exception that you are probably sharing compute resources with another VM on the host. Additionally, you don't have the same ability to select from a set of available operating system for your node, as you do with a PC. You will be defaulted to the standard Emulab Fedora Core 6 image.

To see the configurable options for a PC, read more here.

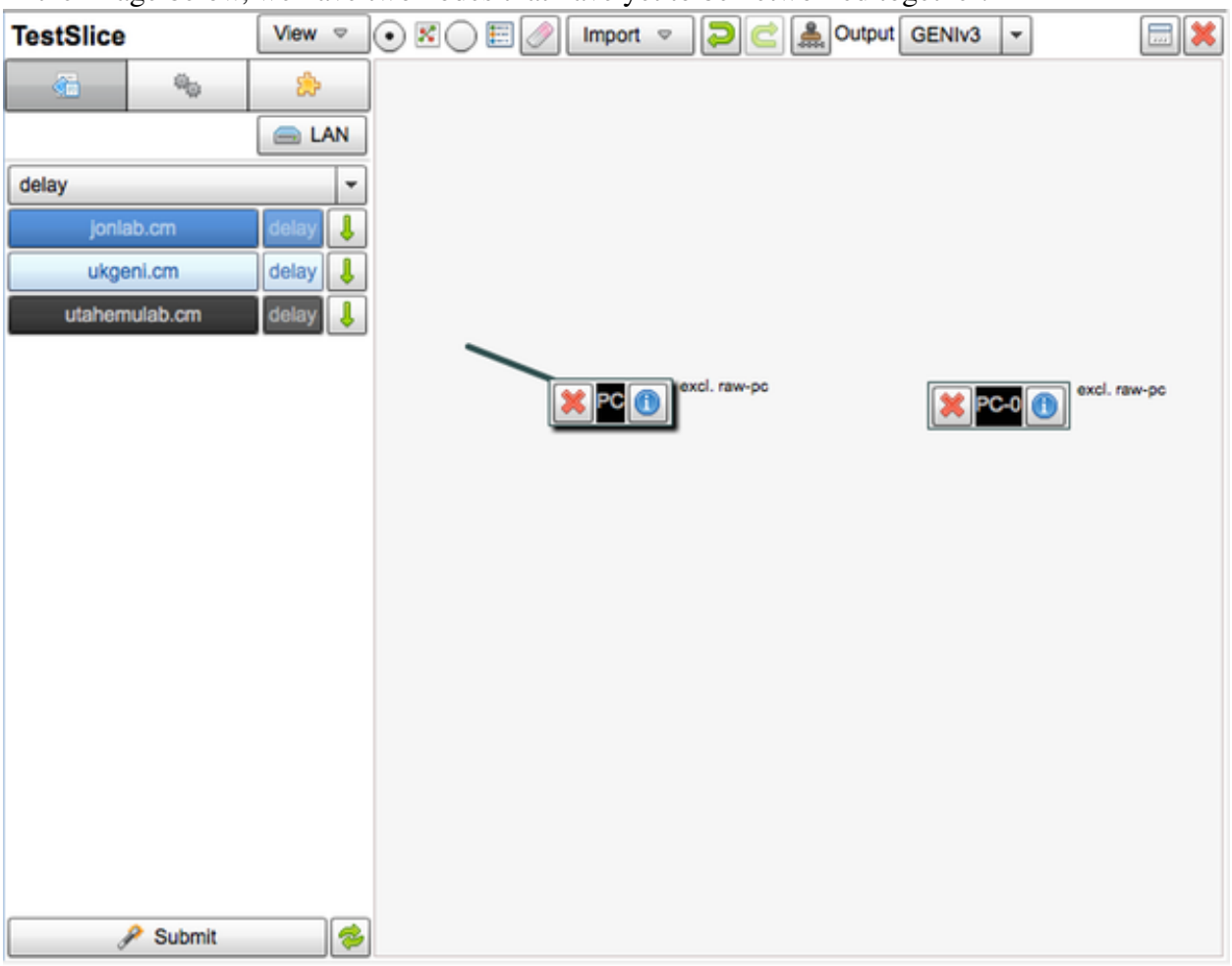
5.4 Delay Node

A delay node is a type of resource with can simulate a data transfer bottleneck. This can be handy for running experiments that would ideally have some set speed of data transfer as opposed to whatever the network is capable of.

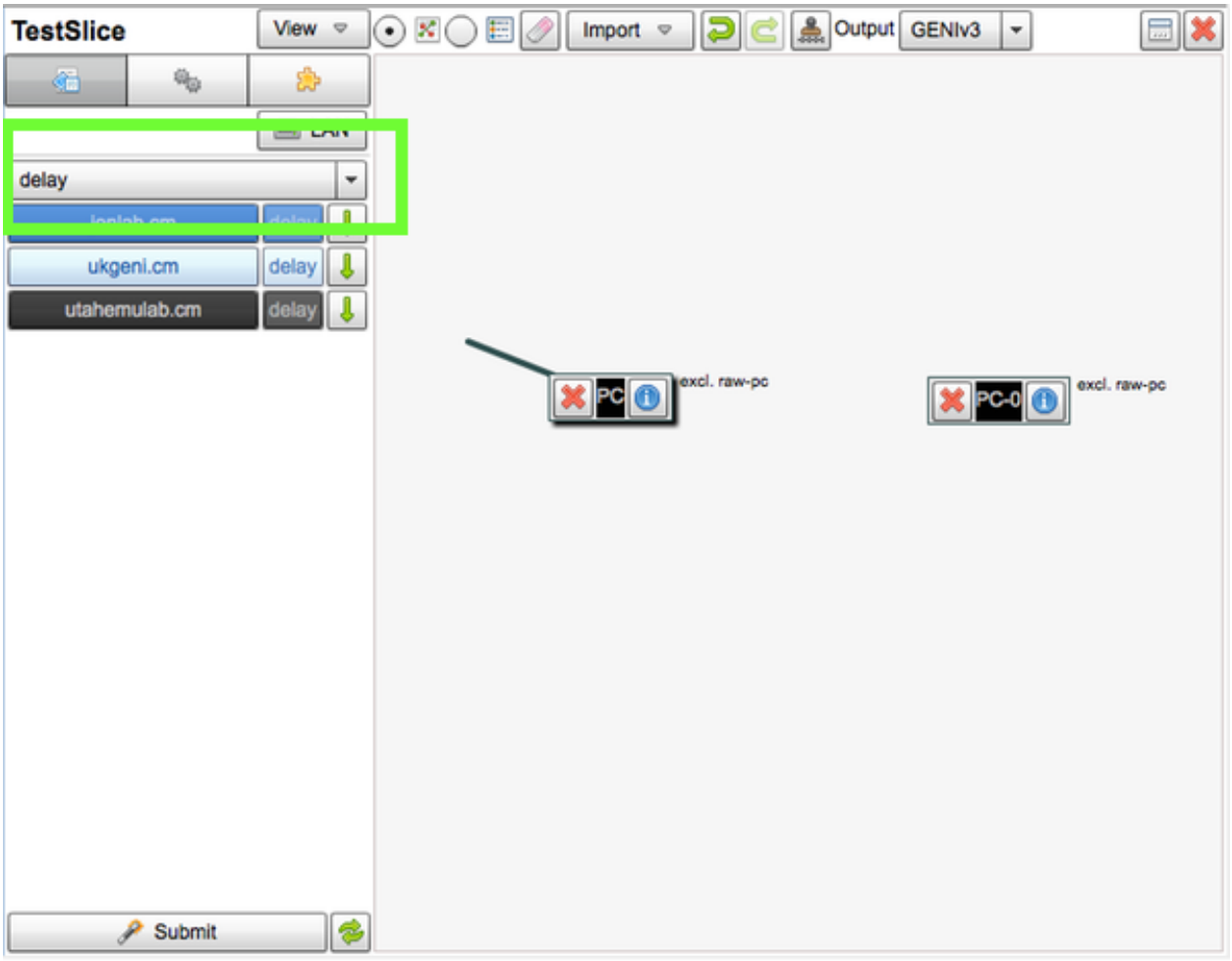
For instance, let's say you are running a GENI experiment on a single site, where the inter-node data transfer speeds are 1000 Mbps. If you're running an experiment that tests new protocols, it might make sense to run the experiment with different qualities of service and/or network characteristics.

5.4.1 Changing Network Characteristics

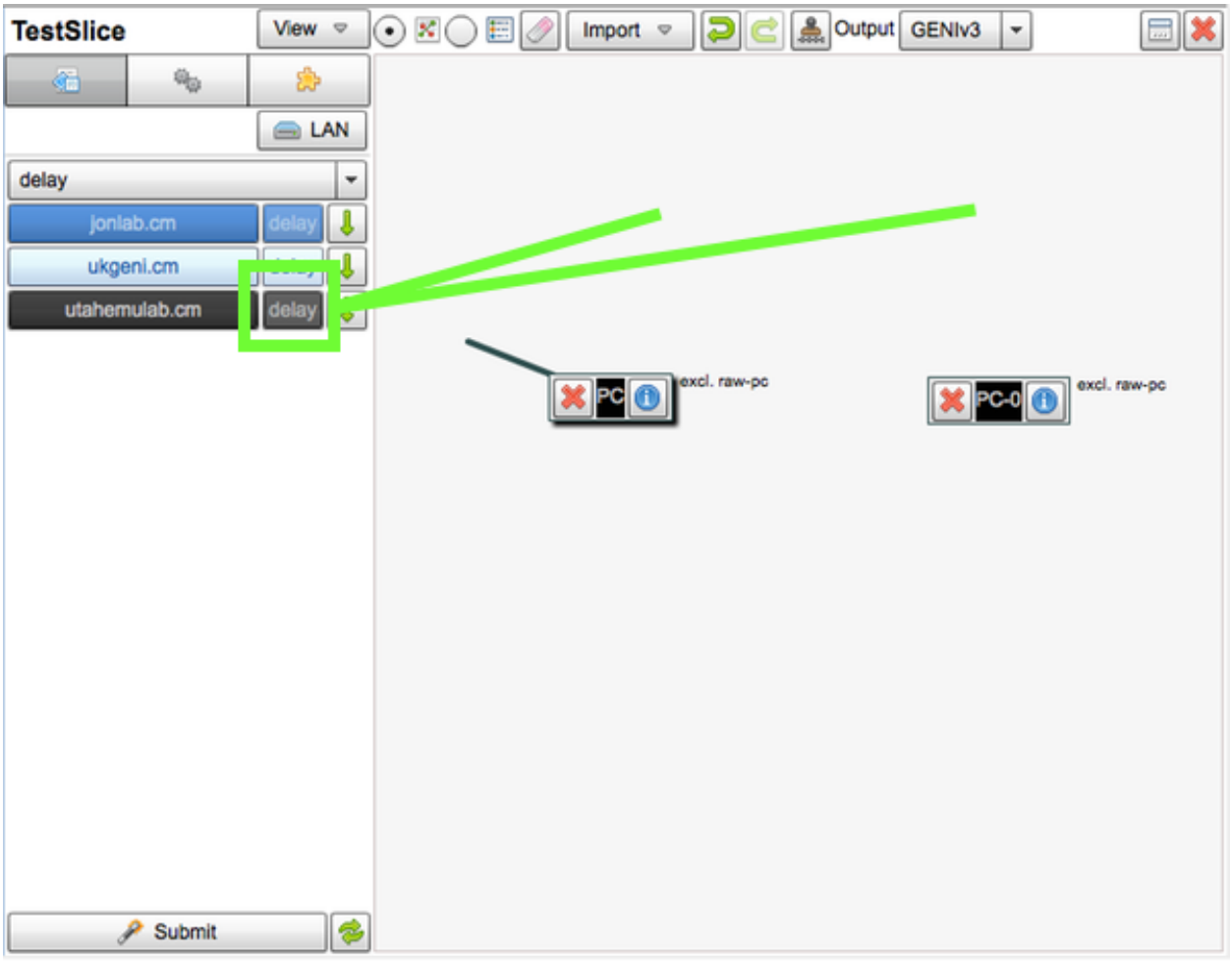
In the image below, we have two nodes that have yet to be networked together:



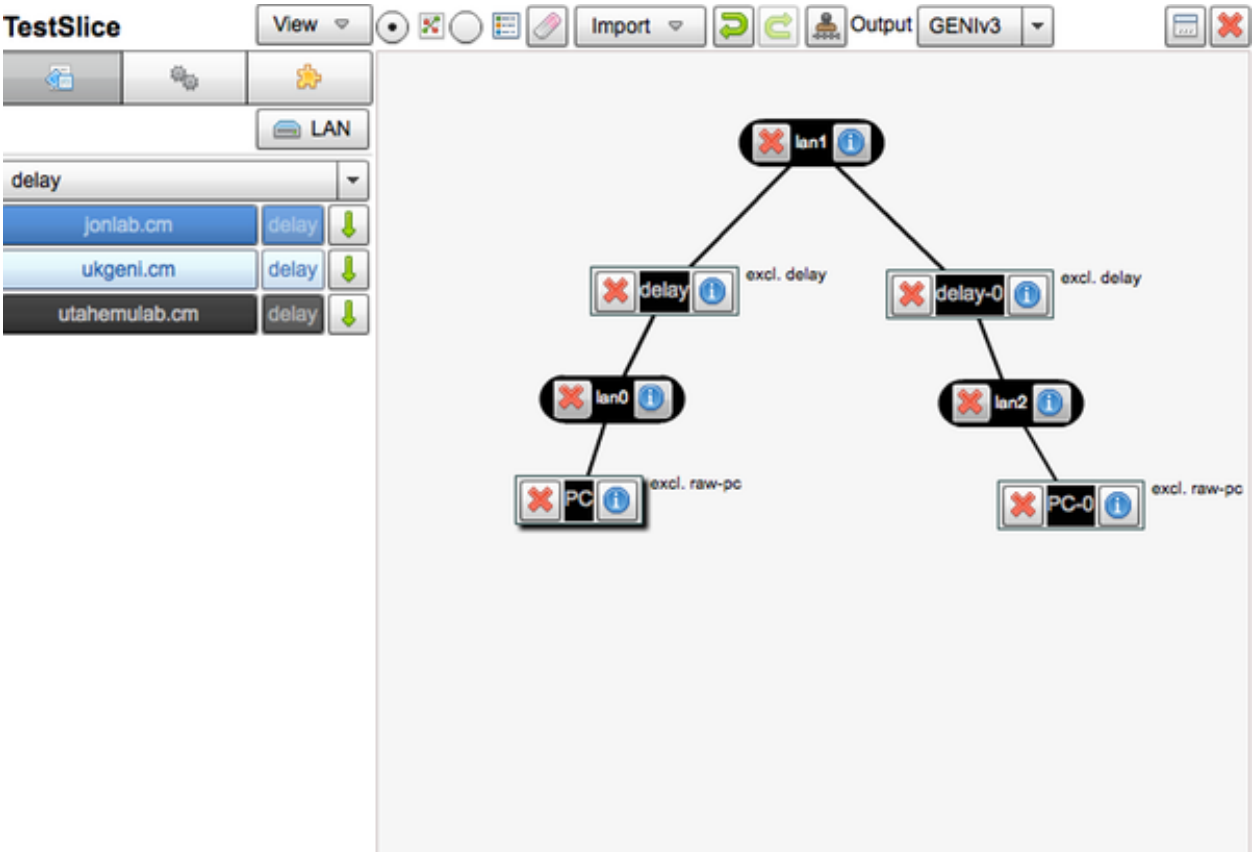
In the resource list drop down, we're going to select the 'delay' option:



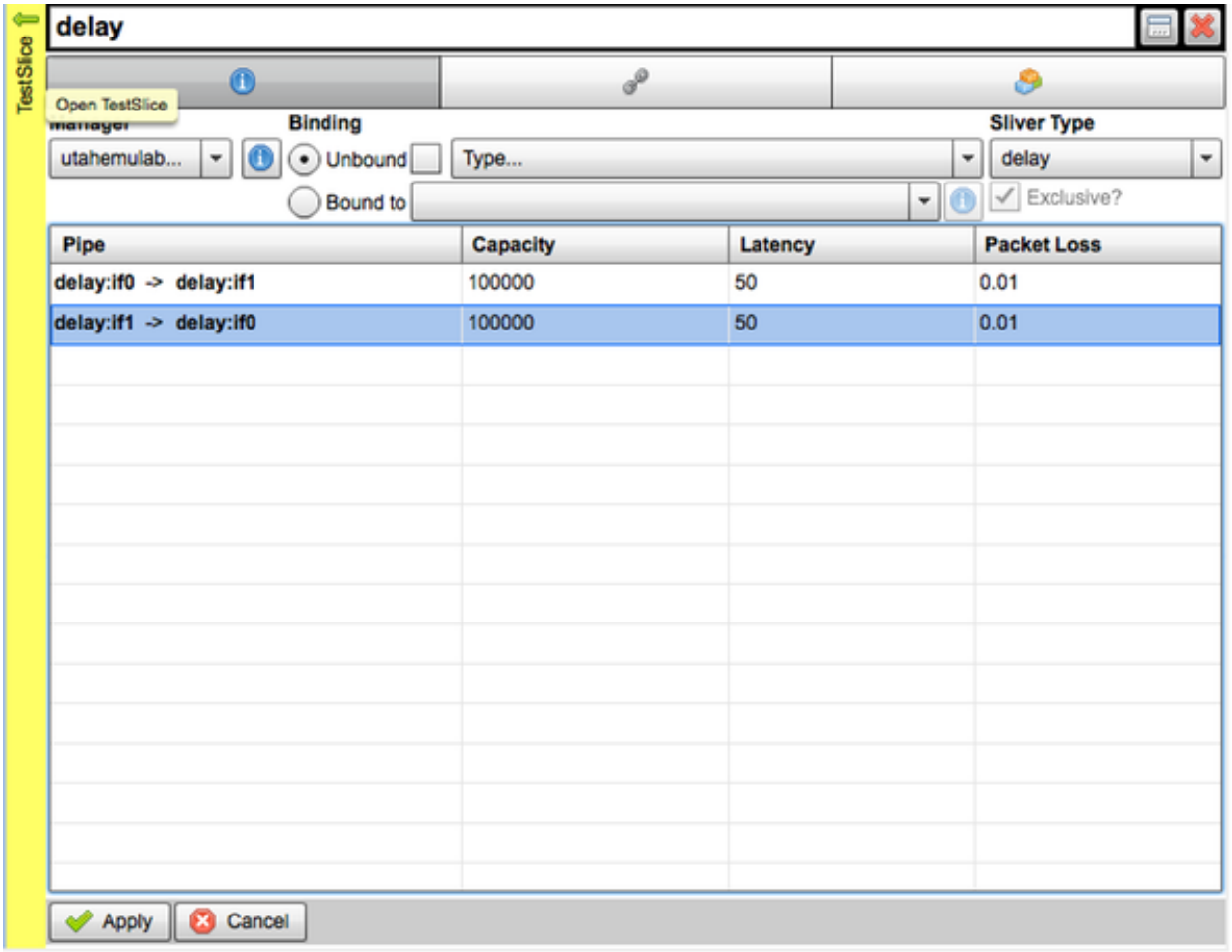
From there, we'll want to drag a delay node (or two) onto the pane



Finally, let's drag links between the nodes so the LANs are set up automatically:



At this point, let's click the *i* icon on the delay node. We'll see a properties pane appear with areas to set capacity, latency, and packet loss on either incoming or outgoing traffic. At this time, capacity is in kb/s, latency in milliseconds, and packet loss as a number between 0 and 1.



Apply your settings to finish it off.

CHAPTER 6

GENI Resources

What kind of resources do you generally have available to you with GENI? This section describes setting up virtual machines, physical machines, networks, and more.

Contents

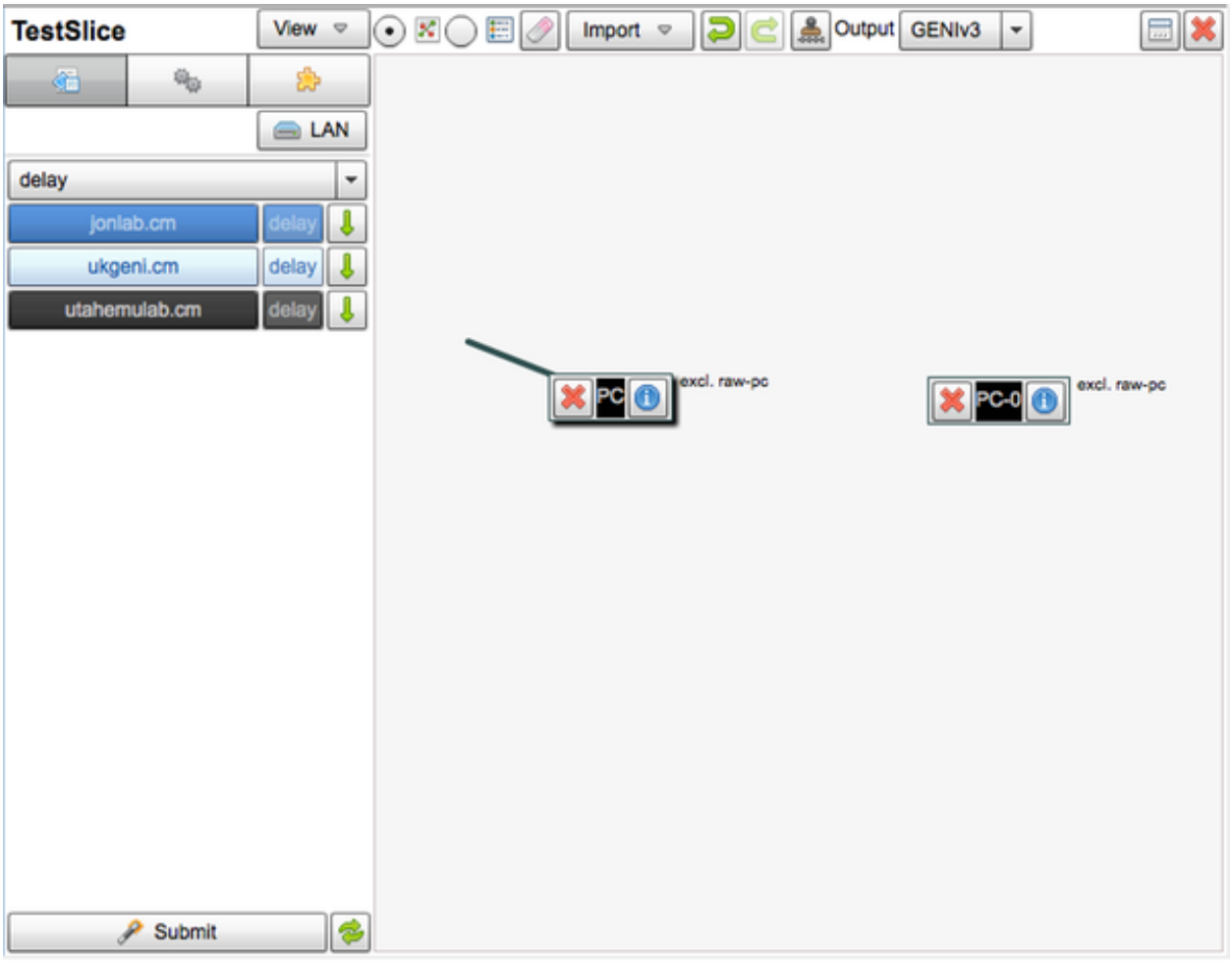
6.1 Simulating Gigabit Speeds

If you aren't quite ready to set up a multi-site gigabit network, you can also *simulate* gigabit speeds with a special node type called a delay.

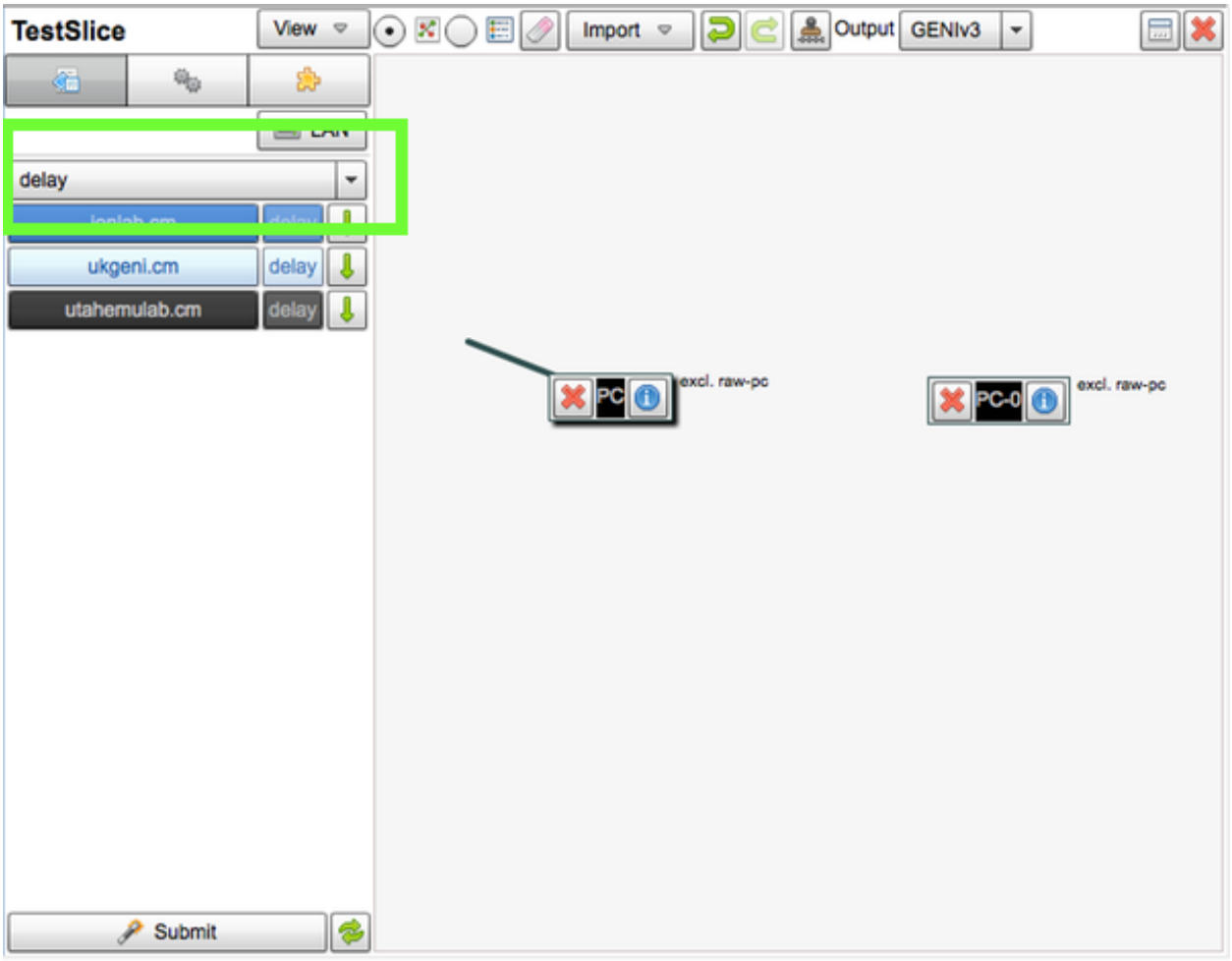
The delay node allows you to configure extra characterists such as link capacity, packet loss rates, and general latency. This is useful for capping your connection speed to other nodes, or simulating a less-than-desirable environment for your app. After all, in reality, latency and packet loss are real issues that must be dealt with.

6.1.1 Changing Network Speed Artificially

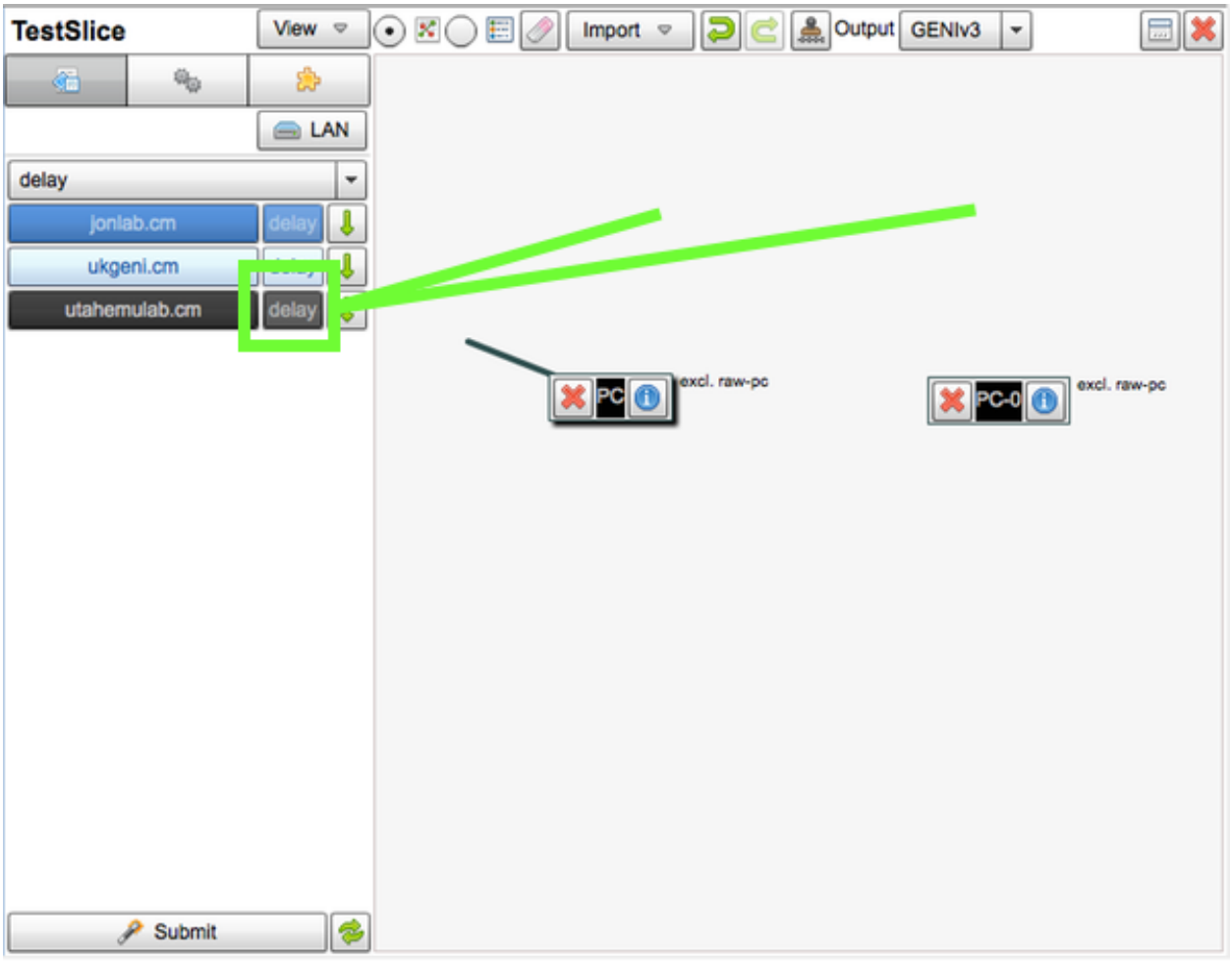
In the image below, we have two nodes that have yet to be networked together:



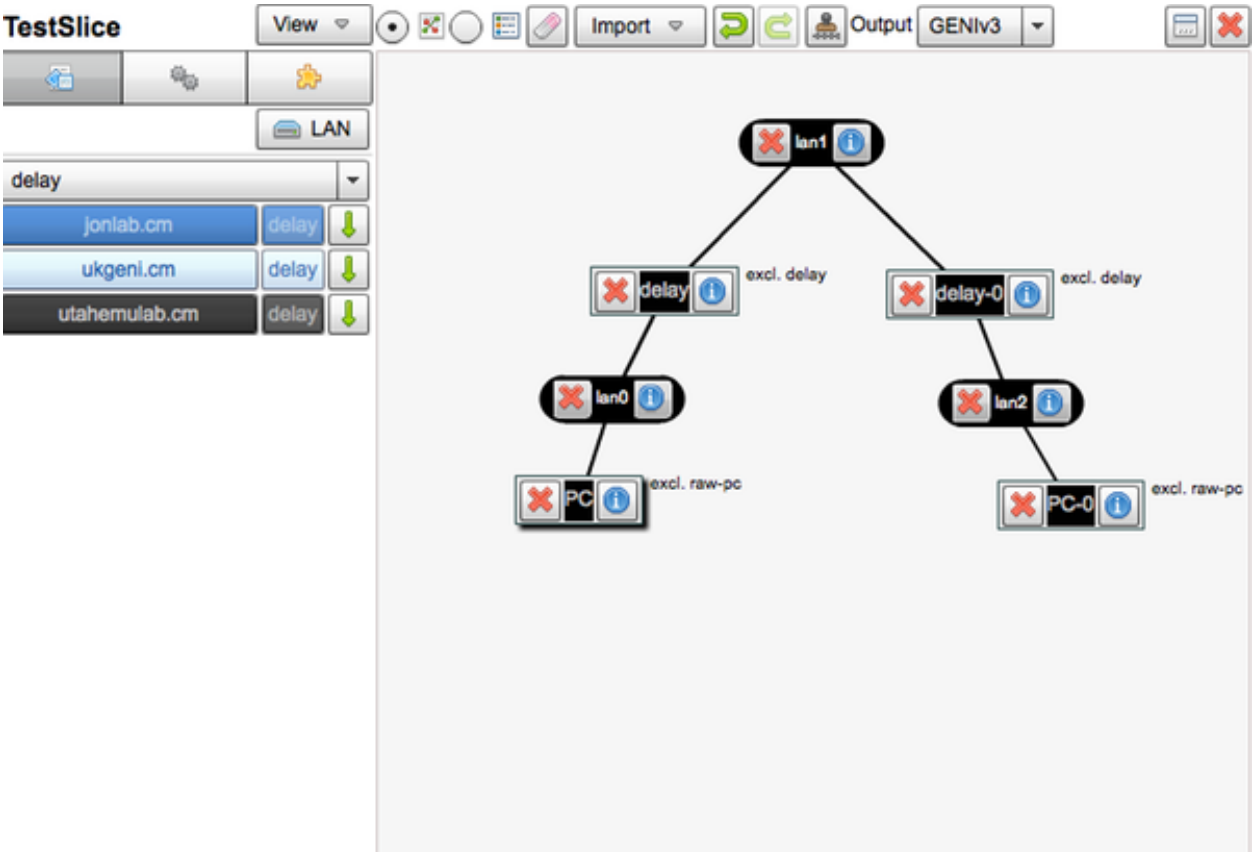
In the resource list drop down, we're going to select the 'delay' option:



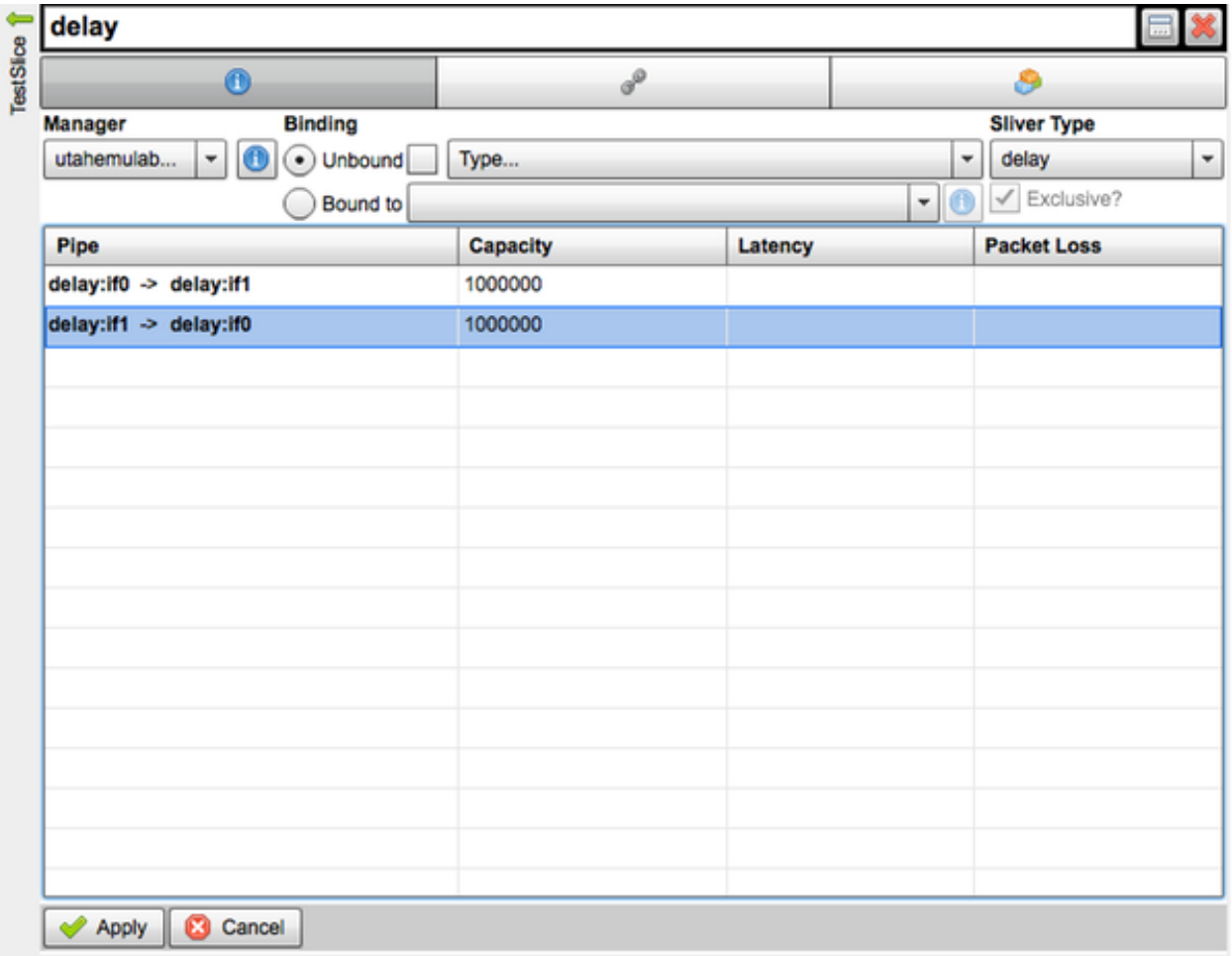
From there, we'll want to drag a delay node (or two) onto the pane



Finally, let's drag links between the nodes so the LANs are set up automatically:



At this point, let's click the *i* icon on the delay node. We'll see a properties pane appear with an area to set the link capacity.



Apply your settings, and you can now build and run your app as if you're on a gig network.

6.2 Real Gigabit Speeds

Getting an experiment or app up and running on GENI means that you have access to gigabit-speed networking. You might reserve one machine that will act as a server, and perhaps 9 others that will act as clients. Between them, you might have have high-speed gigabit connections.

As a web developer, building an app that depends on high speed is difficult — most end-users today aren't on a gigabit-capable internet service, so it makes building such applications impractical or impossible. On GENI, although, you have the ability to run your app in an environment where speed isn't an issue. You can truly build an app *for the future*.

The second example project, listed in the projects section, demonstrates how easy it is to get set up on a high speed connection within the GENI environment:

- Click here to head over to Example #2

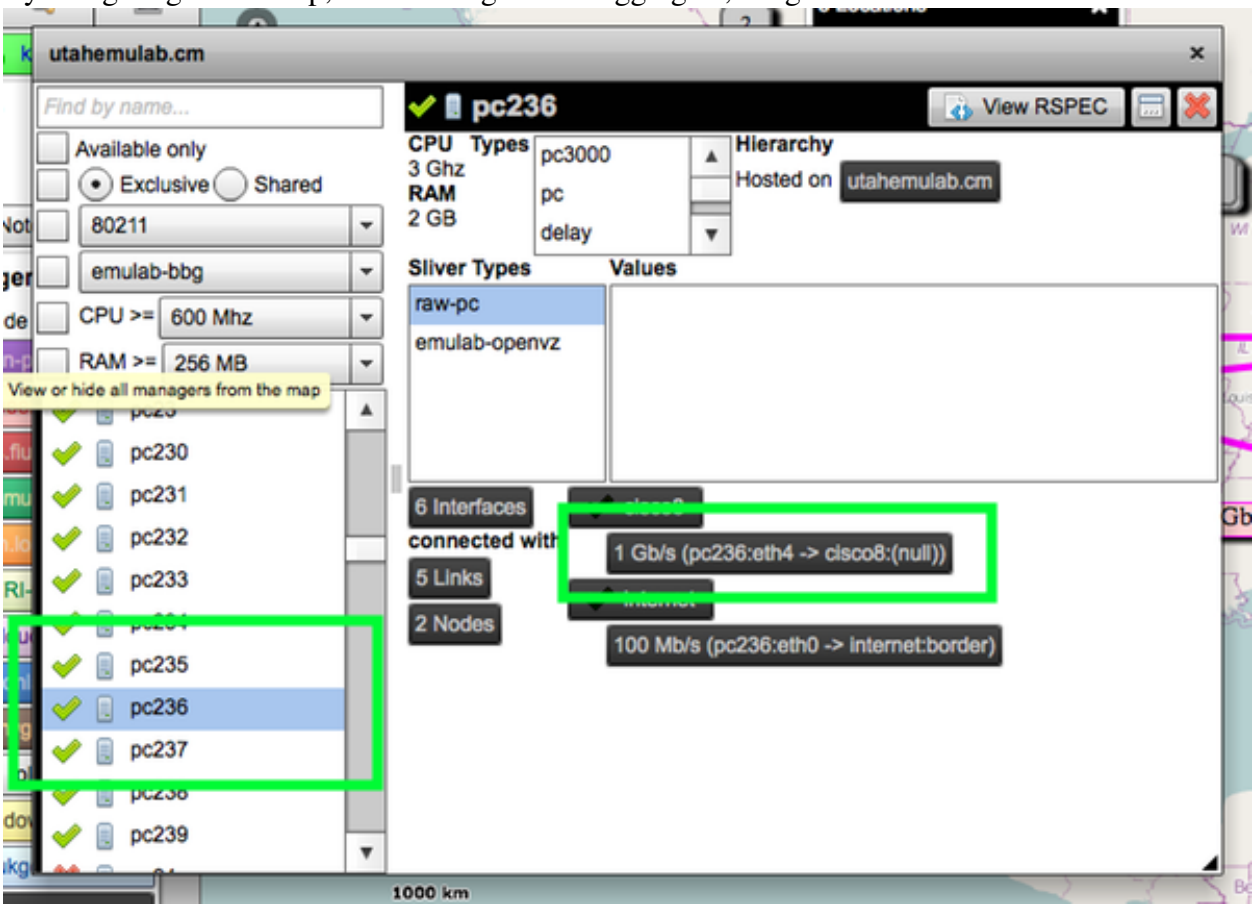
6.2.1 Requesting Gigabit Links On GENI

When you're setting up your experiment architecture, you have the ability to set the bandwidth properties of the network that the nodes are in. There are three important steps in making sure you get a gigabit link:

1. Select nodes with have a gigabit interface
2. Select nodes that are on a gigabit backbone (if applicable)
3. Request that network link between the nodes is for 1000000 kb/s

1. Selecting the Right Nodes

In the image below, we already have a slice set up, and we're selecting nodes for our experiment. By navigating to the map, and clicking into an aggregate, we get a list of nodes available to us:



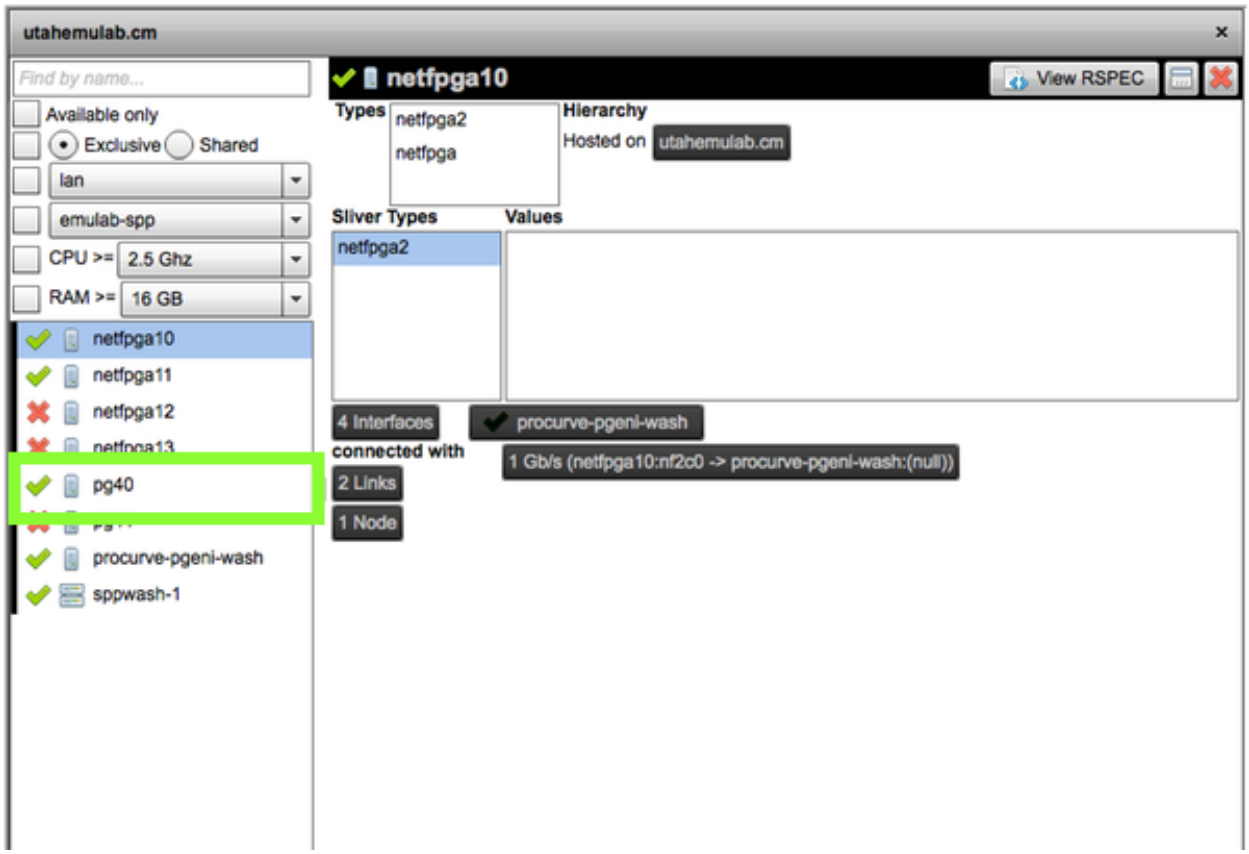
Clicking through, we finally find one available that has a 1 Gb/s connection. This is the type of node we'll want to use. The next step would be simply dragging that node to our project pane (along with any others that we need).

2. Getting on A Gig Backbone

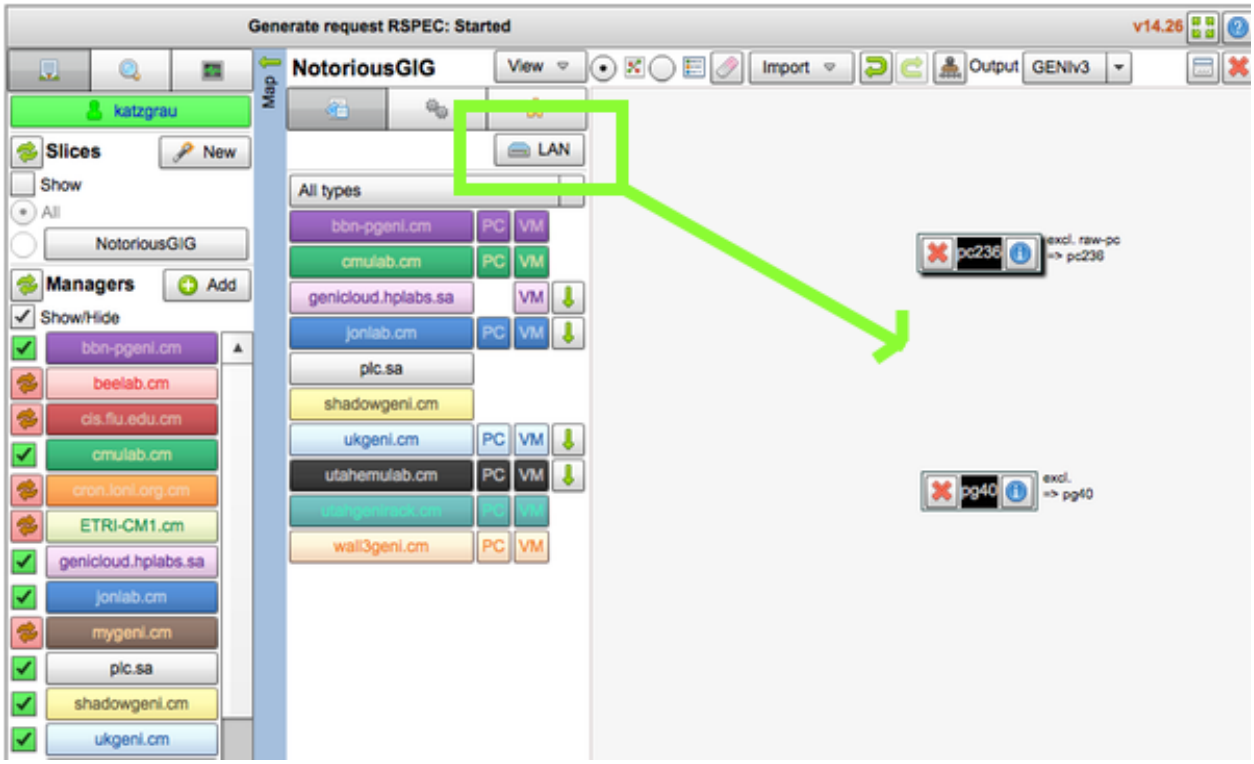
Now that we've got those gigabit interfaces available, we want to use them! If we're running a multi-site experiment, we need to find nodes on another site that is linked on a *gigabit backbone* to the first node(s) we selected.

Looking at our Flack map, we see that we have resources connected to out first node's aggregate by a pink/magenta line. In this case, we're focused on an aggregate in Washington DC.

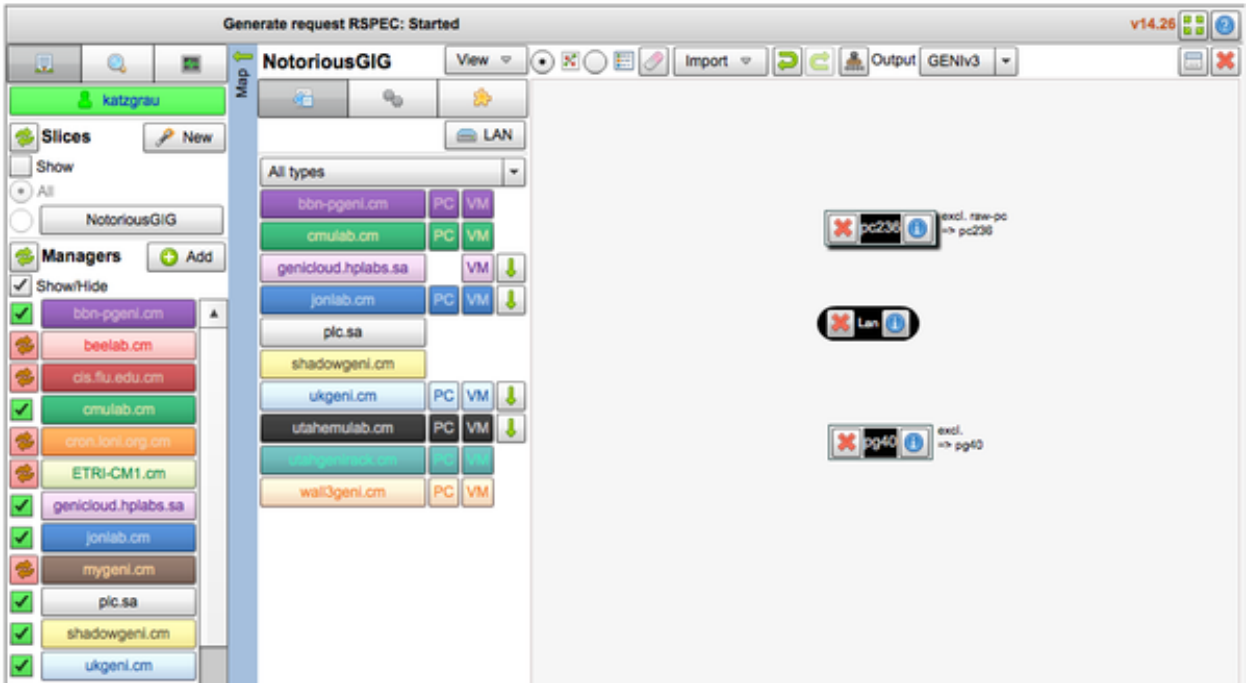
Clicking into into that aggregate, we again get a list of resources. Our goal is to find another machine with a gig interface.



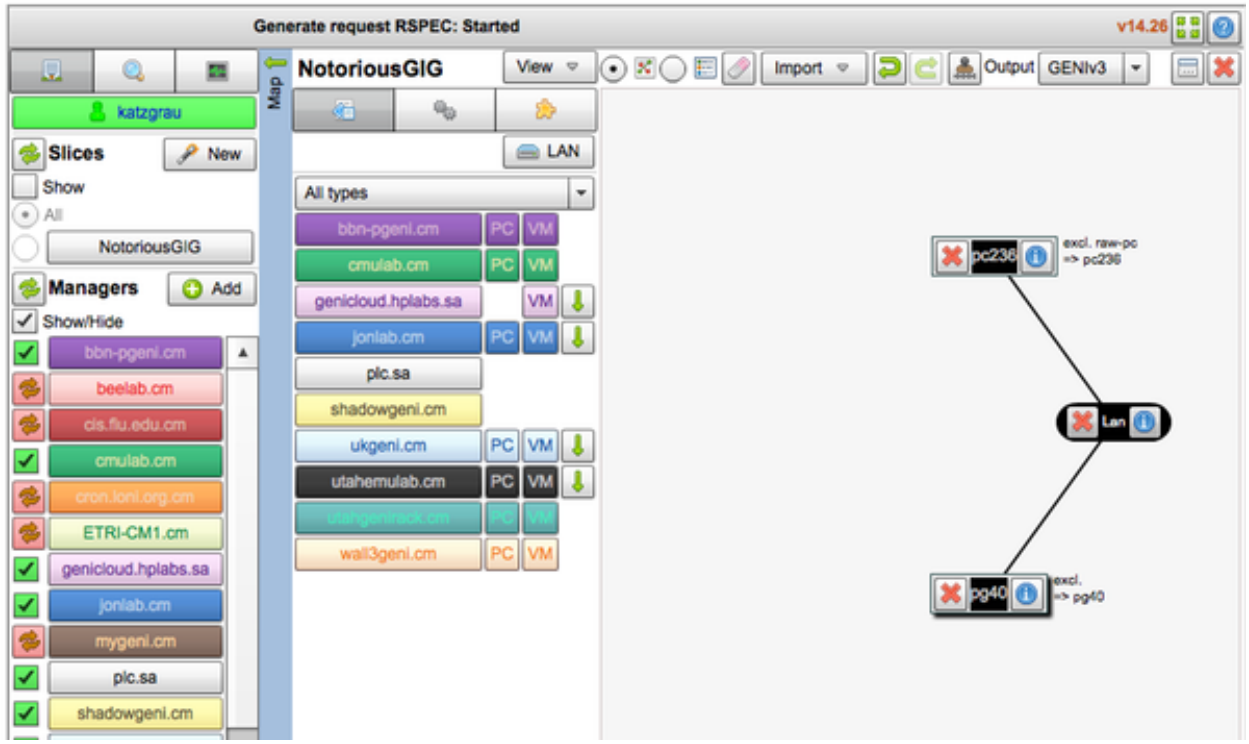
Now that we've found one, we'll drag that resource onto our map too, and create a network between them:



And:



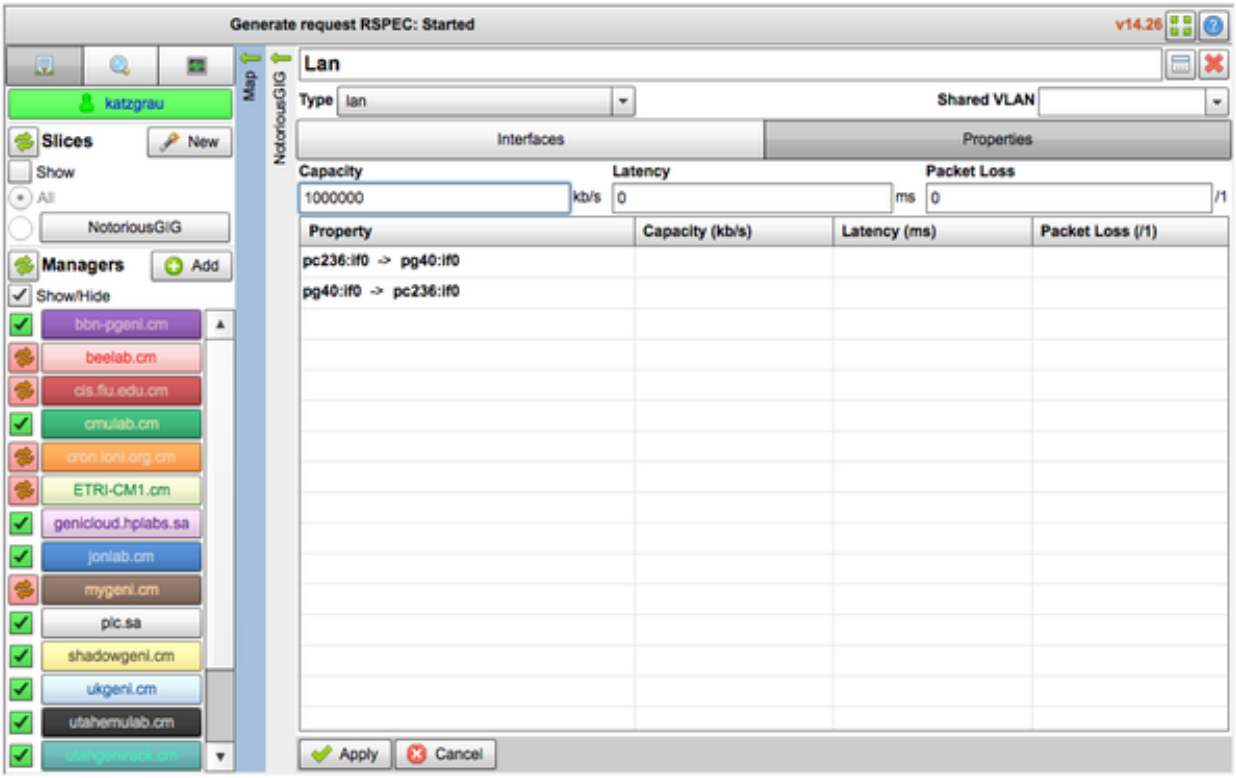
And:



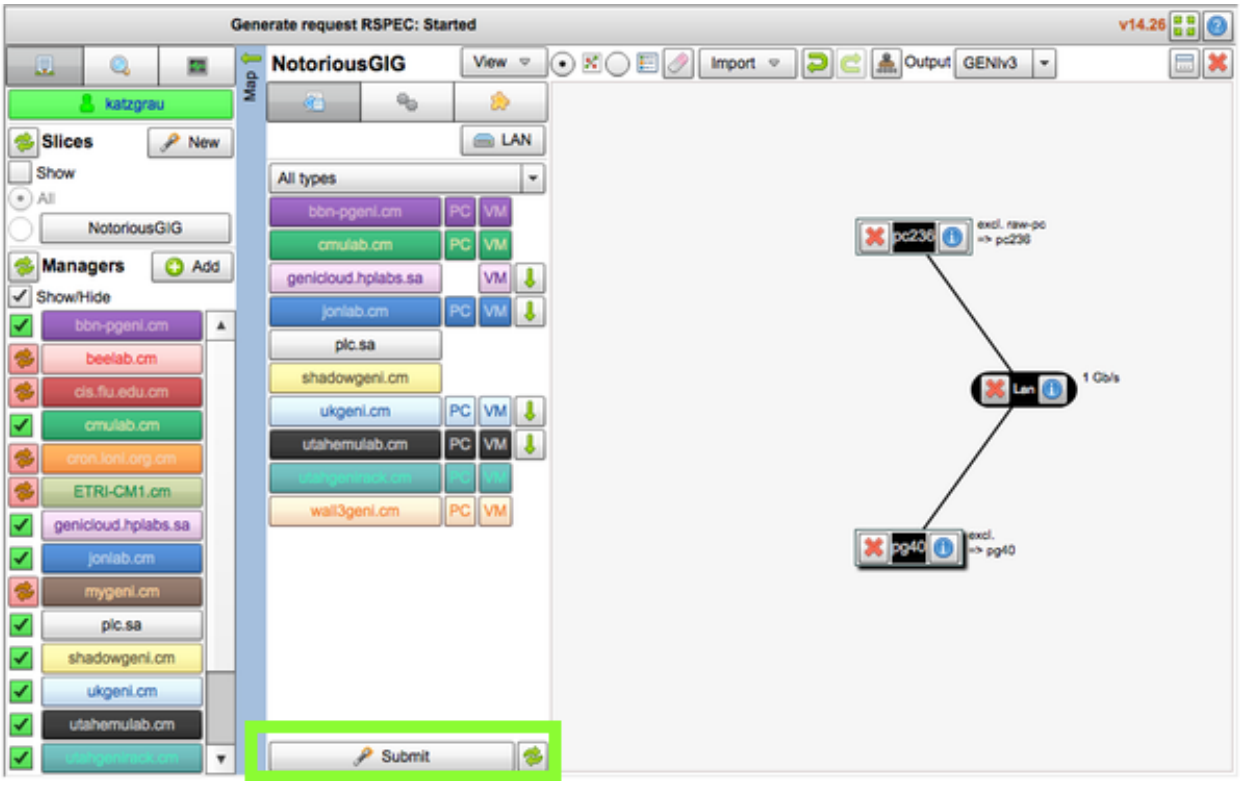
We're ready to finalize our gigabit network now.

3. Requesting the Right Bandwidth

Now that we've nearly got our network set up, we're ready to request gigabit speed between our nodes. We'll click the *information* icon on our lan. We'll see a pane that opens up. Click the "properties" tab. We want to specify that there should be a 1000000 kb/s connection on the network, like so:



When we click "Apply," our project will reflect our requested speed in the experiment pane:



You're now ready to submit your requested topology, and use the gig network.

CHAPTER 7

Programmable Networks

One of the most exciting aspects of GENI is that you are capable of experimenting with OpenFlow . OpenFlow is an open API that is implemented by some network hardware and software tools, allowing developers and network engineers define how their network is configured with **software**.

Contents

7.1 OpenFlow Basics

Note: This is a computer networking-focused document. It's expected that you have basic knowledge of computer network architecture as specified in the What You Need to Know Before Starting.

7.1.1 Introduction

There is sometimes a need for network architects (or developers like you) to define the behavior of their networks in a custom manner. For example, the architect may want a network switch where they can control how packets are routed, or even define a custom protocol.

Historically, this was possible via closed, proprietary hardware that can be prohibitively expensive or impossible to obtain by researchers and experimenters. Yet, the need for this functionality exists in order to run wide-scale projects implementing new, experimental protocols, or even routing rules for basic network architecture needs.

OpenFlow is an API that a growing number of switch manufacturers (like IBM, or HP and its Procurve switch[1]) are implementing, and it's the leading architecture for Software defined Networking (SDN)[2]. **It's an open API that allows external software to define the routing rules, or "control plane" of routing hardware.**

[1] http://www.openflow.org/wp/switch-hp/
[2] http://en.wikipedia.org/wiki/Software_Defined_Networking

OpenFlow is also implemented by Open vSwitch, a software-based switch you can install on an ordinary machine.

On the GENI network, you can request access to a physical OpenFlow-enabled Procurve, or simply install Open vSwitch on a machine and configure it from there.

Before going much further, you may want to know: What exactly can I do with OpenFlow?

7.2 Additional Resources

- HP ProCurve Product Page: http://www.openflow.org/wp/switch-hp/
- OpenFlow Whitepaper: http://www.openflow.org/documents/openflow-wp-latest.pdf
- OpenFlow API Spec: http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf

7.3 What Can I Do With OpenFlow?

OpenFlow is the leading architecture of Software Defined Networking (SDN). It allows you to alter the routing rules of a compatible switch by giving you access to the *control plane*, or the part of the switch that decides how traffic is handled.

Examples of what you might do with OpenFlow [3]:

- Implement a packet filter[4] (like a firewall)
- Implement policy based routing[5]
- Implement static routes[6]

Keep reading to find out how you can get started with OpenFlow on the GENI network.

7.4 OpenFlow on GENI

There are a few different ways to start experimenting with OpenFlow on the GENI network:

1. Install Open vSwitch on a standard GENI node
2. Request access to the GENI Mesoscale Infrastructure
3. Reserve one of the HP Switches in the GENI environment and install OpenFlow

[3] A blog post on potential industry use-cases: http://blog.ioshints.info/2011/11/openflow-enterprise-use-cases.html
[4] http://en.wikipedia.org/wiki/PF_(firewall)
[5] http://en.wikipedia.org/wiki/Policy-based_routing
[6] http://en.wikipedia.org/wiki/Static_routing

7.4.1 Installing Open vSwitch

The preferred method for experimenters looking to play with OpenFlow is option #1. Open vSwitch is a software package which you can install on an ordinary node and have it act as a switch which implements OpenFlow.

This is also the easiest option with the lowest barrier to entry. The third tutorial, Programming Networks with OpenFlow will walk you through the exact steps required to make this work.

7.4.2 Using the GENI Mesoscale Infrastructure

GENI's Mesoscale environment is a perfect place to run a large-scale OpenFLow experiment beyond a software-based option like Open vSwitch. Because this option requires coordination with the GENi Project Office (GPO), you may want to get you experiment running with Open vSwitch first, if that's an option.

From there, reach out to help@geni.net[7] for more information on obtaining a subnet for your experiment. A third-party, but detailed guide on how you might get your project running on the Mesoscale infrastructure is available on Github[8].

7.4.3 Reserve an HP Switch and install OpenFlow

This is an advanced option for getting up and running on OpenFlow. If you're ambitious enough to reserve HP OpenFlow hardware, or simply have the need, reach out to help@geni.net[9] for more information.

7.4.4 Additional Resources

- A guide on installing Open vSwitch: http://networkstatic.net/2012/06/openflow-openvswitch-lab/
- A guide on using GENI Mesoscale: https://github.com/aaronorosen/GeniTutorial/wiki/Geni-Mesoscale-Tutorial

[7] help@geni.net
[8] https://github.com/aaronorosen/GeniTutorial/wiki/Geni-Mesoscale-Tutorial
[9] help@geni.net

CHAPTER 8

Conclusion

At this point, you've been exposed to the most useful components of GENI for an app developer. The GENI platform is still being developed, however, and new services and tools for it are always being developed. It's time to point you toward additional resources that cover the dark corners of the platform, should you need them.

Contents

8.1 Additional Resources

Here's a list of resources that may be helpful to you as you explore new areas of GENI that may not of been discussed in this documentation.

- The Main GENI Wiki[1]: The documentation written by GENI experimenters and the GENI project office. This contains additional information on what other experimenters are using GENI for, and more projects for beginners.
- The Protogeni Wiki[2]: The documentation for Protogeni, a deployment of GENI. This contains both overlapping information in comparison with the Main GENI Wiki, and additional information as well.
- Open vSwitch Documentation[3]: The documentation for Open vSwitch, the software based switch used in the OpenFlow example project in these docs.

[1] http://groups.geni.net/geni
[2] http://www.protogeni.net/trac/protogeni
[3] http://openvswitch.org/support/

8.2 Additional Help

If you should need assistance, or have specific questions about GENI, send a note to the GENI Project Office at help@geni.net[4].

For issues with documentation, perhaps regarding something that is unclear, reach out to kenny@mozillafoundation.org[5].

[4] help@geni.net
[5] kenny@mozillafoundation.org

CHAPTER 9

Indices and tables

- genindex
- modindex
- search