Design/Setup: Setup Omni with Multiple Users

Omni gives you the capability of giving access to other users on your compute resources. Depending on which AM you are using to get resources from, this is done in a different way. If you are reserving resources in a ProtoGENI AM, then you can specify a comma separated list of users in the `users` attribute of the `[omni]` section, and specify the information for each user in a corresponding section. Let's try this now:

1. Ask the your neighbor for his/her usernane.
2. While in a terminal, generate a public key for them under `~/.ssh/` :
   - `cd ~/.ssh`
   - `ssh-keygen -f id_rsa_<neighbor;_username>`
3. Make a copy of your omni_config
   `cp ~/.gcf/omni_config ~/.gcf/omni_config_multiuser`
4. Open the omni_config_multiuser file; its under
   `~/.gcf/omni_config_multiuser`
5. Add the new user in the users list and add a new section using as his/her public key the key you just generated.

In the example below, Alice is reserving compute resources and wants Bob to also have access to the reserved ProtoGENI resources:

```
<snip>

[omni]
default_cf=portal
users=alice, bob
default_project=GEC16AdvNetw
#default_project=GEC16OpenFlowTutorial

[alice]
urn=urn:publicid:IDN+panther+user+alice
keys=/home/geni/.ssh/geni_key.pub

[bob]
urn=urn:publicid:IDN+panther+user+bob
keys=/home/geni/.ssh/id_rsa_bob.pub

[portal]
type = pgch
ch = https://panther.gpolab.bbn.com:8443/
sa = https://panther.gpolab.bbn.com:8443/
cert = /home/geni/.ssl/geni_cert_portal.pem
key = /home/geni/.ssl/geni_cert_portal.pem

....
</snip>
```

Next: Click Example Experiment

Click Router Example

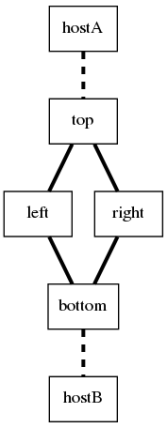
Experiment Description

In this example experiment, you will configure and run a non-IP software routing configuration, using :

- Click modular router
- ProtoGENI hosts

In this example, we'll be running click in user mode.

Please note that you can't just cut and paste all of the commands. There are additional instructions in the text.



Tutorial Instructions

- Part I: Design/Setup
  - Obtain Resources: Create a slice and reserve resources
- Part II: Execute
  - Configure and Initialize Services: Configure the Click Routers
  - Execute Experiment: Use custom routing to forward traffic over multi-path topology
- Part III: Finish
  - Teardown Experiment: Delete Resources

Click Example

1. Obtain Resources

In this step, we are going to setup the experiment. In this tutorial we assume that you are sufficiently comfortable with omni to verify that a **listresources** command works and to know when your slice is ready using **sliverstatus**.

i. **Create a slice**, where `<slicename>` is `click<initials>`:

```
omni.py createslice <slicename>
```

ii. **Create a sliver** :

```
omni.py createsliver -a pg-utah <slicename> <rspec_url>
```

- If you are doing this exercise as part of a tutorial then the `<rspec_url>` is: http://www.gpolab.bbn.com/experiment-support/ClickExampleExperiment/rspecs/click-XX.rspec where **XX** is given to you.
- If you are doing this exercise at home use : http://www.gpolab.bbn.com/experiment-support/ClickExampleExperiment/rspecs/click.rspec

iii. **Check the status of your sliver**

```
omni.py sliverstatus -a pg-utah <slicename>
```

Install scripts

While you wait for your sliver to become ready, we will see how we can automate the installation of our experiment with install scripts. In this experiment we are going to use software routers in order to write our own forwarding scheme. This means that in any experiment we are going to run we want the basic installation of the software router to always be present. The configuration might change from run to run, but the software should always be installed. The software to be installed, and the scripts to be executed at boot time, are defined in the rspecs. Follow these steps to locate your install script and identify the different parts.

i. Download the hellogeni rspec from http://www.gpolab.bbn.com/experiment-support/HelloGENI/hellogeni.rspec

```
cd /tmp
wget <rspec_url>
```

ii. Open your rspec and look for the `install` tag and copy the value of the URL attribute.
iii. Loof for the `execute` tag and write down the name of script to be executed
iv. Download and untar the software

```
cd /tmp
mkdir click
cd click
wget <software_url>
tar xvfz <software_name>
```

v. Look in your rspec and locate the `execute` tag. Note what script is being executed at boot time.
vi. Locate the script and open it. Can you identify the different parts?

Next: Execute

Click Example

1. Configure and Initialize Services: Configure the Click Routers

Once our sliver is ready we will go ahead and configure our click routers. In this example we have 4 routers, so instead of logging into each one of them and configuring it, we are going to use remote execution and configure them from our VM.

First lets reset our environment:

```
cd
rm .ssh/config
touch .ssh/config
```

1a. Login and remote execution

Run the `readyToLogin.py` script to get information about logging in to nodes. The script has a lot of output so lets put that in a file so that we can easily search for the information we want.

```
readyToLogin.py -a pg-utah <slicename> -o
```

This will save all the information to different files. We want to use the ssh configuration file that the script produced:

```
mv ./sshconfig.txt ~/.ssh/config
```

Let's login to our two hosts, the nicknames are `hostA` and `hostB`

i. Open two new terminals
ii. In one terminal type

```
ssh -A hostA
```

and in the other

```
ssh -A hostB
```

Test remote execution

You can execute commands in a remote host using `ssh`. There is an omni script, `remote-execute.py` that automates this.

i. In your local terminal type :

```
remote-execute.py -a pg-utah <slicename> -m "ls -a"
```

This will list all the files under the home directory of all hosts. To run it in only one host use the `--host` option

```
remote-execute.py -a pg-utah <slicename> -m "ls -a"  --host top
```

1b. Configure your routers

We are going to use remote execution to configure our routers. For this specific command we will use ssh directly.

i. On a local terminal run the following command four times, each time substituting the <router_nickname> with one of the top, bottom, left, right:

```
geni@geni-VirtualBox:~$ ssh -A <router_nickname> "/local/click-exampl
```

You'll get output something like this:

```
Your host information:
        hostA: hostA.<slicename>.emulab-net.emulab.net pc347.emulab.net
        top: top.<slicename>.emulab-net.emulab.net pc336.emulab.net
        left: left.<slicename>.emulab-net.emulab.net pc358.emulab.net
        right: right.<slicename>.emulab-net.emulab.net pc278.emulab.net
        bottom: bottom.<slicename>.emulab-net.emulab.net pc348.emulab.n
        hostB: hostB.<slicename>.emulab-net.emulab.net pc353.emulab.net
Done.
```

(If you are prompted for a password, check to make sure that you provided the -A switch in your ssh command above.)

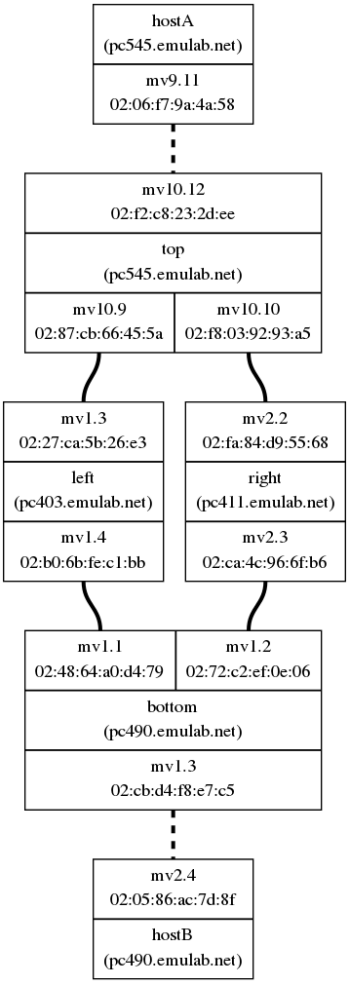
i. The extractClickConfig script produces router configurations for your experiment. It also creates a diagram of your experiment. Get a copy locally from one of the routers, by typing in a local terminal:

```
scp top:myslice.png ./
```

ii. View the diagram by typing :

```
eog myslice.png &
```

Your slice will look something like the one below. The overall configuration should be the same, with two end hosts, named hostA and hostB, and four routers (top, left, right, bottom) in a diamond configuration. The host names, interface names, and MAC addresses will be different, depending on the actual resources assigned to your slice.



The four routers interconnected by solid lines are your "core network," which will run a non-standard, non-IP protocol. The dashed lines out to the end hosts carry standard IP traffic.

1c. Turn off internet protocol

At this point, your network is still running IP. You can check by running a ping. In your **hosta** terminal window, run this command.

```
ping -c 3 hostb
```

The command should succeed, with output like this:

```
PING hostB-link-B (10.10.6.2) 56(84) bytes of data.
64 bytes from hostB-link-B (10.10.6.2): icmp_seq=1 ttl=61 time=1.38 ms
64 bytes from hostB-link-B (10.10.6.2): icmp_seq=2 ttl=61 time=1.19 ms
64 bytes from hostB-link-B (10.10.6.2): icmp_seq=3 ttl=61 time=1.53 ms

--- hostB-link-B ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 1.193/1.370/1.531/0.138 ms
```

Since our experiment doesn't want IP, let's turn it off :

i. On a local terminal run the following command four times, each time substituting the <router_nickname> with one of the top, bottom, left, right:

```
remote-execute.py -a pg-utah <slicename> -m "sh ./stopIP.sh"
```

You'll get output like this (the interface names may be different):

```
Disabling IP on interface mv10.9
Disabling IP on interface mv10.10
```

i. Verify that IP is really off, try another ping. On **hosta**:

```
  ping -c 3 hostb
```

The command should take twelve seconds to time out, then fail with output like this:

```
PING hostB-link-B (10.10.6.2) 56(84) bytes of data.

--- hostB-link-B ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 11999ms
```

2. Execute Experiment: Use custom routing to forward traffic over multi-path topology

2a. Start Click Routers

The extractor script produces a click configuration file for each of your routers.

i. On a local terminal run the following command four times, each time substituting the <router_nickname> with one of the top, bottom, left, right:

```
  remote-execute.py -a pg-utah <slicename> -m "sh ./startClick.sh"
```

You'll get output like this. (Don't worry about the warning messages, Click is just reminding you that you have no IP addresses in your core network.) The output of the click router is redirected to `/tmp/click.out` on each host.

```
Stopping any running Click routers
Starting Click router
top.click:34: While initializing 'FromDevice@18 :: FromDevice':
  warning: eth2: no IPv4 address assigned
top.click:35: While initializing 'FromDevice@21 :: FromDevice':
  warning: eth4: no IPv4 address assigned
```

Congratulations! You are now running a non-IP core network on your four routers, along with a (primitive) non-IP multipath routing algorithm. You're ready to experiment with this configuration.

2b. Send some traffic

Now you'll use your two edge hosts, **hostA** and **hostB** to send traffic along your network. Since these end hosts are not running your modified protocol, they'll rely on the **top** and **bottom** routers to transform their IP packets into your modified protocol on entry to the core network and back into IP packets on exit.

i. In your terminal window on **hostB**, instruct **nc** to listen for a UDP connection on port 24565 (or some other port that catches your fancy).

```
[mberman@hostb ~]$ nc -ul 24565
```

ii. Connect to it from your terminal window on **hostA**:

```
[mberman@hosta ~]$ nc -u hostb 24565
```

You've established a simple text chat connection. Enter a line of text in either window, and it should appear in the other. Of course to do this, the text is travelling through your core network, using your non-standard protocol and routing. So type a message into each window, and make sure it appears in the other.

That's it! Now, let's look inside to see what's going on.

2c. Looking under the hood

Please note: the interface names and MAC addresses below are for the sample configuration shown in the figure above. You will want to refer to your network diagram to get the correct interfaces and addresses for your configuration.

Let's take a look at what's happening in the four routers in your configuration. There are two basic router configurations. (You can find all of these files on any of your router hosts.)

Packet transformation

i. The more interesting configuration appears here, in the **top.click** configuration file. In a local terminal type:

```
 ssh -A top "cat top.click"
```

The output will look like :

```
// This portion accepts IP packets,
// reformats them, and routes them
// to an internal router.
route :: Classifier(27/01%01,-);

modify :: Unstrip(2) ->
    StoreData(0, "AliceWasHere3546") ->
    route;

FromDevice(eth3, PROMISC true) ->
    Classifier(12/0800) ->
    modify;

route[0] -> left :: EtherEncap(0x7744, 00:04:23:b7:14:76, 00:04:23:b7:1
    SimpleQueue ->
    Print(outL) ->
    ToDevice(eth2);

route[1] -> right :: EtherEncap(0x7744, 00:04:23:b7:1c:e0, 00:04:23:b7:
    SimpleQueue ->
    Print(outR) ->
    ToDevice(eth4);

// This portion accepts non-IP packets
// with an ether type of 0x7744
// from an internal router, restores
// them to IP format, and forwards.
restore :: SimpleQueue ->
    Strip(30) ->
    EtherEncap(0x800, 00:04:23:b7:14:77, 00:04:23:b7:20:00) ->
    ToDevice(eth3);

FromDevice(eth2) -> Classifier(12/7744) -> Print(inL) -> restore;
```

```
FromDevice(eth4) ->  Classifier(12/7744) -> Print(inR) -> restore;
```

As indicated in the comments, the top portion of the configuration listens (**FromDevice**) for IP packets arriving on the interface connected to **hostA** (that's **eth3** in this example). It then creates a new 16-byte field at the head of the packet (two bytes added by the **Unstrip** operation, plus the existing 14-byte Ethernet header. It fills that field with what could be important routing instructions, but in this case is just graffiti (**StoreData**). The **route** operation then routes the packet via either the **left** or **right** router toward **hostB**. In either case, it wraps the packet in a fresh Ethernet header (**EtherEncap**) with a distinctive ether type code (0x7744), logs the new packet on its way out (**Print**) and sends it out on the correct interface (**ToDevice**).

The bottom portion of the configuration is intended for packets coming out of the core network to **hostA**. It accepts packets from either the **left** or **right** router, logs them, strips off thirty bytes (Ethernet header plus your 16-byte new header field), puts on a fresh Ethernet header, and sends them along to **hostA**.

The configuration for the **bottom** router is exactly symmetric, routing packets between **hostB** and the core network, but using different graffiti.

Simple Forwarding

The **left** router configuration is much simpler. In a local terminal type:

```
ssh -A left "cat left.click"
```

The output will look like :

```
// Copy packets from top to bottom.
FromDevice(eth2) ->
    StoreEtherAddress(00:04:23:b7:42:b6, dst) ->
    StoreEtherAddress(00:04:23:b7:18:fb, src) ->
    SimpleQueue ->
    Print(top) ->
    ToDevice(eth3);
// Copy packets from bottom to top.
FromDevice(eth3) ->
    StoreEtherAddress(00:04:23:b7:14:76, dst) ->
    StoreEtherAddress(00:04:23:b7:18:fa, src) ->
    SimpleQueue ->
    Print(bottom) ->
    ToDevice(eth2);
```

This configuration just blindly forwards packets. It picks up any packet from the **top** router, updates the Ethernet header, and passes it along to the **bottom** router. The same applies in the reverse direction. Again, the configuration for the **right** router is exactly analogous.

Monitoring your core network

Let's watch how the packets travel through the network.

i. In a local terminal type:

```
  ssh -A top "tail -f /tmp/click.log"
```

ii. Go to your window for hostA, where your **nc** command is still running. Type a message into this window. You should see a log message in three of your four router windows. In this example, you might see:

iii. In the local terminal you will see:

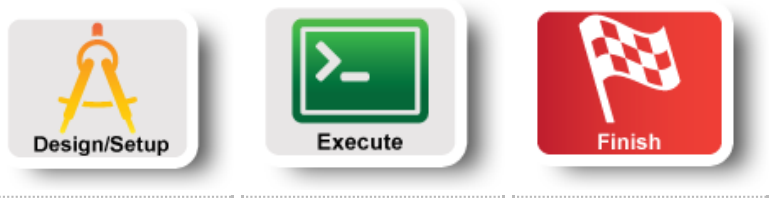
```
outR:   76 | 000423b7 192e0004 23b71ce0 7744416c 69636557 61734865
```

This log entry says that the **top** router received a packet from **hostA**, modified it, and sent it out to the **right** router. If the entry started with **outL**, that would indicate that it sent the packet out to the **left** router. Let's look a bit at the start of the packet (the first 24 bytes are logged). It starts with an Ethernet header. The first six bytes are the MAC address of the destination interface, that's 00:04:23:B7:19:2E, the MAC address of **eth4** on **right**. The next six bytes are the MAC address of the source interface, 00:04:23:B7:1C:E0, or **eth4** on **top**. Next comes your ether type, 0x7744. The remaining bytes, "416c 69636557 61734865" are the start of the first field in your new protocol, "AliceWasHe" in ASCII.

iv. Try typing a few different lines to hostA. You should see some packets routed to the left and some to the right. The routing decision is based on the **route :: Classifier(27/01%01,-);** entry in the **top** router configuration. Here, the router is looking at the low-order bit of the checksum on the initial IP packet (now at byte position 27 with the addition of the new sixteen byte field at the start of the header). Packets with odd checksums go to the left; those with even checksums go right.

Next: Teardown

ClickExample



Cleanup resources

Although all your reservations, have expiration times, you should always release your resources once you have completed your experiment to make them available to other experimenters.

i. Logout from your hosts.
ii. In the terminal, where you have been running your omni commands do:

```
omni.py deletesliver -a  pg-utah <slicename>
```