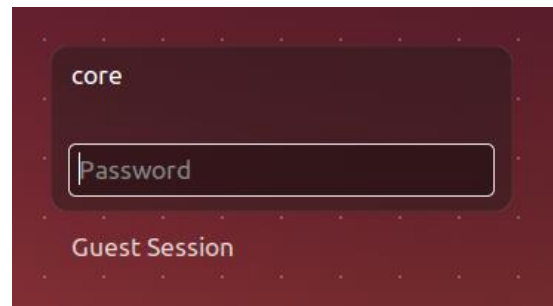
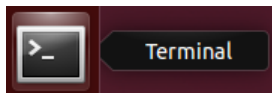


Index: (ctrl-click to jump to a specific section)

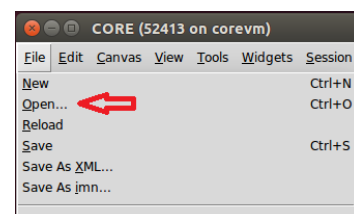
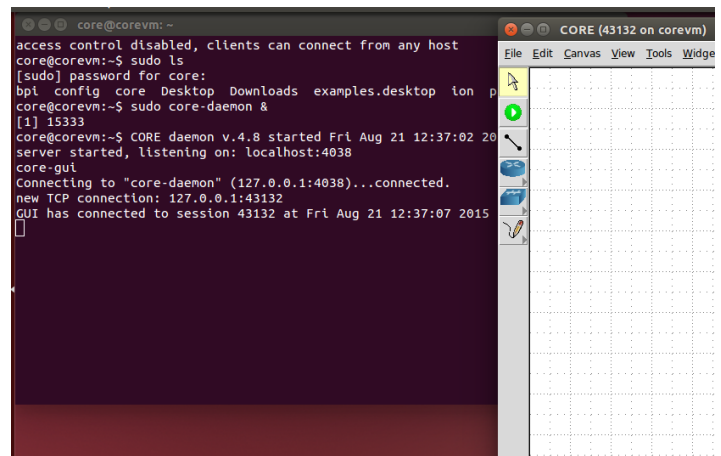
- 1) Installation and Setup
- 2) Bping Demo
- 3) Bpsendfile Demo
- 4) Included Scenarios
- 5) Pieces of a Custom Scenario
- 6) Building a Custom Scenario

1 – Installation and Setup

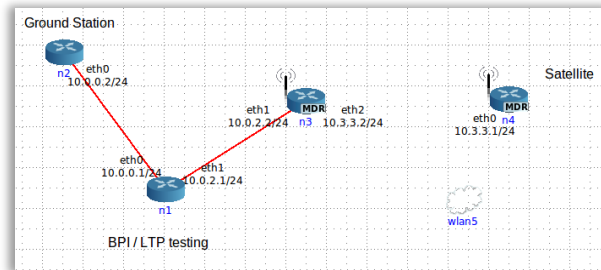
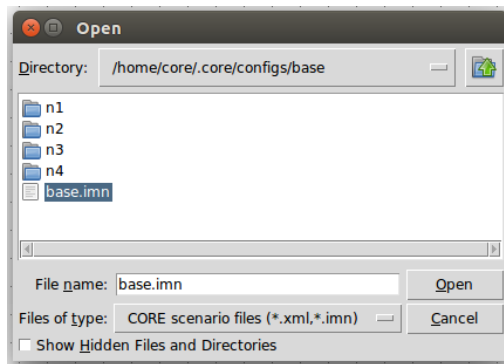
- 1) Download and install Oracle Virtual Box from <https://virtualbox.org/>
- 2) Unzip “coreVM.zip” on your host
- 3) Open the extracted folder and double-click “coreVM.vbox” to open the image
- 4) Log in as user “core” with the password “cvm”



- 5) Click the black rectangle icon on the toolbar to open a terminal
- 6) Type the command “sudo ls” and enter “cvm” when prompted for the password. Now that you’re authenticated, type “sudo core-daemon &” and hit enter. Type the command “core-gui”. You should see this window pop up:
- 7) Minimize the terminal so all you see is the CORE window
- 8) Click “File > Open”. This should open up a dialogue box:




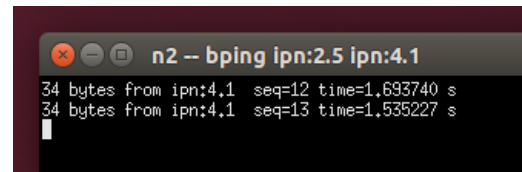
- 9) Double click the folder “base” and then the file “base.imn”



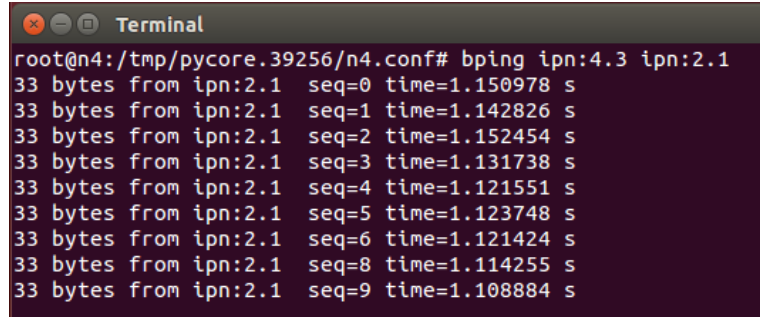
- 10) You should see a scenario come up on the screen that looks like the above ^

2 – Bping Demo

- 1) Open the base simulation as shown in part 1
- 2) Click the green button (“start the session”). This will start the network, the mobility script, and some passive BP programs on all of the nodes. Wait 10-20 seconds for the nodes to boot up and load ION. 
- 3) Drag the mobility script box out of the way. Your satellite node should be moving on it’s course by now. It’s on a 60 second lap, with 30 seconds connected, 30 disconnected. (NOTE: this will only run 30 loops, or 30 minutes. For anything longer, just stop the sim using the red button and restart it)
- 4) Bping is set to start automatically when the scenario starts, so wait about 20 seconds. An Xterm window will pop up, and just as the satellite is beginning to go out of range, the first ping packets will make it through. Bping will continue to make bundles while the satellite is out of range, and when it reconnects, will send them.
This will continue until you click on the xterm and hit “ctrl-c”. Do that when you’re ready to proceed to start up a ping manually.
- 5) To run a ping from the satellite to the ground station, double click the satellite node (n4). This will open up a terminal.

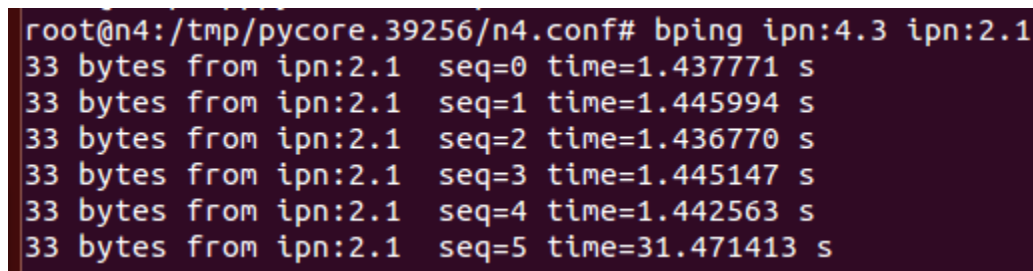


- 6) The satellite's IPN endpoint ID is 4 and the Ground station's EID is 2. Bpecho, the bping receiver, is running on ipn:x.1 for all nodes. Type "bping ipn:4.3 ipn:2.1" in the terminal and hit enter. You should see pings appear on

A terminal window titled "Terminal" showing the command "bping ipn:4.3 ipn:2.1" and its output. The output consists of ten lines, each showing "33 bytes from ipn:2.1" followed by a sequence number (seq=0 to seq=9) and a time value in seconds. The times are relatively low, around 1.1 to 1.5 seconds.

```
root@n4:/tmp/pycore.39256/n4.conf# bping ipn:4.3 ipn:2.1
33 bytes from ipn:2.1 seq=0 time=1.150978 s
33 bytes from ipn:2.1 seq=1 time=1.142826 s
33 bytes from ipn:2.1 seq=2 time=1.152454 s
33 bytes from ipn:2.1 seq=3 time=1.131738 s
33 bytes from ipn:2.1 seq=4 time=1.121551 s
33 bytes from ipn:2.1 seq=5 time=1.123748 s
33 bytes from ipn:2.1 seq=6 time=1.121424 s
33 bytes from ipn:2.1 seq=8 time=1.114255 s
33 bytes from ipn:2.1 seq=9 time=1.108884 s
```

the screen as they complete the trip. (Note: when the satellite is on its 30 second period of disconnect, the pings will not go through. The satellite will issue ping requests, and you'll see them all come in when the satellite reconnects. See picture below, notice that seq=5 has a time of 31 seconds. This is because it was sitting in the outbound queue while the satellite was out of range for 30 seconds and then had a 1.5 second round trip. In addition, ION takes about 25 seconds from when you press the green button to start up. You can start the ping whenever you want, and it'll start responding as soon as it can.)

A terminal window showing the command "bping ipn:4.3 ipn:2.1" and its output. The output shows six lines. The first five lines have times around 1.4 seconds, but the sixth line (seq=5) has a significantly longer time of 31.471413 seconds, indicating a long delay in the network.

```
root@n4:/tmp/pycore.39256/n4.conf# bping ipn:4.3 ipn:2.1
33 bytes from ipn:2.1 seq=0 time=1.437771 s
33 bytes from ipn:2.1 seq=1 time=1.445994 s
33 bytes from ipn:2.1 seq=2 time=1.436770 s
33 bytes from ipn:2.1 seq=3 time=1.445147 s
33 bytes from ipn:2.1 seq=4 time=1.442563 s
33 bytes from ipn:2.1 seq=5 time=31.471413 s
```

- 7) When you're finished, click into the terminal window that's running the ping and hit Ctrl-C twice to stop the bping.
- 8) You can ping between any two nodes by opening a command line (double clicking on the source node that you want) and typing "bping ipn:X.3 ipn:Y.1". X is the EID of the source node, Y is the EID of the destination node. 3 is the next open "port" as .1 is an echo program and .2 is a file receiver program.

3 – Bpsendfile Demo

- 1) Open the base simulation as shown in part 1
- 2) Click the green button ("start the session"). This will start the network, the mobility script, and some passive BP programs on all of the nodes.



- 3) Drag the mobility script box out of the way. Wait 10-20 seconds for the nodes to boot up and load ION.
- 4) Open a terminal on node 4 by double clicking on it. Type in the command “bpsendfile ipn:4.3 ipn:2.2 README”. This will package the file README into a bundle (or if it’s a bigger file, multiple bundles) and send it to the bprecvfile program running on node 2. That argument can be replaced by whatever filename you want.

- 5) You should see this message when bpsendfile

```
root@n4:/tmp/pycore.59387/n4.conf# bpsendfile ipn:4.3 ipn:2.2 README
Stopping bpsendfile.
root@n4:/tmp/pycore.59387/n4.conf#
```

is finished sending. (Note: if the satellite is disconnected at the time of sending, the file will not be delivered immediately after bpsendfile is finished. If you use the “bplist” command, you can see that the bundle containing the README is in the queue waiting for delivery).

- 6) Open a terminal on the receiving node (N2). Type “ls” and hit enter. You should see “testfile1” among the output. That’s the file that you just sent to this endpoint.

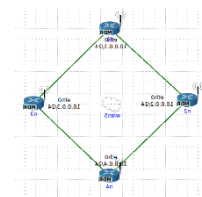
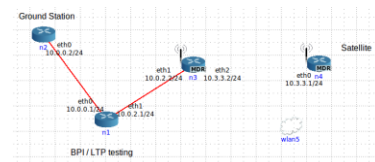
```
startsshd.sh
testfile1
usr.local.etc.c
```

- 7) Type “cat testfile1” and hit enter. You should see the text of the file README on node 4.

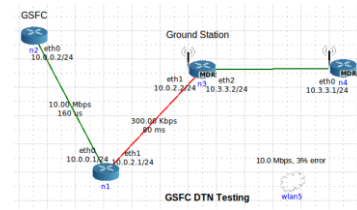
4 – Included Scenarios

There are a number of pre-built scenarios included with the Core VM. Each one has a short explanation and picture below.

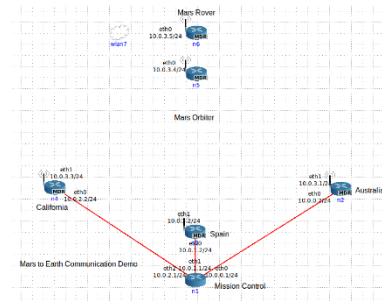
- Base: (Automatic Bping demo)
 - Standard model, 4 nodes. Mission control, link, ground station, and satellite
 - Topology: line of 4, one wireless
 - Services: bpecho on x.1, bprecvfile on x.2 (standard for all scenarios)
- Diamond:
 - Topology: 4 node ring, all wireless
 - Connection alternates between N2-N4 and N3-N4 every 30 seconds for 30 min



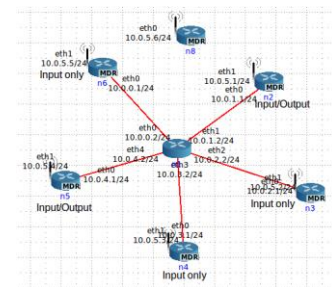
- GSFC: (Automatic BPI demo)
 - Mockup of the Goddard Space Flight Center. High bandwidth down from the satellite but low bandwidth from ground to MCC
 - Topology: line of 4, all wired



- Mars:
 - Simulation of a mars lander taking to earth via orbiter and DSN
 - Topology: 3 DSN nodes wired to mission control. The orbiter can talk to the rover and to the DSN nodes one at a time wireless.

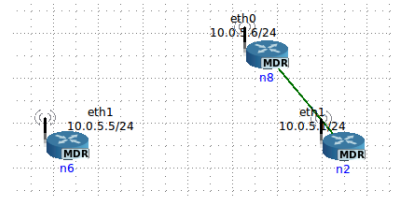


- Miss:
 - A modified version of planet
 - The satellite misses the first, third, and all other odd numbered connections to node 5.

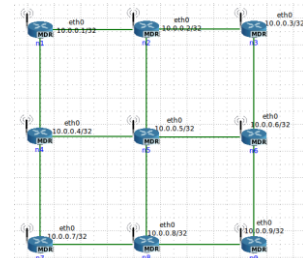


- Planet:
 - Topology: planetary network is a star. 5 ground stations wired to MCC. One satellite orbits and connects to ground stations over wireless.
 - Only nodes 2 and 5 have uplink. All ground stations have downlink.

- Small:
 - Topology: two nodes and one messenger
 - The messenger moves back and forth shuttling bundles and providing a connection



- Square:
 - Topology: 9 node mesh, 3x3 grid connected by wireless
 - Used for routing algorithm and ECOS testing



5 – Pieces of a Custom Scenario

Before you can build custom scenarios, you need to know the various pieces of the puzzle and how they fit together. (Note: the following files have the syntax \$name and \$x. \$name is the name of the scenario (like base or planet) and \$x is the node number.

- \$name.imn: The CORE save file containing the physical layout of the nodes, links, services, mobility scripts, and hooks (scripts that CORE runs at startup and shutdown.) This file is made by the drag-and-drop interface in core-gui.
- \$name.scen: The mobility script file. Any time a node needs to physically disconnect or reconnect to a network, it must be scripted in this file. You can set the initial positions of nodes at the start and you send them movement commands at certain times. There's more on how to actually make this file below. This file is not necessary if there is no mobility.
- mobility.sh: A script that sets the file /tmp/ionrcdate to the correct startup time. The nodes use this file to set their 0 time. This script should be run when the mobility script starts up (which should be set to 0.0 seconds after the scenario starts up.) This will ensure accurate synchronization between all time-dependent programs. One of the pre-runtime hooks should delete any old copies of this file. This file is not necessary if there is no mobility.
- generalSetup.sh: General CORE node setup. Links in all files from the respective n\$x directories. Also runs n\$x/nodeSetup.sh which starts up the node's software.
- n\$x directories (one per node, corresponding to the node numbers): Individual node folders
 - Things you don't need to modify:
 - ionstart.ipn: startup script for ION
 - ionstop.ipn: shutdown script for ION
 - node\$x.acsrc, .cfdp, .ionsecrc, and .ionconfig: All default settings that can stay the same. For advanced scenarios, these may need to be edited.
 - node\$x.bprc: bpradmin configuration file. Configures the endpoint, protocols, it's inducts and outducts, and has an option to turn on watch characters to help diagnostics.
 - node\$x.ionrc: ionadmin config files. Sets up routes and contacts for all nodes. This is where you can specify static or dynamic links.
 - node\$x.ipnrc: config file that maps contacts to protocols and to physical routes.
 - node\$x.ltp: ltpadmin config file. Sets up LTP routes and gives them the UDP address to point at.

- rcgen.sh: A script that generates the node\$x.ionrc file. Use this when you have some kind of mobility that loops on a regular period. rcgen.sh is synced with the mobility inside CORE so that it will set the 0 time to when the mobility script started.
- prc: Preset Routes and Contacts file. Any static routes/contacts in the scenario should be hard coded here, as this is copied directly into the dynamic ionrc file when rcgen.sh runs.
- nodeSetup.sh: Setup commands and scripts go here. If you're running rcgen.sh, call it here under the setpath line. Use `"/rcgen.sh $x"`. This file launches startION.sh
- startION.sh: calls out to ionstart, passing in all of the config files defined above. In addition, starts utilities that depend on ION. The default in the scenarios included in this VM are running bping on ipn:\$x.1 and bprecvfile on ipn:\$x.2. (Note: both this script and nodeSetup.sh have sleep 10's. This means starting up ION does take about 20-30 seconds from green button startup.)

6 – Building a Custom Scenario

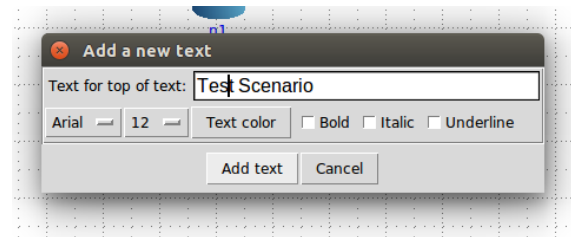
The next section goes over how to take a base scenario and transform it into a scenario that better fits your own mission. It also goes over some instructions for building scenarios from scratch, but that's more time-intensive than modifying an existing scenario.

1. Go to the ~/core directory and copy over an existing scenario using `"cp -r base new && cd new && ls"`. You should see something like this:

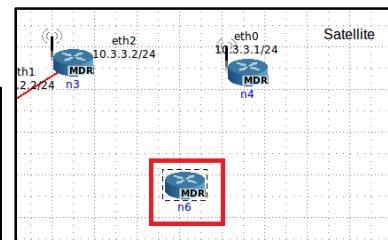
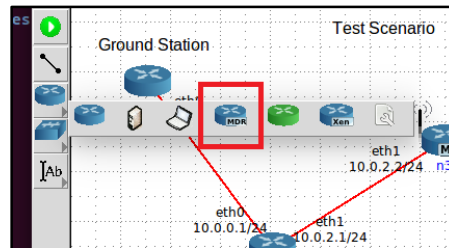
```
core@corevm:~/core$ cp -r base new && cd new && ls
30-30.scen  base.imn      mobility.sh  n2  n4  rcgen.sh
60-60.scen  generalSetup.sh n1          n3  prc  README
core@corevm:~/core/new$
```

2. You can remove the files README and 60-60.scen. Neither of these will be used.
3. Rename base.imn with `"mv base.imn new.imn"` and the scen using `"mv 30-30.scen new.scen"`
4. Open up a new terminal and open up a GUI. Load up your new scenario. We're going to add in another wireless node, add in a wired route, and change the mobility scripting.

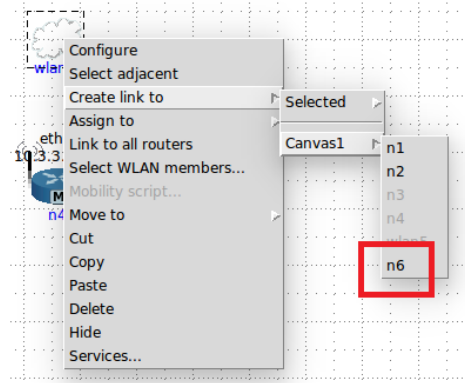
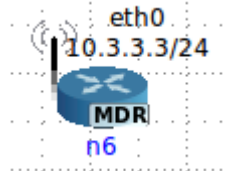
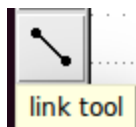
- Remove the “BPI...” heading and click the pencil button, then click the AB text button. Click anywhere to put a heading onto the field. Type “Test Scenario” or any other title, then click “add text”. You can change your labels’ color, font, size, and more with this box.



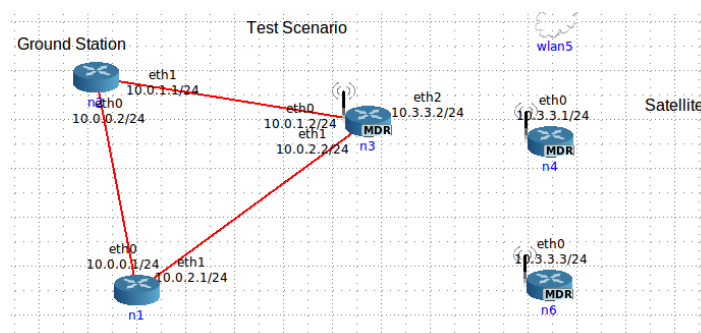
- Click back to the pointer button at the top and you can click and drag your labels around. Double click anything on the GUI to open up its configuration box.
- Drag your wlan5 object and put it above the satellite node to get it out of the way.
- Click the router menu and click MDR router. Click under the satellite to place one of these on the field.



- Create a link from wlan5 to the new node 6 by right clicking the wlan, selecting “Create link to”, selecting “Canvas1”, and then selecting “n6”. You should see an interface pop up and a little antenna on the node:



- Now to put in a wired route in addition. Click the link tool on the sidebar and then click and drag from node 2 to node 3. (Note: make sure to go 2->3 and not 3->2. Normally it doesn’t matter,



but for consistency with this guide, it does. The IP addresses change if you drag the link a different way.) Your scenario should look roughly like this:

11. Node 6 now needs to run ION. First, click “File” and save your scenario to new.imn. Go back to your terminal and go to the “~/core/new” directory. Make a directory for node 6 using the command “cp -r n4 n6 && cd n6 && ls”.
12. Observe that all of the files are node4.xyz. Rename all of them to node6.xyz using “mv”

```
core@corevm:~/core/new/n6$ ls
curiosity.png  node6.bprc      node6.ionsecrc  prc
ionstart.ipn   node6.cfdprc    node6.ipnrc     rcgen.sh
ionstop.ipn    node6.ionconfig node6.ltprc      rover.png
node6.acsrc    node6.ionrc      nodeSetup.sh     startION.sh
```

13. Open ionstart.ipn in your text editor of choice and change BASE_NAME to = “node6”. Also change the two echo statements to output the correct ipn: address. ionstop.ipn has echoes that can be fixed as well.
14. Open nodeSetup.sh and change the argument of rcgen.sh. In addition, add a new static route for the satellite:

```
./rcgen.sh 6
ip route add 10.3.3.2/24 dev eth0
ip route add 10.3.3.1/24 dev eth0
```

15. Change the nodeSetup.sh of node 3 and 4 as well (vi ../n3/nodeSetup.sh) to have a static route to each of the other two wireless nodes with the same syntax.
16. Edit startION.sh. There are a number of references to node4.xyz files in the line with “ionstart”. Change those to 6, and change the bping and bprecvfile lines to ipn:6.x.

```
ionstart -a node6.acsrc -b node6.bprc
sleep 10
bpecho ipn:6.1 >& bpecho.log &
bprecvfile ipn:6.2 &
```

17. Now we need to edit the network configuration files for all of the nodes. This is the most time intensive part of custom building. For each node we may need to modify the bprc, the ipnrc, and the ltprc. Then we’ll go change all of the ionrc’s at once by editing rcgen.sh. Node 1’s config files stay the same because it doesn’t have any new neighbors. Run “cd ../n2” to get to the node 2 directory. That’s where we’ll start.

18. Edit node2.bprc and go down to the TCP outduct section. Put in a line at the bottom of the section saying “a outduct tcp 10.0.1.2:4556 tcpclo”.

```
a outduct tcp 10.0.0.1:4556 tcpclo
a outduct tcp 127.0.0.1:4556 tcpclo
a outduct tcp 10.0.1.2:4556 tcpclo
```

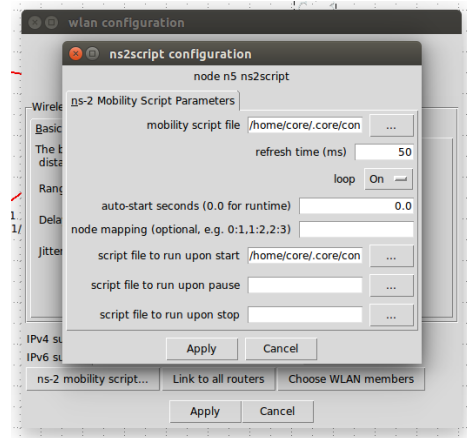
19. Save, exit, and edit node2.ipnrc. Add in the line “a plan 3 tcp/10.0.1.2:4556” at the bottom. These two lines tell node 2 how to send traffic to node 3. Since this is a wired connection, we’re using TCP and there’s no need to configure the ltprc.

```
a plan 1 tcp/10.0.0.1:4556
a plan 2 tcp/127.0.0.1:4556
a plan 3 tcp/10.0.1.2:4556
```

20. Change to the n3 directory. Put an outduct to node 2 into the bprc and a plan for node 2 into the ipnrc. In addition, we now need to add an outduct over LTP to node 6, a plan, and a line in the ltprc: “a outduct ltp 6 ltpclo” (bprc), “a plan 6 ltp/6” (ipnrc), and “a span 6 100 100 64000 100000 1 'udplso 10.3.3.3:1113 40000000' ”. There are a lot of numbers in the ltprc line, but the important ones are “span 6” and the udplso IP address. The rest can stay as the defaults.
21. Change node 4’s bprc, ipnrc, and ltprc to add in the same lines.

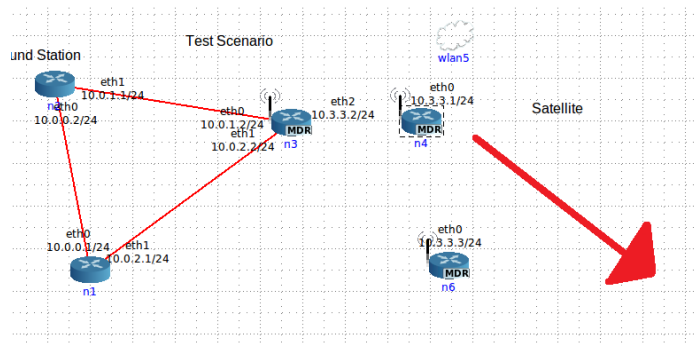
22. Go to the n6 directory, and then set up the bprc, the ionrc, and the ltprc. You need outducts to n3 and n4, plans to both over ltp, and udplso commands all using the same syntax as above. Make sure to change the endpoint ID’s to ipn:6.x in the bprc!

23. Now the only file left to configure is the ionrc. Before we do this, we need to write a mobility script. Double click the router and click the “ns-2 mobility script” button at the bottom. A window will pop up with a number of settings.



24. Make sure “mobility script file” is pointed at new.scen in the correct directory, and make sure “script file to run upon start” is set to our copy of mobility.sh. These are both picker menus, so just hit the “...” button and navigate to the file. In addition, make sure the “auto-start seconds” field is set to 0.0 so the mobility script runs on startup.

25. For this scenario, I want n4 to always have a connection to n6, but only sometimes have a connection to n3. (n6 functioning as a deep space relay). When n4 is in range of n3, it should take the shortest path and send data directly to n3.



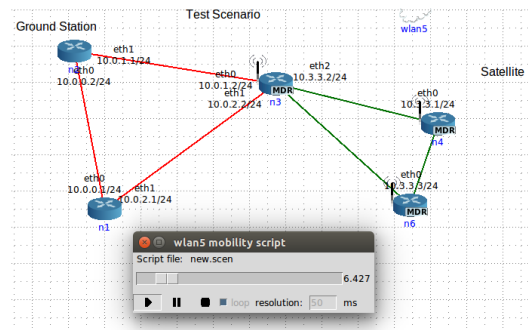
However when it is out of range, it should send data through n6 to n3 and beyond. Thus, I want node 4 to start close to n3 and then slowly start moving away, always staying in range of n6.

26. Edit new.scen. First we need to find the starting position of n4. Mouse over the node and the position will appear in the bottom left corner of CORE. It’ll be different for everyone, but mine is 525, 150. Put that into the first two lines of the .scen: “\$node_(4) set X_ 525.0”.

27. Now we decide what the total loop time will be. I'm picking an arbitrary 60 seconds. This means 30 seconds travelling out, 30 seconds back. To move a node, you use this line: "\$ns_ at 00.00 "\$node_(4) setdest 775.0 350.0 12.0". Translation: at \$time, \$node should move towards \$x, \$y at \$speed. I picked (775, 350) because that's just about as far as it can go in N6's range, and I picked 12 as an arbitrary speed that I think should get the node to its destination in 30 seconds.

28. At 30 seconds, the node needs to start coming back to the original position. Copy the first movement line and then input the original coordinates, (525, 150). Set the time to 30.00. Once you enter this line, copy it and put it at time 60.00. This is repetitive, but it will force the script to run for 60 seconds and then loop itself. We then use this when we create our rcgen.sh as we can track everything in 60 second intervals.

29. Save your mobility script and then hit the green button and make sure it all goes as planned. For 30 seconds, the node should move outwards, and then it should come back to its original position by 60 seconds.



30. Now that you have a good script, you need the times that the link from n4 to n3 dies and when it reconnects. Start the mobility script, and when the node is on its way out, watch for the last second the connection is valid. Ex: if the connection breaks at 8.5 seconds, you'll want to write down 8. The opposite on the way back in. If it restores at 47.5, you'll want to write down 48. There's a fraction of a second lost by doing this, but it allows some room for error in case nodes get nudged out of correct alignment.

31. When you have the times, go to the "new" folder and find the master copy of rcgen.sh (not the copies in the node folders!) This is the master copy, so edit it. Skip past the time stuff until you get to the "#set other vars" section. Leave start at 0, and set stop to 8 (since the first connection goes from the start of the script to 8 seconds). In addition, make two more variables start2 and stop2, and set them to 48 and 60 respectively. Make sure looptime is still set to 60, and set numloops anywhere from 20-60, depending on how long you want this to keep going.

```
#set other vars
looptime="60"
numloops="30"
bps="500000"
premaes="prc"
start="0"
stop="30"
start2="48"
stop2="60"
thisnode=$1
outfile='node'$th
```

32. When your parameters are set, skip down to the FOR loop at the bottom. This is where the ranges and contacts are generated. There is already one basic echo set, so copy it and paste right below. Each set should have 5 lines. Change the “start” and “stop” variables in the second set to “start2” and “stop2”, and leave the “3 4, 4 3” structure alone. This says that there is a range from node 3 to node 4 and node 4 to node 4 at the following times. To make a 1 directional connection, just comment out the line going in the blocked direction.

```
((base=($looptime*$i)))
echo "a range +${($start + $base)} +${($stop + $base)} 3 4 1" >> $outfile
echo "a range +${($start + $base)} +${($stop + $base)} 4 3 1" >> $outfile
echo "a contact +${($start + $base)} +${($stop + $base)} 3 4 $bps" >> $outfile
echo "a contact +${($start + $base)} +${($stop + $base)} 4 3 $bps" >> $outfile
echo "" >> $outfile
echo "a range +${($start2 + $base)} +${($stop2 + $base)} 3 4 1" >> $outfile
echo "a range +${($start2 + $base)} +${($stop2 + $base)} 4 3 1" >> $outfile
echo "a contact +${($start2 + $base)} +${($stop2 + $base)} 3 4 $bps" >> $outfile
echo "a contact +${($start2 + $base)} +${($stop2 + $base)} 4 3 $bps" >> $outfile
echo "" >> $outfile
```

33. Once you have this, save and exit that file and open up prc. There are premade ranges and contacts for all of the nodes that were here already, but we have to write in the links we added. Following the syntax, add a range/contact for nodes 2 <--> 3, 3 <--> 6, and 4 <--> 6.

```
# Node 1 to 2
a range +0 +4000000 1 2 1
a range +0 +4000000 2 1 1
a contact +0 +4000000 1 2 500000
a contact +0 +4000000 2 1 500000

# Node 6 to 3
a range +0 +4000000 6 3 1
a range +0 +4000000 3 6 1
a contact +0 +4000000 6 3 500000
a contact +0 +4000000 3 6 500000

# Node 6 to 4
a range +0 +4000000 6 4 1
a range +0 +4000000 4 6 1
a contact +0 +4000000 6 4 500000
a contact +0 +4000000 4 6 500000
```

34. Save this file and then copy prc and rcgen.sh into each of the node folders. (“cp prc n1/”, etc)
35. The last thing to do is to tell CORE that the new node we put in should be running ION. With the simulation *not* running, double click n6. Click “services”, and a giant window appears. Look for “UserDefined” in the Utility column at the bottom. Click it so it is selected and then click the wrench icon beside it.

36. Under “File name:” put “setup.sh”. Click “Use text below...” and in the big text box, type:
- ```
foo=`dirname $SESSION_FILENAME`
sh $foo/generalSetup.sh
```

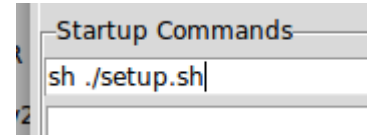
Use text below for file contents:

```
foo=`dirname $SESSION_FILENAME`
sh $foo/generalSetup.sh
```

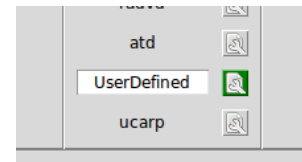


37. Click the white page with the star icon next to the File name box.
38. Go to the Directories tab and click the same page icon. Select /var/ion and hit “ok”

39. Go to the Startup/shutdown tab and type “sh ./setup.sh” into the Startup Commands box. Hit the white page with the star again, and you should see this:



40. Hit “Apply”. Notice that the box around the wrench icon is now green. That means this service has a custom configuration. Again, check that the UserDefined service has a white box around it, meaning it’s turned on. Hit “Apply” twice.



41. Everything should be set up! Hit the green button and test it out. Wait 20-30 seconds as usual and then try to ping between nodes.

Other important things:

- To modify hooks, go to the Session menu and there’s an option for Hooks. If you go there on base or the new sim, you can see the mobility time sync hook.