DTN Development Kit

V1.2

Table of Contents

1 Purpose

The DTN development kit provides mission designers with a set of tools to evaluate the potential benefits of using the DTN protocol suite in their missions. The main components of the kit are:

- A set of installers for the NASA Interplanetary Overlay Network (ION) software. These installers can be used to instantiate ION nodes (DTN routers) on virtual or physical machines.
- A set of pre-built scenarios running in an emulation environment that use the DTN protocol suite for communications. These scenarios can be used to demonstrate how the DTN protocols can function for various missions, and can be adapted to more closely emulate a particular mission. The configurations from the pre-built scenarios can also be extracted and run on physical nodes for increased performance.

2 Background

2.1 Delay / Disruption Tolerant Networking (DTN)

Delay / Disruption Tolerant Networking (DTN) is best described as a suite of protocols (akin to the Internet suite of protocols) capable of delivering data when the end-to-end path is NOT necessarily contemporaneously connected.

The ipnsig.org website has some good introductory material on DTN, including a Tutorial and a Primer.

Figure 1 shows the Bundle Protocol suite:

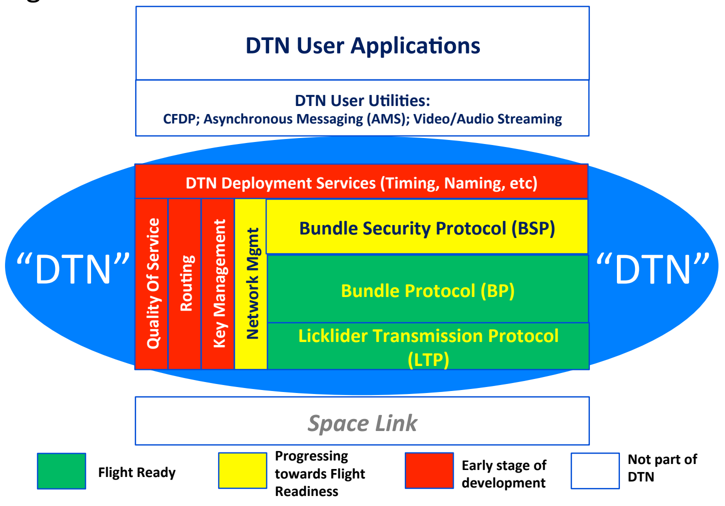

DTN User Applications
DTN User Utilities:
CFDP; Asynchronous Messaging (AMS); Video/Audio Streaming
DTN Deployment Services (Timing, Naming, etc)
"DTN"
Quality Of Service
Routing
Key Management
Network Mgmt
Bundle Security Protocol (BSP)
Bundle Protocol (BP)
Licklider Transmission Protocol (LTP)
"DTN"
Space Link
Flight Ready
Progressing towards Flight Readiness
Early stage of development
Not part of DTN

Figure 1: The Bundle Protocol Suite of protocols.

Figure 2 shows how the Bundle Protocol functions end-to-end.

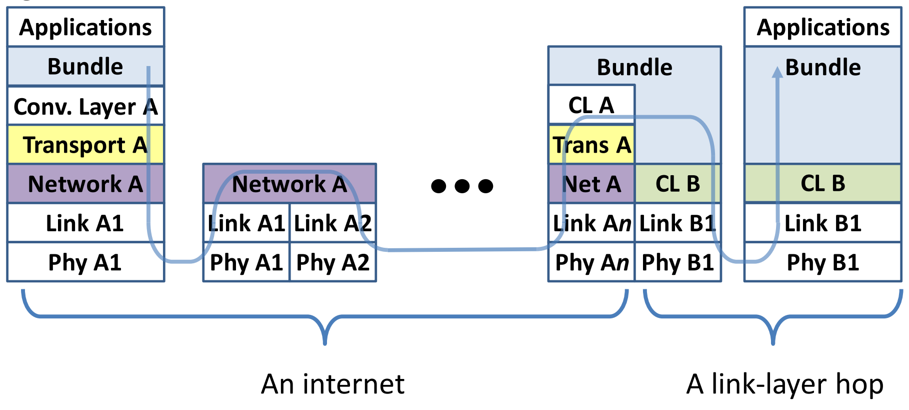


Figure 2: End-to-end data flow with the Bundle Protocol.

Note that the Bundle Protocol's function in the end-to-end drawing is very similar to the end-to-end function of the Internet Protocol (IP) when running over a number of data link layers in an end-to-end path.

2.2 Bundle Protocol Specifications and Implementations

2.2.1 BP Specifications

The base Bundle Protocol is described in the Bundle Protocol Specification [RFC5050], and was developed by the Internet Research Task Force (IRTF) Delay Tolerant Networking research group (dtnrg). The dtnrg also specified a reliable 'shim' for use above unreliable data links in the Licklider Transmission Protocol [RFC5326].

The Consultative Committee for Space Data Systems [CCSDS] has produces *profiles* of the BP [CCSDS BP] and LTP [LTP for CCSDS] specifications and standardized them for use by CCSDS member agencies. The profiles are interoperable with the base specification, and include mechanisms for increased bit efficiency (e.g. limiting the range of some random values in the protocols) as well as protocol extensions such as the Enhanced Class of Service mechanisms for BP.

2.2.2 BP Implementations

There are a number of Bundle Protocol implementations, including:

DTN2	Maintained by the Internet Engineering Research Task Force's DTN Research Group NASA's Marshall Space Flight Center maintains a fork of the DTN2 implementation that they are using for the ground segment for ISS communications
ION	Maintained by NASA
IBR-DTN	Maintained by Technische Universitat Braunschweig; targeted at embedded devices (e.g. home routers); also runs on android devices and desktops.
JDTN	Written by researchers at Cisco, not maintained.
Bytewalla	Not maintained; targeted at android devices.

The various implementations interoperate via the core set of capabilities described in RFC5050.

This document focuses on the Interplanetary Overlay Network (ION) BP implementation written and maintained by NASA.

2.2.2.1 ION BP Implementation

The ION code is available from the Ohiou University site at https://ion.ocp.ohiou.edu as well as from SourceForge. The main difference is that the Ohio University site includes a 'NASA Baseline' version for those with login credentials that fully supports the security mechanisms of the protocol suite. The open-source version that does NOT include full support for security tends to lead in terms of features (features are usually added to the open-source version before being migrated into the NASA baseline version). The SourceForge site distributes snapshots of the open-source version that do not contain the full software version control history.

The ION BP implementation is POSIX-based, and is regularly verified to run on the following architectures:

Linux
OS/X
FreeBSD
VxWorks
RTEMS

ION has also been compiled for and is occasionally tested on:

Microsoft Windows
BIONIC (the underpinning of the Android OS)

2.2.2.2 Installing the ION software

Note: instructions to install an entire environment including Ubuntu Linux, the CORE emulation environment, ION, and sample scenarios is provided in section 3.6 below.

2.2.2.2.1 From Source

The development kit includes a copy of the NASA Baseline source tree for ION. Building ION from source follows the standard:

```
./configure
make
make install
```

format for those systems on which ION has been verified to compile and run (see above).

2.2.2.2.2 Debian Package

The development kit includes a Debian package for the ION NASA Baseline version ION 3.3.1 release. To install the package on a system that uses the debian package manager use:

```
dpkg --install ion-open-source_3.3.1-1_amd64.deb
```

2.2.2.2.3 Windows Installer

Marshall Space Flight Center has developed a Windows installer as part of their Telescience Resource Kit (TReK). We hope to include this installer in future releases of the Development Kit.

2.2.3 Configuration

The development kit includes a configuration tool written by NASA's Jet Propulsion Laboratory. Because BP is designed to provide end-to-end connectivity in situations where there may be scheduled breaks (due e.g. to spacecraft pointing, unavailability of antenna time, etc.), it allows for scheduled links. ION's Contact Graph Routing (CGR)

3 DTN Scenarios

The second major component of the development kit is a set of mission-relevant scenarios using the Common Open Research Emulator (CORE) emulator. CORE provides a set of management functions on top of Linux Virtual Containers that house the various ION node implementations (including configurations).

The scenarios can be imported directly into a Linux host running CORE, are available in a Linux virtual machine in Oracle's VirtualBox format, and can be run off of a bootable ISO image of the virtual machine. Note that the bootable ISO currently does not provide the mechanism to install the VM onto a local virtual disk.

3.1 Sample Applications

The main applications used by the scenarios are:

- `bping`
  The bundle ping (bping) application sends a bundle to a destination that is presumed to be running a bpecho server, and waits for a reply.
- `bpsendfile / bprecvfile`
  The bpsendfile / bprecvfile applications are test applications that come with the ION distribution to effect simple file transfers. They do NOT include real 'file transfer' functionality (like sending the file metadata – such as the filename).
- `bpisend / bpirecv`
  The bpisend / bpirecv applications send an image file line-by-line, with each 'scan line'

The `bpisend / bpirecv` programs deserve special mention, since bpirecv displays an image to the screen. Because of the way that Linux containers work, you cannot directly connect to the X display server of the host machine from inside a container. Thus if you run bpirecv from inside a container it will *appear* to work correctly until it receives a bundle containing an image scan line, at which point (when it attempts to open an X window to display the line) it will fail. The solution to this is to use secure shell (ssh)'s X forwarding. SSH into the container from the host, port-forwarding X back to the host, and run bpirecv from there. To support this, the scenarios each include a 'backdoor' network between the host and the containers on 192.168.1.0/24. In general, the backdoor address of node X is 192.168.1.X

3.2 CORE Primer

The DTN scenarios run using the CORE emulator available from: http://www.nrl.navy.mil/itd/ncs/products/core. This section provides an EXTREMELY BRIEF overview of the commands needed to get the scenarios running under CORE. The full documentation for CORE is available at: http://downloads.pf.itd.nrl.navy.mil/docs/core/core_manual.pdf

CORE has two components, a core daemon (core-daemon) and a graphical user interface (GUI, the core-gui). The core daemon must run as root, while the core GUI can be run as an unprivileged user.

To see if the CORE daemon is running, type:

```
    ps aux | grep core-daemon | grep -v grep
```

If there is a core-daemon process then the daemon is running. If not, start it with:

```
    sudo core-daemon &
```

To start the core gui, use:

```
    core-gui &
```

Use the File menu to open a core scenario (a `.imn` file). The green 'Run' button ( ) on the left side of the core window will start the emulation.

When a simulation is running, you can double-click on a node (any of the blue router icons) to get a shell window on that node.

3.3 Relationship Between CORE and BP Connectivity

In ION, scheduled connectivity is controlled via the contact plan which controls any links that use the Licklider Transmission Protocol (LTP) convergence layer. Note that the contact plan does NOT affect other links, such as those that use the Internet's Transmission Control Protocol. The contact plan tells ION when those links are available and when they are not.

CORE has a number of ways of modeling wireless links. The scenarios included with the development kit use a simple distance method to determine connectivity. Mobility scripts in CORE can automatically move nodes according to predefined paths, and that movement may bring them into and out of contact with other nodes.

There is **NO INTRINSIC RELATIONSHIP** between CORE's notion of connectivity and that in the ION contact plans. For those scenarios that include changing connectivity, the mobility of the CORE nodes (which affects their connectivity) is carefully orchestrated to be (to the degree possible) synchronous with the ION contact plans.

3.4 Pre-Configured Scenarios

All of the scenarios included with the development kit start ION processes on each of the nodes, and run a `bpecho` server on BP service ID 1 of each node, and a `bprecvfile` server on BP service ID 2 of each node. This any node should respond to a bping command of the form:

```
        bping ipn:src_node_id.unused_service_ID ipn:dst_node_id.unused_serviceID
```

e.g. to ping from n2 to n3:

```
        bping ipn:2.5 ipn:3.1
```

And you can send a file to any node using:

```
        bpsendfile ipn:src_node_id.unused_service_ID ipn:dst_node_id.unused_serviceID
        FILENAME
```

e.g. to send the file README from n2 to n3:

```
        bpsendfile ipn:2.5 ipn:3.2 README
```

3.4.1 Base (Automatic Bping demo)

The ‘base’ scenario includes mobility and a very basic linear topology as shown in Figure 1.

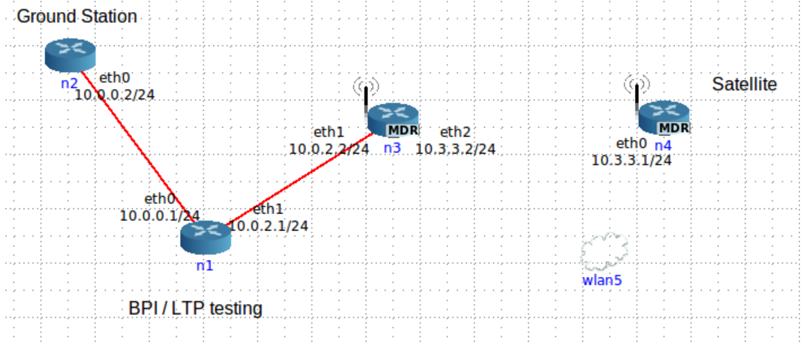


Figure 3: Topology for the 'Base' scenario.

When run, the scenario will automatically start the mobility script which moves the satellite into and out of range periodically, and will launch a `bping` command from node 2 (far left) to node 4 (the satellite). The ION contact plan is synchronized with the mobility so that ION correctly suspends the wireless link when the satellite is out of range and resumes it when the satellite comes back.

The first few pings may be lost while ION completes its startup process, but after that pings will be received whenever the satellite is connected to the fixed network. When the satellite is disconnected, bundles are queued for later delivery.

3.4.2 Diamond

The ‘Diamond’ scenario has no mobility and two paths from the source (n1, top) to the destination (n4, bottom), with connectivity alternating between the last links of the left hand

and right hand paths every 30s. That is: n1 is connected to n2 and n3, and n2 and n3 are each intermittently connected to n4 with 50% duty cycles and 180 degrees out of phase.

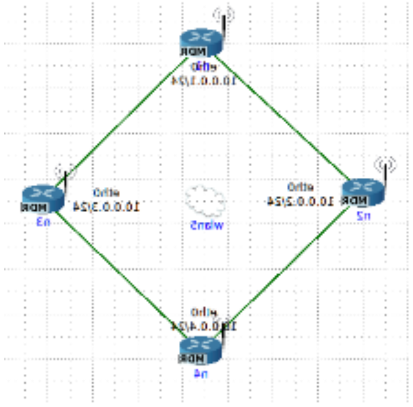


Figure 4: Diamond topology with alternating connectivity between the top and bottom nodes and the destination (n4).

Interesting experiments:
From a shell on n1, use dtn to ping n4 using:

```
bping ipn:1.5 ipn:4.1
```

From a shell on n3, tcpdump its wireless interface to see that bundles sometimes flow through n3 and sometimes through n2.

```
tcpdump -n -l -i eth0 udp
```

3.4.3 GSFC (Automatic BPI demo)

The 'GSFC' scenario emulates a scenario where a satellite has a high-rate downlink to a ground station, and the ground station has a lower-rate link across the terrestrial network. This scenario will automatically start the `bpisend / bpirecv` applications to send an image from the satellite (n4, far right) to the mission control center (n2, far left). The image transfer / display should start automatically.

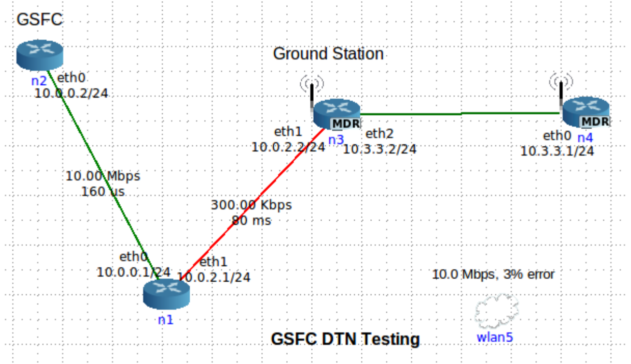


Figure 5: Emulation of the 'Rate-Mismatch' condition.

This scenario should automatically transfer and display the following image:

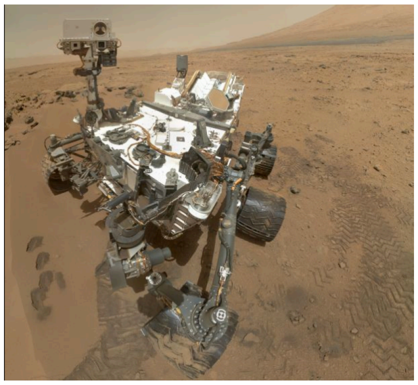


Figure 6: Image automatically transferred using the bpisend / bpirecv applications in the GSFC scenario.

To initiate an image transfer by hand:

From the HOST machine (NOT a shell on one of the CORE containers), do:

```
ssh –X –t –I /root/.ssh/id_rsa root@192.168.0.2 "bpirecv ipn:2.5" &
```

This starts the bpirecv program on node 2 and exports the X connection to the host machine (needed to display the file when using CORE).

From a shell on the n4 CORE node:

```
bpisend ipn:4.5 ipn:2.5 curiosity.png &
```

3.4.4 Mars

This scenario uses an emulated Mars orbiter as a 'data mule' to transfer information from the emulated rover to the mission control center via one of the emulated DSN stations. The scenario will automatically start a bping from the mission control node at the bottom to the mars rover.

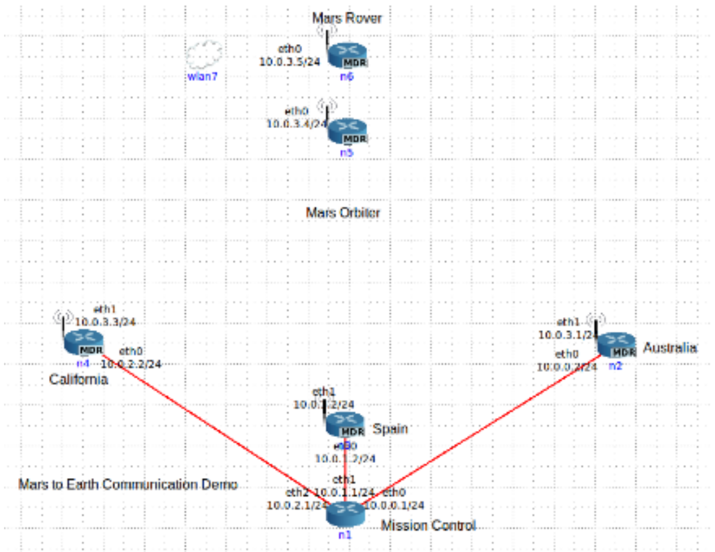


Figure 7: Scenario with an emulated orbiter serving as a data mule between an rover and control center, using three ground stations.

3.4.5 Planet

The Planet scenario emulates a LEO spacecraft communicating with a number of different ground stations, where all ground stations can receive from the satellite but only two (node n2 and n5) can transmit to the satellite.

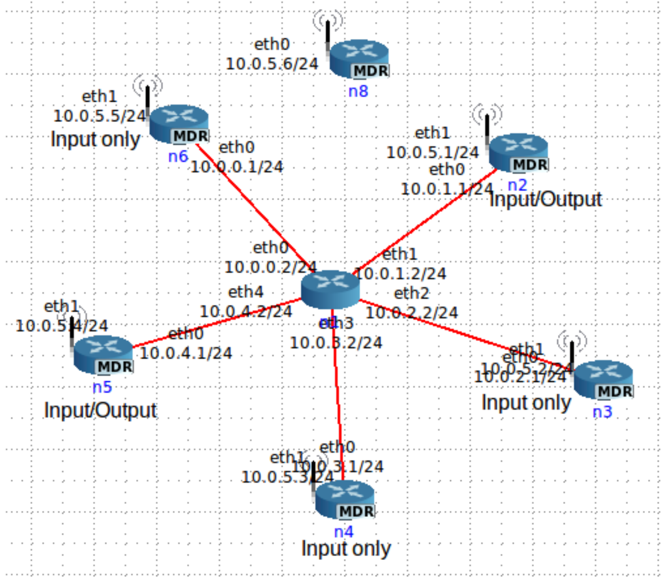
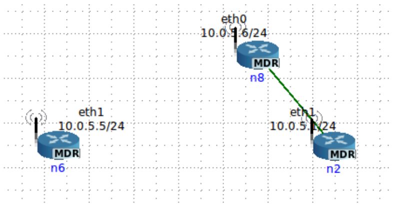


Figure 8: Planet sceanrio with multiple downlinks and a few uplinks.

3.4.6 Miss

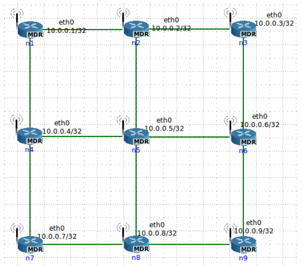
The 'Miss' scenario is a modified version of the planet scenario where the connectivity in CORE between the satellite and node 5 is missed on the 1st, 3rd, 5th, 7th, etc orbits. The connectivity IS PRESENT in the ION configuration file that controls LTP. Thus the ION system believes that there will be connectivity, but CORE forces this assumption to be false.

3.4.7 Data Mule



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

3.4.8 Square



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

3.5 Building Custom Scenarios

This section describes how to modify an existing scenario. The methods described here can be used, for example, to generate a scenario that is more representative of a prospective mission.

3.5.1 Scenario Components

This section describes the configuration files that form a scenario. (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.

- `$name.imn`: The CORE file containing the physical layout of the nodes, links, services, mobility scripts, and hooks (scripts that CORE runs at startup and shutdown.) This file is made by the drag-and-drop interface in core-gui.
- `$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 of the scenario. The nodes use this file to set their 0 time in the ION configuration files. 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`, `node$x.cfdprc`, `node$x.ionsecrc`, and `node$x.ionconfig`: All default settings that can stay the same. For advanced scenarios, these may need to be edited.

- `node$x.bprc`: bpadmin configuration file. Configures the endpoint, protocols, it's inducts and outducts, and has an option to turn on watch characters to help diagnostics.
- `node$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.ltprc`: 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 built-in delays to allow for ION to completely start up before trying to launch applications that depend on ION. This means that a scenario is not fully functional until 20-30 seconds after invocation with the core-gui.

The layout of the various configuration files in a scenario is shown below:

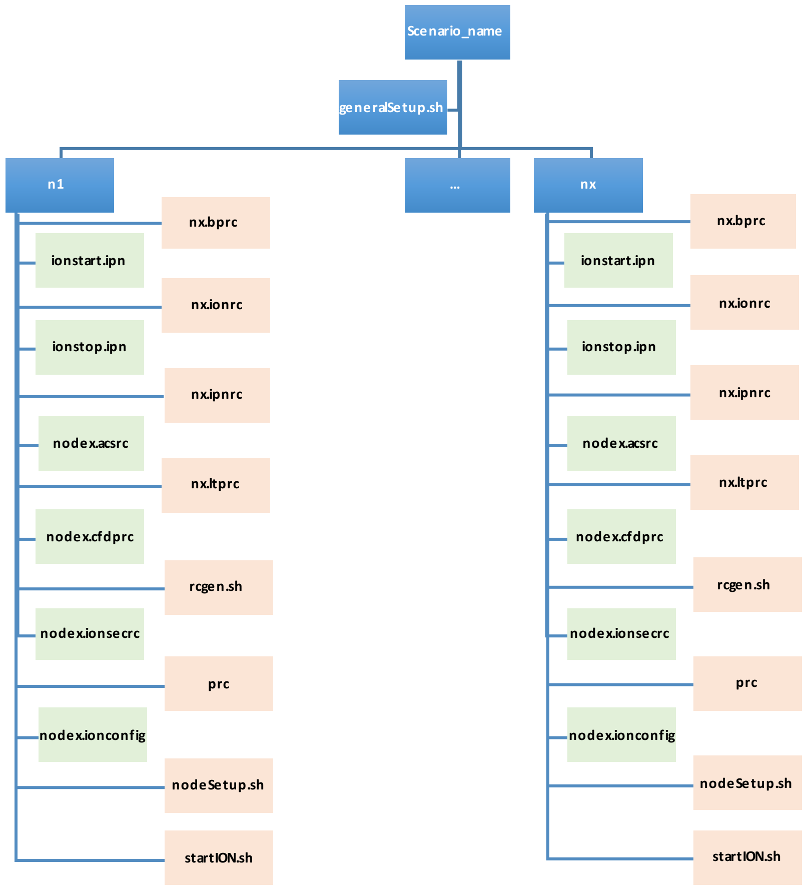


Figure 9: Directory structure for development kit scenarios; node-specific items in the left columns (ionstart.ion, etc.) should be reusable among scenarios. Node-specific configuration files in the right columns (nx.bprc, etc.) will probably need to be modified to fit the specific scenario.

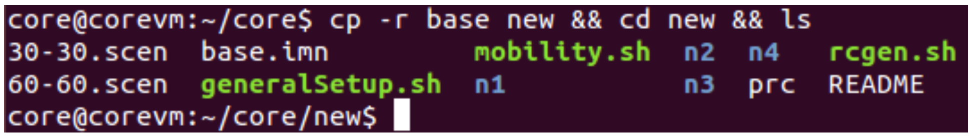
3.5.2 Building a Custom Scenario

This 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$ 
```

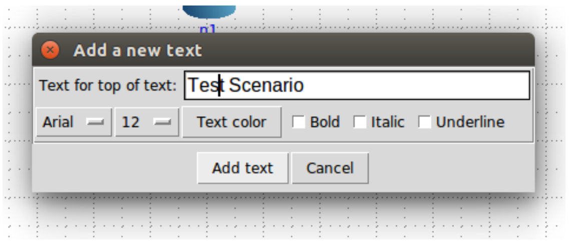
2. You can remove the files README and 60-60.scen. Neither of these will be used.
3. Rename base.imn with

   ```
   mv base.imn new.imn
   ```

   and the scenario using

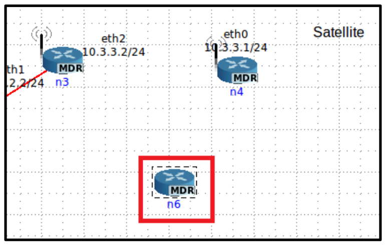
   ```
   mv 30-30.scen new.scen
   ```

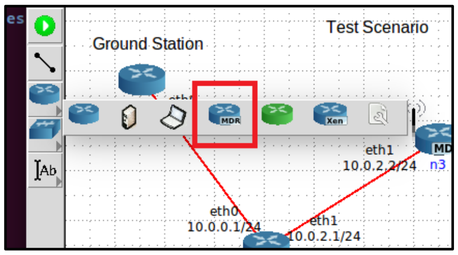
4. Open up a new terminal and open up a GUI. Load up your new scenario. We're going to add in another wireless node, add in a wired route, and change the mobility scripting.
5. Remove the "BPI…" heading and click the pencil button, then click the AB text button. Click anywhere to put a heading onto the 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.



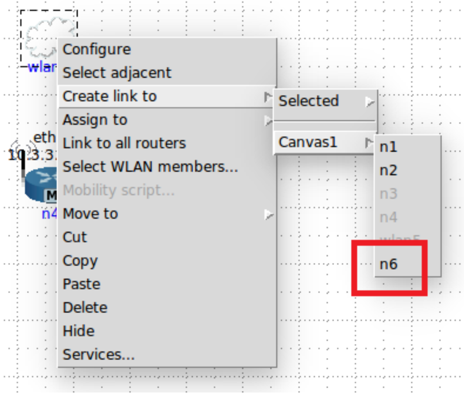
Text for top of text: Test Scenario

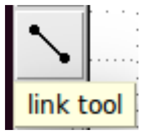
6. Click back to the pointer button at the top and you can click and drag your labels around. Double click anything on the GUI to open up its configuration box.
7. Drag your wlan5 object and put it above the satellite node to get it out of the way.
8. Click the router menu and click MDR router. Click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

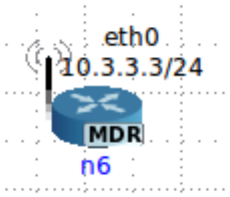




9. Create a link from wlan5 to the new node 6 by right clicking the wlan, selecting “Create link to”, selecting “Canvas1”, and then selecting “n6”.





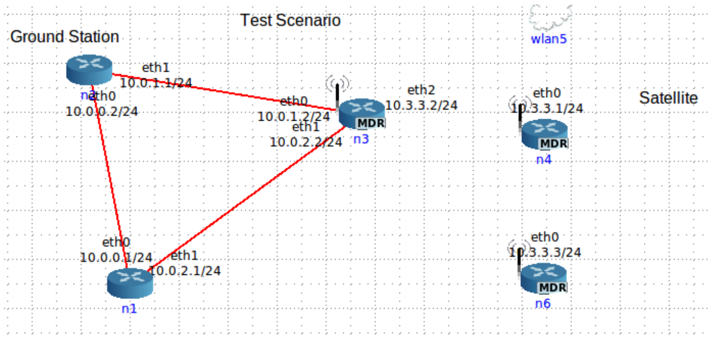


You should see an interface pop up and a little antenna on the node:

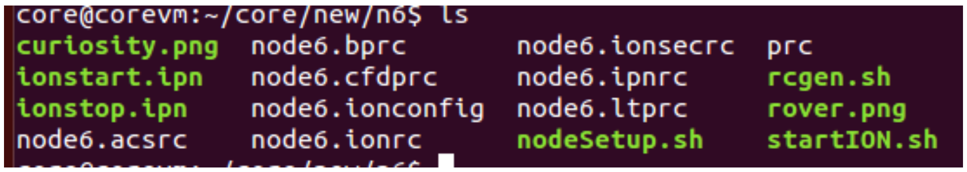
10. Now to put in a wired route in addition. Click the link tool on the sidebar and then click and drag from node 2 to node 3. (Note: make sure to go 2->3 and not 3->2. Normally it doesn’t matter, but for consistency with this guide, it does. The IP addresses change if you drag the link a different way.) 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
```



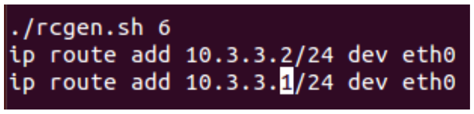
12. Observe that all of the files are node4.xyz. Rename all of them to node6.xyz using “mv”



```
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
```

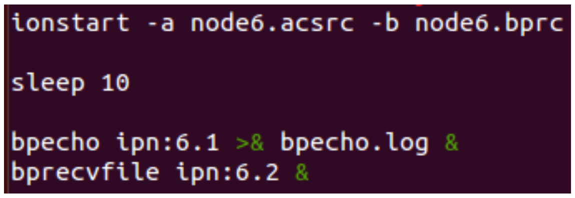
13. Open `ionstart.ipn` in your text editor of choice and change BASE_NAME to = “node6”. Also change the two echo statements to output the correct ipn: address. `ionstop.ipn` has echoes that can be fixed as well.

14. Open nodeSetup.sh and change the argument of `rcgen.sh`. In addition, add a new static route for the satellite:



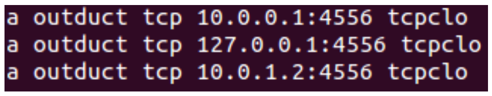
```
./rcgen.sh 6
ip route add 10.3.3.2/24 dev eth0
ip route add 10.3.3.1/24 dev eth0
```

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`

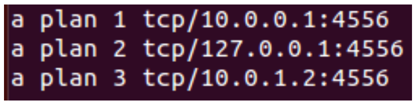


```
ionstart -a node6.acsrc -b node6.bprc

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
```

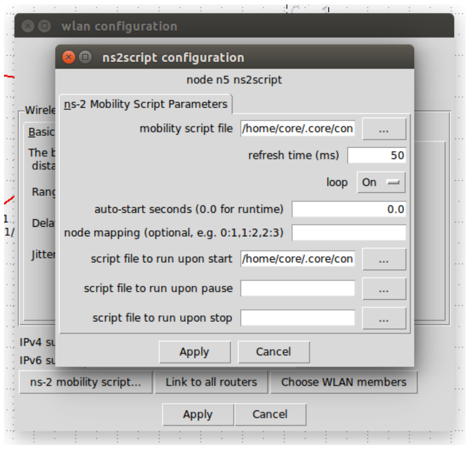


```
a plan 1 tcp/10.0.0.1:4556
a plan 2 tcp/127.0.0.1:4556
a plan 3 tcp/10.0.1.2:4556
```

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 file.
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.


wlan configuration

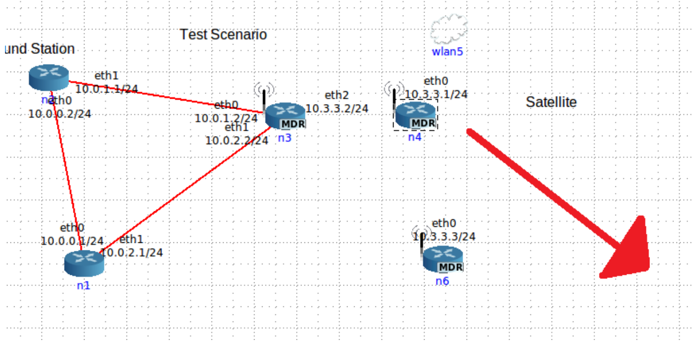
ns2script configuration

node n5 ns2script

ns-2 Mobility Script Parameters

mobility script file	/home/core/.core/con
refresh time (ms)	50
loop	On
auto-start seconds (0.0 for runtime)	0.0
node mapping (optional, e.g. 0:1,1:2,2:3)	
script file to run upon start	/home/core/.core/con
script file to run upon pause	
script file to run upon stop	

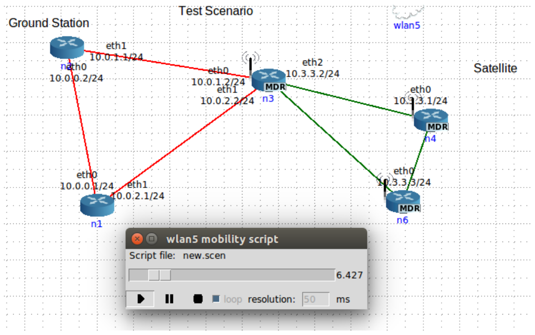
24. Make sure "mobility script file" is pointed at new.scen in the correct directory, and make sure "script file to run upon start" is set to our copy of mobility.sh. These are both picker menus, so just hit the "…" button and navigate to the file. In addition, make sure the "auto-start seconds" field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

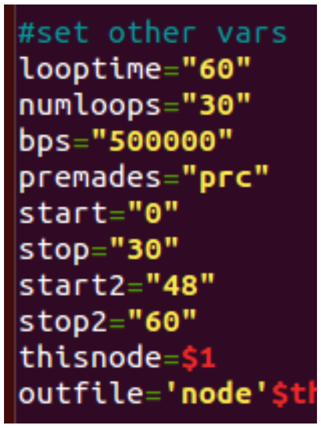


Script file: new.scen

resolution: 50 ms

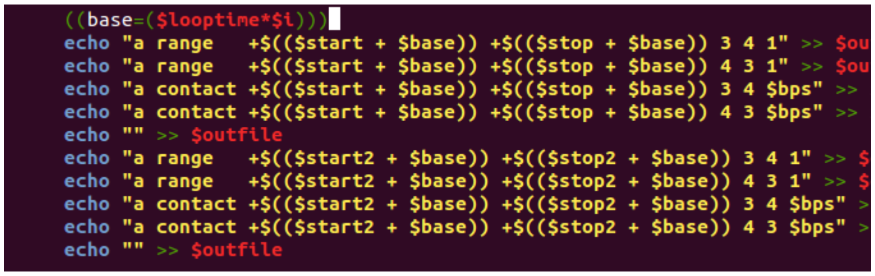
29. Save your mobility script and then hit the green button and make sure it all goes as planned. For 30 seconds, the node should move outwards, and then it should come back to its original position by 60 seconds.
30. Now that you have a good script, you need the times that the link from n4 to n3 dies 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"
premades="prc"
start="0"
stop="30"
start2="48"
stop2="60"
thisnode=$1
outfile='node'$th
```

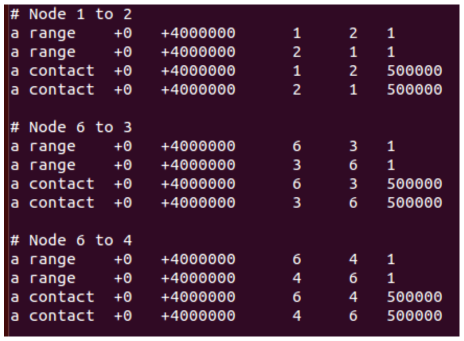
32. When your parameters are set, skip down to the FOR loop at the bottom. This is where the ranges and contacts are generated. There is already one basic echo set, so copy it and paste right below. Each set should have 5 lines. Change the “start” and “stop” variables in the second set to “start2” and “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



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

following times. To make a 1 directional connection, just comment out the line going in the blocked direction.

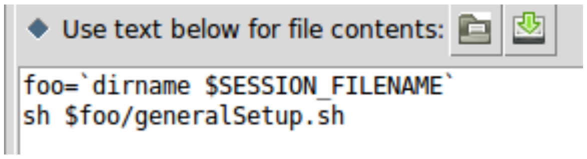
33. Once you have this, save and exit that file and open up prc. There are premade ranges and contacts for all of the nodes that were here already, but we have to write in the links we added. Following the syntax, add a range/contact for nodes 2 <--> 3, 3 <--> 6, and 4 <--> 6.



```
# 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
```

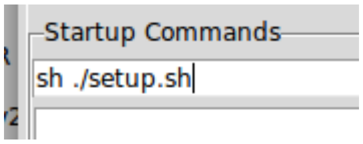
34. Save this file and then copy prc and rcgen.sh into each of the node folders. (“cp prc n1/”, etc)

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

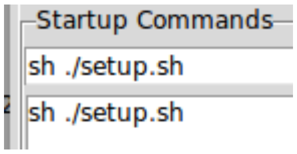


```
foo=`dirname $SESSION_FILENAME`
sh $foo/generalSetup.sh
```

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

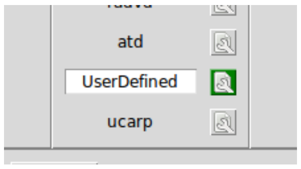


```
sh ./setup.sh
```



```
sh ./setup.sh
sh ./setup.sh
```

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.

3.6 CORE Virtual Machine

The Linux virtual machine that is part of the package contains the CORE emulation environment, the ION BP implementation, and the DTN scenarios above.

3.6.1 Installing VirtualBox

To install the VirtualBox virtual machine server:

- Download and install the Oracle VirtualBox virtualization server for your environment. VirtualBox can be downloaded from: https://www.virtualbox.org/
- Double-click on the coreVM.vbox file. This should load the virtual machine into VirtualBox.
- The virtual machine will come up to a login screen fore the 'core' user. The password is 'cvm' (without the quotes)

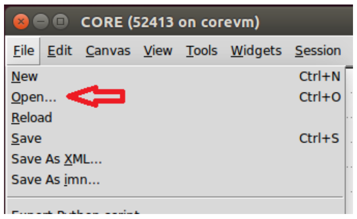
3.6.2 Start the Core-Daemon and Core-GUI

- To run the scenarios, you must first start the core daemon process. As explained above, this must be done by hand to allow for the X-based applications to run in the core nodes.

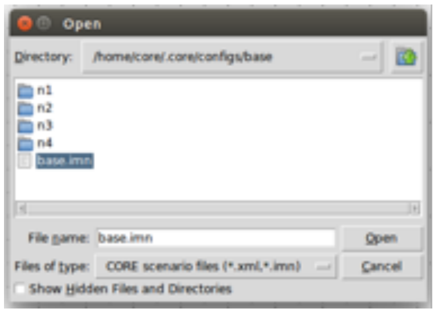
  Click on the 'Terminal' icon (Terminal) to get a terminal.
- Type "`sudo ls`" and hit return and enter the core user password (cvm) when prompted. This step just lists the files in the core user's home directory and is used here to authenticate the core user to the sudo process.
- Type "`sudo core-daemon &`". This will start the core deamon and put it in the background.
- Next type "`core-gui &`" to start the core Graphical User Interface.

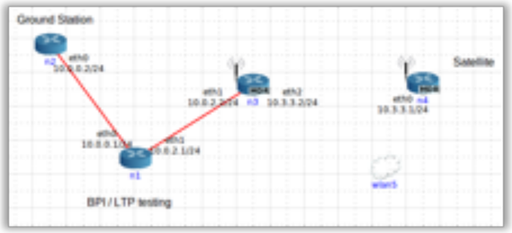
3.6.3 Running an Emulation in CORE

- In the core gui application, click "File > Open". This should open up a dialogue box:



- Double click the folder "base" and then the file "base.imn". That should get you the scenario shown on the right below:


Open
Directory: /home/core/.core/configs/base
n1
n2
n3
n4
base.imn
File name: base.imn
Open
Files of type: CORE scenario files (*.xml,*.imn)
Cancel
Show Hidden Files and Directories



To start the emulation, click the green 'Run' button ().

3.7 CORE Bootable ISO

The bootable ISO image is a copy of the virtual machine, and includes the CORE emulation environment and scenarios. To run the bootable ISO, insert the DVD into a 64-bit capable machine and invoke the 'boot from CD/DVD' option in the machine's BIOS. This will lead to the Development Kit Core VM login screen. The username / password for the bootable ISO are the same as for the CORE VM above.

If you want to install the VM from the ISO image to a local environment (e.g. so that you can make persistent changes), select the Ubuntu program menu and search for 'install', then run the installer. For this to work, the VM that booted the ISO image will need to have a virtual disk available.

Follow the instructions above for the CORE Virtual Machine to start the core emulation in the bootable ISO environment.

4 Additional Information

Information on the ION implementation is available at:
http://ipnsig.org/wp-content/uploads/2015/05/Whats-new-in-ION.pdf

5 References

[ipnsig] The Internet Society's Interplanetary Networking Special Interest Group (ipnsig) web page is located at: http://ipnsig.org/

[RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC5050, November 2007.

[RFC5326] Ramadas, M., Burleigh, S., and S. Farrell, "Licklider Transmission Protocol – Specification", RFC5326, September 2008,

[CCSDS_BP] CCSDS Bundle Protocol, Recommendation for Space DataSystem Standards, CCSDS xxx-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, FORTHCOMING.

[ION_Impl] The ION implementation is available from the following locations:
http://sourceforge.net/projects/ion-dtn/
https://ion.ocp.ohiou.edu/

[CORE] The Common Open Research Emulator is available for download from:
http://www.nrl.navy.mil/itd/ncs/products/core