

# Weekly Report

## Design Oriented Project



### **Birla Institute of Technology and Science, Pilani (KK Birla Goa Campus)**

**Objective:** To explore UnetStack standard libraries and create a 3-node simulation.

**Authors:** Umang Gupta(2017A8PSO510G), Krishna Pranay Reddy Chennareddy(2018AAPS0302G), Amrit M (2018AAPS0700G) , Krishi Ajaykumar Agrawal(2019AAPS0235G)

**Guided by:** Dr. Sarang C. Dhongdi.

**Summary:** First week we examined different protocols in the UnetStack framework. We examined a three node network and through experimentation discovered the range and connectivity for different depths and distances for a  $-10\text{dB re } 1 \mu\text{Pa}^2/\text{Hz}$  power level. A multi hop route was finally set up such that the farthest node which was initially disconnected can be connected.

## **Contents**

1. What's in UnetStack ?
  - a. Unet Network
2. Week1:
  - a. Task
  - b. Procedure
3. Results
4. Observations
5. Add ons

## **What's in UnetStack ?**

About 71% of Earth's surface is covered with water, and about 97% of the water is in our oceans. Although the ocean plays a critical role in everything from the air we breathe to daily weather and climate patterns, we know very little about it. To really understand our oceans, we need a way to sense and observe the numerous complex processes that drive the ocean environment, and to report the data collected back to our data centers. While cabled ocean observatories have been established in a few locations, they are too expensive to setup and maintain for large scale data collection across the vast oceans. Over the past few decades, wireless communication technology has percolated into every aspect of our lives, and we have come to take it for granted. This technology forms the bedrock of wireless sensor networks, allowing us to gather data with ease. Most of the wireless communication technology we use relies on electromagnetic waves (e.g. radio waves, visible light) that get rapidly absorbed by water. Hence the technology is ineffective for underwater communication, except at very short distances or extremely low data rates. Most underwater communication systems today use acoustic waves, which can travel long distances in the right conditions. At short distances in clear waters, optical communication systems are sometimes used for high speed communications. Although these communication technologies can be leveraged to establish point-to-point communication links, these links do not integrate well with networking technology available today. The Unet project strives to develop technologies that allow us to build communication networks that extend underwater, be it via acoustic, optical, or even wired links. Some nodes in such networks may be above water, while others are underwater. In this handbook, we explore how to build such networks using UnetStack3, an agent-based network technology that was developed in the Unet project.

## **Unet Network**

A "Unet" network consists of several nodes (underwater, on the surface of water, or above water) that communicate over various types of links. Unet nodes are equipped with one or more network interfaces that allow communication over some of these links. For example, to communicate over an underwater acoustic link, we need an underwater acoustic modem. For an underwater optical link, we use an underwater optical modem. Most RF, GSM, satellite or wired links would be accessed over a standard TCP/IP network interface. In all cases, each Unet node would run the UnetStack software that allows us to effectively communicate over all of these types of links using a common Application Programming Interface (API). UnetStack API bindings are available for several languages including Java, Groovy, Python, Julia, C, Javascript, etc.

# Week 1

**Task :** To set up a simple 3-node network that can communicate with each other.

## Procedure:

- 1.) We used the RealTimePlatform class to create a real time simulator for the nodes.
  - a.) RealTimePlatform runs the agents in its containers in real-time. The notion of time in this platform is that of elapsed time in the real world, and hence this is a suitable platform for most applications.
  - b.) Inherited methods from class Platform: addContainer, getBuildVersion, getContainers, getHostname, getNetworkInterface, getPort, isIdle, isRunning, setHostname, setNetworkInterface, setNetworkInterface, setPort, shutdown, start. There are the very primary classes that provide the basic functionality for our simulations
- 2.) We used the TCP/IP networking model.
- 3.) We initiated the simulation and we used the stack originally provided for creating the nodes.

```
import org.arl.fjage.Agent

boolean loadAgentByClass(String name, String clazz) {
    try {
        container.add name, Class.forName(clazz).newInstance()
        return true
    } catch (Exception ex) {
        return false
    }
}

boolean loadAgentByClass(String name, String... clazzes) {
    for (String clazz: clazzes) {
        if (loadAgentByClass(name, clazz)) return true
    }
    return false
}

loadAgentByClass 'arp', 'org.arl.unet.addr.AddressResolution'
loadAgentByClass 'ranging', 'org.arl.unet.localization.Ranging'
loadAgentByClass 'mac', 'org.arl.unet.mac.CSMA'
loadAgentByClass 'uwlink', 'org.arl.unet.link.ECLink', 'org.arl.unet.link.ReliableLink'
loadAgentByClass 'transport', 'org.arl.unet.transport.SWTransport'
loadAgentByClass 'router', 'org.arl.unet.net.Router'
loadAgentByClass 'rdp', 'org.arl.unet.net.RouteDiscoveryProtocol'
loadAgentByClass 'statemanager', 'org.arl.unet.state.StateManager'

container.add 'remote', new org.arl.unet.remote.RemoteControl(cwd: new File(home, 'scripts'), enable: false)
container.add 'bbmon', new org.arl.unet.bb.BasebandSignalMonitor(new File(home, 'logs/signals-0.txt').path, 64)
```

a.) import the class Agent because it is the Base class to be extended by all agents. An agent must be added to a container in order to run. An agent usually overrides the `init()` and `shutdown()` methods to provide the appropriate behavior.

b.) Then we load all the agents like ones for transport, ranging, routing etc.

c.) We used CSMA for MAC.

4.) Finally we set different distances for our nodes to check at which range the signals start breaking. The distances between the nodes was decided upon by trial and error checking for any packet losses at different distances.

Here's the groovy setup for our demonstration.

```
import org.arl.fjage.RealTimePlatform

println '''
Demo 3-node network
-----

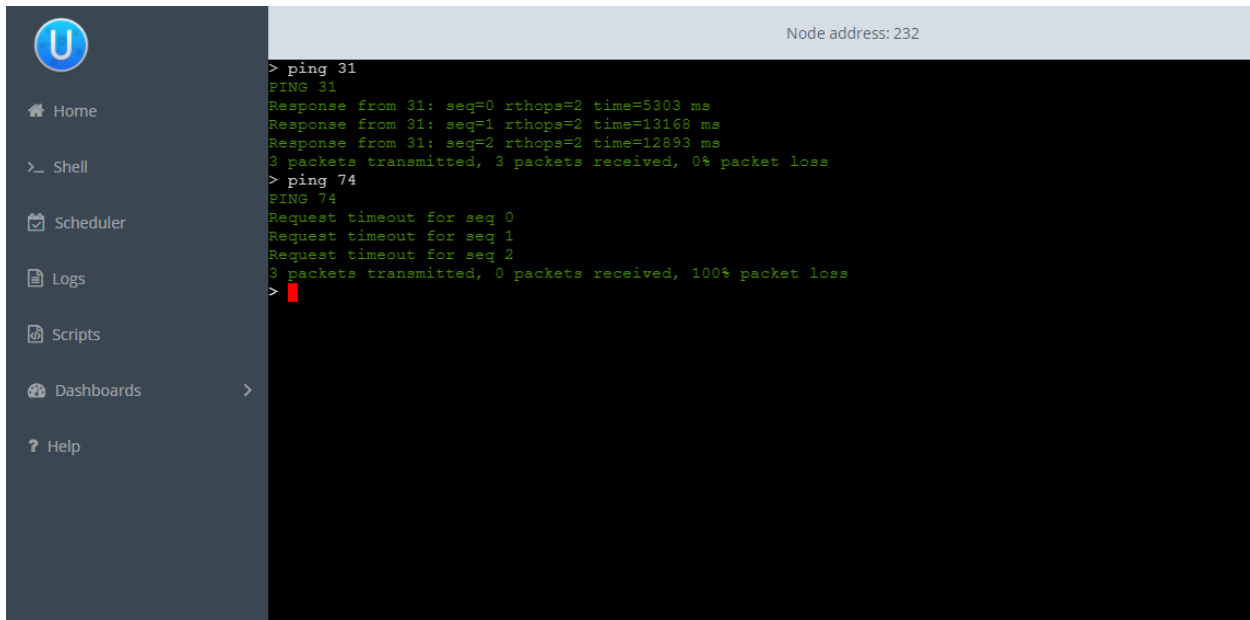
Node A: tcp://localhost:1101, http://localhost:8081/
Node B: tcp://localhost:1102, http://localhost:8082/
Node C: tcp://localhost:1103, http://localhost:8083/
...

platform = RealTimePlatform
origin = [0.0,0.0]

simulate {
    node 'A', location: [100.0,1.0,-15], web: 8081, api: 1101, stack: "$home/etc/setup"
    node 'B', location: [2400.7,1.0,-15], web: 8082, api: 1102, stack: "$home/etc/setup"
    node 'C', location: [3300,1.0,-15.0], web: 8083, api: 1103, stack: "$home/etc/setup"
}
```

## Results

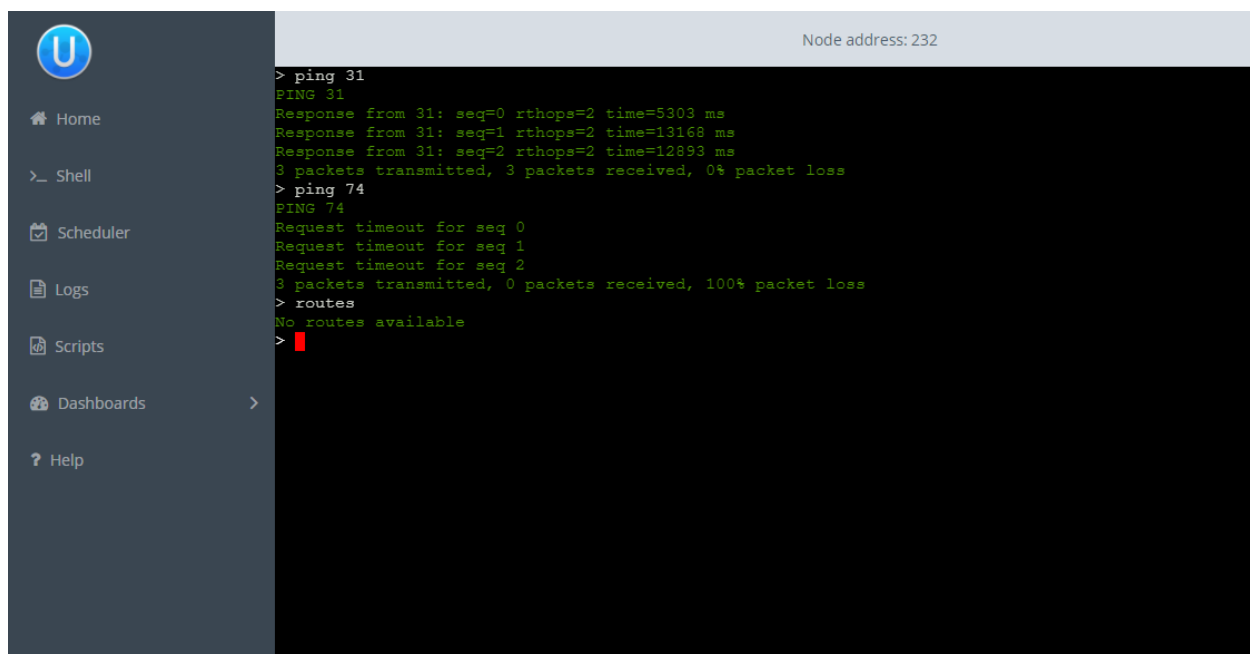
- 1.) We check connectivity to the other nodes using the 'ping' command. We see that Node 'B' is reachable and there is 0% loss and Node 'C' is unreachable. Node 'B' is at a distance 2.3 km from the Node 'A' and the Node 'C' is at a distance of 4.1 km from Node 'A'.



The screenshot shows a terminal window with a dark background. On the left is a sidebar with a blue circle containing a white 'U' at the top, followed by menu items: Home, Shell, Scheduler, Logs, Scripts, Dashboards, and Help. The main area of the terminal has a light blue header bar that says 'Node address: 232'. Below this, the following text is displayed:

```
> ping 31
PING 31
Response from 31: seq=0 rthops=2 time=5303 ms
Response from 31: seq=1 rthops=2 time=13168 ms
Response from 31: seq=2 rthops=2 time=12893 ms
3 packets transmitted, 3 packets received, 0% packet loss
> ping 74
PING 74
Request timeout for seq 0
Request timeout for seq 1
Request timeout for seq 2
3 packets transmitted, 0 packets received, 100% packet loss
>
```

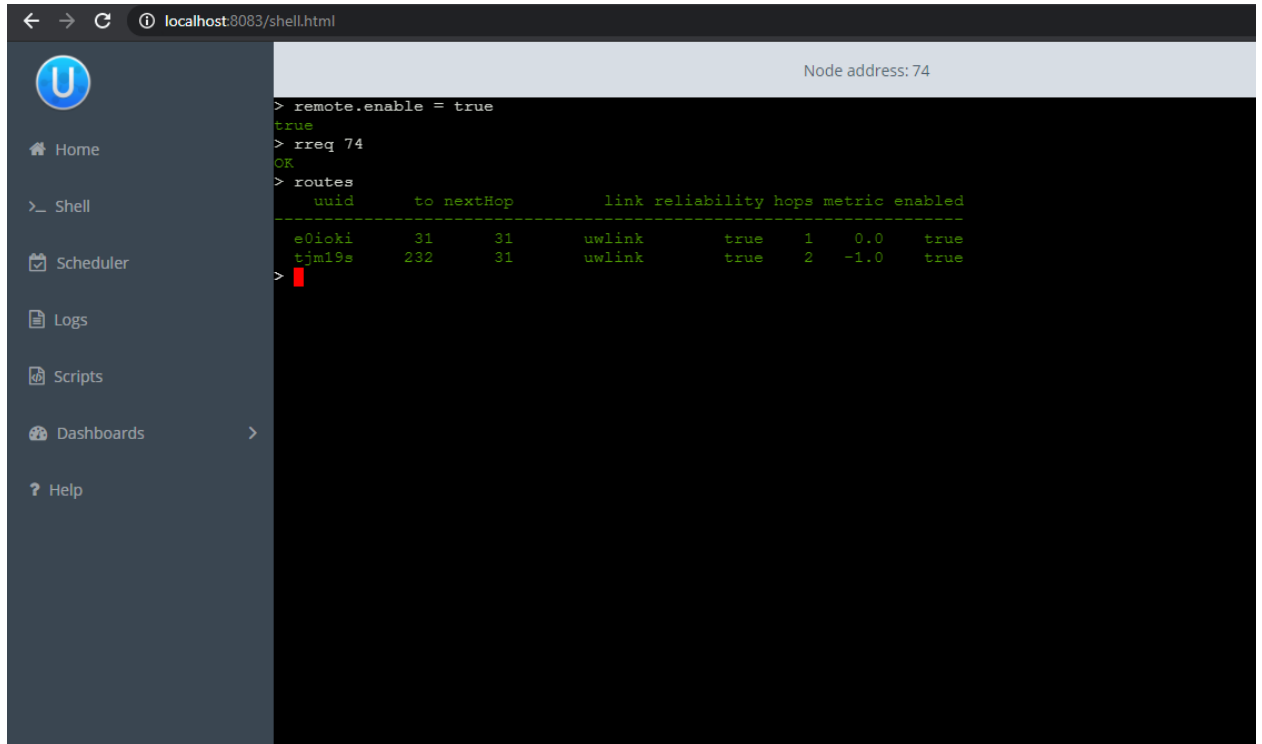
- 2.) Now we'll check if there are any routes from Node 'A' to Node 'C'.



This screenshot is similar to the previous one, showing the same terminal interface. The output is as follows:

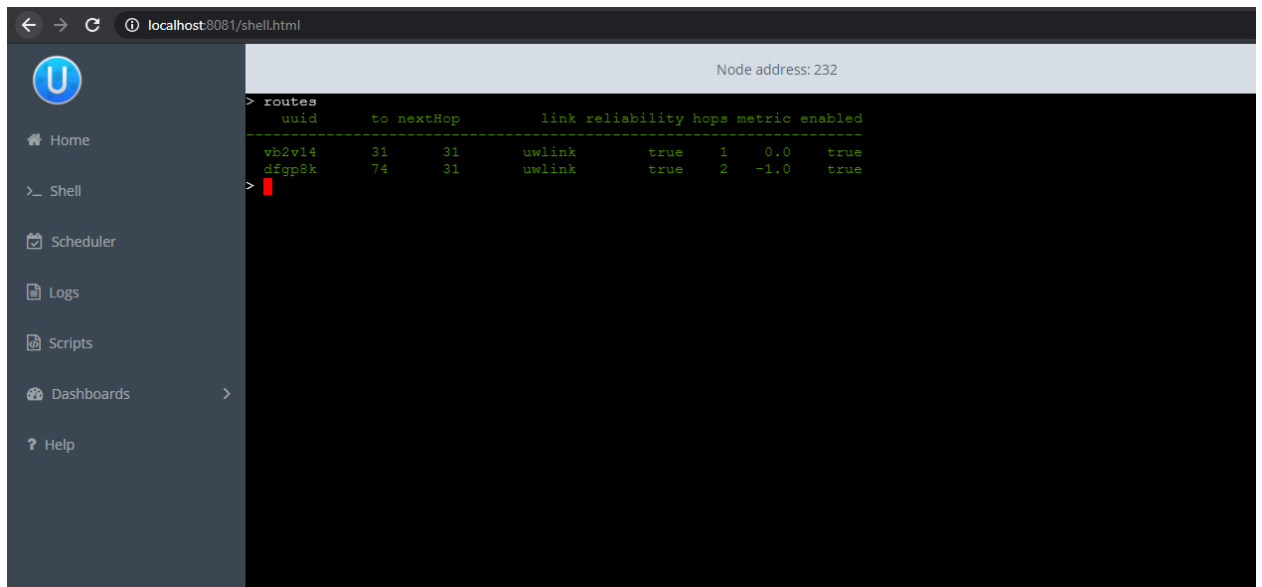
```
> ping 31
PING 31
Response from 31: seq=0 rthops=2 time=5303 ms
Response from 31: seq=1 rthops=2 time=13168 ms
Response from 31: seq=2 rthops=2 time=12893 ms
3 packets transmitted, 3 packets received, 0% packet loss
> ping 74
PING 74
Request timeout for seq 0
Request timeout for seq 1
Request timeout for seq 2
3 packets transmitted, 0 packets received, 100% packet loss
> routes
No routes available
>
```

- 3.) Now to add routes we'll enable remote access to all the nodes and use the 'rreq' command mentioned in the Routing Protocols.
- 4.) This adds routes from A via B to C and also at C it adds a route from C to A via B.



```
> remote.enable = true
true
> rreq 74
OK
> routes
```

uuid	to	nextHop	link	reliability	hops	metric	enabled	
e0ioxi	31	31	uwl	link	true	1	0.0	true
tjml9s	232	31	uwl	link	true	2	-1.0	true

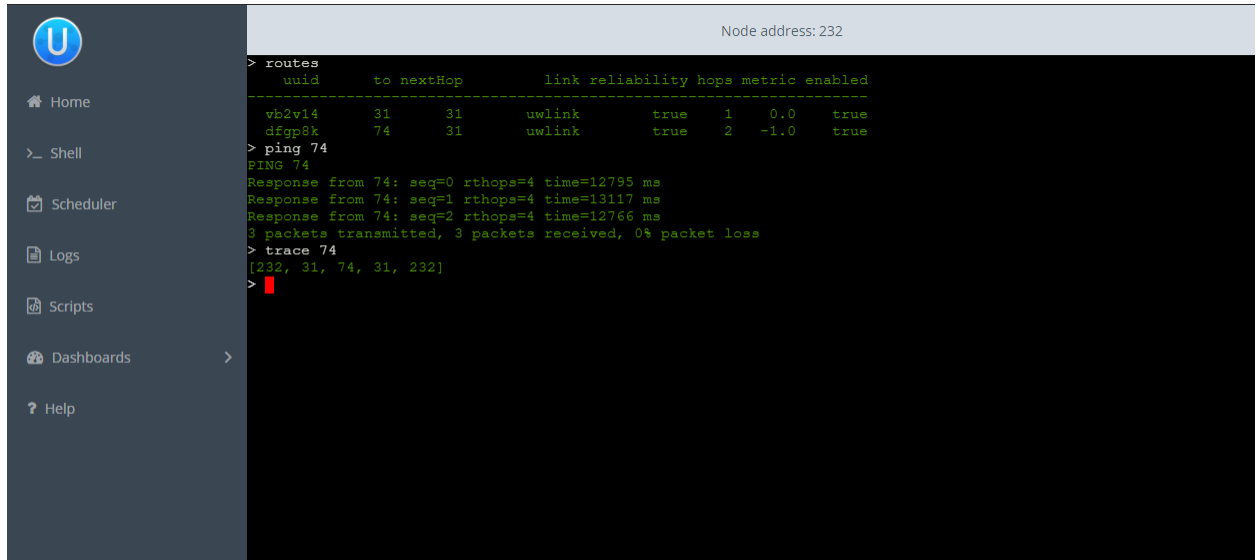


```
> routes
```

uuid	to	nextHop	link	reliability	hops	metric	enabled	
vb2v14	31	31	uwl	link	true	1	0.0	true
dfgp8k	74	31	uwl	link	true	2	-1.0	true

- 5.) We use routes to check the routing process and observe the routes have been added.

- 6.) Now we'll check the connectivity again using 'ping' and then check the routing using 'trace'.
- 7.) Trace command shows that there is a route to 'C' and it is through node 'B' and there is no data loss.



The screenshot shows a terminal window with a dark background and a sidebar on the left. The sidebar contains a blue circle with a white 'U' logo at the top, followed by menu items: Home, Shell, Scheduler, Logs, Scripts, Dashboards, and Help. The main terminal area has a light blue header bar that says 'Node address: 232'. Below this, the following commands and their outputs are shown:

```
> routes
      uuid      to nextHop      link reliability hops metric enabled
-----
vb2v14      31      31      uwl      true      1      0.0      true
dfgp8k      74      31      uwl      true      2      -1.0      true

> ping 74
PING 74
Response from 74: seq=0 rthops=4 time=12795 ms
Response from 74: seq=1 rthops=4 time=13117 ms
Response from 74: seq=2 rthops=4 time=12766 ms
3 packets transmitted, 3 packets received, 0% packet loss

> trace 74
[232, 31, 74, 31, 232]
>
```

- 8.) We can also add these routes manually using 'addroute'.



## Observations

- 1.) We can connect nodes for a range of 2.5km such that no data loss takes place with a power level of -10 dB re 1  $\mu\text{Pa}^2/\text{Hz}$  operating at a depth of 15m.
- 2.) For the same depth, with the power level of 0 dB re 1  $\mu\text{Pa}^2/\text{Hz}$ , the maximum range observed was about 4km.

## Add ons

We have also explored remote shell command (`rsh`) which can be used to access the shells of physically inaccessible nodes.