Breakdown of Discrete Event Simulation Script in Unet Handbook

Author : Umang Gupta 2017A8PS0510G

Introduction

This document explores the scripting procedures available in Unet stack 3 with respect to MAC protocols (except for ALOHA). This documents the process of exploring the software script for MAC simulation from the discrete simulation portion from the handbook. We try to figure out what command does what function simply because something like that is not clearly specified in the handbook and understanding it is very time demanding, This document is written like notes are written with simple ideas that you can have while understanding the scripts and thus can save you a great deal of time. It is some of the earliest exploratory work we did so if you've reached a basic understanding of the scripting process you can skip this particular document. If however you're just starting maybe reading through this could help you save some time and get some basic understanding.

What do we know so far ?

They have simulated a simple MAC aloha protocol -- transit a frame as soon as data arrives

There are some theoretical assumptions for ALOHA derivations :

1. Random arrival is Poisson distribution
2. If two frames arrive at the same time -- they are lost
3. Half duplex nodes -- cannot transmit while receiving
4. No noise losses
5. No propagation delay

While testing for different MAC protocols we would want to keep the assumptions 2 and 4 from the handbook -- same as the protocols for

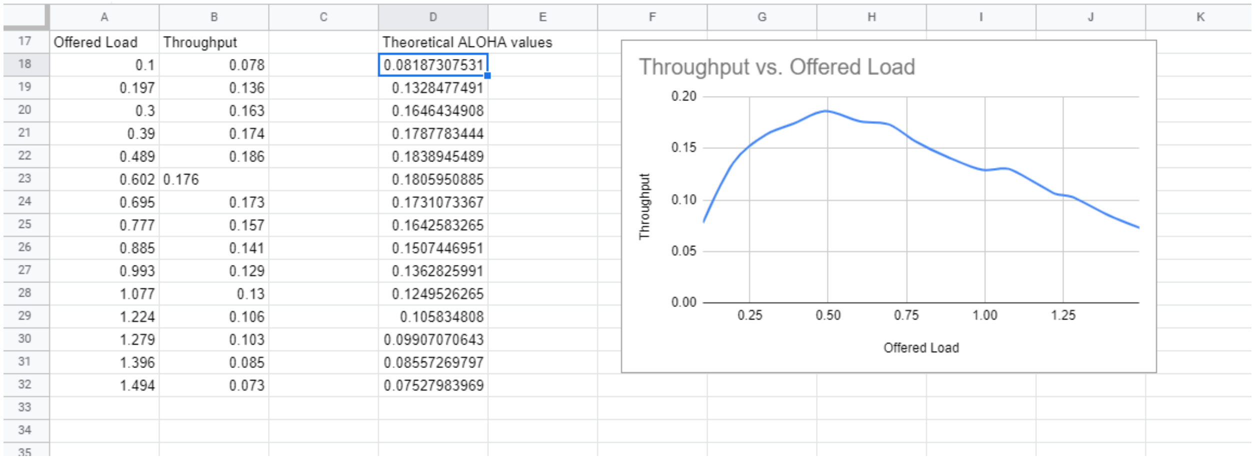
- Thus we will be using the protocol channel model

# IDEA 1 : What happens if during the simulations of the nodes if the nodes are created with the stack set to etc/setup:

Result: Simply changing the node formation doesn't work. You have to change the on startup behaviour to be able to implement any protocols.

Experiment done with ALOHA sim script where CSMA was used during node setup but the on startup behaviour was from the ALOHA sim script. Clearly doesn't reflect CSMA as the simulated throughput is nearly identical as the theoretical throughput for ALOHA. This is because in our script we are manually overriding the sending process.

Conclusion : You need to develop your own “onstartup” behaviour to simulate different MAC procedures.



	A	B	C	D	E
17	Offered Load	Throughput		Theoretical ALOHA values	
18	0.1	0.078		0.08187307531	
19	0.197	0.136		0.1328477491	
20	0.3	0.163		0.1646434908	
21	0.39	0.174		0.1787783444	
22	0.489	0.186		0.1838945489	
23	0.602	0.176		0.1805950885	
24	0.695	0.173		0.1731073367	
25	0.777	0.157		0.1642583265	
26	0.885	0.141		0.1507446951	
27	0.993	0.129		0.1362825991	
28	1.077	0.13		0.1249526265	
29	1.224	0.106		0.105834808	
30	1.279	0.103		0.09907070643	
31	1.396	0.085		0.08557269797	
32	1.494	0.073		0.07527983969	
33					
34					
35					

In order to be able to implement any protocol we must have an understanding of the physical services provided in Unet Sim. This is because this agent will be used in sending datagrams.

Notes on services provided in UnetSim

- Terminology:
    - Agent : basically a service or anything that does anything (*My understanding is basically a class in JAVA with certain methods*)
    - Message : Agents interact with each other or -- *(implementations that trigger certain function of the objects)*
        - Request: ask agent to perform certain tasks : has a suffix
        - Response : agents have to give a response for the request: AGREE / REFUSE / FAILURE / NOT_UNDERSTOOD
        - Notif and topic we’ll not consider for now
        - Parameters : KEY-VALUE pairs !!!! very important / can be set by **agent.parameter** or by a ParameterReq(request)

Learning how to send Datagrams :

Syntactically : uwlink << new DatagramReq(to: 31, data: new byte[64]) // this would do

There is something called as Short - Circuit optimization : this means you can use the “phy” layer to send and receive datagrams. This is what is used in all simulation scripts.

Note: Physical services typically bypass any MAC implementation and directly access the channel layer.

/// **Problem**: Even if we figure out how to send multiple messages from scripts and they generate trace files the problem would remain that none of this would amount to anything if signals are sent without involving the MAC layers.

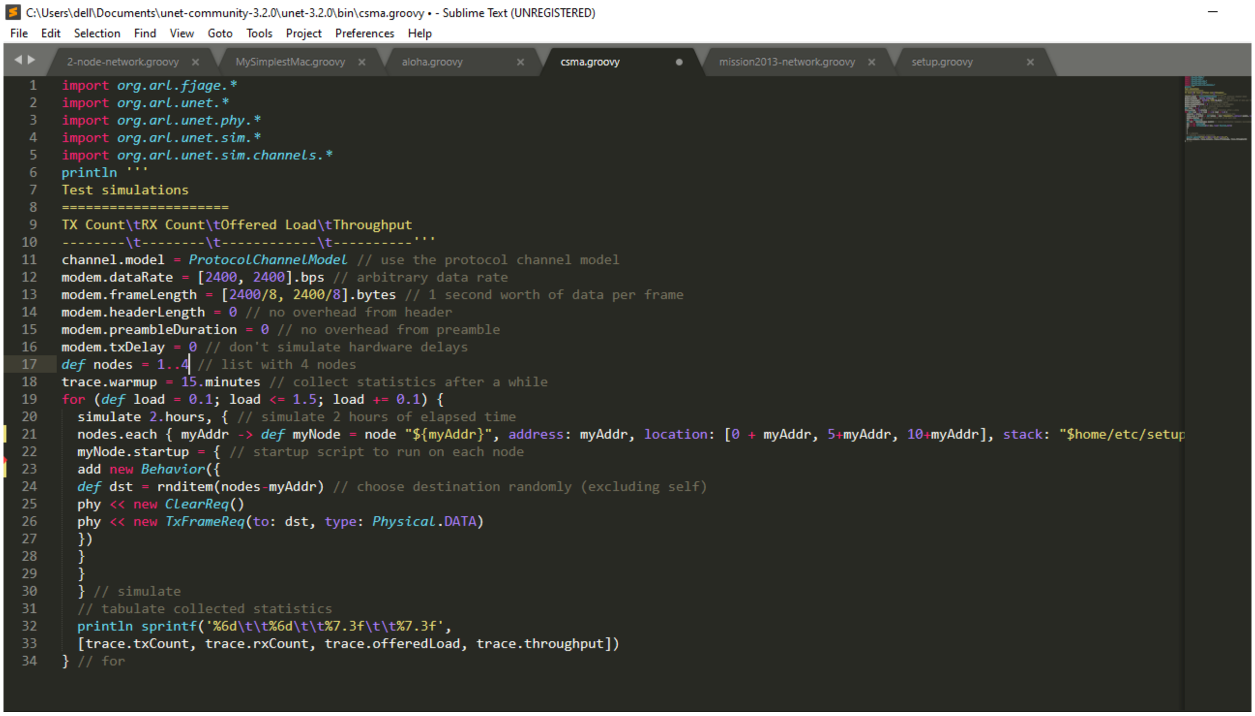
/// Problem with Testing MAC protocols in handbook : In the hand book they test MAC protocols by checking if the reservation request is being accepted -- this creates a problem because we don’t know how we will translate this into our ‘testing script’

/// A possible method : Since these MAC scripts mention reserving channels maybe we can change/modify the script such that we have certain reservation before doing the ClearReq() and TxFrameReq()

/// Conclusion we’ll need to use the random poisson behaviour class in ALOHA for all MAC simulations

Some Experiments with the Simulation Script

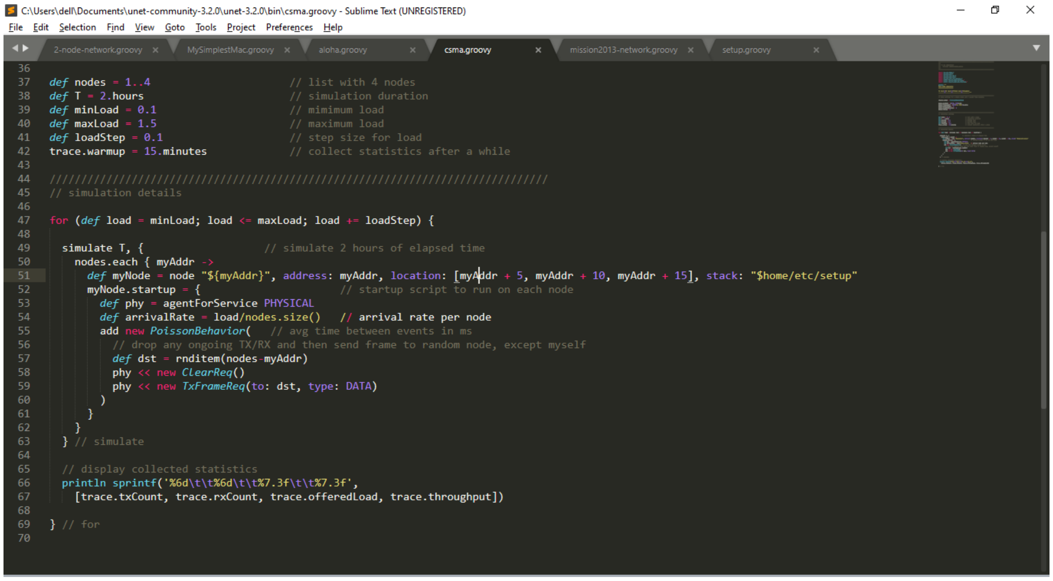
Experiment 1 : I wanted to experiment how the simulation script handles a datagram process without the poisson behaviour. I used the setup model to set up this route to explore what my results would look like. There is no arrival rate or load mentioned: this is just to explore what portion of the script does what:
Also I have varied the locations of the nodes.
Here is the script:


C:\Users\dell\Documents\unet-community-3.2.0\unet-3.2.0\bin\csma.groovy • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
2-node-network.groovy
MySimplestMac.groovy
aloha.groovy
csma.groovy
mission2013-network.groovy
setup.groovy

```
import org.arl.fjage.*
import org.arl.unet.*
import org.arl.unet.phy.*
import org.arl.unet.sim.*
import org.arl.unet.sim.channels.*
println '''
Test simulations
====================
TX Count\tRX Count\tOffered Load\tThroughput
--------\t--------\t------------\t----------'''
channel.model = ProtocolChannelModel // use the protocol channel model
modem.dataRate = [2400, 2400].bps // arbitrary data rate
modem.frameLength = [2400/8, 2400/8].bytes // 1 second worth of data per frame
modem.headerLength = 0 // no overhead from header
modem.preambleDuration = 0 // no overhead from preamble
modem.txDelay = 0 // don't simulate hardware delays
def nodes = 1..4 // list with 4 nodes
trace.warmup = 15.minutes // collect statistics after a while
for (def load = 0.1; load <= 1.5; load += 0.1) {
  simulate 2.hours, { // simulate 2 hours of elapsed time
  nodes.each { myAddr -> def myNode = node "${myAddr}", address: myAddr, location: [0 + myAddr, 5+myAddr, 10+myAddr], stack: "$home/etc/setup
  myNode.startup = { // startup script to run on each node
  add new Behavior({
  def dst = rnditem(nodes-myAddr) // choose destination randomly (excluding self)
  phy << new ClearReq()
  phy << new TxFrameReq(to: dst, type: Physical.DATA)
  })
  }
  }
  } // simulate
  // tabulate collected statistics
  println sprintf('%6d\t\t%6d\t\t%7.3f\t\t%7.3f',
  [trace.txCount, trace.rxCount, trace.offeredLoad, trace.throughput])
} // for
```

// Conclusion : Nothing was sent and nothing was received. Clearly need to have some sort of slotting mechanism.

Experiment 3: Used a new script // got rid of the average time and arrival rate bits:


C:\Users\dell\Documents\unet-community-3.2.0\unet-3.2.0\bin\csma.groovy - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
2-node-network.groovy
MySimplestMac.groovy
aloha.groovy
csma.groovy
mission2013-network.groovy
setup.groovy

```

def nodes = 1..4                       // list with 4 nodes
def T = 2.hours                        // simulation duration
def minLoad = 0.1                      // mimimum load
def maxLoad = 1.5                      // maximum load
def loadStep = 0.1                     // step size for load
trace.warmup = 15.minutes              // collect statistics after a while

////////////////////////////////////////////////////////////////////////////////
// simulation details

for (def load = minLoad; load <= maxLoad; load += loadStep) {

  simulate T, {                      // simulate 2 hours of elapsed time
    nodes.each { myAddr ->
      def myNode = node "${myAddr}", address: myAddr, location: [myAddr + 5, myAddr + 10, myAddr + 15], stack: "$home/etc/setup"
      myNode.startup = {                      // startup script to run on each node
        def phy = agentForService PHYSICAL
        def arrivalRate = load/nodes.size()   // arrival rate per node
        add new PoissonBehavior(   // avg time between events in ms
          // drop any ongoing TX/RX and then send frame to random node, except myself
          def dst = rnditem(nodes-myAddr)
          phy << new ClearReq()
          phy << new TxFrameReq(to: dst, type: DATA)
        )
      }
    }
  } // simulate

  // display collected statistics
  println sprintf('%6d\t\t%6d\t\t%7.3f\t\t%7.3f',
    [trace.txCount, trace.rxCount, trace.offeredLoad, trace.throughput])

} // for

```

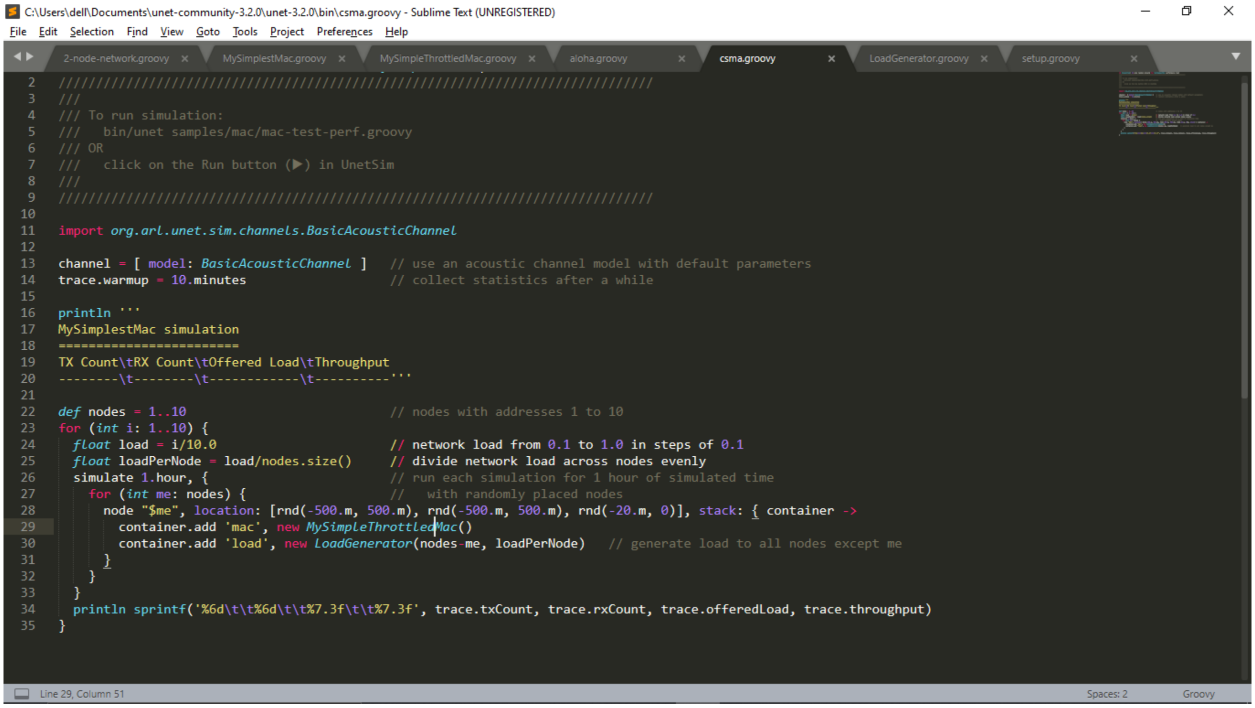
// Conclusion : Same result as in ALOHA

/// Experiment 4: Working without clearing ongoing process : basically without using the ClearReq() to let the half duplex modems internal coding handle conflicts.

Result: Failure // nothing sent and received

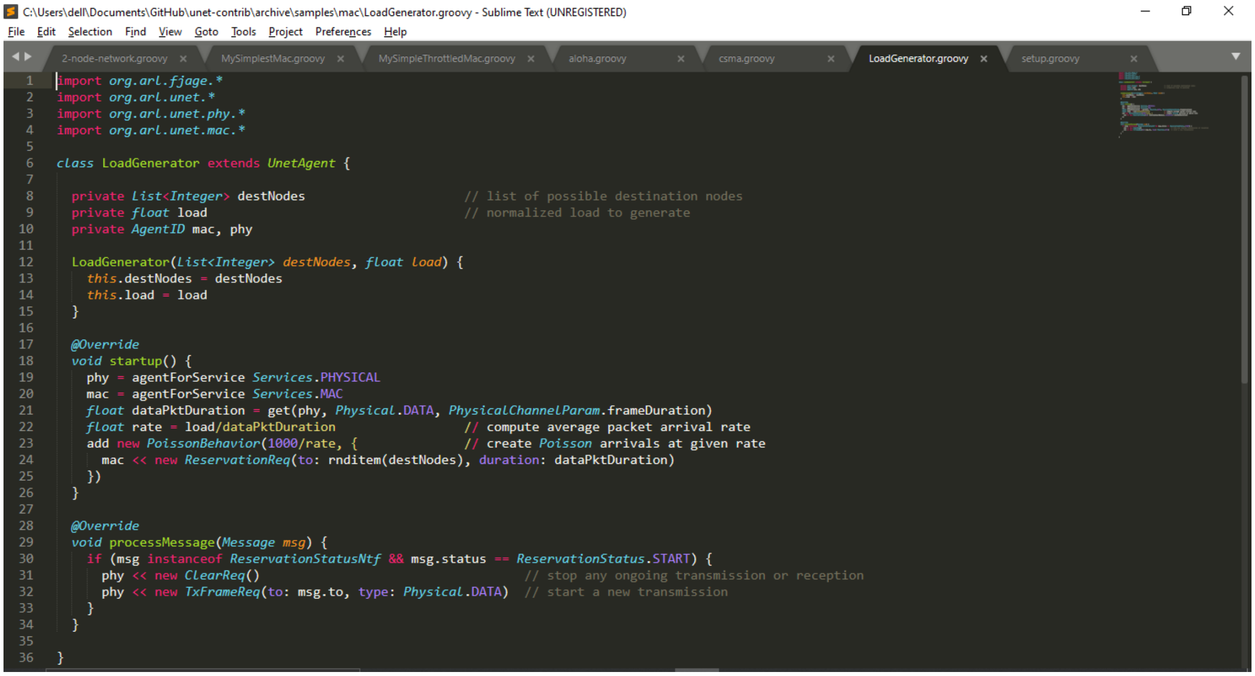
/// Experiment 5: Using the load and simulation script from the unet-contrib (The Git File)

There was no data transferred in this script// perhaps there are some issues with the way MAC notifications / responses are being interpreted. However, a new starting point for simulations as now we have a script that sends datagrams after receiving the reservation signal from the MAC layer --- (this in itself could be the problem however from a syntax point of view very very useful in understanding the present scenario).

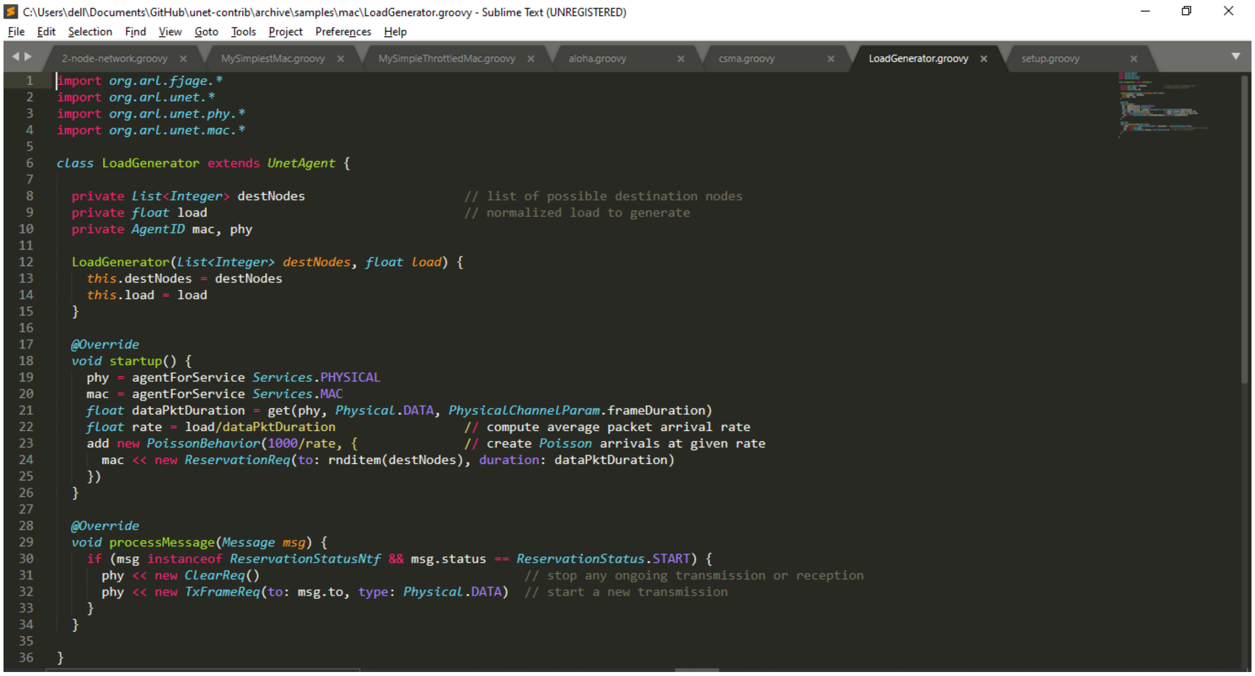

C:\Users\dell\Documents\unet-community-3.2.0\unet-3.2.0\bin\csma.groovy - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
2-node-network.groovy
MySimplestMac.groovy
MySimpleThrottledMac.groovy
aloha.groovy
csma.groovy
LoadGenerator.groovy
setup.groovy
Line 29, Column 51
Spaces: 2
Groovy

```
///////////////////////////////////////////////////////////////////////////////
///
/// To run simulation:
///   bin/unet samples/mac/mac-test-perf.groovy
/// OR
///   click on the Run button (▶) in UnetSim
///
///////////////////////////////////////////////////////////////////////////////

import org.arl.unet.sim.channels.BasicAcousticChannel

channel = [ model: BasicAcousticChannel ]   // use an acoustic channel model with default parameters
trace.warmup = 10.minutes                   // collect statistics after a while

println '''
MySimplestMac simulation
========================
TX Count\tRX Count\tOffered Load\tThroughput
--------\t--------\t------------\t----------'''

def nodes = 1..10                           // nodes with addresses 1 to 10
for (int i: 1..10) {
  float load = i/10.0                       // network load from 0.1 to 1.0 in steps of 0.1
  float loadPerNode = load/nodes.size()     // divide network load across nodes evenly
  simulate 1.hour, {                        // run each simulation for 1 hour of simulated time
    for (int me: nodes) {                   //   with randomly placed nodes
      node "$me", location: [rnd(-500.m, 500.m), rnd(-500.m, 500.m), rnd(-20.m, 0)], stack: { container ->
        container.add 'mac', new MySimpleThrottledMac()
        container.add 'load', new LoadGenerator(nodes-me, loadPerNode)   // generate load to all nodes except me
      }
    }
  }
  println sprintf('%6d\t\t%6d\t\t%7.3f\t\t%7.3f', trace.txCount, trace.rxCount, trace.offeredLoad, trace.throughput)
}
```

This was the simulation script used -- I tried this for both MySimplestMac and MySimpleThrottledMac -- same result -- no output


C:\Users\dell\Documents\GitHub\unet-contrib\archive\samples\mac\LoadGenerator.groovy - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
2-node-network.groovy
MySimplestMac.groovy
MySimpleThrottledMac.groovy
aloha.groovy
csma.groovy
LoadGenerator.groovy
setup.groovy

```
import org.arl.fjage.*
import org.arl.unet.*
import org.arl.unet.phy.*
import org.arl.unet.mac.*

class LoadGenerator extends UnetAgent {

  private List<Integer> destNodes                     // list of possible destination nodes
  private float load                                  // normalized load to generate
  private AgentID mac, phy

  LoadGenerator(List<Integer> destNodes, float load) {
    this.destNodes = destNodes
    this.load = load
  }

  @Override
  void startup() {
    phy = agentForService Services.PHYSICAL
    mac = agentForService Services.MAC
    float dataPktDuration = get(phy, Physical.DATA, PhysicalChannelParam.frameDuration)
    float rate = load/dataPktDuration                 // compute average packet arrival rate
    add new PoissonBehavior(1000/rate, {              // create Poisson arrivals at given rate
      mac << new ReservationReq(to: rnditem(destNodes), duration: dataPktDuration)
    })
  }

  @Override
  void processMessage(Message msg) {
    if (msg instanceof ReservationStatusNtf && msg.status == ReservationStatus.START) {
      phy << new ClearReq()                                   // stop any ongoing transmission or reception
      phy << new TxFrameReq(to: msg.to, type: Physical.DATA)  // start a new transmission
    }
  }

}
```


C:\Users\dell\Documents\GitHub\unet-contrib\archive\samples\mac\LoadGenerator.groovy - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
2-node-network.groovy
MySimplestMac.groovy
MySimpleThrottledMac.groovy
aloha.groovy
csma.groovy
LoadGenerator.groovy
setup.groovy

```
import org.arl.fjage.*
import org.arl.unet.*
import org.arl.unet.phy.*
import org.arl.unet.mac.*

class LoadGenerator extends UnetAgent {

  private List<Integer> destNodes                     // list of possible destination nodes
  private float load                                  // normalized load to generate
  private AgentID mac, phy

  LoadGenerator(List<Integer> destNodes, float load) {
    this.destNodes = destNodes
    this.load = load
  }

  @Override
  void startup() {
    phy = agentForService Services.PHYSICAL
    mac = agentForService Services.MAC
    float dataPktDuration = get(phy, Physical.DATA, PhysicalChannelParam.frameDuration)
    float rate = load/dataPktDuration                 // compute average packet arrival rate
    add new PoissonBehavior(1000/rate, {              // create Poisson arrivals at given rate
      mac << new ReservationReq(to: rnditem(destNodes), duration: dataPktDuration)
    })
  }

  @Override
  void processMessage(Message msg) {
    if (msg instanceof ReservationStatusNtf && msg.status == ReservationStatus.START) {
      phy << new ClearReq()                                   // stop any ongoing transmission or reception
      phy << new TxFrameReq(to: msg.to, type: Physical.DATA)  // start a new transmission
    }
  }

}
```

The load generator script -- basically is syntactically supposed to confirm my reservation and then transfer the datagram -- however there must be an issue !

Sample output -- rest is the same !

/// Conclusion: look for better working of the simulation script. Look into the load script first !