An Extensible Platform for Constructing *NS3* Models

James M. Westall

April 7, 2020

Abstract

This document describes the use of the extensible *netsim* system for constructing *ns3* network models.

1 Introduction

The *netsim* system is an extensible platform that supports the construction and reconfiguration of complex network models without the need for code reconfiguration except when the system is extended to support a new network technology.

The system consists of three components that interoperate with each other and the underlying *ns3* system. The components described here reside in *tgifdb:/home/ns3/devel/mw*.

1.1 The model component

A model is defined by a configuration file. The configuration file contains the definition of the networks and the workloads. The model layer processing configuration file and sets up the data structures that drive the model and collect performance data. It also is responsible for printing performance data. The model component is implemented in the file *model.c* and the main script file *netsim.cc*.

1.2 The application component

This component is implemented in the files *udpsock.c* and *tcpsock.c*. It implements the behavior of the applications defined in the configuration file, collects end-to-end performance metrics and contains the interfaces to the *ns3* socket emulation.

1.3 The *ns3* interface component

This component is implemented in the file *ns_tools.c*. It contains the interface to the *ns3* procedures that create and configure model elements.

2 The model component

The model component presently supports 255 subnets of 255 hosts each in the 192.168.snet.host space. Each host or router is represented by an instance of an *ns3 Node* class.

The *ns3 Nodes* are contained in instances of *ns3 NodeContainer* classes. The model component contains an array of *ns3 NodeContainer* structures, *nodeCon[MAX NETS]*. Nodes whose primary interface is associated with subnet *snet* are created in *nodeCon[snet]*.

A node with multiple interfaces will also 'reside' in the *nodeCon* of every subnet upon which it has an interface. Each node is associated with a unique instance of *ns3 NetDevice* class for every subnet upon which it has an interface. Instances of the *NetDevice* classes are contained in the *deviceCon[MAX NETS]* array of the model component.

The *NetDevice* instance associated with a *Node* holding address 192.168.snet.host can be recovered as *deviceCon[snet].Get(host-1)* where the -1 on the host side is caused by the first host on a subnet having address 1 and not 0.

2.1 The configuration file

The configuration file contains the network and workload definitions. It is structured as a collection of elements of the form:

```
entity-name
{
   attribute-name   attribute value  comment
   attribute-name   attribute value  comment
   :
}
```

Entity definitions *except for netDef* can be presented in any order. There are no constraints on the presentation of attribute definition

The *netsim* system requires that config file names end in the suffix *.cfg*.

We now illustrate the tcp7.cfg model that is in /home/ns3/NS3/ns-3-allinone/ns-3-dev/mw.

2.1.1 The sysData entity

The *sysData* entity is designed to contain global parameters. The *tcpSegSize* attribute should move to *SendModel* and a *udpSegSize* should be added.

```
sysData                    // general system configuration
{
   stopTime       100.0     // stop simulation time
   routeMgr       1         // use olsr route manager
   rateMgr        1         // use ideal dynamic rate manager
   mcsMode        0         // irrelevant with non-constant rate mgr
   tcpSegSize     1448      // standard tcp seg size
}
```

2.1.2 The sendModel entity

There may be multiple send models but each connection must have one. At present only constant send rate is supported but on-off, Poisson, and scripted senders are easily implemented.

```
sendModel                  // models one or more senders
{
   connectTime      18     // seconds after start of simulation
   startTime        20     // tx start seconds
   stopTime         80     // tx stop seconds
   sendRate     4.0e+4     // Bps
   sendBufSize       7     // Tcp packets at sending host
   packetSize     1432     // Bytes .. Tcp segsize - header size
   distIPT           0     // deterministic dist for interpacket time
};
```

2.1.3 The gridNet entity

At present there are two network types supported: grid and point to point. Grid nets do not presently support mobility and have rectangular structure with possibly different node counts and spacing in the x and y spacing between nodes. At present grid nets are realized as 802.11n networks at 20 Mhz and short intersymbol gaps with single spatial streams with default Tx power, antennas and gains.

```
gridNet                    // Topology of a wireless adhoc grid net
{
   xLoc     0              // location of node (0, 0)
   yLoc     0
   xNodes   5              // number of nodes on X axis
```

```
   yNodes   5              // number of nodes on Y axis
   xSize    30             // spacing in meters
   ySize    30
}
```

2.1.4 The p2pNet entity

The point to point net entity can describe a new point to point net connecting two new nodes, two existing nodes, or an existing node to a new node. The *nodeMask* field must define what is desire. Values are:

```
   0   -   Both nodes are new
   1   -   Source node exists but destination does not
   2   -   Destination exists but source source does not
   3   -   Both nodes exist
```

The value of newCon is *required* and defines the subnet address of the point to point link. Nodes are created or copied to *nodeCons[newCon]* as required.

```
p2pNet                  // describes a point to point link
{
   nodeMask    1        // 1 -> source node exists
   sourceCon   0        // source container / network number
   sourceNode  24       // source node (host - 1) number
   destCon     0        // dest data is ignored for netmask 1
   destNode    0
   newCon      1
   rate        8        // in Mbps
   latency     2.0      // Over the air / link in ms
}
```

2.1.5 The connection entity

The connection entity describes one or more connections. The one shown here creates 25 connections because of the repDest attribute. The connections are from 192.168.2.2 to every node on the wireless grid: 192.168.0.1 through 192.168.0.25. There is also a repSource attribute that can be used to create many to one connections.

```
connection              // describes a single connection
{
   sourcePort      32769    // base port numbers
   destPort        4097
```

```
    sourceConId      2        // container / network numbers
    sourceNodeId     1        // node  / host-1 numbers
    destConId        0        // source node is 192.168.2.2
    destNodeId       0        // dest node is 192.168.0
    protocol         6        // TCP
    sendModel        0        // index of sender node in table
    repDest         24        // destination replications
}
```

2.1.6 The netDef entity

Instances of the netDef entity determine exactly which grid and point to point networks are included in the final simulated network. This may appear to be an unneeded abstraction but in fact it simplifies the implementation code as it complexifies the config file because there are some underlying hidden assumptions in play.

It is implicitly assumed that grid networks are created before point to point links connecting them to each other or external routers are created. It is also assumed that when a new point to point link connecting an existing point to point router to a new point to point router is created that the existing node already exist.

The order in which the netDef entities are presented *must* ensure that this ordering in preserved in the order in which the netDefs occur.

```
netDef
{
    netType          0        // Wireless grid
    gridDef          0        // Index of the grid description
}

netDef
{
    netType          1        // Point to point
    p2pDef           0        // Index of a point to point desc.
}
```

2.2 A complete config file

The file *tcp7.cfg* demonstrates the use of multiple networks. It defines a 25 node adhoc wireless grid having network address 192.168.0.0. Then node indices are 0-24 and the corresponding IP addresses are 192.168.0.0 to 192.168.0.25.

The node with IP address 192.168.0.25 is connected by 2 point to point networks to a remote host. The two point to point networks have network addresses 192.168.1.0 and and 192.168.2.0. Since the

node with IP address, has interfaces both the grid and on 192.168.1.0, its address on 192.168.1.0 is 192.168.1.1. The intermediate node between the grid and the server also has two interfaces. Its address on 192.168.1.0 is 192.168.1.2 and its address on 192.168.2.0 is 192.168.2.1. The remote host's address on 192.168.2.0 is then 192.168.2.1.

The other three "corner" nodes of the grid also are connected by separate 2 hop paths to the server. So the server has 4 interfaces and the final IP address assignment is as shown.

Recall that the network number *is* the index into the array of *nodeContainers* where the node resides. And the host address - 1 is the index of the node within the container.

The number of containers in which a single node resides is equal to the number of interfaces it possesses.

```
Grid corners                Middle Hop                 Server
192.168.0.25 _____________ 192.168.1.2 ___________ 192.168.2.2
192.168.1.1   192.168.1.0   192.168.2.1 192.168.2.0

192.168.0.1 _____________ 192.168.3.2 ___________ 192.168.4.2
192.168.3.1   192.168.3.0   192.168.4.1 192.168.4.0  192.168.2.2

192.168.0.21 _____________ 192.168.5.2 ___________ 192.168.6.2
192.168.5.1   192.168.5.0   192.168.6.1 192.168.6.0  192.168.2.2

192.168.0.5 _____________ 192.168.7.2 ___________ 192.168.8.2
192.168.7.1   192.168.7.0   192.168.8.1 192.168.8.0  192.168.2.2
```

2.3 Running a model

After constructing a model such as tcp7.cfg, the next step is to run it. This can be done in two ways: an uncontrolled run; and a debug run under the control of the *gdb* debugger.

The simplest way to initiate a run is to use the *runsim.sh* or *dbgsim.sh* script.

2.3.1 runsim.sh

To run in to in the first mode there are several options:

```
./runsim.sh tcp7
./runsim.sh tcp7       "--routeMgr=1"
./runsim.sh tcp7.olsr  "--routeMgr=1"
./runsim.sh tcp7.aodv  "--routeMgr=2"
```

All four of these invocations will expect to find a config file named tcp7.cfg. In the first two invocations the names of the output files will begin *tcp7.*. In the last two they will begin *tcp7.olsr* and *tcp7.aodv* respectively.

The script itself is:

```
#!/bin/bash
if [ $# -lt 1 ]
then
   echo Usage is  runsim.sh work2 \"--sendBuf=2 --routeMgr=2\"
   echo where work2 is the prefix of the config file output files
   echo and the --parameters are optional but the \"s are not!
   echo If you enter runsim.sh work2.rt3 \"--routeMgr=3\"
   echo it will be assumed the config file is work2.cfg and the
   echo output files will start with work2.rt2
   exit
fi

rm -f $1.dsc

PARMS=""

if [ $# -gt 1 ]
then
   PARMS=$2
fi

NSDIR=/home/ns3/NS3/ns-3-allinone/ns-3-dev
CONFIG="$1"
echo $CONFIG > configname.txt
HERE=`pwd`
cd $NSDIR
./waf -v --cwd $HERE --run "netsim $PARMS"  2> $HERE/$CONFIG.2
cd $HERE
srt.sh $1
```

2.3.2 dbgsim.sh

To run the program under control of the gdb debugger enter the commands:

```
./dbgsim.sh tcp7
./dbgsim.sh tcp7.olsr
```

Note that the *dbgsim.sh* script does not accept command line parameters. They can be provided at the initial gdb prompt in the *run* command as:

```
./dbgsim.sh tcp7.aodv
gdb>  run "--routeMgr=2"
```

Take care to be sure to included the – prefix to the parameter name.

2.4 Simulation output

The command

```
./runsim.sh tcp7.aodv "--routeMgr=2"
```

will produce the following output files.

2.4.1 tcp7.aodv.2

This is the stderr file that contains contemporaneous messages produced by the netsim simulation during the run. The *grep*

program can be used to extract messages of specific type in time order. 7

```
> grep Tx tcp7.aodv.2
Tx  2     0    5402  20005128
Tx 22     0    5917  20101824
Tx  6     0    5921  20207026
Tx 18     0    5922  20265585
Tx  2     1    5923  20291528
```

2.4.2 tcp7.aodv.act

As packets traverse the simulated network each of the following processing actions is captured in a memory resident packet log and written to a file at the end of the simulation. The action codes are as follows:

0. Packet transmitted by the application component simulated socket layer.
1. Packet processed by the Ipv4 network layer at sending host.
2. Packet forwarded by the Ipv4 network layer at an intermediate hop.
3. Packet processed by the Ipv4 network layer at destination host.
4. Packet consumed by the simulated application
5. (TCP only) Outgoing ack processed by Ipv4 network layer.
6. (TCP only) Incoming ack processed by Ipv4 network layer.

Each record in the packet log consists of the following fields:

- The action code from the list above.
- The connection identifier which is an index into the table of connection records.
- The connection type which is 0 for unicast and 1 for over the top broadcast.
- The node Id of the node that processed the packet.
- The simulated time in microseconds of the processing event.
- The 64 bit unique identifier assigned to the Packet class instance by ns3.
- The logical sequence number of this packet within its specific connection .

The packet log as constructed is naturally sorted on the simulated time of the processing event and the data is written to file *tcp7.aodv.log*.

It is subsequently sorted using the keys: connection identifier, packet Uid, and processing time and processed by the *latency* filter program. The resulting file is *tcp7.aodv.act*.

In this file all of the packets of each single connection are grouped and presented in order of Uid and processing time.

The connection definition entity shown in section 2.1.5 was used to generate this data. The source node is the node with index 1 in node container 2. Consequently the source IP address is 192.168.2.2. Because of the order in which the networks are created intermediate hop at 192.168.2.1 has global node Id 25 and 192.168.2.2 is on global node Id 26.

Because repDest 24 was specified in the connection definition 25 connections – one to each node in the 5×5 grid — are established. Connection 0 is to global node 0 at 192.168.0.1 and connection 2 is to global node Id 2 at 192.168.0.3.

The optimal route from the remote server on global node Id 26 to global node Id 2 is via the two hop connection from the server to the origin of the grid global node Id 0. Because of the order in which the nets were created the intermediate hop has global node Id 27. From node 0 the packets proceeds via node 1 to node 2. This can be observed in the action records associated with record 18 of connection 2.

Act	Id	T	Node	IncLat	AggLat	Time	ByteSeq	PktSeq
0	2	0	26	0	0	10205200	46578	18
1	2	0	26	0	0	10205200	46578	18
2	2	0	27	3502	3502	10208702	46578	18
2	2	0	0	3502	7004	10212204	46578	18
2	2	0	1	364	7368	10212568	46578	18
3	2	0	2	527	7895	10213095	46578	18
4	2	0	2	0	7895	10213095	46578	18
5	2	0	2	200000	207895	10413095	46578	18
6	2	0	26	4607	212502	10417702	46578	18

The *Time* column is the Time in microseconds from the time the connection was initiate until the time the processing occurred. The incremental latency is the time between the previous processing of the packet and the current processing. The aggregate latency is the sum of previous latencies.

Since there are 25 active connections each individual connection is sending at 40 Kbps time between packet origination is almost 300 ms. Consequently, every packet is acked via a delayed ack timeout of 200 ms as shown here.

2.4.3 tcp7.aodv.lat

This file contains the one-way end to end latency for every packet of every connection. The connection Id and PktSeq are as defined above and the one-way latency is given in milliseconds. *AODV* routing produces long latencies early in a connection.

CxId	PktSeq	Lat(ms)
:		
24	1	1001.572
18	1	1771.507
24	2	726.284
:		

2.4.4 tcp7.aodv.perfsum.txt

This file contains final summaries of connection performance for each connection. Drop data is relevant only for UDP transport.

AODV Routing		OLSR Routing	
Connection Id:	12	Connection Id:	12
Total Bytes:	8.98e+04	Total Bytes:	3e+05
Total Packets	62	Total Packets	207
Tx Rate(Kbps)	4e+01	Tx Rate(Kbps)	4e+01
Thpt (Kbps)	12	Thpt (Kbps)	40
Latency (ms)	5189.973	Latency (ms)	8.700
Total Drops	0	Total Drops	0
Drop events	0	Drop events	0
Connection Id:	13	Connection Id:	13
Total Bytes:	0	Total Bytes:	3e+05
Total Packets	0	Total Packets	207
Tx Rate(Kbps)	4e+01	Tx Rate(Kbps)	4e+01
Thpt (Kbps)	0	Thpt (Kbps)	40
Latency (ms)	-nan	Latency (ms)	8.241
Total Drops	0	Total Drops	0
Drop events	0	Drop events	0
Connection Id:	14	Connection Id:	14
Total Bytes:	1.38e+05	Total Bytes:	3.03e+05
Total Packets	95	Total Packets	209
Tx Rate(Kbps)	4e+01	Tx Rate(Kbps)	4e+01
Thpt (Kbps)	18.3	Thpt (Kbps)	40.4
Latency (ms)	6006.302	Latency (ms)	8.011
Total Drops	0	Total Drops	0
Drop events	0	Drop events	0

The above data is for nodes 12, 13, and 14 of the grid. Node 12 is the node in the center of the grid. There are 25 connections from a single external server with one to each of the 25 grid nodes. The simulated environments are identical except for routing system.

For AODV connections to node 13 and node 22 failed likely because of routing failure. The source is configured to send 40 Kbps at the application layer. For the other 23 nodes throughput ranged from 39.4 Kbps at node 24 down to 5.02 Kbps at node 7. Mean latency was best at 326 ms for node 24 but was over 1 second at other nodes with a mean latency of over 8 seconds observed at two nodes.

For OLSR routing all connections were successful and observed throughput matched the offered load. The connection to node 5 had the worse mean latency at 168 ms for reasons not yet determined. All other mean latencies were between 7 and 9 ms.