Volume 2, Issue 5, May 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Network Simulator NS2-2.35

Neha Singh, Prof. Rajeshwar Lal Dua,
Department of ECE
JNU, Jaipur

Vinita Mathur
Department of ECE
JECRC Jaipur
vinitamathur12@gmail.com

Abstract - **A wireless sensor network (WSN) consists of spatially distributed autonomous sensors to monitor physical or environmental conditions. There are many challenges for a sensor network from hardware point of view, to produce low cost and tiny sensor nodes so that energy dissipation is less. Energy is the scarcest resource of WSN nodes, and it determines the lifetime of WSNs. Algorithms and protocols are required to address issues such as lifetime maximization, robustness and fault tolerance and self configuration. Simulation of algorithms can be done using custom platform, network simulators like OPNET, NetSim, and NS2 can be used to simulate wireless sensor network. Here in this paper we will be taking into account an overview of ns2, its installation in ubuntu, and its scripting.**

***Keywords*- Wireless Sensor Network, Simulator, Ns-2, TCP,C++.**

I. Introduction

A wireless sensor network (WSN) consists of spatially distributed autonomous sensors to monitor physical or environmental conditions such a temperature, sound, vibration, pressure and humidity. The WSN as shown in figure 1 is built of nodes, each sensor network nodes has radio transceiver, a microcontroller and a battery.

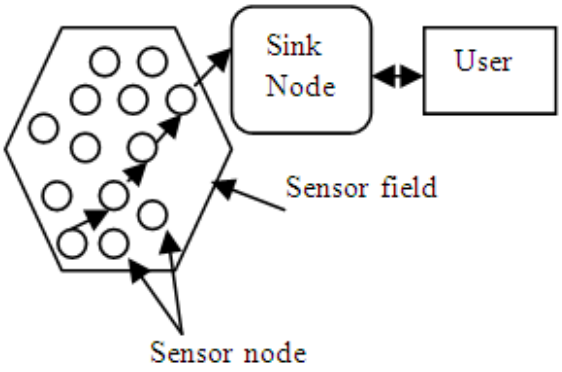

Figure1 Basic Structure of WSN

A sensor node might vary in size and correspondingly its cost. Size and cost constraints on sensor nodes result in corresponding constraints on resources such as energy, memory, computational speed and communications bandwidth. The topology of the WSNs can vary from a simple star network to an advanced multi- hop wireless mesh network. The propagation technique between the hops of the network can be routing or flooding. Applications of WSN lie in the area of air pollution, greenhouse, area monitoring, machine health monitoring, water/wastewater monitoring, agriculture, and structural monitoring. The main characteristics of a WSN include power consumption constrains for nodes, node failures, mobility of nodes, communication failures, heterogeneity of nodes, and ability to withstand harsh environmental conditions and ease of use. Sensor nodes can be imagined as small computers. They usually consist of a processing unit with limited computational power and limited memory, a communication device and a power source [1]. Several standards such as Wireless HART, IEEE1451, and Zigbee/802.15.4 are currently under development by organizations such as WAVE2M, IEEE, Internet Engineering Task force, and International Society of Automation. Energy is the scarcest resource of WSN nodes, and it determines the lifetime of WSNs. Algorithms and protocols are required to address issues such as lifetime maximization, robustness and fault tolerance and self-configuration. Network simulation is a technique where a program models the behavior of a wired or wireless network by calculating the interaction between the different network entities (hosts/routers, data links, packets etc). The behavior of the network and the various applications and services it supports can then be observed in a test lab, various attributes can be modified in a controlled manner to assess how the network would behave under different conditions. Basically a network simulator is a piece of software or hardware that predicts the behavior of the network, without an actual network being present.

Most of the simulators are graphical user interface driven, while some network simulators require input scripts or commands. The network parameters describe the state of the network (node placement, existing links) and the events. An important output of simulations is the trace file. Examples of network simulation software are ns2/ns3, OPNET, NetSim. Network simulators are particularly used to design

various kinds of networks, simulate and then analyze the effect of various parameters on the network performance.

II .NS-2 Overview and Architecture

Latest advances in processing, storage, and communication technologies have advanced the capabilities of small scale and cost effective sensor systems to support numerous applications. A sensor network is defined as an autonomous, multihop, wireless network with non deterministic routes over a set of possibly heterogeneous physical layers. The NS-2 simulation environment offers great flexibility in investigating the characteristics of sensor networks because it already contains flexible models for energy-constrained wireless ad hoc networks. In this environment a sensor network can be built with many of the same set of protocols and characteristics as those available in the real world. The mobile networking environment in NS-2 includes support for each of the paradigms and protocols. The wireless model also includes support for node movement and energy constraints.

NS-2 has many and expanding uses including:

- To evaluate the performance of existing network protocols.
- To evaluate new network protocols before use.
- To run large scale experiments not possible in real experiments.
- To simulate a variety of IP networks.

NS is an object-oriented, discrete event driven network simulator that simulates a variety of IP networks, written in C++ and OTcl . It is primarily useful for simulating local and wide area networks. It implements network protocols such as TCP and UDP, traffic behavior such as FTP, Telnet, Web, CBR and VBR, router queue management mechanism such as Drop Tail, RED and CBR, routing algorithms such as Dijkstra, and more. NS also implements multicasting and some of the MAC layer protocols for LAN simulations. NS develops tools for simulation results display, analysis and converters that convert network topologies to NS formats.

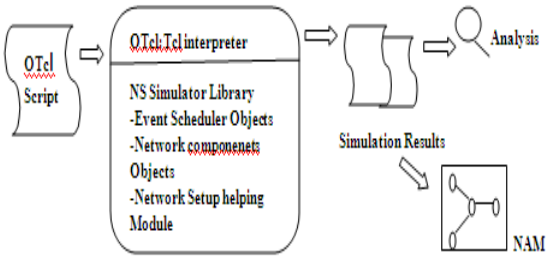

Figure 2 Simplified User's View of NS-2

As shown in figure 2, in a simplified user's view NS is object-oriented Tcl (OTcl) script interpreter that has a simulation event scheduler and network component object libraries, and network setup module libraries. To use NS we program in OTcl script language.

An OTcl script will do the following.

- Initiates an event scheduler.
- Sets up the network topology using the network objects.
- Tells traffic sources when to start/stop transmitting packets through the event scheduler.[2]

NS is written not only in OTcl but in C++ also. For efficiency reason, NS separates the data path implementation from control path implementations. In order to reduce packet and event processing time, the event scheduler and the basic network component objects in the data path are written and compiled using C++. These compiled objects are made available to the OTcl interpreter through an OTcl linkage that creates a matching OTcl object for each of the C++ objects and makes the control functions and the configurable variables specified by C++ object act as member functions and member variables of the corresponding OTcl objects. In this way, the control of the C++ objects is given to OTcl. It is also possible to add member function and variable to a C++ linked OTcl object.

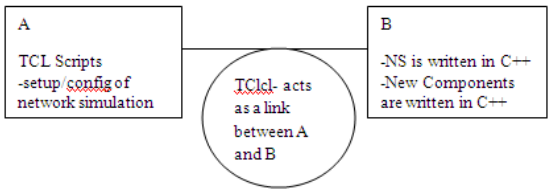

Figure3 TclCl provides the linkage between C++ and OTcl

The objects in the C++ that do not need to be controlled in a simulation or internally used by another object do not need to be linked to OTcl. Likewise, an object can be entirely entitled in OTcl, shows an object hierarchy example in C++ and OTcl. One thing to note in figure 4 is that for C++ object that have an OTcl linkage forming an hierarchy, there is a matching OTcl hierarchy very similar to that of C++.

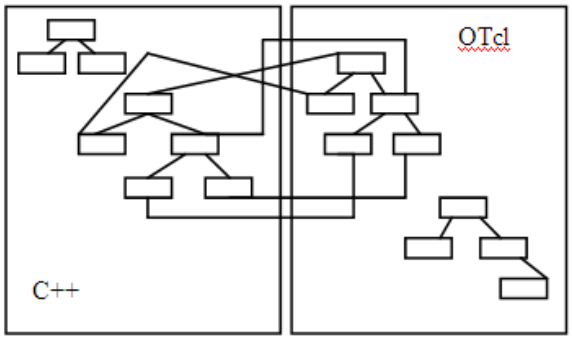

Figure 4 C++ and OTcl: The Duality

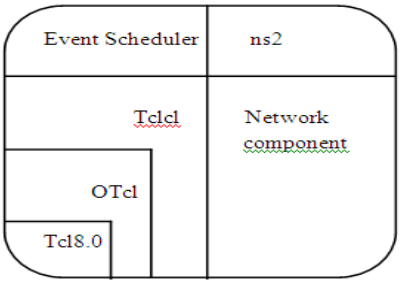

Figure 5 Architectural Views of NS

Figure 5 shows the general architecture of NS. In this figure a general user can be thought of standing at the left bottom corner, designing and running simulation in the

TCL using the simulator object in the OTcl library. The event schedulers and the most of the network components are implemented in the C++ and available to OTcl through an OTcl linkage that is implemented using tclcl. The whole thing together makes NS, which is a OO extended TCL interpreter with network simulator libraries. [3]

III. Installation Steps of NS2 in Ubuntu

Following steps discuss about the installation of NS2-2.35 on ubuntu platform.

1. Download (http://sourceforge.net/projects/nsnam/files/allinone/ns-allinone-2.35/ns-allinone-2.35.tar.gz/download)
2. Unzip or untar it to any folder that contains it using the command from GUI terminal
   $tar-xzvf ns-allinone-2.35.tar.gz
   After unrar it will create ns-allinone-2.35 folder go into the folder by using command
3. $ cd ns-allinone-2.35
4. Run this command
   $sudo apt-get install build-essential autoconf automake libxmu-dev
5. Run this command to start ns-2 installation wait until installation is over
   $./install
6. Once installed the PATH information will have to be provided. Copy the PATH and LD_LIBRARY__PATH variable to .bashrc
   Input the path information in .bashrc file like this
   export PATH=$PATH:<Place your path here>
   export LD_LIBRARY_PATH=$LD_LIBRARY PATH:<place the LD_LIBRARY_PATHS> here.
7. Once done, save the file and close
   Execute the command and run
   ./validate
8. Restart the system open the terminal and type ns
   If % sign appears it means ns-2 is installed in system. [4]

IV. Simulation Script

The ns-2 simulation environment offer great flexibility in investigating the characteristics of sensor networks because it already contains flexible models for energy constrained wireless ad hoc networks. The wireless model also includes support for node movements and energy constraints.

This network consists of 4nodes (n0, n1, n2, n3) as shown in figure6.

The duplex links between n0 and n2, and n1 and n2 have 2 Mbps of bandwidth and 10ms of delay. The duplex link between n2 and n3 has 1.7 Mbps of bandwidth and 20 ms of delay. Each node uses a Drop tail queue, of which the maximum size is 10. A "tcp" agent is attached to n0, and a connection is established to a tcp "sink" agent attached to n3. As default, the maximum size of a packet that a "tcp" agent can generate is 1Kbyte. A tcp "sink" agent generates and sends ACK packets to the sender (tcp agent) and frees the received packets. An "udp" agent that is attached to n1 is connected to a "null" agent attached to n3. A "NULL" agent just frees the packets received. An "ftp" and a "cbr" traffic generator are attached to "tcp"and "udp" agents respectively, and the "cbr" is configured to generate 1 Kbyte packets at the rate of 1 Mbps. The "cbr" is set to start at 0.1 sec and stop at 4.5 sec, and "ftp" is set to start at 1.0 sec and stop at 4.0 sec. [5]

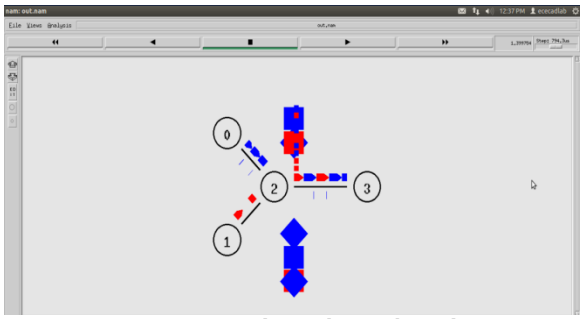

Figure6 Simple wired network topology

Following are the steps for writing a script in ns-2

- Create a new simulator object.
- Turn on tracing [Open your own trace files].
- Create network (Physical Layer).
- Create link and queue (Data Link Layer).
- Define routing protocol.
- Create transport connection (Transport layer).
- Create traffic (Application Layer).
- Insert errors.

Setting up a sensor network in NS-2

Setting up a sensor network in ns-2 follows the same format as mobile node simulations. The best way to create your own simulations is to modify the examples distributed with code.

1. Configure a phenomenon channel and data channel: Phenomenon nodes should emanate in a different channel than sensor nodes in order to avoid contention at the physical layer. All phenomenon nodes should be configured on the same channel, even if they are emanating different types of phenomenon

Set chan-1-[new $val(chal)]
Set chan-1-[new $val(chal)]

2. Configure a MAC protocol for the phenomenon channel: Choose a MAC layer to use for emanating phenomena over the phenomenon channel using 802.11 is not appropriate, since phenomena should be emanating without regard to collisions or congestion control, MAC class is used:

Set val(mac) mac/802.11
Set val(PHENOMmac) mac

3. Configure phenomenon nodes with the PHENOM" routing "protocol: Use node-config, just like with mobile nodes, but specify PHENOM as the routing protocol so the phenomenon is emanated according to the methods defined in phenom\phenomena.cc. also be sure to configure in the channel and MAC layer previously

specified for phenomena broadcast. A sample node configuration statement is shown below.

```
$ns-node-config\
-adhocrouting PHENOM\
-channel$chan-1-\
-II type LL\
-mac type $val(PHENOM mac)\
-if queue type Queue\Droptail\priqueue\
-ifqlen 50\
-anttype Antenna\Omni Antenna \
-phytype phy\wirelessphy\
-topoinstaance $ topo\
-agentTrace ON\
-routerTrace ON\
-mactrace on\
-movement trace on\
-prop Type propogation\Two Ray Ground
```

4. Configue the phenomenon node's pulse rate and type

The two parameters that can be used to customize phenomena are listed below, they are both optional

(i)Pulse rate FLOAT

- Float must be a real number.
- Describes how frequently a phenomenon node broadcasts its presence.
- Defaults to 1 broadcast

(ii) Phenomenon Pattern

- Pattern must be any one of the following keywords: Co, Heavy-GEO, Light-GEO,SOUND,TEST-PHENOMENON corresponding Monooxide, heavy seismic activity, light seismic activity, audible sound, and some other generic phenomenon.
- This option is mostly useful for simulations invoking multiple phenomenon nodes, so that it is easier to distinguish who a sensor node is detecting by looking at the ns trace file.

5. Configure sensor nodes: Sensor nodes must be configured with the PHENOM channel attribute and the –channel attribute. PHENOMchannel must be the same as the channel we configured the phenomenon node with. The other channel is the channel that will be used communicating sensor reports. Sensor node configuration must also specify a MAC protocol (such as MAC/802-11) for the channel shared with other wireless node. This is done with the –PHENOMacTYPE and – macTYPE attributes.-PHENOMmacTYPE should be the same as the macTyppe used in other nodes participating in the IP network. If desired,a sensor node can be configurd so that a specified amount of energy will be deducted from its energy reserve each time it receives a phenomenon broadcast.To set this up,include the following parameters in the sensor node's node config routine:

```
-energy model Energy Model\
-rxpower 0.175\
-txpower 0.175\
-sense power 0.0\
-initial Energy 0.5
```

6. Configure non-sensor nodes, such as data collection points, or gateways for the sensor network nodes should not be configured with a PHENON channel, since their only interface is to the MANET network.. This is done with the –PHENOM channel 'OFF' attribute, as follows:

```
$ns_node-coding\
-adhocRouting$val(rp)\
-channel$chan-2-\
-PHENOMchannel 'off'
```

7. Attach sensor agents: Create a sensor for each sensor node, and attach agent to its respective node. Also specify that all packets coming in from the PHENOM channel should be received by the sensor agent. In the following example, $i would represent the node number for the sensor node currently being configured.

```
Set sensor_($i) [new\Agent\$sensor_($i)
# specify the sensor agent
# as the up-target for the
# sensor node's link layer
# configured on the PHENOM
#interface, so that the
#sensor agent handles the
# received PHENOM packets
# Instead of any other agent
# attached to the node.
[$node_ ($i)set11_(1)]\up-target $sensor_($i)
```

8. Attach a UDP agent and sensor application to each node(optional)

How the sensor nodes react once detect their target phenomenon is a behavior that should be defined in the sensor application. One such application might involve sensor nodes alerting a data collection point via UDP with information about the phenomenon. [6]

V. Conclusion

Network simulator involves addressing a wide range of issues steaming from limited energy reserves, computation power, communication capabilities and self managing sensor nodes. The NS-2 simulation environment is a flexible tool for network engineers to vestigate how various protocols perform with different topologies and configurations. NS-2.35 is an object-oriented event-driven simulator. NS provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks. This paper discusses about need for wireless sensor networks, one of its simulator ns-2, and its installation in linux.

References

[1]I.F.Akyildiz,W.Su,Y.Sankarasubramaniam,E.Cayirci. Wireless Sensor Networks:a survey,Computer Networks 38 (2002) 393-422.

[2]Paul Meeneghan and Declan Delaney,An Introduction to NS, NAM and OTcl scripting,April 2004.

[3]Thammakit Sriporamanont and gu Liming, Wireless Sensor Network Simulator, Technical report, IDE0602,January 2006.
[3]Harsh Sundai, Haoyue Li, Vijay Mansoor Alam Davabhaktuni,and Prabir Bhattachrya,Wireless Sensor Network Simulators A Survey and Comparisions, International Journal of Computer Networks (IJCN), Volume(2):Issue(5).
[4]http://opensourcetips.org/linux/ubuntu/827-install-ns2-in-ubuntu-in-6-easy-steps
[5] Jae Chung and Mark Claypool, NS by Examples, WPI Computer Science.
[6]S.Koteswararao,Dr.M.Sailaja,P.Ramesh,E.Nageswararao,V.Rajesh,Sensor Networks Simulation in NS2.26,IJECT Vol.2 ,SP-1,Dec. 2011.

