CHAPTER 8

PROBLEMS WITH READYMADE ROUTING PROTOCOL SIMULATORS

8.1 NS2

8.1.1 Introduction

The NS2 [23, 25, 41] is a simulator, written in C++, with a TCL interpreter as a front-end. Ns-2 is a discrete event simulator, which means that events (e.g., packets to send, timeouts, etc.) are scheduled in a global event queue according to their time of execution. When a simulation is run the simulator removes events from the head of the queue, moves the simulator time to that of the currently removed event and executes it. When done, it continues to the next event and so forth. Each simulation is defined by a scenario that contains a number of predefined events that define the scenario. The NS2 simulator supports a class hierarchy in C++ and a similar class hierarchy within the TCL interpreter. The two hierarchies are very closely related to each other. The source of this hierarchy is the class TclObject. Users create the new simulator objects through the interpreter and are closely reflected by a equivalent object in the compiled hierarchy.

Ns-2 scenarios are implemented in TCL-scripts that contain the commands to initialize the simulator and to create the nodes and their configuration. Each simulation run generates a trace file containing all the data packets that are sent between the nodes during the course of the scenario. By analyzing this file it is possible to determine the performance effect of parameter variations, different routing protocols and more. Ns-2 is run from a UNIX terminal window. To edit the scripts used in this assignment you need a text editor, e.g., Emacs, Vi or similar.

The interpreted class hierarchy is involuntarily established through the methods that are written in the class TclClass. User defined objects are mirrored through the methods that are written in the class TclObject. Some other hierarchies are also in the C++ code and TCL scripts which are not mirrored in the manner of TclObject.

Why use two languages**?** NS2 simulator uses the two different languages because it has two different kinds of objectives it needs to accomplish. On one hand, simulations of protocols need a programming language that can efficiently and effectively manipulate bytes, packet headers, and implement algos. that run over large data sets. For such tasks, the run-time speed is very important and run time is less important.

On the other hand, a very large area of network study entails very minutely changing parameters or configurations, or quickly exploring many scenarios simultaneously. In such cases, iteration times (change the model and re-run) are more important. Since the configuration runs once, running time of this part of the objective is less vital.

NS2 fulfills both of these requirements with two languages viz. C++ and OTcl, as C++ is fast to run but sluggish to change thus, making it suitable for detailed protocol implementation. OTcl runs comparatively slower than C++ but it can be changed very quickly and efficiently, making it best for simulation design. NS2 provides bond to make objects and variables appear on both the languages.

The code to edge with the interpreter exists in a separate directory, tclcl. The rest of the simulator code resides in the directory, ns-2. We will use the ~tclcl/hfilei to point to a particular hfilei in the Tcl directory. Likewise, we will use ~ns/hfilei to point to a particular hfilei in the ns-2 directory.

There are various classes defined in ~tclcl/. We will only focus on the six classes that are used in ns2 simulator: The Class Tcl containing the methods used by the C++ code to access the interpreter. The TclObject class is the base class for all simulator objects that are also reflected in the compiled hierarchy. The TclClass,

defines the interpreted class hierarchy, and the methods to permit the user to instantiate TclObjects. TclCommand is used to define global interpreter commands. EmbeddedTcl class contains the methods to load higher level commands that help in making the configurations of simulations easier. And lastly, the InstVar class consists of methods to access C++ member variables as OTcl instance variables.

8.1.2 How do I make my own protocol for NS2

Let us suppose we are now going to apply a new MANET routing protocol called - protoname.

To allocate our code we will firstly create a new directory by the name protoname inside your NS2 base directory. We will create five files there:

protoname.h This is the header file where will be defined all useful timers and the routing agent which performs protocol's functionality.

protoname.cc In this file are actually implemented all timers, routing agent and Tcl hooks.

protoname pkt.h Here all packets protoname protocol needs to exchange among nodes in the manet are declared

protoname rtable.h Header file where our own routing table is declared.

protoname rtable.cc Routing table implementation.

To implement a routing protocol in NS2 simulator one must create an agent by inheriting from Agent class. This is the main class in which we will have to program in order to implement our routing protocol. In addition, this class offers a linkage with Tcl interface, so that we will be able to monitor our routing protocol via simulation scripts written in Tcl language.

The routing agent will maintain the internal state and a routing table. Internal state can be shown as a new class or a bunch of attributes inside the routing agent. We will deal with the routing table as a new class, protoname_rtable.

Also our new protocol must define a minimum of one new packet type which will show the format of its control packets. As we said these packet types are defined in protoname/protoname_pkt.h. When the protocol needs to send packets occasionally or after some time from the occurrence of an event, it is useful to count on a Timer class.

The class Trace is the base class for writing log files with information about the happenings during the simulation. The roaming node scenario can be run using AODV as the default routing protocol by passing the scenario file as input to ns-2. The optional prot argument selects either AODV or OLSR routing (AODV is default).

$ ns roaming-node-DK2.tcl -prot [aodv|olsr]

Running this command in a terminal window will produce two output files. The first is the .tr file, which is the main trace file and the other is a .nam file, which is used to visualize the scenario using the nam animation tool. If you want to view an animation of the node movements you can run nam, e.g.:

$ nam roaming-node-udp-aodv.nam

However, the .tr trace file is the important file for this assignment and is used for the analysis.

8.1.3 Problems with NS2

- To implement a routing protocol in NS2 we have to create too many files that are not easy to implement.
- In NS2, open source is used. To use this open source we must know the programming used.
- It is freeware so training is not easily available.
- Editing is very difficult in NS2.
- Compilation and Execution is tough.
- AWK file is created for ouput. To see output we have to change awk file.

- TCL script language is to be learned to change the parameters.

8.2 QUALNET

8.2.1 Introduction

QualNet Simulator [26, 27] is a simulator for very large, different networks and the distributed applications that run on those networks. QualNet Simulator is a very scalable simulation engine, accommodating high-fidelity models of networks of thousands of nodes. QualNet Simulator makes good use of computational assets and models large-scale networks with heavy traffic and mobility, in rational simulation times.

QualNet has the following features:

- Quick model set up with a very powerful Graphical User Interface for custom code development and reporting options
- Instant playback of simulation results to reduce unwanted model executions
- Quick simulation results for meticulous exploration of model parameters
- Scalable up to thousands of nodes
- Real-time simulation for man-in-the-loop and hardware-in-the-loop models
- Supported on many platforms

Qualnet consists of different tools that provide specific features. These tools are classified as -

- **QualNet Scenario Designer**

  It is a graphical tool that provides an insightful model set up capability and is used to create and design experiments in QualNet. The Scenario Designer helps a user to define the geographical circulation, physical connections and the functional parameters of the network nodes, all using insightful click and drag tools, and to define network layer protocols and traffic individuality for each node.

- **QualNet Animator**

  It is used to execute and real-time animate experiments created in the Scenario Designer. Using the Animator a user can view traffic flow through the network and create dynamic graphs of critical routine metrics as a simulation is running.

- **QualNet 3D Visualizer**

  QualNet 3D visualizer is similar to QualNet Animator, but it provides the added capability of visualizing scenarios in 3D.

- **QualNet Analyzer**

  It is a statistical graphing tool that displays network data generated from a QualNet experiment. Using the Analyzer, one can view statistics as they are being generated, as well as compare results from diverse experiments.

- **QualNet Packet Tracer**

  It is a packet-level visualization tool for viewing the contents of packets as they travel up - down the protocol stack.

- **QualNet Protocol Stack**

  QualNet uses a layered architecture quite similar to that of the TCP/IP network protocol stack. Within that architecture, data moves between nearby layers. QualNet's protocol stack consists of, from top to bottom, the Application, Transport, Network, Link (MAC) and Physical Layers. Adjacent layers in the protocol stack communicate via well-defined APIs, and generally, layer communication occurs only among adjacent layers. E.g.: Transport Layer protocols can get and pass data to and from the Application and Network Layer protocols, but cannot do so with the Link (MAC) Layer protocols or the Physical Layer protocols.

8.2.2 Problems with Qualnet

- In Qualnet, there is very long and complex code for the protocols. Eg. AODV codes are of 196 pages.
- The output parameters cannot be changed.
- Editing is also a very tough task in Qualnet.

8.3 Summary

In this chapter we will discuss the following point is:

- NS2
- How do I make my own protocol for NS2
- QUALNET
- Problems with NS2
- Problems with Qualnet