Chapter 7

7. Network Simulator Version 2

7.1 Introduction

Simulations are inexpensive to perform and do not need any infrastructure. By creating virtual networks in lab, one can deploy along with testing, optimizing and integrating next generation network technologies rather than using physical test beds. Using simulations, complex scenarios, new ideas can be verified easily and results can be obtained quickly.

NS2 is a network simulator that is most extensively used event-driven. It is an open source network simulator that simulates dynamic characteristics of a number of wired and wireless networks. It implements number of network protocols and allows users to study their behavior by setting various parameters. NS2 offers a command ns which takes as argument the name of a Tcl scripting file. The command once used creates a simulation trace file which can then be utilized to plot a x-graph or to generate animations.

NS2 makes is developed using two languages: C++ and Object-oriented Tool Command Language (OTcl). C++ is used as a backend for proficiently changing packet headers, bytes for implementing algorithms and OTcl is used as a front end for examining a number of network
scenarios by modifying various parameters. The C++ and the OTcl are linked together using TclCL.

### 7.2 NS2 Simulation Steps

Simulation scenarios in NS2 are created in three steps:

1. The first step is to find the objectives of simulation, configuration of the network, assumptions of various performance parameters and type of expected results.
2. The second step involves creating and configuring the network components, starting and running the events until the simulation time specified in Tcl simulation script.
3. The third step involves checking the performance of the network under simulation i.e analyzing the trace files.

The simulator on the whole is characterized by a TCL class Simulator, which specifies number of interfaces for setting up a simulation, for selecting the type of event scheduler used to run the simulation. At the beginning of the TCL simulation script, we create an instance of the class Simulator and then invoke different methods to create nodes, network topologies, and set up other parameters of the simulation.
The Simulator instance is created by using the command “new Simulator” which returns a Simulator handle and is stored in a variable ns.

### 7.2.1 Creating a Scenario

```plaintext
set ns [new Simulator]
```

For analyzing the performance of the network with various protocols, NS2 opens files `out.tr` and `out.nam` for collecting trace information for a text-based regular trace and for a NAM trace respectively.

The basic primitive for creating a node is

```plaintext
set ns [new Simulator]
```

```plaintext
$ns node
```

```plaintext
$ns duplex-link <node1> <node2> <bw> <delay> <qtype> <args>
```

The above procedure connects node1 and node2 with a bi-directional link [14] where node1 is the source node and node2 is the destination node, bw is the bandwidth of the link in bits per second, delay is the propagation delay associated with the link in seconds and qtype is the type of queue between node1 and node2 and depending on the queue type different arguments are passed through <args>.

When a node sends CBR traffic to another node, then the underlying transport protocol is User Datagram Protocol (UDP) which is a connectionless transport protocol, used when applications require quick delivery rather than accuracy. When a node transfers data to
another node using a file transfer protocol (FTP), then the underlying transport protocol is Transmission Control Protocol (TCP), which is a connection-oriented transport protocol, used when applications require accuracy rather than quick delivery. In NS2, the sending objects of CBR and FTP protocols are UDP agent and a TCP agent, while the receivers are a Null agent and a TCP sink agent, respectively.

**Constant Bit Rate** A CBR object is present in the OTCL class[14] Application/Traffic/CBR. Its member variables are:

- `rate_` the sending rate
- `interval_` (Optional) interval between packets
- `packetSize_` the constant size of the packets generated
- `random_` flag indicating whether or not to introduce random “noise” in the scheduled departure times (default is off)
- `maxpkts_` the maximum number of packets to send

example: Creation of a new CBR traffic generator:

```bash
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 48
$cbr1 set rate_ 64Kb
$cbr1 set random_ 1
```
A simulation should either specify rate or an interval (but not both) for a CBR object.

An agent connects an application and a low-level network. An agent is an endpoint, responsible for constructing and destroying packets. There are two important agent types: routing agents and transport layer agents. A routing agent constructs and accepts routing control packets, and instructs routing protocols to take action accordingly. A transport-layer agent connects an application to a low level network, and checks congestion and reliability of data depending on the type of transport layer protocol (e.g., UDP or TCP).

Based on the application layer protocol, a sending agent builds packets and sends them to a receiving agent through a low-level network. At the top level, a Constant Bit Rate (CBR) application sends data periodically where a UDP-agent creates UDP-packets. The UDP agent stores port numbers and IP addresses of both the source and destination in the packet header, and forwards the packet along with the header to the attached node. Using the routing table entries, the network sends the packet to the destination node, which then hands over the packet to the agent attached to the port. A Null receiving agent destroys the received packet.
An agent can be used either as a sending agent or as a receiving agent. A sending agent connects to both an application and a low-level network, while a receiving agent does not have a connection to an application. Agent configuration consists of four main steps:

(i) Create sending and receiving agent, and an application using new.
(ii) Attach agents to the application using Command attach-agent.
(iii) Attach agents to the nodes using attach-agent.
(iv) Connect the sending agent with the receiving agent using command connect.

The following commands are used to set up UDP agents in OTCL scripts:

set udp0 [new Agent/UDP]
This command creates an instance of the UDP agent.

$ns attach-agent <node> <agent>
This command attaches any <agent> to a given <node>.

$traffic-gen attach-agent <agent>
This a class Application/Traffic/<traffictype>method which connects the traffic generator to the given <agent>.

$ns connect <src-agent> <dst-agent>
This command connects the source and destination agents.
Example:

```tcl
set ns [new Simulator]
set n1 [ns node]
set n2 [ns node]
set n3 [ns node]

$ns duplex-link $n1 $n2 5Mb 2ms DropTail
$ns duplex-link $n2 $n3 5Mb 2ms DropTail
$ns duplex-link $n1 $n3 5Mb 2ms DropTail

set udp1 [new Agent/UDP]      # Create a UDP agent
set null1 [new Agent/Null]
set cbr1 [new Application/Traffic/CBR]   # Create a CBR traffic source
$cbr1 attach-agent $udp1
$ns attach-agent $n1 $udp1
$ns attach-agent $n3 $null1
$ns connect $udp1 $null1
$cbr1 set packetSize_ 1000
$cbr1 set rate_ 2Mb
```

In the above example, the simulator is initialized firstly, then 3 nodes are created and linked to each other using duplex-link and then following that sending and receiving agents are created and attached to applications and nodes and then sending and receiving agents are connected with connect command. For the CBR connection, packet size and generation rate are set to 1000 bytes and 2 Mbps, respectively.
$ns at <time> <event> - This command executes an <event> (which is normally a piece of code) at the specified <time>.

Ex: $ns at 10.0 "$cbr1 start"

$ns run - The scheduler is started with this command

Packet tracing which can be text-based or NAM based, accounts for details of packet flow during a simulation. Packet tracing which is text-based is initiated by executing "$ns trace-all $file", where file is a handle associated with the file which stores the tracing text.

7.2.2 Packet Tracing (Text-Based)

+ 0.110419 1 2 tcp 1040 ------- 2 1.0 4.0 5 12
+ 0.110419 1 2 tcp 1040 ------- 2 1.0 4.0 6 13
- 0.110431 1 2 tcp 1040 ------- 2 1.0 4.0 5 12
- 0.110514 1 2 tcp 1040 ------- 2 1.0 4.0 6 13
r 0.11308 0 2 cbr 1000 ------- 1 0.0 3.0 2 8
+ 0.11308 2 3 cbr 1000 ------- 1 0.0 3.0 2 8
- 0.11308 2 3 cbr 1000 ------- 1 0.0 3.0 2 8
r 0.11316 0 2 cbr 1000 ------- 1 0.0 3.0 3 9
+ 0.11316 2 3 cbr 1000 ------- 1 0.0 3.0 3 9
- 0.113228 2 3 cbr 1000 ------- 1 0.0 3.0 3 9
r 0.115228 2 3 cbr 1000 ------- 1 0.0 3.0 0 6
r 0.115348 1 2 tcp 1040 ------- 2 1.0 4.0 3 10
+ 0.115348 2 4 tcp 1040 ------- 2 1.0 4.0 3 10
- 0.115348 2 4 tcp 1040 ------- 2 1.0 4.0 3 10
r 0.115376 2 3 cbr 1000 ------- 1 0.0 3.0 1 7
r 0.115431 1 2 tcp 1040 ------- 2 1.0 4.0 4 11
...

The following are the fields which correspond to one line in a text-based trace file:

- The type identifier
- time
- the source node
- destination node
- Type of packet
- Packet size
- Flags
- Flow Id
- Source address
- Destination address
- Sequence number
- Packet unique id

The 1st field corresponds to four event types: r (received), + (enqueued), (dequeued), and d (dropped). The 2nd field indicates time at which the event occurred. Fields 3 and 4 are the source and destination nodes.
Fields 5 and 6 specify packet type and packet size. The 7th field is a series of flags, specifying any unusual behavior and the output "-------" denotes no flag. Following the flags is a packet flow ID. Fields 9 and 10 specify the source and the destination addresses, respectively, in the form of [node.port]. Next field is packet sequence number and the last field is a packet unique ID.

Trace file is useful only when significant analysis is performed on the data. After the simulation, one usually extracts data of interest from the trace file and further analyzes it. For example, average throughput can be calculated by extracting specific columns from the trace file. AWK and Perl are the two of the most popular languages that help in this process.

7.3 NAM - The Network Animator

Nam is a TCL based animation tool for visualizing real world packet trace data following the simulation of a network. NAM records simulation details in a text file, and then make use of the text file to play the simulation using animation. NAM trace is initiated by the command “$ns namtrace-all $file1”, where file1 is a handle associated with the file, which stores the NAM trace information.

The first step in using NAM is to produce the NAM trace file which contains topology information, e.g., nodes, links, as well as packet traces. The trace file when generated can be animated by NAM. When the
animator is started, it will read the trace file, construct the topology, pop up a window and then pauses at time 0. We can control many aspects of animation through its user interface

There are many visualization features like coloring the nodes, coloring the links, giving different shapes to the nodes etc.
8. Conclusion and Future Enhancements

Cache invalidation strategies use broadcasting to distribute the information to the large population of MHs for effective cache consistency in the MWCN. Although scalable broadcasting creates lot of traffic on the network. This proposed model of MDD to maintain cache consistency eliminates the traffic problem and improves the availability of data. The use of an efficient CRP namely SNN-CRP increases the cache hit ratio on DTAC by 15 percent as shown in chapter 6 further increasing the system performance. The server is relieved from the burden of servicing a group of clients because DTA can service some of the clients on behalf of server. The server can use its valuable time for other critical activities. The proposed strategy reduces the number of uplink requests to the server by the clients and also guarantees cache consistency among the data items present in clients’ cache and the server. The model implemented also minimizes the delay associated with answering query in MWCN and provides the sleeping MHs who missed the broadcast of Invalidation Reports with the latest data updates instantly. Thus all the possible resources are utilized in the best possible manner by incorporating the model discussed in Section 2.

The DTA cache organization and management can be enhanced in the future work. Indexing techniques can be used by the DTA to the provide instant information to the clients in case the group is large. Prediction Algorithms can be used to implement CRP for DTAC. The
methods of enhancements given above can increase cache hit ratio of DTAC and thus improve the overall system performance.

Adhoc Multicast Routing can be used route the data from DTA effectively in order to manage small and large groups. The Integration of adhoc networks to the internet and Fairness in case of congestion during multicasting also requires attention for the future study that can reduce query delay to great extent.

The effectiveness of security issues imposed on DTA and energy efficient algorithms used can be open areas of research in future study.
References


36. Yu Huang, Jiannong Cao, Beihong Jin, Xianping Tao, Jian Lu, Member and Yulin Feng  “*Flexible Cache Consistency Maintenance over Wireless Ad Hoc Networks*” IEEE transactions on parallel and distributed systems, Vol. 21, no. 8, August 2010.
