Revised On: 15/12/2016

Problem Statement:

Study of any network simulation tools - To create a network with three nodes and establish a TCP connection between node 0 and node 1 such that node 0 will send TCP packet to node 2 via node 1.

Theory:

Generally speaking, network simulators try to model the real world networks. The principal idea is that if a system can be modeled, then features of the model can be changed and the corresponding results can be analyzed. As the process of model modification is relatively cheap than the complete real implementation, a wide variety of scenarios can be analyzed at low cost (relative to making changes to a real network). Network simulator always contain the

However, network simulators are not perfect. They can not perfectly model all the details of the networks. However, if well modeled, they will be close enough so as to give the researcher a meaningful insight into the network under test, and how changes will affect its operation.

In communication and computer network research, **network simulation** is a technique whereby a software program models the behavior of a network either by calculating the interaction between the different network entities (routers, switches, nodes, access points, links etc.). Most simulators use discrete event simulation - the modeling of systems in which state variables change at discrete points in time. The behavior of the network and the various applications and services it supports can then be observed in a test lab; various attributes of the environment can also be modified in a controlled manner to assess how the network / protocols would behave under different conditions.

A **network simulator** is software that predicts the behavior of a computer network. Since communication Networks have become too complex for traditional analytical methods to provide an accurate understanding of system behavior, network simulators are used. In simulators, the computer network is modeled with devices, links, applications etc. and the performance is analysed. Simulators come with support for the most popular technologies and networks in use today such as Wireless LANs, Mobile Adhoc Networks, Wireless Sensor Networks, Vehicular Adhoc Networks, Cognitive Radio networks, LTE / LTE- Advanced Networks, Internet of things (IOT) etc

Type of network simulators

Different types of network simulators can be categorized and explained based on some criteria such as if they are commercial or free, or if they are simple ones or complex ones.

1. Commercial and open source simulators

Some of the network simulators are commercial which means that they would not provide the source code of its software or the affiliated packages to the general users for free. All the user s have to pay to get the license to use their software or pay to order specific packages for their own specific usage requirements. One typical example is the OPNET [OPNET]. Commercial simulator has its advantage and disadvantage. The advantage is that it generally has complete and up-to-date documentations and they can be consistently maintained by some specialized staff in that company. However, the open source network simulator is disadvantageous in this aspect, and generally there are not enough specialized people working on the documentation. This problem can be serious when the different versions come with many new things and it will become difficult to trace or understand the previous codes without appropriate documentations.

On the contrary, the open source network simulator has the advantage that everything is very open and everyone or organization can contribute to it and find bugs in it. The interface is also open for future improvement. It can also be very flexible and reflect the most new recent developments of new technologies in a faster way than commercial network simulators. We can see that some advantages of commercial network simulators, however, are the disadvantage for the open source network simulators. Lack of enough systematic and complete documentations and lack of version control supports can lead to some serious problem s and can limit the applicability and life-time of the open source network simulators. Typical open source network simulators include NS2 [NS2][NS2-wiki], NS3[NS3]. We will introduce and analyze them in great detail in the following sections.

Table 1 Network simulators

	Network simulators name
Commercial	OPNET, QualNet
Open source	NS2, NS3, OMNeT++, SSFNet, J-Sim

Note that limited by the length, not all of them will be introduced in detail in this paper. However, we will focus on some typical ones and introduce others briefly.

2 Simple vs. complex

Currently there are a great variety of network simulators, ranging from the simple ones to the complex ones. Minimally, a network simulator should enable users to represent a network topology, defining the scenarios, specifying the nodes on the network, the links between those nodes and the traffic between the nodes. More complicated systems may allow the user to specify everything about the protocols used to process network traffic. Graphical applications also allow users to easily visualize the workings of their simulated environment. Some of them may be text-based and can provide a less visual or intuitive interface, but may allow more advanced forms of customization. Others may be programming-oriented and can provide a programming framework that allows the users to customize to create an application that simulates the networking environment for testing.

TCP Connection Establishment and Termination

To aid in our understanding of the connect, accept, and close functions and to help us debug TCP applications using the netstat program, we must understand how TCP connections are established and terminated, and TCP's state transition diagram.

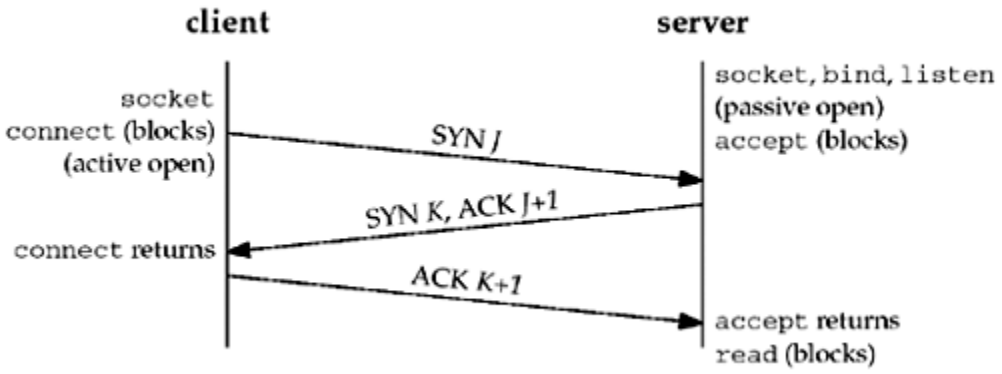
Three-Way Handshake

The following scenario occurs when a TCP connection is established:

1. **The server must be prepared to accept an incoming connection. This is normally done by calling socket, bind, and listen and is called a *passive open*.**
2. The client issues an *active open* by calling connect. This causes the client TCP to send a "synchronize" (SYN) segment, which tells the server the client's initial sequence number for the data that the client will send on the connection. Normally, there is no data sent with the SYN; it just contains an IP header, a TCP header, and possible TCP options (which we will talk about shortly).
3. The server must acknowledge (ACK) the client's SYN and the server must also send its own SYN containing the initial sequence number for the data that the server will send on the connection. The server sends its SYN and the ACK of the client's SYN in a single segment.
4. The client must acknowledge the server's SYN.

The minimum number of packets required for this exchange is three; hence, this is called TCP's *three-way handshake*. We show the three segments in Figure 2.2.

Figure 2.2. TCP three-way handshake.



We show the client's initial sequence number as *J* and the server's initial sequence number as *K*. The acknowledgment number in an ACK is the next expected sequence number for the end sending the ACK. Since a SYN occupies one byte of the sequence number space, the acknowledgment number in the ACK of each SYN is the initial sequence number plus one. Similarly, the ACK of each FIN is the sequence number of the FIN plus one.

An everyday analogy for establishing a TCP connection is the telephone system [Nemeth 1997]. The socket function is the equivalent of having a telephone to use. Bind is telling other people your telephone number so that they can call you. Listen is turning on the ringer so that you will hear when an incoming call arrives. Connect requires that we know the other person's phone number and dial it.accept is when the person being called answers the phone. Having the client's identity returned by accept (where the identify is the client's IP address and port number) is similar to having the caller ID feature show the caller's phone number. One difference, however, is that accept returns the client's identity only after the connection has been established, whereas the caller ID feature shows the caller's phone number before we choose whether to answer the phone or not. If the DNS is used, it provides a service analogous to a telephone book. Getaddrinfo is similar to looking up a person's phone number in the phone book. getnameinfo would be the equivalent of having a phone book sorted by telephone numbers that we could search, instead of a book sorted by name.

TCP Connection Termination

While it takes three segments to establish a connection, it takes four to terminate a connection.

1.One application calls close first, and we say that this end performs the *active close*. This end's TCP sends a FIN segment, which means it is finished sending data.

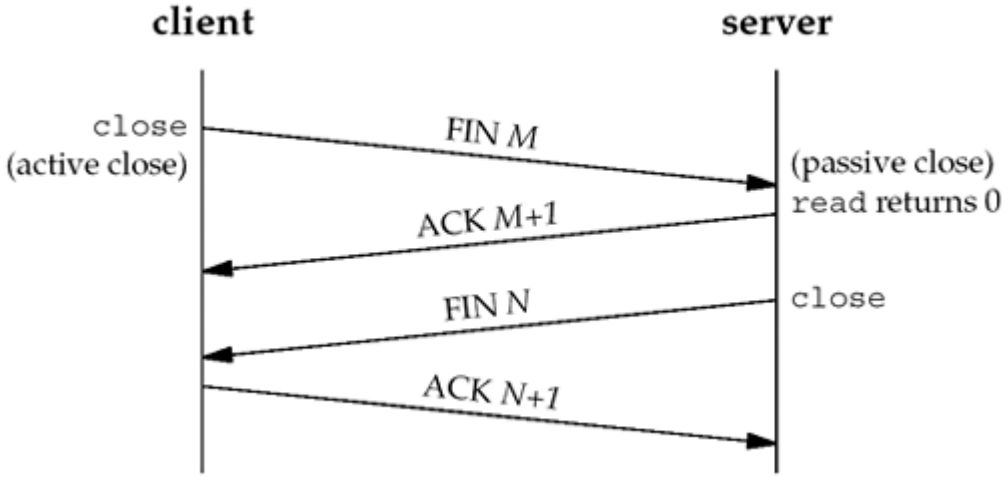
2.The other end that receives the FIN performs the *passive close*. The received FIN is acknowledged by TCP. The receipt of the FIN is also passed to the application as an end-of-file (after any data that may have already been queued for the application to receive), since the receipt of the FIN means the application will not receive any additional data on the connection.

3.Sometime later, the application that received the end-of-file will close its socket. This causes its TCP to send a FIN.

4.The TCP on the system that receives this final FIN (the end that did the active close) acknowledges the FIN.

Since a FIN and an ACK are required in each direction, four segments are normally required. We use the qualifier "normally" because in some scenarios, the FIN in Step 1 is sent with data. Also, the segments in Steps 2 and 3 are both from the end performing the passive close and could be combined into one segment. We show these packets in fig

Figure 2.3. Packets exchanged when a TCP connection is closed.



NS2 Installation steps:

STEP 1: Download Ns2 package

STEP 2: paste it in any file system in Linux

STEP 3: Open Terminal and Navigate to folder in which ns package is pasted

STEP 4: Please check gcc & g++ libraries are installed

STEP 6: untar the file using Command >>> [tar -zxvf ns-allinone-2.34.tar.gz]

STEP 7: navigate in to folder using command >>> [cd ns-allinone-2.34]

STEP 6: type the command >>> [./install]

NS 2 INSTALLATION IN LINUX:

STEP 1:

You should be logged in as an root because root has all rights to configure the system.

STEP 2:

Copy the ns-all –in-one-2.3.4.tar.gz file in to the root

STEP 3:

Untar the gun zip file using the tar command which is given in the screen shot.

STEP 4:

Now install the ns 2 by using ./install command.

TO RUN NS FILE:

STEP 1:

Create the path which is like given below

PATH=$PATH: /root/ns all-in-one/bin

STEP 2:

To run ns2 type the command ns(filename)which file you are going to run.

TCL Script for Simple Simulation Example

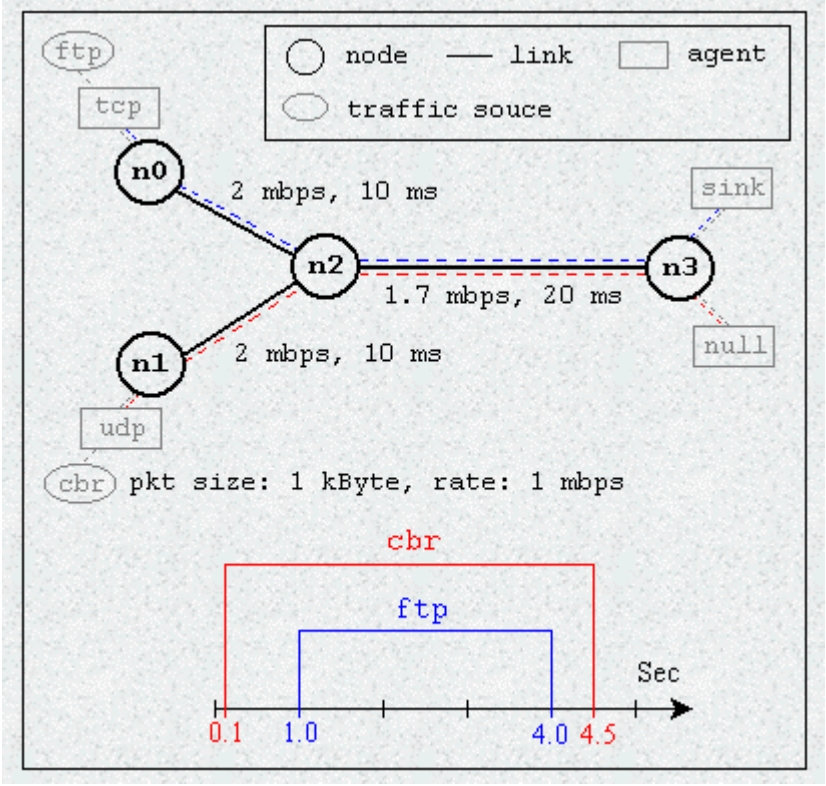


Figure.A
Simple Network Topology and Simulation Scenario

This network consists of 4 nodes (n0, n1, n2, n3) as shown in above figure. The duplex links between n0 and n2, and n1 and n2 have 2 Mbps of bandwidth and 10 ms of delay. The duplex link between n2 and n3 has 1.7 Mbps of bandwidth and 20 ms of delay. Each node uses a DropTail 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. A "udp" agent that is attached to n1 is connected to a "null" agent attached to n3. A "null" agent just frees the packets received. A "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.

In this simple example, there are two parallel TCP sessions sharing a single bottleneck link between
node 1 and node 2 (of capacity 700kb).

Create a dumbbell topology

$ns duplex-link $s(0) $n(0) 1Mb 5ms DropTail

```
$ns duplex-link $s(1) $n(0) 1Mb 5ms DropTail
$ns duplex-link $n(0) $n(1) 1Mb 20ms RED/myRIO
$ns duplex-link $n(1) $n(2) 700Kb 25ms RED/myRIO
$ns duplex-link $n(2) $r(0) 1Mb 5ms DropTail
$ns duplex-link $n(2) $r(1) 1Mb 5ms DropTail
```

One of the sessions runs from node 3 to node 5, periodically transmitting 1000 packets. Therefore, all flows in this session are long according to our definition. On the contrary, in the session from node 4 to node 6, only 4 packets are transmitted in each flow. The simulation script is shown below:

```
# Create sessions
proc build-fore-tcp { idx size intv stime } {
        global ns ftcp fsink
        set ftcp($idx) [new Agent/TCP/Newreno]
        set fsink($idx) [new Agent/TCPSink]
        $ns at $stime "start-conn 1 $idx $intv $size"
}
proc start-conn { firsttime idx intv size } {
        global ns ftcp fsink s r
        set now [$ns now]
        if { $firsttime == 0 } {
                $ns detach-agent $s([expr $idx%2]) $ftcp($idx)
                $ns detach-agent $r([expr $idx%2]) $fsink($idx)
                $ftcp($idx) reset
                $fsink($idx) reset
        }
```

```
        $ns attach-agent $s([expr $idx%2]) $ftcp($idx)
        $ns attach-agent $r([expr $idx%2]) $fsink($idx)
        $ns connect $ftcp($idx) $fsink($idx)
        $ftcp($idx) set fid_ 0
        $ftcp($idx) proc done {} "close-conn $idx $intv $size"
        $ftcp($idx) advanceby $size
}
proc close-conn { idx intv size } {
        global ns
        set now [$ns now]
        $ns at [expr $now + $intv] "start-conn 0 $idx $intv $size"
        puts "at $now + $intv start next"
}
set forel_intv 1
set fores_intv 0.05
set ssize 4
set lsize 1000
build-fore-tcp 1 $ssize 1 0.1
build-fore-tcp 0 $lsize $forel_intv 0.5
for {set i 0} {$i < 5} { incr i} {
        build-fore-tcp [expr 2*$i+3] $ssize $fores_intv [expr 1.2+$i*0.1]
}
```

Node 0 is the "size-aware classifier", which counts the incoming packets from each flow. Once the count exceeds a certain threshold (in this case 5), the remaining packets from the corresponding flow are identified as long flow packets. In the simulation, long flow packets are

colored in blue. All the other packets, i.e., packets from flows of size less than 5, and the first 5 packets from a long flow, are all identified as short flow packets, and colored in red in the simulation.

To upload a size-aware classifier with threshold set to 5 to node 0, do the following:

```
# Load a size-aware classifier to node 0

set cls [new Classifier/Hash/SizeAware 128]

$cls set default_ -1

$cls set flowlen_thr_ 5

$cls set refresh_intv_ 2

$cls set dynamic_update_ 0

set n(0) [node_with_classifier $cls]
```

At the bottleneck link (link between node 1 and node 2), a differentiated dropping scheme is employed. This is achieved by:

```
$ns duplex-link $n(1) $n(2) 700Kb 25ms RED/myRIO

$ns duplex-link-op $n(1) $n(2) orient right

Queue/RED/myRIO set gentle_ true

Queue/RED/myRIO set thresh_ 1

Queue/RED/myRIO set maxthresh_ 15

Queue/RED/myRIO set weight_ 10

Queue/RED/myRIO set setbit_ false

set redq [[$ns link $n(1) $n(2)] queue]

$redq set q_weight_ [expr 1.0/2]

$redq set linterm_ [expr 4.0]

$ns queue-limit $n(1) $n(0) 30
```

Notice that to obtain a more drastic effect of size-aware differentiation, we choose a relatively high ratio between long and short flow packet dropping rate (weight_ is set to 10).

Oral Questions

1. what is mean by network simulation tools?
2. What is types of network simulation tools?
3. What is three-way handshake?
4. How to transfer data using TCP connection?