Ex. No: **STUDY OF NETWORK IP ADDRESS AND NETWORK**
Date: **SIMULATOR (NS)**

AIM:

To Study the network IP and the basics of Network simulator (NS2).

IP Address

Classification of IP address
Sub netting
Super netting

Classification of IP Address:

Class	Address Range	Supports
Class A	1.0.0.1 to 126.255.255.254	Supports 16 million hosts on each of 127 networks.
Class B	128.1.0.1 to 191.255.255.254	Supports 65,000 hosts on each of 16,000 networks.
Class C	192.0.1.1 to 223.255.254.254	Supports 254 hosts on each of 2 million networks.
Class D	224.0.0.0 to 239.255.255.255	Reserved for multicast groups.
Class E	240.0.0.0 to 254.255.255.254	Reserved.

Subnetting:

Sub netting is a process of dividing large network into the smaller networks based on layer 3 IP address. Every computer on network has an IP address that represents its location on network. Two version of IP addresses are available IPv4 and IPv6.

Advantages of Subnetting:

- Subnetting reduces network traffic by removing collision and broadcast traffic, that overall improve performance.
- Subnetting allows you to apply network security polices at the interconnection between subnets.
- Subnetting allows you to save money by reducing requirement for IP range.

Supernetting:

A super network, or **super net**, is an Internet Protocol (IP) network that is formed from the combination of two or more networks (or subnets) with a common Classless Inter-

Domain Routing (CIDR) prefix. The new routing prefix for the combined network aggregates the prefixes of the constituent networks.

NETWORK SIMULATOR (NS2)

NS (Version 2) is an open source network simulation tool. It is an object oriented, discrete event driven simulator written in C++ and Otcl. The primary use of NS is in network researches to simulate various types of wired/wireless local and wide area networks; to implement network protocols such as TCP and UPD, traffic source behavior such as FTP, Telnet, Web, CBR and VBR, router queue management mechanism such as Drop Tail, RED and CBQ, routing algorithms such as Dijkstra, and many more.

Ns2 is written in C++ and Otcl to separate the control and data path implementations. The simulator supports a class hierarchy in C++ (the compiled hierarchy) and a corresponding hierarchy within the Otcl interpreter (interpreted hierarchy).

The reason why ns2 uses two languages is that different tasks have different requirements: For example simulation of protocols requires efficient manipulation of bytes and packet headers making the run-time speed very important. On the other hand, in network studies where the aim is to vary some parameters and to quickly examine a number of scenarios the time to change the model and run it again is more important.

In ns2, C++ is used for detailed protocol implementation and in general for such cases where every packet of a flow has to be processed. For instance, if you want to implement a new queuing discipline, then C++ is the language of choice. Otcl, on the other hand, is suitable for configuration and setup. Otcl runs quite slowly, but it can be changed very quickly making the construction of simulations easier. In ns2, the compiled C++ objects can be made available to the Otcl interpreter. In this way, the ready-made C++ objects can be controlled from the OTcl level.

Assigning values to variables

In tcl, values can be stored to variables and these values can be further used in commands:

set a 5 set b
[expr $a/5]

In the first line, the variable a is assigned the value “5”. In the second line, the result of the command [expr $a/5], which equals 1, is then used as an argument to another command, which in turn assigns a value to the variable b. The “$” sign is used to obtain a value contained in a variable and square brackets are an indication of a command substitution.

Procedures

One can define new procedures with the proc command. The first argument to proc is the name of the procedure and the second argument contains the list of the argument names to that procedure. For instance a procedure that calculates the sum of two numbers can be defined as follows:

```
proc sum {a b} {
        expr $a + $b
}
```

The next procedure calculates the factorial of a number:

```
proc factorial a {
        if {$a <= 1} {
            return 1
        }
    #here the procedure is called again
    expr $x * [factorial [expr $x-1]]
}
```

It is also possible to give an empty string as an argument list. However, in this case the variables that are used by the procedure have to be defined as global. For instance:

```
proc sum {} { global
    a b
    expr $a + $b
}
```

Files and lists

In tcl, a file can be opened for reading with the command: **set**

```
testfile [open test.dat r]
```

The first line of the file can be stored to a list with a command:

```
gets $testfile list
```

Now it is possible to obtain the elements of the list with commands (numbering of elements starts from 0) :

set first [lindex $list 0] set
second [lindex $list 1]

Similarly, a file can be written with a puts command:

set testfile [open test.dat w] puts $testfile "testi"

Calling subprocesses

The command exec creates a subprocess and waits for it to complete. The use of exec is similar to giving a command line to a shell program. For instance, to remove a file:

exec rm $testfile

The exec command is particularly useful when one wants to call a tcl-script from within another tclscript. For instance, in order to run the tcl-script example.tcl multiple times with the value of the parameter "test" ranging from 1 to 10, one can type the following lines to another tcl-script:

```
for {set ind 1} {$ind <= 10} {incr ind} {
    set test $ind exec ns example.tcl
    test
}
```

Creating the topology

To be able to run a simulation scenario, a network topology must first be created. In ns2, the topology consists of a collection of nodes and links.

set ns [new Simulator]

The simulator object has member functions that enable creating the nodes and the links, connecting agents etc. All these basic functions can be found from the class Simulator. When using functions belonging to this class, the command begins with "$ns", since ns was defined to be a handle to the Simulator object.

Nodes

New node objects can be created with the command:

set n0 [$ns node] set
n1 [$ns node] set n2
[$ns node] set n3
[$ns node]

The member function of the Simulator class, called "node" creates four nodes and assigns them to the handles n0, n1, n2 and n3. These handles can later be used when referring to the nodes. If the node is not a router but an end system, traffic agents (TCP, UDP etc.) and traffic sources (FTP,CBR etc.) must be set up, i.e, sources need to be attached to the agents and the agents to the nodes, respectively.

Agents, applications and traffic sources

The most common agents used in ns2 are UDP and TCP agents. In case of a TCP agent, several types are available. The most common agent types are:

- Agent/TCP – a Tahoe TCP sender
- Agent/TCP/Reno – a Reno TCP sender
- Agent/TCP/Sack1 – TCP with selective acknowledgement

The most common applications and traffic sources provided by ns2 are:

- Application/FTP – produces bulk data that TCP will send
- Application/Traffic/CBR – generates packets with a constant bit rate
- Application/Traffic/Exponential – during off-periods, no traffic is sent. During onperiods, packets are generated with a constant rate. The length of both on and offperiods is exponentially distributed.
- Application/Traffic/Trace – Traffic is generated from a trace file, where the sizes and interarrival times of the packets are defined.

In addition to these ready-made applications, it is possible to generate traffic by using the methods provided by the class Agent. For example, if one wants to send data over UDP, the method.

send(int nbytes)

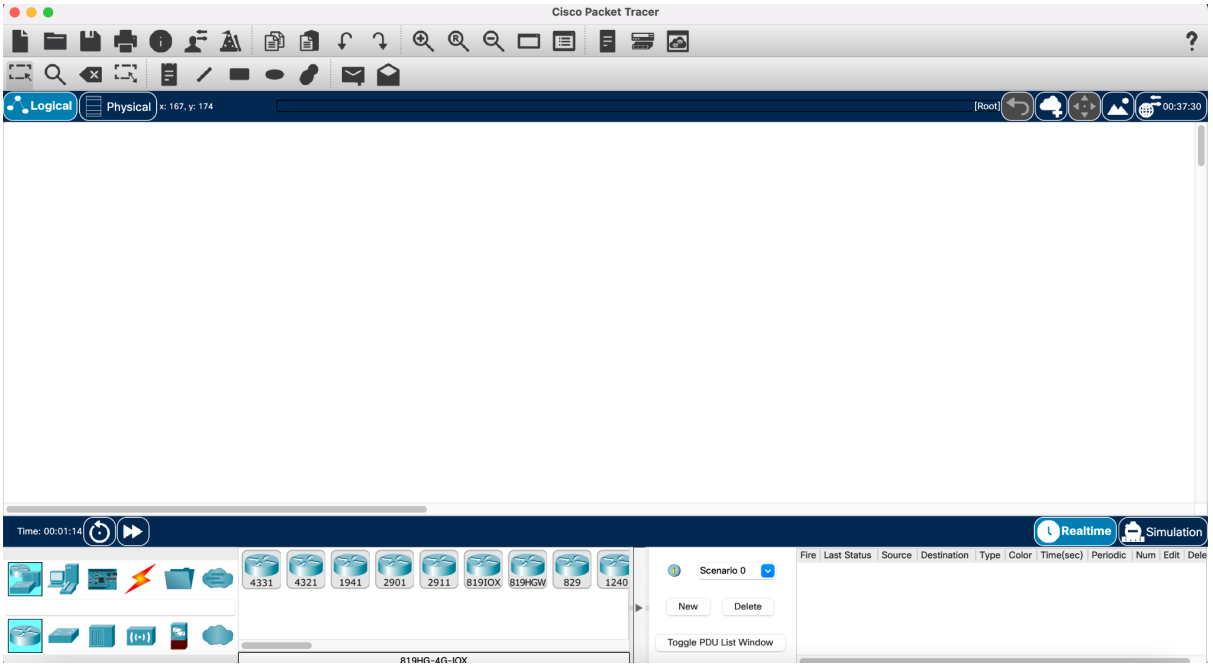
Below is a complete example of how to create a CBR traffic source using UDP as transport protocol and attach it to node n0:

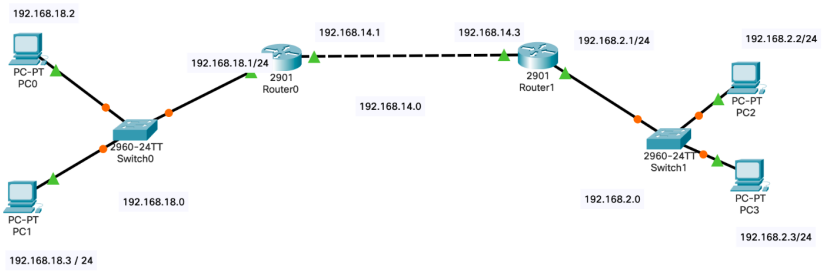
```
set udp0 [new Agent/UDP] $ns
attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set packet_size_ 1000
$udp0 set packet_size_ 1000
$cbr0 set rate_ 1000000
```

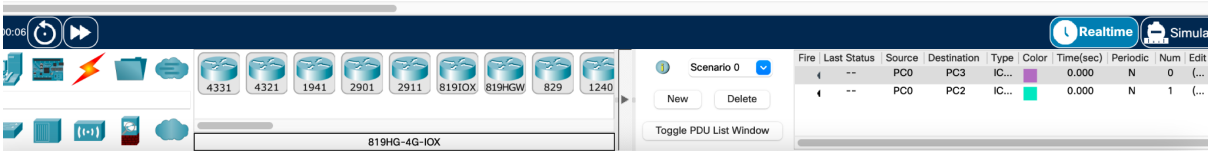
An FTP application using TCP as a transport protocol can be created and attached to node n1 in much the same way:

```
set tcp1 [new Agent/TCP] $ns
attach-agent $n1 $tcp1 set
ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$tcp1 set packet_size_ 1000
```

The UDP and TCP classes are both child-classes of the class Agent. With the expressions [new Agent/TCP] and [new Agent/UDP] the properties of these classes can be combined to the new objects udp0 and tcp1. These objects are then attached to nodes n0 and n1. Next, the application is defined and attached to the transport protocol. Finally, the configuration parameters of the traffic source are set.


Cisco Packet Tracer
Logical
Physical
Time: 00:01:14
Realtime
Simulation
4331
4321
1941
2901
2911
819IOX
819HGW
829
1240
Scenario 0
New
Delete
Toggle PDU List Window
Fire
Last Status
Source
Destination
Type
Color
Time(sec)
Periodic
Num
Edit


192.168.18.2
PC-PT
PC0
192.168.18.1/24
2901
Router0
192.168.14.1
192.168.14.3
192.168.14.0
2901
Router1
192.168.2.1/24
192.168.2.2/24
PC-PT
PC2
2960-24TT
Switch0
2960-24TT
Switch1
PC-PT
PC1
192.168.18.0
192.168.2.0
PC-PT
PC3
192.168.18.3 / 24
192.168.2.3/24


Realtime
4331
4321
1941
2901
2911
819IOX
819HGW
829
1240
Scenario 0
New
Delete
Toggle PDU List Window
819HG-4G-IOX
Fire
Last Status
Source
Destination
Type
Color
Time(sec)
Periodic
Num
Edit
PC0
PC3
PC0
PC2

Result:

PREPARATION	30	
LAB PERFORMANCE	30	
REPORT	40	
TOTAL	100	
INITIAL OF THE FACULTY		