# Lab Manual (Addendum)
# of

# Data Communications and Networks Lab

**(R18) B.Tech ECE : III Year / I Semester**


Dept. of Electronics and Communications Engineering
Mahatma Gandhi Institute of Technology,
Gandipet, Hyderabad-500 0075

**Prepared by**
Dr. Fayazur Rahaman M
Associate Professor
mfrahaman_ece@mgit.ac.in

**Head of Dept.**
Dr. S.P. Singh
Professor
spsingh@mgit.ac.in

# Contents

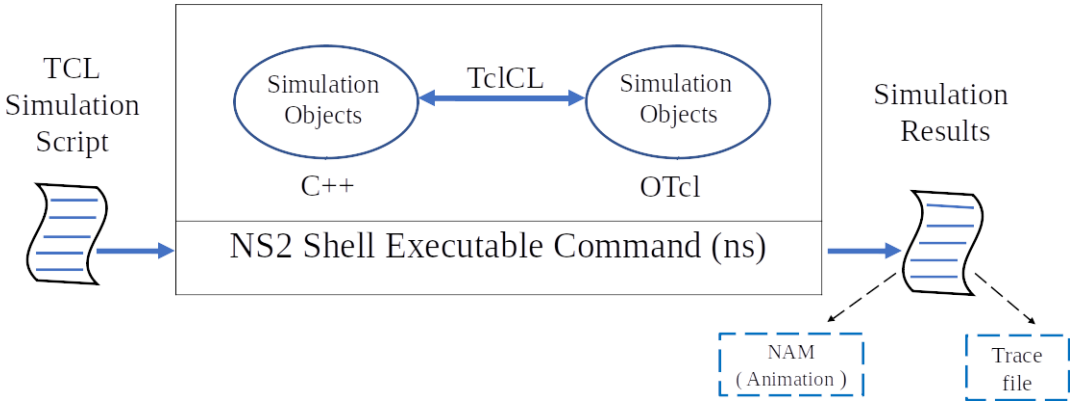
# Part I.

# The Network Simulator (ns-2) Software

## About The Network Simulator (ns-2) Software:

- Website: https://isi.edu/nsnam/ns/
- Help pages: http://http://nsnam.sourceforge.net/wiki/index.php/Main_Page
- FAQ pages: http://nsnam.sourceforge.net/wiki/index.php/Ns_Users_FAQ

### ns-2 Architecture:




### Frequently used terminal commands:

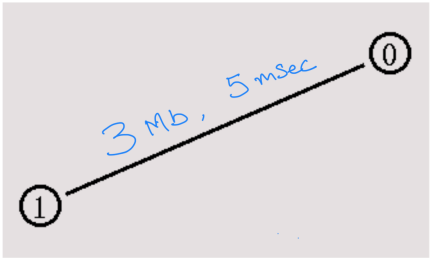
| Command | Description |
|---|---|
| **'cd '** | *Change directory* |
| **'ls'** | *Lists the files in a directory* |
| **'gedit Expxx.tcl &'** | *Create and Open Expxx.tcl script file to write tcl code using gedit (like notepad) editor* |
| **'java -jar NSG2.jar &'** | *Open the 'NS 2 Scenario Generator' application to generate tcl scripts for required network scenarios* |
| **'ns Expxx.tcl'** | *If script is error free, ns interpretor will generate two corresponding output files - NAM file (.nam) and Trace file (.tr)* |
| **'nam Expxx.nam'** | *The network animator utility will graphically display the network scenario created in the .nam file* |
| **'awf -f mgit_wired_05.awk Expxx.tr'** | *Analyze the network scenario data provided in the wired .tr file and compute the performance parameters. This script also creates Expxx-th.dat files for plotting throughput curves.* |
| **'awf -f mgit_wireless_01.awk Expxx.tr'** | *Analyze the network scenario data provided in the wireless .tr file and compute the performance parameters. This script also creates Expxx-th.dat files for plotting throughput curves* |
| **'gnuplot mgit_thputplot.p'** | *Searches the current directory for '*-th.dat' files and plots the throughput curve for each file* |
| **'gnuplot mgit_cwndplot.p'** | *Searches the current directory for '*-cwnd.dat' files and plots the congestion window size curve for each file* |

# 1. Addendum: Writing a TCL Script to create two nodes and links between nodes

## 1.1. TCL Simulation Script (use 'gedit Exp01.tcl')

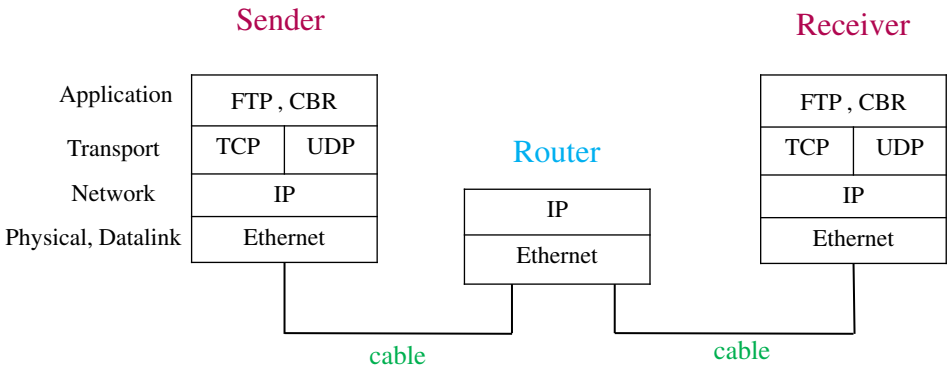
| Code | Description |
|---|---|
| `set ns [new Simulator]` | Create a simulator object |
| `set tracefile [open Exp01.tr w]`<br>`$ns trace-all $tracefile` | Open a file named 'Exp01.tr' for writing trace data |
| `set namfile [open Exp01.nam w]`<br>`$ns namtrace-all $namfile` | Open a file named 'Exp01.nam' for writing nam trace data |
| `set n0 [$ns node]`<br>`set n1 [$ns node]` | Create two nodes and assign them to the handles 'n0' and 'n1' |
| `$ns duplex-link $n0 $n1 3Mb 5ms DropTail` | Connect the nodes 'n0' and 'n1' with a duplex link with 3 Megabit, a delay of 5 milli seconds and DropTail queue |
| `proc finish {} {`<br>`global ns tracefile namfile`<br>`$ns flush-trace`<br>`close $tracefile`<br>`close $namfile`<br>`exit 0`<br>`}` | 'finish' procedure to close the trace file |
| `$ns at 5.0 'finish'` | Execute the 'finish' procedure after 5.0 secs of simulation time |
| `$ns run` | Start the simulation |

## 1.2. Expected network animation output (using 'nam')

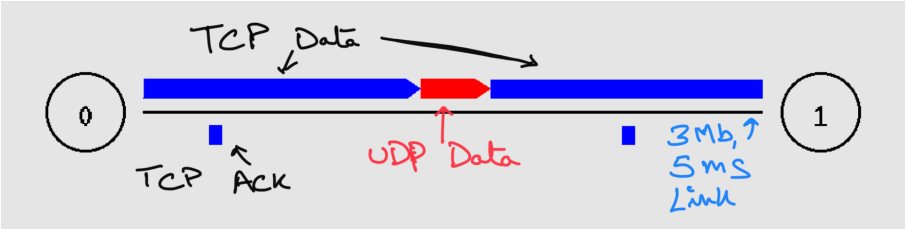


# 2. Addendum: Writing a TCL Script to transmit data between nodes

## 2.1. Network in ns-2:




## 2.2. Expected network animation output (using 'nam')



## 2.3. TCL Simulation Script (use 'gedit Exp02.tcl')

| Code | Description |
|---|---|
| `set ns [new Simulator]` | Create a simulator object |
| `set tracefile [open Exp02.tr w]`<br>`$ns trace-all $tracefile` | Open a file named 'Exp01.tr' for writing trace data |
| `set namfile [open Exp02.nam w]`<br>`$ns namtrace-all $namfile` | Open a file named 'Exp01.nam' for writing nam trace data |
| `set n0 [$ns node]`<br>`set n1 [$ns node]` | Create two nodes and assign them to the handles 'n0' and 'n1' |
| `$ns duplex-link $n0 $n1 3Mb 5ms DropTail` | Connect the nodes 'n0' and 'n1' with a duplex link with 3 Megabit, a delay of 5 milli seconds and DropTail queue |
| `set udp0 [new Agent/UDP]`<br>`$ns attach-agent $n0 $udp0` | Create a UDP agent and attach it to node n0 |
| `set cbr0 [new Application/Traffic/CBR]`<br>`$cbr0 attach-agent $udp` | Create a CBR traffic source and attach it to udp0 |
| `set null0 [new Agent/Null]`<br><br>`$ns attach-agent $n1 $null0`<br>`$ns connect $udp0 $null0` | Create a Null agent which acts as traffic sink and attach it to node n1<br><br>Connect the two agents to each other |
| `set tcp0 [new Agent/TCP]`<br>`$ns attach-agent $n0 $tcp0` | Create a TCP agent and attach it to node n0 |
| `set ftp0 [new Application/FTP]`<br>`$ftp0 attach-agent $tcp0` | Create a FTP traffic source and attach it to tcp0 |
| `set sink0 [new Agent/TCPSink]`<br>`$ns attach-agent $n1 $sink0` | Create TCP sink and attach it to node n1 |
| `$ns connect $tcp0 $sink0` | Connect the two agents to each other |
| `$ns at 1.0 "$cbr0 start"`<br>`$ns at 8.5 "$cbr0 stop"`<br>`$ns at 2.5 "$ftp0 start"`<br>`$ns at 8.0 "$ftp0 stop"` | Specify the start and stop data transfer time for each source |
| `$ns at 10.0 "finish"` | |
| `proc finish {} {`<br>`  global ns tracefile namfile`<br>`  $ns flush-trace`<br>`  close $tracefile`<br>`  close $namfile`<br>`  exit 0`<br>`}` | 'finish' procedure to close the trace file |
| `$ns run` | Start the simulation |

## 2.4. Format of the trace file generated:

| Event | Time | From Node | To Node | Pkt Type | Pkt Size | Flags | Flow id | Src Addr | Dst Addr | Seq Num | Pkt Id |
|---|---|---|---|---|---|---|---|---|---|---|---|

### 2.4.1. Event field

| | |
|---|---|
| **+** | An enqueue operation occurred on the device queue |
| **-** | A dequeue operation occurred on the device queue |
| **d** | A packet was dropped, typically because the queue was full |
| **r** | A packet was received by the net device |

### 2.4.2. Sample Trace file generted (.tr)

```
+ 8.48875 0 1 cbr 210 ------- 1 0.0 1.0 1997 5401
- 8.48875 0 1 cbr 210 ------- 1 0.0 1.0 1997 5401
r 8.49056 0 1 cbr 210 ------- 1 0.0 1.0 1996 5400
+ 8.4925 0 1 cbr 210 ------- 1 0.0 1.0 1998 5402
- 8.4925 0 1 cbr 210 ------- 1 0.0 1.0 1998 5402
r 8.49431 0 1 cbr 210 ------- 1 0.0 1.0 1997 5401
+ 8.49625 0 1 cbr 210 ------- 1 0.0 1.0 1999 5403
- 8.49625 0 1 cbr 210 ------- 1 0.0 1.0 1999 5403
r 8.49806 0 1 cbr 210 ------- 1 0.0 1.0 1998 5402
r 8.50181 0 1 cbr 210 ------- 1 0.0 1.0 1999 5403
```

## 2.5. Performance parameters obtained:

Compute and note down the performance parameters

| Parameter | TCP values | UDP values |
|---|---|---|
| **Src_addr → Dst_addr** | | |
| **Pkt Type** | | |
| **Pkt Size** | | |
| **Transfer start time** | | |
| **Transfer end time** | | |
| **# Sent Packets** | | |
| **# Dropped Packets (Congestion)** | | |
| **# Dropped Packets (Link Errors)** | | |
| **# Discarded Packets (Broadcasts)** | | |
| **# Retransmitted Packets** | | |
| **# Duplicate Packets** | | |
| **# Delivered Packets** | | |
| **Pkt Delivery Ratio** | | |
| **Pkt Drop Ratio** | | |
| **Average Delay** | | |
| **Throughput** | | |

Where the following parameters are computed using the formulaes:

$$\textbf{Packet Delivery Ratio} = \frac{\#\text{ of packets received}}{\#\text{ of Sent packets}} \times 100 \quad (\%)$$

$$\textbf{Packet Drop Ratio} = \frac{\#\text{ of dropped packets}}{\#\text{ of Sent packets}} \times 100 \quad (\%)$$

$$\textbf{Average Delay} = \frac{\sum \text{Transfer delay of each Delivered pkt}}{\#\text{ of Delivered pkts}} \quad (\text{seconds})$$

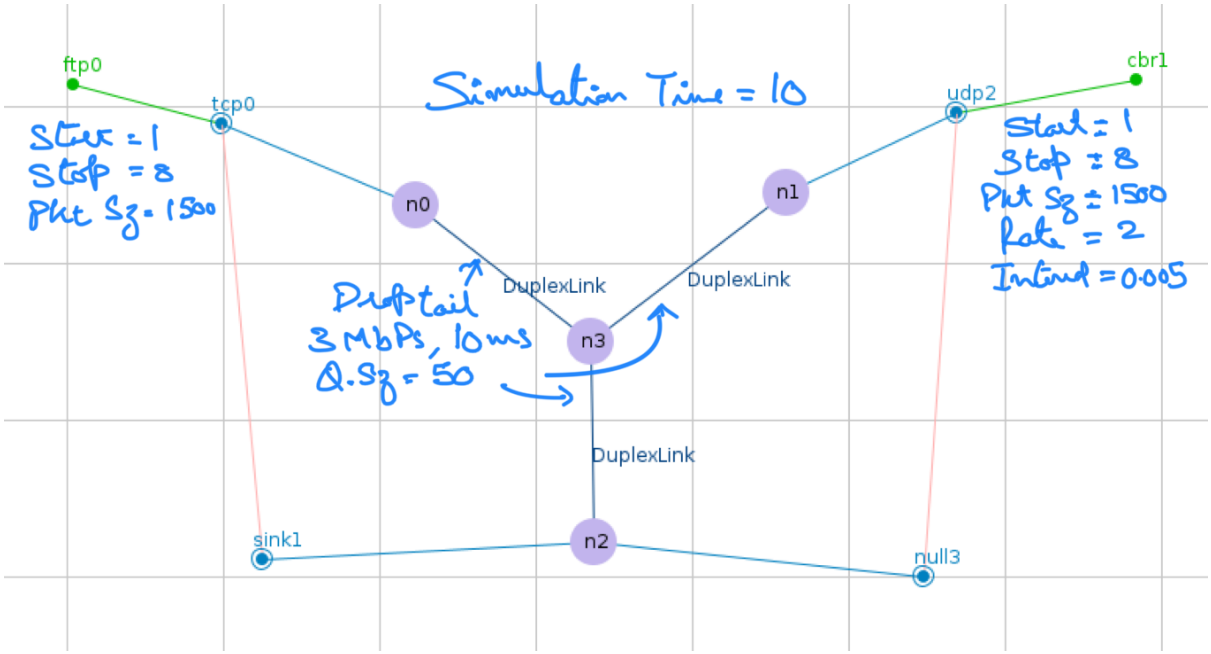
$$\textbf{Throughput} = \frac{\#\text{ Delivered Pkts} \times \text{Pkt Size} \times 8}{\text{Total transfer time}} \quad (\text{bps})$$

# 3. Addendum: Evaluate the performance of various LAN Topologies

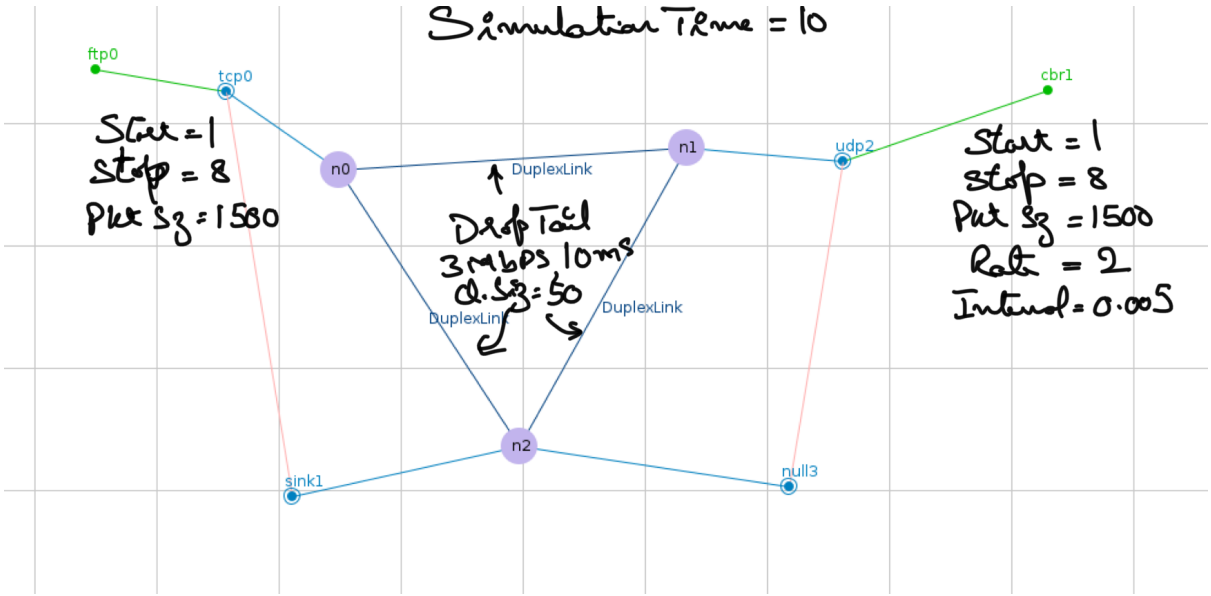
## 3.1. Network Scenarios and configurations:

Construct the LAN topologies with the following configurations

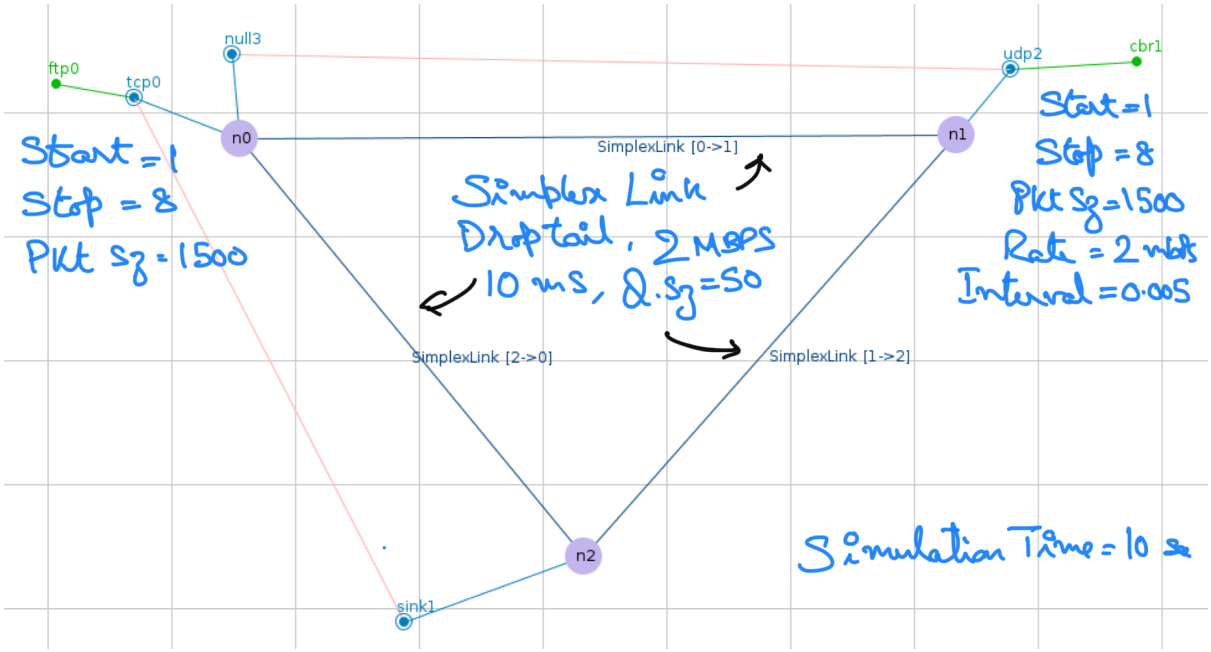
### 3.1.1. Star Topology:




### 3.1.2. Mesh Topology:



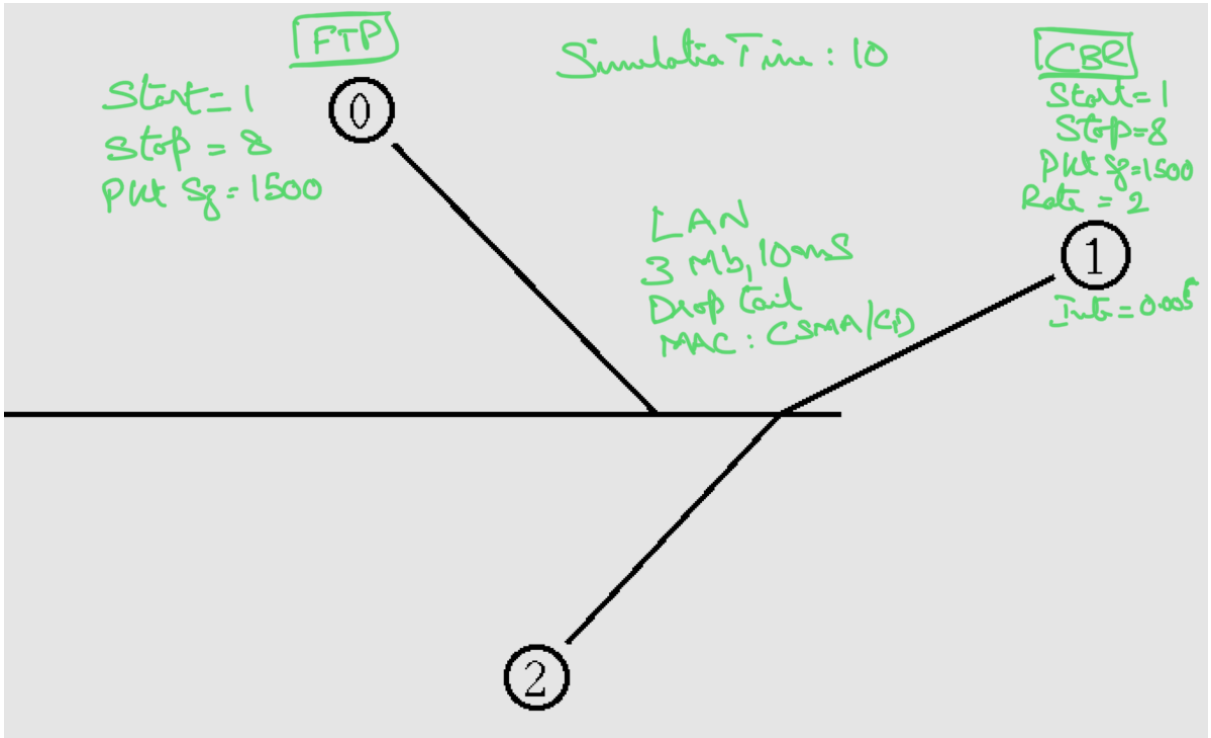
### 3.1.3. Ring Topology:




### 3.1.4. Bus Topology:

The below code shows how to setup a LAN (bus topology) with the three nodes.

```
set lan [$ns newLan "$n0 $n1 $n2" 3Mb 10ms LL Queue/DropTail MAC/Csma/Cd Channel]
```



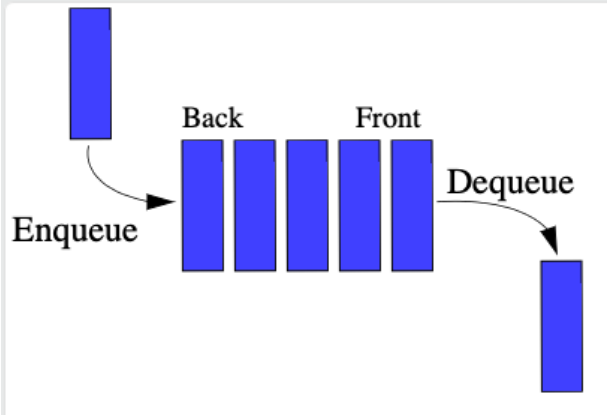

## 3.2. Obtained Performance parameters:

Construct the network scenarios for the above mentioned topologies and compute their performance

| **Parameter** | **Star** | | **Mesh** | | **Ring** | | **Bus** | |
|---|---|---|---|---|---|---|---|---|
| | **TCP** | **UDP** | **TCP** | **UDP** | **TCP** | **UDP** | **TCP** | **UDP** |
| **Src_addr → Dst_addr** | | | | | | | | |
| **Pkt Type** | | | | | | | | |
| **Pkt Size** | | | | | | | | |
| **Transfer start time** | | | | | | | | |
| **Transfer end time** | | | | | | | | |
| **# Sent Pkts** | | | | | | | | |
| **# Dropped Pkts (Q Full)** | | | | | | | | |
| **# Dropped Pkts (Errors)** | | | | | | | | |
| **# Discard Pkts (BrdCst)** | | | | | | | | |
| **# Retransmitted Pkts** | | | | | | | | |
| **# Duplicate Pkts** | | | | | | | | |
| **# Delivered Pkts** | | | | | | | | |
| **Pkt Delivery Ratio** | | | | | | | | |
| **Pkt Drop Ratio** | | | | | | | | |
| **Average Delay** | | | | | | | | |
| **Throughput** | | | | | | | | |

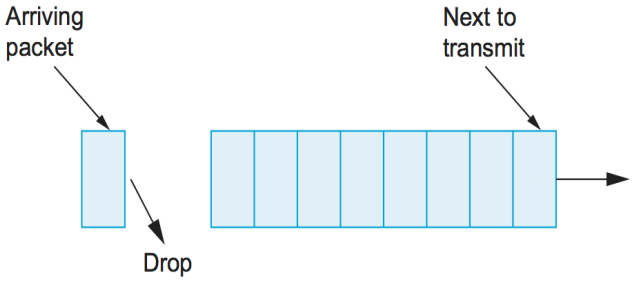
# 4. Addendum: Evaluate the performance of Drop Tail and RED queue management schemes

## 4.1. Definitions:



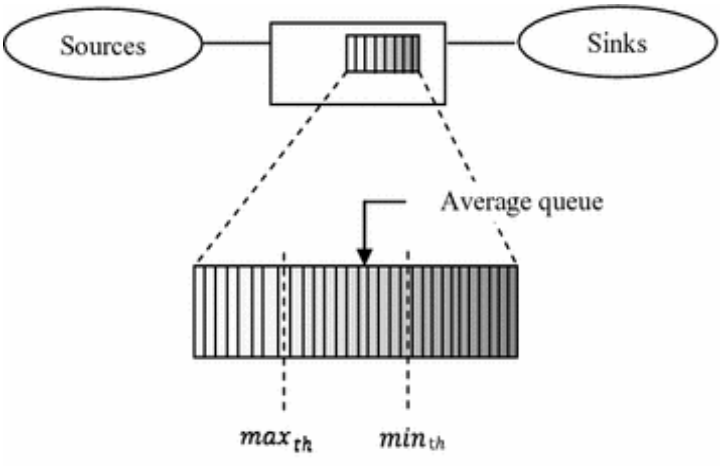

### 4.1.1. DropTail queue management schemes

Tail drop is a simple queue management algorithm used by network schedulers in network equipment to decide when to drop packets. With tail drop, when the queue is filled to its maximum capacity, the newly arriving packets are dropped until the queue has enough room to accept incoming traffic.




### 4.1.2. RED (Random Early Detection) queue management schemes

Random early detection (RED), also known as random early discard or random early drop is a queuing discipline for a network scheduler suited for congestion avoidance

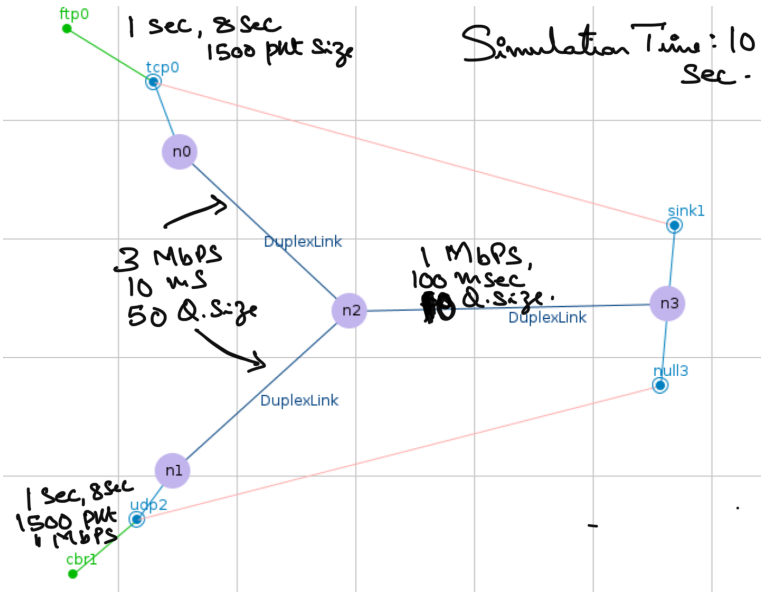



## 4.2. Network Scenarios and configurations:

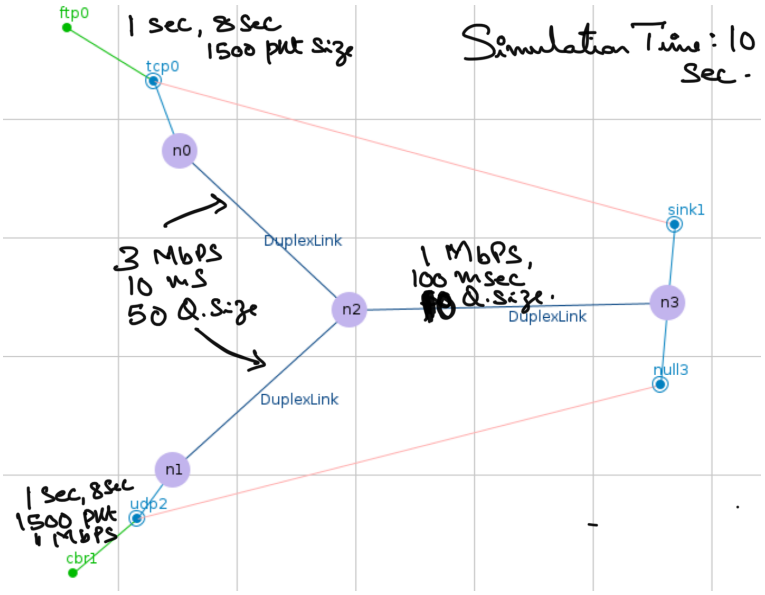
Construct the networks with the following configurations. And use the following lines of code to change the color of the packets

| Code | Comment |
|---|---|
| `$ns color 1 Red`<br>`$ns color 2 Blue`<br>`$ns color 3 Green` | **Initialization:** Assign an identifier for each color |
| `$ns duplex-link-op $n2 $n3 queuePos 0.5` | **Link Definition:** Instruction for NAM to monitor the queue size |
| `$tcp0 set fid_ 1`<br><br>`$udp1 set fid_ 2` | **Agent Definition:** Assign a different identifier for each agent |

### 4.2.1. DropTail:



#### 4.2.2. RED:



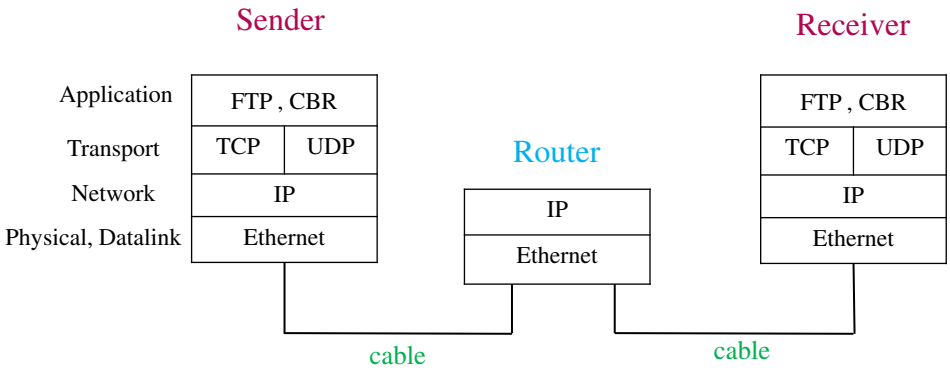

### 4.3. Obtained Performance parameters:

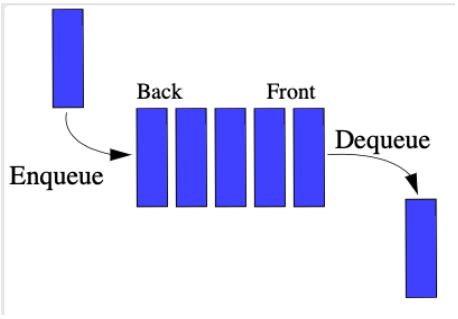
Construct the network scenarios for the above mentioned topologies and compute their performance

| Parameter | DropTail | | RED | |
|---|---|---|---|---|
| | TCP | UDP | TCP | UDP |
| Src_addr → Dst_addr | | | | |
| Pkt Type | | | | |
| Pkt Size | | | | |
| Transfer start time | | | | |
| Transfer end time | | | | |
| # Sent Pkts | | | | |
| # Dropped Pkts (Q Full) | | | | |
| # Dropped Pkts (Errors) | | | | |
| # Discard Pkts (BrdCst) | | | | |
| # Retransmitted Pkts | | | | |
| # Duplicate Pkts | | | | |
| # Delivered Pkts | | | | |
| Pkt Delivery Ratio | | | | |
| Pkt Drop Ratio | | | | |
| Average Delay | | | | |
| Throughput | | | | |

# 5. Addendum: Evaluate the performance of FQ and CBQ Scheduling Mechanisms

## 5.1. Definitions:

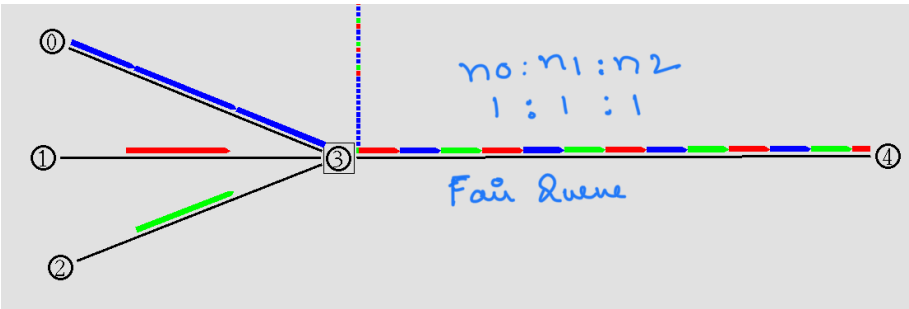


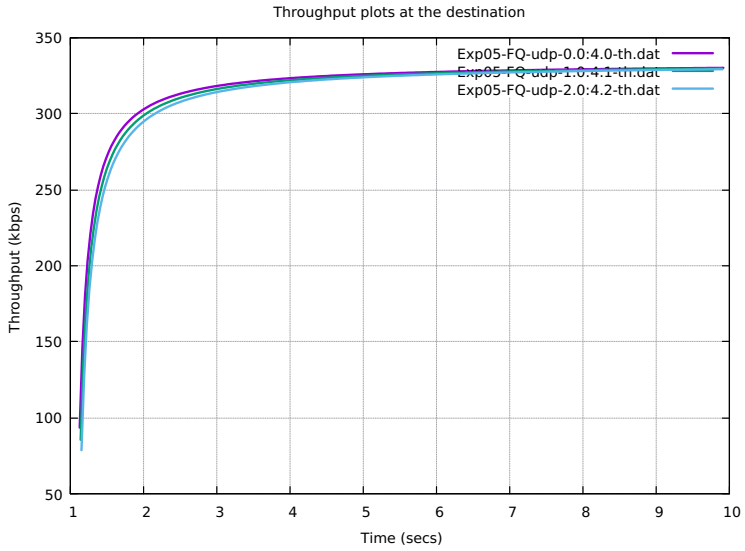




## 5.2. Fair Queue (FQ) Scheduling Mechanism

Fair queuing algorithm is designed to achieve fairness when a limited resource is shared. Fair queuing uses one queue per packet flow and services them in rotation, such that each flow can "obtain an equal fraction of the resources".

### Expected Results:

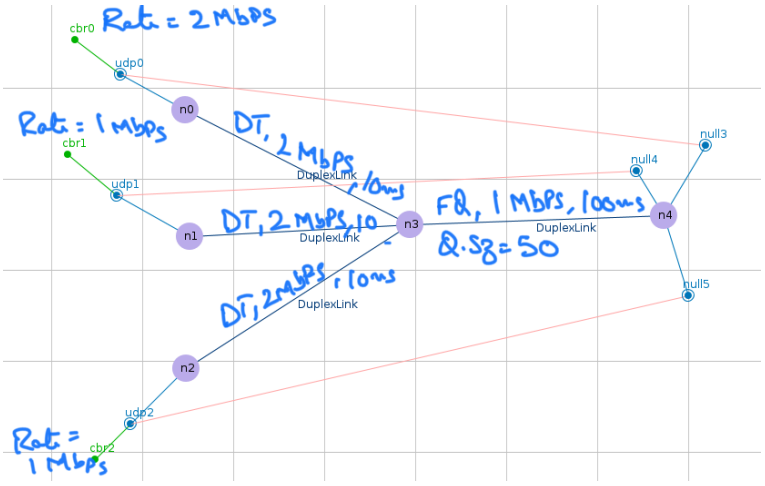





### 5.2.1. Network Scenarios and configurations:

Construct the networks with the following configurations. Use the following lines of code to change the color of the packets, monitor the queue, etc

| Code | Comment |
|---|---|
| `$ns color 1 Red`<br>`$ns color 2 Blue`<br>`$ns color 3 Green` | **Initialization:** Assign an identifier for each color |
| `$ns duplex-link-op $n2 $n3 queuePos 0.5` | **Link Definition:** Instruction for NAM to monitor the queue size |
| `$udp0 set fid_ 1`<br><br>`$udp1 set fid_ 2`<br>`$udp2 set fid_ 3` | **Agent Definition:** Assign a different identifier for each agent |




### 5.2.2. Obtained Performance parameters for FQ:

Construct the network scenarios for the above mentioned FQ topology and compute the performance

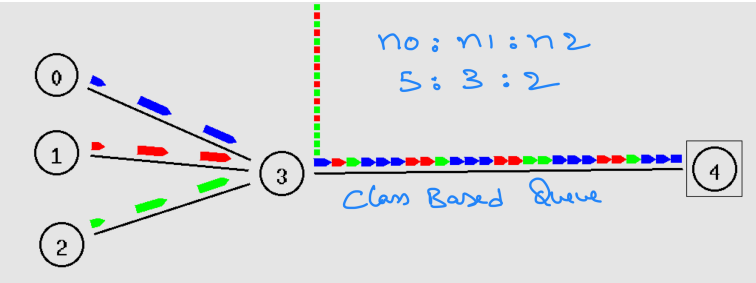
| Parameter | Fair Queue | | | Class-Based Queue | | |
|---|---|---|---|---|---|---|
| | Node 0 | Node 1 | Node 2 | Node 0 | Node 1 | Node 2 |
| Src_addr → Dst_addr | | | | | | |
| Pkt Type | | | | | | |
| Pkt Size | | | | | | |
| Transfer start time | | | | | | |
| Transfer end time | | | | | | |
| # Sent Pkts | | | | | | |
| # Dropped Pkts (Q Full) | | | | | | |
| # Dropped Pkts (Errors) | | | | | | |
| # Discard Pkts (BrdCst) | | | | | | |
| # Retransmitted Pkts | | | | | | |
| # Duplicate Pkts | | | | | | |
| # Delivered Pkts | | | | | | |
| Pkt Delivery Ratio | | | | | | |
| Pkt Drop Ratio | | | | | | |
| Average Delay | | | | | | |
| Throughput | | | | | | |

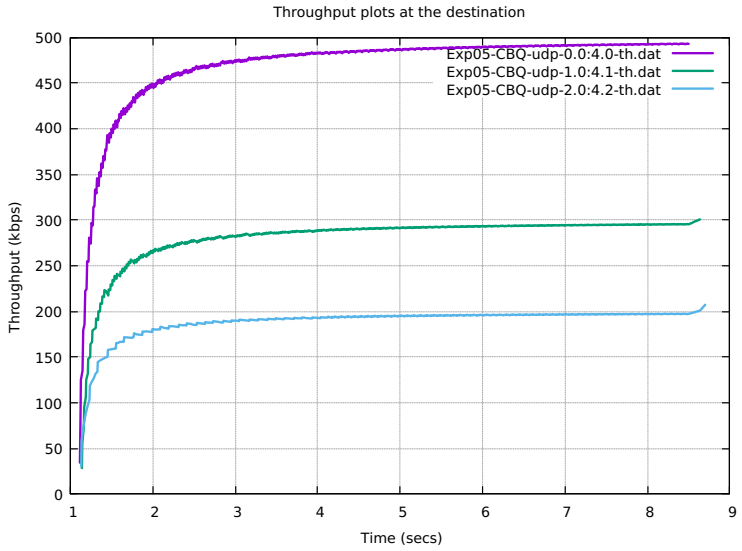
## 5.3. Class-Based Queue (CBQ) Scheduling mechanism

### 5.3.1. Class-based queuing (CBQ)

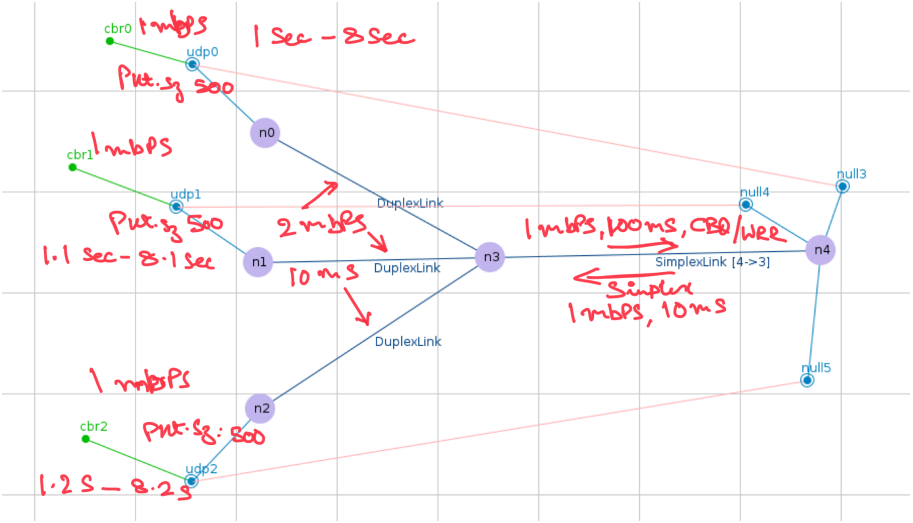
CBQ is a queuing discipline for the network scheduler that allows traffic to share bandwidth equally, after being grouped by classes. The classes can be based upon a variety of parameters, such as priority, interface, or originating program.

Expected Results:






### 5.3.2. Network Scenario




### 5.3.3. TCL script for CBQ Scheduling mechanism

Construct the networks with the following configurations. Use the following lines of code to define the CBQ mechanism.

| Code | Comment |
|---|---|
| `$ns simplex-link $n3 $n4 1Mb 100ms CBQ/WRR`<br>`$ns simplex-link $n4 $n3 1Mb 10ms DropTail`<br><br>`set cbqlink [$ns link $n3 $n4]`<br>`set topclass [new CBQClass]`<br>`$topclass setparams none 0 1 auto 8 2 0`<br><br>`set class1 [new CBQClass]`<br>`set queue1 [new Queue/DropTail]`<br>`$class1 install-queue $queue1`<br>`$class1 setparams $topclass true 0.5 auto 1 1 0`<br><br>`set class2 [new CBQClass]`<br>`set queue2 [new Queue/DropTail]`<br>`$class2 install-queue $queue2`<br>`$class2 setparams $topclass true 0.3 auto 1 1 0`<br><br>`set class3 [new CBQClass]`<br>`set queue3 [new Queue/DropTail]`<br>`$class3 install-queue $queue3`<br>`$class3 setparams $topclass true 0.2 auto 1 1 0`<br><br>`$cbqlink insert $topclass`<br>`$cbqlink insert $class1`<br>`$cbqlink insert $class2`<br>`$cbqlink insert $class3`<br><br>`$cbqlink bind $class1 1`<br>`$cbqlink bind $class2 2`<br>`$cbqlink bind $class3 3` | **Link Definition:** Define a simplex-link with CBQ between the nodes. Further define the CBQ classes with 50%, 30% and 20% data rate reservation in the link. |

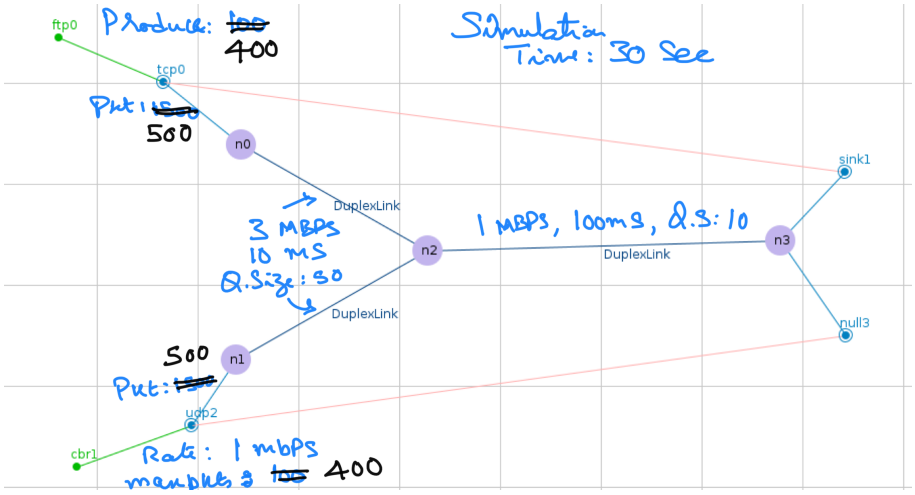
† Replace the 'X' in tcpX, udpX, cbrX, ftpX etc, with the appropriate digit from your TCL script. Further use the appropriate digit for node variables i.e., $nX.

### 5.3.4. Obtained Performance parameters for CBQ:

Construct the network scenarios for the above mentioned CBQ topology and compute the performance

# 6. Addendum: Evaluate the performance of TCP and UDP protocols

## 6.1. Performance under Network congestion:



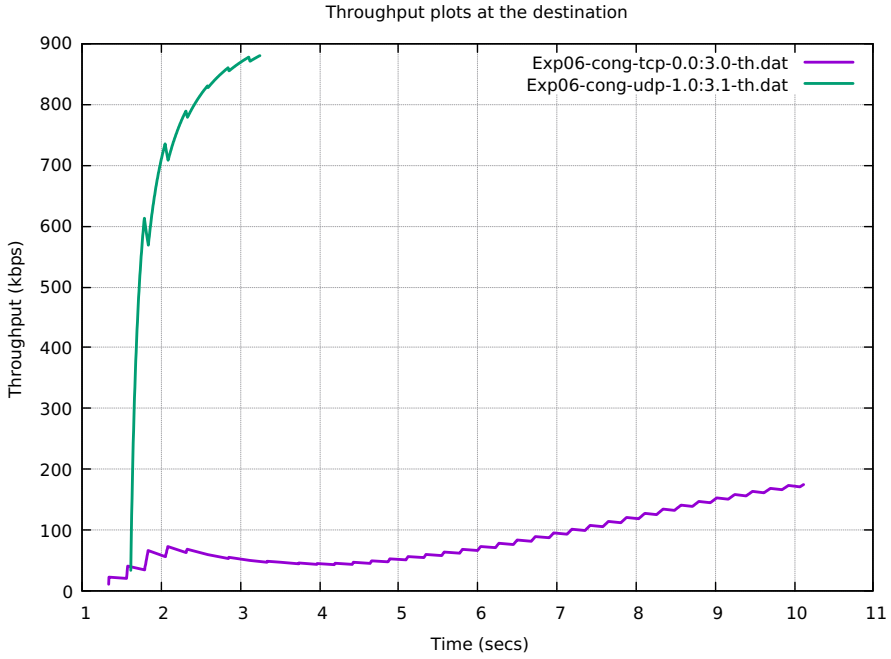

### 6.1.1. Network Scenarios and configurations:

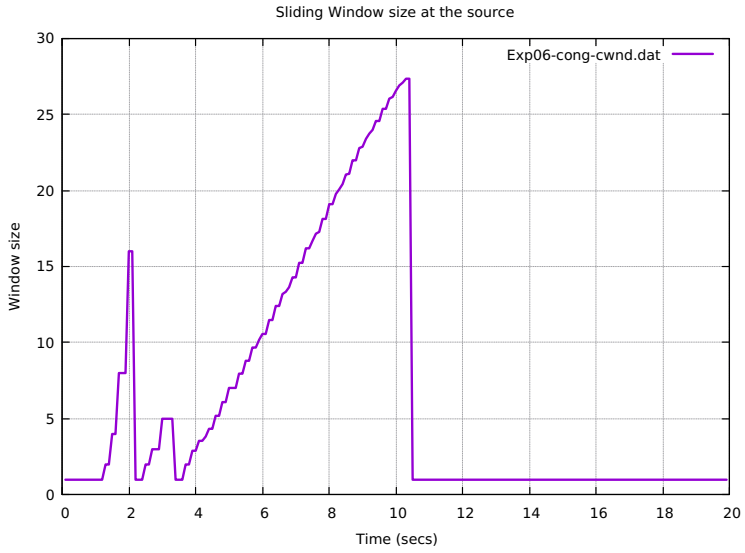
Construct the networks with the following configurations. Use the following lines of code to change the color of the packets, monitor the queue, etc

| Code | Comment |
|---|---|
| `$ns color 1 Red`<br>`$ns color 2 Blue`<br><br>`set cwndfile [open Exp06-cong-cwnd.dat w]` | **Initialization:** Assign an identifier for each color<br><br>File to store the TCP window size at different time stamps |
| `$ns duplex-link-op $n2 $n3 queuePos 0.5` | **Links Definition:** Instruction for NAM to monitor the queue size |
| `$tcpX set fid_ 1`<br>`$udpX set fid_ 2`<br><br>`proc recWin { tcp file } {`<br>`global ns`<br>`set time 0.1`<br>`set ctime [$ns now]`<br>`set wnd [$tcp set cwnd_]`<br>`puts $file "$ctime $wnd"`<br>`$ns at [expr $ctime + $time] "recWin $tcp $file"`<br>`}`<br>`$ns at 0.1 "recWin $tcp0 $cwndfile"` | **Agents Definition:** Assign a different identifier for each agent<br><br>Recursive function called to store the congestion TCP window size into a file |
| `$cbrX set maxpkts_ 400`<br>`$ns at 1.0 "$ftpX produce 400"` | **Applications Definition:** Instruction to produce 400 CBR and FTP packets each |
| `global xxxxxxxxxxx cwndfile`<br>`close $cwndfile` | **Termination:** Instructions to close the cwnd data file. Include these in the "finish" function. |

† Replace the 'X' in tcpX, udpX, cbrX, ftpX etc, with the appropriate digit from your TCL script. Further use the appropriate digit for node variables i.e., $nX.

**Expected Results:**

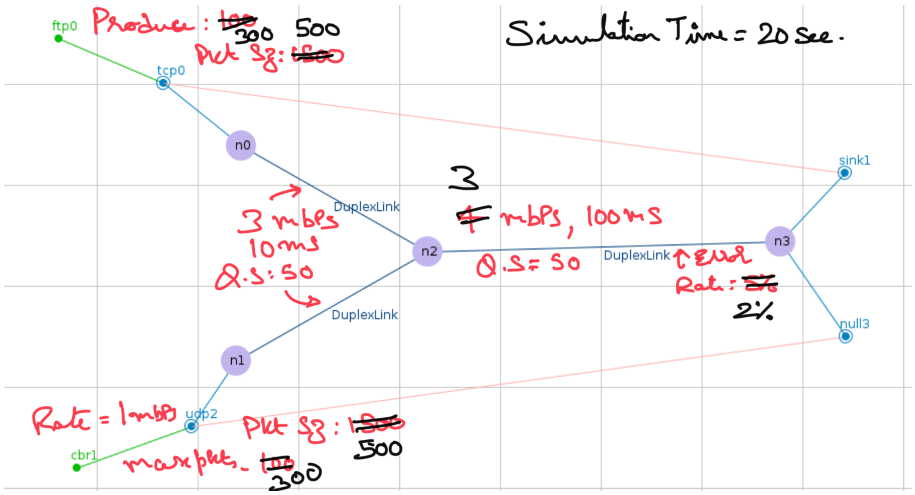


## 6.2. Capture the performance parameters:

| Parameter | Congestion | | Noisy Link | |
|---|---|---|---|---|
| | TCP | UDP | TCP | UDP |
| Src_addr → Dst_addr | | | | |
| Pkt Type | | | | |
| Pkt Size | | | | |
| Transfer start time | | | | |
| Transfer end time | | | | |
| # Sent Pkts | | | | |
| # Dropped Pkts (Q Full) | | | | |
| # Dropped Pkts (Errors) | | | | |
| # Discard Pkts (BrdCst) | | | | |
| # Retransmitted Pkts | | | | |
| # Duplicate Pkts | | | | |
| # Delivered Pkts | | | | |
| Pkt Delivery Ratio | | | | |
| Pkt Drop Ratio | | | | |
| Average Delay | | | | |
| Throughput | | | | |

## 6.3. Performance under Noisy Link (i.e., under link errors):



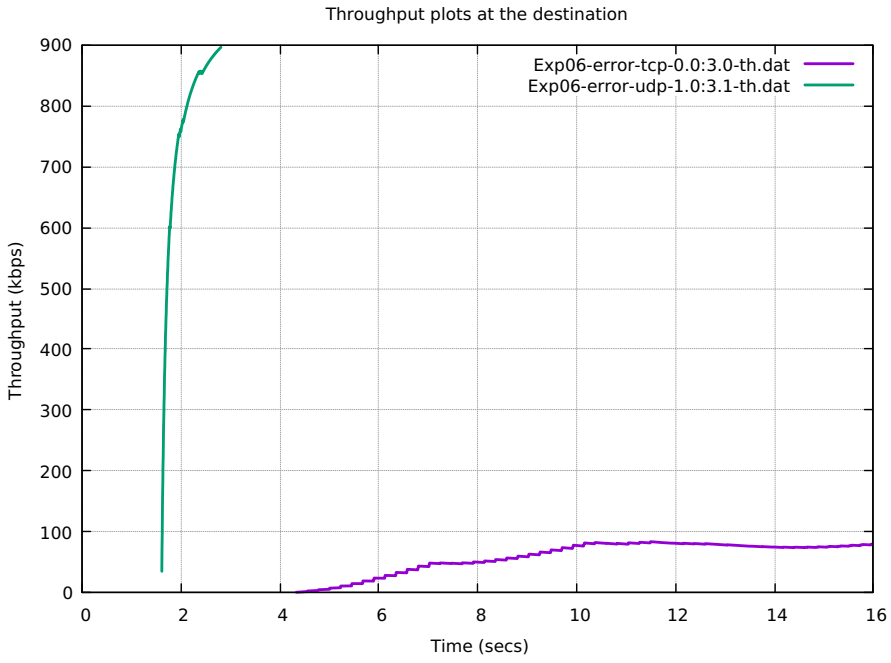

### 6.3.1. Network Scenarios and configurations:

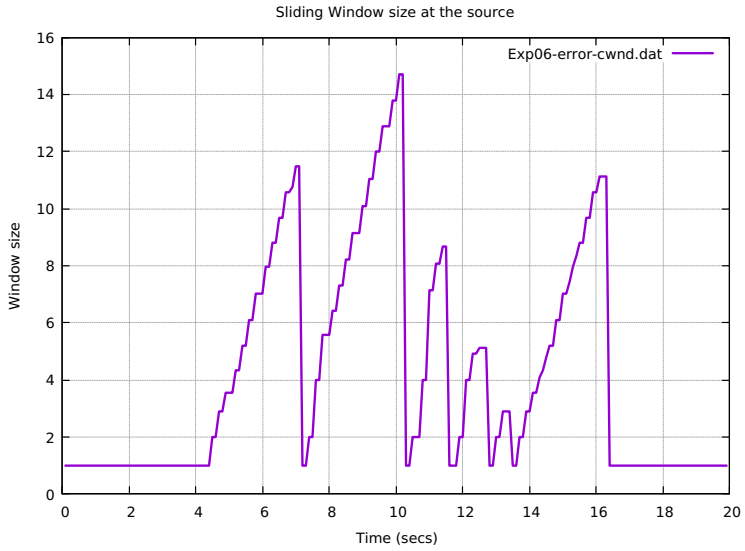
Construct the networks with the following configurations. Use the following lines of code to change the color of the packets, monitor the queue, etc

| Code | Comment |
|---|---|
| `$ns color 1 Red`<br>`$ns color 2 Blue` | **Initialization:** Assign an identifier for each color |
| `set cwndfile [open Exp06-noisy-cwnd.dat w]` | File to store the TCP window size at different time stamps |
| `$ns duplex-link-op $n2 $n3 queuePos 0.5` | **Links Definition:** Instruction for NAM to monitor the queue size |
| `set em [new ErrorModel]`<br>`$em set rate_ 0.02`<br>`$ns link-lossmodel $em $n2 $n3` | Define an Error Model with error rate of 2% |
| `$tcpX set fid_ 1`<br>`$udpX set fid_ 2` | **Agents Definition:** Assign a different identifier for each agent |
| `proc recWin { tcp file } {`<br>`global ns`<br>`set time 0.1`<br>`set ctime [$ns now]`<br>`set wnd [$tcp set cwnd_]`<br>`puts $file "$ctime $wnd"`<br>`$ns at [expr $ctime + $time] "recWin $tcp $file"`<br>`}`<br>`$ns at 0.1 "recWin $tcp0 $cwndfile"` | Recursive function called to store the congestion TCP window size into a file |
| `$cbrX set maxpkts_ 300`<br>`$ns at 1.0 "$ftpX produce 300"` | **Applications Definition:** Instruction to produce 400 CBR and FTP packets each |
| `global xxxxxxxxxxx cwndfile`<br>`close $cwndfile` | **Termination:** Instructions to close the cwnd data file. Include these in the "finish" function. |

† Replace the 'X' in tcpX, udpX, cbrX, ftpX etc, with the appropriate digit from your TCL script. Further use the appropriate digit for node variables i.e., $nX.

## Expected Results:

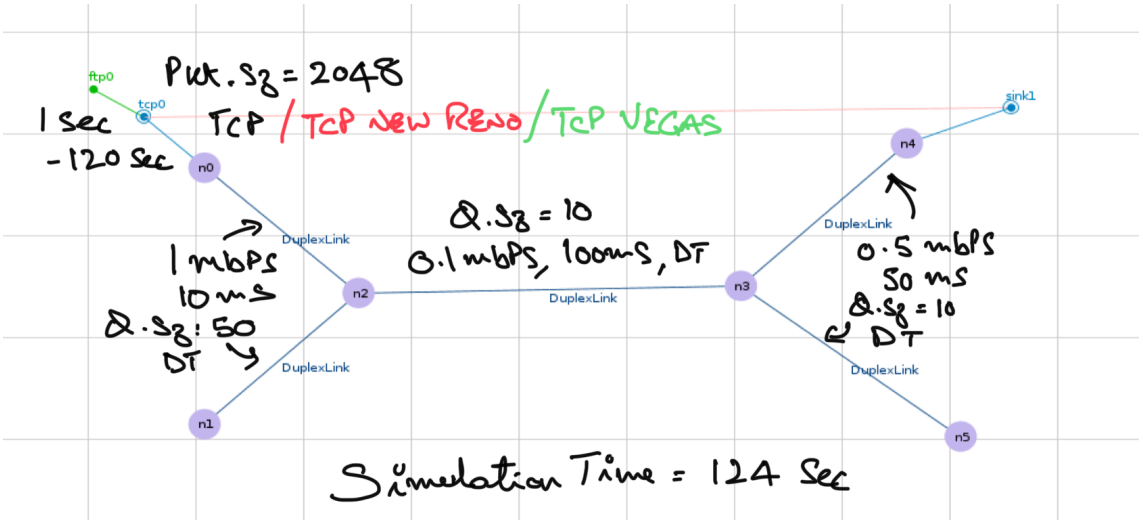





| Command | Description |
|---|---|
| **'cd '** | *Change directory* |
| **'ls'** | *Lists the files in a directory* |
| **'gedit Expxx.tcl &'** | *Create and Open Expxx.tcl script file to write tcl code using gedit (like notepad) editor* |
| **'java -jar NSG2.jar &'** | *Open the 'NS 2 Scenario Generator' application to generate tcl scripts for required network scenarios* |
| **'ns Expxx.tcl'** | *If script is error free, ns interpretor will generate two corresponding output files - NAM file (.nam) and Trace file (.tr)* |
| **'nam Expxx.nam'** | *The network animator utility will graphically display the network scenario created in the .nam file* |
| **'awf -f mgit_wired_05.awk Expxx.tr'** | *Analyze the network scenario data provided in the wired .tr file and compute the performance parameters. This script also creates Expxx-th.dat files for plotting throughput curves.* |
| **'awf -f mgit_wireless_01.awk Expxx.tr'** | *Analyze the network scenario data provided in the wireless .tr file and compute the performance parameters. This script also creates Expxx-th.dat files for plotting throughput curves* |
| **'gnuplot mgit_thputplot.p'** | *Searches the current directory for '*-th.dat' files and plots the throughput curve for each file* |
| **'gnuplot mgit_cwndplot.p'** | *Searches the current directory for '*-cwnd.dat' files and plots the congestion window size curve for each file* |

# 7. Addendum: Evaluate the performance of TCP, New Reno and Vegas

## 7.1. Performance evaluation of TCP



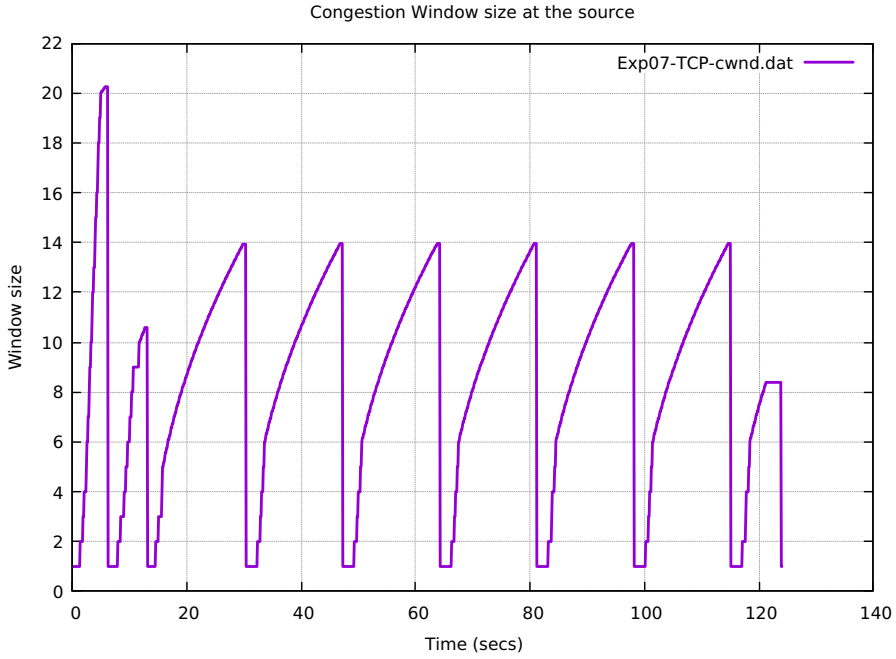

### 7.1.1. Network Scenarios and configurations:

Construct the networks with the following configurations. Use the following lines of code to change the color of the packets, monitor the queue, etc

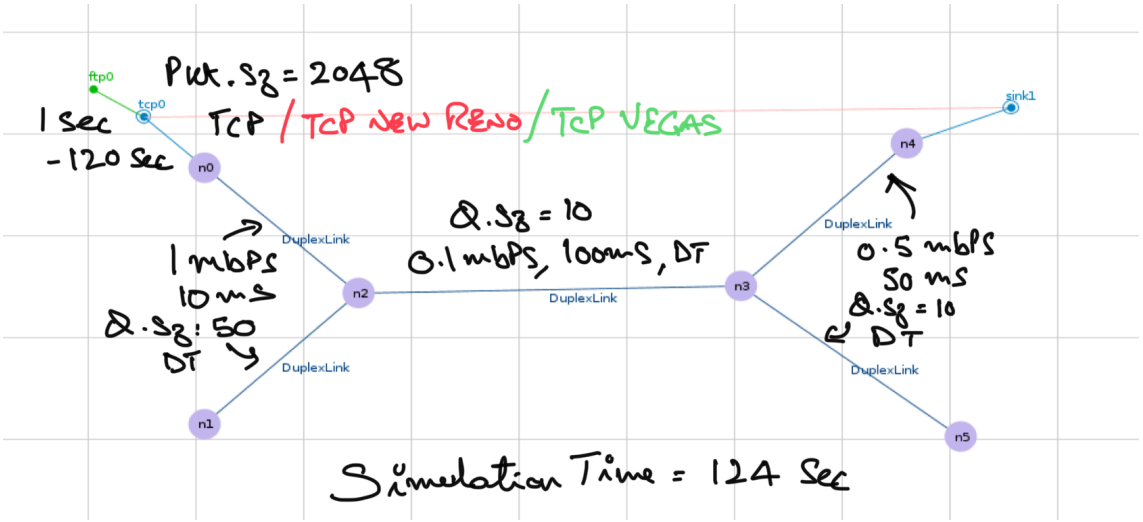
| Code | Comment |
|---|---|
| `$ns color 1 Red` | **Initialization:** Assign an identifier for each color |
| `set cwndfile [open Exp07-TCP-cwnd.dat w]` | File to store the TCP congestion window size at different time stamps |
| `$ns duplex-link-op $n2 $n3 queuePos 0.5` | **Link Definition:** Instruction for NAM to monitor the queue size |
| `$tcpX set fid_ 1` | **Agents Definition:** Assign a different identifier for each agent |
| `proc recWin { tcp file } {`<br>`global ns`<br>`set time 0.1`<br>`set ctime [$ns now]`<br>`set wnd [$tcp set cwnd_]`<br>`puts $file "$ctime $wnd"`<br>`$ns at [expr $ctime + $time] "recWin $tcp $file"`<br>`}`<br>`$ns at 0.1 "recWin $tcp0 $cwndfile"` | Recursive function called to store the congestion TCP window size into a file |
| `global xxxxxxxxxxx cwndfile`<br>`close $cwndfile` | **Termination:** Instructions to close the cwnd data file. Include these in the "finish" function. |

† Replace the 'X' in tcpX, udpX, cbrX, ftpX etc, with the appropriate digit from your TCL script. Further use the appropriate digit for node variables i.e., $nX.

**Expected Results:**




## 7.2. Performance evaluation of TCP New Reno



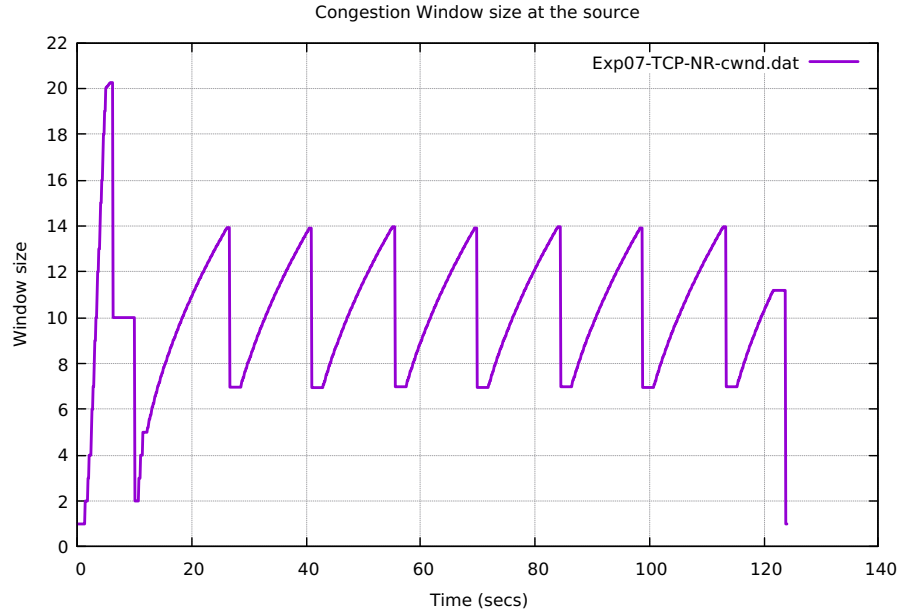

### 7.2.1. Network Scenarios and configurations:

Construct the networks with the following configurations. Use the following lines of code to change the color of the packets, monitor the queue, etc

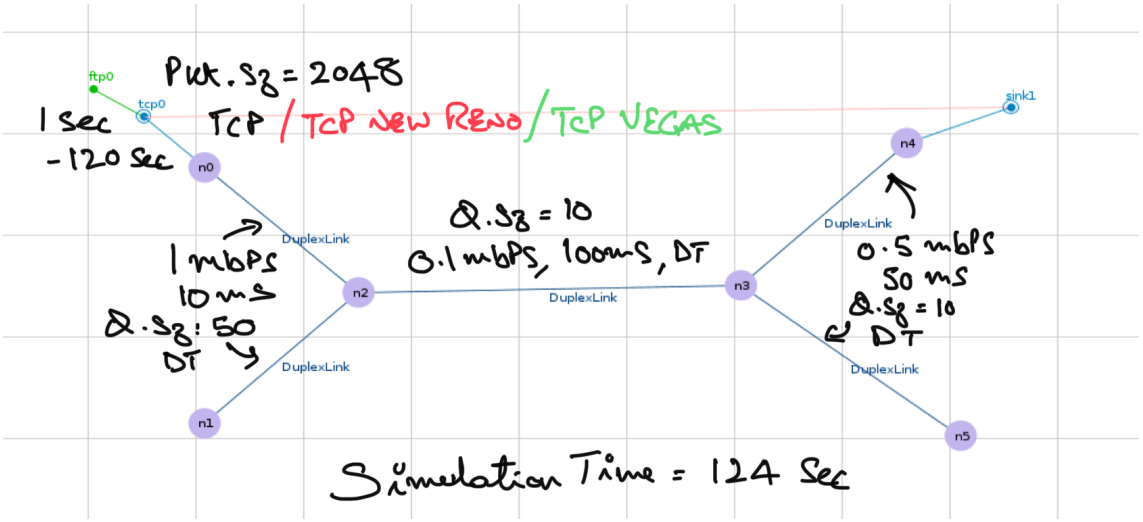
| Code | Comment |
|---|---|
| `$ns color 1 Red`<br><br>`set cwndfile [open Exp07-TCP-NR-cwnd.dat w]` | **Initialization:** Assign an identifier for each color<br><br>File to store the TCP congestion window size at different time stamps |
| `$ns duplex-link-op $n2 $n3 queuePos 0.5` | **Link Definition:** Instruction for NAM to monitor the queue size |
| `$tcpX set fid_ 1`<br><br>`proc recWin { tcp file } {`<br>`global ns`<br>`set time 0.1`<br>`set ctime [$ns now]`<br>`set wnd [$tcp set cwnd_]`<br>`puts $file "$ctime $wnd"`<br>`$ns at [expr $ctime + $time] "recWin $tcp $file"`<br>`}`<br>`$ns at 0.1 "recWin $tcp0 $cwndfile"` | **Agents Definition:** Assign a different identifier for each agent<br><br>Recursive function called to store the congestion TCP window size into a file |
| `global xxxxxxxxxxx cwndfile`<br>`close $cwndfile` | **Termination:** Instructions to close the cwnd data file. Include these in the "finish" function. |

† Replace the 'X' in tcpX, udpX, cbrX, ftpX etc, with the appropriate digit from your TCL script. Further use the appropriate digit for node variables i.e., $nX.

## Expected Results:



## 7.3. Performance evaluation of TCP Vegas



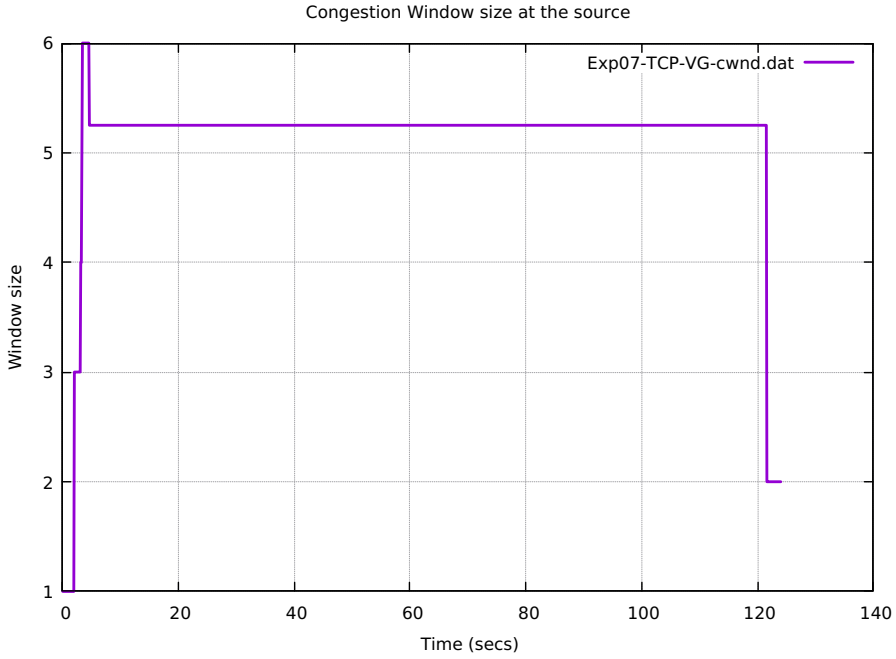

### 7.3.1. Network Scenarios and configurations:

Construct the networks with the following configurations. Use the following lines of code to change the color of the packets, monitor the queue, etc

| Code | Comment |
|---|---|
| `$ns color 1 Red`<br><br>`set cwndfile [open Exp07-TCP-VG-cwnd.dat w]` | **Initialization:** Assign an identifier for each color<br><br>File to store the TCP congestion window size at different time stamps |
| `$ns duplex-link-op $n2 $n3 queuePos 0.5` | **Link Definition:** Instruction for NAM to monitor the queue size |
| `$tcpX set fid_ 1`<br><br>`proc recWin { tcp file } {`<br>`global ns`<br>`set time 0.1`<br>`set ctime [$ns now]`<br>`set wnd [$tcp set cwnd_]`<br>`puts $file "$ctime $wnd"`<br>`$ns at [expr $ctime + $time] "recWin $tcp $file"`<br>`}`<br>`$ns at 0.1 "recWin $tcp0 $cwndfile"` | **Agents Definition:** Assign a different identifier for each agent<br><br>Recursive function called to store the congestion TCP window size into a file |
| `global xxxxxxxxxxx cwndfile`<br>`close $cwndfile` | **Termination:** Instructions to close the cwnd data file. Include these in the "finish" function. |

† Replace the 'X' in tcpX, udpX, cbrX, ftpX etc, with the appropriate digit from your TCL script. Further use the appropriate digit for node variables i.e., $nX.

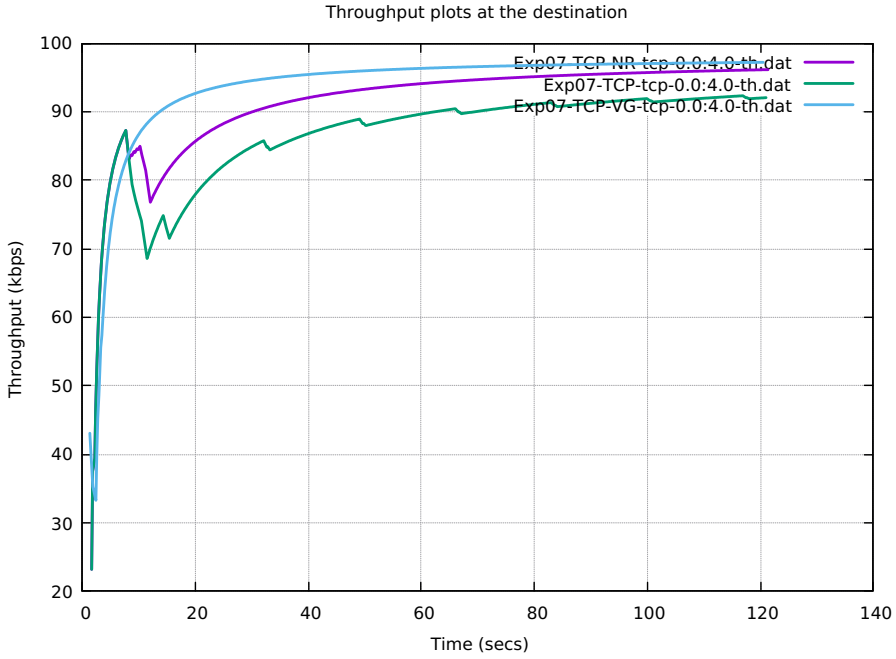
**Expected Results:**




## 7.4. Capture the consolidated performance parameters:

| Parameters | TCP | TCP New Reno | TCP Vegas |
|---|---|---|---|
| Src_addr → Dst_addr | | | |
| Pkt Type | | | |
| Pkt Size | | | |
| Transfer start time | | | |
| Transfer end time | | | |
| # Sent Pkts | | | |
| # Dropped Pkts (Q Full) | | | |
| # Dropped Pkts (Errors) | | | |
| # Discard Pkts (BrdCst) | | | |
| # Retransmitted Pkts | | | |
| # Duplicate Pkts | | | |
| # Delivered Pkts | | | |
| Pkt Delivery Ratio | | | |
| Pkt Drop Ratio | | | |
| Average Delay | | | |
| Throughput | | | |

**Expected Results:**



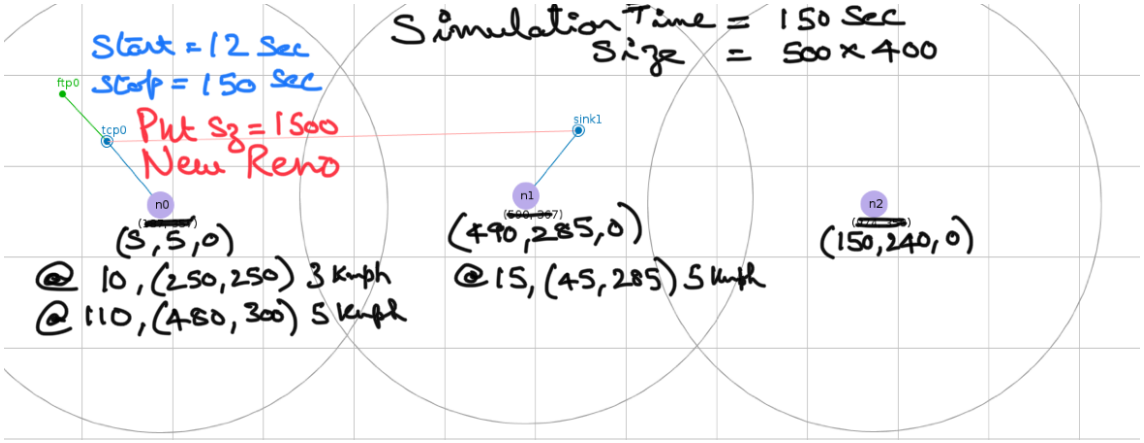

**Frequently used terminal commands:**

| Command | Description |
|---|---|
| **'cd '** | *Change directory* |
| **'ls'** | *Lists the files in a directory* |
| **'gedit Expxx.tcl &'** | *Create and Open Expxx.tcl script file to write tcl code using gedit (like notepad) editor* |
| **'java -jar NSG2.jar &'** | *Open the 'NS 2 Scenario Generator' application to generate tcl scripts for required network scenarios* |
| **'ns Expxx.tcl'** | *If script is error free, ns interpretor will generate two corresponding output files - NAM file (.nam) and Trace file (.tr)* |
| **'nam Expxx.nam'** | *The network animator utility will graphically display the network scenario created in the .nam file* |
| **'awf -f mgit_wired_05.awk Expxx.tr'** | *Analyze the network scenario data provided in the wired .tr file and compute the performance parameters. This script also creates Expxx-th.dat files for plotting throughput curves.* |
| **'awf -f mgit_wireless_01.awk Expxx.tr'** | *Analyze the network scenario data provided in the wireless .tr file and compute the performance parameters. This script also creates Expxx-th.dat files for plotting throughput curves* |
| **'gnuplot mgit_thputplot.p'** | *Searches the current directory for '*-th.dat' files and plots the throughput curve for each file* |
| **'gnuplot mgit_cwndplot.p'** | *Searches the current directory for '*-cwnd.dat' files and plots the congestion window size curve for each file* |

# 8. Addendum: Evaluate the performance of AODV and DSR routing protocols

# 9. Addendum: Evaluate the performance of AODV and DSDV routing protocols

## 9.1. Performance evaluation of AODV (Ad-hoc On Demand Distance Vector) routing protocol

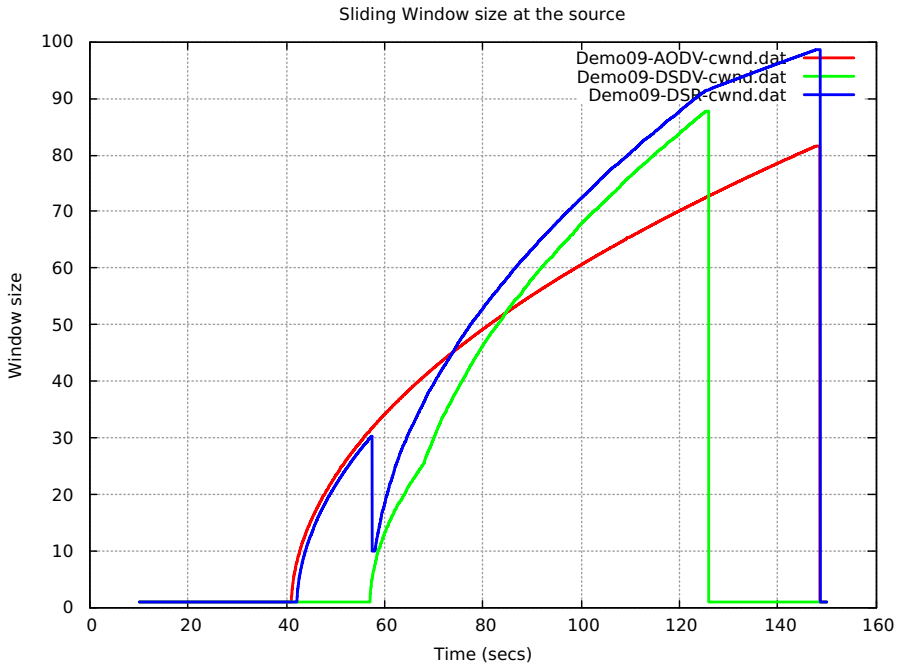


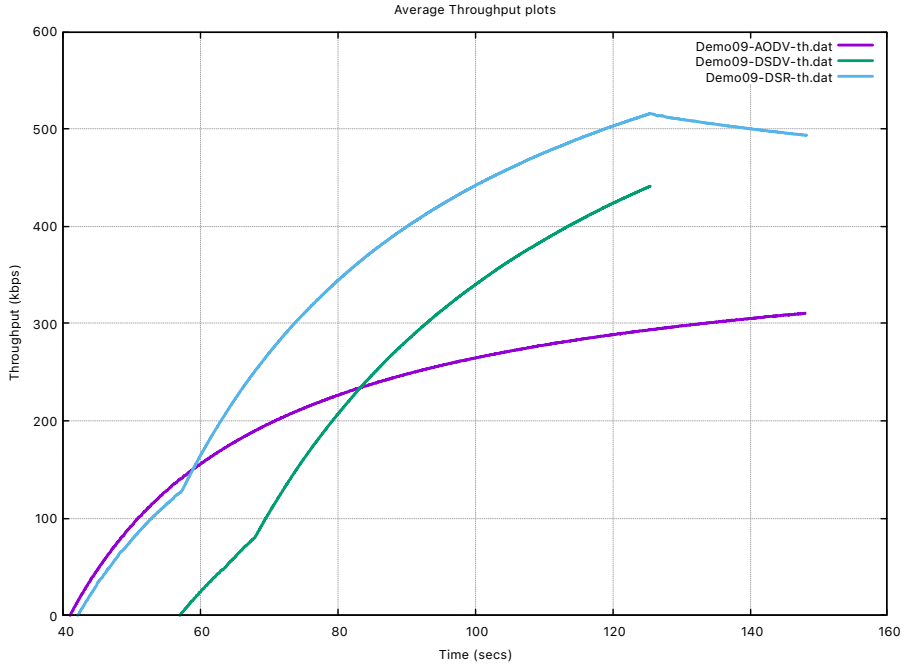

### 9.1.1. Network Scenarios and configurations:

Construct the networks with the following configurations. Use the following lines of code to change the color of the packets, monitor the queue, etc

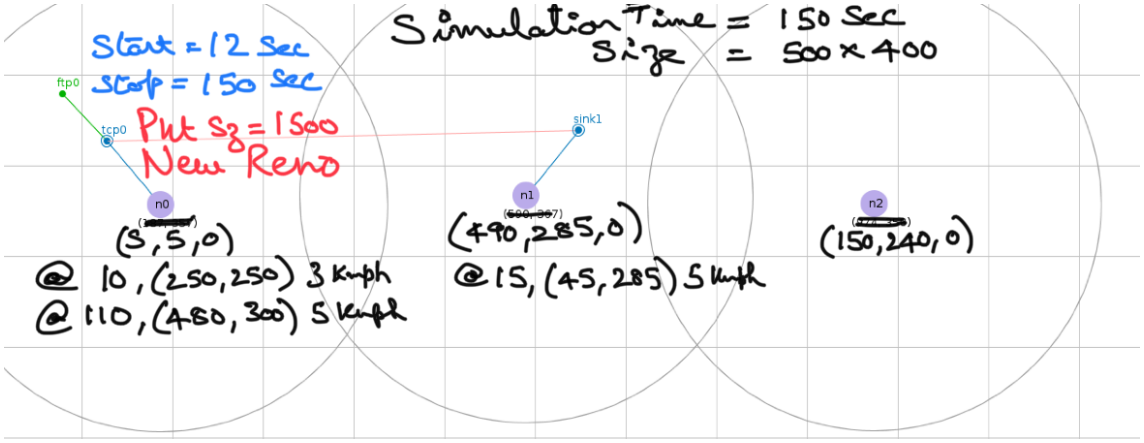
| Code | Comment |
|---|---|
| `set val(rp) AODV`<br>`set val(x) 500`<br>`set val(y) 400` | **Simulation parameters setup:** Use this code to set the routing protocol and topology size |
| `set cwndfile [open Exp09-AODV-cwnd.dat w]` | **Initialization:** File to store the TCP congestion window size at different time stamps |
| `$n0 set X_ 5`<br>`$n0 set Y_ 5`<br>`$n0 set Z_ 0` | **Nodes Definition:** Instructions to define the initial position of a node. Repeat these instructions for each node. |
| `proc recWin { tcp file } {`<br>`global ns`<br>`set time 0.1`<br>`set ctime [$ns now]`<br>`set wnd [$tcp set cwnd_]`<br>`puts $file "$ctime $wnd"`<br>`$ns at [expr $ctime + $time] "recWin $tcp $file"`<br>`}`<br>`$ns at 0.1 "recWin $tcp0 $cwndfile"` | **Agents Definition:** Recursive function called to store the congestion TCP window size into a file |
| `global xxxxxxxxxxx cwndfile`<br>`close $cwndfile` | **Termination:** Instructions to close the cwnd data file. Include these in the "finish" function. |

**Expected Results:**



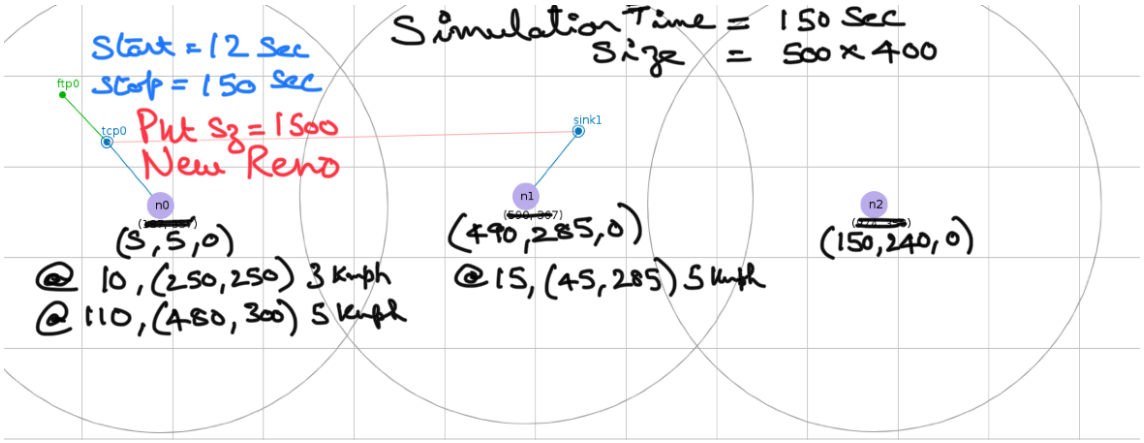



## 9.2. Performance evaluation of DSDV (Destination Sequenced Distance Vector) routing protocol




| Code | Comment |
|---|---|
| `set val(rp) DSDV`<br>`set val(x) 500`<br>`set val(y) 400` | **Simulation parameters setup:** Use this code to set the routing protocol and topology size |
| `set cwndfile [open Exp09-DSDV-cwnd.dat w]` | **Initialization:** File to store the TCP congestion window size at different time stamps |
| `$n0 set X_ 5`<br>`$n0 set Y_ 5`<br>`$n0 set Z_ 0` | **Nodes Definition:** Instructions to define the initial position of a node. Repeat these instructions for each node. |
| `proc recWin { tcp file } {`<br>`global ns`<br>`set time 0.1`<br>`set ctime [$ns now]`<br>`set wnd [$tcp set cwnd_]`<br>`puts $file "$ctime $wnd"`<br>`$ns at [expr $ctime + $time] "recWin $tcp $file"`<br>`}`<br>`$ns at 0.1 "recWin $tcp0 $cwndfile"` | **Agents Definition:** Recursive function called to store the congestion TCP window size into a file |
| `global xxxxxxxxxxx cwndfile`<br>`close $cwndfile` | **Termination:** Instructions to close the cwnd data file. Include these in the "finish" function. |

## 9.3. Performance evaluation of DSR (Dynamic Source Routing) routing protocol




| Code | Comment |
|---|---|
| `set val(rp) DSR`<br>`set val(ifq) CMUPriQueue`<br>`set val(x) 500`<br>`set val(y) 400` | **Simulation parameters setup:** Use this code to set the routing protocol and topology size |
| `set cwndfile [open Exp09-DSR-cwnd.dat w]` | **Initialization:** File to store the TCP congestion window size at different time stamps |
| `$n0 set X_ 5`<br>`$n0 set Y_ 5`<br>`$n0 set Z_ 0` | **Nodes Definition:** Instructions to define the initial position of a node. Repeat these instructions for each node. |
| `proc recWin { tcp file } {`<br>`global ns`<br>`set time 0.1`<br>`set ctime [$ns now]`<br>`set wnd [$tcp set cwnd_]`<br>`puts $file "$ctime $wnd"`<br>`$ns at [expr $ctime + $time] "recWin $tcp $file"`<br>`}`<br>`$ns at 0.1 "recWin $tcp0 $cwndfile"` | **Agents Definition:** Recursive function called to store the congestion TCP window size into a file |
| `global xxxxxxxxxxx cwndfile`<br>`close $cwndfile` | **Termination:** Instructions to close the cwnd data file. Include these in the "finish" function. |

## 9.4. Capture the consolidated performance parameters:

| Parameters | AODV | DSDV | DSR |
|---|---|---|---|
| **Transfer start time** | | | |
| **Transfer end time** | | | |
| **# Sent Pkts** | | | |
| **# Dropped Pkts** | | | |
| **# Delivered Pkts** | | | |
| **Pkt Delivery Ratio** | | | |
| **Pkt Drop Ratio** | | | |
| **Average Delay** | | | |
| **Throughput** | | | |

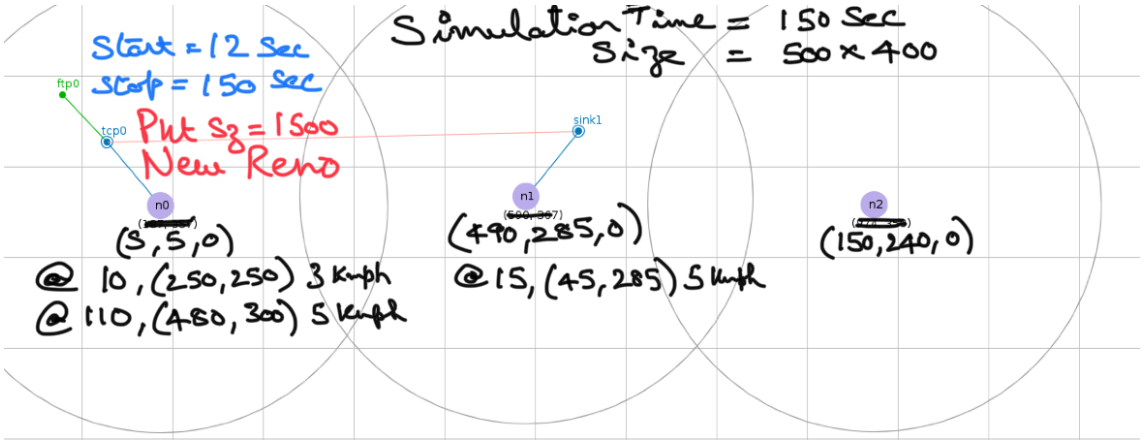
## Frequently used terminal commands:

| Command | Description |
|---|---|
| **'cd '** | *Change directory* |
| **'ls'** | *Lists the files in a directory* |
| **'gedit Expxx.tcl &'** | *Create and Open Expxx.tcl script file to write tcl code using gedit (like notepad) editor* |
| **'java -jar NSG2.jar &'** | *Open the 'NS 2 Scenario Generator' application to generate tcl scripts for required network scenarios* |
| **'ns Expxx.tcl'** | *If script is error free, ns interpretor will generate two corresponding output files - NAM file (.nam) and Trace file (.tr)* |
| **'nam Expxx.nam'** | *The network animator utility will graphically display the network scenario created in the .nam file* |
| **'awf -f mgit_wired_05.awk Expxx.tr'** | *Analyze the network scenario data provided in the wired .tr file and compute the performance parameters. This script also creates Expxx-th.dat files for plotting throughput curves.* |
| **'awf -f mgit_wireless_01.awk Expxx.tr'** | *Analyze the network scenario data provided in the wireless .tr file and compute the performance parameters. This script also creates Expxx-th.dat files for plotting throughput curves* |
| **'gnuplot mgit_thputplot.p'** | *Searches the current directory for '*-th.dat' files and plots the throughput curve for each file* |
| **'gnuplot mgit_cwndplot.p'** | *Searches the current directory for '*-cwnd.dat' files and plots the congestion window size curve for each file* |

# 10. Evaluate the performance of IEEE 802.11 and IEEE 802.15.4

# 11. Evaluate the performance of IEEE 802.11 and SMAC

## 11.1. Evaluate the performance of IEEE 802.11 (Wireless LAN)

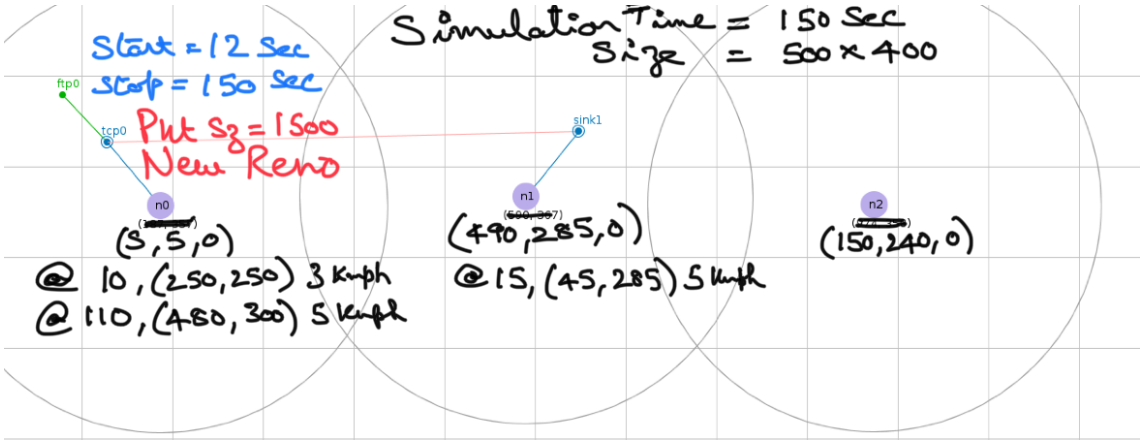



### 11.1.1. Network Scenarios and configurations:

Construct the networks with the following configurations. Use the following lines of code to change the color of the packets, monitor the queue, etc

| Code | Comment |
|---|---|
| `set val(netif) Phy/WirelessPhy`<br>`set val(mac) Mac/802_11`<br>`set val(rp) AODV`<br>`set val(x) 500`<br>`set val(y) 400` | **Simulation parameters setup:** Use this code to set the routing protocol and topology size |
| `set cwndfile [open Exp10-802.11-cwnd.dat w]` | **Initialization:** File to store the TCP congestion window size at different time stamps |
| `$n0 set X_ 5`<br>`$n0 set Y_ 5`<br>`$n0 set Z_ 0` | **Nodes Definition:** Instructions to define the initial position of a node. Repeat these instructions for each node. |
| `proc recWin { tcp file } {`<br>`global ns`<br>`set time 0.1`<br>`set ctime [$ns now]`<br>`set wnd [$tcp set cwnd_]`<br>`puts $file "$ctime $wnd"`<br>`$ns at [expr $ctime + $time] "recWin $tcp $file"`<br>`}`<br>`$ns at 0.1 "recWin $tcp0 $cwndfile"` | **Agents Definition:** Recursive function called to store the congestion TCP window size into a file |
| `global xxxxxxxxxxx cwndfile`<br>`close $cwndfile` | **Termination:** Instructions to close the cwnd data file. Include these in the "finish" function. |

## 11.2. Evaluate the performance of IEEE 802.15.4 (Wireless Personal Area Network)

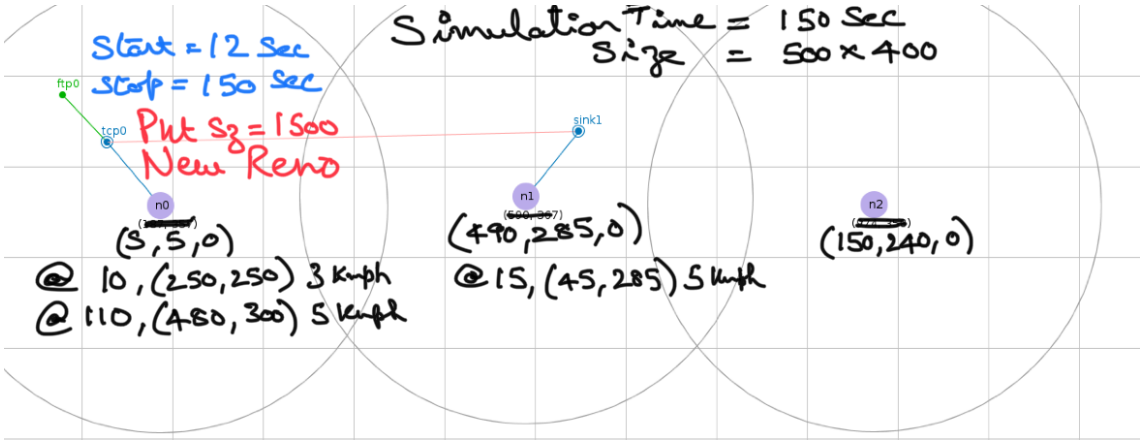



### 11.2.1. Network Scenarios and configurations:

Construct the networks with the following configurations. Use the following lines of code to change the color of the packets, monitor the queue, etc

| Code | Comment |
|---|---|
| `set val(netif) Phy/WirelessPhy/802_15_4`<br>`set val(mac) Mac/802_15_4`<br>`set val(rp) AODV`<br>`set val(x) 500`<br>`set val(y) 400` | **Simulation parameters setup:** Use this code to set the routing protocol and topology size |
| `set cwndfile [open Exp10-802.15.4-cwnd.dat w]` | **Initialization:** File to store the TCP congestion window size at different time stamps |
| `$n0 set X_ 5`<br>`$n0 set Y_ 5`<br>`$n0 set Z_ 0` | **Nodes Definition:** Instructions to define the initial position of a node. Repeat these instructions for each node. |
| `proc recWin { tcp file } {`<br>`global ns`<br>`set time 0.1`<br>`set ctime [$ns now]`<br>`set wnd [$tcp set cwnd_]`<br>`puts $file "$ctime $wnd"`<br>`$ns at [expr $ctime + $time] "recWin $tcp $file"`<br>`}`<br>`$ns at 0.1 "recWin $tcp0 $cwndfile"` | **Agents Definition:** Recursive function called to store the congestion TCP window size into a file |
| `global xxxxxxxxxxx cwndfile`<br>`close $cwndfile` | **Termination:** Instructions to close the cwnd data file. Include these in the "finish" function. |

## 11.3. Evaluate the performance of SMAC (Sensor MAC)

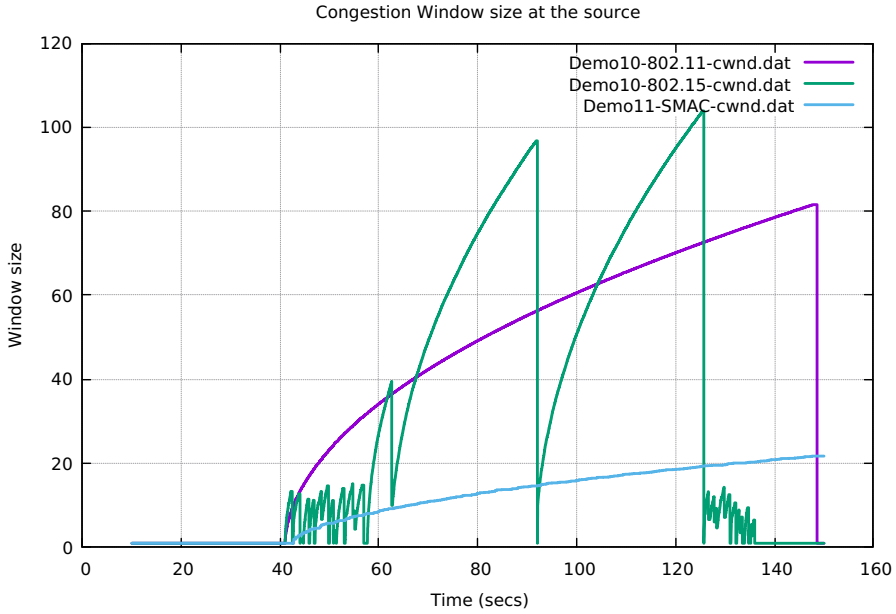


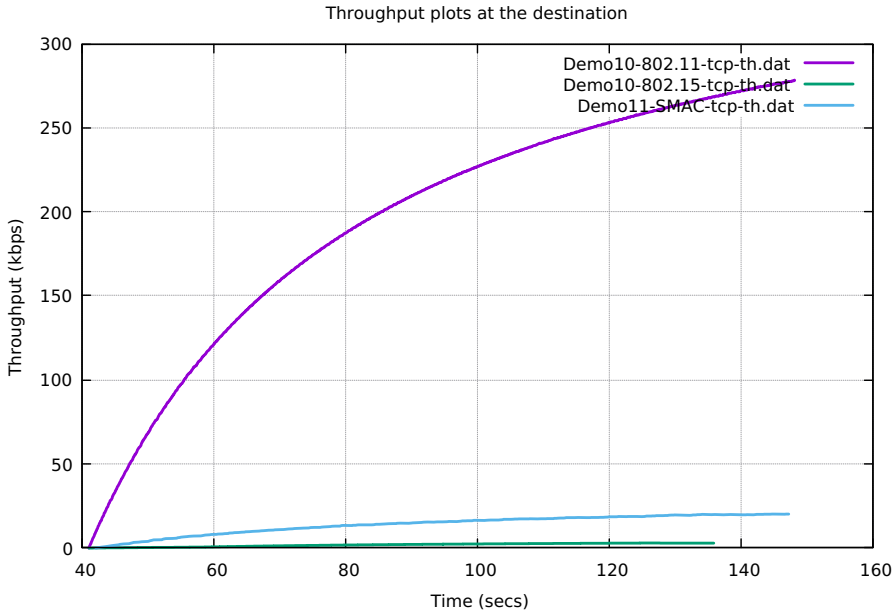

### 11.3.1. Network Scenarios and configurations:

Construct the networks with the following configurations. Use the following lines of code to change the color of the packets, monitor the queue, etc

| Code | Comment |
|---|---|
| `set val(netif) Phy/WirelessPhy`<br>`set val(mac) Mac/SMAC`<br>`set val(rp) AODV`<br>`set val(x) 500`<br>`set val(y) 400` | **Simulation parameters setup:** Use this code to set the routing protocol and topology size |
| `set cwndfile [open Exp11-SMAC-cwnd.dat w]` | **Initialization:** File to store the TCP congestion window size at different time stamps |
| `$n0 set X_ 5`<br>`$n0 set Y_ 5`<br>`$n0 set Z_ 0` | **Nodes Definition:** Instructions to define the initial position of a node. Repeat these instructions for each node. |
| `proc recWin { tcp file } {`<br>`global ns`<br>`set time 0.1`<br>`set ctime [$ns now]`<br>`set wnd [$tcp set cwnd_]`<br>`puts $file "$ctime $wnd"`<br>`$ns at [expr $ctime + $time] "recWin $tcp $file"`<br>`}`<br>`$ns at 0.1 "recWin $tcp0 $cwndfile"` | **Agents Definition:** Recursive function called to store the congestion TCP window size into a file |
| `global xxxxxxxxxxx cwndfile`<br>`close $cwndfile` | **Termination:** Instructions to close the cwnd data file. Include these in the "finish" function. |

**Expected Results:**



## 11.4. Capture the consolidated performance parameters:

| Parameters | IEEE 802.11 | IEEE 802.15.4 | SMAC |
|---|---|---|---|
| **Transfer start time** | | | |
| **Transfer end time** | | | |
| **# Sent Pkts** | | | |
| **# Dropped Pkts** | | | |
| **# Delivered Pkts** | | | |
| **Pkt Delivery Ratio** | | | |
| **Pkt Drop Ratio** | | | |
| **Average Delay** | | | |
| **Throughput** | | | |

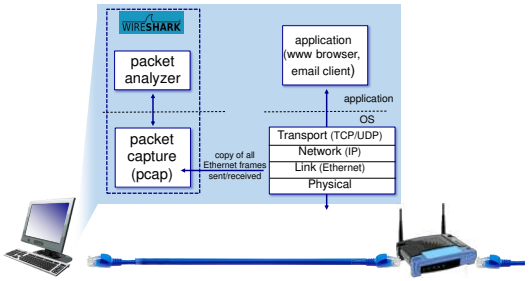
## Frequently used terminal commands:

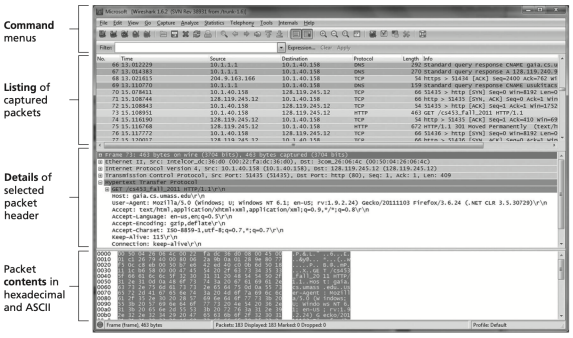
| Command | Description |
|---|---|
| **'cd '** | *Change directory* |
| **'ls'** | *Lists the files in a directory* |
| **'gedit Expxx.tcl &'** | *Create and Open Expxx.tcl script file to write tcl code using gedit (like notepad) editor* |
| **'java -jar NSG2.jar &'** | *Open the 'NS 2 Scenario Generator' application to generate tcl scripts for required network scenarios* |
| **'ns Expxx.tcl'** | *If script is error free, ns interpretor will generate two corresponding output files - NAM file (.nam) and Trace file (.tr)* |
| **'nam Expxx.nam'** | *The network animator utility will graphically display the network scenario created in the .nam file* |
| **'awf -f mgit_wired_05.awk Expxx.tr'** | *Analyze the network scenario data provided in the wired .tr file and compute the performance parameters. This script also creates Expxx-th.dat files for plotting throughput curves.* |
| **'awf -f mgit_wireless_01.awk Expxx.tr'** | *Analyze the network scenario data provided in the wireless .tr file and compute the performance parameters. This script also creates Expxx-th.dat files for plotting throughput curves* |
| **'gnuplot mgit_thputplot.p'** | *Searches the current directory for '*-th.dat' files and plots the throughput curve for each file* |
| **'gnuplot mgit_cwndplot.p'** | *Searches the current directory for '*-cwnd.dat' files and plots the congestion window size curve for each file* |

# Part II.

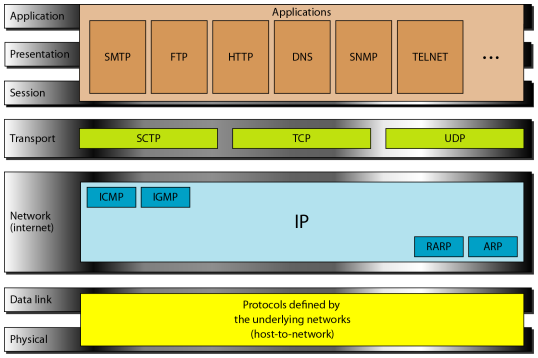
# WireShark Opensource Software (www.wireshark.org)

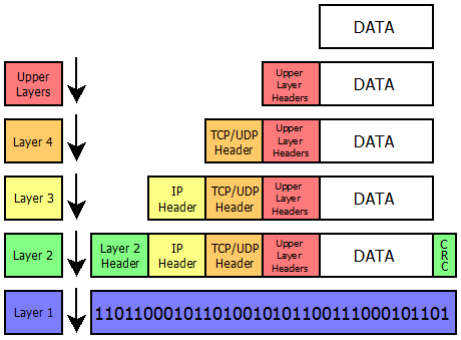
## About WireShark Opensource Software (www.wireshark.org):




## TCP / IP Protocol Suite:




## Traffic generation Tools and Utilities:

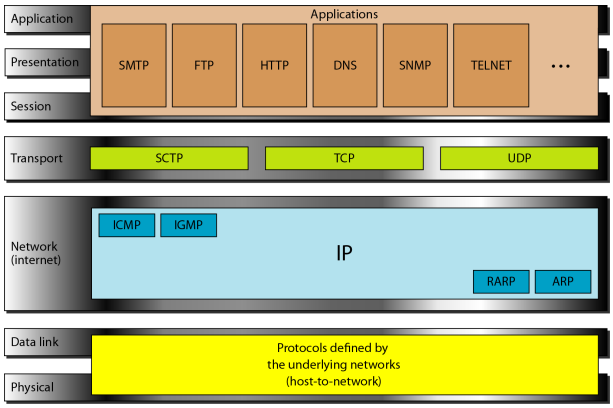
| Tools → | Browser ↓ | 'ftp' client↓ | 'nslookup'↓ | 'ping', 'tracert' or 'traceroute' ↓ |
|---|---|---|---|---|
| **Application Layer** | *HTTP message* | *FTP message* | *DNS message* | |
| **Transport Layer** | *TCP segments* | | *UDP datagrams* | |
| **Network Layer** | *IP pkts, ARP* | | | *ICMP, IP pkts, ARP* |
| **Data Link Layer** | *Frames* | | | |
| **Physical Layer** | *Bit streams* | | | |

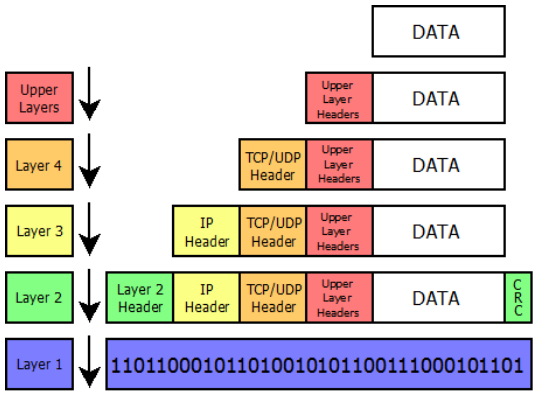
## Commands to obtain machine details:

| Command | Description |
|---|---|
| **'ipconfig /all'** | *To get the details of machine IP, machine MAC address, Gateway IP, etc* |
| **'ipconfig /displaydns'** | *To view the DNS mappings stored in your machine* |
| **'ipconfig /flushdns'** | *To delete the DNS mappings stored in your machine* |
| **'arp -a'** | *To view the IP address and MAC address mappings stored in your machine* |
| **'arp -d'** | *To empty the IP and MAC address mapping table* |

# 12. Addendum: Capture and analyse TCP and IP packets

## 12.1. TCP / IP Protocol Suite




## 12.2. TCP / IP traffic generation

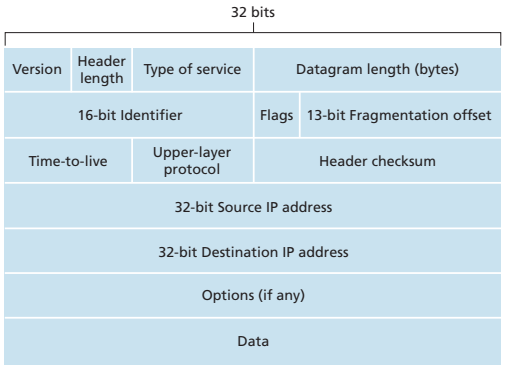
- Visit **www.google.com** or any other site from your browser.

### 12.2.1. IP Packet observed:

- Note down the details of any of the **IP packets** generated because of your browsing activity

| Field Name | Field length (# of bits) | Field Value (content carried) |
|---|---|---|
| Version | | |
| Header length | | |
| Type of service | | |
| Datagram length | | |
| 16-bit identifier | | |
| Flags | | |
| 13-bit fragmentation offset | | |
| Time-to-live | | |
| Upper-layer protocol | | |
| Header checksum | | |
| 32-bit source IP address | | |
| 32-bit destination IP address | | |
| Options (if any) | | |
| Data | | |

**Expected IP packet format:**

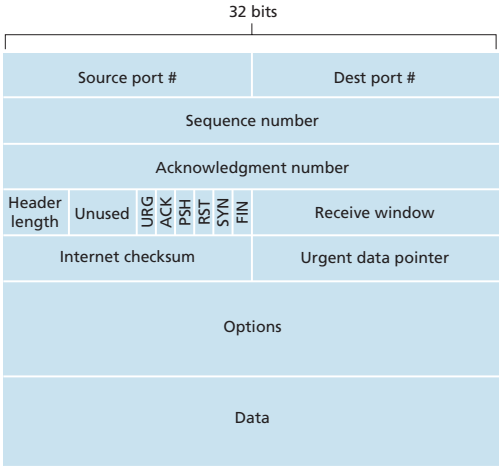



### 12.2.2. TCP Segment observed:

- Note down the details of any of the **TCP segments** generated because of your browsing activity

| Field Name | Field length (# of bits) | Field Value (content carried) |
|---|---|---|
| Source port | | |
| Dest port | | |
| Sequence number | | |
| Acknowledgement number | | |
| Header length | | |
| Flags (URG, ACK, PSH, RST, SYN, FIN) | | |
| Receive window | | |
| Internet checksum | | |
| Urgent data pointer | | |
| Options | | |
| Data | | |

**Expected TCP segment format:**




## 12.3. Additional Practical and Viva questions:

Note: Write the answers to these questions in your record

a. Observe and list the hexadecimal codes used in captured dataframes for characters A-Z, a-z and 0-9.

b. Observe the hexadecimal values corresponding to an image in the captured dataframes and save it to disk. Record the following

   → Name of the image,

   → Image type,

   → Size in bytes,

   → Starting and ending byte location in the dataframe

c. Analyze TCP and IP packets generated by applications like telnet or ftp

d. Capture and analyze UDP and IP packets (you can use the '**nslookup**' terminal utility to generate the UDP traffic)

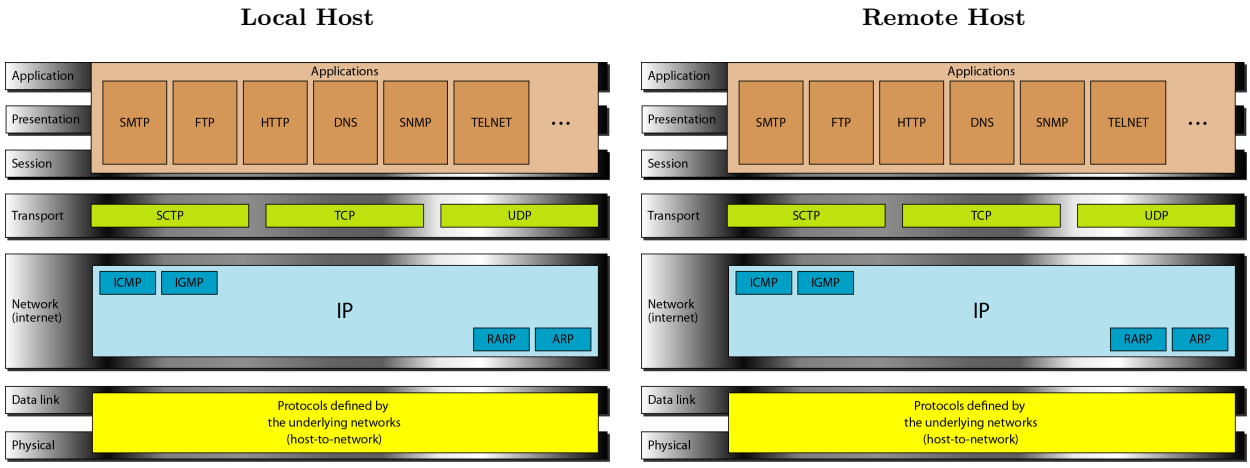
# 13. Addendum: Simulation and analysis of ICMP and IGMP packets

## 13.1. Analysis of ICMP packets

### 13.1.1. ICMP traffic generation

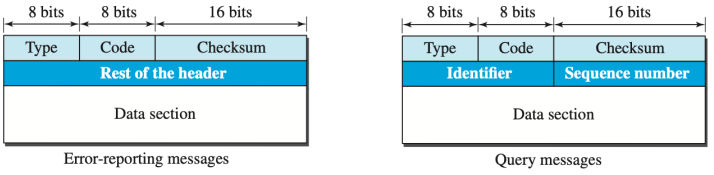
- Open your terminal and run the utility '**ping**' command. Ex: 'ping www.google.com'

### 13.1.2. TCP / IP Protocol Suites




### 13.1.3. Expected ICMP Packet format:

**Figure 19.8** *General format of ICMP messages*




**Type and code values**

| Error-reporting messages | Query messages |
|---|---|
| 03: Destination unreachable (codes 0 to 15) | 08 and 00: Echo request and reply (only code 0) |
| 04: Source quench (only code 0) | 13 and 14: Timestamp request and reply (only code 0) |
| 05: Redirection (codes 0 to 3) | |
| 11: Time exceeded (codes 0 and 1) | |
| 12: Parameter problem (codes 0 and 1) | |

### 13.1.4. ICMP Packet observed:

- Note down the details of a '**Request**' and a '**Reply**' ICMP packets generated because of your ping activity

Table 13.1.: **Request message**

| Field Name | Field length (# of bits) | Field Value (content carried) |
|---|---|---|
| Type | | |
| Code | | |
| Checksum | | |
| Content | | |

Table 13.2.: **Reply message**

| Field Name | Field length (# of bits) | Field Value (content carried) |
|---|---|---|
| Type | | |
| Code | | |
| Checksum | | |
| Content | | |

### 13.1.5. Additional Practical and Viva questions:

Note: Write the answers to these questions in your record

a. List the different values that the 'type' field of ICMP packet can take

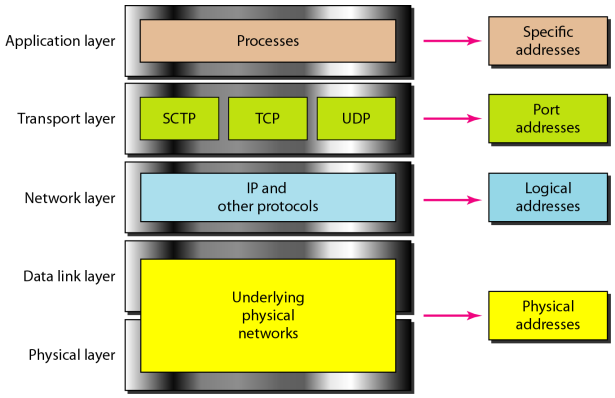
b. What does the 'checksum' field represent in an ICMP packet

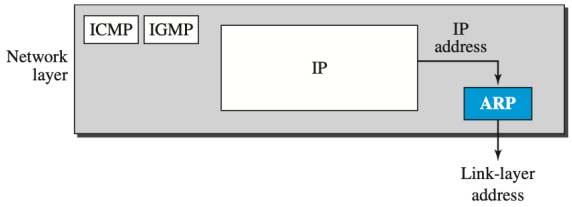
c. What does the 'sequence number' field represent in an ICMP packet

d. Capture and analyze the ICMP packets generated by '**tracert**' utility. What do you inference observing
   → the various types of ICMP packets generated
   → the TTL field values of various IP packets generated

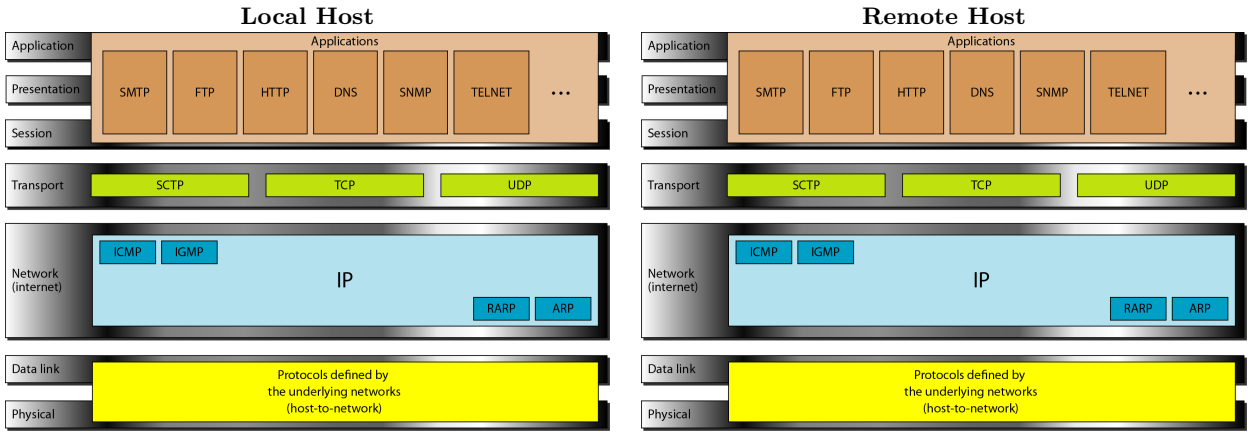
# 14. Addendum: Analyze the ARP protocol

## 14.1. Analyze ARP ( Address Resolution Protocol):




### 14.1.1. TCP / IP Protocol Suite




### 14.1.2. Capture ARP Traffic

i. Use '**ipconfig /all**' in command prompt to display the default gateway address. Note down the **Default Gateway** displayed

ii. Use '**arp -d**' in command prompt to clear the ARP cache (run the command prompt as administrator)

iii. Start Wireshark and start capture

iv. Use '**ping**' to ping the default gateway address (or any other IP address in the same LAN)

v. Use '**arp -a**' to view the ARP cache and confirm an entry has been added for the default gateway address

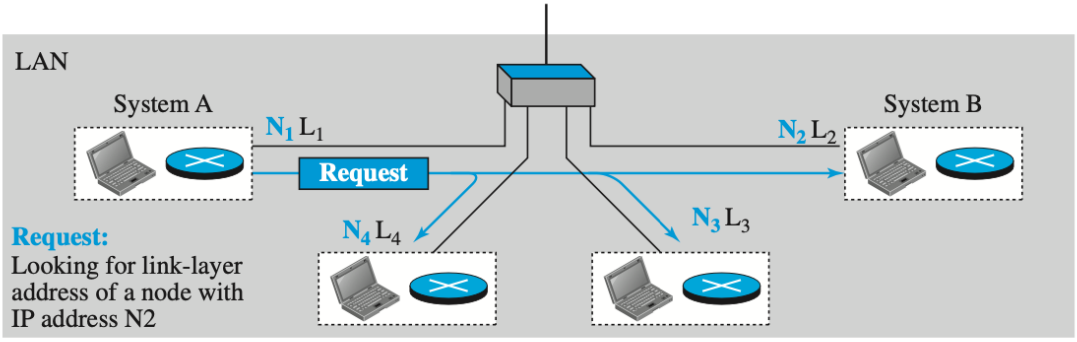
vi. Stop the Wireshark capture

vii. Note down the following values

Table 14.1.: **Machine Details**

| Address Name | Value |
|---|---|
| Your Machine IP addr | |
| Your Machine MAC addr | |
| Default Gateway IP addr | |
| Default Gateway MAC addr | |

### 14.1.3. Analyze an ARP Request




a. ARP request is broadcast

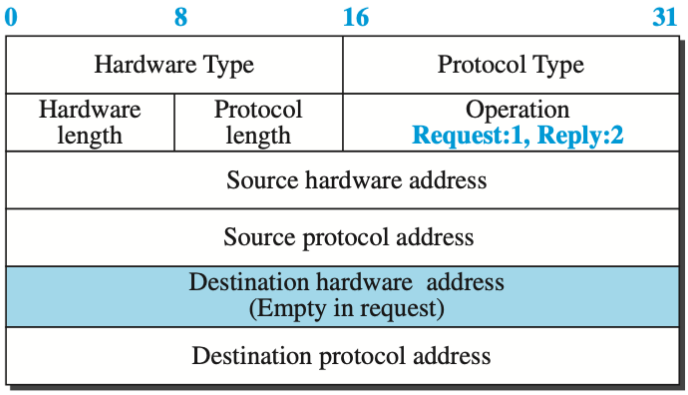
i. Observe the traffic captured in the top Wireshark packet list pane

ii. Select the first ARP packet request packet that you generated. (To identify the request-response pair that you generated, set the display filter in wireshark as '**arp.opcode==2**')

iii. Observe the Ethernet details in the middle pane

iv. Note that the format of the observed ARP packet should resemble the format shown below (however, the length of the hardware address, i.e., MAC address should be 48-bits in your case)

**Figure 9.8** *ARP packet*




v. Observe the Destination field. Notice the destination field is the Ethernet broadcast address i.e., 00 : 00 : 00 : 00 : 00 : 00. All devices on the network will receive this ARP request packet

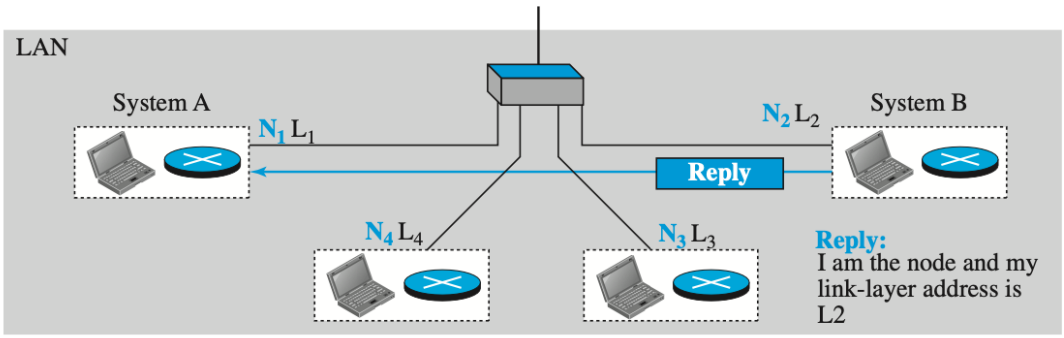
vi. Observe the Source field. This should contain your MAC address

vii. Note down your observation of the ARP request packet in the table below

Table 14.2.: **ARP Request Details**

| Field Name | Field length (# of bits) | Field Value (content carried) |
|---|---|---|
| Hardware type | | |
| Protocol type | | |
| Hardware length | | |
| Protocol length | | |
| Operation code | | |
| Source MAC address | | |
| Source IP address | | |
| Destination MAC address | | |
| Destination IP address | | |

### 14.1.4. Analyze an ARP Reply




b. ARP reply is unicast

i. Select the second ARP packet (reply packet)

ii. Observe the Ethernet details in the middle pane

iii. Note that the format of the observed ARP packet should resemble the format given above

iv. Observe the Destination field. Notice the destination field is your Ethernet address

v. Observe the Source field. This should contain the MAC address of the default gateway

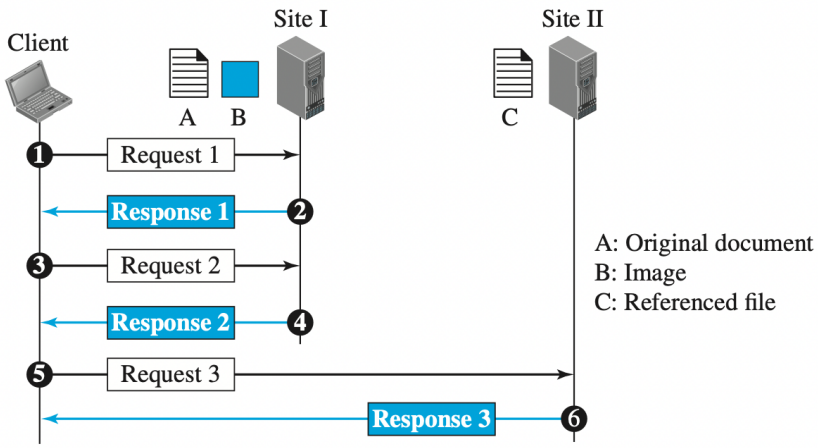
vi. Note down your observation of the ARP reply packet in the table below

Table 14.3.: **ARP Reply Details**

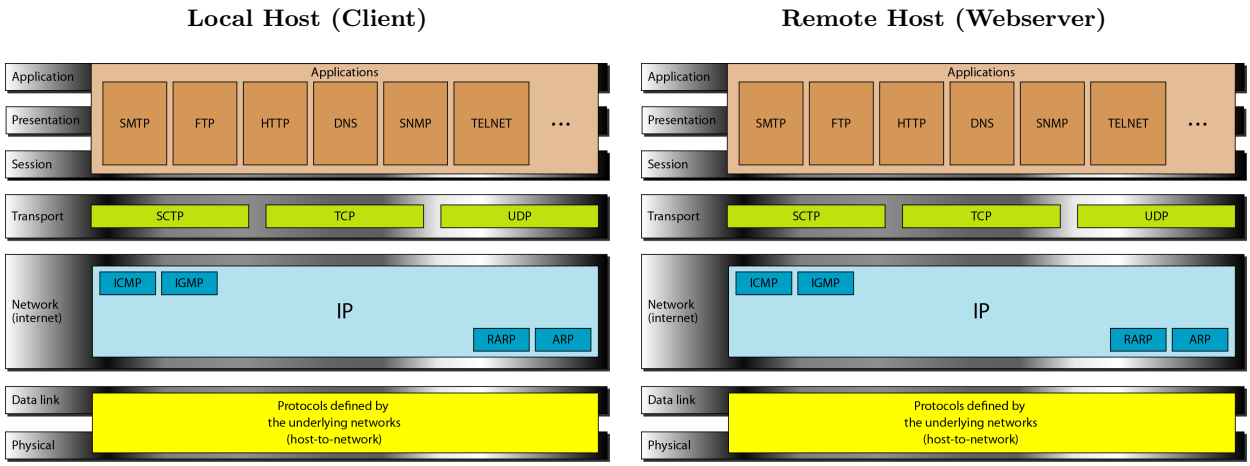
| **Field Name** | **Field length (# of bits)** | **Field Value (content carried)** |
|---|---|---|
| Hardware type | | |
| Protocol type | | |
| Hardware length | | |
| Protocol length | | |
| Opcode | | |
| Source MAC address | | |
| Source IP address | | |
| Destination MAC address | | |
| Destination IP address | | |

# 15. Addendum: Analyze the HTTP and DNS protocol

## 15.1. Analyze HTTP (Hypertext Transfer Protocol):




### 15.1.1. TCP / IP Protocol Suites




### 15.1.2. Capture HTTP Traffic

i. Open a new web browser window or tab (clear the history/cache of the browser when required)

ii. Start a Wireshark capture

iii. Navigate to any http website (Note that it should not be a **https** website)

iv. Stop the Wireshark capture

v. Note down the following values

Table 15.1.: **Machine Details**

| Parameter Name | Value |
|---|---|
| Your Machine IP addr | |
| Your Machine MAC addr | |
| Default Gateway MAC addr | |
| Website URL | |
| Website IP addr | |

### 15.1.3. Select Request Traffic

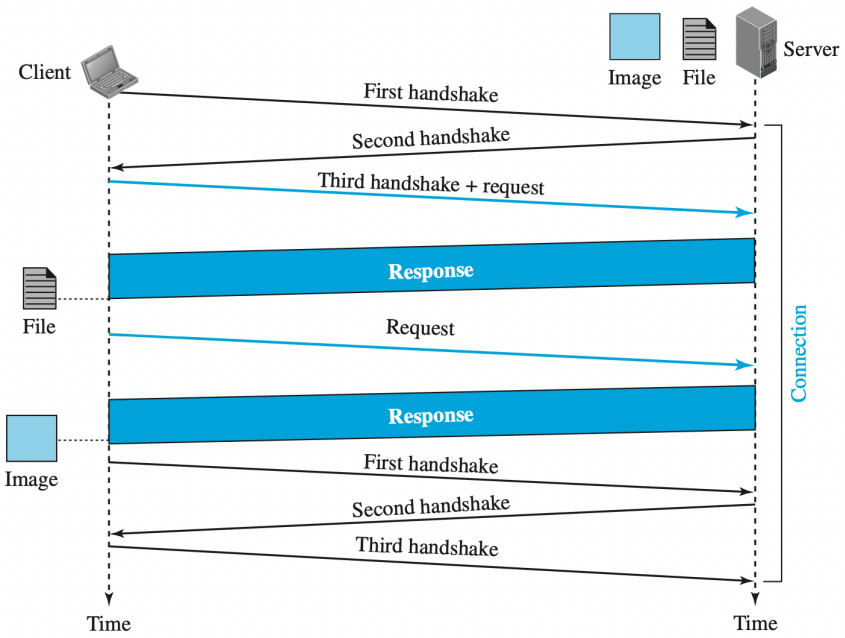
i. Observe the HTTP traffic captured in the Wireshark

ii. Select the first HTTP packet labeled **GET /**

iii. Observe the destination IP address

iv. To view all related traffic for this connection, change the Wireshark display filter to **ip.addr==destination ip address**

### 15.1.4. Analyze TCP Connection Traffic




i. Observe the traffic captured in the Wireshark. The first three packets (TCP SYN, TCP SYN/ACK, TCP ACK) are the TCP three way handshake. Select the first packet

ii. Observe the packet details. Notice that it is an Ethernet / Internet Protocol Version 4 / Transmission Control Protocol frame

iii. View the Ethernet details

→ Observe the Destination and Source fields. The destination should be your default gateway's MAC address and the source should be your MAC address. (This can be confirmed by using '**ipconfig /all**' and '**arp -a**' commands)

iv. Expand Internet Protocol V4 to view IP details

→ Observe the Source address. Notice that the source address is your IP address

→ Observe the Destination address. Notice that the destination address is the IP address of the HTTP server

v. Expand Transmission Control Protocol to view TCP details.

→ Observe the Source port. Notice that it is a dynamic port selected for this HTTP connection.

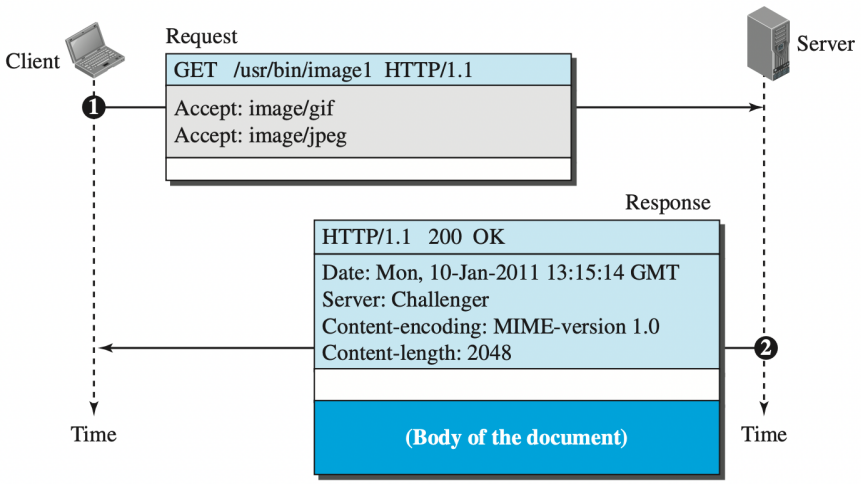
→ Observe the Destination port. Notice that it is http (80).

vi. Note down your observation in the table below

Table 15.2.: **TCP Connection Segment Details**

| Field Name | Field length (# of bits) | Field Value (content carried) |
|---|---|---|
| Destination MAC addr | | |
| Source MAC addr | | |
| Destination IP addr | | |
| Source IP addr | | |
| Destination TCP port | | |
| Source TCP port | | |

### 15.1.5. Analyze HTTP Request Traffic




i. Select the fourth packet, which is the first HTTP packet and labeled GET /.

ii. Observe the packet details in the middle Wireshark packet details pane.

iii. Notice that it is an Ethernet II / Internet Protocol Version 4 / Transmission Control Protocol / Hypertext Transfer Protocol frame.

iv. Also notice that the Ethernet II, Internet Protocol Version 4, and Transmission Control Protocol values are consistent with the TCP connection analyzed above

v. Expand Hypertext Transfer Protocol to view HTTP details.

→ Observe the GET request, Host, Connection, User-Agent, Referrer, Accept, and Cookie fields. This is the information passed to the HTTP server with the GET request.

vi. Select the fifth packet, labeled TCP ACK. This is the server TCP acknowledgement of receiving the GET request.

vii. Note down your observation in the table below

Table 15.3.: **HTTP Request Message Details**

| Field Name | Field length (# of bits) | Field Value (content carried) |
|---|---|---|
| Method | | |
| Host | | |
| Accept | | |
| User-Agent | | |
| Accept-Language | | |
| Accept-Encoding | | |
| Connection | | |

### 15.1.6. Analyze HTTP Response Traffic

i. Observe the traffic captured in the top Wireshark packet list pane.

ii. Select the second HTTP packet, labeled **HTTP 200 OK**.

iii. Expand Hypertext Transfer Protocol to view HTTP details.

iv. Observe the HTTP response, Server, Expires, Location, and other available information.

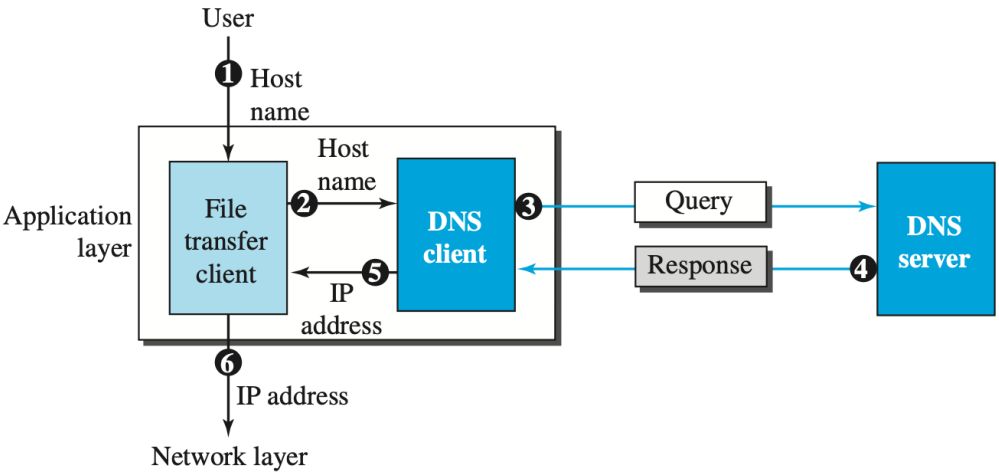
v. Select the next packet, labeled TCP ACK. This is the client TCP acknowledgement of receiving the HTTP response.

vi. Note down your observation in the table below

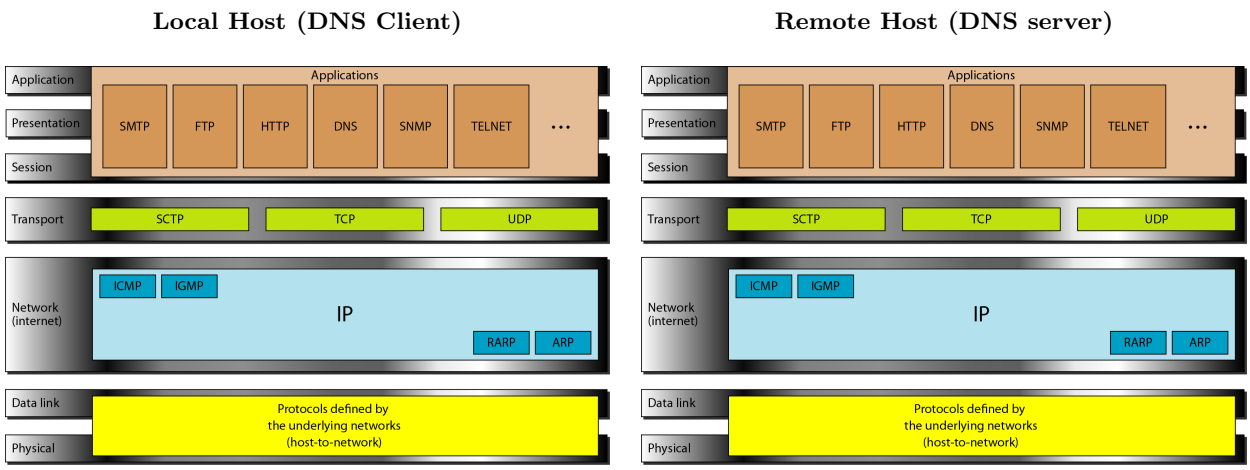
Table 15.4.: **HTTP Response Message Details**

| Field Name | Field length (# of bits) | Field Value (content carried) |
|---|---|---|
| Method | | |
| Date | | |
| Server | | |
| Last-modified | | |
| Accept-Ranges | | |
| Content-Length | | |
| Connection | | |
| Content-Type | | |

## 15.2. Analyze DNS (Domain Name System) protocol:




### 15.2.1. TCP / IP Protocol Suites




### 15.2.2. Capture DNS Traffic

i. Open a command prompt and type '**ipconfig /flushdns**' to clear the DNS cache

ii. Start a Wireshark capture

iii. Type '**nslookup nptel.ac.in**' in the command prompt (or browse the site using any web browser)

iv. Stop the Wireshark capture

v. Note down the following values observed by using '**ipconfig /all**' command

Table 15.5.: **Machine Details**

| Parameter Name | Value |
|---|---|
| Your Machine IP addr | |
| Your Machine MAC addr | |
| Default Gateway MAC addr | |
| DNS server IP addr | |

### 15.2.3. Analyze DNS Query Traffic

i. Observe the DNS traffic captured in the top Wireshark packet list pane (set display filter as 'dns')

ii. Select the DNS packet labeled **Standard query A nptel.ac.in**

iii. Notice that it is an Ethernet II / Internet Protocol Version 4 / User Datagram Protocol / Domain Name System (query) frame.

iv. Expand Ethernet II to view Ethernet details.

→ Observe the Destination and Source fields. The destination should be either your local DNS server's MAC address or your default gateway's MAC address and the source should be your MAC address. You can use '**ipconfig /all**' and '**arp -a**' to confirm.

v. Expand Internet Protocol Version 4 to view IP details.

→ Observe the Source address. Notice that the source address is your IP address.

→ Observe the Destination address. Notice that the destination address is the IP address of the DNS server.

vi. Expand User Datagram Protocol to view UDP details.

→ Observe the Source port. Notice that it is a dynamic port selected for this DNS query.

→ Observe the Destination port. Notice that it is domain (53), the DNS server port.

vii. Expand Domain Name System (query) to view DNS details.

→ Expand Flags to view flags details.

→ Observe the Recursion desired field. Notice that a recursive query is requested.

→ Expand Queries to view query details. Observe the query for nptel.ac.in

viii. Note down the following values

Table 15.6.: **DNS Query Details**

| Field Name | Field length (# of bits) | Field Value (content carried) |
|---|---|---|
| Destination MAC addr | | |
| Source MAC addr | | |
| Destination IP addr | | |
| Source IP addr | | |
| Destination UDP port | | |
| Source UDP port | | |
| DNS Tx Id | | |
| DNS Flags | | |
| DNS Questions | | |
| DNS Queries | | |

### 15.2.4. Analyze DNS Response Traffic

i. In the top Wireshark packet list pane, select the next DNS packet, labeled **Standard query response nptel.ac.in**

ii. Notice that it is an Ethernet II / Internet Protocol Version 4 / User Datagram Protocol / Domain Name System (response) frame.

iii. Expand Ethernet II to view Ethernet details.

→ Observe the Destination and Source fields. The destination should be your MAC address and the source should be your local DNS server's MAC address or your default gateway's MAC address.

iv. Expand Internet Protocol Version 4 to view IP details.

→ Observe the Source address. Notice that the source address is the DNS server IP address.

→ Observe the Destination address. Notice that the destination address is your IP address.

v. Expand User Datagram Protocol to view UDP details.

→ Observe the Source port. Notice that it is domain (53), the DNS server port.

→ Observe the Destination port. Notice that it is the same dynamic port used to make the DNS query in the first packet.

vi. Expand Domain Name System (query) to view DNS details.

→ Expand Flags to view flags details. Observe the flags. Notice that this is a recursive response.

→ Expand Queries to view query details.

→ Observe the query for *nptel.ac.in*

→ Expand Answers to view answer details.

→ Observe the CNAME and A records returned in response to this DNS query.

vii. Note down the following values

Table 15.7.: **DNS Response Details**

| **Field Name** | **Field length (# of bits)** | **Field Value (content carried)** |
|---|---|---|
| Destination MAC addr | | |
| Source MAC addr | | |
| Destination IP addr | | |
| Source IP addr | | |
| Destination UDP port | | |
| Source UDP port | | |
| DNS Tx Id | | |
| DNS Flags | | |
| DNS Queries | | |
| DNS Answers | | |

# Part III.

# Appendices

# A. Lab practice questions

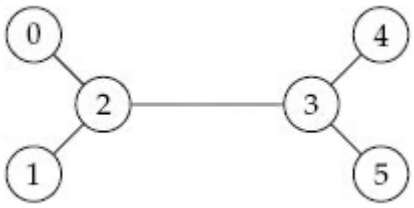
*Note: For each questions below, (a) Record the performance statistics of the network, (b) Obtain the throughput plots and (c) Obtain the congestion window size plots for the TCP sources if applicable*

1. Write a TCL program to create a four-node network, where n0 and n1 are connected to node n2. Node n2 is connected to n3. All links are of 10 Mbps bandwidth. Attach CBR traffic source to both n0 and n1, and attach the destination to n3. Assign different colors to different traffic and run the simulation for 50 seconds.

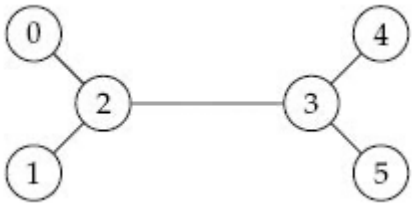
2. Write a TCL program to connect a ring network of five nodes with a star network of six nodes. Send traffic between the two networks

3. Create a network as given in figure. Now attach two TCP agents to node 0 and node 1 and sink agents to nodes 4 and 5, respectively. Make one TCP to follow stop-and-wait ARQ and the other to follow a sliding window of size 8 packets. Now run the program and compare the throughput of different connections.
   Hint: The congestion window of a TCP agent can be set to a fixed value n by using the code statement

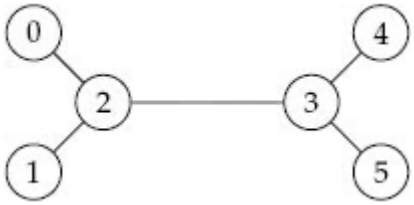
   ```
   $tcpX set maxcwnd_ n
   ```



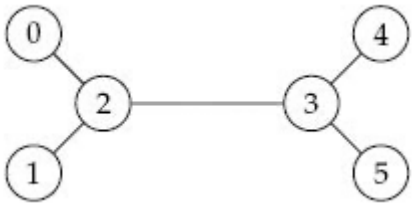

4. In the network shown in figure, attach one TCP agent to node 1. Now measure the congestion window variation and throughput for different queue limits (of 5, 8, 10 and 12) for the link 2-3.



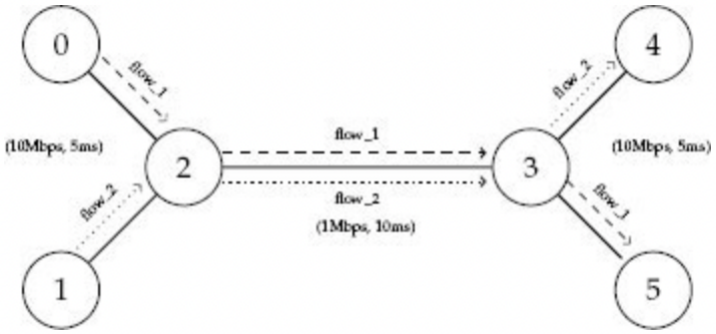

5. In the network shown in figure, attach one UDP agent to node 0 and another TCP agent to node 1. Measure the congestion window variation and throughput for different packet sizes (i.e., of 500 bytes, 750 bytes, 1000, 1500 and 2000 bytes)



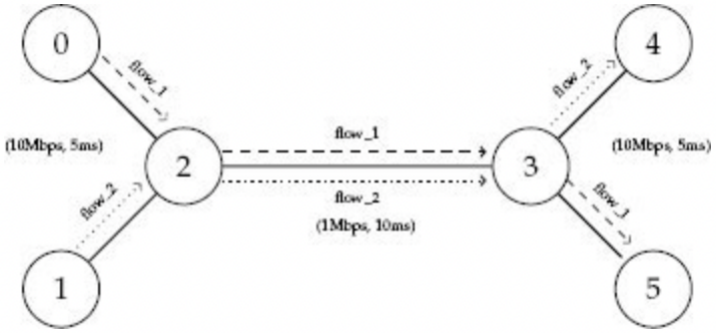

6. In the network shown in figure, attach two TCP agents to node 0 and node 1 with different packet sizes (500 and 1000 bytes). Now measure the cogestion window variation and throughput for both TCPs.



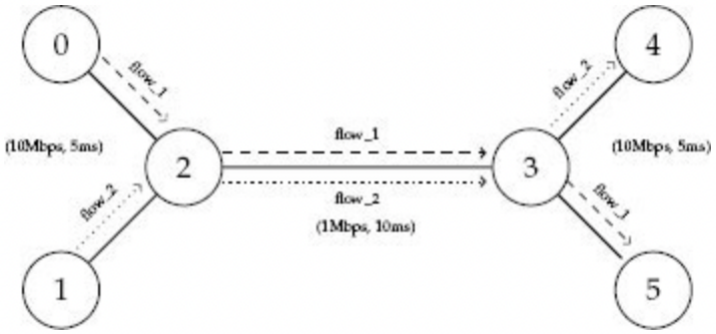

7. Find the average throughput for different flows by varying the CBR packet size from 200 to 1000 bytes, with a step size of 100 bytes.



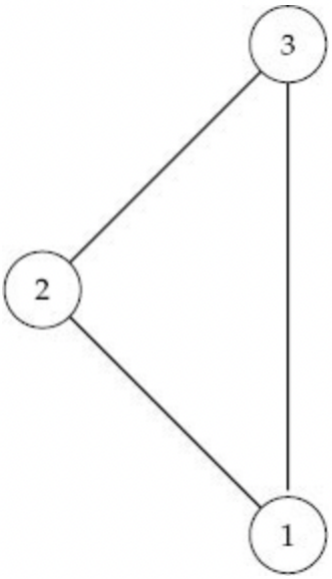

8. Find the average throughput for different flows by varying the CBR packet interval from 0.001 to 0.006, with a step size of 0.001




9. In the network shown in figure below, use two ftp traffics and plot the congestion window and throughput plots for different flows using drop tail, RED, FQ and CBQ




10. In the network shown in figure below, attach a CBR traffic between node 1 and node 3. Attach an error model with the probability of packet drop set as 0.01 to the link 1-3. Plot the performance with varying simulation time.


3
2
1

# B. References


1. Website: https://isi.edu/nsnam/ns/
2. Help pages: http://http://nsnam.sourceforge.net/wiki/index.php/Main_Page
3. FAQ pages: http://nsnam.sourceforge.net/wiki/index.php/Ns_Users_FAQ
4. NS simulator for beginners, Lecture Notes, by Eltan Altman and others
5. Computer Network Simulation using NS2, Ajith Kumar Nayak and others
6. www.wireshark.org