1. Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth, and find the number of packets dropped.

Step 1: Open the Terminal in Fedora (Username: root Password: fedora)

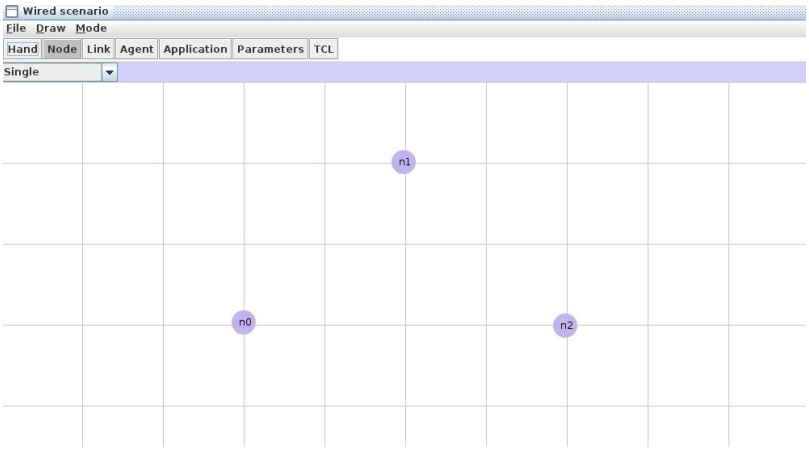
Step 2: Type the command in the Terminal

java -jar NSG2.1.jar

Step 2: In the new window

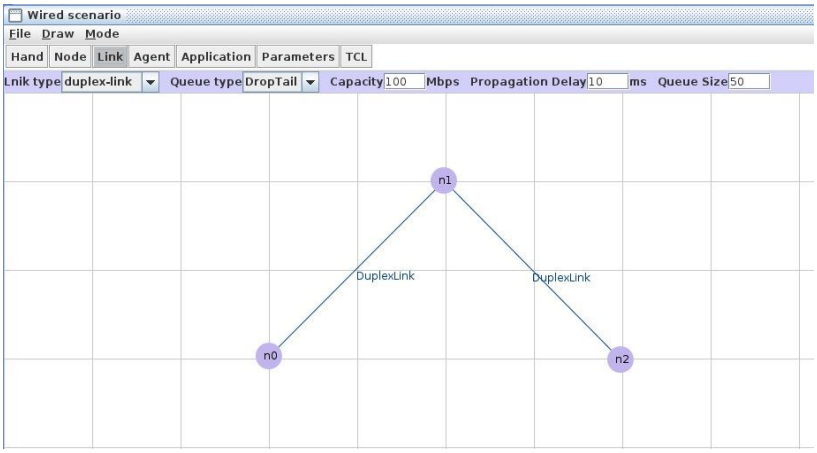
Scenario-> New wired scenario (this creates your .tcl file)

Step 3: Click **Node** and add three nodes on screen



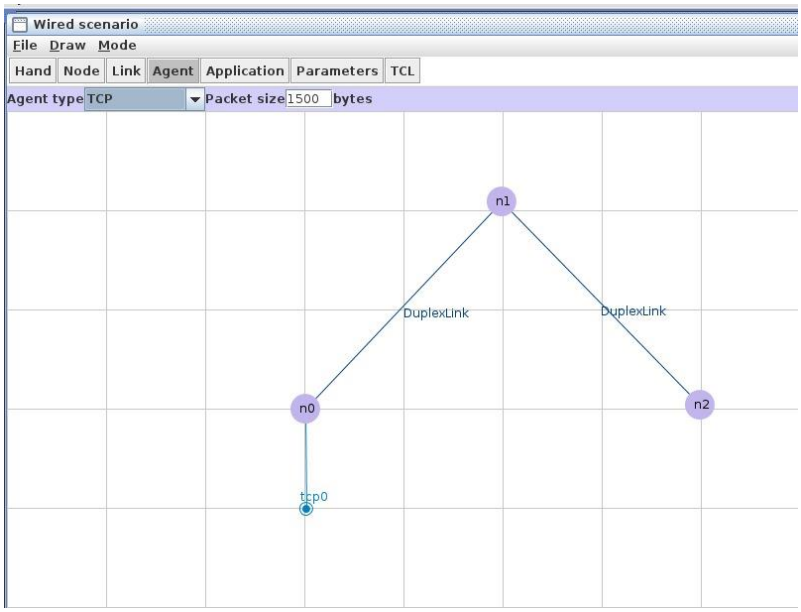
Step 4: **Link -> duplex-link**

Connect n0 -> n1 and n1->n2



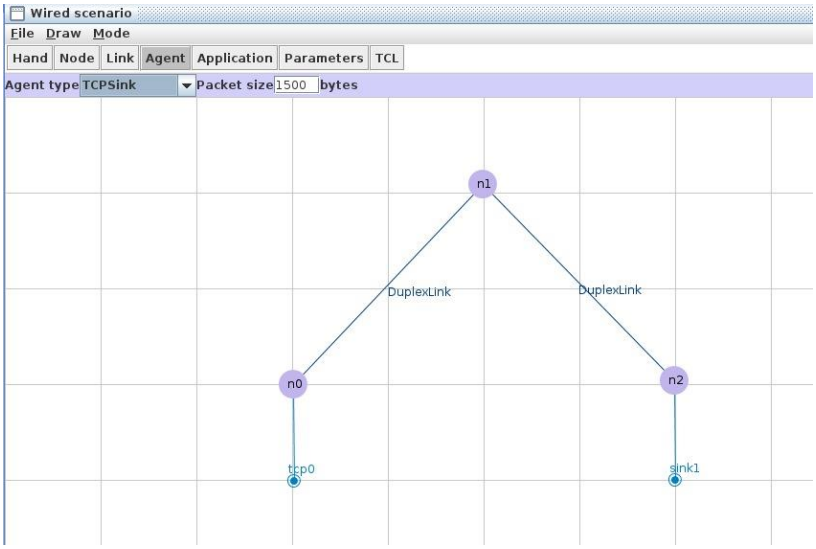
Step 5: **Agent->TCP**

Click on n0 and drag down and click



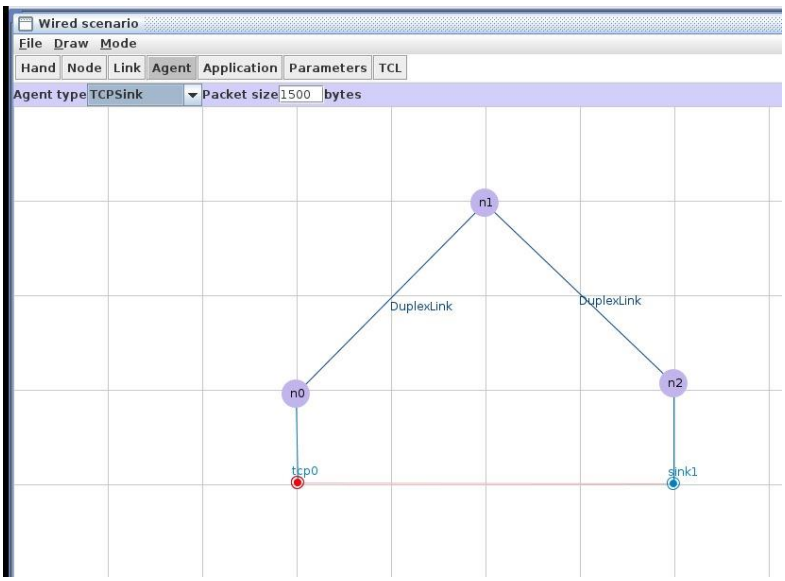
Step 6: **Agent->TCPSink**

Click on n2 and drag down and click



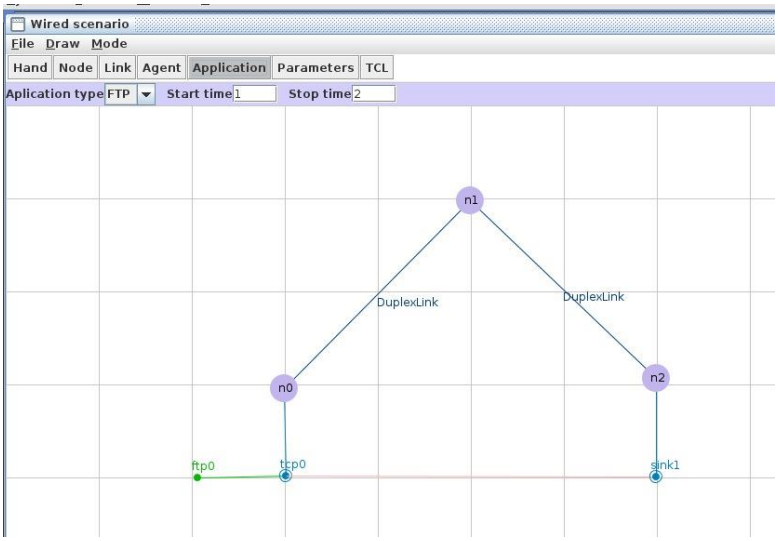
Step 7: **Agent->TCPSink**

Connect tcp0->sink1 (click on n0 first and THEN n2)

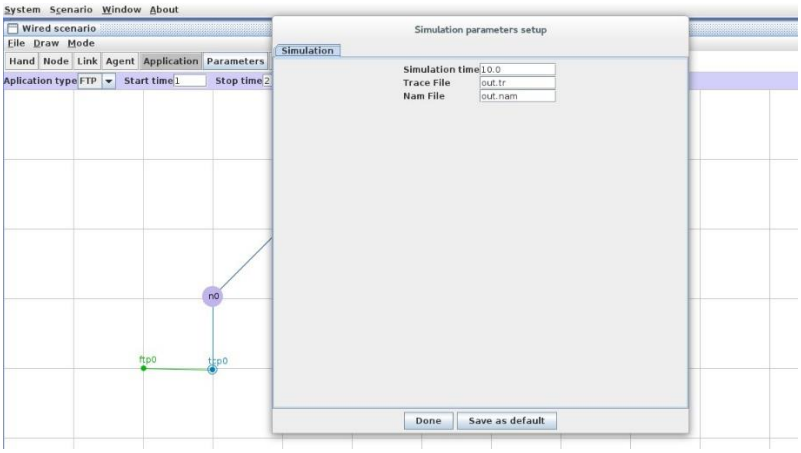


Step 8: **Application -> FTP**

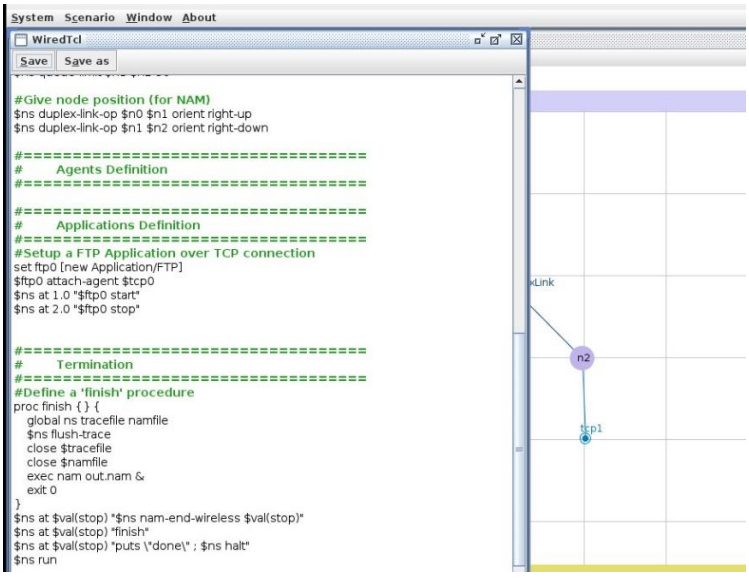
Click on tcp0 and drag to the left and click

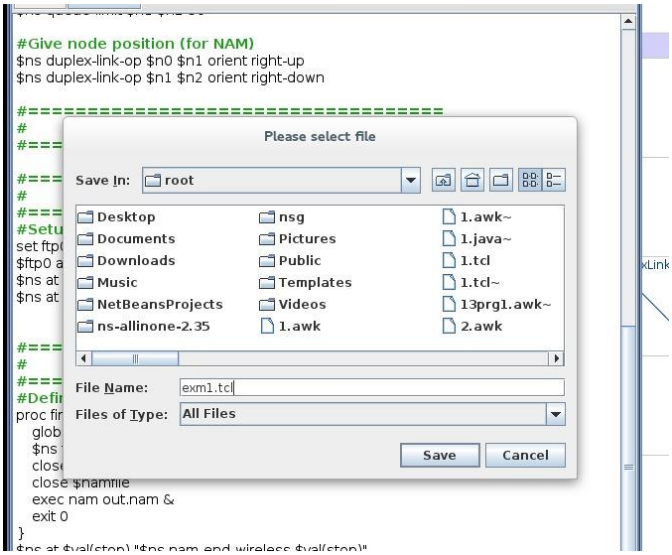


Step 9: **Parameters -> Save as default -> Done**



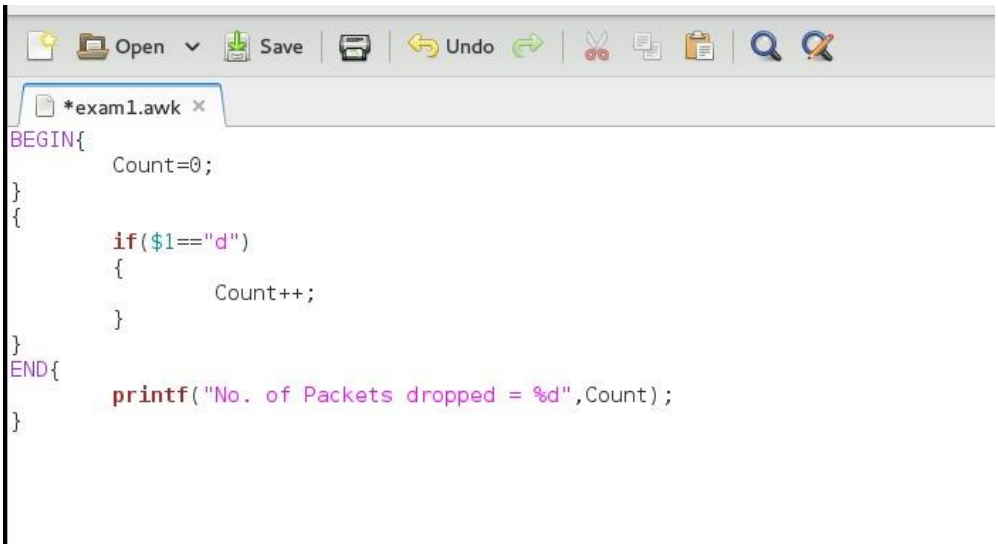
Step 10: **TCL -> Save as -> exam1.tcl -> Save**





Step 11: Close the NSG2 window. **In Terminal type the command gedit exam1.awk**

And type the following code



Save using Ctrl+S

Step 12: **In Terminal type the command gedit exam1.tcl**

Change the following sections

```
#=====================================
#        Links Definition
#=====================================
#Createlinks between nodes
$ns duplex-link $n0 $n1 100.0Mb 10ms DropTail
$ns queue-limit $n0 $n1 0
$ns duplex-link $n1 $n2 100.0Mb 10ms DropTail
$ns queue-limit $n1 $n2 0
```

<- Set queue limit as 0

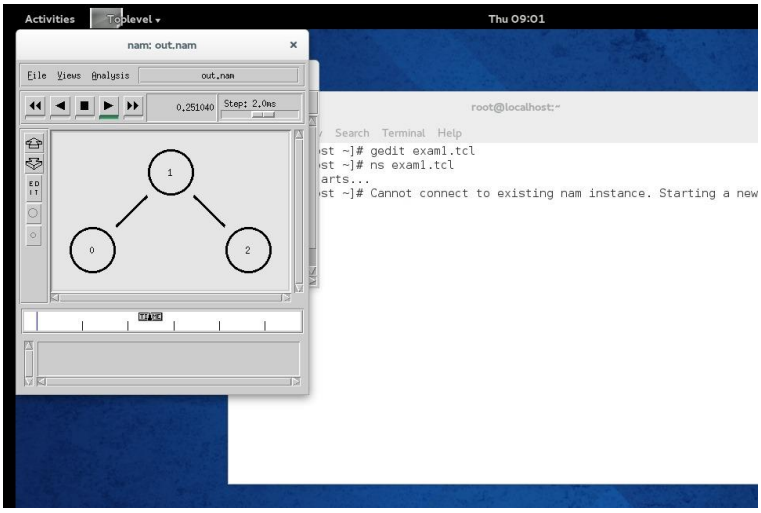
<- Set queue limit as 0

```
#====================================
#        Applications Definition
#====================================
#Setup a FTP Application over TCP connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ns at 0.5 "$ftp0 start"
$ns at 2.0 "$ftp0 stop"
$ns at 2.5 "finish"
```

Save using Ctrl+S

Step 14: **In Terminal type the command ns exam1.tcl**

Click **Play and Fast forward till the end**



Step 14: **In Terminal type the command awk -f exam1.awk out.tr**

Final output should be:

```
awk -f exam1.awk outawk -f exam1.awk out.tr
No. of Packets dropped = 1[root@localhost ~]#
```

2. Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.

Step 1: Open the Terminal in Fedora (Username: root Password: fedora)

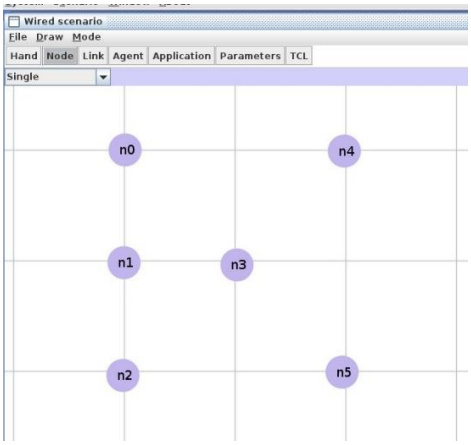
Step 2: Type the command in the Terminal

java -jar NSG2.1.jar

Step 2: In the new window

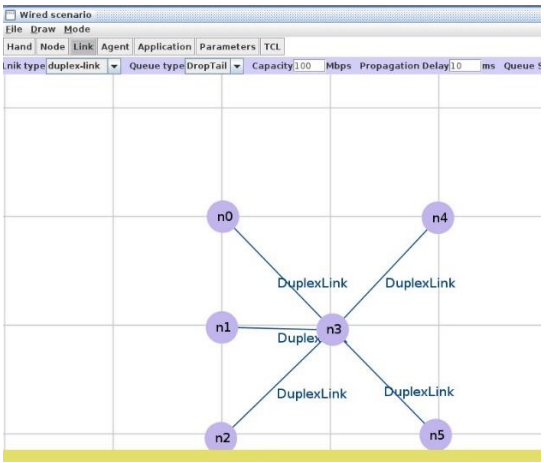
Scenario-> New wired scenario (this creates your .tcl file)

Step 3: Click **Node** and add 6 nodes on screen



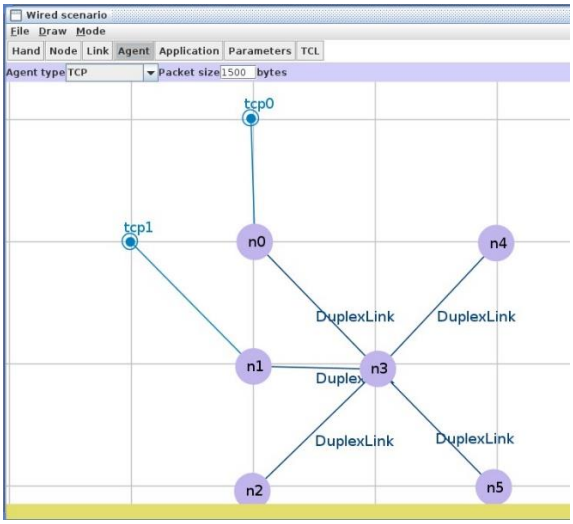
Step 4: **Link -> duplex-link**

Connect n0 -> n3 , n1 -> n3 , n2 -> n3 , n3 -> n4 , n3 -> n5



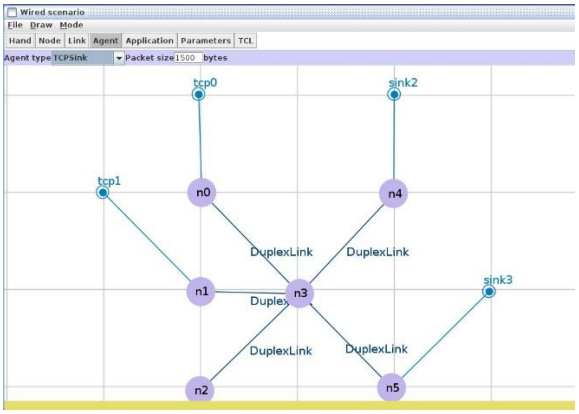
Step 5: **Agent->TCP**

Click on n0 and drag and click. Click on n1 and drag and click



Step 6: **Agent->TCPSink**

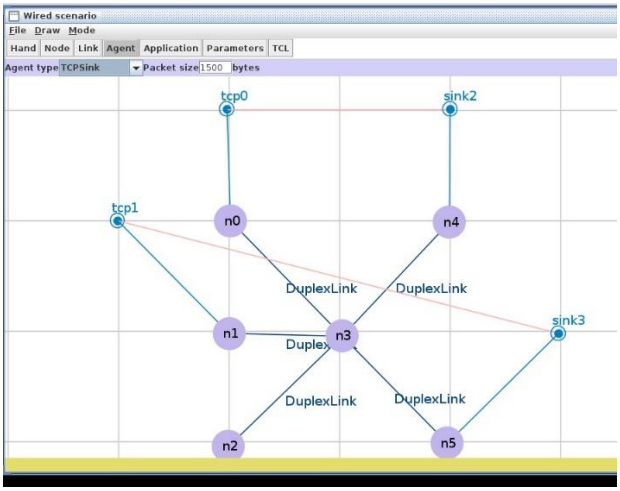
Click on n4 and drag and click. Click on n5 and drag and click.



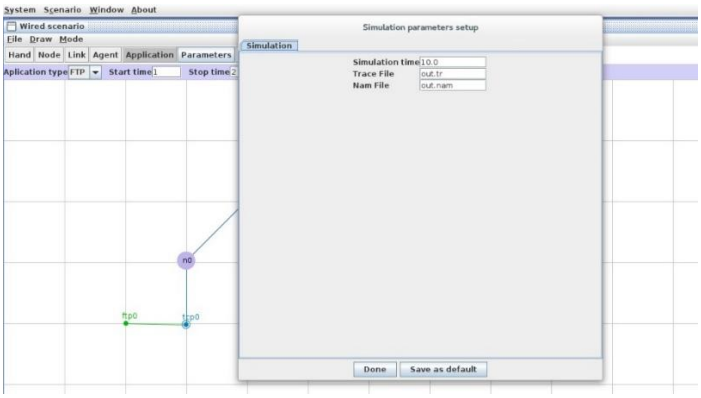
Step 7: **Agent->TCPSink**

Connect tcp0->sink2 (click on **tcp0** first and THEN **sink2**)

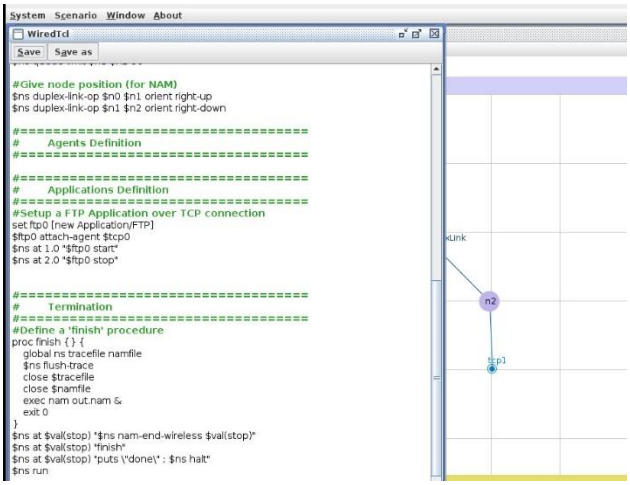
Connect tcp1->sink3 (click on **tcp1** first and THEN **sink3**)

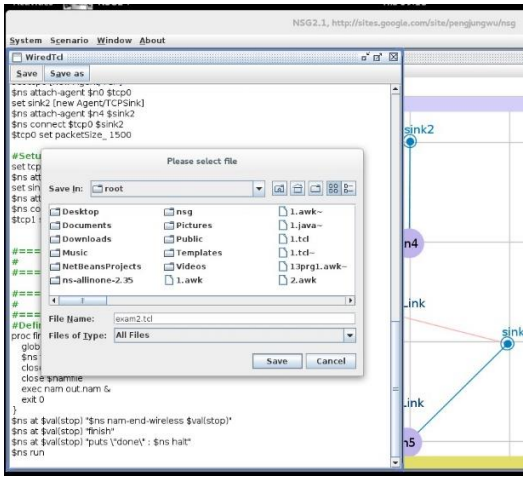


Step 8: **Parameters -> Save as default -> Done**



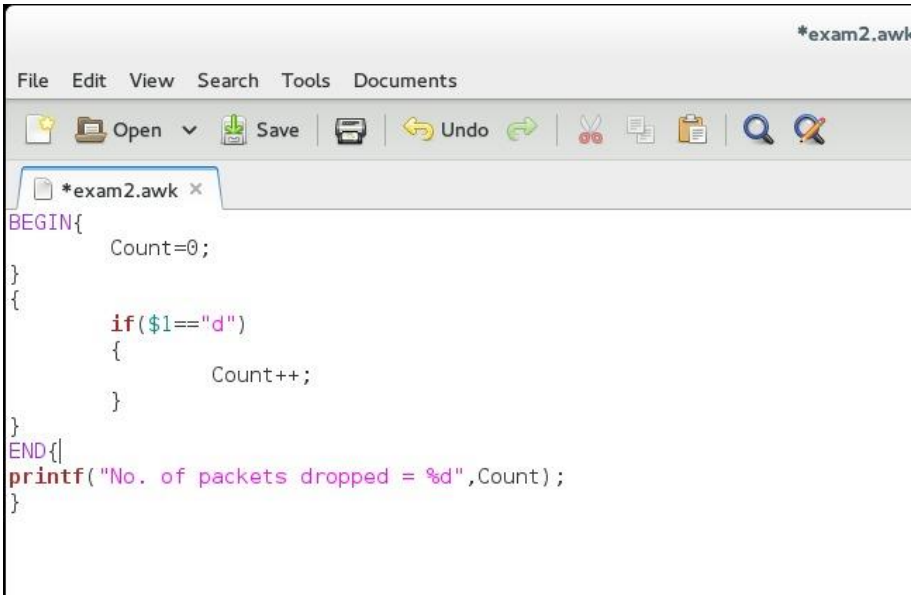
Step 9: **TCL -> Save as -> exam2.tcl -> Save**





Step 11: Close the NSG2 window. **In Terminal type the command gedit exam2.awk**

And type the following code



Save using Ctrl+S

Step 12: **In Terminal type the command gedit exam2.tcl**

Change the following sections

```
#======================================
#        Agents Definition
#======================================

Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "Node [$node_ id] recieves response from $from with RTT=$rtt ms"}

#Setup a TCP connection
set tcp0 [new Agent/Ping]
$ns attach-agent $n0 $tcp0
set sink1 [new Agent/Ping]
$ns attach-agent $n4 $sink1
```

Add those 3 lines

```
#==================================
#        Applications Definition
#==================================

$ns at 1.0 "$tcp0 send"
$ns at 1.1 "$tcp0 send"
$ns at 1.2 "$tcp0 send"
$ns at 1.3 "$tcp0 send"
$ns at 1.0 "$tcp1 send"
$ns at 1.1 "$tcp1 send"
$ns at 1.2 "$tcp1 send"
$ns at 1.3 "$tcp1 send"

#----------------------------------
```

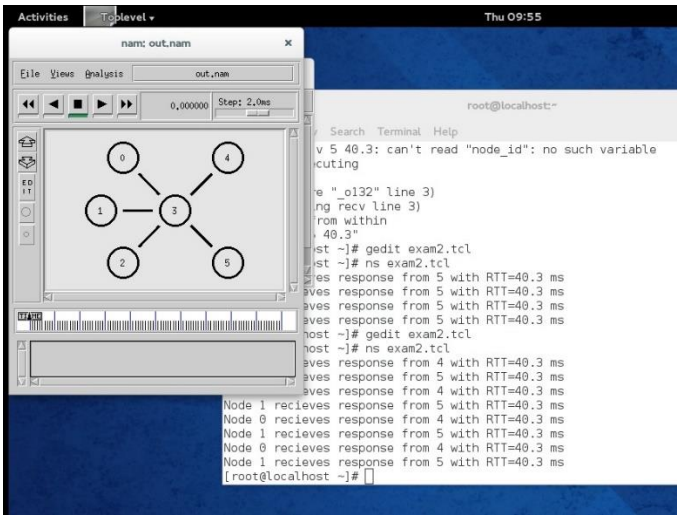
```
#==================================
#        Links Definition
#==================================
#Createlinks between nodes
$ns duplex-link $n0 $n3 100.0Mb 10ms DropTail
$ns queue-limit $n0 $n3 50
$ns duplex-link $n1 $n3 100.0Mb 10ms DropTail
$ns queue-limit $n1 $n3 50
$ns duplex-link $n2 $n3 100.0Mb 10ms DropTail
$ns queue-limit $n2 $n3 50
$ns duplex-link $n3 $n4 100.0Mb 10ms DropTail
$ns queue-limit $n3 $n4 0
$ns duplex-link $n3 $n5 100.0Mb 10ms DropTail
$ns queue-limit $n3 $n5 50
```

<=Set queue limit as 0

Save using Ctrl+S

Step 14: **In Terminal type the command ns exam2.tcl**

Click **Play and Fast forward till the end**



Step 14: **In Terminal type the command awk -f exam2.awk out.tr**

Final output should be:

```
[root@localhost ~]# Missing required flag -x in: W -t 10

Missing required flag -y in: W -t 10.0

Parsing error in event.
awk -f exam2.awk out.tr
No. of packets dropped = 4[root@localhost ~]#
```

3. Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.

Step 1: Open the Terminal in Fedora (Username: root Password: fedora)

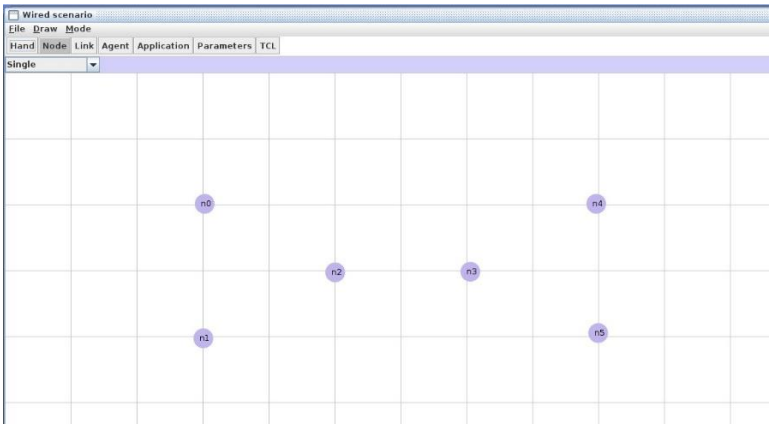
Step 2: Type the command in the Terminal

java -jar NSG2.1.jar

Step 2: In the new window

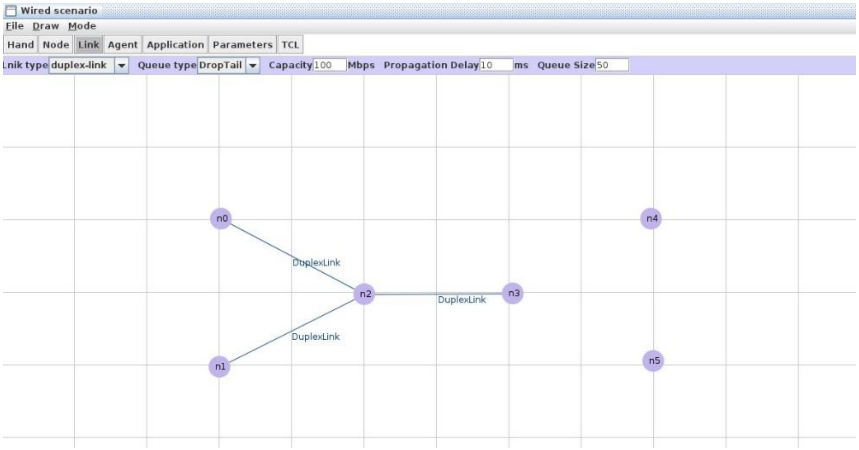
Scenario-> New wired scenario (this creates your .tcl file)

Step 3: Click **Node** and add 6 nodes on screen



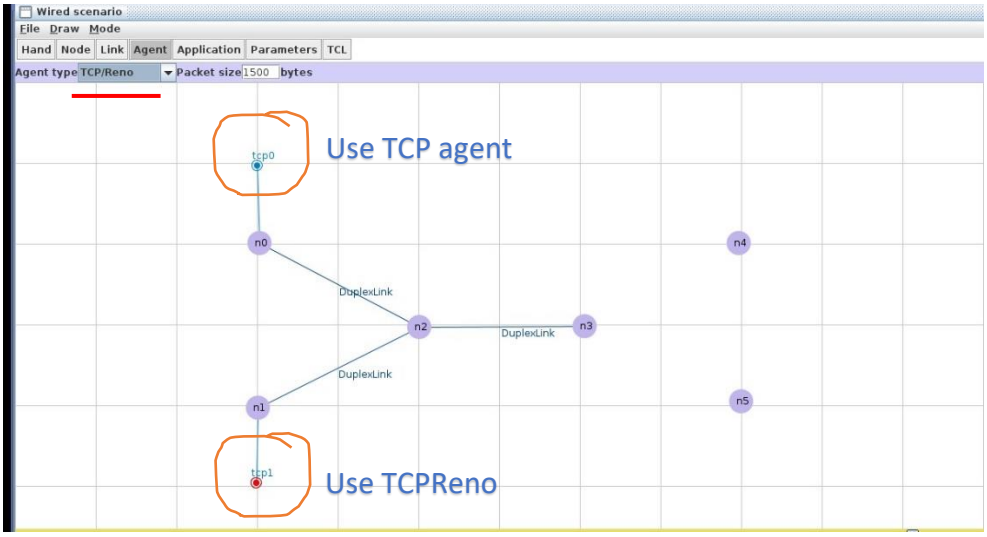
Step 4: **Link -> duplex-link**

Connect n0 -> n2 , n1 -> n2 , n2 -> n3



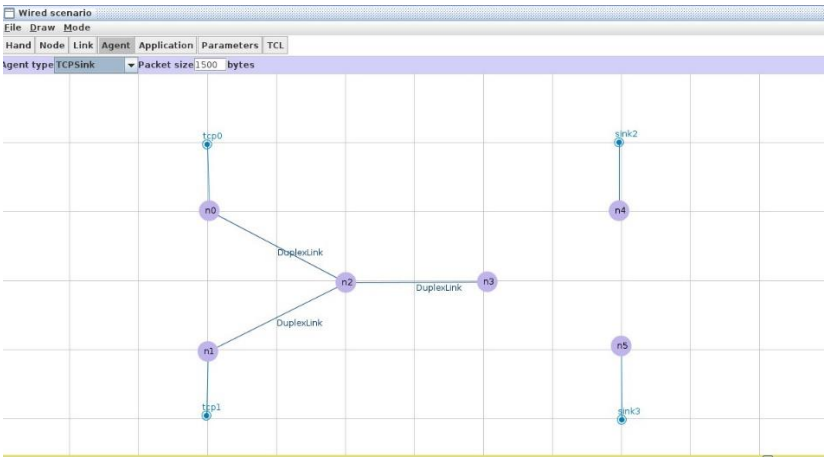
Step 5: **Agent->TCP**

Click on n0 and drag and click use TCP. Click on n1 and drag and click use TCPReno from the dropdown



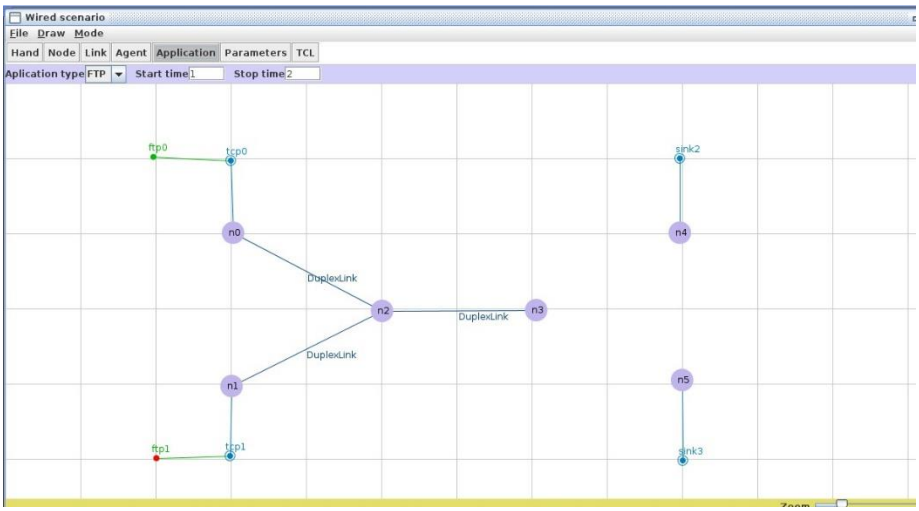
Step 6: **Agent->TCPSink**

Click on n4 and drag and click. Click on n5 and drag and click.



Step 7: **Application -> FTP**

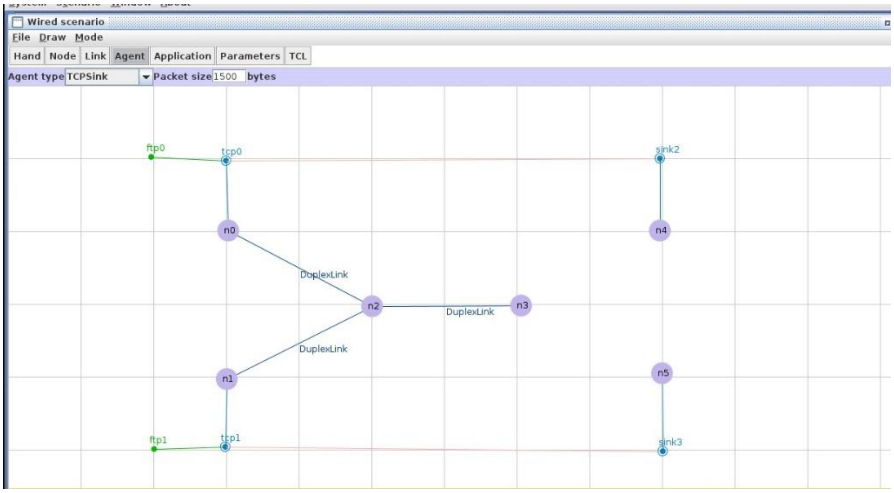
Click on tcp0 and drag to the left and click. Click on tcp1 and drag to the left and click



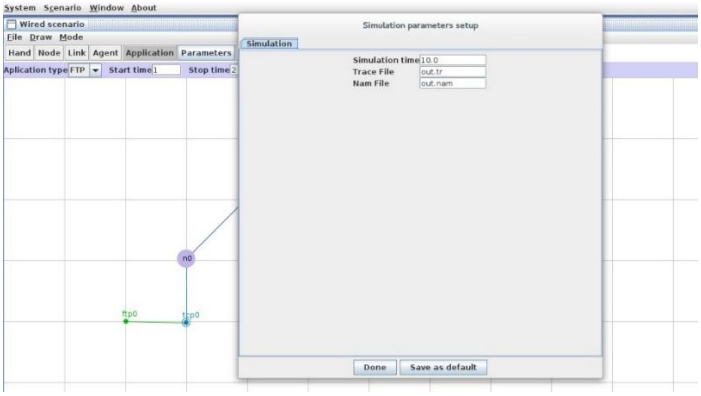
Step 8: **Agent->TCPSink**

Connect tcp0->sink2 (click on **tcp0** first and THEN **sink2**)

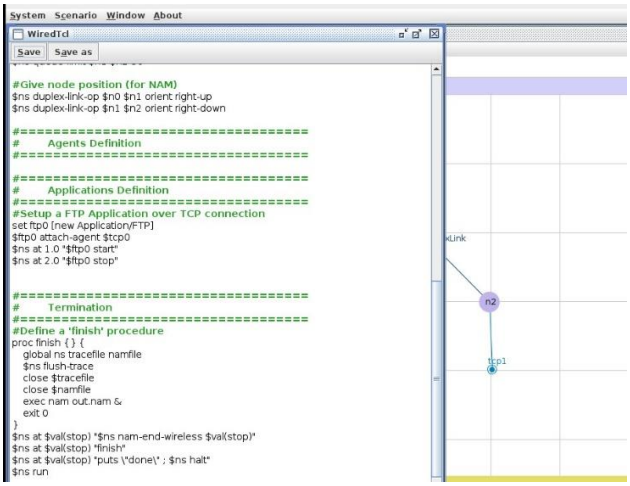
Connect tcp1->sink3 (click on **tcp1** first and THEN **sink3**)

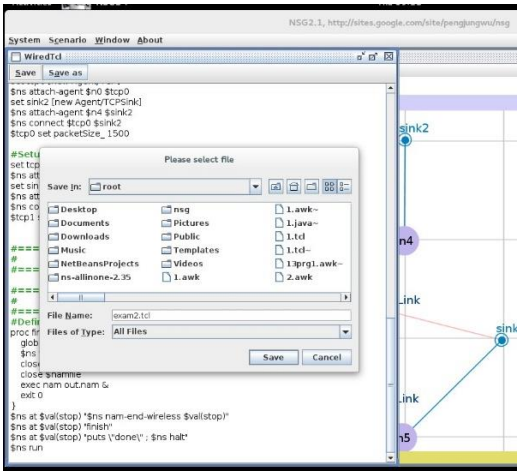


Step 8: **Parameters -> Save as default -> Done**



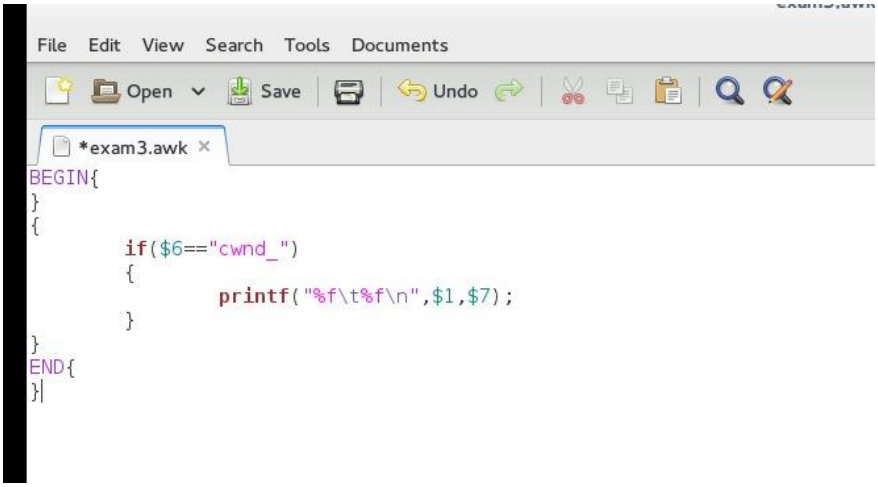
Step 9: **TCL -> Save as -> exam3.tcl -> Save**





Step 11: Close the NSG2 window. **In Terminal type the command gedit exam3.awk**

And type the following code



Save using Ctrl+S

Step 12: **In Terminal type the command gedit exam3.tcl**

Change the following sections

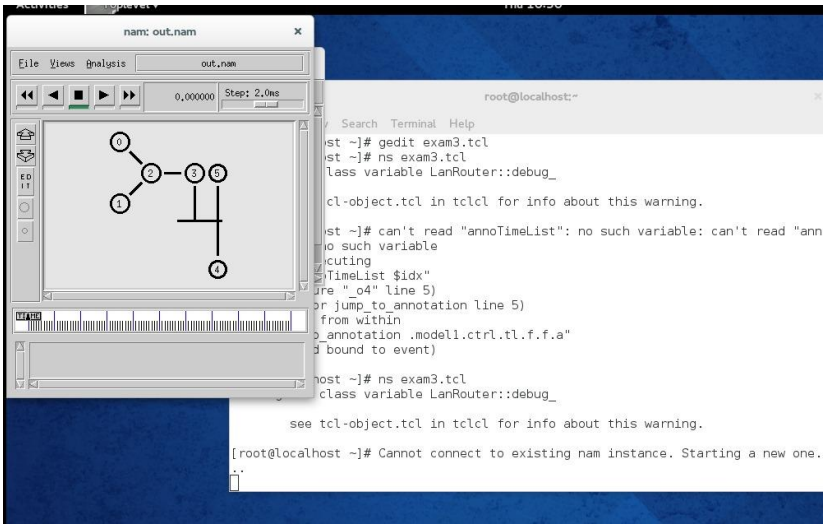
```
#===================================
#        Links Definition
#===================================
#Createlinks between nodes
$ns duplex-link $n0 $n2 100.0Mb 10ms DropTail
$ns queue-limit $n0 $n2 50
$ns duplex-link $n1 $n2 100.0Mb 10ms DropTail
$ns queue-limit $n1 $n2 50
$ns duplex-link $n2 $n3 100.0Mb 10ms DropTail
$ns queue-limit $n2 $n3 50

set lan [$ns newLan "$n3 $n4 $n5" 1mb 40ms LLQueue/DropTail Mac/802 3 channel]

#Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
```

```
#===================================
#        Applications Definition
#===================================
#Setup a FTP Application over TCP connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ns at 1.0 "$ftp0 start"
$ns at 2.0 "$ftp0 stop"

#Setup a FTP Application over TCP/Reno connection
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns at 1.0 "$ftp1 start"
$ns at 2.0 "$ftp1 stop"

set f1 [open f1.tr w]
$tcp0 attach $f1
set f2 [open f2.tr w]
$tcp1 attach $f2

$tcp0 trace cwnd_
$tcp1 trace cwnd_
```

Save using Ctrl+S

Step 14: **In Terminal type the command ns exam3.tcl**

Click **Play and Fast forward till the end**



Step 14: **In Terminal type the commands**

awk -f exam2.awk f1.tr

awk -f exam2.awk f2.tr

awk -f exam2.awk f1.tr>a

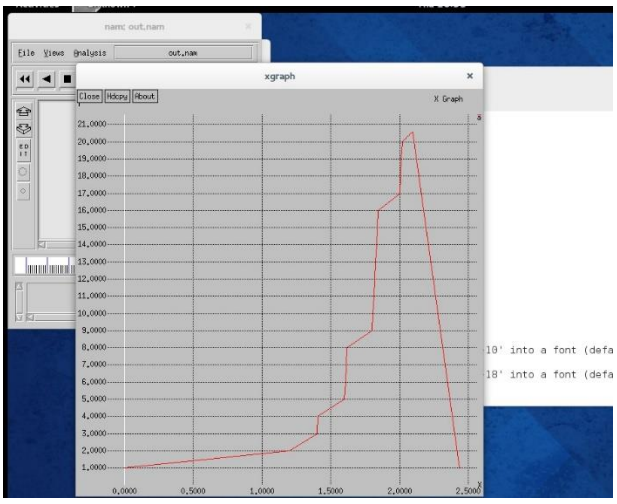
awk -f exam2.awk f2.tr>b

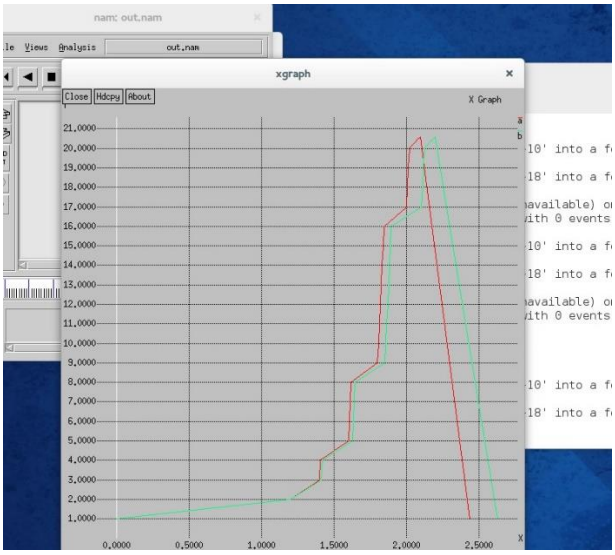
Step 15:To get the graphs **In Terminal type the commands**

xgraph a

xgraph b

xgraph a b





4.Develop a program for error detecting code using CRC-CCITT (16- bits).

```
def xor(cs, g):
    return [(cs[i] ^ g[i]) for i in range(len(g))]

def crc(data, g):
    cs = data[:len(g)]
    for i in range(len(data) - len(g) + 1):
        if cs[0] == 1:
            cs = xor(cs, g)
        cs = cs[1:] + [data[i + len(g)] if i + len(g) < len(data) else 0]
    return cs[:-1]

def main():
    data = list(map(int, input("Enter the data bits: ").split()))
    g = [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1]
    data += [0] * (len(g) - 1)
    checksum = crc(data, g)
    codeword = data[:len(data) - len(g) + 1] + checksum
    print("Final Codeword:", codeword)

    if int(input("Test Error detection 0(yes) 1(no) ?: ")) == 0:
        pos = int(input("Enter position where error is to be inserted: "))
        codeword[pos] ^= 1
        print("Erroneous data:", codeword)

    if any(crc(codeword, g)):
        print("ERROR in Received Codeword")
    else:
        print("No Error in Received Codeword")
```

```
if __name__ == "__main__":
    main()
```

Input:

1. **Enter the data bits: 1001**

   Test Error detection 0(yes) 1(no) ?: 0

   Enter position where error is to be inserted: 3

2. **Enter the data bits: 1001**

   Test Error detection 0(yes) 1(no) ?: 1

Output:

1. Enter the data bits: 1001

Final Codeword: [1001, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Enter position where error is to be inserted: 3

Erroneous data: [1001, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

ERROR in Received Codeword

2. Enter the data bits: 1001

Final Codeword: [1001, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Test Error detection 0(yes) 1(no) ?: 1

No Error in Received Codeword

5.Develop a program to implement a sliding window protocol in the data link layer.

```
import random
import time

# Initialize variables
window_size = int(input("Enter the window size: "))
total_frames = int(input("Enter the total number of frames to be sent: "))
send_base = 0
next_seq_num = 0
acknowledged = [False] * total_frames


# Function to simulate sending a frame
def send_frame(frame):
    print(f"Sending frame {frame}")
```

```
# Function to simulate receiving an acknowledgment with random loss
def receive_ack():
    # 90% chance of successful acknowledgment
    if random.random() < 0.9:
        ack = random.randint(send_base, min(send_base + window_size - 1, total_frames
- 1))
        print(f"Acknowledgment received for frame {ack}")
        return ack
    print("Acknowledgment lost!")
    return -1  # No acknowledgment received


# Function to slide the window when an ACK is received
def slide_window(ack):
    global send_base
    while send_base <= ack < total_frames:
        acknowledged[send_base] = True
        print(f"Frame {send_base} acknowledged.")
        send_base += 1


# Main function to run the sliding window protocol
def run_sliding_window():
    global next_seq_num
    while send_base < total_frames:
        # Send frames within the window
        while next_seq_num < send_base + window_size and next_seq_num < total_frames:
            send_frame(next_seq_num)
            next_seq_num += 1

        # Simulate receiving an acknowledgment
        ack = receive_ack()
        if ack != -1:
            slide_window(ack)
        else:
            print("Timeout! Resending frames...")
            next_seq_num = send_base  # Restart sending from the unacknowledged base
frame

# Run the protocol
run_sliding_window()
```

Input:

Enter the window size: 4

Enter the total number of frames to be sent: 10

Output:

Sending frame 0

Sending frame 1

Sending frame 2

Sending frame 3

Acknowledgment received for frame 0

Frame 0 acknowledged.

Sending frame 4

Acknowledgment received for frame 1

Frame 1 acknowledged.

Sending frame 5

Acknowledgment received for frame 5

Frame 2 acknowledged.

Frame 3 acknowledged.

Frame 4 acknowledged.

Frame 5 acknowledged.

Sending frame 6

Sending frame 7

Sending frame 8

Sending frame 9

Acknowledgment received for frame 7

Frame 6 acknowledged.

Frame 7 acknowledged.

Acknowledgment received for frame 9

Frame 8 acknowledged.

Frame 9 acknowledged.

6. Develop a program to find the shortest path between vertices using the Bellman-Ford and path vector routing algorithm.

```
class Graph:
    def __init__(self, vertices):
```

```
        self.vertices = vertices
        self.edges = []

    def add_edge(self, u, v, weight):
        self.edges.append((u, v, weight))

    def bellman_ford(self, src):
        distance = [float('inf')] * self.vertices
        distance[src] = 0

        for _ in range(self.vertices - 1):
            for u, v, weight in self.edges:
                if distance[u] != float('inf') and distance[u] + weight < distance[v]:
                    distance[v] = distance[u] + weight

        for u, v, weight in self.edges:
            if distance[u] != float('inf') and distance[u] + weight < distance[v]:
                print("Graph contains a negative-weight cycle")
                return None
        return distance

# Get user input
vertices = int(input("Enter number of vertices: "))
edges = int(input("Enter number of edges: "))
g = Graph(vertices)

print("Enter edges as: start end weight")
for _ in range(edges):
    u, v, weight = map(int, input().split())
    g.add_edge(u, v, weight)

source_vertex = int(input("Enter source vertex: "))
distances = g.bellman_ford(source_vertex)

if distances:
    print("Vertex Distance from Source")
    for i in range(vertices):
        print(f"{i}\t\t{distances[i]}")
```

Input:

Enter number of vertices: 5

Enter number of edges: 8

Enter edges as: start end weight

0 1 -1

0 2 4

1 2 3

1 3 2

3 1 1

4 3 -3

Enter source vertex: 0

Output:

Vertex Distance from Source

0	**0**
1	**-1**
2	**2**
3	**-2**
4	**1**

7. Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.

Server Side:

```
import socket

def main():
    host, port = '127.0.0.1', 4000

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
        server.bind((host, port))
        server.listen(1)
        print("Server ready for connection")

        conn, addr = server.accept()  # Accept the connection
        print("Connection successful, waiting for chatting")
        try:
            filename = conn.recv(1024).decode()  # Receive filename
            with open(filename, 'r') as file:
                for line in file:
                    conn.sendall(line.encode())  # Send file contents
            print("File contents sent.")
        except FileNotFoundError:
            conn.sendall(b"File not found.")
        finally:
```

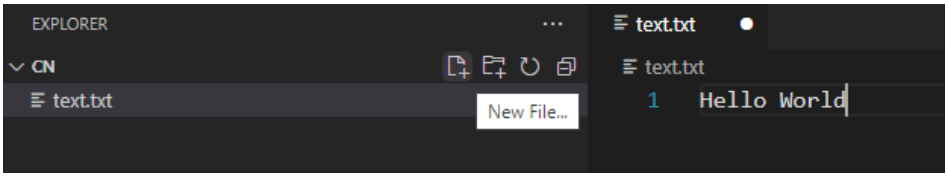
```
            conn.close()  # Ensure the connection is closed

if __name__ == "__main__":
    main()
```

Client Side:

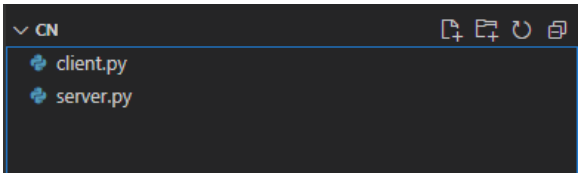
```
import socket

def main():
    host = '127.0.0.1'  # Server's IP address
    port = 4000

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
        client_socket.connect((host, port))

        # Sending the filename to the server
        filename = input("Enter the file name: ")
        client_socket.sendall(filename.encode())

        # Receiving and displaying the contents of the file from the server
        print("Contents of the File:")
        while True:
            data = client_socket.recv(1024).decode()
            if not data:
                break
            print(data, end="")

if __name__ == "__main__":
    main()
```

Input:

1.Make a text file with text content like "Hello World"



2.Make two python files server and client



3.Run server side first

>> Server ready for connection

>>Connection successful, waiting for chatting

4.Run the client side

>> Enter the file name: text.txt

Client output:

>> Contents of the File:

>>Hello World

Server output:

>> File contents sent.

8. Develop a program on a datagram socket for client/server to display the messages on client side, typed at the server side.

Server Side:

```
import socket


udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
try:
    udp_socket.bind(('', 21))  # '' means it listens on all network interfaces
except PermissionError:
    print("Permission denied. Try using a port above 1024 (e.g., 9876).")
    exit(1)

print("Waiting for a message from the server...")

# Receive data from the server
buffer_size = 1024
data, server_address = udp_socket.recvfrom(buffer_size)

# Decode and print the received message
message = data.decode('utf-8')
print("Message from Server:")
print(message)

udp_socket.close()
```

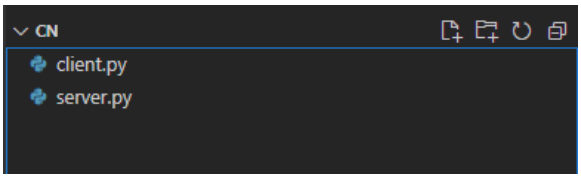
Client Side:

```
import socket
```

```python
udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
message = input("Enter the message and press ENTER to send: ")
server_address = ("127.0.0.1", 21)

try:
    # Send the message
    udp_socket.sendto(message.encode(), server_address)
    print("Message sent!")
finally:
    # Close the socket
    udp_socket.close()
```

Input:

1.Make two python files server and client



2.Run server side first

>> Waiting for a message from the server...

3.Run the client side

>> Enter the message and press ENTER to send: Hi

Client output:

>> Message sent!

Server output:

>> Message from Server:

>>Hi

9. Develop a program for a simple RSA algorithm to encrypt and decrypt the data.

```python
def mod_exp(base, exp, mod):
    result = 1
    base %= mod
    while exp > 0:
        if exp % 2:
            result = (result * base) % mod
        exp //= 2
```

```
        base = (base * base) % mod
    return result

def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def find_d(e, Z):
    d = 1
    while (e * d) % Z != 1:
        d += 1
    return d

def main():
    p = int(input("Enter a prime number p: "))
    q = int(input("Enter a prime number q: "))
    n = p * q
    Z = (p - 1) * (q - 1)
    e = int(input("Enter an integer e (1 < e < Z, coprime with Z): "))
    while gcd(e, Z) != 1:
        e = int(input("Invalid e. Enter again: "))

    d = find_d(e, Z)

    message = input("Enter message to encrypt: ")
    pt = [ord(c) for c in message]
    ct = [mod_exp(c, e, n) for c in pt]
    print(f"p = {p}, q = {q}, n = {n}, Z = {Z}")
    print(f"Public key: ({e}, {n}), Private key: ({d}, {n})")
    print("Cipher Text:", ct)

    decrypted_message = ''.join(chr(mod_exp(c, d, n)) for c in ct)
    print("Decrypted Text:", decrypted_message)

if __name__ == "__main__":
    main()
```

Input:

>>Enter prime p: 13

>>Enter prime q: 29

>>Enter e (1 < e < Z, coprime with Z): 31

>>Enter message to encrypt: Hello World

Output:

p = 13, q = 29, n = 377, Z = 336

Public key: (31, 377), Private key: (271, 377)

Cipher Text: [279, 192, 329, 329, 136, 85, 87, 136, 166, 329, 22]

Decrypted Text: Hello World

10. Develop a program for congestion control using a leaky bucket algorithm.

```
def leaky_bucket():
    size = int(input("Enter the bucket size: "))
    nop = int(input("Enter the number of packets: "))

    datarate = []
    print("Enter the data rate for each packet:")
    for i in range(nop):
        datarate.append(int(input()))

    opr = int(input("Enter the output rate: "))

    for rate in datarate:
        if rate > size:
            print("Bucket overflow")
        else:
            temp = rate
            while temp > opr:
                print("Packet transmission:", opr)
                temp -= opr
            print("Packet transmission:", temp)

# Call the function to execute the program
leaky_bucket()
```

Input:

Enter the bucket size: 50

Enter the number of packets: 3

Enter the data rate for each packet:

35

76

10

Enter the output rate: 10

Output:

Packet transmission: 10

Packet transmission: 10

Packet transmission: 10

Packet transmission: 5

Bucket overflow

Packet transmission: 10