Computer Networks Assignment-04

NS3 BASED SIMULATION OF A COMPUTER NETWORK

Abhay Dagar
2022014

Aditya Sharma
2022038

INTRODUCTION

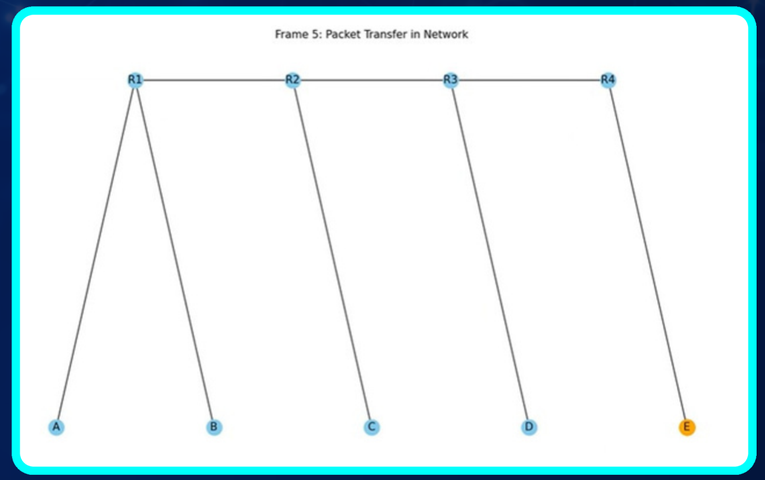
Goal

To simulate a network topology with given routers and workstations.

Visuals

Simple Topology diagram with:

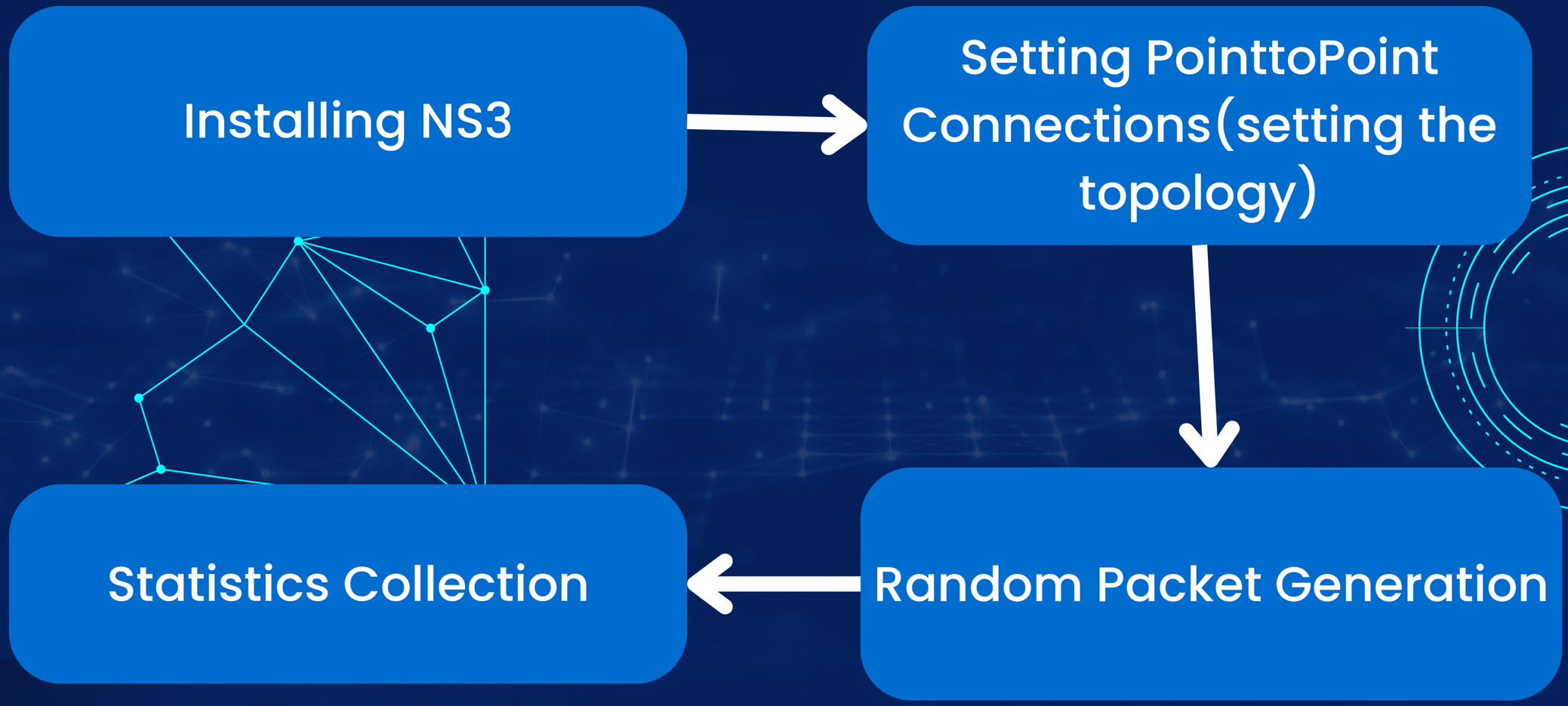
- R1,R2R3,R4 => Routers
- A,B,CD,E,=> Workstations



Assumptions

- UDP sockets are used with no guaranteed delivery, and sender/receiver are set to localhost with fixed IPs and ports.
- Packet generation and transmission delays are uniformly distributed within specified intervals (T1-T4). A finite number of packets are simulated.
- Fixed window size (7 frames) with modulo-8 sequence numbering is used.
- Timeout intervals are constant and not adaptive.
- Controlled Environment: Only a single sender-receiver pair is assumed, with simplified, mock data to focus on transmission and sequencing rather than content.

SYSTEM FLOW


Installing NS3
Setting PointtoPoint Connections(setting the topology)
Random Packet Generation
Statistics Collection

Network Parameters

```
// Create node containers for routers and workstations
NodeContainer routers, workstations;
routers.Create(4);        // 4 Routers
workstations.Create(5);   // 5 Workstations
```

In these lines, we are creating two containers of nodes using NodeContainer:

- routers: A container that holds 4 router nodes, representing the core network infrastructure.
- workstations: A container that holds 5 workstation nodes, acting as sources or destinations for the network traffic.

```
// Introduce an error model to simulate packet drops
Ptr<RateErrorModel> errorModel = CreateObject<RateErrorModel>();
errorModel->SetAttribute("ErrorRate", DoubleValue(0.01));
p2p.SetDeviceAttribute("ReceiveErrorModel", PointerValue(errorModel));
```

- FlowMonitorHelper flowmonHelper; creates an instance of the FlowMonitorHelper class, which is used to configure and manage flow monitoring for the network simulation.
- Ptr<FlowMonitor> monitor = flowmonHelper.InstallAll(); installs the FlowMonitor on all nodes and devices in the network, enabling the collection of flow-related statistics such as packet counts, delays, and packet drops for all traffic flows in the simulation. The monitor pointer holds the handle to the collected flow data.

Key Parameters

```
// Set up queue disciplines for the core router links
TrafficControlHelper tch;
tch.SetRootQueueDisc("ns3::RedQueueDisc");
```

The TrafficControlHelper class is used to manage traffic control settings on network devices. By calling tch.SetRootQueueDisc("ns3::RedQueueDisc");, we configure the routers to use Random Early Detection (RED), a congestion control algorithm. RED helps prevent buffer overflow by selectively dropping packets before the queue is full, simulating more realistic network behavior under varying traffic loads.

```
// Assign IP addresses to all devices
Ipv4AddressHelper ipv4;
ipv4.SetBase("10.1.1.0", "255.255.255.0");
ipv4.Assign(d_r1r2);
ipv4.Assign(d_r2r3);
ipv4.Assign(d_r3r4);
ipv4.Assign(d_r1r4);
ipv4.Assign(d_r1a);
ipv4.Assign(d_r1b);
ipv4.Assign(d_r2c);
ipv4.Assign(d_r3d);
ipv4.Assign(d_r4e);
```

Ipv4AddressHelper ipv4; creates an instance for assigning IPv4 addresses. ipv4.SetBase("10.1.1.0", "255.255.255.0"); sets the base IP range and subnet mask. The subsequent ipv4.Assign() calls assign IP addresses to devices on the specified links, enabling communication between routers and workstations. This ensures each device has a unique IP address within the defined subnet.

Protocol Operation

```
// Generate random traffic between workstations
GenerateRandomTraffic(workstations);
```

```
void GenerateRandomTraffic(NodeContainer workstations) {
    uint16_t port = 9;

    Ptr<UniformRandomVariable> randStart = CreateObject<UniformRandomVariable>();
    Ptr<UniformRandomVariable> randDest = CreateObject<UniformRandomVariable>();

    for (uint32_t i = 0; i < workstations.GetN(); ++i) {
        Ptr<Node> src = workstations.Get(i);

        Ptr<Node> dst;
        do {
            dst = workstations.Get(randDest->GetValue(0, workstations.GetN() - 1));
        } while (src == dst);

        Ipv4Address dstAddress = dst->GetObject<Ipv4>()->GetAddress(1, 0).GetLocal();

        double startTime = randStart->GetValue(1, 10);
        double stopTime = startTime + randStart->GetValue(1, 5);

        CreateTraffic(src, dstAddress, port, DataRate("10Mbps"), 4096, startTime, stopTime);
    }
}
```

In this function, GenerateRandomTraffic is used to create random traffic between workstations in the network:

- Port Setup: A fixed port number (9) is used for all traffic.
- Random Start and Destination:
  - randStart and randDest are UniformRandomVariable objects used to generate random start times and destination nodes for traffic.
  - For each workstation, a random destination is chosen, ensuring it's not the same as the source.
- Traffic Generation:
  - The CreateTraffic function is called for each source-destination pair, setting up UDP traffic with:
    - A data rate of 10 Mbps.
    - A packet size of 4096 bits.
    - Random start and stop times.

models random communication between workstations, with traffic starting at random times and varying durations.

Protocol Operation (Receiver)

```
[0/2] Re-checking globbed directories...
[2/2] Linking CXX executable ../build/scratch/ns3.43-assign-default
Average Delay Matrix (ns):
0       4       7       8       6
4       0       6       7       3
10      2       0       3       8
1       10      4       0       7
1       7       3       7       0
```

(A) End-to-end one-way delays – average have been tabulated in the form of a source-destination matrix,

```
Packet Drop Matrix:
0       0       0       1377    0
0       0       0       2209    0
1134    0       0       0       0
2675    0       0       0       0
0       917     0       0       0
```

(B) Packet drops – to be tabulated in the form of a source-destination matrix.

Diagonal Entries signify sending packet to same workstation that is not allowed

```
Queue Length Statistics (Routers):
Router 0 -> Device 1:
        Packets Currently in Queue: 1
        Bytes Currently in Queue: 1000 bytes
        Packets Dropped: 0
Router 0 -> Device 2:
        Packets Currently in Queue: 2
        Bytes Currently in Queue: 2000 bytes
        Packets Dropped: 1
Router 0 -> Device 3:
        Packets Currently in Queue: 2
        Bytes Currently in Queue: 2000 bytes
        Packets Dropped: 0
Router 0 -> Device 4:
        Packets Currently in Queue: 2
        Bytes Currently in Queue: 2000 bytes
        Packets Dropped: 1
Router 1 -> Device 1:
        Packets Currently in Queue: 2
        Bytes Currently in Queue: 2000 bytes
        Packets Dropped: 1
Router 1 -> Device 2:
        Packets Currently in Queue: 4
        Bytes Currently in Queue: 4000 bytes
        Packets Dropped: 0
Router 1 -> Device 3:
        Packets Currently in Queue: 2
        Bytes Currently in Queue: 2000 bytes
        Packets Dropped: 0
Router 2 -> Device 1:
        Packets Currently in Queue: 4
        Bytes Currently in Queue: 4000 bytes
        Packets Dropped: 1
Router 2 -> Device 2:
        Packets Currently in Queue: 3
        Bytes Currently in Queue: 3000 bytes
        Packets Dropped: 0
Router 2 -> Device 3:
        Packets Currently in Queue: 1
        Bytes Currently in Queue: 1000 bytes
        Packets Dropped: 0
Router 3 -> Device 1:
        Packets Currently in Queue: 4
        Bytes Currently in Queue: 4000 bytes
        Packets Dropped: 1
Router 3 -> Device 2:
        Packets Currently in Queue: 1
        Bytes Currently in Queue: 1000 bytes
        Packets Dropped: 1
Router 3 -> Device 3:
        Packets Currently in Queue: 4
        Bytes Currently in Queue: 4000 bytes
        Packets Dropped: 1
```

Network Simulation

In this code, queue disciplines are configured using NS-3's TrafficControlHelper class, with Random Early Detection (RED) applied to manage congestion on the router links. RED helps prevent buffer overflow by selectively dropping packets before the queue becomes full.

The PrintQueueStatistics function collects and displays important queue data, including:

- Packets in Queue (GetNPackets()),
- Queue Size in bytes (GetNBytes()),
- Packets Dropped (GetDropCount()).

These statistics provide insights into network performance, showing how NS-3 manages traffic congestion at the router level and how effectively RED controls queue behavior by dropping packets under high load.

Statistics Collection

```
abhaydagar@abhaydagar-VirtualBox:~/Desktop/ns-allinone-3.43/ns-3.43$ ./ns3 run scratch/assign.cc
[0/2] Re-checking globbed directories...
[2/2] Linking CXX executable ../build/scratch/ns3.43-assign-default
Sent Packets: 8694
Received Packets: 5509
Dropped packets: 318
Reason codes:
Reason Code 0: 100
Reason Code 1: 122
Reason Code 2: 4
Reason Code 3: 92
```

Codes and their reason for dropping

0 => Packet dropped due to a transmission error or collision.
1 => Packet dropped due to a queue overflow.
2 => Packet dropped due to the TTL (Time-to-Live) expiring.
3 => Packet dropped because it did not match any forwarding rules (e.g., no route found).
4 => Packet dropped due to specific protocol-related issues or constraints.

THANK YOU FOR YOUR ATTENTION

Abhay Dagar 2022014
Aditya Sharma 2022038