Detailed Explanation of NS-3 Code for Traffic Matrix and Routing Simulation

Palak Bhardwaj (2022344) and Yashovardhan Singhal (2022591)

29th Nov 2024

1 Overview

This code is a network simulation implemented in **NS-3** that involves the generation of a traffic matrix, IP-to-node mappings, and the configuration of routing tables. The network includes multiple hosts and routers connected by point-to-point links. The simulation integrates error models and traffic generation to analyze network behavior.

2 Code Breakdown

2.1 Header Files

The code includes several NS-3 modules and standard C++ libraries:

- `core-module`, `network-module`, `point-to-point-module`, etc., provide core functionalities for creating and managing network simulations.
- Standard libraries like `<map>`, `<utility>`, `<fstream>` are used for data storage and file operations.

2.1.1 Network Topology

The network consists of 7 hosts (A-G) and 4 routers (R1-R4). Nodes are created and the Internet stack is installed:

```
NodeContainer hosts;
hosts.Create(7);
NodeContainer routers;
routers.Create(4);

InternetStackHelper stack;
stack.Install(hosts);
stack.Install(routers);
```

2.2 Traffic Matrix and IP-to-Node Mapping

```
std::map<std::pair<std::string, std::string>, uint32_t>
    trafficMatrix;
std::map<Ipv4Address, std::string> ipToNodeName;
```

- `trafficMatrix`: Stores traffic load between nodes.
- `ipToNodeName`: Maps IP addresses to node names for easy identification.

2.3 Flow Monitor Module in NS3

The `FlowMonitor` module in NS3 provides a way to gather statistics about network traffic, including packet transmission, reception, and loss details for individual flows. It also supports tracking delay, jitter, and throughput metrics. Below is an explanation of its integration and usage in the provided code:

2.4 Delay Configuration

Delays in the network are modeled using the `PointToPointHelper` class, which specifies the transmission characteristics of point-to-point links.

2.5 Host-to-Router Links

The delay for host-to-router links is set to a constant value of 2 ms using the following command:

```
p2p.SetChannelAttribute("Delay", StringValue("2ms"));
```

The data rate for these links is set to 1 Mbps:

```
p2p.SetDeviceAttribute("DataRate", StringValue("1Mbps"));
```

2.6 Router-to-Router Links

Delays and data rates for router-to-router links are customized for each pair of connected routers:

- `R1-R2`: Data rate = 3 Mbps
- `R1-R3`: Data rate = 2.5 Mbps
- `R3-R4`: Data rate = 1.5 Mbps
- `R2-R4`: Data rate = 1 Mbps

The delay attribute remains consistent across all router-router links, reflecting realistic transmission conditions.

2.7 Delay Measurement

To measure the end-to-end delays between all hosts, the simulation uses the `FlowMonitor` tool:

- **Average Delay**: Computed as the total delay (`delaySum`) divided by the number of received packets (`rxPackets`).
- **Variance of Delay**: Approximated using jitter data (`jitterSum`).

The delay calculation is implemented as:

```
double avgDelay = flowStat.second.delaySum.GetSeconds() / flowStat.
    second.rxPackets;
double delayVar = ((flowStat.second.jitterSum.GetSeconds() /
    flowStat.second.rxPackets) * 2);
```

2.8 Initialization of the Flow Monitor

The following steps demonstrate how the `FlowMonitor` is set up in the simulation:

1. An instance of `FlowMonitorHelper` is created.
2. The `InstallAll()` function is called to attach the `FlowMonitor` to all nodes in the network.

```
Ptr<FlowMonitor> flowMonitor;
FlowMonitorHelper flowHelper;
flowMonitor = flowHelper.InstallAll();
```

2.9 Node and Link Configuration

We set up 7 hosts and 4 routers, connecting them with point-to-point links of varying data rates.

```
NodeContainer hosts;
hosts.Create(7);
NodeContainer routers;
routers.Create(4);

InternetStackHelper stack;
stack.Install(hosts);
stack.Install(routers);

std::string hostToRouterRates[7] = {"1Mbps", "2Mbps", "1.5Mbps", "3
    Mbps", "1Mbps", "2Mbps", "2.5Mbps"};
NetDeviceContainer hostDevices[7];
for (uint32_t i = 0; i < 7; ++i) {
    PointToPointHelper p2p;
    p2p.SetChannelAttribute("Delay", StringValue("2ms"));
    p2p.SetDeviceAttribute("DataRate", StringValue(
        hostToRouterRates[i]));
```

```
    hostDevices[i] = p2p.Install(NodeContainer(hosts.Get(i),
        routers.Get(i % 4)));
}
```

Listing 1: Host-to-Router Configuration

Router-to-router links are similarly configured with higher data rates.

```
std::string routerToRouterRates[4] = {"4Mbps", "5Mbps", "3.5Mbps",
    "4.5Mbps"};
NetDeviceContainer routerDevices[4];
for (uint32_t i = 0; i < 4; ++i) {
    PointToPointHelper p2p;
    p2p.SetChannelAttribute("Delay", StringValue("2ms"));
    p2p.SetDeviceAttribute("DataRate", StringValue(
        routerToRouterRates[i]));
    if (i == 0) routerDevices[i] = p2p.Install(NodeContainer(
        routers.Get(0), routers.Get(1)));
    else if (i == 1) routerDevices[i] = p2p.Install(NodeContainer(
        routers.Get(0), routers.Get(2)));
    else if (i == 2) routerDevices[i] = p2p.Install(NodeContainer(
        routers.Get(2), routers.Get(3)));
    else if (i == 3) routerDevices[i] = p2p.Install(NodeContainer(
        routers.Get(1), routers.Get(3)));
}
```

Listing 2: Router-to-Router Configuration

2.10 NetAnim in the Code

The **NetAnim** module in this code is used for visualizing the network topology and its dynamics during the simulation. Below is the detailed explanation of its implementation:

1. **Creating the Animation Interface:**

```
    AnimationInterface anim("custom_network_topology.xml");
```

   This line initializes a NetAnim animation interface and specifies the output file, `custom_network_topology.xml`, which contains the animation data for visualization.

2. **Enabling Route Tracking:**

```
    anim.EnableIpv4RouteTracking("route-tracking.xml",
                                 Seconds(1.0), Seconds(10.0),
                                     Seconds(5.0));
```

   This feature tracks the routing changes in the network over time and logs them to `route-tracking.xml`. The parameters specify the start, stop, and interval times for recording the route changes.

3. **Setting Node Positions:** Each node (hosts and routers) is assigned a fixed position in the simulation space for better visualization:

```
    anim.SetConstantPosition(hosts.Get(0), 10, 10); // Host A
    anim.SetConstantPosition(routers.Get(0), 20, 20); //
        Router R1
```

The `SetConstantPosition` method assigns (x, y) coordinates to each node.

4. **Adding Descriptions to Nodes:** Nodes are labeled with their respective names (e.g., "A", "R1") for easy identification:

```
    anim.UpdateNodeDescription(hosts.Get(0), "A");
    anim.UpdateNodeDescription(routers.Get(0), "R1");
```

This method updates the description of each node in the animation output.

5. **Purpose:**
   - NetAnim allows the user to visually analyze the network topology, traffic flow, and routing changes.
   - The animation enhances debugging by providing a graphical representation of the simulation.

2.11 Queue Logging Function

The function `LogQueueLength` records the current queue length of a specified network link.

```
void LogQueueLength(Ptr<Queue<Packet>> queue, std::string
    linkDescription) {
    uint32_t queueLength = queue->GetNPackets();
    logFile << Simulator::Now().GetSeconds() << "s: " <<
        linkDescription
            << " Queue Length: " << queueLength << " packets" <<
                std::endl;
}
```

Listing 3: Queue Logging Function

2.12 Generating and Printing the Traffic Matrix

```
void PrintTrafficMatrix(const std::map<std::pair<std::string, std::
    string>, uint32_t>& trafficMatrix) {
    // Code to format and print the matrix
}
void GenerateTrafficMatrix(const std::vector<std::string>& hosts,
    double lambda) {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::poisson_distribution<> poissonDist(lambda);
    for (const auto& src : hosts) {
        for (const auto& dst : hosts) {
```

```
            if (src != dst) {
                uint32_t trafficLoad = poissonDist(gen);
                trafficMatrix[std::make_pair(src, dst)] =
                    trafficLoad;
            }
        }
    }
}
```

- `GenerateTrafficMatrix`: Uses a Poisson distribution to generate traffic between nodes.
- `PrintTrafficMatrix`: Formats the matrix into a tabular structure and writes it to a file for visualization.

2.13 Node Configuration and IP Address Assignment

```
NodeContainer hosts, routers;
hosts.Create(7);
routers.Create(4);
InternetStackHelper stack;
stack.Install(hosts);
stack.Install(routers);
```

- `hosts.Create(7)`: Creates 7 host nodes (A-G).
- `routers.Create(4)`: Creates 4 routers (R1-R4).
- `stack.Install`: Installs the Internet stack on all nodes for communication.

2.14 Point-to-Point Links and Error Models

```
PointToPointHelper p2p;
p2p.SetDeviceAttribute("DataRate", StringValue("1Mbps"));
p2p.SetChannelAttribute("Delay", StringValue("2ms"));
NetDeviceContainer hostDevices[7], routerDevices;
hostDevices[0] = p2p.Install(NodeContainer(hosts.Get(0), routers.
    Get(0)));
```

- Configures point-to-point links with specific data rates and delays.
- Links hosts to their respective routers based on a predefined topology.

2.15 Routing Table and Traffic Applications

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables();
UdpEchoServerHelper echoServer(9);
ApplicationContainer serverApps = echoServer.Install(hosts.Get(0));
serverApps.Start(Seconds(1.0));
serverApps.Stop(Seconds(10.0));
UdpEchoClientHelper echoClient(Ipv4Address("10.1.0.1"), 9);
ApplicationContainer clientApps = echoClient.Install(hosts.Get(i));
clientApps.Start(Seconds(2.0 + i));
```

- `PopulateRoutingTables`: Computes routes using global routing.
- Sets up a UDP echo server on Host A and echo clients on other hosts (B-G).

2.16 Error Model Integration

```
Ptr<RateErrorModel> errorModel = CreateObject<RateErrorModel>();
errorModel->SetAttribute("ErrorRate", DoubleValue(0.01));
hostDevices[0].Get(1)->SetAttribute("ReceiveErrorModel",
    PointerValue(errorModel));
```

- Introduces a 1% packet loss rate on specific links to simulate network errors.

3 Execution Flow

1. Hosts and routers are created and configured.
2. Links are established between nodes with defined capacities and delays.
3. Traffic matrix is generated and printed.
4. Routing tables are computed, and applications are installed on nodes.
5. The simulation runs with logging enabled for debugging. The simulation is run for 60 seconds:

```
Simulator::Stop(Seconds(60.0));
Simulator::Run();
Simulator::Destroy();
```

Here is the detailed explanation of the provided C++ program written in LaTeX. This includes structured sections, code snippets, and explanations, making it suitable for documentation or presentations.

Here is the LaTeX documentation for the NS-3 simulation code. Let me know if you need additional explanations, changes, or further sections to be added!

4 Output

- The traffic matrix is saved to `traffic_matrix.txt`.
- The delays are printed in `end_to_end_delay.txt`.
- Drop rates are printed in `packet_drop.txt`.
- Routing tables and simulation results can be logged for analysis.