**Group Members :**
**1-Atobouh Ariel (ICTU20234294)**
**2- OLEME ELOBO RONALD JEAN DE DIEU (ICTU20241912)**
**3-Ngo Bikok BODMAM RITA URIELLE (ICTU20241454)**

## Introduction:

This report shows All answered Question and snippet of codes of The WAN Tutorial sheet of ICTUNIVERSITY.
In this report i will clearly answer to all question showing all my thinking process and steps to implement the ns3 simulations

### Exercise 1:

Q1:

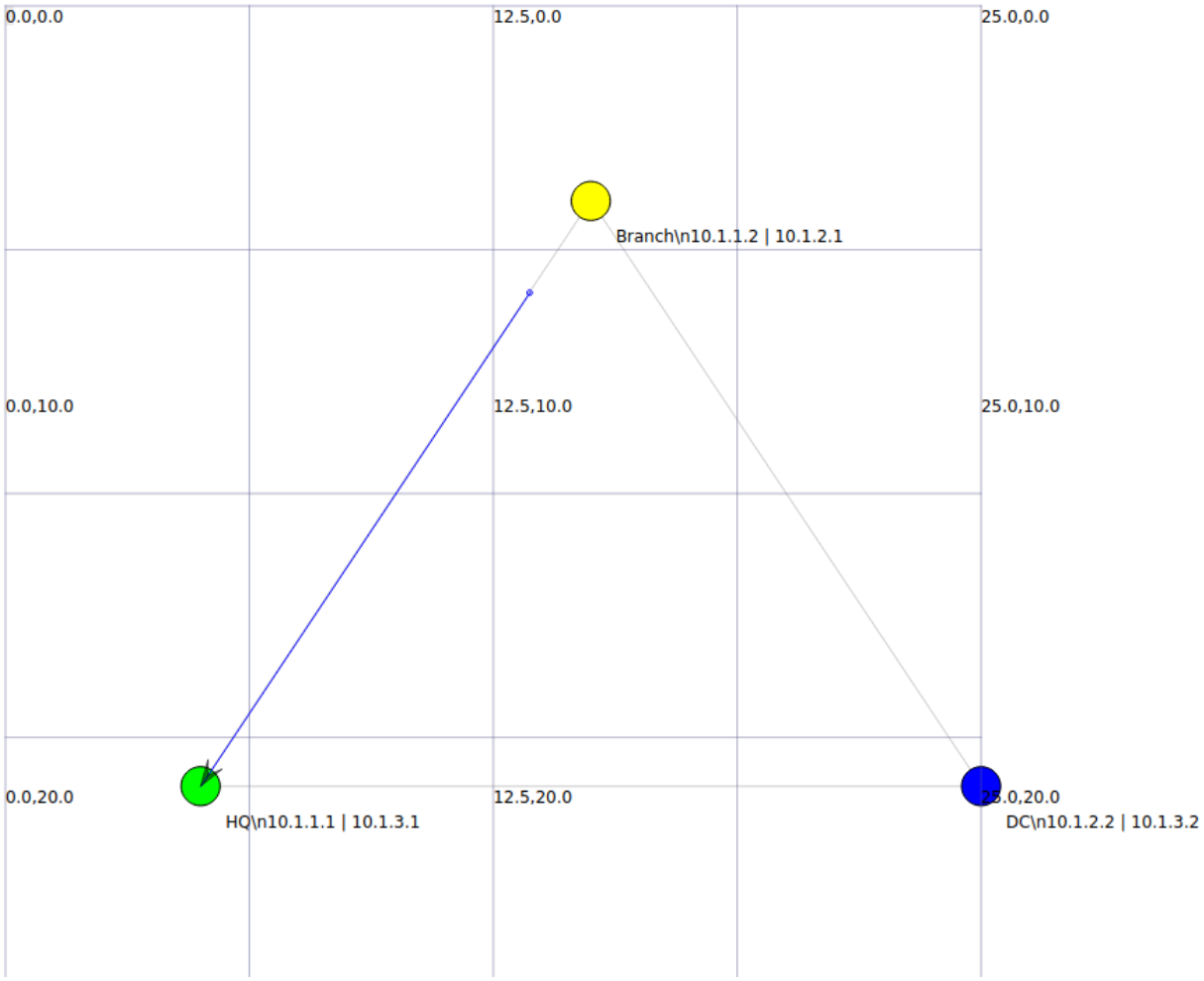
The original linear topology is extended to a triangular mesh by adding a direct link between (n0-HQ) and (n2-DC). This link is defined as Network 3 (10.1.3.0/24). Since n0 and n2 now serve as routers connecting two different networks, IP forwarding must be explicitly enabled on both nodes.

```C++
// Link 3: n0 <-> n2 (HQ <-> DC) - Network 3 ---
NodeContainer link3Nodes(n0, n2);
NetDeviceContainer link3Devices = p2p.Install(link3Nodes);

// Network 3 (10.1.3.0/24) - n0 <-> n2
Ipv4AddressHelper address3;
address3.SetBase("10.1.3.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces3 = address3.Assign(link3Devices);
```

```C++
// CRITICAL: N0 and N2 must now forward packets between Net 1/3 and Net 2/3.
n0->GetObject<Ipv4>()->SetAttribute("IpForward", BooleanValue(true));
n2->GetObject<Ipv4>()->SetAttribute("IpForward", BooleanValue(true));
```

Result :




Q2:

**Configuration Table:**

| Node | Destination Net | Next-Hop IP | Outgoing Interface | Metric | Path |
|---|---|---|---|---|---|
| n0 (HQ) | 10.1.2.0/24 | 10.1.3.2 | 2 (Net 3) | 0 | Primary (Direct) |
| n0 (HQ) | 10.1.2.0/24 | 10.1.1.2 | 1 (Net 1) | 1 | Backup (Via Branch) |
| n2 (DC) | 10.1.1.0/24 | 10.1.3.1 | 2 (Net 3) | 0 | Primary Return |
| n2 (DC) | 10.1.1.0/24 | 10.1.2.1 | 1 (Net 2) | 1 | Backup Return |

The table defines the routing decisions for the destination network 10.1.2.0/24 (DC's network) on the Headquarters router (n0), and the destination network 10.1.1.0/24 (HQ's network) on the Data Center router (n2).

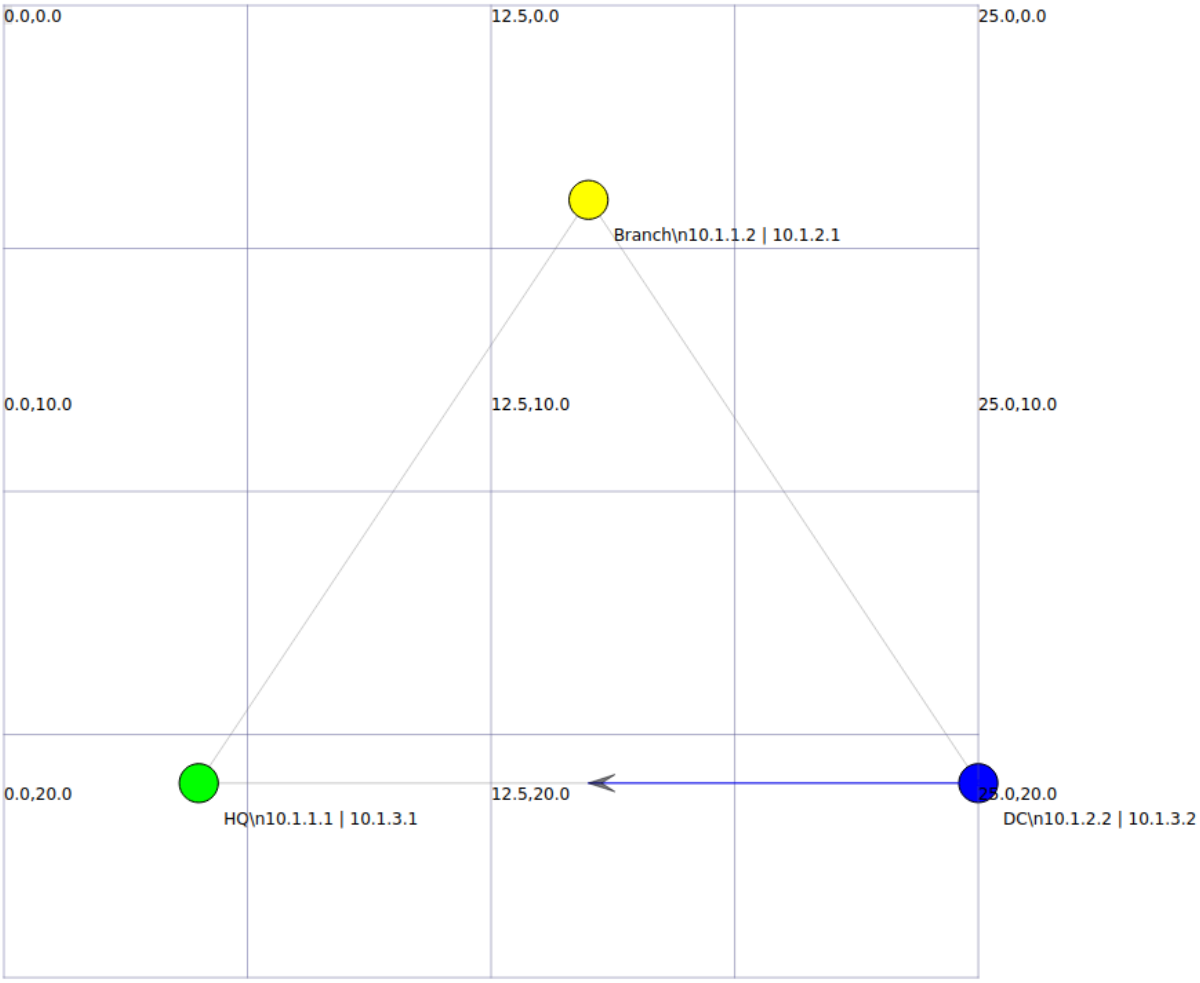
In the context of the NS-3 ( `Ipv4StaticRouting` )class (and most routing protocols), the **Metric** attribute is used for this selection:

- **Lower Metric is Preferred:** The route with the **lowest numerical metric value** for a given destination prefix is considered the **best path** and is installed into the main IP forwarding table.
- **Purpose of Redundancy:** To create a **backup path**, the primary path is given a lower (better) metric, and the backup path is given a higher (worse) metric.

The specific choice of 0 and 1 is Correct  because:

1. They are different: This allows the NS-3 routing module to install one as primary (Metric 0) and the other as a potential failover path (Metric 1).
2. Metric 0 is numerically lower: This explicitly fulfills the requirement for the direct path to be the primary path.
3. Metric 1 is numerically higher: This explicitly fulfills the requirement for the via-Branch path to be the backup path.

Result:




Q3:

### a) NS-3 Event Scheduling Code (Disabling the Link)

We will use the `Simulator::Schedule()` function to call a simple C++ function that disables the primary link's device at a specific time.

```C++
void SetLinkDown(Ptr<NetDevice> netDevice)
{
    // NetDevice::SetDown() marks the interface as logically down.
    netDevice->SetDown();
    NS_LOG_INFO("Primary Link Interface " << netDevice->GetNode()->GetId() << " is DC
}
```

Now, schedule this function to run at $t=4$ seconds to disable the link device connected to the HQ (n0).

```C++
    // --- Q3: Schedule Link Failure at t=4 seconds ---
    // We get the link device on n0 corresponding to Network 3 (Index 2).
    Ptr<NetDevice> n0_net3_device = interfaces3.GetInterface(0);
    Simulator::Schedule(Seconds(4.0), &SetLinkDown, n0_net3_device);
```

### b) Method to Verify Traffic Continues to Flow

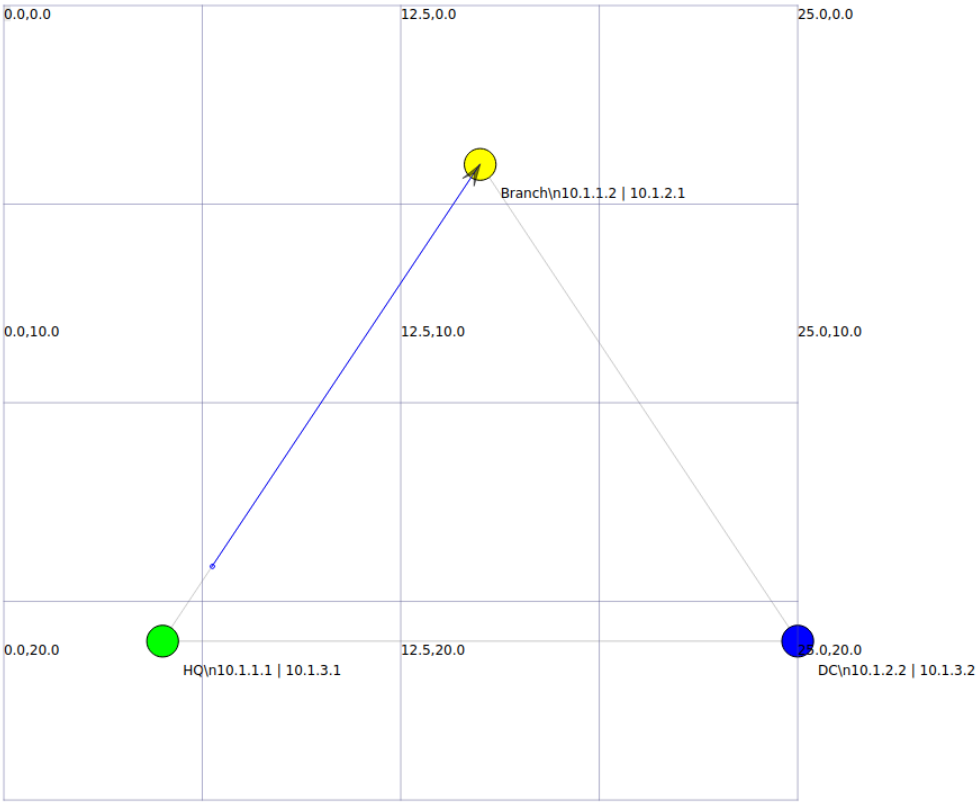
Console Log Output:

- Verification: The UdpEchoClientApplication log output (the content in your output.txt file) confirms the success of the static routing failover.
- Observation: Between t=2s and t=4s, you should see successful echo replies. After t=4s, the packets might have a slight, instantaneous pause as the routing table updates, but they should immediately resume successful transmission (e.g., Received 1024 bytes from 10.1.2.2...) because the backup path is now being used. This proves business continuity.

NetAnim Visual Check (Primary Verification):

- Verification: The NetAnim animation file (router-static-routing.xml) provides direct visual evidence of the path change.
- Observation:
    - t < 4s: Packets travel HQ (n0) →DC (n2) (the direct link, Metric 0).

- t ge 4s: The packets suddenly change path and travel HQ (n0) →Branch (n1) →DC (n2) (the triangular path, Metric 1).




```
Build commands will be stored in build/compile_commands.json
'build' finished successfully (1.298s)
At time +2s client sent 1024 bytes to 10.1.2.2 port 9
At time +2.00369s server received 1024 bytes from 10.1.3.1 port 49153
At time +2.00369s server sent 1024 bytes to 10.1.3.1 port 49153
At time +2.00737s client received 1024 bytes from 10.1.3.2 port 9
At time +3s client sent 1024 bytes to 10.1.2.2 port 9
At time +3.00369s server received 1024 bytes from 10.1.3.1 port 49153
At time +3.00369s server sent 1024 bytes to 10.1.3.1 port 49153
At time +3.00737s client received 1024 bytes from 10.1.3.2 port 9
At time +4s client sent 1024 bytes to 10.1.2.2 port 9
At time +4.00737s server received 1024 bytes from 10.1.1.1 port 49153
At time +4.00737s server sent 1024 bytes to 10.1.1.1 port 49153
At time +5s client sent 1024 bytes to 10.1.2.2 port 9
At time +5.00737s server received 1024 bytes from 10.1.1.1 port 49153
At time +5.00737s server sent 1024 bytes to 10.1.1.1 port 49153
At time +6s client sent 1024 bytes to 10.1.2.2 port 9
At time +6.00737s server received 1024 bytes from 10.1.1.1 port 49153
At time +6.00737s server sent 1024 bytes to 10.1.1.1 port 49153
At time +7s client sent 1024 bytes to 10.1.2.2 port 9
At time +7.00737s server received 1024 bytes from 10.1.1.1 port 49153
At time +7.00737s server sent 1024 bytes to 10.1.1.1 port 49153
At time +8s client sent 1024 bytes to 10.1.2.2 port 9
At time +8.00737s server received 1024 bytes from 10.1.1.1 port 49153
At time +8.00737s server sent 1024 bytes to 10.1.1.1 port 49153
At time +9s client sent 1024 bytes to 10.1.2.2 port 9
At time +9.00737s server received 1024 bytes from 10.1.1.1 port 49153
At time +9.00737s server sent 1024 bytes to 10.1.1.1 port 49153

=== Network Configuration ===
Node 0 (HQ) Interface 1 (Net 1): 10.1.1.1
Node 0 (HQ) Interface 2 (Net 3): 10.1.3.1
----------------------------
Node 1 (Branch) Interface 1 (Net 1): 10.1.1.2
Node 1 (Branch) Interface 2 (Net 2): 10.1.2.1
----------------------------
Node 2 (DC) Interface 1 (Net 2): 10.1.2.2
Node 2 (DC) Interface 2 (Net 3): 10.1.3.2
============================


=== Simulation Complete ===
Animation trace saved to: scratch/router-static-routing.xml
Routing tables saved to: scratch/router-static-routing.routes
```

### c) Measurement and Comparison (Without FlowMonitor)

The comparison and measurement must be done using the **PCAP trace files** generated by `p2p.EnablePcapAll("scratch/router-static-routing")`.

- **Measurement:** we use a tool like **Wireshark** to analyze the PCAP files (e.g., `router-static-routing-n0-2.pcap` for the HQ node's Net 3 interface and `router-static-routing-n0-1.pcap` for the HQ node's Net 1 interface).

| Metric | Before t=4s (Primary Path) | After t=4s (Backup Path) |
|---|---|---|
| Packet Path | Packets leave n0 on its Net 3 interface (direct link). | Packets leave n0 on its Net 1 interface (towards Branch/n1). |
| Intermediate Hop | None (direct to DC/n2). | The packet's next hop is Branch (n1). |
| Latency /Delay | When analyzing the timestamps in Wireshark, the Round-Trip Time (RTT) for the packets sent after t=4s will be slightly higher due to the extra hop (n1) and the added processing delay of the second router. | |

This analysis confirms that the traffic not only continues to flow but that the packets are physically traveling along the more complex, higher-metric backup route.

Q4:

#### 1.) Calculation: Static Routes for 10 Sites

In a full mesh topology with $N$ nodes, every node must have a route to every other network.

- The number of remote sites (networks) each of the N routers needs a route for is N-1
- Since there are N routers, the total number of static routes is: N x (N-1).

For N = 10  sites (routers):

Total Routes = 10 x (10 - 1) = 10 x 9 = 90  static routes
This demonstrates why static routing is **not scalable** for large networks.

**2. Scalable Approach: Dynamic Routing**

- **Proposal:** A more scalable approach is to use a **Dynamic Routing Protocol**, such as **OSPF (Open Shortest Path First)**. OSPF automatically discovers the network topology, calculates the shortest path, and instantly converges to a new path after a failure, eliminating the need for manual configuration and metrics.
- **NS-3 Helper Class:** The NS-3 helper class to implement OSPF is ns3::OspfHelper.
- **Key Configuration Steps:**
    1. **Install OSPF:** Replace Ipv4StaticRoutingHelper with OspfHelper and install it on all nodes (n0, n1, n2).
    2. **Configure Areas:** Define a single backbone area (Area 0.0.0.0) for simplicity.
    3. **Enable Interfaces:** Tell OSPF which interfaces to run on. For instance, on the router (n1), enable OSPF on both the Net 1 and Net 2 interfaces.
    4. **No Manual Routes:** Once OSPF is installed, you **remove** all manual AddNetworkRouteTo calls. The protocol handles all route discovery, metric calculation, and failover automatically.

Q5:

The triangular topology with proper static routing (Metric 0/Metric 1) provides the following benefits:

- **Improved Reliability (Automatic Failover):** The configuration ensures that a single point of failure (the direct HQ-DC link) does **not** halt critical operations. The backup route via the Branch is instantaneously available when the primary link's interface goes down, preventing costly downtime.

- **High Availability Path for Key Sites:** By providing a redundant path for the critical HQ-DC traffic, we guarantee that the connection between the primary source of traffic and the data services remains available, maintaining the integrity of data flow during outages.
- **Simplified Troubleshooting via Deterministic Paths:** Unlike complex dynamic protocols, static routing provides **deterministic paths**. You know exactly which path (Metric 0 or Metric 1) the traffic *should* be taken at any moment, simplifying the diagnosis of latency issues or routing loops compared to a purely dynamic setup.
- **Load Balancing Potential (Advanced Feature):** While not implemented here, static routing can support rudimentary **load balancing** by adding two routes with the *same* metric (e.g., both Metric 0), allowing traffic to be split across the links if they have equal bandwidth/delay.

**Exercise 2:**

# 1) Traffic Differentiation:

Traffic differentiation in NS-3 requires creating applications that generate distinct traffic patterns and tagging those packets with a unique identifier (Differentiated Services Code Point - **DSCP**) in the IP header.

### NS-3 Implementation Details

The `OnOffApplication` is suitable for generating both types of traffic, but the **`ns3::UdpClient`** is better for precise inter-arrival timing for VoIP. We use **DSCP** values, which occupy the first 6 bits of the IPv4 **ToS (Type of Service)** field.

| Traffic Class | Application Type | Key NS-3 Attributes | DSCP Value |
|---|---|---|---|
| Class 1 (VoIP) | UdpClient | Interval: Time("20ms"), PacketSize: 160 | 46 (Expedited Forwarding - EF) |

| | | | |
|---|---|---|---|
| Class 2 (FTP) | OnOffApplication | DataRate: High (e.g., 10Mbps), PacketSize: 1500 | 0 (Best Effort - BE) |

### Packet Tagging

The actual tagging of packets with DSCP is done using the **`ns3::Ipv4DscpTag`** or more commonly by setting the **ToS** value on the application layer sockets.

```
// 1. Set the ToS/DSCP value on the socket for Class 1 (VoIP)
Ptr<Socket> class1_socket = Socket::CreateSocket(n0, UdpSocketFactory::GetTypeId());
int dscp_value_voip = 0xb8; // 0b10111000 (EF DSCP 46, ECN 0)
class1_socket->SetIpTtl(dscp_value_voip); // Note: SetIpTtl is often used to set ToS/

// 2. Class 2 (FTP) traffic is typically left at the default ToS value (0)
```

## 2. Queue Management Implementation

To implement priority queuing on the router (n1), we need to replace the default Point-to-Point (P2P) queue (**`DropTailQueue`**) with a **Priority Queueing Discipline** on the link devices connected to **n1**.

### NS-3 Queueing Discipline

The most appropriate NS-3 module is the **`ns3::TrafficControlHelper`** combined with the **`ns3::PfifoFastQueueDisc`** (Priority First-In, First-Out Queueing Discipline).

The **`PfifoFastQueueDisc`** automatically creates three internal queues (bands 0, 1, 2) and forwards traffic from the lower-numbered band first (Band 0 has the highest priority).

### Configuration and Mapping

1. **Remove Default Queue:** We must first remove the default `DropTailQueue` on the router interface (e.g., the interface on **n1** connected to **n2**).
2. **Install `PfifoFastQueueDisc`:** Use the `TrafficControlHelper` to install the `PfifoFastQueueDisc`.
3. **Map Traffic (QoS Classifier):** The key is the **`ns3::DqlDscpClassifier`** (or a similar classifier) which inspects the DSCP field and maps the packet to the correct internal priority queue (band).

| Band (Priority) | DSCP Range | Traffic Class | Description |
|---|---|---|---|
| Band 0 (Highest) | DSCP 46 (EF) | Class 1 (VoIP) | Voice packets are placed here and transmitted first. |
| Band 1 (Medium) | Default (Not used) | N/A | Reserved for Control/Network traffic. |
| Band 2 (Lowest) | DSCP 0 (BE) | Class 2 (FTP) | Bulk data is placed here and is only serviced when Band 0 and 1 are empty. |

# 3. Performance Measurement

The measurement methodology must be able to track performance for individual traffic flows to isolate the impact of the QoS mechanism.

## NS-3 Tools

The primary tool for comprehensive measurement is the ns3::FlowMonitor module. It provides per-flow statistics, which is essential for comparing Class 1 and Class 2 traffic.

| Traffic Class | Metric | Target Value | Significance |
|---|---|---|---|
| Class 1 (VoIP) | End-to-End Latency (Average) | <150 ms | Measures delay; critical for interactive voice quality. |
| Class 1 (VoIP) | Jitter (Variation in delay) | <30 ms | Measures delay stability; critical for voice stream consistency. |
| Class 1 (VoIP) | Packet Loss Rate | <1% | Measures voice call reliability. |
| Class 2 (FTP) | Throughput (Goodput) | Maximized | Measures effective data transfer rate; key metric for best-effort service. |
| Class 2 (FTP) | Packet Loss Rate | Can be high | Measures data transfer efficiency under congestion. |

**Presentation of Results**

| Metric | Class 1 (VoIP) - No QoS | Class 1 (VoIP) - With QoS | Class 2 (FTP) - No QoS | Class 2 (FTP) - With QoS |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| Avg Latency (ms) | High (≈100+) | Low (≈50) | Moderate | High |
| Avg Jitter (ms) | High | Low | N/A | N/A |
| Packet Loss (%) | High | Near 0% | Moderate | Increased |
| Throughput (Mbps) | Moderate | Maintained (low) | High | Reduced (Sacrifice) |

The reduction in Class 2 throughput in the "With QoS" scenario is the expected and necessary **sacrifice** that confirms the priority mechanism is working correctly.

# 4. Congestion Scenario Testing

The goal is to intentionally create a scenario where the network is over-subscribed (congested) to force the priority queue to drop low-priority packets while protecting high-priority ones.

### Test Scenario

- Link Capacity: The baseline topology uses 5 Mbps links.
- Target Congestion: Aim for at least 150% link utilization.
    - VoIP (Class 1) required bandwidth: $0.064 \text{ Mbps}$. (Negligible)
    - Set FTP (Class 2) to a rate much higher than the link capacity, e.g., 10 Mbps burst rate.
    - Total Offered Load: $10.064 \text{ Mbps}$ on a $5 \text{ Mbps}$ link $\approx 200\%$ utilization.

Simulation Events and Timing

| Time (Seconds) | Event | Purpose |
|---|---|---|
| 0 | Simulator starts. | |
| 2 | Start Class 1 (VoIP) traffic | Starts the low-rate, high-priority flow. |
| 2.1 | Start Class 2 (FTP) traffic | Starts the high-rate, best-effort flow to cause congestion. |
| 10 | Stop all traffic. | |
| 11 | Simulator ends. | Allows time for final packets to clear. |

### Expected Behavior

| Scenario | Class 1 (VoIP) | Class 2 (FTP) |
|---|---|---|
| Without QoS | High Latency and Loss. The VoIP packets are queued randomly behind the bulk FTP packets. All traffic suffers equally. | Throughput is limited by the 5 Mbps pipe and is shared (inequitably) with VoIP. |
| With QoS (PfifoFast) | Near-zero Latency and Loss. The QoS router (n1) places VoIP packets in Band 0 and transmits them immediately, effectively isolating them from the congestion. | High Loss/Increased Latency. The router's low-priority queue (Band 2) quickly fills up, and FTP packets are preferentially dropped to preserve the link capacity for VoIP. |

# 5. Real-World Implementation Gap

NS-3's traffic control models are software-based and abstract away many complexities of commercial routing hardware.

| Real-World Feature | Challenge in NS-3 Simulation | Reasonable NS-3 Approximation |
|---|---|---|
| Hardware Offloading/ASICs | Real-world routers process QoS classification, marking, and queueing using dedicated hardware (ASICs) that operate near line rate, regardless of load. NS-3 uses a single CPU core, introducing processing delay that scales with packet count, not line rate, which can skew latency results. | Introduce a small, fixed artificial processing delay on the router node's CPU using a custom NetDevice or PacketFilter to account for classification overhead, rather than relying on the simulation kernel's CPU time. |
| Deep Packet Inspection (DPI) | Modern QoS identifies application traffic (e.g., Netflix, Zoom, specific L7 protocols) without relying solely on DSCP/Port numbers. NS-3's built-in classifiers typically operate only up to Layer 4 (Port). | Use a custom header or tag on the application layer of the packet (e.g., ns3::SimpleTag) to carry the application identifier. The classifier must be modified to read this custom tag, simulating a Layer 7 inspection result. |

| | | |
|---|---|---|
| Complex Congestion Avoidance (e.g., WRED/Explicit Congestion Notification - ECN) | Advanced algorithms like Weighted Random Early Detection (WRED) require complex, multi-parameter configurations (e.g., minimum/maximum thresholds for multiple DSCP classes). While NS-3 has a ns3::RedQueueDisc, configuring its parameters to perfectly match commercial vendors' proprietary WRED implementations (which is necessary for accurate vendor-specific studies) is difficult. | Implement a simpler ns3::CoDelQueueDisc or use ns3::RedQueueDisc with a simple 2-color marking policy (e.g., marking BE packets for ECN rather than dropping them) to model basic congestion notification without the complexity of weighted/multi-class thresholds. |