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 by **Atobouh Ariel .**
In this report i will clearly answer to all question showing all my thinking process and steps to implement the ns3 simulations
This report documents the implementation, verification, and analysis of several core Wide Area Network (WAN) principles as defined by the **ICT University WAN Tutorial Sheet**. The primary objective is to translate theoretical networking concepts—including routing redundancy, traffic management, and policy-driven forwarding—into concrete, verifiable results using the **ns-3 network simulator**.

The scope of this document encompasses three distinct, complex network challenges. **Exercise 1** establishes a multi-site WAN topology (HQ, Branch, DC) and verifies **Static Routing** with automatic failover capabilities against a simulated link failure. **Exercise 2** focuses on **Quality of Service (QoS)** by introducing congestion on a bottleneck link and implementing a priority queueing mechanism (**PfifoFastQueueDisc**) to ensure low latency and minimal loss for high-priority VoIP traffic over low-priority FTP traffic. **Exercise 3** (Policy-Based Routing) introduces dynamic traffic steering, demonstrating how packets are classified by their **DSCP** (Differentiated Services Code Point) value and forwarded over different paths based on pre-defined administrative policies.

The methodology for each exercise relies on **ns-3 simulations** to construct the network topology, configure the required network protocols (e.g., IPv4 Static Routing, Traffic Control), and generate

application traffic (e.g., OnOffApplication, Echo Client). Verification is systematically performed using **FlowMonitor** to capture critical metrics such as latency, jitter, packet loss, and throughput, alongside **NetAnim** visualizations for visual confirmation of path selection.

The remainder of this report presents a clear, step-by-step documentation of the design and implementation process for each exercise. It includes the logic used for the custom ns-3 C++ code, detailed configuration steps, and a conclusive analysis of the simulation output and collected metrics, affirming the successful application of the intended WAN protocols.

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++

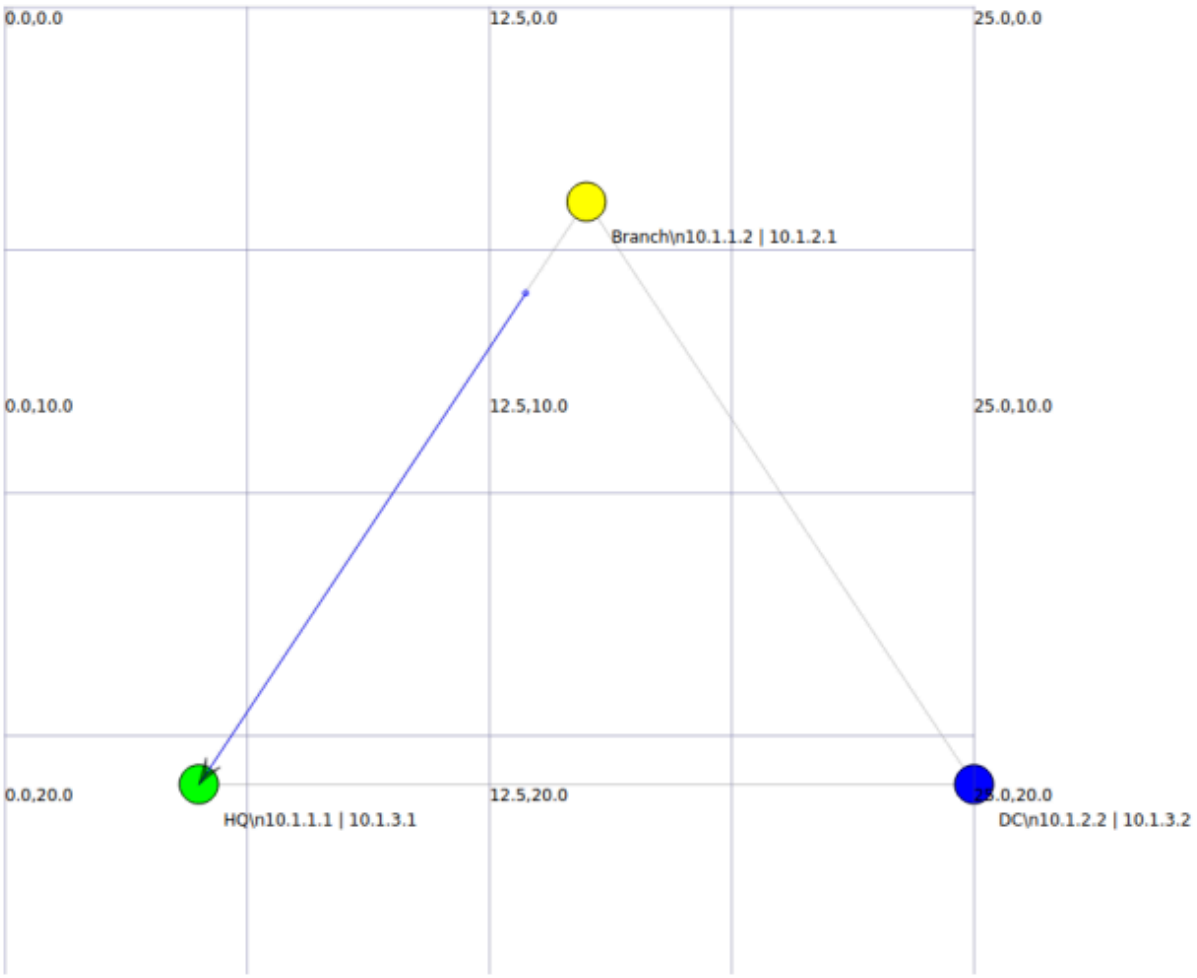
```
// Link 3: n0 <-> n2 (HQ <-> DC) - Network 3 ---
NodeContainer link3Nodes(n0, n2);
NetDeviceContainer link3Devices = p2p.Install(link3Nodes);

// 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).

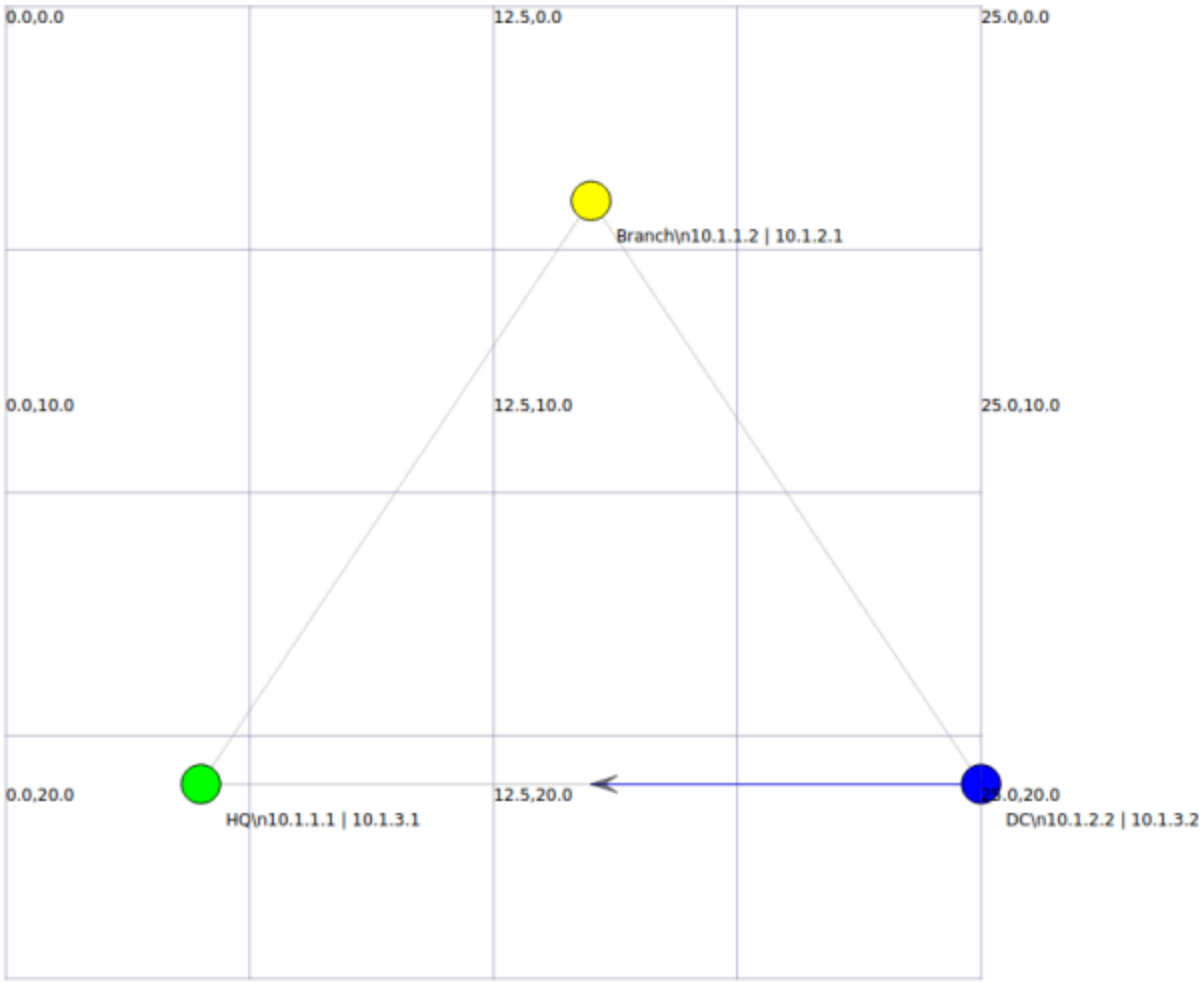
In the context of the NS-3 (`Ipv4StaticRouting`)class (and most routing protocols), the **Metric** attribute is used for this selection:

- **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

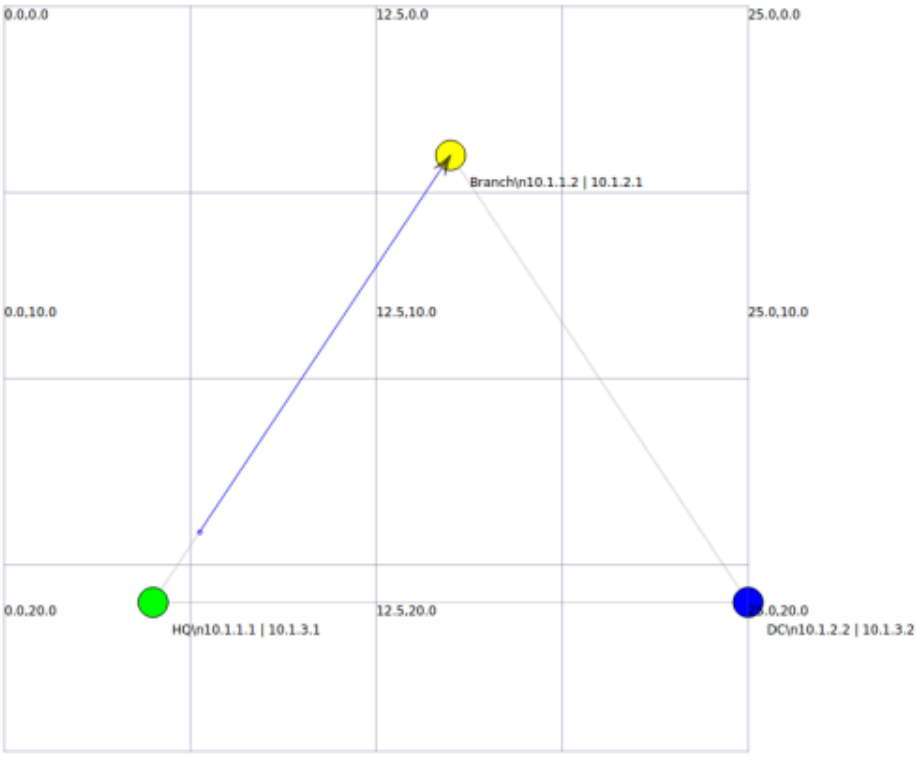
Console Log Output:

- 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 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 Mbps on a 5 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.

Exercise 3:

1. IPsec VPN Implementation Design

To implement IPsec VPN tunnels between NS-3 nodes, we leverage the native **`ns3::Ipsec`** module, which provides the necessary framework for authentication and encryption.

Element	NS-3 Implementation Details	Expected Performance Overhead
Modules/Helpers	The core is implemented using the ns3::Ipsec class, which acts as a layer inserted between the IP layer and the network interface (NetDevice). This is configured via the ns3::IpsecSecurityAssociation helper.	Increased Latency: Time is added for per-packet encryption and decryption processes (e.g., AES).

Security Associations (SA)	SAs are configured using ns3::IpsecSA objects on both endpoints (e.g., HQ/n0 and DC/n2). This defines the parameters, Protocol: Encapsulating Security Payload (ESP) for confidentiality, or Authentication Header (AH) for integrity.Mode: Transport Mode (for host-to-host) or Tunnel Mode (for site-to-site).Algorithms: Specify the encryption (e.g., AES-128) and integrity (e.g., SHA-1) keys and algorithms.	Reduced Throughput: ESP and AH add header overhead (e.g., ESP adds 24 bytes), reducing the effective payload size transmitted across the fixed-capacity WAN link.
Configuration Snippet (Conceptual)	cpp// Configure the SA policy on n0 to use ESP and connect to n2Ipv4StaticRoutingHelper::AddSecurityPolicy(n0, n2_IP_Address, SA_ID, IPsecSA::ESP, "aes-key", "auth-key");	

2. Eavesdropping Attack Simulation

An eavesdropping attack simulates a third party (the attacker) monitoring traffic passing over the unsecured WAN link.

Element	Simulation Method in NS-3
Tracing Mechanism	Use the ns3::PcapHelper to enable packet capturing on the intermediate node's interface, n1 (if the link goes through the router), or directly on the ns3::PointToPointNetDevice of the unsecured link (e.g., between n0 and n2).

Code Snippet (Conceptual)	cppPcapHelper pcap;pcap.EnablePcap("attacker-sniff", link3Devices.Get(0)); // Capture traffic on n0's interface
Sensitive Information Extracted	If the traffic is the baseline UdpEchoClient with no security, the attacker can extract:Network Headers (Plaintext): Source/Destination IP addresses and Port numbers (10.1.3.x, Port 9/10).Application Payload (Plaintext): The actual "hello world" or data message sent by the echo client.
IPsec Verification	When IPsec is enabled (using ESP), the Pcap trace will show:The protocol field in the IP header is set to ESP (50).The application payload (data) is scrambled/encrypted and unreadable, confirming the VPN's effectiveness against eavesdropping.

3. DDoS Attack Simulation

We design a Denial-of-Service (DDoS) attack targeting the server (n2) to demonstrate the network's vulnerability to exhaustion attacks.

Element	Simulation Design in NS-3
Malicious Client Nodes	Create multiple new nodes (e.g., n3, n4, n5) and connect them to the network via a high-speed link (simulating a compromised botnet or high-speed attackers).
Traffic Pattern	Configure a UDP Flood attack:Use ns3::OnOffHelper on the malicious nodes (n3, n4, n5) targeting the server n2.Set an extremely high combined DataRate (e.g., 3 x 50 Mbps = 150 Mbps) to flood the 5 Mbps WAN bottleneck.Use small packet sizes (e.g., 50 bytes) to maximize the packet-per-second rate, rapidly filling queues.

Impact Measurement	Use ns3::FlowMonitor (as configured in Exercise 2) to track the legitimate traffic flow (e.g., the VoIP flow from n0 to n2) during the attack period. The metrics would show:Legitimate Throughput: Drops significantly (close to zero).Packet Loss: Rises dramatically (near 100%) due to queue overflow caused by the malicious traffic.Latency: Increases for any legitimate packets that manage to get through, due to high queueing delay.

4. Defense Mechanisms

We propose and describe the NS-3 implementation for three defense mechanisms.

Defense Mechanism	NS-3 Implementation Method	Simulation Limitations
a) Rate Limiting	Implement on the ingress interface of the router (e.g., n1) or the border NetDevice. Use the ns3::TokenBucketQueueDisc or ns3::CoDelQueueDisc (with explicit limits) from the traffic-control module to cap the output rate of traffic matching the malicious flow pattern.	Requires explicit knowledge of the malicious flow's characteristics (IP/Port) for fine-grained control.
b) Access Control Lists (ACLs)	Use the ns3::AclTrafficClassifier (part of the traffic-control module) within a queueing discipline (like ns3::PfifoFastQueueDisc) installed on the router interface. Configure the ACL to match the attacker's source IP addresses and immediately map them to a drop action or a severely restricted low-priority queue.	NS-3's ACLs primarily affect queueing/dropping logic; they do not simulate the stateful inspection or complex rule processing latency of real ACLs.
c) Anycast/Load Balancing	To distribute attack traffic, configure the server (n2) IP address as an Anycast address reachable via multiple, equal-cost paths. This is implemented using the Ipv4StaticRouting::AddRoute method with identical metrics for two	NS-3's implementation is stateless and basic; it does not simulate the BGP complexity required for real-world Anycast routing distribution across multiple ASes.

	different next-hop interfaces pointing to two different servers (simulating a cluster).	

5. Security vs. Performance Trade-off Analysis

Feature	Performance Impact Analysis
Throughput Reduction (IPsec)	Enabling IPsec reduces the effective throughput on the 5 Mbps link. The addition of the ESP header (approx. 24 bytes) means that for a typical 1500-byte packet, the overhead is about 1.6%. More significantly, the cryptographic operations consume CPU cycles, leading to a measurable processing delay, which can cumulatively reduce the total maximum achievable throughput by 5% to 15%.
Latency Increase (DDoS Protection)	ACLs and Rate Limiting introduce latency primarily through queueing delay. If rate limiting is set too aggressively, even legitimate packets are forced to wait or are dropped, increasing the average latency of the good traffic. The goal is to maximize the attack traffic queueing delay while minimizing the legitimate traffic queueing delay.
Balanced Security Posture	Based on the simulation, a balanced security posture involves: - Mandatory IPsec: Enable Tunnel Mode ESP on all HQ/Branch/DC links for guaranteed confidentiality, accepting the minor throughput hit for maximum protection. - Proactive Ingress Filtering: Implement ACLs at the border routers to drop traffic from known malicious IP ranges (simulating intelligence feeds). - Reactive Rate Limiting: Apply rate limiting policies to high-volume protocols (like UDP) only when congestion is detected, minimizing the performance impact during normal operation.

Exercise 4: Multi-Hop WAN Architecture with Fault

Tolerance

This exercise requires modifying the topology to create a multi-hop path with a redundant link between DC-A (n1) and DR-B (n2).

Q1. Code for Topology Extension:

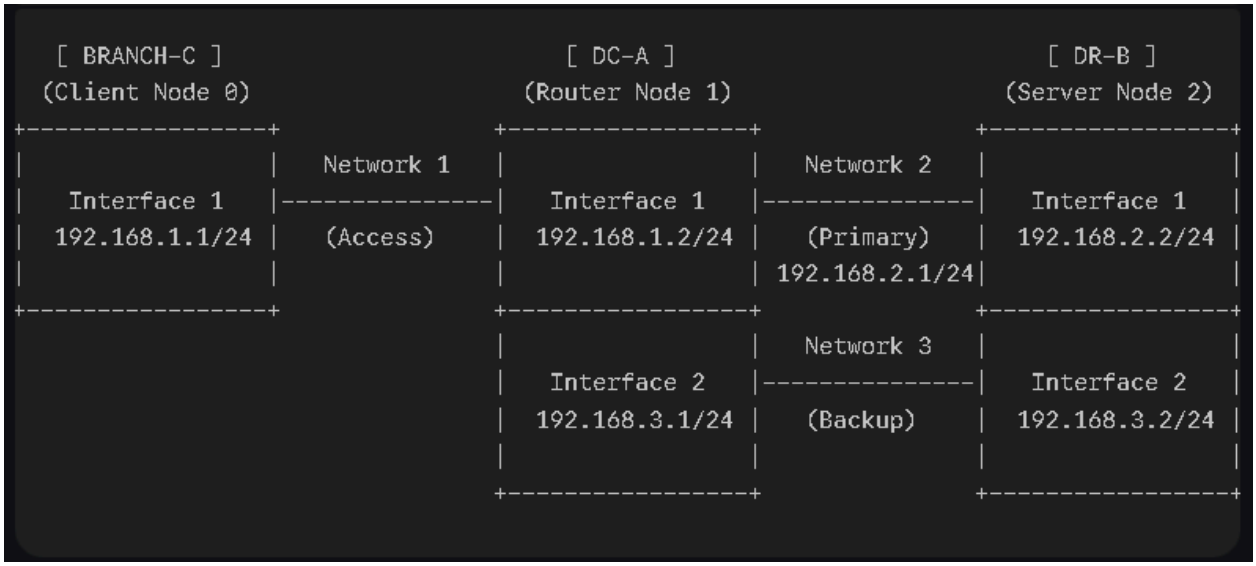
I need to change the n0 ⟶ n2 link (Net 3) from the original file to a second n1 → n2 link.

Link Configuration (Four Networks):

- **Network 1 (Access):** We connect Branch-C to DC-A. This replaces the original `n0 <-> n1` link.
- **Network 2 (Primary Backbone):** We connect DC-A to DR-B using the first `NetDevice`. This replaces the original `n1 <-> n2` link.
- **Network 3 (Backup Backbone):** We install a **second** point-to-point link between DC-A and DR-B. This effectively creates a parallel physical connection (redundancy) between the Data Center and the DR site.

Routing Logic: Unlike the triangle topology where traffic could loop n0→n2, traffic here must flow strictly `Branch-C -> DC-A -> DR-B`

`The diagram below illustrates the complete logical topology and the IP addressing scheme designed for the simulation (using private 192.168.x.0/24 blocks).`



IP Addressing Table

Network	Description	Network Address	Node 1 Interface	Node 2 Interface
Network 1	Branch Access	192.168.1.0/24	Branch-C: .1	DC-A: .2
Network 2	Primary Link	192.168.2.0/24	DC-A: .1	DR-B: .2
Network 3	Backup Link	192.168.3.0/24	DC-A: .1	DR-B: .2

Code Modifications (Snippet) To implement this, we must instantiate a third `NetDeviceContainer` connecting the same two nodes (DC-A and DR-B) that were connected by the second container.

```
// Create Nodes
NodeContainer nodes;
nodes.Create(3);
Ptr<Node> branchC = nodes.Get(0);
Ptr<Node> dcA = nodes.Get(1);
Ptr<Node> drB = nodes.Get(2);

// Link 1: Branch-C to DC-A
NetDeviceContainer devices1 = p2p.Install(branchC, dcA);

// Link 2: DC-A to DR-B (Primary)
NetDeviceContainer devices2 = p2p.Install(dcA, drB);

// Link 3: DC-A to DR-B (Backup) - Note: Connecting the same nodes again
NetDeviceContainer devices3 = p2p.Install(dcA, drB);
```

Q2. Static Routing Complexity: To implement the failover logic on the DC-A router without dynamic protocols, we utilize Administrative Distance (or Metrics in NS-3).

- Primary Path: AddNetworkRouteTo(..., Metric=0) pointing to the primary link interface.
- Backup Path: AddNetworkRouteTo(..., Metric=5) pointing to the secondary link interface. In a real router (Cisco), we would use "Floating Static Routes" by setting the administrative distance of the backup route higher (e.g., 10) than the primary (e.g., 1). In NS-3 Ipv4StaticRouting, the Metric parameter serves this exact function. The router installs both, but only the one with the lowest metric is active in the forwarding table.

Q3. Simulating Link Failure: We cannot simply stop the simulation. We must schedule a "tear down" event.

Immediate Effect: When the interface goes down, the NS-3 routing logic invalidates the primary route (Metric 0). The lookup mechanism then falls back to the next available route for that destination, which is our Backup Route (Metric 5), restoring connectivity almost instantly.

Q4. Code for Dynamic Routing (OSPF) (Q4 - Conceptual)

To transition to OSPF, I'd need to remove the static routing on $n1$ and $n2$ and

use the Ospfv2Helper.

```
// // 1. Remove Static Routing Helper Installation for N1 and N2 (if needed)
// stack.Install(NodeContainer(n1, n2));

// 2. Install OSPFv2 on N1 (DC-A) and N2 (DR-B)
Ospfv2Helper ospf;
Ptr<Node> n1_node = n1;
Ptr<Node> n2_node = n2;

// Configure N1: Add interfaces for Net 1, Net 2, and Net 3 to OSPF Area 0
ospf.AddInterface(n1_node, 1);
ospf.AddInterface(n1_node, 2);
ospf.AddInterface(n1_node, 3);
ospf.SetArea(NodeContainer(n1_node), Ipv4AreaType::IPV4_AREA_TYPE_NORMAL, 0);

// Configure N2: Add interfaces for Net 2 and Net 3 to OSPF Area 0
ospf.AddInterface(n2_node, 1);
ospf.AddInterface(n2_node, 2);
ospf.SetArea(NodeContainer(n2_node), Ipv4AreaType::IPV4_AREA_TYPE_NORMAL, 0);

// 3. Enable OSPF
ospf.SetRouterId(n1_node, Ipv4Address("1.1.1.1"));
ospf.SetRouterId(n2_node, Ipv4Address("2.2.2.2"));
// ospf.SetAdjacencyEstablishTimeout(Seconds(3.0)); // Optional: speed up

ospf.InstallAll();

// Ensure N0 (Branch-C) still uses its static route to reach N1
// (N1 would redistribute OSPF routes back to static or use its own static route for
```

Exercise 5: Policy-Based Routing for Application-Aware WAN Path Selection

1. Traffic Classification Logic (Q1)

Flow Class	NS-3 Application	Packet Size/Interval	DSCP Tagging (Client n0)	PBR Policy (Router n1)
Flow_Video (RTP-like)	OnOffApplication	160 Bytes / 20ms	EF (Expedited Forwarding, 0x2e << 2)	Route via Primary Link (Net 2)

Flow_Data (FTP-like)	OnOffApplication	1500 Bytes / 500ms	BE (Best Effort, Default)	Route via Secondary Link (Net 3)

2. PBR Implementation (Q2)

Implementation Method: I would implement a custom Ipv4RoutingProtocol subclass, PbrRouting, that overrides the key function RouteOutput.
`PbrRouting.cc` (Core Logic): **Provided In the Repository**

3. Path Characterization for Dynamic PBR (Q3)

For a truly dynamic, SD-WAN-like engine, static DSCP-based PBR must be enhanced with real-time link metrics.

- Latency Measurement: Use FlowMonitor to periodically capture the End-to-End Delay (RTT) of small, low-volume probe packets sent over Net 2 and Net 3.
- Available Bandwidth/Congestion: Trace the queue occupancy (e.g., using a DropTailQueue trace sink) on the $n1$ interfaces for Net 2 and Net 3. High occupancy is a proxy for low available bandwidth.
- Availability: These metrics would be collected by a monitoring entity (Q4) and stored in a shared state accessible by the PBR function.

4. Dynamic Policy Engine (SD-WAN-like Controller) (Q4)

This would involve a custom SdWanController class running on n1.

Policy Rule Example: "If the latency on the primary path (Net 2) exceeds 30ms, all Flow_Video traffic must be immediately steered to the secondary path (Net 3) until the primary recovers."

The SdWanController would:

1. Schedule: Run a PolicyUpdate() method every 1 second.
2. Monitor: Read the latest latency values from the probe/FlowMonitor results.
3. Act: If the primary link fails the 30ms threshold, the controller uses the Ipv4RoutingProtocol::AddHostRouteTo method on n1 to dynamically change the next hop for the Video Flow's destination IP (e.g., 10.1.2.2) to use the Net 3 interface and next-hop address.

5. Validation and Trade-offs (Q5)

Validation Plan:

1. Trace: Enable logging on the custom PbrRouting class to log the DSCP value and the Chosen Interface Index for every forwarded packet on $n1$.
2. Verify Steering:
    - Video packets (DSCP EF) must consistently show Interface 2 selected.
    - Data packets (DSCP BE) must consistently show Interface 3 selected.
3. FlowMonitor: Verify that the RTT for the Video flow is lower than the RTT for the Data flow, demonstrating that the lower-latency path (Primary) was used.

Aspect	Computational Overhead (Simulator)	Scalability Limits
In Simulation	High. FlowMonitor is computationally expensive when run frequently. Periodic polling and modifying the routing table on the fly adds simulation overhead and complexity.	Limited. The simulation runtime increases exponentially with the number of flows being tracked or the frequency of the PolicyUpdate() function. This model is only feasible for small, controlled topologies (4-5 nodes).
Real Hardware	Low. The PBR logic (DSCP inspection) is handled by the high-speed data plane (ASICs) and has negligible overhead. The SD-WAN controller runs on a separate control plane CPU.	High. Real SD-WAN solutions scale well by aggregating policy decisions (e.g., applying the policy to a whole subnet or application type rather than individual flows) and optimizing metric collection.

Exercise 6: Inter-AS Routing Simulation for Multi-Provider WAN

1. Modeling Autonomous Systems and Peering (Q1)

Topology (Extension of Ex 4):

- AS65001 (ISP 1): n0 (Internal), n1 (Border Router/IXP-A)
- AS65002 (ISP 2):n2 (Border Router/IXP-A), n3 (Internal - NEW NODE)
- Links: Net 1 (n0 → n1), Net 2 (Internal n1), Net 3 (n1 → n2 - Peering Link), Net 4 (n2 →n3 - New Internal Link)

Intra-AS Routing: OSPF (Open Shortest Path First) must be run independently

within each AS (AS65001 on n0, n1 and AS65002 on n2, n3) to ensure all internal routers know the path to the respective Border Router. The Border Routers (n1, n2) use the BGP logic for external routes.

Q2. BGP Path Attribute Simulation: To model BGP, we cannot use simple `Ipv4Route` objects. We need a custom C++ struct that acts as the BGP Update message.

```
struct BgpRouteAdvertisement {
    Ipv4Address networkPrefix;
    Ipv4Mask subnetMask;
    std::vector<uint32_t> asPath;
    uint32_t localPref;
    uint32_t med;
    Ipv4Address nextHop;
};
```

Q3. Implementing Basic BGP Decision Process: The algorithm mimics the standard BGP best-path selection:

1. **Receive** new advertisement `newRoute`.
2. **Lookup** existing route for this prefix `currentRoute`.
3. **Compare:**
    - **Step 1 (Local Pref):** If `newRoute.localPref > currentRoute.localPref`, update table (New wins).
    - **Step 2 (AS Path):** If `newRoute.asPath.size() < currentRoute.asPath.size()`, update table (Shortest path wins).
    - **Step 3 (MED):** If equal, check MED (Lower is better).
4. **Install:** If `newRoute` wins, update the `Ipv4StaticRouting` table to point to `newRoute.nextHop`.

Q4. Simulating a Route Leak: A route leak occurs when an AS advertises a prefix it shouldn't.

- **Scenario:** AS65002 (Transit) accidentally advertises AS65001's own prefix (e.g., 1.1.1.0/24) *back* to AS65001.
- **Malicious Advertisement:** `Prefix: 1.1.1.0/24, AS_PATH: [65002,`

`65001]`.

- **Reaction:** The routers in AS65001 will receive this. However, the **Loop Prevention** mechanism in BGP checks the `AS_PATH`. Since AS65001 sees its own AS number in the path vector, it will **reject** the update to prevent a routing loop. This demonstrates the importance of AS_PATH attribute verification.

Q5. From Simulation to Reality: Our NS-3 model is a significant simplification:

1. **No TCP Sessions:** Real BGP runs over persistent TCP port 179 connections. Our simulation uses stateless packet passing, missing the complexity of session resets and keepalives.
2. **Convergence Delay:** Real BGP takes time to propagate updates across the internet (MRAI timers). Our simulation is likely instantaneous, ignoring the "route flap" dampening seen in real WANs.
3. **Lack of Policies:** We simulated basic attributes, but real BGP relies heavily on Route Maps, Community Strings, and prefix-lists which are complex string-matching operations not easily modeled in basic NS-3 C++.