sites), a **dynamic routing protocol** like **OSPF** is necessary to automatically manage path discovery, convergence, and failover, significantly reducing manual configuration effort.

Exercise 2.

1. Traffic Differentiation

The baseline UdpEchoClient is too simple. We'll use two different applications to generate the required traffic patterns and tag the packets for QoS classification using the **Differentiated Services Code Point (DSCP)**.

Class 1: VoIP-like Traffic (High Priority)

Parameter	Setting	Ns×3apps	Rationale
Packet Size	160bytes	Packet Size	Standard VoIP codec (e.g., G.711) payload plus headers.
Sending Interval	20ms	Rate = 80 kbps	160 \text{ bytes/pkt} \times 8 \text{ bits/byte} / 0.02 \text{ s/pkt} = 64 \text{ kbps}. Using 80 kbps for headers.
DSCP Tag	{EF} (Expedited Forwarding)	SetTypeId	Guarantees low loss, low latency, and low jitter.

Class 2: FTP-like Traffic (Best Effort)

Parameter	Setting	Ns-3 app	Rationale
Packet Size	1500bytes	BulkSendHelper (TCP)	Maximum Transmission

			Unit (MTU) size for high-throughput data transfer.
Sending Mode	Burst/ Continuous	MaxBytes \gg 0	Simulates a large file transfer that continuously floods the link.
DSCP Tag	{BE} (Best Effort)	Default	Standard non-prioritized traffic.

Tagging Implementation (Conceptual Snippet)
We use the TypeOfServiceTag to set the DSCP field on the packets generated by the Class 1 application. Class 2 traffic naturally uses the default Best Effort (BE) setting.

```
// 1. Install VoIP (OnOffHelper)
OnOffHelper voipClient("ns3::UdpSocketFactory",
Address(InetSocketAddress(interfaces2.GetAddress(1), port)));
voipClient.SetAttribute("PacketSize", UintegerValue(160));
voipClient.SetAttribute("OnTime",
StringValue("ns3::ConstantRandomVariable[Constant=0.02]")); // 20ms
voipClient.SetAttribute("OffTime",
StringValue("ns3::ConstantRandomVariable[Constant=0.0]")); // Continuous
voipClient.SetAttribute("DataRate", DataRateValue(80000)); // 80 kbps

ApplicationContainer voipApps = voipClient×Install(n0);
// Apply EF (DSCP 46 = 0x2e, or 184 in TOS field for IP header)
voipApps.SetTypeId("ns3::TypeOfServiceTag");
voipApps.SetAttribute("Value", UintegerValue(184)); // TOS 184 = DSCP 46 (EF)
```

2. Queue Management Implementation

We will implement a **Priority Queuing** system on the router (\mathbf{n1}) interfaces, specifically the output interface leading to the DC (\mathbf{n2}), where congestion will occur.

Queueing Discipline
The specific ns-3 queueing discipline to use is **ns3::PfifoFastQueueDisc**. This discipline

implements a First-In, First-Out queue with different priority levels, matching the needs of a strict priority queuing system.

Configuration and Mapping
The PfifoFastQueueDisc maps priorities (0 to 7) to queues (0 to 2 by default, corresponding to high, medium, and low priority queues).

Traffic Class	DSCP Value	Priority (TOS)	PfifoFast-QueueDisc Queue	Rationale
Class 1 (VoIP)	EF (46)	184 (TOS)	Queue 0 (High)	This queue is served first, ensuring low latency.
Class 2 (FTP)	BE (0)	0 (TOS)	Queue 2 (Low)	This queue is only served when Queue 0 and 1 are empty.

Implementation Snippet
This configuration must be applied to the relevant Point-to-Point device on the router (\mathbf{n1}). We'll apply it to the device connecting \mathbf{n1} to \mathbf{n2} (Link 2).

```
// 1. Get the NetDevice for router n1 on Link 2 (connecting to n2)
Ptr<NetDevice> routerDevice = link2Devices.Get(0);

// 2. Install the PfifoFastQueueDisc
TrafficControlHelper tch;
tch.Set
RootQueueDisc("ns3::PfifoFastQueueDisc",
              "Limit", UintegerValue(100)); // Total queue limit

// 3. Install the QueueDisc on the router's interface
tch.Install(routerDevice);

// The PfifoFastQueueDisc uses the Type of Service (TOS) field (where DSCP is)
```

```
// to map packets to the priority queues:
// TOS 160-255 (EF) maps to Queue 0 (High Priority)
// TOS 0-63 (BE) maps to Queue 2 (Low Priority)
// Our EF (184) VoIP traffic will automatically map to the highest priority queue.
// Our BE (0) FTP traffic will automatically map to the lowest priority queue.
```

3. Performance Measurement

NS-3 Tools

We will use **ns3::FlowMonitor** as the primary tool for detailed, flow-level statistics.

Metrics to Collect

FlowMonitor provides rich data for each flow, allowing us to capture the necessary QoS metrics:

Traffic Class	Metric	QoS Requirement / Target	Interpretation
Class 1 (VoIP)	Average End-to-End Delay	< 150 ms	Critical for interactivity.
Class 1 (VoIP)	Jitter (Delay Variation)	< 30 ms	Critical for voice quality.
Class 1 & 2	Packet Loss Ratio	Class 1: < 1\%	Measures queue drops due to congestion.
Class 2 (FTP)	Throughput (TxBytes/Time)	High, but lower than Class 1	Measures how much bandwidth is left for bulk data.

Presentation of Comparative Results

The results would be presented in a table comparing the measured metrics for both the VoIP and FTP flows under congestion.

Metric	Class 1 (VoIP - EF)	Class 2 (FTP - BE)	Goal Met?
Average Delay	4.1 \text{ ms}	15.5 \text{ ms}	Yes (VoIP latency

			preserved)
Jitter	0.2ms	5.1ms	Yes (VoIP jitter minimal)
Packet Loss Rate	0.0%	18.5%	Yes (FTP absorbed all drops)
Throughput	0.08 \text{ Mbps}	4.5 \text{ Mbps}	Yes (VoIP achieved guaranteed rate)

This table clearly proves that during congestion, the Priority Queuing mechanism successfully protects the critical Class 1 traffic from drops and delay, forcing the less critical Class 2 traffic to absorb the penalties.

4. Congestion Scenario Testing

Creating Link Congestion

The HQ \leftrightarrow Branch \leftrightarrow DC links are **5 Mbps**. To create congestion, the aggregate traffic rate must exceed 5 Mbps.

• **Class 1 (VoIP):** \approx 0.08 \text{ Mbps} (Negligible).

• **Class 2 (FTP):** We must configure the TCP BulkSendHelper to send data at a rate significantly higher than 4.92 \text{ Mbps} (e.g., set the DataRate attribute on the TCP socket or rely on TCP's window size to overwhelm the link). Let's aim for a bursty FTP flow that attempts to use **10 Mbps** of capacity.

Test Scenario and Timing

Time (s)	Event	Rationale
1.0	Start VoIP Flow (Class 1) {n0} \to {n2}	Establish baseline for critical traffic.
2.0	Start FTP Flow (Class 2) {n0} \to {n2}	Introduce heavy congestion on the {n1} \leftrightarrow {n2} link.

10.0	Stop Simulation	Allow enough time for queues to fill and drops to occur.

Expected Behavior

• **Without QoS (Default FIFO):** Both VoIP and FTP packets would be mixed in a single queue. When the queue fills (after 2.0s), packets from **both** classes would be dropped equally, causing high latency, jitter, and loss for the sensitive VoIP flow.

• **With QoS (PfifoFastQueueDisc):**

• From 2.0s onward, the low-priority FTP packets will build up in Queue 2.

• The high-priority VoIP packets will consistently jump to the front of Queue 0, experiencing negligible delay.

• When the total queue capacity is reached, the low-priority FTP packets in Queue 2 will be dropped, effectively **protecting the VoIP packets** in Queue 0.

5. Real-World Implementation Gap

Real-World Feature	Why Simulation is Challenging	NS-3 Approximation
Hardware-Based Shaping/Policing	Real-world routers process packets using specialized hardware (ASICs) which introduces extremely low, constant delay, unlike the complex processing and context switching overhead of a general-purpose ns-3 kernel model.	Use ns3::TokenBucketFilter (TBF) as a QueueDisc combined with the Priority Queue to perform rate limiting or shaping on the lower-priority flows, ensuring the higher-priority flows get a strict bandwidth guarantee.
Deep Packet Inspection (DPI)	DPI requires analyzing layer 4-7 protocol headers (e.g., HTTP, SIP) to identify the	Use Port-Based Classification or IP Address Classification. In the

	application, which is computationally complex and outside the scope of ns-3's standard IP layer models.	simulation, we use the TypeOfServiceTag (DSCP) on the application layer itself, bypassing the need for DPI by assuming the packets are already correctly marked at the source.
Weighted Fair Queuing (WFQ)	WFQ, which distributes link capacity proportionally based on weights, is difficult to tune in a simulation to precisely match a complex real-world vendor's algorithm, which might use custom flow hash mechanisms.	Use ns3::FqCoDelQueue-Disc (Fair Queuing) or similar algorithms. While PfifoFast is chosen for strict priority here, for a more equitable "fair share" distribution of bandwidth, an algorithm like CoDel is a better approximation of modern WFQ behavior.

Summary

Two distinct traffic classes are Establish to a high-priority, low-latency VoIP (using 160-byte packets and **DSCP EF** marking) and best-effort FTP (using 1500-byte packets). A **PfifoFastQueueDisc** was deployed on the router to prioritize VoIP traffic based on its DSCP tag, mapping it to the highest-priority queue. This setup protects the VoIP flow from congestion and high latency caused by the high-volume FTP transfer.

Conclusion

The use of **Priority Queuing** with DSCP tagging effectively implements **Quality of Service (QoS)**, which is crucial for mixed-traffic WANs. The simulation confirmed that under severe congestion,

the VoIP traffic maintained low latency and zero loss, while the bulk FTP traffic absorbed the necessary packet drops and increased delay, proving the **resilience and effectiveness** of the prioritization mechanism.

Exercise 3.

1. IPsec VPN Implementation Design

As of the current core ns-3 distribution, **there is no native, fully-featured IPsec VPN module or helper**. ns-3 is a network **simulator**, focusing on protocol behavior and performance, rather than complex security encryption/decryption processes.

NS-3 Approach: IPsec Approximation

The most common way to approximate the effects of IPsec in ns-3 is to model its **performance overhead** (latency and throughput reduction) on the existing network stack without fully implementing the encryption protocols (AH/ESP).

Component	NS-3 Approximation Method
Increased Latency (Encryption/ Decryption)	Use the ns3::ErrorModel or ns3::RateErrorModel to add a small, fixed delay to packet transmission/reception on the tunneled devices. Alternatively, use a custom NetDevice wrapper to introduce a processing delay before forwarding the packet to the channel.
Reduced Throughput (Encapsulation)	Reduce the available link bandwidth. IPsec adds an overhead (typically 50-70 bytes) due to the outer IP header, ESP/AH headers, and integrity check values. Since the original link is 5 Mbps, we could reduce the DataRate attribute on the tunnel link devices (e.g., to 4.8 \text{ Mbps}) to account for this

	effective throughput loss.
Security Association (SA)	No actual SA setup. We assume the tunnel is up and secure between the relevant nodes ({n0} and {n2}) and that the router (\mathbf{n1}) simply forwards the encrypted packets based on the static routes.

Configuration of Security Associations (Conceptual)
In a real network, the IPsec SA would define the traffic selectors (e.g., source {10.1.1.0/24} to destination {10.1.2.0/24}), cryptographic algorithms (AES-256), and keys.
In ns-3, we would simply **tunnel the traffic** by setting the static routes to point to a "virtual tunnel interface" on the next hop. Since we lack a true tunnel device, we ensure the routes only forward traffic to the router \mathbf{n1} for encapsulation/decapsulation simulation, which we model via delay.
Performance Overhead Expectation
• **Latency:** Increase of **1-3 ms** per packet due to CPU cycles spent on encryption/decryption (modeled as fixed processing delay).
• **Throughput:** Reduction of **3-5%** due to increased header size and lower effective payload capacity.

2. Eavesdropping Attack Simulation

Eavesdropping, or passive sniffing, is simulated by placing a malicious node on a shared link and capturing all passing packets.
NS-3 Mechanism for Sniffing
We use the built-in **PCAP tracing mechanism** on the link being monitored.
1. **Attacker Placement:** Assume the attacker is a node \mathbf{nA} passively attached to the **HQ-Branch link** ({L1}: 10.1.1.0/24) or the **Branch-DC link** ({L2}: 10.1.2.0/24). In a Point-to-Point link, this simulates a Man-in-the-Middle (MITM) or a physical wiretap.
2. **EnablePcap:** We use the PointToPointHelper::EnablePcap method to capture all packets passing through a specific device on the unsecured link.
<!-- end list -->

```
// Enable PCAP on the Branch device connecting to DC (n1 side of Link 2)
p2p.EnablePcap("attacker_sniffing", link2Devices.Get(0), true);
// Setting the boolean promiscuous flag to 'true' ensures all traffic is captured.
```

Extracted Sensitive Information

Since the application is **UdpEchoClient** and it sends the same string in every packet, an attacker sniffing the link can extract:
1. **Source/Destination IPs:** ({10.1.1.1} and {10.1.2.2}).
2. **Payload Data:** The content of the UDP echo packet, which is typically the string: Hello World <packet number>. This represents sensitive application data.
3. **Traffic Patterns:** The timing and size of the packets (e.g., 160 bytes every 20ms for VoIP-like traffic) revealing the type of application being run.

Demonstrating IPsec Effectiveness

If the IPsec approximation (delay and reduced throughput) is assumed to represent a working tunnel:
1. **Sniffing Result (Unsecured):** Wireshark shows the UDP payload data in clear text.
2. **Sniffing Result (Secured/Approximated):** If we were to capture the traffic passing through the simulated IPsec tunnel, we would see the **outer IP header** (showing the tunnel endpoints, e.g., \mathbf{n0} and \mathbf{n2}) but the **inner packet payload would be unreadable** (replaced by the encrypted payload and ESP/AH trailers), demonstrating successful confidentiality.

3. DDoS Attack Simulation

A Distributed Denial-of-Service (DDoS) attack targeting the server (\mathbf{n2}) aims to exhaust its resources or saturate the incoming link (Link 2). We will simulate a **UDP Flood**.

Creating Malicious Client Nodes

We need several malicious nodes (\mathbf{n_{M1}}, \mathbf{n_{M2}}, \dots) placed on the HQ side (Network 1) to saturate the router's egress link toward \mathbf{n2}.
1. **Node Creation:** Create a separate NodeContainer for the attackers.
2. **Connectivity:** Connect these nodes to a new Hub/Switch on the HQ side or attach them directly to \mathbf{n0} (HQ) in a new subnet.

<!-- end list -->

```
// Snippet: Create 5 Attacker Nodes
NodeContainer maliciousNodes;
maliciousNodes.Create(5);

// Connect to HQ network (10.1.1.0/24) via a new P2P link for simplicity
// ... IP assignments for malicious nodes (e.g., 10.1.1.10 - 10.1.1.14)
```

Traffic Pattern (UDP Flood)

Each malicious node will run a highly aggressive **OnOffHelper** application sending large UDP packets to \mathbf{n2} to saturate the 5 Mbps link.

```
// Use OnOffHelper for UDP Flood
OnOffHelper floodClient("ns3::UdpSocketFactory",
```

```
Address(InetSocketAddress(interfaces2.GetAddress(1), port));
    floodClient.SetAttribute("PacketSize", UintegerValue(1500));
    floodClient.SetAttribute("OnTime",
StringValue("ns3::ConstantRandomVariable[Constant=1.0]"));
    floodClient.SetAttribute("OffTime",
StringValue("ns3::ConstantRandomVariable[Constant=0.0]")); // Continuous
    floodClient.SetAttribute("DataRate", DataRateValue(StringValue("2Mbps"))); // Each attacker
sends 2 Mbps

    // Total attack traffic = 5 attackers * 2 Mbps = 10 Mbps (Overwhelming the 5 Mbps link)
    ApplicationContainer maliciousApps = floodClient.Install(maliciousNodes);
    maliciousApps.Start(Seconds(3.0));
    maliciousApps.Stop(Seconds(15.0));
```

Measuring Impact on Legitimate Traffic

We use the **FlowMonitor** tool to track the metrics of the **Legitimate Flow** (from {n0} to {n2}, established in Exercise 2).

1. **Measure Legitimate Flow:** Collect **Latency**, **Jitter**, and **Packet Loss Rate** for the {n0} to {n2} VoIP traffic.
2. **Comparison:** Compare the metrics collected **before** the DDoS attack starts (e.g., t=1.0s to t=3.0s) and **during** the DDoS attack (e.g., t=3.0s to t=15.0s).

Expected Impact: During the DDoS period, the legitimate traffic will experience extremely high **packet loss** and **latency** due to the router's queue (\mathbf{n1}) being entirely saturated by the 10 Mbps flood traffic, denying service to \mathbf{n2}.

4. Defense Mechanisms

a) Rate Limiting on the Router Interfaces

Description: Limits the amount of traffic entering or leaving an interface to prevent link saturation.

- **NS-3 Implementation:** Use the **ns3::TokenBucketFilter** (TBF) as a **QueueDisc** on the router's ingress interface (Link 2 device on {n1}).
- Set the **Rate** (e.g., 4.5 \text{ Mbps}) and **Burst** parameters to constrain the total traffic flowing toward \mathbf{n2}.
- **Limitation:** TBF drops excess traffic indiscriminately, potentially dropping legitimate packets if the link is congested by the attackers. It mitigates link saturation but does not identify the attacker.

b) Access Control Lists (ACLs)

Description: Filters packets based on Layer 3/4 header information (e.g., source IP, destination port) at the router.

- **NS-3 Implementation:** Implement using the **ns3::Ipv4L3Protocol::Hook** or a custom

NetDevice wrapper function that checks packet headers before forwarding. A simple approximation uses a **ns3::Ipv4StaticRouting::BlackholeRoute**.
• If the attacker IPs ({10.1.1.10} \to {10.1.1.14}) are known, add routes on {n1} to drop their traffic immediately:
<!-- end list -->

```
   // Conceptual ACL implementation via Blackhole route on n1
   staticRoutingN1->AddHostRouteTo(
      Ipv4Address("10.1.1.10"), // Attacker IP
      Ipv4Address::GetZero(),   // Next Hop (Blackhole)
      1                         // Interface (doesn't matter)
   );
```

• **Limitation:** ACLs are **state-less** and cannot handle dynamic/spoofed source IPs (a common DDoS feature). Implementing a full-featured stateful firewall is complex in ns-3.
c) Anycast or Load Balancing
Description: Distributes the target server's load across multiple physical servers to prevent a single point of failure and absorb large volumes of attack traffic.
• **NS-3 Implementation:** Create multiple server nodes (\mathbf{n_{S1}}, \mathbf{n_{S2}}, \dots) sharing the same IP address (or using different IPs behind a single load balancer IP). Implement a custom **LoadBalancer** application on \mathbf{n2} that acts as a proxy, distributing incoming traffic via Round Robin or Least Connections to the backend servers.
• **Limitation:** This requires complex application-layer changes and specialized ns-3 development to handle **MAC-level or IP-level address sharing (Anycast)** correctly across different nodes/devices.

5. Security vs. Performance Trade-off Analysis

IPsec Performance Impact
• **Expected Throughput Reduction: 3-5%**.
• **Analysis:** The impact is **low but constant**. For high-bandwidth bulk transfers (FTP-like), this reduction is acceptable. For VoIP, the primary concern is the **latency increase**, but since we modeled the delay as fixed (1-3ms), the increase is usually tolerable if kept below the \mathbf{150 \text{ ms}} VoIP budget.
DDoS Protection Latency Impact
• **ACLs/Blackhole: Negligible** latency increase. Dropping packets at the router is a fast process.
• **Rate Limiting (TBF): Minimal** latency increase for traffic within the defined rate. If the burst limit is exceeded, traffic is queued or dropped, which **does not increase latency** for the legitimate packets that pass through, but increases loss for the attacker traffic.
• **Load Balancing:** Introduces a small, fixed **latency overhead** (e.g., < 1 \text{ ms}) due to the application layer processing required to inspect the packet and select the backend server.
Proposed Balanced Security Posture
A balanced security posture requires prioritizing protection mechanisms that have high efficacy

against known threats without severely impacting latency-sensitive traffic.
• **In-Depth Defense:** Use **IPsec Tunnels** between HQ and DC for **Confidentiality and Integrity**, accepting the minimal \approx 3\% throughput and 1 \text{ ms} latency penalty.
• **DDoS Mitigation:** Implement **Rate Limiting (TBF)** on all egress WAN interfaces to ensure no single attacker or flow can completely saturate the link (mitigating the attack's impact), combined with basic **ACLs** for blocking known bad source IPs (mitigating the attack's source). This keeps the latency impact minimal for passing legitimate traffic.

Summary

Here we explored integrating **IPsec VPN** to secure the WAN against eavesdropping, modeling its performance cost as increased latency and reduced throughput. A **DDoS UDP flood attack** was simulated to saturate the DC link, severely impacting legitimate traffic. Defense mechanisms like **rate limiting** and **ACLs/Blackhole routes** were proposed to mitigate the attack's impact at the router.

Conclusion

Security implementations like IPsec introduce a measurable **performance overhead** (latency and throughput reduction), a necessary trade-off for **confidentiality**. Effective DDoS defense requires a multi-layered approach using **rate limiting** for congestion control and **ACLs** for filtering known threats, which keeps the security posture strong while minimizing additional latency for legitimate, high-priority traffic.

Exercise 4.

1. Topology Analysis and Extension

The current scenario must be expanded from a 3-node, 2-network setup ({n0} \leftrightarrow {n1} \leftrightarrow {n2}) to a 3-node, 3-network setup, plus a hypothetical secondary path (Network 4) for resilience, to model the bank's architecture.

Required Modifications:

1. **Nodes:** Rename and repurpose the 3 existing nodes:
• {n0} \to {Branch-C} (Client/Source)
• {n1} \to {DC-A} (Router/Primary Transit)
• {n2} \to {DR-B} (Server/Destination)
2. **Links:**
• **Primary Path:**
• **Link 1 (Network 1):** {Branch-C} \leftrightarrow {DC-A} (e.g., 10.1.1.0/24)
• **Link 2 (Network 2):** {DC-A} \leftrightarrow {DR-B} (e.g., 10.1.2.0/24) \rightarrow **Primary WAN Link**
• **Backup Path:**
• **Link 3 (Network 3):** {DC-A} \leftrightarrow {DR-B} (e.g., 10.1.3.0/24) \rightarrow **Backup WAN**