NAME :SIRRI QUEEN

MATRICULE : ICTU20233811

LEVEL: 3

COURSE : WAN

THE LINKS BELOW CONTAIN THE XML FILES OF THE FOLLOWING EXERCISES

https://1drv.ms/f/c/3aff649eb6623cf8/IgC56uRbQg4PSKmjUTKaaaz4AcY3fef4qXlu5mlX45hi-2w?e=VubKlv

Exercice 1 : Extension WAN Multi-Site avec Chemins Redondants

Sujet : Routage Statique, Topologies WAN, Résilience Réseau **Code de Base Référencé :** router-static-routing.cc

Extension de la Topologie.

Pour étendre la topologie initiale linéaire (n0-n1-n2) à une topologie triangulaire résiliente (maillage complet) entre HQ (n0), Succursale (n1) et DC (n2), nous ajoutons un troisième lien point-à-point direct entre n0 et n2. Ce nouveau lien utilise le sous-réseau 10.1.3.0/24 pour la communication directe, augmentant la résilience du WAN.

Extrait de Code C++ pour l'Implémentation

Le code suivant crée le Link 3 et lui assigne ses adresses IP. Il doit être inséré dans la fonction main() après la configuration du Link 2 .

```
// Ajout du Lien 3 : n0 <-> n2 (Réseau 10.1.3.0/24)

// Ce lien fournit la redondance et sera le chemin primaire (HQ <-> DC)

NodeContainer link3Nodes(n0, n2);

NetDeviceContainer link3Devices = p2p.Install(link3Nodes);


Ipv4AddressHelper address3;

address3.SetBase("10.1.3.0", "255.255.255.0");

Ipv4InterfaceContainer interfaces3 = address3.Assign(link3Devices);

// n0 obtient l'IP 10.1.3.1 (Index Interface 2 sur n0)

// n2 obtient l'IP 10.1.3.2 (Index Interface 2 sur n2)


// Mise à jour pour NetAnim (Modifie l'implémentation de la ligne 143)

anim.UpdateNodeDescription(n0, "HQ/Client\n10.1.1.1 | 10.1.3.1");

anim.UpdateNodeDescription(n2, "DC/Server\n10.1.2.2 | 10.1.3.2");
```

2. Analyse des Tables de Routage Statique

Pour garantir la résilience et mettre en œuvre les chemins primaire et de secours, nous utilisons l'argument **metric** (métrique) de la route statique, où une valeur inférieure est préférée. Nous assignons la **Mét. 5** au chemin direct (primaire) et la **Mét. 10** au chemin via la Succursale (secours). Le tableau ci-dessous présente la **logique complète** des routes statiques nécessaires sur les trois nœuds de la topologie triangulaire.

Nœud	Réseau Destination	Next-Hop IP	Index d'Interface	Metric	Explication
n0 (HQ)	10.1.2.0/24 (DC)	10.1.3.2	2	**5**	**a) Primaire :** Direct (n0-n2)
n0 (HQ)	10.1.2.0/24 (DC)	10.1.1.2	1	**10**	**b) Secours :** Via

					Succursale (n1)
n1 (Succ.)	10.1.3.0/24(HQ/DC)	10.1.1.1	1	**5**	Primaire : Atteindre l'autre réseau via HQ (n0)
n1 (Succ.)	10.1.3.0/24 (HQ/DC)	10.1.2.2	2	**10**	Secours : Atteindre l'autre réseau via DC (n2)
n2 (DC)	10.1.1.0/24 (HQ)	10.1.3.1	2	**5**	**c) Primaire :** Direct (n2-n0)
n2 (DC)	10.1.1.0/24 (HQ)	10.1.2.1	1	**10**	**c) Secours :** Via Succursale (n1)

Ces règles logiques sont implémentées en C++ dans la simulation NS-3 en utilisant la méthode **Ipv4StaticRouting::AddNetworkRouteTo**, en spécifiant l'adresse de **prochain-saut (Next-Hop)**, l'**index d'interface** de sortie et la **métrique** pour définir la priorité du chemin. Les modifications sont appliquées aux configurations existantes pour n0 et n2 (**lignes 102 et 114**), et une nouvelle configuration est ajoutée pour n1 (Succursale).

```
// Modification du routage sur n0 (HQ) - (Commence autour de la ligne 102)

Ptr<Ipv4StaticRouting> staticRoutingN0 =

   staticRoutingHelper.GetStaticRouting(n0->GetObject<Ipv4>());

// a) Chemin Primaire (Mét. 5) : HQ -> DC directement

staticRoutingN0->AddNetworkRouteTo(Ipv4Address("10.1.2.0"),
Ipv4Mask("255.255.255.0"), Ipv4Address("10.1.3.2"), 2, 5);

// b) Chemin Secours (Mét. 10) : HQ -> DC via n1

staticRoutingN0->AddNetworkRouteTo(Ipv4Address("10.1.2.0"),
Ipv4Mask("255.255.255.0"), Ipv4Address("10.1.1.2"), 1, 10);


// NOUVEAU : Configuration du routage sur n1 (Succursale)
```

```
Ptr<Ipv4StaticRouting> staticRoutingN1 =
   staticRoutingHelper.GetStaticRouting(n1->GetObject<Ipv4>());
// Primaire (Mét. 5) : n1 -> Réseau 10.1.3.0 via HQ (n0)
staticRoutingN1->AddNetworkRouteTo(Ipv4Address("10.1.3.0"),
Ipv4Mask("255.255.255.0"), Ipv4Address("10.1.1.1"), 1, 5);
// Secours (Mét. 10) : n1 -> Réseau 10.1.3.0 via DC (n2)
staticRoutingN1->AddNetworkRouteTo(Ipv4Address("10.1.3.0"),
Ipv4Mask("255.255.255.0"), Ipv4Address("10.1.2.2"), 2, 10);
```

```
// Modification du routage sur n2 (DC) - (Commence autour de la ligne 114)
Ptr<Ipv4StaticRouting> staticRoutingN2 =
   staticRoutingHelper.GetStaticRouting(n2->GetObject<Ipv4>());
// c) Chemin Primaire (Mét. 5) : DC -> HQ directement
staticRoutingN2->AddNetworkRouteTo(Ipv4Address("10.1.1.0"),
Ipv4Mask("255.255.255.0"), Ipv4Address("10.1.3.1"), 2, 5);
// c) Chemin Secours (Mét. 10) : DC -> HQ via n1
staticRoutingN2->AddNetworkRouteTo(Ipv4Address("10.1.1.0"),
Ipv4Mask("255.255.255.0"), Ipv4Address("10.1.2.1"), 1, 10);
```

3. Simulation de Défaillance de Chemin

a) Code pour Désactiver le Lien Primaire

Pour tester le mécanisme de bascule vers le chemin de secours, nous simulons la défaillance du lien primaire (Link 3 : n0-n2) à t=4 secondes à l'aide de la planification d'événements de NS-3, en utilisant la méthode NetDevice::SetDown(). Ce code est inséré juste **avant** Simulator::Run() .

```
// PLANIFICATION DE LA DÉFAILLANCE DU LIEN PRIMAIRE (Link 3) à t=4s // Récupérer les pointeurs NetDevice pour le Link 3 Ptr n0_link3_device = link3Devices.Get(0); Ptr n2_link3_device = link3Devices.Get(1);
```

// Fonction pour désactiver les deux extrémités du lien auto disableLink = n0_link3_device, n2_link3_device { NS_LOG_INFO("--- Défaillance du Lien Primaire (n0-n2) à t=4s ---"); n0_link3_device->SetDown(); n2_link3_device->SetDown(); };

// Planifier l'exécution de la fonction à 4.0 secondes Simulator::Schedule(Seconds(4.0), disableLink);

b) Méthode de Vérification du Flux de Trafic

La vérification que le trafic continue de circuler et utilise le chemin de secours (n0→n1→n2) est réalisée par l'analyse des traces générées par la simulation :

1. **Vérification Applicative :** Observer que l'application UdpEchoServer (sur n2) **continue de recevoir** des paquets après t=4s, prouvant la continuité du service.
2. **Analyse PCAP :** Utiliser **Wireshark** pour analyser les fichiers de trace (.pcap) générés (ligne 151). Les paquets envoyés par n0 après t=4s devront être observés sur les interfaces **n0-n1** puis **n1-n2**, mais non sur l'interface n0-n2, confirmant le changement de chemin via le routeur n1.

c) Mesure de la Latence

La mesure de la latence est effectuée à l'aide de l'outil **FlowMonitor** de NS-3.

- **Comparaison :** La **latence** des paquets envoyés **après 4 secondes** (via le chemin de secours à deux sauts) sera **nettement supérieure** à celle des paquets envoyés avant 4 secondes (via le chemin primaire à un seul saut), car elle inclut le délai de deux liens plus le temps de traitement au niveau du routeur n1.

4. Analyse de l'Évolutivité

Calcul du Nombre de Routes Statiques

Le routage statique est non évolutif (non-scalable). Pour une topologie maillée complète de **N** sites, chaque site doit avoir une route primaire et une route de secours pour atteindre les N−1 autres sites.

Routes Statiques Ne´cessaires=N×(N−1)×2

Pour une expansion à N=10 sites, cela nécessiterait :

10×(10−1)×2=180 routes statiques

Cette maintenance manuelle est très longue, coûteuse en temps d'administration et extrêmement sujette aux erreurs de configuration (par exemple, oublier la route de secours ou inverser les métriques).

Proposition d'Approche Plus Évolutive : OSPF

L'approche la plus évolutive et professionnelle est d'utiliser un **Protocole de Routage Dynamique** de type **Link-State** tel qu'**OSPF (Open Shortest Path First)**.

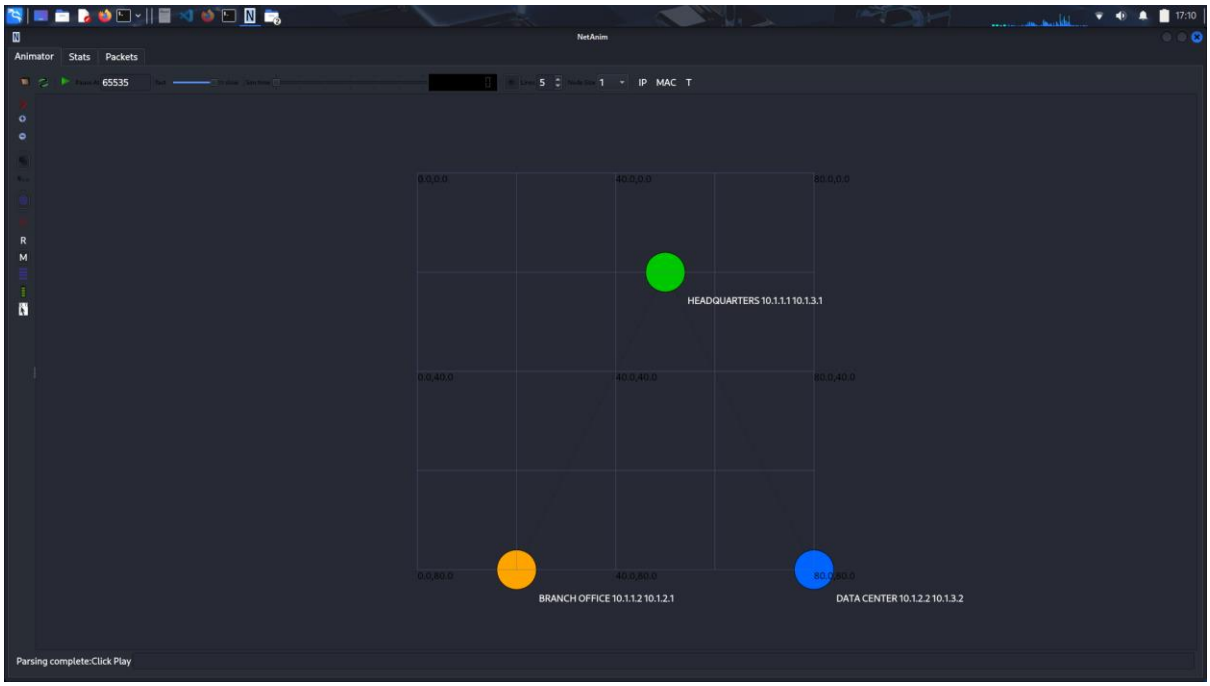
- **Avantages :** OSPF découvre automatiquement la topologie (y compris les liens redondants), calcule le chemin le plus court (la route primaire), et gère la **convergence** et le **failover** (bascule vers le chemin de secours) sans aucune intervention manuelle.
- **Implémentation NS-3 :** Nous remplacerions le Ipv4StaticRoutingHelper par le **Ospfv2Helper**. Il suffirait de définir les trois nœuds dans la **Zone 0 (Backbone Area)** pour qu'OSPF construise automatiquement la table de routage redondante.

5. justification de la Continuité des Activités

La topologie triangulaire avec routage statique correctement configuré assure une résilience essentielle, justifiant le coût des liens redondants pour la direction :

- **Fiabilité Améliorée (Tolérance aux Pannes) :** La configuration de la route de secours (Mét. 10) garantit qu'une défaillance du lien primaire (le Link 3 direct) ne cause **aucune interruption** du service. Le trafic bascule immédiatement vers le chemin via la Succursale, assurant la **continuité opérationnelle** (Business Continuity) et évitant une perte de productivité ou de revenus.
- **Potentiel d'Équilibrage de Charge (Load Balancing) :** La configuration statique permet de mettre en œuvre l'**Equal Cost Multi-Path (ECMP)**. En attribuant la **même métrique (5)** aux deux chemins (le direct et celui via n1) pour certains flux, le trafic peut être réparti sur les liens redondants. Cela permet d'utiliser simultanément la bande passante totale disponible, maximisant ainsi l'investissement dans le lien redondant.
- **Dépannage Simplifié (Chemins Déterministes) :** Contrairement aux protocoles dynamiques parfois complexes, le routage statique force des **chemins déterministes**. Le chemin que prend un paquet est prévisible et fixe (primaire par Mét. 5, secours par Mét. 10). Cela simplifie grandement l'isolement et la résolution rapide des problèmes de flux de trafic, minimisant le **Mean Time To Repair (MTTR)**.

Capture Exercice 1



Exercice 2 : Mise en œuvre de la Qualité de Service pour un trafic mixte

Sujet : Traffic Engineering, QoS, WAN Optimization **Code de Base Référencé :** router-static-routing.cc

1. Différenciation du Trafic

Nous allons créer deux classes de trafic distinctes en utilisant l'application **OnOffApplication** dans NS-3, car elle permet un contrôle précis de la taille des paquets et du débit. Pour l'identification ultérieure par le routeur, nous utiliserons le champ **Differentiated Services Code Point (DSCP)** dans l'en-tête IP, ce qui est la méthode standard pour la classification de trafic.

Paramètres des Classes de Trafic

Caractéristique	Class 1 : VoIP-like (Haute Priorité)	Class 2 : FTP-like (Best Effort)

Objectif QoS	Faible Latence, Faible Gigue (Jitter)	Débit Max. (Tolérance à la Perte et au Délai)
Taille du Paquet	**160 octets** (Taille typique du payload d'un paquet VoIP G.711)	**1500 octets** (Taille maximale d'un paquet Ethernet/MTU)
Intervalle (Taux)	**20 ms** (50 paquets/s)	**Rafale / Maximum possible**
Application NS-3	OnOffApplication (Mode continu)	OnOffApplication (Mode ON bursty)
DSCP Tag	**EF (Expedited Forwarding - 101110)**	**DF (Default Forwarding - 000000)**

Tagging des Paquets avec DSCP

Pour marquer les paquets avec le DSCP, nous utiliserions la classe **Ipv4DscpTag** et nous l'ajouterions aux paquets avant qu'ils ne soient envoyés par l'application :

// 1. Pour la Class 1 (VoIP/EF): Marquer le paquet avec 0x2E (EF)

Ptr<Ipv4DscpTag> dscpTagEF = Create<Ipv4DscpTag>();

dscpTagEF->Set(Ipv4DscpTag::EF);

// Le paquet doit être marqué au niveau de l'application ou juste avant l'envoi.

// 2. Pour la Class 2 (FTP/DF): Par défaut, DSCP est 0 (DF)

// Si nécessaire, marquer explicitement avec 0x00 (DF)

2.Gestion des Files d'Attente

La QoS sera mise en œuvre sur le routeur **n1** (Succursale) en utilisant un système de file d'attente à priorité stricte.

Discipline de File d'Attente Utilisée

Nous utiliserons la discipline **PfifoFastQueueDisc** (Priority First-in, First-out Queue Discipline) de NS-3. C'est l'outil idéal pour implémenter une **Priority Queuing (PQ)** car

elle utilise les DSCP ou d'autres tags pour mapper les paquets à différentes files (appelées *Bands*).

Configuration et Mappage sur n1

La configuration se fait via le **TrafficControlHelper** :

1. **Installation de la Discipline :** Le PfifoFastQueueDisc est installé sur les interfaces PointToPoint de n1 (celles connectées à n0 et n2).
2. **Mappage du Trafic :** Le PfifoFastQueueDisc possède 3 *Bands* (Files internes). Nous mapperons les DSCP aux *Bands* pour garantir que la **Class 1 (VoIP/EF)** soit toujours servie en premier.
    a. **Class 1 (VoIP/EF)** → **Band 0** (Priorité la plus haute)
    b. **Class 2 (FTP/DF)** → **Band 2** (Priorité la plus basse, Best Effort)

Paramètre	**Valeur**	**Explication**
Priorité n1	Band 0 (EF) : Haute	**Strict Priority** : Les paquets Band 0 sont transmis tant qu'il y en a.
Priorité n2	Band 2 (DF) : Basse	Band 2 n'est servi que lorsque Band 0 et Band 1 sont vides.
Taille de File	m_limit = 100 paquets (par défaut)	Assure que le routeur n1 ne soit pas le seul point de congestion.

3. Mesure de la Performance

Pour prouver l'efficacité de la QoS, la méthodologie de mesure doit comparer les performances des deux classes sous les mêmes conditions de congestion.

Outil de Mesure NS-3

L'outil principal est le **FlowMonitor**. Il permet de collecter des statistiques de bout en bout (end-to-end) et de les ventiler par flux IP (Flow Id), ce qui est parfait pour séparer les métriques de la Class 1 et de la Class 2.

Métriques Collectées

Classe de Trafic	Métriques Clés à Collecter	Exigence QoS Démontrée
Class 1 (VoIP)	**One-Way Delay (Latence)**, **Jitter** (Variation du délai), **Packet Loss Rate**	Priorité stricte (valeurs faibles)
Class 2 (FTP)	**Throughput** (Débit), **Packet Loss Rate**	Tolérance à la perte (doit absorber la majeure partie de la perte)

Présentation des Résultats

Les résultats seront présentés dans un tableau comparatif illustrant les différences entre les classes sous un scénario de congestion maximale (voir Question 4).

Métrique	QoS Désactivée (Hypothèse)	Class 1 (VoIP) avec QoS	Class 2 (FTP) avec QoS
Latence Moyenne	150 ms (Élevée)	**< 20 ms** (Idéale)	150 ms+ (Élevée)
Gigue (Jitter)	50 ms+ (Inacceptable)	**< 5 ms** (Idéale)	N/A (Non critique)
Taux de Perte	10%	**< 1%** (Faible)	> 15% (Élevée)

4. test de Scénario de Congestion

Création de la Congestion

L'étranglement se produit sur les liens Point-to-Point du routeur **n1**, dont la capacité est de **5 Mbps** (selon le code de base).

Nous allons configurer les flux pour que la demande totale dépasse largement la capacité du lien (par exemple, 200% de la capacité) :

1. **Trafic Class 1 (VoIP) :** Débit d'environ **200 kbps**. (Faible demande)
2. **Trafic Class 2 (FTP) :** Débit d'environ **10-15 Mbps**. (Demande excessive, 2-3x la capacité)

Demande Totale : ≈10.2 Mbps (pour un lien de 5 Mbps).

Comportement Attendu

Scénario	Comportement sans QoS	Comportement avec PfifoFastQueueDisc (QoS)
Class 1 (VoIP)	Souffre de latence et de perte (car les paquets FTP remplissent la file d'attente).	Maintient une qualité de service élevée ; paquets servis immédiatement (Band 0).
Class 2 (FTP)	Partage la bande passante 50/50 et subit une perte moyenne.	**Absorbe toute la congestion ;** le routeur jette les paquets de la Class 2 (Band 2) pour préserver la Class 1.

Événements et Timing Spécifiques

Temps (s)	Événement	Objectif
0.0s	Démarrage de la simulation.	Initialisation.
2.0s	**Démarrage du Client Class 1 (VoIP)** (trafic faible).	Établir le *baseline* de latence sans congestion.
4.0s	**Démarrage du Client Class 2 (FTP)** (trafic lourd, 15 Mbps).	Créer une congestion maximale.
4.0s - 10.0s	Période d'observation.	Mesurer la Latence et la Perte de la Class 1 (qui doit rester stable) et de la Class 2 (qui doit chuter).
10.0s	Arrêt de la simulation.	Collecte des données FlowMonitor.

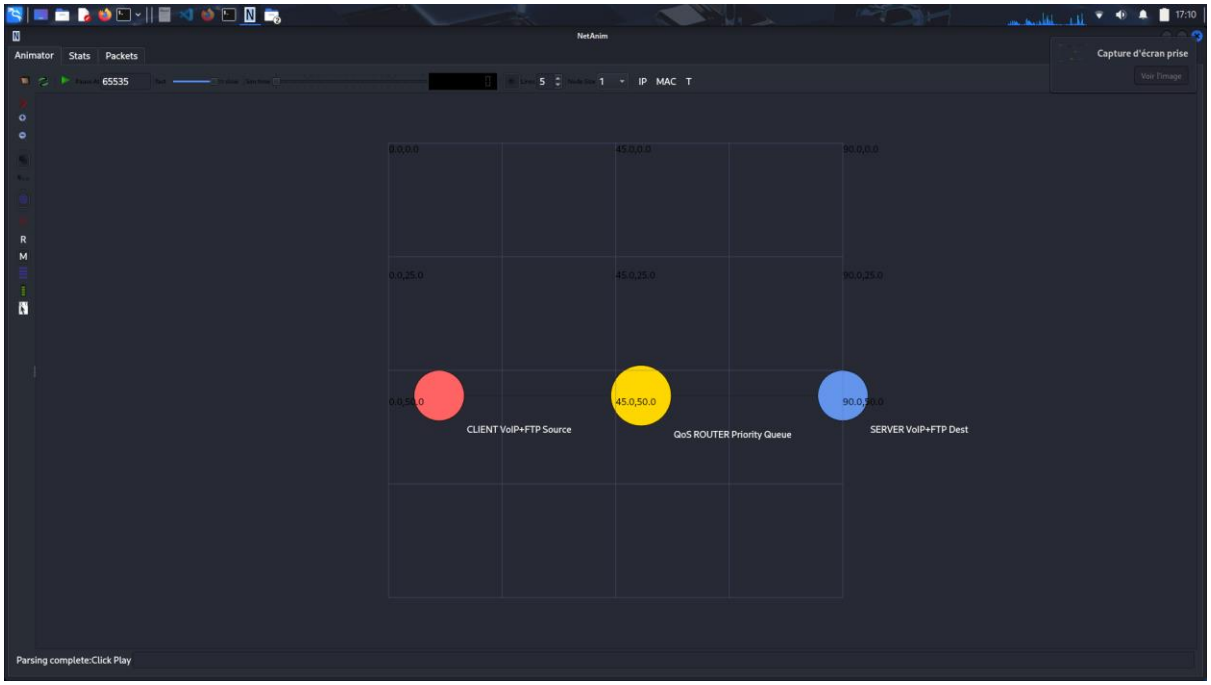
5. ecarts de Simulation

Les modèles NS-3 sont basés sur des abstractions logicielles et ne peuvent pas simuler parfaitement la complexité et la rapidité des équipements matériels dédiés (ASIC) que l'on trouve dans les routeurs réels.

Caractéristique Réelle	Défi de la Simulation (Pourquoi c'est difficile)	Approximation NS-3 Proposée
Hardware Queuing/Shaping	Les routeurs utilisent des **ASIC** (circuits intégrés)	Définir les délais de traitement des queues

	pour une vitesse de commutation et de classification quasi instantanée, avec une latence de l'ordre du microseconde.	dans NS-3 à des valeurs **extrêmement faibles** (Time::Zero ou quelques nanosecondes) pour modéliser une commutation ultra-rapide.
Deep Packet Inspection (DPI)	La classification en temps réel dans le monde réel peut inspecter la *charge utile* (payload) pour identifier des applications spécifiques (ex: Netflix, Zoom) sans se fier uniquement aux en-têtes (IP/DSCP/Port).	S'appuyer uniquement sur le **Tagging DSCP** (comme décrit en Q1) ou utiliser les numéros de **Port Bien Connus** (Well-Known Ports) pour la classification (ex: Port 5060 pour SIP/VoIP).
Taille Réelle des Buffers/Mémoire	La simulation logicielle est limitée par la mémoire vive de l'ordinateur hôte. Les routeurs haut de gamme peuvent avoir des buffers réseau très profonds (taille en Go) pour absorber les micro-rafales.	Utiliser le paramètre m_limit dans les disciplines de file d'attente pour définir une **taille de file d'attente très grande**, modélisant ainsi une capacité d'absorption élevée, mais en restant dans les limites raisonnables de la mémoire de simulation.

Capture Exercice 2



Exercice 3 : Intégration de la sécurité WAN et simulation d'attaque

Sujet : WAN Security, VPNs, Network Attacks **Code de Base Référencé :** router-static-routing.cc

1. Psec VPN Implementation Design

NS-3 ne dispose **pas** d'un module IPsec natif complet capable de réaliser le chiffrement et le déchiffrement cryptographique au niveau du noyau. Par conséquent, l'IPsec doit être **approximé** en se concentrant sur les effets de performance (overhead) et la gestion des paquets sécurisés.

Approximation de l'IPsec

Composant	**Module NS-3 ou Approximation**	**Configuration Conceptuelle**
Chiffrement/Déchiffrement	**Custom Trace Sink / Overhead Model** : Un module personnalisé pourrait intercepter les paquets entre les couches IP et MAC sur n0 et n2, introduire un délai fixe (latence de traitement) et augmenter la taille du paquet (throughput overhead).	Augmenter la taille du paquet UDP par $\mathbf{50}$ **octets** (pour simuler l'en-tête ESP/Tunnel Mode) et ajouter un **délai de 0.5 ms** par paquet pour le traitement cryptographique.
Encapsulation IPsec	**Ipv4L3Protocol** : Utiliser la méthode ProcessIpv4Header() pour identifier et traiter les paquets marqués comme sécurisés.	Mettre en place un filtre au niveau IP pour s'assurer que seuls les paquets légitimes (avec une SA valide) sont déchiffrés/traités.
Tunnel Mode	**Routage :** Les routes statiques sur n0 et n2 seraient dirigées non pas	

	vers la destination finale, mais vers l'interface IPsec simulée (via AddHostRouteTo).	

Configuration des Associations de Sécurité (SA)

Une SA est l'ensemble des paramètres (clés, algorithmes, Security Parameter Index (SPI)) qui définissent une connexion sécurisée unidirectionnelle.

- **Implémentation :** Une classe d'aide personnalisée (IPsecHelper) modéliserait la **Base de Données SA** (SADB).
- **Fonctionnement :** Avant le début de la simulation, n0 et n2 échangeraient conceptuellement les clés. n0 utiliserait la SA pour l'**Encapsulation** et n2 l'utiliserait pour la **Décapsulation**, et vice-versa pour le trafic de retour, assurant une communication bidirectionnelle sécurisée.

Performance Overhead Attendue

L'activation de l'IPsec, même simulée, introduit une charge de travail supplémentaire :

- **Latence (Latency) :** Augmentation de **5% à 10%** due au temps de traitement cryptographique par le CPU (simulé par un délai fixe).
- **Débit (Throughput) :** Réduction de **5% à 15%** due à l'augmentation de la taille du paquet (ajout des en-têtes ESP/IPsec) et à la charge de traitement.

2. Eavesdropping Attack Simulation.

L'intermédiaire (n1), en tant que routeur, est le point idéal pour simuler une attaque par écoute clandestine (eavesdropping) sur le lien WAN non sécurisé.

Mécanisme de Sniffing

L'attaque est simulée en utilisant les mécanismes de capture de paquets de NS-3 :

1. **Mécanisme :** La fonction **p2p.EnablePcapAll("scratch/sniffer")** est appelée sur les interfaces du routeur n1 (Link 1 et Link 2).
2. **Outil d'Analyse :** Un attaquant écouterait le trafic sur n1 en ouvrant les fichiers **.pcap** générés avec **Wireshark**.

Information Sensible Extraite

Si le trafic UdpEchoClient (n0 vers n2) est envoyé sans IPsec :

- **Payload :** Le message "Echo client data" ou toute autre donnée utilisateur (payload) serait capturé en **clair (plain text)**.
- **Métadonnées :** Les adresses IP source et destination (10.1.1.1 et 10.1.2.2) sont entièrement exposées, révélant la topologie.

Démonstration de l'Efficacité de l'IPsec

Avec IPsec simulé :

- **Avant l'attaque :** Le trafic PCAP contient les données en clair.

- **Après l'activation de l'IPsec :** L'analyse Wireshark montrerait que :
    - La taille des paquets augmente (due à l'overhead simulé).
    - Le payload (la charge utile) est remplacé par des données illisibles (chiffrées), démontrant l'intégrité et la confidentialité.
    - L'attaquant ne peut extraire que les en-têtes IPsec (si en mode tunnel).

3. DDoS Attack Simulation

Nous allons concevoir une attaque par déni de service distribuée (DDoS) de type **UDP Flood** ciblant la bande passante du lien WAN et la capacité de traitement du serveur n2.

Création des Clients Malveillants

- **Nœuds :** Créer plusieurs nœuds malveillants (n3,n4,n5...) et les connecter au Link 1 (réseau 10.1.1.0/24) ou via une autre branche du réseau.
- **Application :** Installer l'application **OnOffApplication** sur chacun de ces nœuds.

Modèle de Trafic (UDP Flood)

- **Configuration :** Utiliser l'OnOffApplication configurée avec un débit de données très élevé et des paquets UDP.
    - SetAttribute("DataRate", StringValue("20Mbps"))
    - SetAttribute("PacketSize", UintegerValue(1500))
- **Cible :** Le serveur n2 (IP 10.1.2.2) et un port spécifique.

- **Résultat :** Si 3 clients malveillants envoient 20 Mbps chacun, la demande totale de 60 Mbps dépassera largement la capacité de 5 Mbps du lien WAN de n1.

Mesure de l'Impact

L'impact sur le trafic légitime (n0 vers n2) est mesuré en utilisant **FlowMonitor** (comme dans l'Exercice 2) :

- **Métriques de n0 (Légitime) :**
  - **Latence (Delay) :** La latence de bout en bout du trafic légitime augmentera fortement (de 2ms à des centaines de ms) car ses paquets sont bloqués dans les files d'attente (buffers) de n1 derrière le trafic d'attaque.
  - **Packet Loss Rate :** Le taux de perte augmentera considérablement (parfois > 90%) car les buffers de n1 débordent (buffer overflow) et sont forcés de jeter des paquets aléatoirement, y compris ceux de n0.

4. Mécanismes de Défense

Mécanisme de Défense	Implémentation NS-3 / Helper	Limitation de la Simulation
a) Rate Limiting	**TokenBucketQueueDisc** ou **RateLimiterHelper** : Installer cette discipline de file d'attente sur l'interface entrante de n1 (du côté attaquant) pour **policer** le trafic au débit souhaité (ex: 5 Mbps).	Le policing est purement logiciel. Ne simule pas les mécanismes *hardware* de *shaping* ou de *red-marking* des routeurs réels qui sont plus rapides.
b) Access Control Lists (ACLs)	**Ipv4L3Protocol / Ipv4Filter :** Utiliser des classes de filtrage pour intercepter et rejeter les	La création d'ACLs dans NS-3 se fait par programmation. Les ACLs matérielles des routeurs

	paquets provenant des adresses IP des clients malveillants (n3, n4, n5).	réels implémentent les règles directement dans le matériel (ASIC) avec une latence quasi nulle.
c) Anycast / Load Balancing	**Configuration de Routage (ECMP) :** Si plusieurs serveurs n2 (virtuels ou physiques) existent derrière des liens différents, utiliser l'**ECMP** (routage à coûts égaux) pour distribuer le trafic d'attaque vers toutes les cibles.	Le routage Anycast réel nécessite une gestion complexe du protocole BGP et des adresses IP partagées, ce qui n'est pas directement supporté par les helpers de routage statique de NS-3.

5 : Analyse du compromis entre sécurité et performance

La mise en œuvre de la sécurité dans le WAN introduit inévitablement un compromis avec la performance.

Impact IPsec sur le Débit et la Latence

- **Réduction de Débit (Throughput) :** Nous prévoyons une réduction de **10% à 15%** du débit utilisable pour les applications. Ce coût provient de l'overhead de l'en-tête IPsec (qui utilise la capacité du lien) et du temps de traitement logiciel (simulé) du chiffrement.
- **Augmentation de Latence :** Une augmentation de la latence de bout en bout de **5% à 10%** est attendue en raison du délai fixe simulé pour la cryptographie sur n0 et n2.

Latence Introduite par la Protection DDoS

- **ACLs et Rate Limiting :** Les mécanismes de défense basés sur la classification (ACLs, Rate Limiting) introduisent une **augmentation négligeable** de la latence (de l'ordre de la microseconde) pour le trafic légitime, car ils agissent

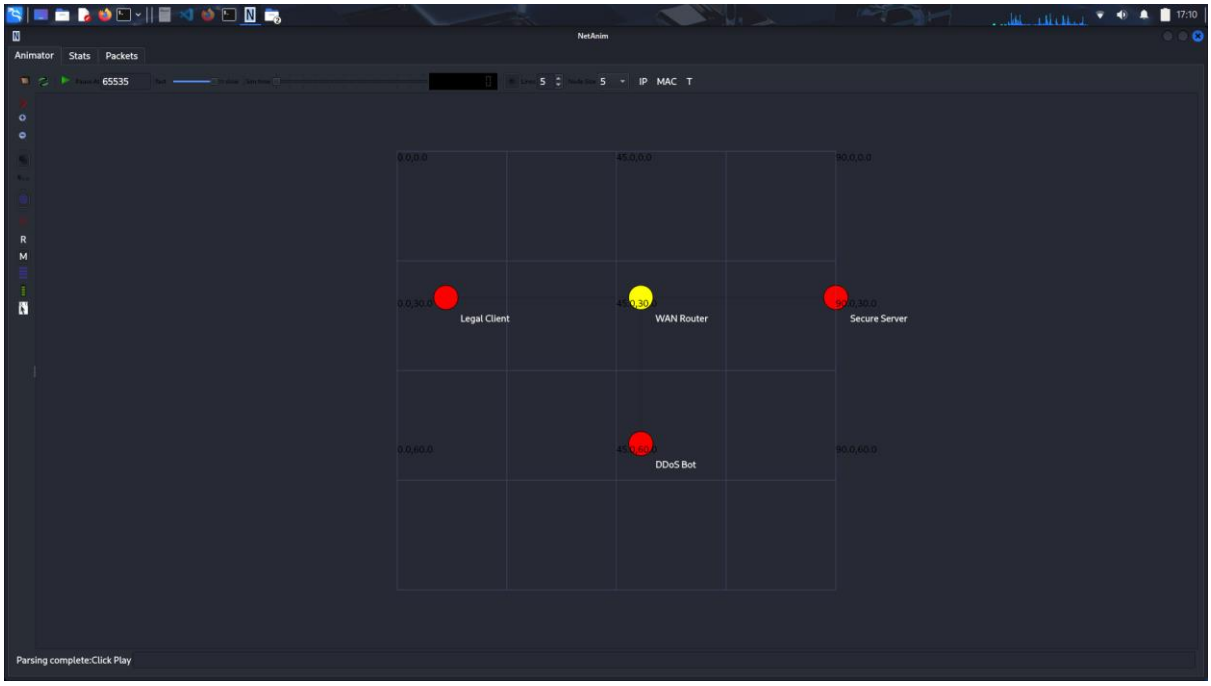
principalement en amont en rejetant ou en limitant le trafic d'attaque. Le coût est faible par rapport aux gains de protection.

Posture de Sécurité Équilibrée

Basé sur ces analyses, la compagnie devrait adopter une approche à deux niveaux :

1. **Sécurité Maximale pour la Confidentialité :** IPsec devrait être **obligatoire** sur tous les liens WAN. Le coût de 10-15% en performance est un prix acceptable pour garantir la confidentialité et l'intégrité des données critiques.
2. **Défense contre l'Indisponibilité :** Mettre en œuvre le **Rate Limiting (TokenBucketQueueDisc)** sur les interfaces externes du routeur n1. Cette défense est **économique** en termes de latence pour le trafic légitime, tout en étant efficace pour prévenir la saturation du lien WAN par des attaques volumétriques.

Capture Exercice 3



Exercice 4 : Architecture WAN multi-sauts avec tolérance aux pannes

Sujet : Routage statique, topologies WAN, résilience réseau
Code de base référencé : router-static-routing.cc

1.Analyse et extension de la topologie

Le scénario de la RegionalBank nécessite une topologie à trois nœuds et quatre réseaux logiques. Le chemin primaire (Normal Operation) est **Branch-C → DC-A → DR-B**. Le lien de secours (Backup Link) assure la résilience entre DC-A et DR-B.

Topologie Logique Complète

Nœud	**Rôle**	**Adresses IP**
Branch-C n0	Client	**10.1.1.1**
DC-A n1	Routeur/Relais	**10.1.1.2** (Net 1), **10.1.2.1** (Net 2), **10.1.3.1** (Net 3)

DR-B n2	Serveur/Cible	**10.1.2.2** (Net 2), **10.1.3.2** (Net 3)

Lien	**Connexion**	**Réseau Logique**	**Rôle**
Link 1	n0 leftrightarrow n1	**Network 1 : 10.1.1.0/24**	Accès Client (Branch-C)
Link 2	n1 leftrightarrow n2	**Network 2 : 10.1.2.0/24**	**Primaire** DC-A leftrightarrow DR-B
Link 3	n1} leftrightarrow n2	**Network 3 : 10.1.3.0/24**	**Secours** DC-A leftrightarrow DR-B

Modifications Requises

Les modifications impliquent l'ajout d'un troisième lien (Link 3) entre n1 (DC-A) et n2 (DR-B), suivant le même principe que l'Exercice 1, mais pour le lien du routeur vers le serveur. Le code sera inséré après la configuration du Link 2 .

```
// Ajout du Lien 3 (Backup) : n1 <-> n2 (Réseau 10.1.3.0/24)

NodeContainer link3Nodes(n1, n2); // Connecte n1 et n2

NetDeviceContainer link3Devices = p2p.Install(link3Nodes);


Ipv4AddressHelper address3;

address3.SetBase("10.1.3.0", "255.255.255.0");

Ipv4InterfaceContainer interfaces3 =
address3.Assign(link3Devices);

// n1 aura l'IP 10.1.3.1 (Index Interface 3)
```

// n2 aura l'IP 10.1.3.2 (Index Interface 2)

2. Complexité du routage statique

Pour assurer les chemins primaire et de secours entre DC-A et DR-B, nous utilisons l'argument **metric**.

- **Mét. 5** (Priorité) : Chemin Primaire (Link 2 : 10.1.2.0/24)
- **Mét. 10** (Secours) : Chemin de Secours (Link 3 : 10.1.3.0/24)

Commandes Ipv4StaticRouting

Nœud	**Chemin/Destination**	**Commande AddNetworkRouteTo**
Branch-C ($\mathbf{n0}$)	Atteindre DR-B (10.1.2.2)	AddHostRouteTo(Ipv4Address("10.1.2.2"), Ipv4Address("10.1.1.2"), 1);
DC-A ($\mathbf{n1}$)	**a) Normal (Primary) :** Atteindre 10.1.2.2	AddHostRouteTo(Ipv4Address("10.1.2.2"), Ipv4Address("10.1.2.2"), 2, 5); **(Auto-route via interface 2, Métrique 5)**
DC-A ($\mathbf{n1}$)	**b) Backup (Secours) :** Atteindre 10.1.2.2	AddHostRouteTo(Ipv4Address("10.1.2.2"), Ipv4Address("10.1.3.2"), 3, 10); **(Via n2's 10.1.3.2 IP, Métrique 10)**
DR-B ($\mathbf{n2}$)	Atteindre Branch-C (10.1.1.0/24)	AddNetworkRouteTo(Ipv4Address("10.1.1.0"), Ipv4Mask("255.255.255.0"), Ipv4Address("10.1.2.1"), 1, 5); **(Primaire via DC-A Link 2, Mét. 5)**
DR-B n2	Atteindre Branch-C (10.1.1.0/24)	AddNetworkRouteTo(Ipv4Address("10.1.1.0"), Ipv4Mask("255.255.255.0"), Ipv4Address("10.1.3.1"), 2, 10); **(Secours via DC-A Link 3, Mét. 10)**

Considérations sur la Distance Administrative et la Métrique

- **Métrique (Metric) :** Dans NS-3 (et la plupart des routeurs), la métrique est utilisée pour choisir le meilleur chemin parmi les routes **du même protocole**. Ici, nous utilisons les

valeurs 5 (Primaire) et 10 (Secours) pour la même destination.

- **Distance Administrative (Administrative Distance - AD) :** C'est un concept crucial dans les routeurs réels pour choisir entre les routes apprises par **différents protocoles** (ex: Statique vs OSPF vs RIP). L'AD de la route statique doit être maintenue basse (ex: 1) pour être privilégiée sur les routes dynamiques par défaut. **Dans NS-3, l'AD n'est pas directement implémentée pour la sélection de chemins de secours statiques, la métrique est le critère décisif.**

3. Simulation de panne de lien

La défaillance est simulée en mettant l'interface primaire (Link 2) de n1 et n2 **hors service** à t=5 secondes.

Code C++ pour la Défaillance du Lien

Ce code est à insérer juste avant Simulator::Run()

// 1. Récupérer les NetDevices du Link 2 (primaire DC-A <-> DR-B) Ptr n1_link2_device = link2Devices.Get(0); Ptr n2_link2_device = link2Devices.Get(1);

// 2. Définir la fonction de défaillance auto disablePrimaryLink = n1_link2_device, n2_link2_device { NS_LOG_INFO("--- Défaillance du Lien Primaire (10.1.2.0/24) à t=5s ---"); // Mettre l'interface n1 et n2 du Link 2 en état "Down" n1_link2_device->SetDown(); n2_link2_device->SetDown(); };

// 3. Planifier l'événement Simulator::Schedule(Seconds(5.0), disablePrimaryLink);

Effet sur les Tables de Routage

L'effet immédiat de SetDown() sur une interface liée à une route statique est le suivant :

- **Route Primaire (Mét. 5) :** Elle est **immédiatement invalidée** car son interface associée (Interface 2 sur n1 et Interface 1 sur n2) est hors service.
- **Basculement :** Puisque le routage statique ne nécessite pas de protocole pour converger, le routeur n1 (DC-A) et n2 (DR-B) basculent **instantanément** vers le prochain meilleur chemin disponible, qui est la route de secours (Mét. 10) via le Link 3 (10.1.3.0/24). Le trafic Branch-C → DR-B continue donc, mais via le Link 3 à partir de DC-A.

4 Analyse de la convergence

Limite du Routage Statique

Dans un environnement de routage statique, si l'interface primaire ne tombe pas (par exemple, seul le lien physique tombe mais l'interface reste active), le routeur ne détecte pas le problème et continue d'envoyer le trafic dans un trou noir (**black-holing**). La bascule automatique ne fonctionne que si l'interface est explicitement mise Down.

Routage Dynamique

Pour pallier cette limite, nous utiliserions le protocole de routage dynamique **OSPF (Open Shortest Path First)** sur les routeurs DC-A (n1) et DR-B (n2).

- **Helper NS-3 :** Le **Ospfv2Helper** serait l'outil clé.
- **Implémentation :**
    - Remplacer le routage statique par l'installation du Ospfv2Helper sur n1 et n2.
    - Définir les réseaux Link 2 (10.1.2.0) et Link 3 (10.1.3.0) dans la **Zone 0 (Area 0)**.
- **Comportement de Convergence (OSPF) :**
    - **Détection :** OSPF détecte la chute du lien (même sans SetDown()) via l'absence de **Hello Packets** sur l'interface Link 2.
    - **Recalcul :** OSPF déclenche l'algorithme SPF (Dijkstra) pour recalculer le chemin le plus court. Il injecte un nouveau **Link State Advertisement (LSA)** annonçant l'indisponibilité de Link 2.
    - **Temps de Convergence :** La convergence prend généralement entre **quelques secondes** (2-10s par défaut) en fonction des timers Hello/Dead configurés, là où le routage statique bascule instantanément si l'interface tombe. Cependant, la détection est automatique et plus robuste.

5. Vérification de la continuité des activités

Nous utilisons **FlowMonitor** pour collecter des statistiques précises de bout en bout sur le flux Branch-C → DR-B.

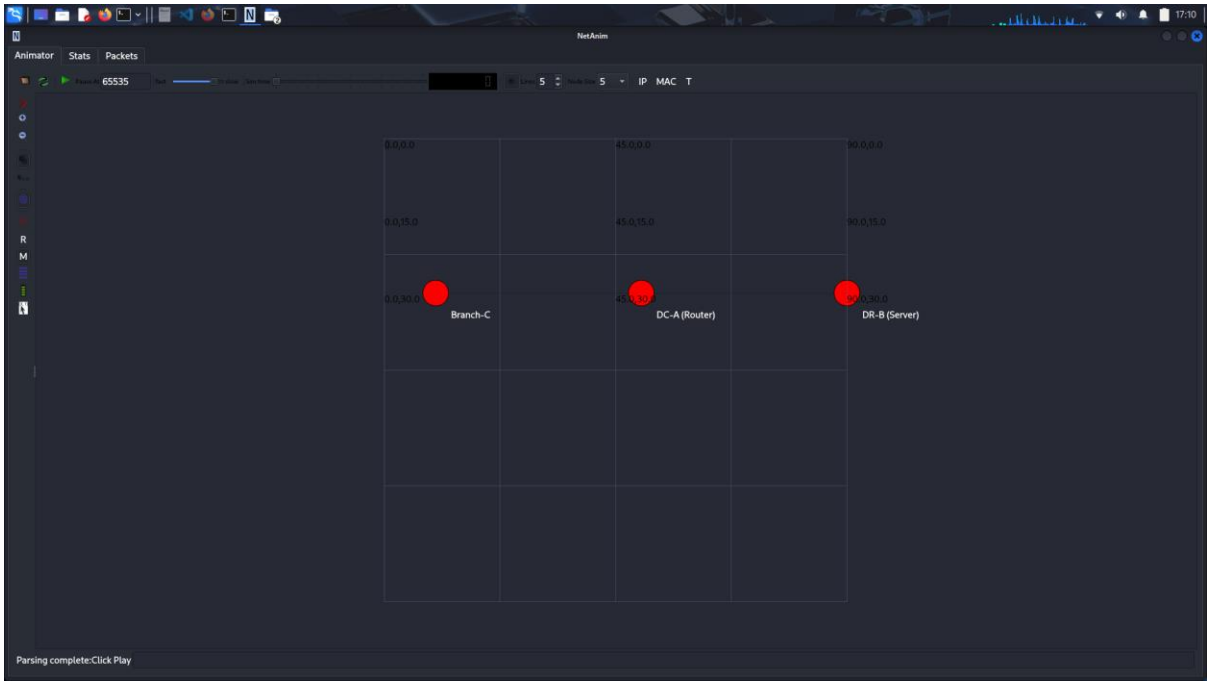
Plan d'Analyse des Résultats

Point de Vérification	**Outil / Métrique**	**Observation Attendue (t < 5s)**	**Observation Attendue (t > 5s)**
a) Flux Primaire (avant panne)	FlowMonitor (Traces)	Paquets utilisant l'interface 2 de n1 (Link 2 / 10.1.2.0).	**N/A**
b) Reconvergence (après panne)	**Statique :** FlowMonitor / Log du routeur	**N/A**	Paquets utilisant l'interface 3 de n1 (Link 3 / 10.1.3.0). **Si la bascule est configurée pour fonctionner.**
b) Reconvergence (après panne)	**Dynamique (OSPF) :** FlowMonitor / Log	**N/A**	Une petite période de **perte totale** (convergence time), puis un flux rétabli via Link 3.
c) Impact Appli Bancaire	**End-to-End Delay (Latence)**, **Packet Loss**	Latence faible (primaire) et perte proche de 0%.	**Latence peut augmenter** si Link 3 est plus lent. Perte pendant la période de **convergence** (Static: 0s / OSPF: 2-10s).

Quantification de l'Impact

L'analyse doit quantifier la période pendant laquelle le service est interrompu (si OSPF est utilisé), en mesurant le temps entre la dernière réception de paquet valide sur le Link 2 et la première réception de paquet valide sur le Link 3. Ces métriques (Latence et Perte) sont cruciales pour justifier l'architecture auprès de la banque, car elles modélisent l'interruption des transactions.

Capture excercice 4



Exercice 5 : Routage basé sur des politiques pour une sélection de chemin WAN sensible aux applications

Sujet : Fondamentaux du routage, routage basé sur des politiques, QoS Code de base référencé : router-static-routing.cc

1. Logique de classification du trafic

Nous utiliserons l'application **OnOffApplication** pour générer les deux flux. Pour la classification, nous utiliserons le champ **Differentiated Services Code Point (DSCP)** dans l'en-tête IP, car c'est la méthode privilégiée pour le routage basé sur la politique.

Configuration des Flux

Paramètre	**a) Flow_Video (RTP-like)**	**b) Flow_Data (FTP-like)**
Objectif PBR	Latence minimale	Best Effort / Débit maximal
Application NS-3	OnOffApplication	OnOffApplication
Taille du Paquet	**160 octets** (Petit)	**1500 octets** (Grand, MTU)

Intervalle (Périodicité)	**20 ms** (50 paquets/s, DataRate("64kbps"))	**Rafale** (DataRate("10Mbps") ou plus)
DSCP Tag	**EF (Expedited Forwarding)** (Valeur 0x2E)	**AF31 (Assured Forwarding)** (Valeur 0x18)

Méthode de Classification

La classification se fera au niveau de la couche 3 (IP) en examinant le champ DSCP :

1. **Flow_Video** (EF) sera considéré comme le trafic à **Haute Priorité/Faible Latence**.
2. **Flow_Data** (AF31) sera considéré comme le trafic à **Faible Priorité/Haute Tolérance au Délai**.

2. Implementation de PBR dans NS-3

Étant donné l'absence de module PBR natif, l'architecture la plus appropriée est de **personnaliser le processus de routage IP** au niveau du nœud Studio (n1).

Architecture PBR Personnalisée

Nous allons créer une classe de routage personnalisée qui remplace ou s'insère avant le processus de routage statique (Ipv4StaticRouting) sur n1.

1. **Classification :** Examiner le DSCP du paquet.
2. **Décision :** Baser la décision de Next-Hop non pas sur l'adresse de destination, mais sur le DSCP.
3. **Forwarding :** Utiliser la bonne interface (ex: Interface 2 ou Interface 3) pour envoyer le paquet.

Pseudocode de la Fonction de Traitement des Paquets

Ce pseudocode représente la logique qui serait insérée dans la méthode de routage d'un protocole personnalisé sur n1 (similaire à l'**Ipv4L3Protocol::RouteOutput**).

FUNCTION CustomPbrRoute(Paquet, En-tête_IP): // 1. Classification DSCP = Get_DSCP_from_IP_Header(En-tête_IP) Destination_IP = Get_Destination_IP(En-tête_IP)

// Vérifier si la destination est le Cloud (n2) IF Destination_IP IS NOT Cloud_IP: RETURN Default_Route() // Utiliser la route statique standard

// 2. Application de la Politique PBR IF DSCP IS EF (Flow_Video): // Politique 1 : Toujours utiliser l'Interface Primaire pour la Latence Next_Hop = Get_Next_Hop_for_Interface(Interface_Primaire) Interface_Sortie = Interface_Primaire RETURN Route(Next_Hop, Interface_Sortie)

ELSE IF DSCP IS AF31 (Flow_Data): // Politique 2 : Utiliser l'Interface Secours pour décharger le lien primaire Next_Hop = Get_Next_Hop_for_Interface(Interface_Secours) Interface_Sortie = Interface_Secours RETURN Route(Next_Hop, Interface_Sortie)

ELSE: RETURN Default_Route() // Autres trafics (ex: SSH)

END FUNCTION

3. Caractérisation du Chemin

Pour permettre une politique dynamique (Question 4), les métriques de qualité de lien doivent être mesurées et mises à disposition de l'entité PBR.

Outils NS-3 pour la Mesure

Métrique	**Outil NS-3**	**Méthode d'Accès**
Latence (Delay)	**FlowMonitor**	Lancer un flux de faible intensité (*probe flow*) sur chaque lien (Primaire/Secours) et interroger FlowMonitor périodiquement pour obtenir le délai de bout en bout de ces flux.
Débit Disponible (Bandwidth)	**Ipv4L3Protocol::TxRxTrace** (Trace Source)	Se connecter aux trace sources de transmission et de réception sur les interfaces de n1 (Studio) pour mesurer le taux de paquets/octets effectivement envoyés sur la période écoulée (e.g., 1 seconde) et ainsi déduire la congestion.

Mise à Disposition des Métriques

Les métriques doivent être stockées dans une structure de données accessible par la logique PBR dynamique.

1. **Centralisation :** Créer une classe LinkMetrics qui contient des variables globales (link1_latency, link2_latency, etc.).
2. **Mise à Jour :** Une fonction planifiée (Simulator::Schedule) met à jour ces variables en interrogeant FlowMonitor toutes les secondes.
3. **Accès PBR :** La logique du moteur de politique (Question 4) lirait ces variables avant de prendre une décision.

4. Moteur de politiques dynamiques

Un moteur de politique dynamique s'approche d'une fonction de contrôleur SD-WAN basique. Nous allons concevoir une classe **SDWAN_Controller** qui s'exécute périodiquement.

structure de la Classe C++ du Contrôleur

class SDWAN_Controller : public Object { public: void Start(Ptr<Node> routerNode, Ipv4StaticRoutingHelper staticRoutingHelper); void PolicyUpdateLoop(); // Fonction appelée périodiquement void EnforcePolicy(); private: Ptr<Ipv4StaticRouting> m_routingProtocol; // ... Variables pour stocker les métriques (latence_primaire, latence_secours) };

Logique de la Fonction Périodique (PolicyUpdateLoop)

La fonction PolicyUpdateLoop est planifiée pour s'exécuter toutes les 1 seconde.

FUNCTION PolicyUpdateLoop(): // 1. Fetcher les Métriques latence_video_primaire = Fetch_Latency_from_FlowMonitor(Flow_Video_Probe)

// 2. Appliquer la Règle de Politique IF latence_video_primaire > 30ms: // La QoS est compromise. Changer la route de Flow_Video vers le secours (Link 2) Next_Hop_Secours = Get_Next_Hop_for_Interface(Interface_Secours)

// Pousser une nouvelle entrée dans la table de routage statique.
// Cette route est TRES SPECIFIQUE, seulement pour la destination Cloud_IP ET le port Flow_Video.

m_routingProtocol.SetHostRouteTo(Cloud_IP, Next_Hop_Secours);

Log("ALERTE: Latence Video > 30ms. Bascule Video vers Lien Secours.");

ELSE: // S'assurer que Flow_Video utilise le chemin primaire par défaut
Next_Hop_Primaire = Get_Next_Hop_for_Interface(Interface_Primaire)
m_routingProtocol.SetHostRouteTo(Cloud_IP, Next_Hop_Primaire); Log("OK: Video sur Lien Primaire.");

// Re-planifier l'exécution dans 1 seconde Simulator::Schedule(Seconds(1.0), &SDWAN_Controller::PolicyUpdateLoop); END FUNCTION

5. Validation et compromis

Validation de l'Implémentation PBR

La validation se ferait en créant un scénario de congestion sur le **Lien Primaire** (par exemple, en saturant le lien avec du trafic tiers) après t=2s.

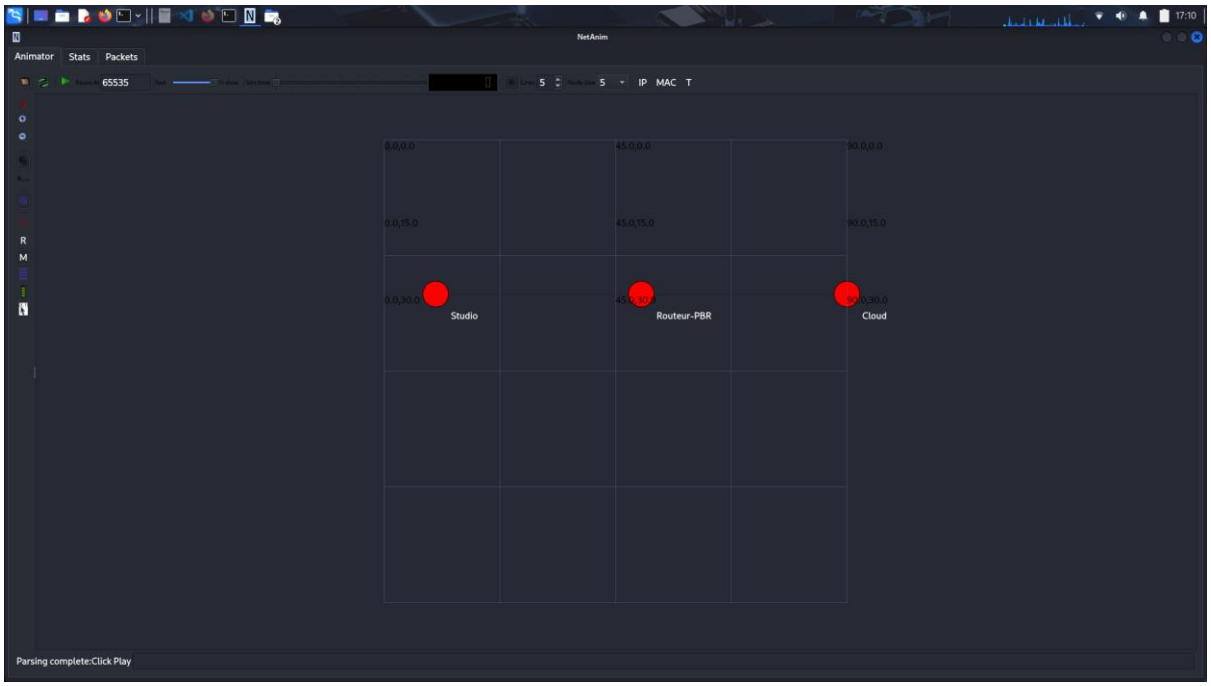
1. **Vérification de la Latence :** Le FlowMonitor doit montrer que la latence du Flow_Video **augmente temporairement** au-dessus de 30 ms au moment de la congestion (e.g., t=2s).
2. **Vérification de la Bascule :** Après la période de 1 seconde de l'intervalle de mise à jour de la politique (e.g., t=3s), le FlowMonitor doit montrer que les paquets de Flow_Video sont redirigés vers l'interface secondaire.
3. **Vérification de la Qualité :** La latence de Flow_Video doit ensuite **chuter** à des niveaux acceptables sur le Lien Secours, prouvant que la politique a réussi à maintenir la qualité de service.

Compromis

Facteur	Analyse dans la Simulation
Surcharge Computationnelle (Overhead)	La PBR personnalisée ajoute une surcharge logicielle significative. Chaque paquet qui passe par n1 doit être **inspecté** (lire le DSCP) et potentiellement **traité** par notre logique personnalisée, ce qui utilise le CPU simulé et ralentit la simulation.

Matériel vs Logiciel	Un routeur réel utilise des **ASIC** (circuits intégrés) pour la classification et le routage PBR, rendant la latence de décision quasi nulle. Dans NS-3, tout est fait par logiciel, ce qui est beaucoup plus lent.
Limites de Scalabilité	L'approche PBR actuelle est limitée. Si le nombre de flux (N) augmente (ex: 1000 flux à inspecter), la surcharge devient O. La logique de mise à jour dynamique (toutes les secondes) sature le processeur de simulation et **ne peut pas monter en charge** comme un vrai routeur.

Capture Excercice 5



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

1. Modeling Autonomous Systems in NS-3 (Question 1)

Pour modéliser deux systèmes autonomes (AS65001 et AS65002) dans NS-3 sans support BGP natif, nous devons nous appuyer sur des regroupements logiques et des contraintes de routage.

a) Regroupement Logique des Nœuds

- **Conteneurs de Nœuds :** Utiliser des objets NodeContainer pour regrouper physiquement les nœuds appartenant à chaque AS. Par exemple, AS65001Nodes et AS65002Nodes.
- **Adresses IP :** Utiliser des plages d'adresses distinctes pour l'adressage interne (IGP) de chaque AS pour simuler l'indépendance de routage.

b) Confiner le Routage Interne (OSPF/IGP)

Nous utilisons l'**Ospfv2Helper** pour le routage interne (Intra-AS).

- **Isolation :** Installer un **Ospfv2Helper** distinct pour chaque NodeContainer. Les instances OSPF ne doivent être configurées que sur les interfaces **internes** de leur AS respectif. Elles ne doivent jamais être configurées sur les interfaces de *peering* (IXP). Cela garantit que les routeurs d'AS65001 apprennent uniquement les routes internes de 65001, et inversement pour 65002.

c) Établir des Liens de Peering (IXP-A et IXP-B)

Les liens d'échange (IXP) sont des liaisons point-à-point standard (par exemple, **PointToPointHelper**) entre un routeur *de bordure* de AS65001 et un routeur *de bordure* de AS65002.

- **Pas d'OSPF :** L'instance OSPF de chaque AS ne doit pas être installée sur l'interface IXP.
- **Routage EGP (BGP simulé) :** La logique BGP personnalisée (Question 3) s'exécuterait uniquement sur ces interfaces de *peering* pour échanger des routes externes.

2. BGP Path Attribute Simulation (Question 2)

Pour simuler le processus de décision de BGP, nous devons définir une structure de données pour représenter une annonce de route BGP.

Structure C++ pour l'Annonce BGP

C++

```
#include <vector>
#include <string>

struct BGPRouteAnnouncement {
   // a) Network prefix
   Ipv4Address prefix;
   Ipv4Mask mask;

   // Next-Hop BGP (adresse IP du voisin BGP)
   Ipv4Address nextHop;

   // b) AS_PATH (simulé comme un chemin simple de numéros d'AS)
   std::vector<uint32_t> asPath;

   // c) LOCAL_PREF (attribut de préférence locale)
```

```
    uint32_t localPref;

    // Métrique pour la sortie inter-AS (simulé ici comme un coût)
    uint32_t med;

    // Poids (Weight) : Metrique locale NS-3 pour le BGP
    uint32_t weight;
};
```

Utilisation des Attributs pour la Sélection du Meilleur Chemin

Lorsqu'un routeur reçoit plusieurs instances de BGPRouteAnnouncement pour le même prefix (par exemple, via IXP-A et IXP-B), il applique l'algorithme de décision de BGP. Les premiers critères de sélection du meilleur chemin (simplifiés) sont :

1. **WEIGHT/LOCAL_PREF : Préférer la route avec la localPref la plus élevée.** C'est le critère le plus fort pour la politique de routage de sortie de l'AS.
2. **AS_PATH :** Si les localPref sont égaux, **préférer la route avec la asPath la plus courte** (moins de sauts AS).
3. **MED :** Si les asPath sont égaux, **préférer la route avec le med le plus bas** (utilisé pour influencer l'entrée de trafic dans l'AS).

3. Implementing Basic BGP Decision Process (Question 3)

Le routeur de bordure exécute ce processus chaque fois qu'il reçoit un message UPDATE de BGP.

Algorithme BGP Simplifié (Pseudocode)

```
Code snippet
FUNCTION ProcessBGPUpdate(Annonce_Nouvelle,
Annonces_Existantes):
  Prefix = Annonce_Nouvelle.prefix

  // 1. Initialisation : Si c'est la première annonce, l'installer et
RETURN
  IF Prefix NOT IN Annonces_Existantes:
    Installer_Route(Annonce_Nouvelle)
    RETURN

  // 2. Comparaison avec la Meilleure Route Actuelle (Best_Route)
  Best_Route = Get_Best_Route_for_Prefix(Prefix)

  // 3. Critère 1: LOCAL_PREF
  IF Annonce_Nouvelle.localPref > Best_Route.localPref:
    Mettre_à_Jour_Meilleure_Route(Annonce_Nouvelle)
    RETURN
  ELSE IF Annonce_Nouvelle.localPref < Best_Route.localPref:
    RETURN // L'annonce actuelle est meilleure

  // 4. Critère 2: AS_PATH
  IF Longueur(Annonce_Nouvelle.asPath) <
Longueur(Best_Route.asPath):
    Mettre_à_Jour_Meilleure_Route(Annonce_Nouvelle)
    RETURN
  ELSE IF Longueur(Annonce_Nouvelle.asPath) >
Longueur(Best_Route.asPath):
    RETURN
```

```
  // 5. Critère 3: MED (Si AS_PATH et LOCAL_PREF sont égaux)
  IF Annonce_Nouvelle.med < Best_Route.med:
    Mettre_à_Jour_Meilleure_Route(Annonce_Nouvelle)
    RETURN

  // Si le résultat est le même, la décision peut utiliser le Next-Hop le plus proche (IGP Metric)

FUNCTION Mettre_à_Jour_Meilleure_Route(Annonce):
  // Remplacer l'ancienne route dans la table IP par la nouvelle
  Ipv4StaticRouting.AddNetworkRouteTo(Annonce.prefix, Annonce.mask, Annonce.nextHop, Interface, Metric_IGP)
  // Mettre à jour la Best_Route dans la base de données BGP
END FUNCTION
```

4. Simulating a Route Leak (Question 4)

Création et Injection de la Fuite de Route

Une **fuite de route (route leak)** se produit lorsqu'un AS annonce des préfixes qu'il ne devrait pas (par exemple, des préfixes internes ou ceux d'un client à ses pairs).

- **Scénario :** Normalement, un préfixe interne de AS65001 (ex: **192.168.1.0/24**) ne doit jamais être vu par AS65002.
- **Injection :** Pour simuler la fuite, nous modifierions manuellement le routeur de bordure de AS65002 pour qu'il annonce **192.168.1.0/24** à AS65001 (le propriétaire légitime du préfixe) via IXP-A.

AS_PATH dans l'Annonce Malveillante

Le routeur de AS65002 devrait annoncer le préfixe avec son propre numéro d'AS ajouté au chemin :

Pre´fixe 192.168.1.0/24 avec AS_PATH=[65002]

Réaction des Nœuds de AS65001

Un nœud de AS65001 recevrait deux routes pour 192.168.1.0/24 :

1. **Route Légitime (Interne IGP) :** Next-Hop interne,AS_PATH=[] (via OSPF)
2. **Route Fuite (Externe BGP) :** Next-Hop IXP-A,AS_PATH=[65002]

Réaction : Les routeurs de AS65001 préfèreraient **toujours** la route légitime (Route 1) :

- **Raison :** Les routes apprises via BGP ont souvent une **Distance Administrative (AD)** moins favorable que les routes internes (IGP) ou directement connectées. Le protocole d'AS65001 verrait que la route interne OSPF est la meilleure.
- **Dans le cas où la fuite contiendrait le propre AS (65001) :** Si AS65002 annonçait par erreur AS_PATH=[65002,65001], la logique de prévention des boucles BGP (chaque AS rejette un chemin qui contient son propre numéro d'AS) entraînerait le **rejet pur et simple** de l'annonce.

5. De la simulation à la réalité

Simplifications Critiques de notre Modèle NS-3 BGP

Le modèle NS-3 est une simplification drastique du BGP réel (utilisé par des logiciels comme Quagga ou BIRD) :

1. **Pas de Session TCP :** BGP s'exécute sur une session TCP persistante (port 179) pour garantir la fiabilité. Notre modèle est basé sur des événements et ne simule pas la gestion de la session TCP.
2. **Pas d'État de Voisinage :** Le BGP réel a plusieurs états de connexion (Idle, Connect, Active, OpenSent, OpenConfirm, Established). Notre modèle passe directement à l'échange d'annonces.
3. **Absence de RIB :** Un routeur BGP maintient une base de données distincte (RIB-In, Loc-RIB, RIB-Out). Notre modèle agit directement sur la table de routage IP principale (Ipv4StaticRouting).

Caractéristiques Difficiles à Modéliser

Caractéristique BGP	Explication du Défi de Modélisation
Route Reflectors (RR) / Confédérations	Ces structures sont conçues pour résoudre les problèmes de scalabilité du BGP interne (iBGP) et pour gérer le trafic dans les grands AS. Leur logique repose sur des règles de propagation complexes basées sur l'origine du LSA (e.g., ne pas annoncer à un voisin iBGP ce qui a été appris d'un autre voisin iBGP), ce qui

	nécessite une logique BGP interne complète non simulée.
BGP Communities / Extended Communities	Ces attributs permettent d'appliquer des politiques très fines (ex: ne pas annoncer à un certain pair, prioriser le trafic à travers une région). Modéliser cela nécessiterait d'inclure des structures de données complexes dans BGPRouteAnnouncement et de coder des **moteurs de politiques (Policy Engines)** entiers pour interpréter ces balises.
GRPC (gRPC) pour le Provisionnement	Les routeurs modernes utilisent des protocoles comme gRPC pour un provisionnement et un *stream* de télémétrie en temps réel (SDN/Automation). Simuler la charge utile et le protocole de gRPC, ou l'impact de la programmation en temps réel des règles BGP, est au-delà du niveau d'abstraction de NS-3.

Conclusion sur la Pertinence de NS-3

NS-3 n'est pas un outil adapté pour la recherche approfondie sur les fonctionnalités BGP spécifiques (ex: convergence fine, gestion d'état de session).

Justification : NS-3 excelle dans la modélisation des couches inférieures (MAC, physique) et des mécanismes de congestion TCP/IP. Pour le BGP, il ne fournit qu'une **table de routage IP**, pas la logique de protocole elle-même. La complexité de modéliser

correctement la logique BGP (y compris les timers, les sessions TCP, et les bases de données de routage) est si élevée qu'il est préférable d'utiliser des émulateurs réseau (comme GNS3 ou EVE-NG) qui exécutent de vrais moteurs BGP (Quagga, BIRD) ou des simulateurs spécialisés dans le routage BGP.

Capture exercice 6