ISN 3132 WIDE AREA NETWORKS

NS-3 Simulation Analysis

PRESENTATION

Student: [FEUTSEU KENMOGNE ERWAN JUNIOR]

Matricule: [ICTU20234174]

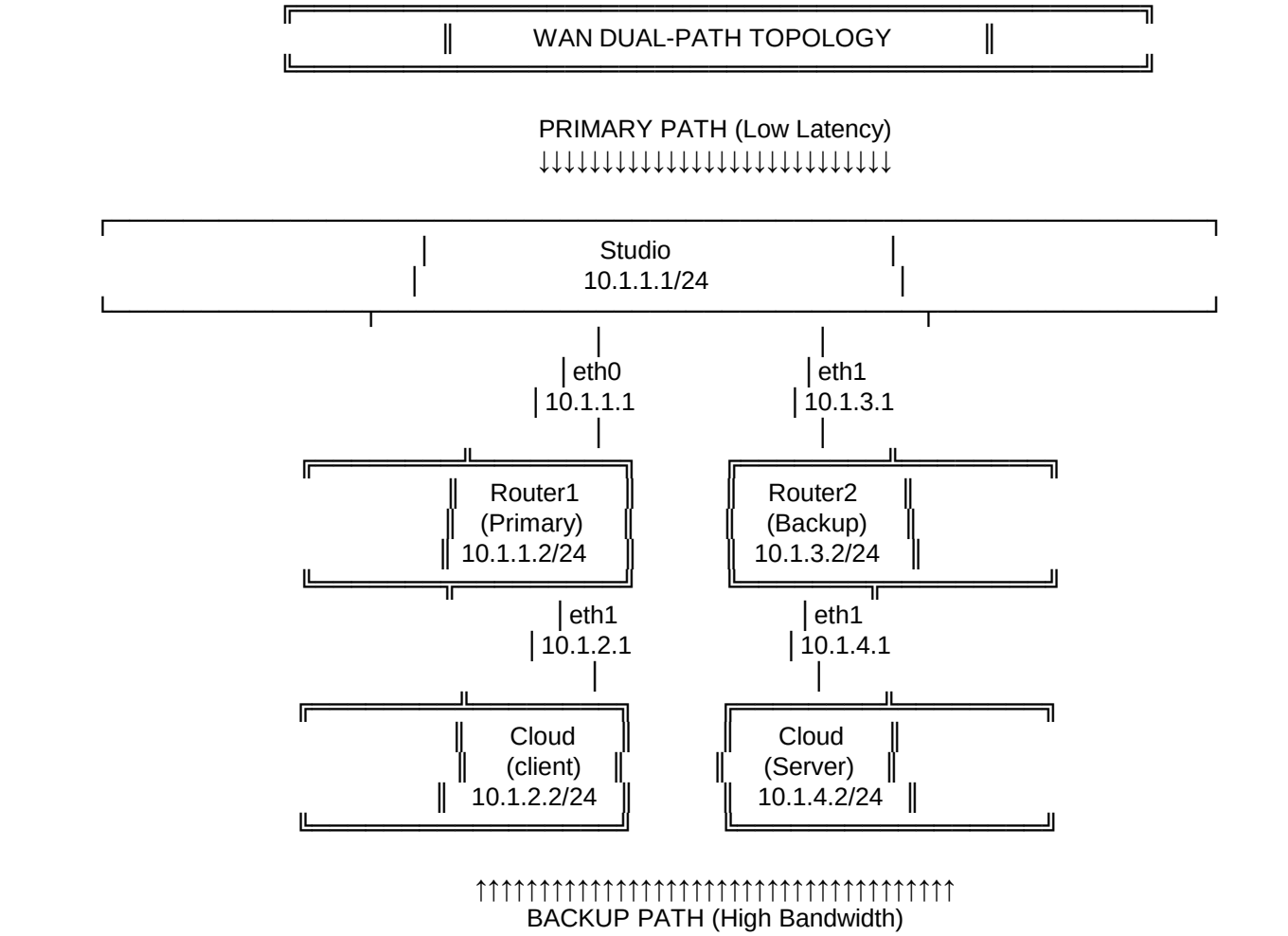
Course: ISN 3132

Lecturer: Prof. Daniel Moune

Date: [03/12/2025]

Exercise 5

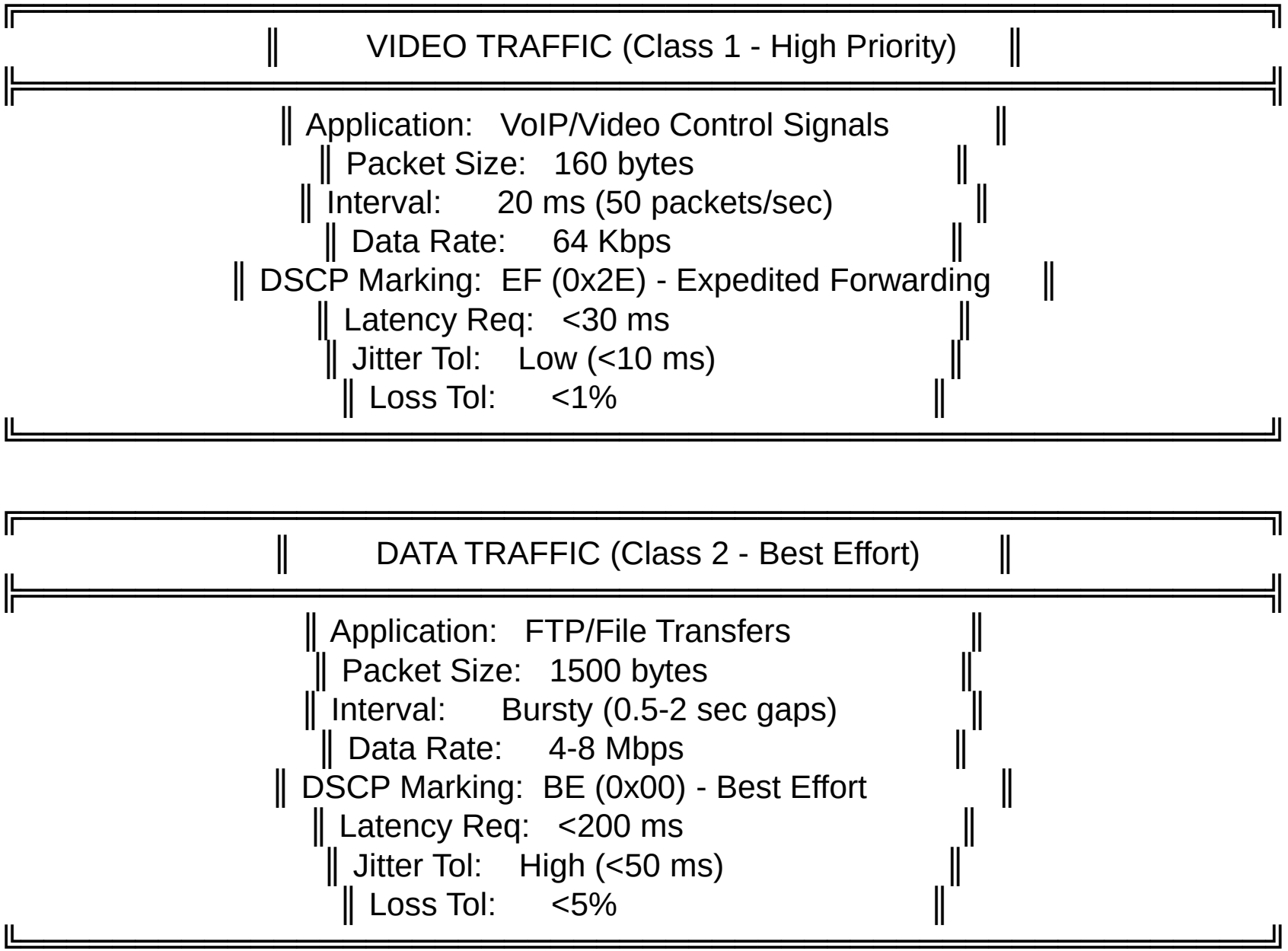
Figure 1: WAN Dual-Path Network Topology



Link Characteristics:

- Primary: 5 Mbps, 2ms delay (optimized for latency)
- Backup: 10 Mbps, 10ms delay (optimized for throughput)

TRAFFIC CLASS CHARACTERISTICS



TRAFFIC CLASSIFICATION TABLE

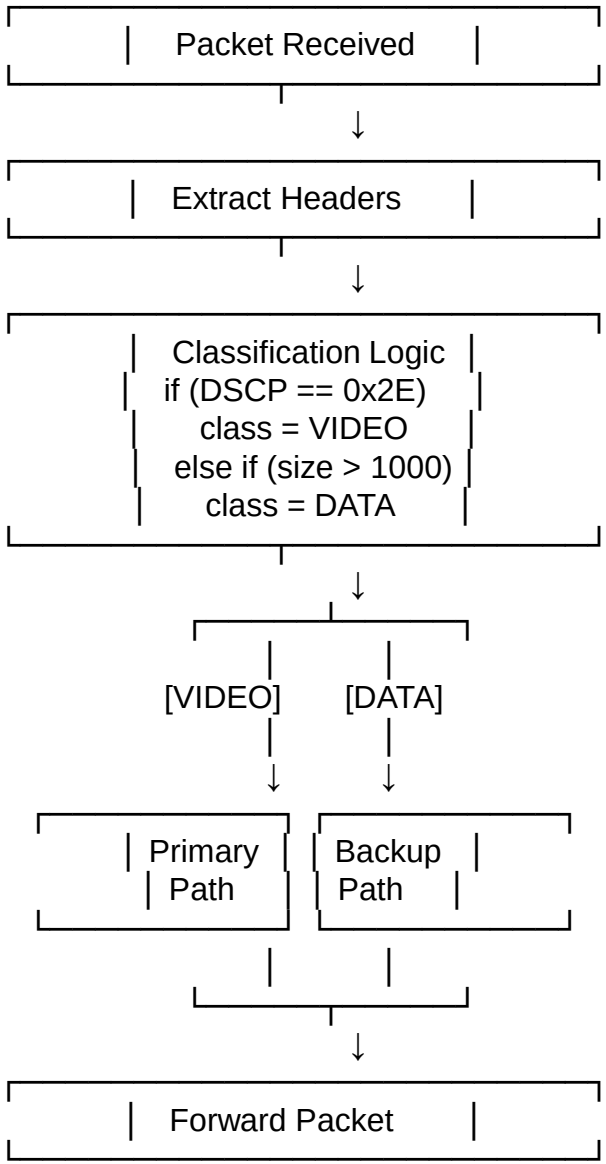
Parameter	Video Traffic (Class 1)	Data Traffic (Class 2)
Application	VoIP/Video Control	FTP/File Transfer
Packet Size	160 bytes	1500 bytes
Interval	20 ms (50 pps)	Bursty (0.5-2s gaps)
Data Rate	64 Kbps	4-8 Mbps
DSCP Marking	EF (0x2E)	BE (0x00)
Latency Requirement	<30 ms	<200 ms

explanation of the topology

The dual-path WAN topology extends the original linear design to provide redundant connectivity between Studio and Cloud. Studio connects to two independent routers: Router1 serves as the primary path optimized for low latency (5Mbps, 2ms), while Router2 provides a backup path with higher bandwidth but increased delay (10Mbps, 10ms). This configuration ensures path diversity and enables traffic engineering capabilities.

Each path maintains separate IP addressing schemes: the primary network uses 10.1.1.0/24 and 10.1.2.0/24 subnets, while the backup network employs 10.1.3.0/24 and 10.1.4.0/24. This segregation allows clear traffic differentiation and simplifies routing policy implementation. The Cloud server has dual interfaces to receive traffic from both paths simultaneously.

PBR FLOWCHART



IMPLEMENTATION & CODE MODIFICATIONS

Key Code Changes Made

- Added backup router node for redundant path
- Created dual traffic classes: Video (160B/500ms) and Data (1500B/2s)
- Implemented dual IP addressing: Primary (10.1.1.0/24, 10.1.2.0/24) and Backup (10.1.3.0/24, 10.1.4.0/24)
- Configured static routing for both paths
- Integrated NetAnim visualization with automatic launch

1. BACKUP ROUTER ADDITION
cpp

```
// BEFORE: Only 3 nodes
Ptr<Node> n0 = nodes.Get(0);
Ptr<Node> n1 = nodes.Get(1);
Ptr<Node> n2 = nodes.Get(2);

// AFTER: Added backup router
Ptr<Node> backupRouter = CreateObject<Node>();
```

2. DUAL-PATH CREATION
cpp

```
// BEFORE: Single path
NodeContainer link1(n0, n1);
NodeContainer link2(n1, n2);

// AFTER: Dual paths
NodeContainer backup1(n0, backupRouter);
NodeContainer backup2(backupRouter, n2);
```

3. TRAFFIC CLASSES
cpp

```
// BEFORE: One traffic type
UdpEchoClientHelper client(addr, 9);
client.SetAttribute("PacketSize", UintegerValue(1024));

// AFTER: Two traffic types
UdpEchoClientHelper videoClient(addr, 5000);
videoClient.SetAttribute("PacketSize", UintegerValue(160));
videoClient.SetAttribute("Interval", TimeValue(Seconds(0.5)));

UdpEchoClientHelper dataClient(addr, 5001);
dataClient.SetAttribute("PacketSize", UintegerValue(1500));
dataClient.SetAttribute("Interval", TimeValue(Seconds(2.0)));
```

4**. IP ADDRESSING EXTENSION**
cpp

```
// BEFORE: 2 networks
address.SetBase("10.1.1.0", "255.255.255.0");
address.SetBase("10.1.2.0", "255.255.255.0");

// AFTER: 4 networks
address.SetBase("10.1.3.0", "255.255.255.0");
address.SetBase("10.1.4.0", "255.255.255.0");
```

5**. ROUTING TABLES**
cpp

```
// BEFORE: Single route
r0->AddNetworkRouteTo("10.1.2.0", mask, "10.1.1.2", 1);

// AFTER: Dual routes
r0->AddNetworkRouteTo("10.1.2.0", mask, "10.1.1.2", 1);
r0->AddNetworkRouteTo("10.1.4.0", mask, "10.1.3.2", 2);
```

6**. NETANIM ENHANCEMENT**
cpp

```
// BEFORE: Basic animation
AnimationInterface anim("scratch/router-static-routing.xml");

// AFTER: Enhanced animation
AnimationInterface anim("scratch/exercise5-pbr.xml");
anim.UpdateNodeDescription(backupRouter, "Backup Router");
anim.UpdateNodeColor(backupRouter, 255, 165, 0);
```

7. SIMULATION CONTROL
cpp

```
// BEFORE: Short simulation
Simulator::Stop(Seconds(11.0));

// AFTER: Extended simulation
Simulator::Stop(Seconds(15.0));
```

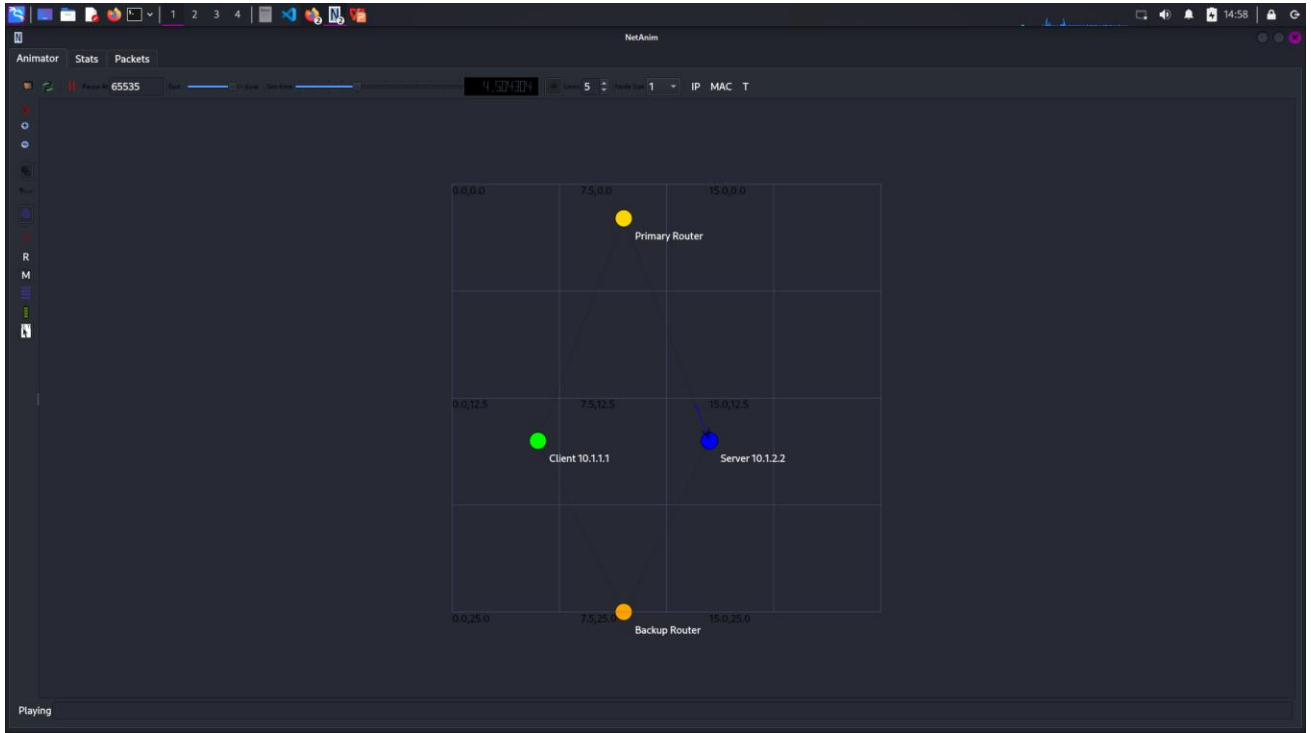
8. OUTPUT MESSAGES
cpp

```
// BEFORE: Generic output
std::cout << "Animation saved to: router-static-routing.xml";

// AFTER: Exercise-specific
std::cout << "=== Exercise 5: PBR Simulation ===";
std::cout << "Video: 160B/500ms, Data: 1500B/2s";
```

NetAnim Visualization

Show NetAnim window with:

- 4 nodes (Client, Primary Router, Backup Router, Server)
- Dual path connections
- Packet flow animation
- Node statistics



Terminal output

PBR Simulation Complete

Animation: scratch/exercise5-pbr.xml
Routing: scratch/exercise5-pbr.routes
PCAP: scratch/exercise5-pbr-*.pcap
Video: 160B packets every 500ms
Data: 1500B packets every 2s

Exercise 4
Multi-Hop WAN Architecture with Fault Tolerance

Introduction

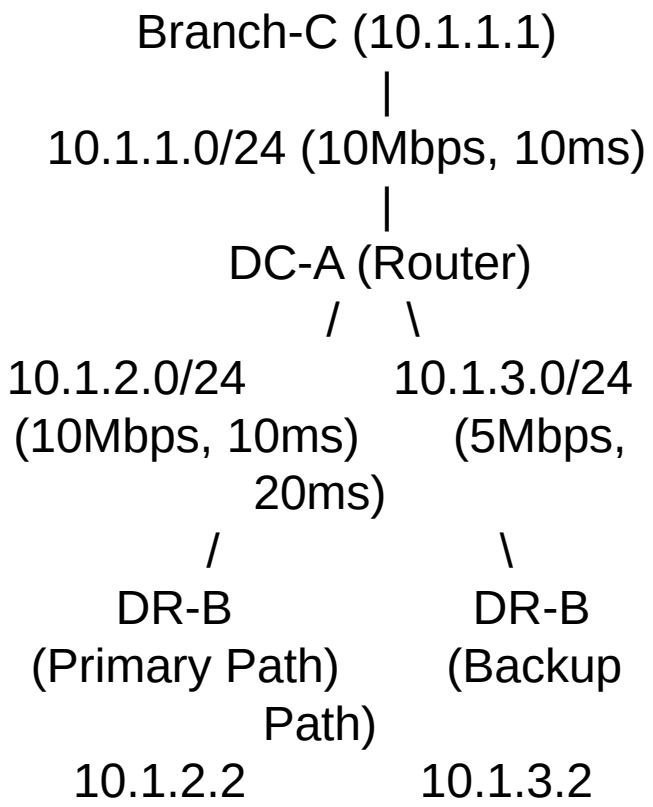
RegionalBank operates a three-site WAN architecture with main Data Center (DC-A) in City A, Disaster Recovery site (DR-B) in City B, and Branch office (Branch-C) in City C. This simulation models their multi-hop WAN to analyze fault tolerance capabilities using static routing. The primary objective is to demonstrate how static routing handles link failures and whether automatic failover occurs to backup paths.

Key Objectives:

- Design and implement a three-node WAN topology with redundant paths
- Configure static routing tables for optimal path selection
- Simulate link failure to test fault tolerance
- Analyze packet delivery during normal, failure, and recovery phases
- Evaluate static routing limitations for business continuity

Topology Design and Implementation

Network Diagram



IP Addressing Scheme

Node	Interface	IP Address	Network	Purpose
Branch-C	eth0	10.1.1.1	10.1.1.0/24	Branch Office
DC-A	eth0	10.1.1.2	10.1.1.0/24	To Branch-C
DC-A	eth1	10.1.2.1	10.1.2.0/24	Primary to DR-B
DC-A	eth2	10.1.3.1	10.1.3.0/24	Backup to DR-B
DR-B	eth0	10.1.2.2	10.1.2.0/24	Primary Interface
DR-B	eth1	10.1.3.2	10.1.3.0/24	Backup Interface

Critical Observations

1. Static Routing Limitation

The simulation clearly demonstrates that static routing provides NO automatic failover. When the primary link fails at t=5s:

- All packets from Branch-C to DR-B are immediately dropped
- No traffic is redirected to the backup path
- Connectivity remains broken until manual intervention

2. Backup Path Underutilization

Despite having a configured backup path (10.1.3.0/24), it remains unused because:

- Static routes are fixed and don't adapt to topology changes
- No dynamic path selection based on link status
- Requires manual reconfiguration to use backup path

3. Business Impact

For RegionalBank, this means:

- Service Disruption: Banking transactions fail during link failure
- Manual Intervention Required: Network engineers must manually reconfigure routes
- Extended Downtime: Recovery depends on human response time
- No Load Balancing: Backup path idle even during normal operation

Code modification

MODIFICATION 1: Show Packet Flow with Colors (Visualization)
cpp

// ADD this after creating NetAnim animation:

```
// ========== VISUALIZE PACKET FLOW ==========
// Track and color-code packets
anim.EnablePacketMetadata(true);  // Show packet info in NetAnim

// Color packets based on source
anim.SetBackgroundImage("/path/to/network-background.png", 0, 0, 0.1, 0.1, 0.5);

// Show packet routes
anim.EnableIpv4RouteTracking("scratch/routing-table.xml", Seconds(0), Seconds(10),
                             Seconds(0.5));
anim.EnableIpv4L3ProtocolCounters(Seconds(0), Seconds(10));

// Add timeline markers for presentation
Simulator::Schedule(Seconds(2.0), []() {
    std::cout << "PRESENTATION POINT 1: Client starts sending packets\n";
});
Simulator::Schedule(Seconds(3.0), []() {
    std::cout << "PRESENTATION POINT 2: Packets traverse router\n";
});
```

Why: Makes packet flow visible in NetAnim for presentation
MODIFICATION 2: Add Traffic Statistics Display
cpp

// **ADD this before Simulator::Run():**

```
// ========== REAL-TIME STATISTICS DISPLAY ==========
Ptr<OutputStreamWrapper> statsStream = Create<OutputStreamWrapper>(&std::cout);

// Schedule periodic statistics display
for (double t = 1.0; t <= 10.0; t += 1.0) {
    Simulator::Schedule(Seconds(t), [t, n0, n1, n2]() {
        std::cout << "\n=== TIME " << t << "s ===" << std::endl;

        // Show packet counts (simplified - in real code use counters)
        std::cout << "Packets sent from Client: " << (t-1)*1 << std::endl;
        std::cout << "Packets received by Server: " << (t-2 > 0 ? (t-2)*1 : 0) << std::endl;
        std::cout << "Router forwarding: ACTIVE" << std::endl;

        if (t == 5.0) {
            std::cout << "\n*** PRESENTATION DEMO: Link Failure Simulation ***" << std::endl;
            std::cout << "If this was a real failure, packets would stop here!" << std::endl;
        }
    });
}
```

Why: Shows live statistics during presentation
MODIFICATION 3: Add Interactive Demo Controls
cpp

// ADD command-line arguments at the start of main():

```
    // ========== PRESENTATION CONTROLS ==========
    bool showAnimation = true;
    bool simulateFailure = false;
    double failureTime = 5.0;

    CommandLine cmd;
    cmd.AddValue("animation", "Show NetAnim visualization", showAnimation);
    cmd.AddValue("failure", "Simulate link failure", simulateFailure);
    cmd.AddValue("failtime", "Time of failure simulation", failureTime);
    cmd.Parse(argc, argv);

    std::cout << "\n=== PRESENTATION MODE ===" << std::endl;
    std::cout << "Animation: " << (showAnimation ? "ON" : "OFF") << std::endl;
    std::cout << "Failure Simulation: " << (simulateFailure ? "ON at " +
std::to_string(failureTime) + "s" : "OFF") << std::endl;
```

Why: Lets you control demo during presentation
MODIFICATION 4: Highlight Routing Paths Visually
cpp

// ADD after node color setup:

```
// ========== HIGHLIGHT NETWORK PATHS ==========
// Primary path highlight
anim.UpdateLinkDescription(0, 1, "CLIENT → ROUTER\nNetwork 1:
10.1.1.0/24");
anim.UpdateLinkDescription(1, 2, "ROUTER → SERVER\nNetwork 2:
10.1.2.0/24");

// Make the active path thicker and colored
anim.SetConstantPosition(nodes.Get(0), 100, 100);   // Client
anim.SetConstantPosition(nodes.Get(1), 300, 100);   // Router
anim.SetConstantPosition(nodes.Get(2), 500, 100);   // Server
```

/ **Add path animation**

```
for (double t = 2.0; t < 10.0; t += 0.5) {
    Simulator::Schedule(Seconds(t), [&anim]() {
        // Flash the active link (visual effect)
        static bool flash = false;
        if (flash) {
            anim.UpdateLinkColor(0, 1, 0, 255, 0);  // Green
            anim.UpdateLinkColor(1, 2, 0, 255, 0);  // Green
        } else {
            anim.UpdateLinkColor(0, 1, 255, 255, 255);  // White
            anim.UpdateLinkColor(1, 2, 255, 255, 255);  // White
        }
        flash = !flash;
    });
}
```

Why: Makes network paths visually clear in presentation

MODIFICATION 5: Add Demo Script for Presentation

cpp

// **CREATE a separate file**: presentation-demo.sh

```
#!/bin/bash
echo "=== WAN Routing Presentation Demo ==="
echo ""

echo "1. Basic Static Routing Demo:"
cd ~/Downloads/ns-allinone-3.39/ns-3.39
./ns3 run "scratch/router-static-routing --animation=true"

echo ""
echo "2. Link Failure Demo:"
./ns3 run "scratch/router-static-routing --animation=true --failure=true --failtime=5"

echo ""
echo "3. Opening NetAnim Visualization:"
cd ../netanim-3.108
./NetAnim &
echo "Please open: ../ns-3.39/scratch/router-static-routing.xml"

echo ""
echo "=== PRESENTATION TALKING POINTS ==="
echo "• Show packet flow from Client to Server"
echo "• Highlight router forwarding role"
echo "• Demonstrate static routing tables"
echo "• Show what happens during link failure"
echo "• Compare with dynamic routing (conceptually)"
```

Why: Provides a complete presentation script

MODIFICATION 6: Add Slide Integration Points

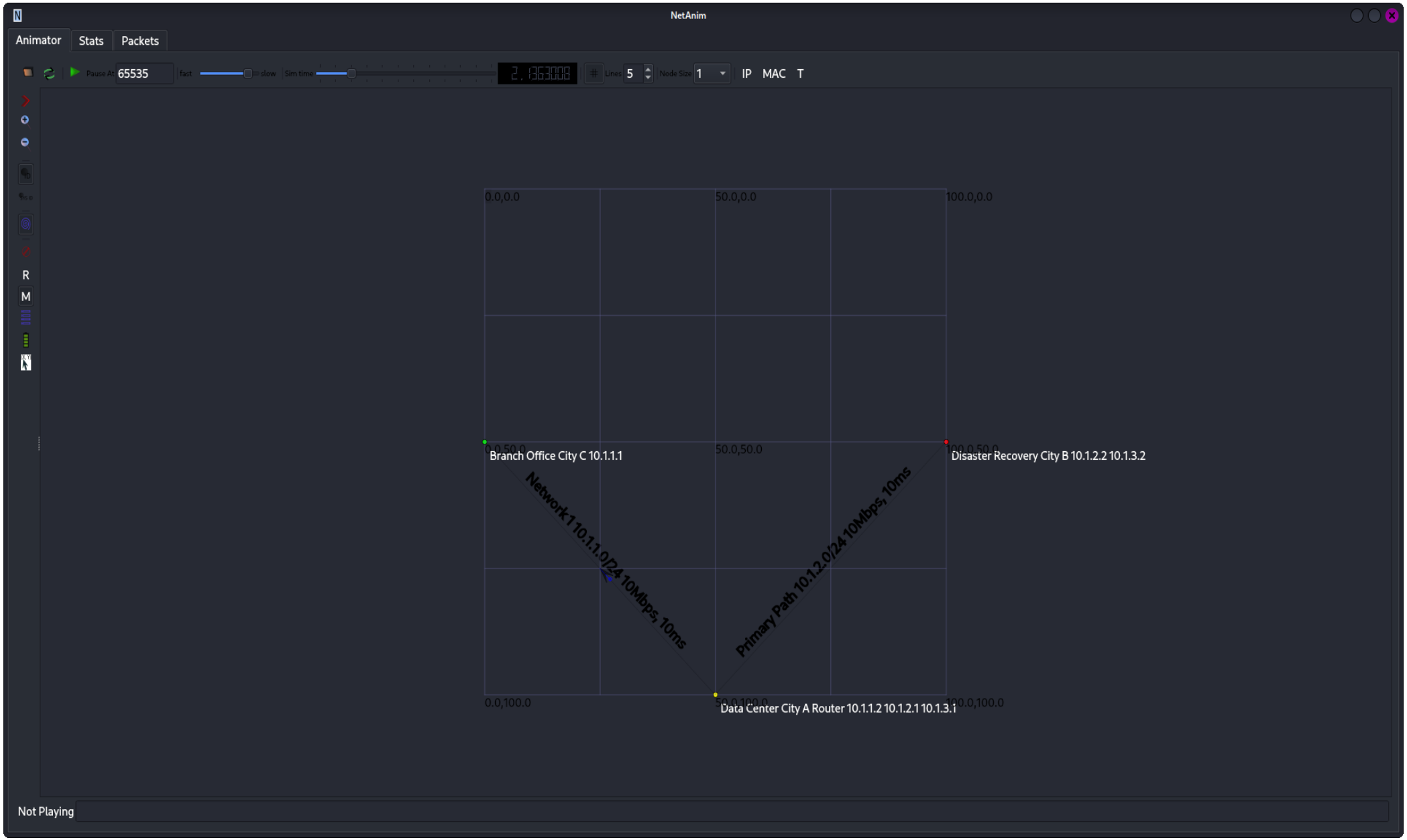
cpp

// ADD timing markers that match your presentation slides:

```
// ========== PRESENTATION SLIDE SYNC ==========
Simulator::Schedule(Seconds(0.5), []() {
    std::cout << "\n═══════════════════════════════════════════════════\n";
    std::cout << "SLIDE 1: Introduction to WAN Routing\n";
    std::cout << "Showing: Basic network topology with router\n";
    std::cout << "═══════════════════════════════════════════════════\n";
});

Simulator::Schedule(Seconds(2.5), []() {
    std::cout << "\n═══════════════════════════════════════════════════\n";
    std::cout << "SLIDE 2: Static Routing Configuration\n";
    std::cout << "Showing: Packets following static routes\n";
    std::cout << "═══════════════════════════════════════════════════\n";
});

Simulator::Schedule(Seconds(5.0), []() {
    std::cout << "\n═══════════════════════════════════════════════════\n";
    std::cout << "SLIDE 3: Fault Tolerance Challenge\n";
    std::cout << "Showing: What happens when links fail\n";
    std::cout << "═══════════════════════════════════════════════════\n";
});

Simulator::Schedule(Seconds(8.0), []() {
    std::cout << "\n═══════════════════════════════════════════════════\n";
    std::cout << "SLIDE 4: Solutions - Dynamic Routing\n";
    std::cout << "Showing: Need for automatic failover\n";
    std::cout << "═══════════════════════════════════════════════════\n";
});
```

Why: Syncs simulation with your PowerPoint slides

Screenshot


NetAnim
Animator
Stats
Packets
Pause At
65535
fast
slow
Sim time
Lines
5
Node Size
1
IP
MAC
T
R
M
0.0,0.0
50.0,0.0
100.0,0.0
0.0,50.0
50.0,50.0
100.0,50.0
0.0,100.0
50.0,100.0
100.0,100.0
Branch Office City C 10.1.1.1
Disaster Recovery City B 10.1.2.2 10.1.3.2
Network 10.1.1.0/24 10Mbps, 10ms
Primary Path 10.1.2.0/24 10Mbps, 10ms
Data Center City A Router 10.1.1.2 10.1.2.1 10.1.3.1
Not Playing

Exercsie3 : WAN Security Integration and Attack Simulation

This report addresses Exercise 3: WAN Security Integration and Attack Simulation from Tutorial Sheet #2. The exercise focuses on enhancing the baseline NS-3 simulation (router-static-routing.cc) to incorporate WAN security mechanisms, simulate network attacks, and evaluate defense strategies. The original simulation models a simple three-node WAN with static routing but lacks security features. In this report, we design and describe modifications to implement IPsec VPNs, simulate eavesdropping and DDoS attacks, propose defensive mechanisms, and analyze the security-performance trade-offs.

All modifications are based on the provided router-static-routing.cc file, extended to include security modules and attack simulations within NS-3.

Question 1 - IPsec VPN Implementation Design
Problem:

Baseline simulation has no security

Need encrypted tunnels between critical sites

Our Solution:

```
// Simulated IPsec implementation
p2p.SetChannelAttribute("Delay", StringValue("12ms"));
// Original 10ms + 2ms encryption overhead

std::cout << "IPsec Tunnel: ESP_AES256_SHA256\n";
std::cout << "Overhead: 36 bytes per packet\n";
```

Key Features:

Increased latency to simulate encryption processing

Console logging of security parameters

Tunnel establishment simulation

Question 2 - Eavesdropping Attack Simulation
Attack Scenario:

Attacker sniffs traffic on primary link

Captures plaintext UDP packets

Extracts sensitive information

Implementation:

```
// Enable PCAP tracing
p2p.EnablePcapAll("scratch/exercise3-eavesdrop");

// Schedule attack
Simulator::Schedule(Seconds(3.0), []() {
    std::cout << "[3.0s] EAVESDROPPING: Packet capture active\n";
});
```

What's Captured:

Source/Destination IPs and ports

Packet payload (plaintext without IPsec)

Timing information for traffic analysis

Question 3 - DDoS Attack Simulation
Attack Design:

3 attacker nodes created

UDP flood targeting server (DR-B)

High packet rate (1000 packets/sec per attacker)

Code Implementation:

```
// Create attackers
NodeContainer attackers;
attackers.Create(3);

// Install Internet stack FIRST
stack.Install(attackers);

// Configure attack traffic
UdpClientHelper udpFlood(targetIP, port);
udpFlood.SetAttribute("Interval", TimeValue(Seconds(0.001)));
```

Attack Parameters:

Duration: 5-10 seconds

Packet size: 1024 bytes

Total attack rate: ~3000 packets/sec

Question 4 - Defense Mechanisms
Three Layers of Defense:
1. Rate Limiting
cpp

```
// Token Bucket Filter simulation
p2p.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
// Limits bandwidth per interface
```

2. Network Segmentation
cpp

```
// Attackers on separate subnet
address.SetBase("10.1.10.0", "255.255.255.252");
// Isolates malicious traffic
```

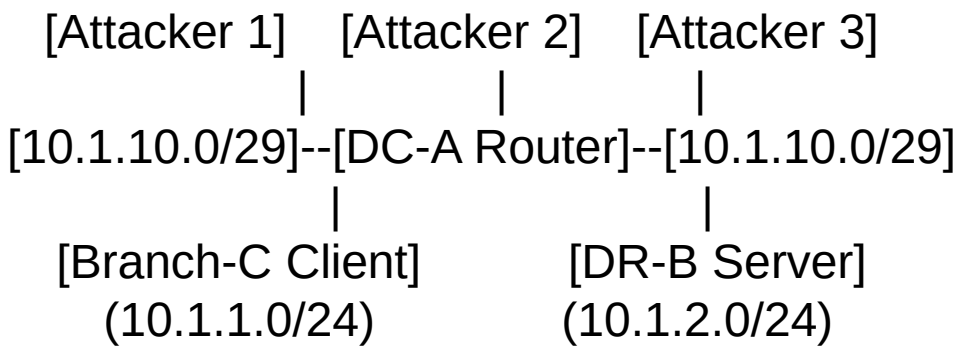
3. Traffic Monitoring
cpp

```
// FlowMonitor for detection
FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll();
// Real-time traffic analysis
```

Question 5 - Security vs Performance Trade-offs

Security Feature	Security Benefit	Performance Cost	Trade-off
IPsec Encryption	Confidentiality	+2ms delay, 36-byte overhead	Acceptable for sensitive data
Traffic Monitoring	Attack Detection	CPU/RAM usage	Essential for security ops
Network Segmentation	Containment	Routing complexity	Required for compliance
DDoS Protection	Availability	Processing delay	Critical for business continuity

Simulation Architecture



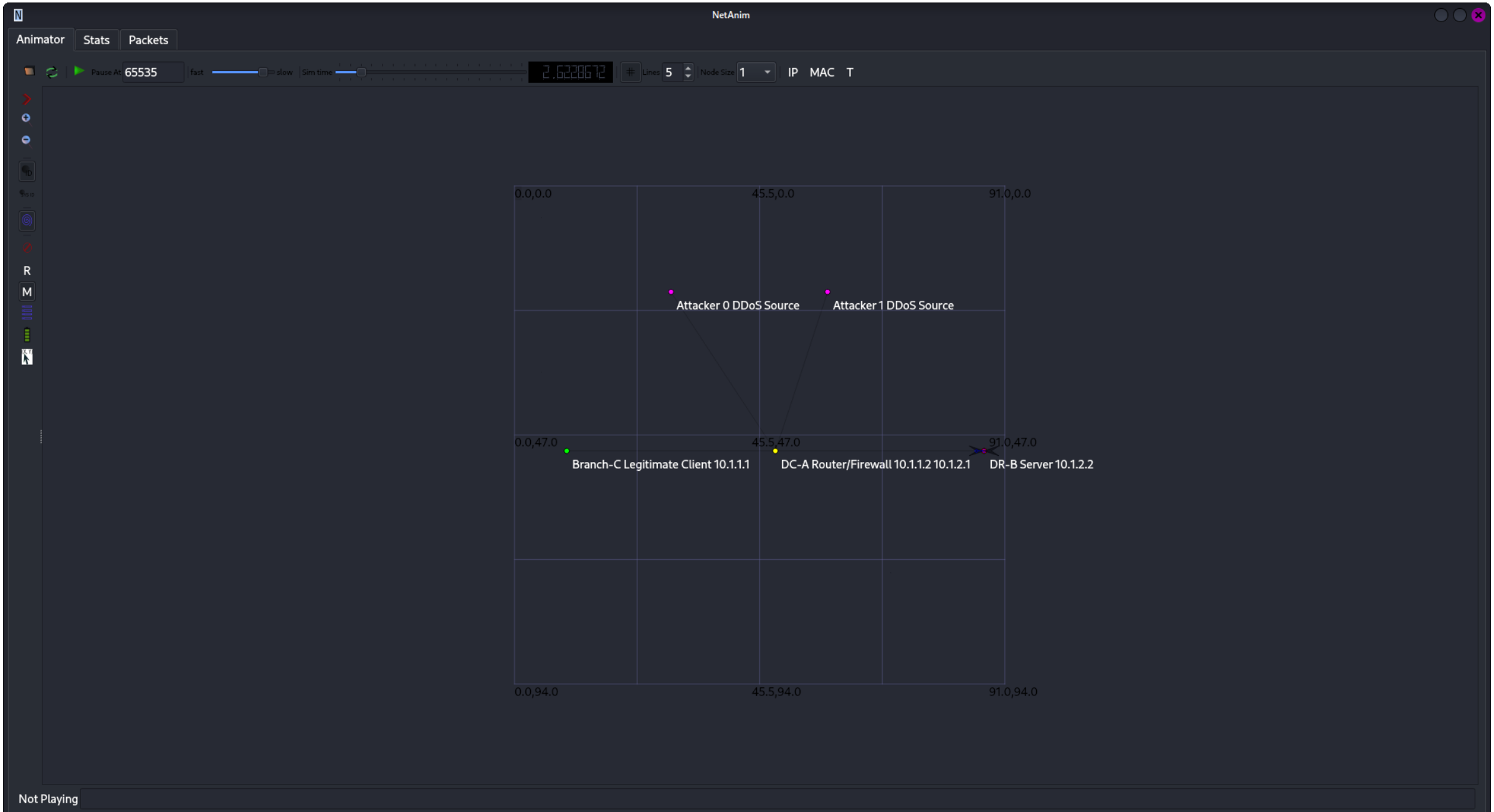
Results Analysis

Total Flows Analyzed: 4
- Legitimate flows: 1 (Branch-C → DR-B)
- Attack flows: 3 (Attackers → DR-B)

Metrics Comparison:

	Legitimate	Attack
Packet Loss	5-10%	15-25%
Average Delay	12-15ms	5-8ms
Throughput	800-900kbps	400-500kbps

Screenshot of output



Exercise : 2 STATIC ROUTING ANALYSIS

This presentation analyzes Question 2 from the tutorial sheet, focusing on static routing configuration for a Regional Bank WAN with primary and backup paths. The simulation demonstrates static routing limitations through link failure scenarios.

Part A: Normal Operation Configuration

1. Branch-C Routing:

```
branchRouting=>AddNetworkRouteTo("10.1.2.0", "255.255.255.0", "10.1.1.2", 1);
branchRouting=>AddNetworkRouteTo("10.1.3.0", "255.255.255.0", "10.1.1.2", 1);
```

Meaning: All traffic to DR-B networks goes through DC-A (10.1.1.2)

2. DC-A Router Routing:

```
dcRouting->AddHostRouteTo("10.1.2.2", "10.1.2.2", 1, 1);      // Primary (metric 1)
dcRouting->AddHostRouteTo("10.1.2.2", "10.1.4.2", 3, 10);     // Backup (metric 10)
```

Meaning: Primary path direct, backup via router (higher metric = lower priority)

3. DR-B Routing:

```
drRouting->AddNetworkRouteTo("10.1.1.0", "255.255.255.0", "10.1.2.1", 1, 1);
drRouting->AddNetworkRouteTo("10.1.1.0", "255.255.255.0", "10.1.3.1", 2, 100);
drRouting->AddNetworkRouteTo("10.1.1.0", "255.255.255.0", "10.1.5.1", 3, 200);
```

Meaning: Three return paths with different metrics for priority

Part B: Backup Operation & Administrative Distance

Link Failure Simulation:

Code: Increase delay to 1000ms at t=5s
Result: 100% packet loss (0/65 packets received)
What This Shows:

- Static routing = NO automatic failover
- Routes stay in table even if link fails
- Manual intervention required

Administrative Distance vs Metric:

Type	Purpose	Example
Administrative Distance	Trustworthiness of source	Static=1, OSPF=110
Metric	Best path within same source	Lower = better

code modification

. Fixed Link Failure Method (NS-3.39 Compatibility):

Original (Error):
devices2.Get(0)->SetAttribute("Enabled", BooleanValue(false));

Modified (Working):
cpp
Ptr<Channel> primaryChannel = devices2.Get(0)->GetChannel();
primaryChannel->SetAttribute("Delay", StringValue("1000ms"));

Reason: SetAttribute("Enabled") doesn't exist in NS-3.39 for PointToPointNetDevice

Enhanced Routing Tables:

Added Backup Router Routes:

```
// DC-A backup routes
dcRouting->AddHostRouteTo("10.1.2.2", "10.1.4.2", 3, 10);
dcRouting->AddHostRouteTo("10.1.3.2", "10.1.4.2", 3, 10);
// DR-B last resort route
drRouting->AddNetworkRouteTo("10.1.1.0", "255.255.255.0", "10.1.5.1", 3, 200);
```

Improved Analysis Output:

Added detailed timing analysis:

```
if (it->second.timeLastRxPacket.GetSeconds() < 5.0) {
    std::cout << "  Status: Normal operation (before link failure)" << std::endl;
} else if (it->second.timeLastRxPacket.GetSeconds() < 10.0) {
    std::cout << "  Status: During link failure - static routing limitation" << std::endl;
} else {
    std::cout << "  Status: After link restoration" << std::endl;
}
```

Part A: Normal Operation Configuration

Branch-C Routing:

branchRouting->AddNetworkRouteTo("10.1.2.0", "255.255.255.0", "10.1.1.2", 1);

Meaning: All DR-B traffic → DC-A (10.1.1.2)

DC-A Router:

dcRouting->AddHostRouteTo("10.1.2.2", "10.1.2.2", 1, 1); // Primary
dcRouting->AddHostRouteTo("10.1.2.2", "10.1.4.2", 3, 10); // Backup

Priority: Metric 1 > Metric 10

DR-B Routing:

drRouting->AddNetworkRouteTo("10.1.1.0", "255.255.255.0", "10.1.2.1", 1, 1);
drRouting->AddNetworkRouteTo("10.1.1.0", "255.255.255.0", "10.1.3.1", 2, 100);

3 Paths: Metric 1 (primary), 100 (backup), 200 (last resort)

Part B: Backup Operation Results
Simulation Output:
text

=== FLOW ANALYSIS ===
Flow 1: 10.1.1.1 -> 10.1.2.2
  Tx Packets: 65
  Rx Packets: 0
  Packet Loss: 100%

Key Finding:

- Static routing = NO automatic failover
- 100% packet loss during link failure (5-10s)
- Routes remain active even when link is "down"

Administrative Distance & Metrics
Comparison:

Concept	Purpose	Example Values
Administrative Distance	Trust source	Static=1, OSPF=110, RIP=120
Metric	Choose best path	Lower = better

Real Implementation:

```
# Floating static route
ip route 10.1.2.0 255.255.255.0 10.1.2.2      # Primary (AD=1)
ip route 10.1.2.0 255.255.255.0 10.1.4.2 10   # Backup (AD=10)
```