Exercise 5 — Policy-Based Routing (PBR) for Application-Aware Path Selection (Difficult)

Q1 — Traffic classification logic (Flow_Video & Flow_Data)

Flow_Video (RTP-like):

- Transport: UDP
- Packet size: 200 bytes (RTP + headers + payload)
- Interval: 20 ms
- Application: custom `OnOff` with `PacketSize=200`, `OnTime=constant`, `OffTime=0`, or periodic UDP sender.

Flow_Data (FTP-like):

- Transport: TCP or UDP (bulk)
- Packet size: 1500 bytes
- Bursty: e.g., OnOff with `DataRate=5Mbps`, `OnTime=0.5s`, `OffTime=1s` to emulate bursts, or `BulkSendApplication` to fill a socket.

Implementation in ns-3:

- Create separate sockets/apps with distinct destination ports, which will be used by classifier to identify the flow.
- Set DSCP for Flow_Video to a high priority value (e.g., EF).

Q2 — Implementing PBR in ns-3 (architectural approach)

NS-3 lacks a built-in PBR helper, so implement PBR by intercepting packet forwarding at the router node (n1) and applying custom forwarding decisions.

Approach:

1. **Classification:** Use `Socket` destination port or the DSCP/TOS in IPv4 header. A packet arriving at the router triggers a `Receive` callback registered on the router's NetDevice (RX hook). Extract packet headers (`Ipv4Header`, UDP/TCP header) and classify.
2. **Forwarding decision:** After classification:
    - For `Flow_Video`: choose next-hop A (interface id = if_primary) unless metrics indicate failover.
    - For `Flow_Data`: choose next-hop B (secondary interface).

3. **Action:** Modify the `Ipv4Header`'s destination next hop or send the packet out the chosen `NetDevice` by calling `SendFrom()` on that device, or use `Ipv4::RouteOutput()` hack to forward via desired route.

Pseudocode for packet processing:

```
void PBRReceiveCallback(Ptr<NetDevice> dev, Ptr<const Packet> p, uint16_t
protocol, const Address &from, const Address &to, NetDevice::PacketType
packetType) {
    Ptr<Packet> packet = p->Copy();
    Ipv4Header iph;
    packet->PeekHeader(iph);
    uint8_t tos = iph.GetTos();
    uint16_t dport = 0;
    if (iph.GetProtocol() == Udp) {
       UdpHeader udph;
       packet->PeekHeader(udph);
       dport = udph.GetDestinationPort();
    }
    // Classification
    bool isVideo = (dport == VIDEO_PORT) || ((tos >> 2) == DSCP_EF); // DSCP
in top bits
    // Choose next interface
    uint32_t outIf = isVideo ? primaryIfIndex : secondaryIfIndex;
    // Forward: build a route or use Ipv4::Send to chosen device
    Ptr<Ipv4> ipv4 = router->GetObject<Ipv4>();
    Ptr<Ipv4Route> route = Create<SimpleRoute>(nextHopAddress, outIf);
    ipv4->Send(packet, iph.GetSource(), iph.GetDestination(),
iph.GetProtocol(), route);
}
```

Notes:

- You have to be careful not to break normal routing of control packets (ICMP, ARP).
- For production-level PBR, implement a forwarding table inside the node that maps flow identifiers to out interfaces.

Q3 — Path characterization (real-time metrics for PBR decisions)

Tools inside ns-3:

- Use `Ipv4L3Protocol` trace sources to monitor `Tx`, `Rx`, and `Drop` events per interface.
- Use `FlowMonitor` to measure per-flow latency throughput.
- Measure per-interface queue occupancy via `QueueDisc` trace sources (`Enqueue`, `Dequeue`, `Drop`) to infer available bandwidth and delay.

Making metrics available to the PBR function:

- Implement metrics collector objects that subscribe to traces and keep moving averages (e.g., last 1s average RTT/latency, average queue length, measured throughput).

- The PBR function polls this collector periodically (e.g., every 1s) and updates its decision logic.

Q4 — Dynamic Policy Engine (SD-WAN like controller)

Periodic function (every 1s):

- Gather path metrics for `primary` and `secondary` links (latency, loss).
- If `Flow_Video` latency on primary > 30 ms → switch `Flow_Video` to secondary.

C++ class structure (outline):

```
class PolicyController : public Object {
private:
   Ptr<Node> router;
   uint32_t primaryIf, secondaryIf;
   MetricsCollector metrics;
   EventId periodicEvent;
public:
   void Start();
   void Stop();
   void EvaluatePolicies(); // called every 1s
   void PushForwardingChange(bool videoToSecondary);
};
```

`EvaluatePolicies()` reads metrics and calls `PushForwardingChange()` which manipulates the node's PBR forwarding table (e.g., change next-hop for flowID or update classification mapping).

Q5 — Validation & trade-offs

Validation:

- Create controlled experiments: force primary link latency to increase and assert that video flows switch to secondary (observe via packet traces and FlowMonitor).
- Use `pcap` or packet counters on the selected egress interface to verify flows go out on intended interface.
- Compare metrics (latency, jitter) before/after policy enforcement to confirm improvement.

Computational overhead vs real hardware:

- The in-simulation PBR engine runs as part of the event loop and adds CPU overhead proportional to the number of flow classifications and frequency of policy evaluation. In real routers, PBR often runs in control plane with hardware-assisted fastpath, so real routers are far more efficient.

- As flow count increases, simulation CPU/time cost grows quickly — a scalability bottleneck. For many flows (> few thousands), consider using aggregated policies (class-based) rather than per-flow PBR.