Exercise 6 — Inter-AS Routing (BGP) (Difficult)

Q1 — Modeling AS in ns-3

a) Logical grouping:

- Represent each AS by a set of nodes and logical configuration flags (store AS number in a node attribute).
- Use route-origin or route-map metadata fields in your BGP simulation objects.

b) Enforce internal routing:

- Use an IGP (e.g., OSPF via DCE or OLSR if appropriate) inside each AS to resolve internal next hops and only advertise AS-level routes to peers.

c) Peering links:

- Model peering nodes (edge routers) and set two links to simulate IXPs (IXP-A and IXP-B). Configure BGP sessions between the peering routers.

Q2 — BGP path attribute data structure

A simplified C++ structure for a BGP announcement:

```
struct BgpAnnouncement {
    Ipv4Prefix prefix;                  // e.g., "192.168.0.0/16"
    std::vector<uint32_t> asPath;       // AS_PATH, e.g., {65002, 65003}
    uint32_t localPref;                 // LOCAL_PREF
    uint32_t med;                       // MULTI_EXIT_DISC
    uint32_t originatorId;              // origin router id
    uint32_t nextHop;                   // next-hop IP address
};
```

Selection logic: node compares `localPref` (higher preferred), then `AS_PATH` length (shorter preferred), then `MED` (lower preferred), and finally tiebreakers (e.g., lowest router ID).

Q3 — BGP decision process (pseudocode)

```
onReceiveAnnouncement(neighbor, announcement):
    existing = RIB_In.lookup(announcement.prefix)
    if existing == NULL:
        RIB_In.add(announcement from neighbor)
        Best = selectBestRoute(RIB_In[announcement.prefix]) // see policy
below
        if Best changed:
            updateFIB(Best.nextHop)
            advertiseToNeighbors(Best) // apply export filters
    else:
        // store/update neighbor's announcement
        RIB_In.update(neighbor, announcement)
        Best = selectBestRoute(RIB_In[announcement.prefix])
        if Best changed:
            updateFIB(Best.nextHop)
            advertiseToNeighbors(Best)

function selectBestRoute(routes):
    // implement simplified BGP steps:
    // 1) Highest LOCAL_PREF
    // 2) Shortest AS_PATH length
    // 3) Lowest originator id or lowest MED depending on policy
    // 4) Lowest neighbor IP as tie-breaker
    sort routes by (localPref desc, asPath.length asc, med asc,
originatorId asc)
    return first route
```

Focus on logical steps rather than API calls; the core idea is maintain per-prefix RIB-in from all neighbors, compute best path, and update forwarding table.

Q4 — Simulating a route leak

How to create: make a router in AS65002 incorrectly advertise a prefix owned by AS65001 back to AS65001 or to other peers, i.e., inject an announcement with an `AS_PATH` that does NOT contain the true origin.

Example malicious AS_PATH (route leak):

- Suppose prefix `10.1.0.0/16` originates in AS65001.
- A misconfigured or malicious router in AS65002 advertises: `AS_PATH = [65002]` or even `AS_PATH = [65002, 65003]` (omitting 65001 as origin), claiming origin.
- The advertisement back into AS65001 may include AS_PATH `[65002]` causing confusion.

How nodes react with decision logic from Q3:

- If the leaked announcement shows a shorter AS_PATH or higher LOCAL_PREF (e.g., due to export policy), some nodes might prefer the leaked route. Nodes that prefer the leak will install it in FIB and start forwarding to the new next-hop, potentially causing misdirection or traffic blackholing.

Whether they prefer it: depends on the BGP decision process (locals with higher LOCAL_PREF or shorter AS_PATH may pick the leaked route). Real BGP has safeguards but route leaks can be impactful. Thus in your simulation, set attributes to demonstrate potential pref selection.

Q5 — From simulation to reality (limitations)

Three critical BGP features hard to model in ns-3:

1. **Route reflection & confederations** — require modeling of complex decision policies and large scale path reflection semantics; harder because ns-3 core focuses on packet forwarding rather than full BGP policy machinery.
2. **Communities & large policy frameworks** — BGP communities are used for complex routing policies; modeling network operators' policy logic and policy repositories is complex to emulate faithfully.
3. **Real BGP implementations (timers, TCP session quirks, MRAI timers, route flap damping tuning)** — robust implementations (Quagga/FRR) have many corner cases; modeling all nuances in ns-3 is difficult.

Conclusion: ns-3 is good for *conceptual* inter-AS routing research, prototyping simplified BGP logic, and measuring forwarding impact. For realistic operator-grade BGP experiments, combining ns-3 with DCE to run real route daemons (Quagga/FRR) or using specialized BGP testbeds (e.g., ExaBGP, real routers) is more appropriate. Community modules (e.g., simplified OSPF/OSPF DCE integrations) can extend ns-3 for deeper protocol research.

Final notes, references & how to run / test

- The answers above are mapped to the tutorial exercises in your uploaded file (tutorial sheet).

  ISN 3132 Wide Area Networks

- For NS-3 API details: `InternetStackHelper`, `Ipv4StaticRouting`, `Ipv4RoutingHelper`, `FlowMonitorHelper`, `TrafficControlHelper`, and `PointToPointHelper` are the main helpers used in code snippets above.
- For **dynamic routing / OSPF** in ns-3: there's no canonical core `OspfHelper` in mainstream ns-3 releases; you should either use DCE + Quagga/FRR or import an external experimental OSPF module if you need realistic OSPF. See community modules / DCE references.