Department of Mathematics and Computer Science

P2P and Wireless networks course
final assignment report

LRA routing implementation in NS3

13/02/2025

Student:
Gabriele Messina (1000052663)

University of Catania

Contents

1 Introduction

Routing in wireless ad hoc networks presents significant challenges due to the dynamic nature of nodes and the absence of centralized control. This study investigates the performance of the **Link Reversal Algorithm** (LRA[2]) in a simulated environment using the **ns-3 network simulator**[1]. The simulation parameters include varying node densities and network area sizes, with a focus on packet loss rates. The results provide insights into the **scalability** and robustness of LRA in different network conditions. The context of this work is **disaster recovery scenarios** where traditional communication infrastructure, such as cellular antennas, is unavailable, and a peer-to-peer (P2P) network is used for communication.

The source code for this study is available on GitHub: `https://github.com/GabrieleMessina/p2p_wireless_networks_course`

2 Link Reversal Algorithm

The **Link Reversal Algorithm** (**LRA**[2]) is a distributed routing protocol mainly used in dynamic and mobile network environments. It is particularly suited for ad hoc and wireless networks where network topology changes frequently. The algorithm operates by reversing the direction of links between nodes to restore lost routes, ensuring uninterrupted connectivity even in the presence of frequent topology changes.

LRA relies on the concept of maintaining a **Directed Acyclic Graph** (**DAG**) rooted at a designated sink node. In a DAG, each node has links pointing towards one or more downstream nodes, with the sink node serving as the ultimate destination. When a link failure occurs, LRA dynamically reverses the direction of affected links, initiating a localized response to restore the DAG structure. This process avoids the need for global routing updates, reducing control overhead and improving scalability.

The operation of LRA can be divided into four main stages. First, during initialization, the network establishes a DAG, ensuring that all nodes can route packets to the sink. Second, nodes consistently observe their connections for faults, particularly the inability to reach the sink, using mechanisms such as acknowledgment timeouts or signal strength analysis. Third, upon detecting a failure, affected nodes reverse their outgoing links, which may propagate asynchronous reversals to neighbouring nodes until the DAG is restored. Finally, the network stabilizes into a new DAG, ready to route subsequent packets.

The decentralized nature of LRA offers several advantages. By enabling nodes to make routing decisions independently based on local information, LRA reduces communication overhead and enhances the protocol's adaptability to dynamic conditions. In addition, proactive maintenance of a DAG minimizes latency, ensuring that routes are immediately available for data transmission.

However, LRA is not without limitations. Frequent topology changes can increase convergence time, delaying route stabilization.

Despite these challenges, LRA and derivative algorithms remain effective solutions for scenarios requiring high adaptability and fault tolerance. The algorithm's ability to dynamically adjust to changing network conditions makes it particularly valuable for applications in disaster recovery, military communications, and vehicular networks, where reliable connectivity is crucial.

3 Network Simulator 3

Network Simulator 3 (**NS3**[1]) is a discrete-event network simulator widely used for research and educational purposes in the field of networking. Designed as an open-source project, NS3 provides a modular and flexible framework for simulating and analyzing a wide variety of networking scenarios, ranging from wired and wireless networks to internet protocols and routing algorithms.

NS3 is built in C++ and provides Python bindings, making it accessible to a broad user base. It is specifically designed to facilitate realistic simulations by incorporating detailed models of network devices, protocols, and mobility patterns. NS3 supports various features, including the simulation of IP-based networks, wireless communication technologies such as Wi-Fi, LTE, and 5G, and many routing protocols like AODV, DSDV, and custom implementations.

NS3 is widely employed in academia and industry for protocol design and testing, performance analysis of existing protocols, and educational purposes. It enables optimization studies by providing detailed metrics for performance evaluation, including throughput, latency, and packet loss. In this study, NS3 was chosen as the simulation platform for implementing and evaluating the LRA routing protocol. Its flexibility and scalability enabled the modeling of a dynamic network environment with +100 nodes, random mobility, and varying traffic patterns. The detailed libraries and customizable features of NS3 facilitated the integration of the LRA protocol, ensuring realistic simulations and accurate performance metrics.

4 Work

4.1 Simulation Setup

The simulation is designed to work with input parameters to assess the performance of the LRA protocol. The key simulation parameters include a variable number of nodes, deployed within a square area of chosen dimensions. Nodes are positioned uniformly at random to ensure a realistic network topology.

To introduce mobility, a **random walk model** is employed, enabling nodes to move freely within the designated area. The speed attribute of the walk model follows a Pareto distribution to simulate a context in which a few nodes move with a vehicle while the majority move by foot. This setup reflects real-world disaster recovery scenarios where emergency responders or supply vehicles travel faster than individuals navigating on foot. At the beginning of the simulation, a **sink node** is designated, serving as the central recipient of network messages.

Each node randomly transmits a fixed number of **echo messages** to the sink exploiting NS3 **UdpEchoServerHelper** and **UdpEchoServerClient** fuctionalities and emulating data communication within the network. The total simulation duration is set from a parameter provided by the user.

The **LRA routing protocol** governs routing decisions, utilizing **link reversal mechanisms** to dynamically adapt to changes in node positions. The success or failure of packet transmissions is monitored, with **packet loss** recorded as a critical metric.

Here is a comprehensive list of the simulation parameters:

- Number of Nodes
- Number of Packets
- Area Side Length
- Simulation Time

```cpp
void LraExample::SetupNodes(int nNodes, double squareSize, double totalTime){
    ns3::NodeContainer nodes(nNodes);
    MobilityHelper mobility;
    mobility.SetPositionAllocator(
      "ns3::RandomRectanglePositionAllocator",
      "X", StringValue("ns3::UniformRandomVariable[Min=0|Max="+squareSize+"]"),
      "Y", StringValue("ns3::UniformRandomVariable[Min=0|Max="+squareSize+"]")
    );

    mobility.SetMobilityModel(
      "ns3::RandomWalk2dMobilityModel",
      "Mode", StringValue("Time"),
      "Time", StringValue(std::to_string(totalTime) + "s"),
      // Pareto: mean = 12, median = 8, min = 6, max = 80
      "Speed", StringValue("ns3::ParetoRandomVariable[Bound=80|Scale=6|Shape=2]"),
      "Bounds", RectangleValue(Rectangle(0, squareSize, 0, squareSize))
    );
    mobility.Install(nodes);
}
```

Listing 1: Simulation Node Initialization

```cpp
void LraExample::SetupNetwork(){
    for (uint32_t i = 0; i < nodes.GetN(); ++i)
    {
        Ptr<Node> node = nodes.Get(i);
        Ptr<LraRoutingProtocol> lraRouting = node->GetObject<LraRoutingProtocol>();
        lraRouting->InitializeNode(m_sinkAddress);
    }

    int echoPort = 9;
    UdpEchoServerHelper echoServer(echoPort);
    ApplicationContainer echoServerApps = echoServer.Install(nodes);
    echoServerApps.Start(Seconds(0));
    echoServerApps.Stop(Seconds(totalTime));

    UdpEchoClientHelper echoClient(Address(m_sinkAddress), echoPort);
    echoClient.SetAttribute("MaxPackets", UintegerValue(n_packets));
    echoClient.SetAttribute("Interval", TimeValue(Seconds(1.0)));
    echoClient.SetAttribute("PacketSize", UintegerValue(32));

    for (uint32_t i = 0; i < nodes.GetN() - 1; ++i)
    {
        auto app = echoClient.Install(nodes.Get(i));
        int randDelay = rand() % 1000;
        app.Start(Seconds(startDelay + randDelay));
        app.Stop(Seconds(totalTime));
    }

    // Trace package send and receive to calculate loss
    Config::Connect(
        "/NodeList/*/ApplicationList/*/$ns3::UdpEchoServer/RxWithAddresses",
        MakeCallback(&LraExample::LogMessageResponse, this)
    );
    Config::Connect(
        "/NodeList/*/ApplicationList/*/$ns3::UdpEchoClient/TxWithAddresses",
        MakeCallback(&LraExample::LogMessageSend, this)
    );
}
```

Listing 2: Application Simulation Initialization

4.2 Network Configuration

The network is established using an **IEEE 802.11ac WiFi standard** with an **ad hoc** configuration. The wireless network is configured as follows:

- The MAC layer is set to **AdhocWifiMac**, ensuring decentralized network operation.
- The physical layer uses the **YansWifiPhy** model, with parameters set for carrier sense threshold and sensitivity at **zero**, optimizing the detection of nearby transmissions.
- The wireless channel follows the **Default YansWifiChannelHelper** configuration.
- The remote station manager utilizes **IdealWifiManager**, optimizing the transmission rate selection.

The Internet stack is deployed as follows:

- Nodes are assigned IP addresses within the **10.0.0.0/8** subnet.
- The sink node is designated as the last node in the list and assigned the highest IP address.

```
void LraExample::SetupNetwork(){
    WifiMacHelper wifiMac("ns3::AdhocWifiMac");

    YansWifiPhyHelper wifiPhy;
    wifiPhy.Set("CcaEdThreshold", DoubleValue (0));
    wifiPhy.Set("CcaSensitivity", DoubleValue (0));

    auto wifiChannel = YansWifiChannelHelper::Default();
    wifiPhy.SetChannel(wifiChannel.Create());

    WifiHelper wifi;
    wifi.SetStandard(WIFI_STANDARD_80211ac);
    wifi.SetRemoteStationManager("ns3::IdealWifiManager");
    wifi.Install(wifiPhy, wifiMac, nodes);

    stack.SetRoutingHelper(LraHelper());
    stack.Install(nodes);

    Ipv4AddressHelper address;
    address.SetBase("10.0.0.0", "255.0.0.0");
    ipv4Interfaces = address.Assign(netDevices);
    m_sinkAddress = ipv4Interfaces.GetAddress(nodes.GetN() - 1);
}
```

Listing 3: Simulation Network Setup

4.3 Routing protocol Implementation

To implement the proposed protocol, a new **NS3** component, **LraRoutingProtocol**, has been developed. This component extends the **Ipv4RoutingProtocol** class provided by the NS3 framework.

The routing table is maintained using a **map data structure**, which establishes a correlation between an **IPv4 address** and its corresponding **LRA Link Status**.

Upon attachment of the routing protocol to a node, the node initiates a **broadcast service packet** known as the **Hello Message**. It subsequently awaits responses from neighboring nodes, which reply with a **Hello Response Message**. This exchange enables the node to **discover its neighbors** and construct its **routing table** accordingly.

When a **non-service packet** is received, the node determines whether it is the intended destination. If the node is the **destination**, the protocol terminates successfully. If the node is **not** the destination or if it is **creating and forwarding a new packet** towards the sink, it searches for the **next hop** within its routing table. If no valid next hop is found, the **Link Reversal** mechanism is triggered, and the search process is repeated. If, after multiple attempts, no next hop is identified, the node **disables itself**, as it is determined to be part of a subgraph that is **disconnected from the sink**.

When a **service packet** is received, the node processes it based on the **packet type**, as follows:

- **Hello Message**: The receiving node updates its routing table by adding the sender with a **link type of "Outgoing"** if the sender's **IP address** is **greater** than that of the recipient; otherwise, the link is classified as **"Ingoing"**.
- **Hello Response Message**: The behavior is identical to that of the **Hello Message**, where the link type is determined based on the comparison of IP addresses.
- **Ack Request Message**: The receiving node responds by transmitting an **Ack Response Message**.
- **Ack Response Message**: Upon reception, the node halts the execution of the **DisableLinkTimer** associated with the sender.
- **Link Reversal Message**: The receiving node updates its routing table by designating the **link towards the sender** as **"Ingoing"**. Additionally, it evaluates whether a **Link Reversal** is necessary.

```
void LraRoutingProtocol::LinkReversal()
{
    // Base case
    if (m_nodeAddress == m_sink)
    {
        return;
    }

    // Actual inversion
    for (auto neighbor : m_neighbors)
    {
        m_linkStatus[neighbor] = 1; //Outgoing
    }
}
```

Listing 4: Link Reversal Method

```
Ipv4Address LraRoutingProtocol::GetNextHop(){
    auto nextHop = GetFirstOutgoingLink();
    if(nextHop != m_broadcastAddress){
        return nextHop;
    }
    else{
        if(nextHop == m_broadcastAddress && m_neighbors.size() > 0){
            LinkReversal();
            auto nextHop = GetFirstOutgoingLink();
            // Notify all nodes that you are now in active state.
            SendReversalMessage(m_broadcastAddress);
            return nextHop;
        }
    }

    return m_broadcastAddress; // fallback address
}
```

Listing 5: Next Hop Find Method

```
void LraRoutingProtocol::RecvLraServicePacket(Ptr<Packet> packet, Ipv4Address origin)
{
    std::string payload = GetPacketPayload(packet);

    // Ack Request Message received
    if (payload == LraRoutingProtocol::LRA_ACK_SEND_MESSAGE)
    {
        if (m_linkStatus[origin] == 1) // Cycle detected
        {
            DisableLinkTo(origin);
            return;
        }
        SendAckResponseMessage(origin);
    }
    // Ack Response received
    else if (payload == LraRoutingProtocol::LRA_ACK_RECV_MESSAGE)
    {
        m_disableLinkToEvent[origin].Cancel(); // link active, avoid disabling it
        m_disableLinkToEvent.erase(origin);
    }
    // Hello Message received
    else if (payload == LraRoutingProtocol::LRA_HELLO_SEND_MESSAGE)
    {
        if (m_nodeAddress < origin)
            EnableLinkTo(origin);
        else
            DisableLinkTo(origin, true);

        SendHelloResponseMessage(origin);
    }
    // Hello Message Response received
    else if (payload == LraRoutingProtocol::LRA_HELLO_RECV_MESSAGE)
    {
        if (m_nodeAddress < origin)
            EnableLinkTo(origin);
        else
            DisableLinkTo(origin, true);
    }
    // Link Reversal Message received
    else if (payload == LraRoutingProtocol::LRA_REVERSAL_SEND_MESSAGE)
    {
        DisableLinkTo(origin);
    }
}
```

Listing 6: Service Packet Received Handling Method

5 Results

For our simulation we have chosen the following parameters:

- **Number of nodes**: Ranging from 2 to 200.
- **Simulation area**: A square area with side lengths varying from 1 to 200 units.
- **Number of packets sent**: Each node transmits 3 packets.
- **Simulation duration**: The total runtime is set to 10 seconds.

The **primary** performance **metric** analyzed in this study is the **number of lost packets** for each network configuration. The results indicate that as the number of nodes increases, the packet loss rate exhibits a non-linear trend.

For **smaller networks** with a limited number of nodes, packet loss remains relatively **low** due to reduced congestion and simpler routing paths.

However, as **network area size increases**, a **higher** number of lost packets is observed, primarily attributed to **packet loss** caused by **nodes being too far apart** from each other, preventing effective communication and leading to dropped packets. Additionally, for networks containing a larger number of nodes, packet loss can be caused by increased contention, collisions, and routing inefficiencies in denser topologies.

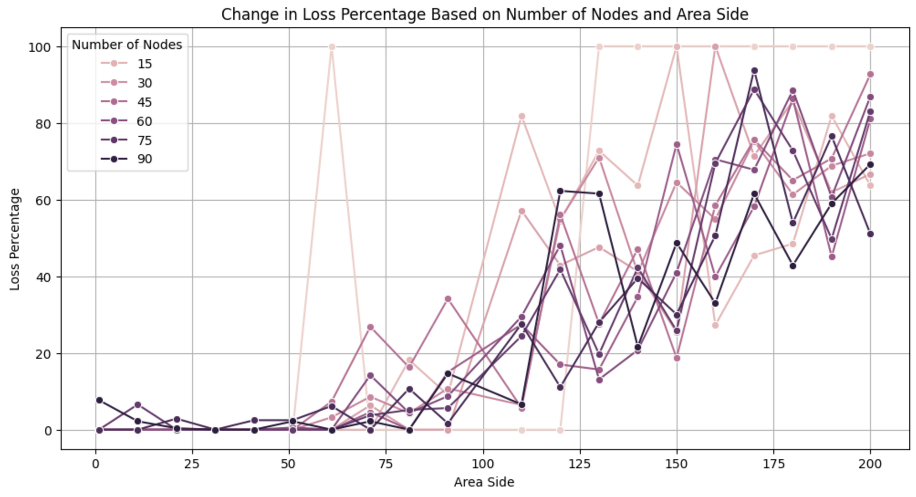

Figure 1: Change in Loss Percentage Based on Number of Nodes and Area Side

Furthermore, another simulation was conducted where the area size was fixed at 100 units while the number of nodes ranged from 2 to 200. This new simulation focused on analyzing the **Hop Count trend** with different **routing heuristics**, providing further insight into the efficiency of LRA.

The chosen heuristics were:

- **Ascending**: Routing tables memorize links based on an ascending order of IP addresses.
- **Descending**: Routing tables memorize links based on a descending order of IP addresses.

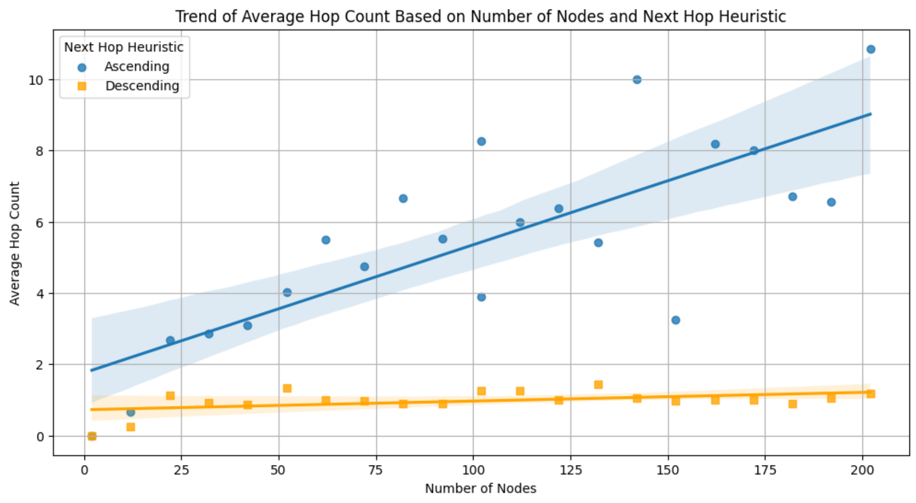

Figure 2: Trend of Average Hop Count Based on Number of Nodes and Next Hop Heuristic

The **Descending** heuristic demonstrates a clear advantage over the **Ascending** heuristic, which is probably caused by the **sink** node being assigned the **highest IP address**, making it more accessible when routing tables prioritize descending IP order. This structure leads to **more efficient** path selection and reduced hop count, ultimately improving network performance.

6 Conclusion

The simulation-based **analysis** of the **LRA routing protocol** provides valuable insights into its performance in varying network scenarios. The results suggest that while LRA is **effective in small to medium sized networks**, its **efficiency declines** as network **area increases**, primarily due to nodes reachability, congestion and mobility-induced route fluctuations.

Future work could explore **optimizations** to LRA, such as **adaptive routing** strategies and **congestion control** mechanisms, to enhance its scalability and reliability in large-scale ad hoc networks. Additionally, the focus on **disaster recovery** scenarios highlights the importance of **robust ad hoc communication** protocols in emergency response efforts, where traditional infrastructure is unavailable, and reliable peer-to-peer communication **is vital**.

References

[1] The NS-3 Consortium. *NS-3 Network Simulator*. `https://www.nsnam.org/`. Accessed: 2025-01-10. 2023.

[2] Eli Gafni and Dimitri Bertsekas. “Distributed algorithms for generating loop-free routes in networks with frequently changing topology”. In: *IEEE transactions on communications* 29.1 (1981), pp. 11–18.