Comparison of Different Search Algorithms in Unstructured Decentralized Peer-to-Peer Networks

Biweekly Update 3
CSC 466 - Overlay and peer-to-peer Networking - Spring 2025
Semester Project
Website

Holly Gummerson
University of Victoria
hgummerson@uvic.ca
V00986098

Mathew Terhune
University of Victoria
mterhune@uvic.ca
V00943466

Ali Gaineshev
University of Victoria
ggaineshev@gmail.com
V000979349

Overview

For our third biweekly update our group has been moving forward with the goals that we set out during the midterm update.

1. Automating test execution and output generation
2. Implementing dynamic node joining and leaving
3. Flood search implementation
4. Modifying query search mechanisms for improved efficiency
5. Consolidating metrics collection and analysis

From these goals, we have achieved most and are working towards completing the others with the knowledge we are gathering throughout the implementation process.

Topologies

We initially started with a simple linear chain structure, where each node was connected sequentially. While this was straightforward to implement, it lacked scalability and did not represent a complex network. To improve this, we developed a function that generates a binary tree topology (see **Figure 1**).

```
// make node connections, devices and chanells
for (uint32_t i = 0; i < numNodes; i++)
{
    uint32_t left = i * 2 + 1;
    uint32_t right = i * 2 + 2;
    std::vector<NodeContainer> curNode;

    if (left < numNodes)
    {
        // make nodes for index i
        NodeContainer nodePair;
        nodePair.Add(allNodes.Get(i));
        nodePair.Add(allNodes.Get(left));
        // make net device
        NetDeviceContainer device = p2p.Install(nodePair);

        // set IPv4 address
        std::ostringstream baseIP;
        baseIP << "10." << ipCounter << ".1.0";

        address.SetBase(baseIP.str().c_str(), "255.255.255.0");
        Ipv4InterfaceContainer interface = address.Assign(device);

        // assign node neihbour
        nodeNeighbors[i].push_back(interface.GetAddress(1));
        nodeNeighbors[left].push_back(interface.GetAddress(0));
        // push node pair
        curNode.push_back(nodePair);
    }
```

Figure 1. Code that generates the left side of the node.

This change introduced new challenges, particularly in routing. In ns-3, a node acts as a virtual computer by hosting network applications, protocols, and network interfaces. Each node connects to a **NetDevice**, which represents a network interface; these interfaces are linked to channels, with IP addresses assigned per interface, allowing a single node to have multiple IP addresses across different networks. Due to the new network structure, nodes started sending queries from a wrong network device/IP address. We are currently working on a fix.

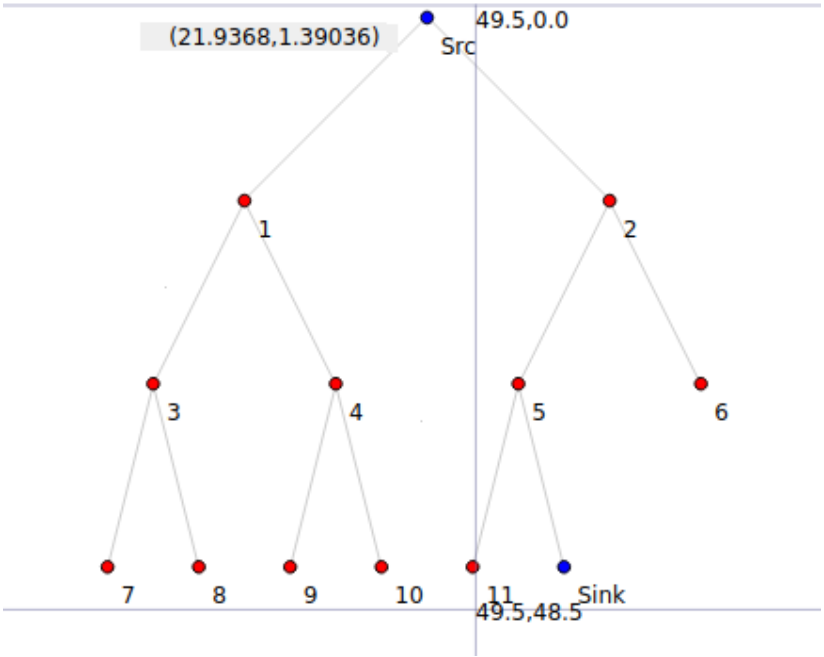

Figure 2. Binary Tree from NetAnim

We can currently generate a binary tree with any number of nodes. Additionally, we observed that extending our tree-based topology into a mesh structure would be relatively easy. All we have to do is to add additional links between some nodes to make loops. Looking ahead, we plan to refine our routing mechanisms to ensure correct packet forwarding, and potentially add a mesh topology.

Packet Structure Changes

```
P2PPacket::P2PPacket(MessageType type, uint32_t msgId, Ipv4Address sender, Ipv4Address dest,
uint8_t ttl, uint8_t hop, uint32_t sinknode, std::vector<Ipv4Address> pathHistory = {})
       : descriptorId(msgId), payloadDescriptor(type), senderIp(sender), destIp(dest),
m_ttl(ttl), hops(hop), sinkNode(sinknode), path(std::move(pathHistory)) {}
```

Modified Packet: After working with the packets more, we have modified the structure to include a sinknode field (number of the node that "holds" the data), and to include a path vector rather than a last hop. The path portion has allowed us to keep track of the path the packet takes before reaching a query hit. This is necessary for the backtracking it will take toward the sender. We opted for the sink node rather than actual data transfer due to time and scope constraints, as we wanted to focus on the search.

Issues: We ran into some issues relating to the new structure in the serialization portion, however it is sending well now

Next: Next steps for the packets is potentially adding a QUERY_RW message type, to keep our query procedures separate, or potentially adding fields for our measurements other than hop count.

Dynamic Node leaving / Joining

Overview

We have started with the basic implementation of our dynamic environment for our simulation. Initially, the topology is built using a loop in our main function to create a chain topology among five nodes. This worked for a static setup, to support a dynamic node addition and removal, we needed to introduce scheduled events that specify when nodes join or leave the network. In NS-3 as far as we are aware, it is not possible to fully "remove" a node since all have to be predefined before starting the simulation. So, our solution for simulating a node leaving the network is to disable all of a selected node's NetDevices, effectively preventing it from communication with other nodes. As well as removing the nodes reference from our nodeIps and nodeNeighbors vectors for consistency.

Issues

One of the issues which our group has identified is node identification and tracking connections between neighbors. In the static topology setup, everything is created in a procedural and predictable order, which makes it easy to manage and identify links. However, when a node decides to join or leave, accurate identification is critical. This topic is being explored and options are being proposed to remedy any issues we are coming across, these updates will be provided in the following updates.

Next

The next object for node leaving and joining is to finalize the functionality and test it under different scenarios. At the moment, all that has been tested is a singular node at the end of a chain leaving the network. So, after the successful completion of the dynamic environment, we aim to have testable environment setups to ensure consistency.

Node Leaving Example Code

```
// Stop the application
Ptr<P2PApplication> app = DynamicCast<P2PApplication>(node->GetApplication(0));
if (app) app -> SetStopTime(Seconds(Simulator::Now().GetSeconds() + 0.1));

// Remove this nodes IP from all the neighbour lists
Ptr<Ipv4> ipv4 = node->GetObject<Ipv4>();
Ipv4Address leaving_ip = ipv4->GetAddress(1, 0).GetLocal(); // Get the IP


// Disable ALL NetDevices on this node
for (uint32_t i = 0; i < node->GetNDevices(); ++i) {
    Ptr<NetDevice> netDevice = node->GetDevice(i);
    netDevice->SetReceiveCallback(MakeNullCallback<bool, Ptr<NetDevice>, Ptr<const Packet>, uint16_t, const Address &>());

    NS_LOG_INFO("Disabled NetDevice " << i << " on Node " << nodeIndex);
}

// Remove the IP from the nodeNeighbors vectors
for (size_t i = 0; i < nodeNeighbors.size(); ++i) {
    auto& neighbors = nodeNeighbors[i];
    neighbors.erase(std::remove(neighbors.begin(), neighbors.end(), leaving_ip), neighbors.end());
}
```

Figure 3. Basic node disconnection code - *subject to change*

Basic Node Creation Example Code

```
// Create a new node
Ptr<Node> newNode = CreateObject<Node>();
nodes.Add(newNode);                    // Add the new node to the container

internet.Install(newNode);

// Keeps a 1 - 1 mapping between nodeIps and nodeNeighbors
nodeIps.push_back(Ipv4Address("0.0.0.0")); // Placeholder IP address
nodeNeighbors.emplace_back();

NS_LOG_INFO("New node added at index " << nodes.GetN() - 1);
```

Figure 4. Basic node creation code - subject to change

Searching

Overview: We have now implemented query flooding functionality into our code. When a node sends a QUERY packet with a specified sink node, it will forward to its neighbours until it reaches the sink. Once reaching, we will receive the packet, change its type to QUERY_HIT and then proceed to forward it back to the original sender in the path it took. We have confirmed linear topologies, and are verifying that tree and multi connection topologies are successful.

```
// once there is a hit, process and forward backward through the path
   void P2PApplication::QueryHit(P2PPacket ppacket);
```

Issues: There were several issues we faced while implementing this, the most prevalent would be backwards forwarding. During the initial coding phases, it would get stuck in a loop and never reach the RecievePacket function. After looking into the ipv4l3 logs, it was apparent the routing table did not contain the backwards route which was fixed. Further the neighbours have caused some issues which we are addressing as we go

Next: After confirmation of different topologies, we are going to implement a modified random walk functionality to see the differences in each. If time permits, we will look into other ways to modify our search to gather new results

Future Work Summary

- Get metrics from both flooding and modified random walk on different topologies
- Implement dynamic peers and gather results
- Generating performance graphs based on test suite output
- Interpret results and consolidate to the final presentation / paper