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

MIDTERM UPDATE

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

Introduction

The primary goal of our midterm update was to establish a solid foundation for our project structure, define our approach to the problem, and identify the key functionalities that needed to be implemented. To validate our initial assumptions and ensure the correctness of our implementation, we developed a small, linear Gnutella-like network consisting of a few nodes. This prototype serves as a baseline, as all subsequent components rely on having a functional core implementation.

Additionally, due to the nature of our approach, certain properties of the Gnutella packet structure were deemed unnecessary for our use case. To streamline communication and improve efficiency, we have proposed and implemented a customized packet structure to the needs of our simulation. To support future development and ensure reproducibility, we are also working on accessory documentation, including a README that provides clear instructions on downloading, installing, and configuring NS3 for the environments we have used. This will include details on any modifications we have made to run and simulate our project on personal systems.

Finally, we plan to incorporate NS3's built-in visualization tool, NETAnim [1], into our workflow. This update will include an explanation of how the tool functions, how we plan to utilize it, and its importance for the next phases of our project, particularly in analyzing network behavior and validating our implementation.

Project Setup

This project is being written using ns3 software. We have created files:

- Network-sim - Main file to run the simulation and animations
- P2p-packet - Designed to define the packet header for gnutella like protocol
- P2p-application - Designed to handle logic and routing behaviour
- Dependencies and configurations required can be found on the README file on the Github

Using these files, we have defined a packet, and started to build functionality in the application. We are currently working to achieve Ping functionality and general packet sending and receiving. Afterward we plan to test and apply these ideas to deploy query functionality. The first goal is to achieve flooding, and then further modify it to fit our proposal. We have also set up our project to produce an .xml file which can be run using NetAnim, to visually show our work. Challenges for this project so far have started with building the project and structuring. It took a couple of tries to get the file structure right, and build correctly. We ended up having to modify the make files to accommodate the new classes. Keeping the packet structure and application separate has been helpful to continue to be organized through building it. Another consideration that has been challenging is deciding which of the core functions to add to the application file. Balancing simplicity and getting the correct requirements to be gnutella like has been a learning curve.

Network Information

The network in our project is currently a structured, decentralized P2P system, inspired by Gnutella. Our initial setup consists of a small, linear network with a few nodes to ensure proper implementation. This setup provides a controlled environment for testing packet transmission, and facilitates easy debugging.

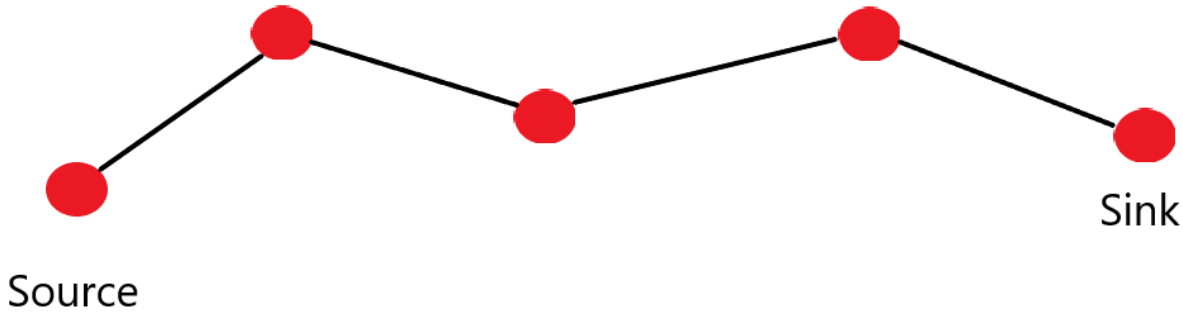

Fig 1. Current Network Topology

To evaluate the effectiveness of flooding, random walk and other techniques, we plan to implement a more complex topology. Instead of simple linear structure, we will use tree based or mesh-like topology. Additionally, if time permits, we will introduce dynamic node addition and removal.

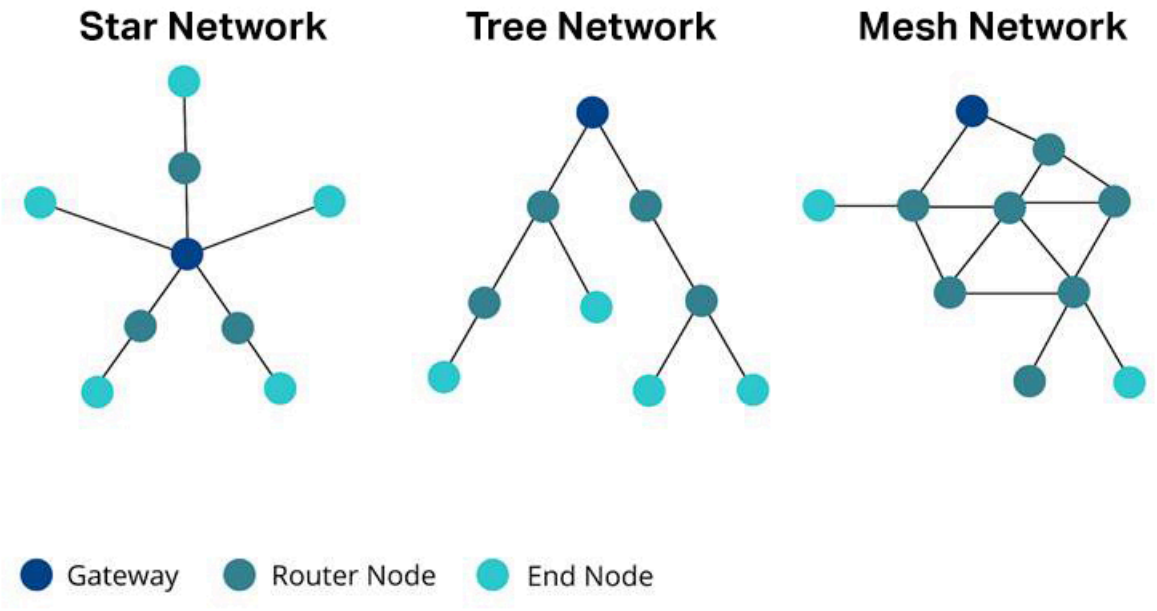

Fig 2. Possible future networks. Adapted from [4]

Packet Structure & Transmission

Based on previous Gnutella, we have defined a packet which contains the certain necessary fields. This is subject to change as we continue to work on the functionality but as a base we have defined certain things that are transferred. We are working towards a closer match to the ideas in [2]. A basic packet has the structure:

Fields	Descriptor ID	Payload Descriptor	TTL	Hops	Payload Length
Byte offset	0….15	16	17	18	19…22

- Where the Descriptor Id is a 16 byte string uniquely identifying the descriptor on the network.
- Payload descriptor 0x00 = Ping, 0x01 = Pong, 0x80 = Query, 0x81 = QueryHits, 0x40 = Push
- TTL - time to live, number of times it will be forwarded before being removed from the network
- Hops - Number of times this has been forwarded (unsigned)
- Payload Len

```
P2PPacket::P2PPacket(): descriptorId(0), payloadDescriptor(PING), senderIp(Ipv4Address("0.0.0.0")),
destIp(Ipv4Address("0.0.0.0")), m_ttl(0), hops(0), previousHop(Ipv4Address("0.0.0.0")) { }
   P2PPacket::P2PPacket(MessageType type, uint32_t msgId, Ipv4Address sender, Ipv4Address dest, uint8_t ttl, uint8_t
hop, Ipv4Address prevhop)
       : descriptorId(msgId), payloadDescriptor(type), senderIp(sender), destIp(dest), m_ttl(ttl), hops(hop),
```

```
previousHop(prevhop) {}
```

Our current constructor for a p2p packet considers these, modifying the descriptor id to be a simplified message string, adding a senderIp field and removing the payload len due to the focus on search not file transfer. Further, we have been testing with a previous hope field for forwarding purposes. This is subject to change as we continue to work and add functionality.

```
void P2PApplication::SendPacket(MessageType type, Ipv4Address dest, uint8_t ttl, Ipv4Address curaddy) {
   Ptr<Packet> packet = Create<Packet>();
   uint32_t msgid = GenerateMessageId();
   P2PPacket p2pPacket(type, msgid, curaddy, dest, ttl, 0, curaddy);
   packet->AddHeader(p2pPacket);

   // Send packet over the socket (how ns3 sends stuff)
   m_socket->SendTo(packet, 0, InetSocketAddress(dest, m_port));
   NS_LOG_INFO("Sent " << type << " packet to " << dest);
}
```

We have begun working to send and receive packets, to later be utilized for searching and finding peers. Our SendPacket() and ReceivePacket() are currently being modified for better forwarding and more adaptability for our future uses. We plan to further test them, add specific query functionality and make sure they work for different topologies and dynamic networks.

```
void P2PApplication::RecievePacket(Ptr<Socket> socket){
   Ptr<Packet> packet = socket->Recv();
   P2PPacket p2pPacket;
   packet->RemoveHeader(p2pPacket);
   if (p2pPacket.GetMessageType() == PING && p2pPacket.GetTtl() > 0) {
       p2pPacket.DecrementTtl();
       Ipv4Address curaddy = GetNode()->GetObject<Ipv4>()->GetAddress(1, 0).GetLocal();
       NS_LOG_INFO("current node after sending " << curaddy);

       // explicit packet drop if ttl is 0
       if (p2pPacket.GetTtl() == 0) {
           NS_LOG_INFO("TTL reached 0, dropping packet at " << curaddy);
           return;
       }
       // Forward the packet to the neighbors
       for (const auto& neighbor : m_neighbours) {
           if (neighbor != p2pPacket.GetSenderIp() && neighbor != p2pPacket.GetPrevHop()) {
               Ptr<Packet> newPacket = Create<Packet>();
               p2pPacket.SetPrevHop(curaddy);
               newPacket->AddHeader(p2pPacket);
               NS_LOG_INFO("send from " << curaddy << " to node at " << neighbor );
               m_socket->SendTo(newPacket, 0, InetSocketAddress(neighbor, m_port));
           }
       }

   }
}
```

Searching and Node Traversal

Searching within our decentralized P2P network currently follows a flood-based approach, inspired by Gnutella. The goal of search is to locate specific nodes within the network based on **Node-ID**. This process involves three types of nodes:

- **Source Node**: The node that initiates the request with a message and broadcasts it to its neighbors.
- **Intermediate Nodes**: Nodes that receive the request and forward it to the their neighbour and update certain parameters such as Time-To-Live(TTL)
- **Sink Node**: The target node. Once found, the sink node sends a message back along the discovered path.

We assume that a source and sink node always exists within the network, meaning that every query will eventually reach a valid destination with appropriate TTL. For the next iteration we plan the following improvements:

- Introduce **Random Walk** Searching: Instead of flooding, we will experiment with random walk searches, where each query is forwarded to a limited number of randomly selected neighbors, reducing network congestion.
- Implementing **Query Caching**: Nodes will maintain a cache of recent queries to avoid processing duplicate requests unnecessarily.
- **Dynamic Nodes**: Allows random nodes to join and leave the network.

Visualization & NS3 Animation Tool

To effectively analyze and demonstrate the behavior of our P2P network, we are utilizing NetAnim [1], the NS3 animation tool. This visualization allows us to observe packet movement, node interaction, and network dynamics in real time.

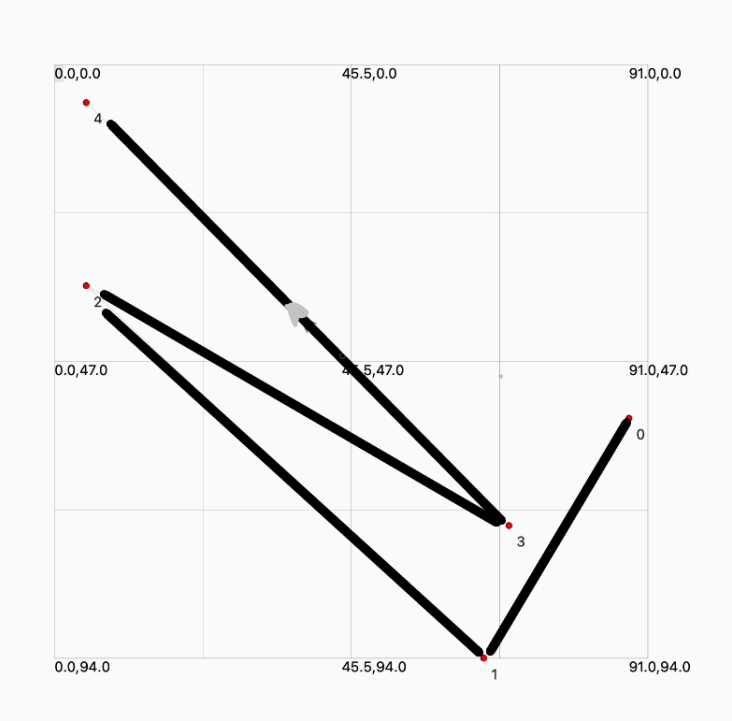

Fig 3. NetAnim visualization (Black lines were added to see links)

Early results from NetAnim show successful packet transmission across the network. Nodes correctly forward packets based on the topology, and reach the final node. However, the animations also helped to uncover an issue where a message was being duplicated. This was not apparent from raw logs, but was discovered through visualization. These observations have been crucial in bug fixing.

Test Suite and Metrics

Testing

For our test suite we aim to have a wrapper function that runs multiple iterations of our code with increasing input arguments, the thought behind this is to have the ability to make specific properties of our project scale and provide output into a csv file which can then be displayed graphically.

Metrics

Metric	Description
Query Success Rate	Measuring if a query has reached its goal as a percentage
Query Failure Rate	Measuring if a query has not reached its goal as a

	percentage (1 - Query Success Rate)
Total Hop Count	Measuring the total number of Hops performed by a single algorithm.
Average Hop Count	Measuring the mean amount of hops made throughout multiple test runs utilizing the same algorithm. (Number of nodes which the query passes through)
Total Query Latency	Total time taken for a query to be completed
Average Query Latency	The average time for a query to be completed under a standardized set of restrictions.

Future Work

As we progress beyond our midterm update, our primary focus for the third biweekly update is to refine our implementation and testing processes while expanding the functionality of our network simulation.

Third Biweekly Update Goals

Assuming that packet sending and tracing are functioning correctly, we aim to develop a comprehensive test suite. This suite will include various predefined network topologies with specific attributes and configurations to validate our assumptions and ensure the correctness of our implementation.

Currently, our network structure is static. By the third biweekly update, we plan to introduce dynamic node joining and leaving, allowing for a more realistic environment. Initially, we will accompany this with static tests, followed by more extensive testing in the final update.

With a deeper understanding of NS3, our network topology, and our implementation, we will proceed with our first search algorithm: network flooding. This will enable the discovery of a destination node from a given starting node through an exhaustive search.

Additionally, we plan to finalize our metrics tracking system and implement structured tracking. This will allow us to systematically evaluate multiple implementations across various network structures, providing meaningful insights into trends.

Final Update Goals

For the final update, we intend to implement an additional search algorithm, prioritizing feasibility based on our timeline and project constraints. Moreover, we aim to utilize test suite outputs to generate

performance graphs over time, providing a comprehensive analysis of efficiency, accuracy, and network behavior. The result of this will assist in concluding our research and semester-long project

Summary of Objectives

Biweekly Update 3

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

Final Update

- Exploring and implementing one additional search strategy.
- Generating performance graphs based on test suite outputs.
- Enhancing overall simulation performance and accuracy.

References

[1] https://www.nsnam.org/docs/models/html/animation.html
[2] https://rfc-gnutella.sourceforge.net/developer/stable/
[3] https://www.sfu.ca/~ljilja/ENSC427/Spring10/Projects/team7/final_report_Gnutella.pdf
[4] image