Establishing Connection between peers using Lidgren network

Amrita Yadava

Department of Computer Science Engineering
Yagyavalkya Institute of Technology , Jaipur, India
amrita.yadava@yahoo.com

***Abstract*— In this paper, we study the application of content-based resource selection and document retrieval to peer-to-peer networks. Lidgren network is used as a library in .net framework. It makes easier to send and receive messages between client and server. This network is used here to make the task easier. Peer to peer network is used to communicate between several peers.**

***Index Terms*— console, network, peer, retrieval**

I. INTRODUCTION

One of the most common networking which have evolved in recent years to become one of the most popular means of data retrieval is Peer to peer networking. This technology is used not only for client data sharing but now also for accessing large repositories of data, software updates and multimedia streaming. The enormous growth of the amount of data stored in the P2P overlays has made the existing methods of search in such overlays (which are usually limited to simple keyword search) insufficient in terms of time and result efficiency.

The world wide web dominates search in the search engines. The search engines use many machines arranged in data centres which they exclusively control. These machines helps in downloading many web pages as they can find, this process is referred to as crawling, and index these documents. An index contains the documents in which the term occurred. With this information, search engines can quickly suggest relevant pages for a given query. This approach of storing the index in large data centres and performing the searches there is termed centralised search.

This paper presents the progress of the research in the field of Information retrieval. It describes the proposed methodology and the preliminary results developed in research.

II. PEER TO PEER NETWORKING

In this paradigm each node is equal and therefore called a peer. Each peer could be said to be a client and a server at the same time and thus can both supply and consume resources. In this paradigm, peers need to cooperate with each other, balancing their mutual resources in order to complete application-specific tasks. For communication with each other, during task execution, the peers temporarily form overlay networks: smaller networks within the much larger network they are part of.

Each peer is connected to a limited number of other peers: its neighbours. Peers conventionally transmit data by forwarding from one peer to the next or by directly contacting other, non-neighbouring, peers using routing tables. Peers usually transmit data by forwarding from one peer to the other or by directly contacting other, non-neighbouring, using routing tables.

The architecture is determined by the shape of overlay network that is the placement of indices, local or global, and the methods used for communication. The choice of architecture influences how the network can be utilised for various tasks such as searching and downloading. The participating machines in peer-to-peer networks are mostly found at the edge of the network which are personal computers in homes.

This is the reason why a peer-to-peer network typically consists of many cheap rated machines and all with diverse processing and storage space as well as different link speeds. This kind of network provides many valuable applications like streaming media, file sharing, and distributed search. Peer-to-peer networks have numerous properties that make them useful for these tasks. They have no centralised index or control point and therefore also no central point of failure. Because of this, they become self-organising that is they can automatically adjust when peers join, depart from it or fail. The language used for communication between peers is common among them and is symmetric as is the term of services. This symmetry is the reason of making a peer-to-peer network as a self-scaling in which each peer that joins the network gets added to the available total capacity.

Construction of the NetPeerConfiguration object

```
Config = new NetPeerConfiguration("test_console");
Config.EnableMessageType(NetIncomingMessageType.DiscoveryRequest);
Config.EnableMessageType(NetIncomingMessageType.DiscoveryResponse);
Config.EnableM essageType(NetIncomingMessageType.ConnectionApproval);
Config.EnableMessageType(NetIncomingMessageType.UnconnectedData);
Config.LocalAddress = NetUtility.Resolve("localhost");
The initialization of the NetPeer object (extremely simple) and starting up the
listener
thread:
Peer = new NetPeer(Config);
```

```
Peer.Start();
Console.WriteLine("listening on " + Config.Port.ToString());
NetWorker = new MyNetWorker(Peer, this);
NetThread = new Thread(NetWorker.ProcessNet);
NetThread.Start();

Sending a basic string message (the MessageType enum is my own, not part of the
framework):
public void SendMessage(string message) {
if (Peer.Connections == null || Peer.Connections.Count == 0)
 {
Console.WriteLine("No connections to send to.");
return;
}
NetOutgoingMessage msg = Peer.CreateMessage();
msg.Write((int)MessageType.StringMessage);
msg.Write(message);
Peer.SendMessage(msg, Peer.Connections, NetDeliveryMethod.ReliableOrde
}

Sending peer info to all connected peers (by passing Peer.Connections to
Peer.SendMessage):
public void SendPeerInfo(IPAddress ip, int port) {
Console.WriteLine(string.Format("Broadcasting {0}:{1} to all (count: {
port.ToString(), Peer.ConnectionsCount));
NetOutgoingMessage msg = Peer.CreateMessage();
msg.Write((int)MessageType.PeerInformation);
byte[] addressBytes = ip.GetAddressBytes();
msg.Write(addressBytes.Length);
msg.Write(addressBytes);
msg.Write(port);
Peer.SendMessage(msg, Peer.Connections, NetDeliveryMethod.ReliableOrde
}
```

III. METHODOLOGY

In this networking, lidgren library is used.
Lidgren network library is all about messages. There are two types of messages:

Library messages telling you things like a peer has connected or diagnostics messages (warnings, errors) when unexpected things happen.

Data messages which is data sent from a remote (connected or unconnected) peer.

The base class for establishing connections, receiving and sending message is the NetPeer class. You can use it to make a peer-to-peer network, but if you are creating a server/client topology there are special classes called NetServer and NetClient. They inherit NetPeer but set some defaults and include some helper methods/properties.

Here's how to set up a NetServer:

```
NetPeerConfiguration config = new NetPeerConfiguration("MyExampleName");
config.Port = 14242;

NetServer server = new NetServer(config);
server.Start();
```

The code above first creates a configuration. It has lots of properties you can change, but the default values should be pretty good for most applications. The string you provide in the constructor (MyExampleName) is an identifier to distinguish it from other applications using the lidgren library. Just make sure you use the same string in both server and client - or you will be unable to communicate between them.

Secondly we've set the local port the server should listen to. This is the port number we tell the client(s) to connect to. The local port can be set for a client too, but it's not needed and not recommended.

Thirdly we create our server object and fourth we Start() it. Starting the server will create a new network thread and bind to a socket and start listening for connections.

IV. RESULTS

Early on we spoke about messages; now is the time to start receiving and sending some. Here's a code snippet for receiving messages:

```
NetIncomingMessage msg;
while ((msg = server.ReadMessage()) != null)
{
    switch (msg.MessageType)
    {
case NetIncomingMessageType.VerboseDebugMessage:
case NetIncomingMessageType.DebugMessage:
case NetIncomingMessageType.WarningMessage:
case NetIncomingMessageType.ErrorMessage:
            Console.WriteLine(msg.ReadString());
            break;
        default:
            Console.WriteLine("Unhandled type: " + msg.MessageType);
            break;
    }
    server.Recycle(msg);
}
```

Here first we declare a NetIncomingMessage, which is the type of incoming message. Then we read a message and handle it, looping back as long as there are messages to fetch. For each message we find, we switch on something called MessageType - it's a description what the message contains. In this code example we only catch those messages which are of type VerboseDebugMessage, DebugMessage,WarningMessage and ErrorMessage. All those four types are emitted by the library to inform about various events. They all contains a single string, so we use the method ReadString() to extract a copy of that string and print it in the console.

Reading data will increment the internal message pointer so you can read subsequent data using the Read*() methods.
For all other message types we just print that they're currently unhandled.
Finally, we recycle the message after we're done with it - this will enable the library to reuse the object and create less garbage.
Sending messages are even easier:

```
NetOutgoingMessage sendMsg = server.CreateMessage();
sendMsg.Write("Hello");
sendMsg.Write(42);
 server.SendMessage(sendMsg, recipient, NetDeliveryMethod.ReliableOrdered);
```

The above code first creates a new message, or uses a recycled message, which is why it's not possible to just create a message using new(). It then writes a string ("Hello") and an integer (System.Int32, 4 bytes in size) to the message.

Then the message is sent using the SendMessage() method. The first argument is the message to send, the second argument is the recipient connection - which we won't go into detail just yet - and the third argument are HOW to deliver the message, or rather how to behave if network conditions are bad and a packet gets lo'st, duplicated or reordered.

V. CONCLUSION

Peer-to-peer technology is a robust solution to the ethical and technical problems and deserves more attention. an overview of what has been done so far and what choices can be made when implementing a system in practice. Both of these aspects are important for future research and development of real-world systems.

Peer-to-peer information retrieval overlaps with federated information retrieval, and shares its central challenges: resource description, collection selection and result merging. However, the key difference is that in federated information retrieval all queries and search results are channelled through a centralised mediator party. This enforces a strict division between the consumers and providers of information. In peer-to-peer information retrieval the predominant mode of interaction is with other peers, not with a centralised mediator. A number of key challenges for peer-to-peer networks with respect to usage guarantees, behaviour of peers and evaluation are identified. Challenges unique and important to peer-to-peer information retrieval are minimising latency, maintaining index freshness and developing test collections.

This paper gives a clear definition of peer-to-peer information retrieval and what distinguishes it from related and overlapping fields, like file sharing and federated information retrieval. These concrete contributions may aid in the design and construction of a large-scale peer-to-peer web search engine, which is the primary goal.

ACKNOWLEDGMENT

I thank all staff members of my college and friends for extending their cooperation during my research work.

REFERENCES

[1] McGraw-Hill. 2002. McGraw-Hill Dictionary of Scientific and Technical Terms 6th Ed. McGraw-Hill Professional. ISBN 9780070423138.
[2] Zeinalipour-Yazti, D., Kalogeraki, V., and Gunopulos, D. 2004. Information Retrieval Techniques for Peer-to-Peer Networks. Computing in Science & Engineering 6,
[3] Kulathuramaiyer, N. and Balke, W.-T. 2006. Restricting the View and Connecting the Dots - Dangers of a Web Search Engine Monopoly. Journal of Universal Computer Science.
[4] White, A. 2009. Search Engines: Left Side Quality versus Right Side Profits. Working paper, Toulouse School of Economics. Dec. http:// dx.doi.org/10.2139/ssrn.1694869 (Retrieved June 25th 2012).
[5] Tene, O. 2008. What Google Knows: Privacy and Internet Search Engines. Utah Law Review 2008, 4, 1434–1490.
[6] Lewandowski, D., Wahlig, H., and Meyer-Bautor, G. 2006. The Freshness of Web Search Engine Databases. Journal of Information Science 32, 2,131–148.
[7] Bergman, M. K. 2001. The Deep Web: Surfacing Hidden Value. Journal of Electronic Publishing 7.