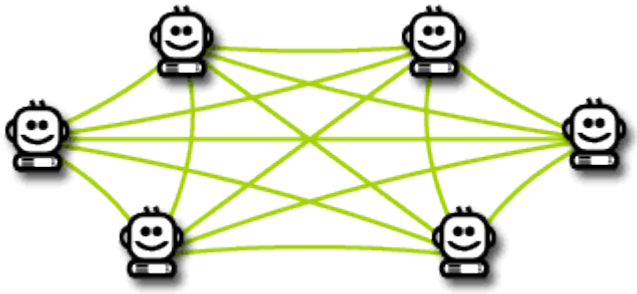
4DN4 Advanced Internet Communication

Lab 3: The Mactella Peer-to-Peer System (v1)

Shown in Class, Mon, Feb. 9, 2015
Post on Web: Tues., Feb 10, 2013
Lab #3 Demos: March 11, 12 2015
March 25, 26 2015
(more dates to be announced)



Introduction

Lab #3 should be a lot of fun if you enjoy socket programming. To be successful at Lab. #3, you need to successfully demonstrate Lab #2 with all commands working successfully. Lab #3 is worth about 5% of your final mark .

In Lab #2, you implemented the McNapster FTP-based application (FTP = File Transfer Protocol) where there was a clear distinction between a client and server, and where all content to be shared resided on a single centralized server. In July 2001, the US Justice Department managed to shut down Napster by getting a court order to shut down the centralized resources (i.e., servers).

The closure of Napster lead to the development of a new type of system called Peer-to-Peer (P2P) systems. Gnutella was the first large P2P system in the world, and Lab #3 develops a simple 4DN4 version of Gnutella. Gnutella was developed in about the year 2000, and the authors apparently ate a lot of Nutella while writing the program, and hence its name. In a P2P system, there are no centralized servers. Each peer can act as a client and a server to exchange data, and there is no centralized server that acts as a single critical resource which can fail (or be shut down). The first P2P system was Gnutella, described in http://en.wikipedia.org/wiki/Gnutella. P2P systems are extremely reliable because they have no centralized servers to fail or shut down. NBC had previously adopted a P2P system to distribute its digital TV content in 2009, showing how quickly this concept has been accepted into the mainstream.

In a P2P system, peers are connected to one another over the Internet using either the TCP or HTTP protocol. The HTTP protocol is mostly used to connect to users behind a firewall. For the purposes of this lab we will use only the TCP protocol.

The Mactella version you will implement in Lab #3 will support the following features: a **distributed search engine**, **distributed file sharing**, and a decentralized network topology. In the next sections we describe the lab requirements.

The Mactella software builds on the FTP application in Lab #2. You will need to have your FTP application in Lab #2 operational to do this lab.

Mactella Shared Content

For simplicity, users will specify a directory (e.g., shared_folder), where all shared content will be stored, when the Mactella peer starts up. Content will be stored and shared by file name.

Mactella Search Engine

Given that there is no central server to store the names and locations of all the available files, how does the Mactella software on your peer find a file on someone else's peer? Here is the process:

- You type in the name of file you want to find to your peer.
- Your peer knows of at least one other Mactella peer somewhere on the network. It knows this because you've given it the IP address of the known peer at start up time. Your peer sends the file name you are searching for to the Mactella peer(s) it knows about. These peer(s) search to see if the requested file is on their local hard disk(s). If so, they send back a reply (and machine IP address) directly back to the requesting peer.
- At the same time, if a peer does not find the file locally, it sends out the same request to the Mactella peers it knows about, and the process repeats.
- A request message has a **TTL** (time to live) limit placed on it. A request might go out over six or seven hops before it stops propagating. If each peer on the Mactella network knows of just four other peers, then your request might reach 4^7 = 16,384 other peers on the Mactella network if it propagates seven hops.

Specifically, Mactella's search engine works by creating a search request with a *TTL* or *MAX-HOPS* value that indicates the maximum number of hops the search request will be propagated in the Mactella network, before the search results are returned to the peer that originated it; For example, if a request has *MAX-HOPS* = 3, the request will be allowed to propagate a maximum of three hops from its origin. As each new peer receives the search request, it decrements the *MAX-HOPS* value before forwarding the request. When the *MAX-HOPS* value reaches zero, the search request is no longer forwarded to other peers.

Each search request carries the (IP address, port number) of peer that initiated the search. Peers reply with their (IP address, port number) directly to the initial requester, if they have the requested media file. (They don't send the media file, they just reply that they have it).

After the originating peer receives the search results, it can request the file directly from any of the peers that have the file.

For simplicity, we'll do this process manually; i.e., the search results are printed on the screen and the user would type "GET <filename>, <my-ip-address> <my-port-number>" to fetch the file from the remote peer with the file.

Mactella Host Discovery

Each peer maintains a list of other *known* peers (peers who are currently connected). It records (IP-address, port-number) for each known peer.

In a practical system, each peer could have a separate thread that pings other peers periodically to update its list of known peers. For simplicity, we will perform this 'ping' task manually using the DISCOVER commend.

To bootstrap your system when you create each Mactella peer, each peer should initially know the (IP-address, port-number) of at least one remote peer. This means that when you create your first peer you would 'cheat' by passing information for a second peer that has not been created yet, but will be created soon.

A peer uses the "DISCOVER, <time to live>" command to find out about more peers. This "DISCOVER,<TTL>" command causes the originating peer to print a list of peers on its screen, reachable within TTL hops. It then adds these peers to its list of known peers. When a peer receives the DISCOVER request, it replies to it the originator, decrements the TTL and passes the request to each peer in its own known list of peers. This process continues until the value of TTL equals zero.

Mactella Command Specification

You should implement the following commands in your application.

Command	Description
LISTL	Lists the shared files in the local peer's current directory.
LISTR <ip_address> <port_number>	Lists the shared files in remote peer specified by its (IP address, port number)
SEARCH <fileName > <TTL>	Sends a search request to other peers with the specified TTL (integer value). The search results are printed on screen

	with (<ip address>, <port number>) of each peer that has the file.
DISCOVER <TTL>	Updates the list of known active peers and displays them up to a certain depth specified by <TTL>
RESET	Resets the list of known active peers to the one original known peer.
GET,<filename>,<ip_address>,<port_number>	Retrieves the file from the peer specified by the ip address and port number
QUIT	Exits from Mactella peer; shut down processes gracefully

You may assume that each peer uses one socket (listening socket) that can handle queries like search requests, discover, etc.

Note that all queries (e.g., search requests) contain the (IP-address, port-number) of the peer that initiates the request.

If you prefer, you could use more than one control socket to handle different aspects of the implementation, but you should clearly document your design in your lab report.

Lab Demonstration

You need to demonstrate your working code in a suprvised lab session. See below for what to demonstrate.

Lab. Submission

You need to submit *a* softcopy of your report on Avenue-to-Learn. Your professionally documented code should be included.

The report should include a title page with your names, a description of your design, detailed flow charts, some screen snap-shots showing the correct operation, a summary of each problem that you encountered and its solutions, and a brief suggestion of ways to improve the lab in the future. The flow-charts will be graded based on how much useful information they convey to the prof and TAs, and how easily the flow-charts enable the prof. to refer to your C, Java or Python code and understand it.

Electronic submissions: On Avenue-to-Learn

Variations

You may submit proposals for a different lab to the professor. If the proposal is interesting and approved, we will identify what we would want to see in the writeup.

Lab Demonstration Details

All the commands given should be demonstrated, both for a positive outcome and a negative outcome. Here is the table of commands:

Mactella Commands

Command	Description
LISTL	Lists the shared files in the local peer's current directory.
LISTR <ip_address> <port_number>	Lists the shared files in remote peer specified by ip address and port number
SEARCH <fileName > <TTL>	Sends a request to other peers with the specified TTL (integer value). The search results are printed on screen with <ip address> and <port number> of each peer that has the file.
DISCOVER <TTL>	Displays peers up to a certain depth specified by <TTL>, and adds these to the list of known peers
RESET	Reset the list of known peers to the one original known peer
GET <filename> <ip_address> <port_number>	Retrieves the file from the peer specified by the ip address and port number
QUIT	Exits from the host Mactella peer

TEST SETUP: Set up your own 'Master' peer A, and several other peers such as B, C and D. Your Master peer A knows about peer B (and its IP address and port number). Peer B knows about Peer C, and peer C knows about peer D. Peer D knows about peer A.

Each peer (A,B,C or D) has one media file, which is a real randomly selected PDF, music (mp3) or Video file. You should be able to GET any remote file into the master peer A's shared folder in the demo. The same applied for peers B, C and D.

(1) The LISTL command should show the shared files found on the host's local shared directory. Demonstrate this on the master peer A, and other peers (B,C,D). It should also work if there are no shared files in a peer.

(2) The LISTR <remote_ip_address><port number> command should work from any peer. It should show on the originating peer's screen the shared file(s) found on that remote peer.

(3) The GET <file> <remote_IP_Address> command should work. You should be able to retrieve the media file from any peer, and open it on the master peer A. (The same applies for peers B, C and D).

(4) The SEARCH <filename> <TT> command should work from any peer.

(5) For example, the SEARCH <B_file>, <1> command on the master peer A will return the <IP address, port#> for peer B, since peer B is 1 hop away.

(6) The SEARCH <C_file>, <1> command from the master peer A will return a null list, since peer C which has C_file is 2 hops away.

(7) The SEARCH <C_file>, <2> command from the master peer A will return the <IP address, port#> for peer C.

(8) The SEARCH <D_file>,<2> from the master peer A will return a null list.

(9) The SEARCH <D_file>, <3> from the master peer A will return the <IP address, port#> for peer D, which has D_file and is 3 hops away.

(10) The GET <C_file>, <C_IP_Address> command should work. You should be able to retrieve file C_file from peer C, and open the media file on the master peer A.

(11) The DISCOVER <1> command executed on the master peer A should return the list of 1 known peer, peer B. Your master peer A initially knows about 1 peer, peer B, so its list of known peers does not change.

(12) The DISCOVER <2> command executed on the master peer A should return the list of peers B and C. Its list of known peers is now {B,C}.

(13) The DISCOVER <3> executed on the master peer A should return the list of peers B, C and D. The list of known peers for peer A is now {B,C,D).

(14) The RESET executed on the master peer A should reset the list of known peers to peer B.

• You should demonstrate the ability to transfer smaller PDF files, larger mp3 files, and large video files.

You also need to submit a written lab report, and flow-charts.

VERY HIGH LEVEL P2P FLOW_CHARTS

(All coding details supressed)

Lab 2 Client

```
- open socket( IP-address, port-number)
loop
        - get User Command MYREAD()
        - call appropriate function
        and perform task
        - tasks = GET, PUT, LIST, QUIT
end loop
- close connection
```

Lab 2 Server

```
- open listenning socket (ip-address, port)
-loop
        call blocking ACCEPT()
        create 2 threads
                (server & client)
        - client thread calls HandleTCPClient
        - server thread continues in loop
end loop
```

Lab 3 Peer v1 (functional peer, without DISCOVER, SEARCH commands)

```
- fork to create 2 processes (client-side, server-side)
or use threads to create 2 threads'
- client-side process/thread runs the client code from Lab 2 as shown:

client-side process or thread: {

loop    - get User Command (with ip-address & port-num)
        - call appropriate function and perform task
        - the user command tells us the ip-address & port #)
        - tasks = GET, PUT, LIST, QUIT (see the lab specification)
        - close  connection
end loop



-server-side process runs the server code from lab 2 as shown:

server-side process or thread {
- open listenning socket
-loop
        call blocking ACCEPT
        create 2 threads (server & client)
        - client thread calls HandleTCPClient
        - server thread continues in loop, answering peer connection
         requests
end loop
```

the above peer is predominantly lab 2 code, with some minor rearrangement
- this will get you a functional peer, that can transfer files with other peers
provided that the user knows the ip-address and port number of the remote peer
- now just add the DISCOVER & SEARCH commands (relatively easy, see next page)

VERY HIGH LEVEL P2P FLOW_CHARTS

(All coding details supressed)

Lab 3 Peer v2 (with DISCOVER & SEARCH Commands)

```
- fork to create 2 processes (client-side, server-side)
or use threads to create 2 threads

- client-side process/thread runs client code from Lab 2:

client-side process or thread {
loop
        - get User Command (with ip-address abd port-num)
        - call appropriate function and perform task
        - the user command tells us the ip-address & port #)
        - tasks = GET, PUT, LIST, QUIT, DISCOVER, SEARCH (see lan specification)
        - add the 2 new functions for DISCOVER  & SEARCH commands
        - these new commands will send a DISCOVER or SEARCH message to a peer
        - close connection
end loop
}

-server-side process or thread {
- open listenning socket
-loop
        call blocking ACCEPT
        create 2 threads  (server & client)
        - client thread calls HandleTCPClient
        - change HandleTCPClient to recognize 2 new messages, DISCOVER, SEARCH
        - this peer may send its ip-address & port-num to originator of DISCOVER
        - this peer will decrement TTL and if new TTL >= 1, forward DISCOVER or SEARCH
          message to every peer it knows about
        - main server thread continues in loop
end loop
}
```

the above is predominantly lab 2 code
the modifications to add the DISCOVER & SEARCH commands are relatively minor