Peer to Peer File Distribution On Local Area Networks

By Pornphawit Manasat and Joakim Pedersen Nilfjord

Group Members: Pornphawit Manasat, ID: , Joakim Pedersen Nilfjord, ID: 5480574

Course: CS223: Data Communication & Networks

Table Of Contents

Introduction

We are suppose to help a lazy admin distribute a file from one computer to other computers in the local network without having him do any heavy lifting. So here is the basic idea. Run the program on every server, admin choose the computer he want to be the master, the master distribute the file (actually other computers come and get the file), once completed, programs reset the state of itself and wait for command to distribute again by the admin.

Techniques And Methods

<u>Overview:</u>

The project contains 11 classes (12 but i dont count clientChecker)

There is also the Command project but I don't really count that.

1. *Service*
    a. registers devices to a specific service and initializes ServiceDiscovery
2. ServiceDiscovery:
    a. Create listeners on every interface
    b. Listens to every computer registered to a specific service
3. Hub
    a. Details in transmission
4. ExtendedClient
    a. Pretty much a thread executed by Hub
    b. Used to start its Client instance when run → share() and waitforcompletion
    c. Also has an observer that is added into the Client instance through addObserver
5. ProgressObserver
    a. extension of Observer
    b. its update will update its progress of the client instance it is attached to and print out the progress
6. TorrentClient
    a. Used by downloader to download the ".torrent" file from the master
7. TorrentServer: → deprecated, no longer used and merged it into Main
    a. Originally planned to be its own server on another thread
8. Notifier
    a. Used by the uploader after the run of the hub to notify other computers that this computer is the master and they can get the file from it.
9. ResetNotifier
    a. Called by the spark server once all the computers are done downloading the file, will send the resetState post request to every single computer participating including itself.
10. StatusReporter
    a. Has the hub to access itself to the progress of the client.
    b. Send post request to the master to report its status

11. Main
    a. Spark server, has several Route
    b. Create service first and start it
    c. Create logger → Not used in demo
    d. Assign the port
    e. Wait for the legit post or get request

Details:

Service Discovery

Our goal for service discovery: o create a program that would allow any computer with Windows or Mac OS X on the same LAN register to a specific service of our choosing. And enable that computer to listen other computers that have registered to the same specific service it registered too. Furthermore when x computers has our program then all of the x computers will have registered and they are all actively listening. And every computer will have created a HashSet of IP's of size x (it includes it's own ip in the hashset)

What we did:

For Service Discovery we decided to use jmDNS in combination with bonjour, we created two classes for this purpose, Service and ServiceDiscovery.The Service class main task is to register a specific service.The Service get's initialized in Main class. And up on initialiazation, it will in the constructor look at all interfaces, and their ip and create an jmDNS instance of that ip if it is an IPv4. This jmDNS instance is added into a HashSet. In addition to that it starts up a serviceDiscovery instance in the constructor. The serviceDiscovery also loops through every interface and for every ip , it it is IPv4 it will create a jmdDNS instance of that ip and start a Servicelistener. The ServiceListener, are listening to a specific service, in our code it is listening to _http._tcp_.local. Furthermore the serviceListener will add every registered service, once we find a computer in the listener we get some information about it, but not the ip. So we have to use it's hostname and query the computer for its ip and add it into a iplis (HashSet). After initialization, a start method is called in Main which starts up a thread and also calls a registering method which register every jmDNS instance in the HashSet created in the Service initialization. There is also a method called getIP's in the Service class which can be used to retrieve the hashSet of Ips

A summary of how it works:

- Service class is initialized -> register services
- ServiceDiscovery -> Listens to the registered services
- It works over multiple interfaces

There is no notion of a master computer in service discovery, since every computer does the same work, they all register and they all listen. Every computer create their own list of IP's which is used for transmission. Hence any computer can be the master and any computer can be the client.

Transmission

Our goal for transmission: To transmit all the file to all other computers in the network from the master without having to interact with any of the computer. Easy, we use the Ttorent library, the bittorrent protocol implementation, in java available to us in order to achieve this. Transmission can be divided into 2 routes: uploader route, downloader route.

Uploader route:

This guy is the initial seeder, the guy with the actual file, the master.

Once he received the command to be the master and upload the file, these are the steps he will take:

Step 1: Setting the Hub and start it

(hub is meant to be a thread but due to some complication, it now works serially even though it's in another thread (sync through CountDownLatch))

1) create the URI to be announced for the tracker server
    a) his own URI usually
2) generate the tracker server → we start it after we craete it
    a) needed the host IP
3) generate the torrent file → announce it after created by the tracker server
    a) needed the actual file itself → it will create the ".torrent" file
    b) URIS which would announce it
4) generate the file client
    a) need the its ip, and a SharedTorrent item which can be easily made with SharedTorrent.fromFile(".torrent" file, desired directory)
5) Start the client afterward

Step 2: setUpTorrentFileServer information for the Spark server

1) Pretty much we just call the function to set up the items

Step 3: Notify the other computers (downloaders) to come and get the torrent file from us

1) Also giving the file name to it.
2) This is also where discovery parts come in since we need the IPs of people in the network

Step 4: Start the reset notifier

1) this is for monitoring and resetting the states of all the clients in all computers in the network after the clients are done

Downloader route:

Step 1: Wait for uploader to tell you to start

Step 2: Set up all the necessary conditions to download

Step 3: Download the “.torrent” file from the server

Step 4: Start the hub and wait for it to finish → as usual the hub was meant to do it in parallel but i kinda lazy to fix the latch out (i was trying to fix the server issue)

Step 5: Start the status reporter → monitoring

Monitoring

Goal: master has to be able to track all the clients’ progress and reset the state of every single clients and itself once the file has been distributed.
This is done by having the clients send its progress to the master for every 3 seconds. The master will then print out the status of the client including its ip address.

Route reportClientStatus() is meant to keep track of the progress and then handle the necessary operations to inform the ResetNotifier thread of the current status of each client. If all the clients reported 100.0, then the ResetNotifier will send the httpPost request to each client including itself to reset the state of the program and wait for another file to be distributed. The httppost request is meant to be handled by Route resetSystem().

How was the tracking of progress actually done?
The Client provided by the Ttorrent library itself contains the function getCompletion() which would return the completed percentage of the download of that instance.
The library also allows us to add an observer to the instance of the Client to observe the state of the client and such. As a result, we are able to easily tracked the progress of the instance of the Client. This is why the library is so damn good even though it lacks a proper documentation to clarify how to use many of its functionalities.

Results

Service Discovery was completed with a satisfactory level of result (thanks Jo). However, it is a bit troubling that changing the name of the service from the one used now may cause the client to not be able to find one another. The code for this is a tad bit ugly and could have been optimized (i'm too lazy and it's exam time anyways) but the same could be said for other parts of the project too.

Meanwhile, the transmission was pretty easy but annoying due to the lack of clarification by the library. In addition, the hub itself could have been made much more optimized by having in run in the main thread instead of branching out. The transmission speed was decent but I believed that I could have used an optimized protocol instead of bittorent protocol to make the transmission speed a ton faster than it is right now. Maybe I will do that after the exam.

On the other hand, the monitoring itself was a totally different story from the other two parts. The basic monitoring, where each computer prints out how much they have downloaded, is completed without any problem through the use of observers. However, the master side is still unable to monitor every single clients on the network.
The question is why?
I have set up the Route reportClientStatus() on the spark server for master so that the thread on the other computers can send the report to it. However, after reporting twice (from that client), the server stop responding to the report request, it did not even go into the route. This situation was the most peculiar and frustrating since I have no idea how it happened. One way to find out this is that I have added logger to check whether the request have actually reach the master or not. However, I did not have time to test it out yet to see why it actually failed.

Overall, the project would be what I considered a success since I have satisfied all the criteria during the demo with a bit of questionable stuff from the monitoring side. It was a rather successful project with the exception of ugly code and inefficient classes due to a lot of changes from the original design.

Credit:
I would also like to thank other groups, especially PeterPan group for helping Jo and I out for almost the entirety of the project, especially the transmission part for me since Pan helped me understand the library and learnt how to use it.

Operation Manual

- Make sure bonjour is installed on every computer
- Run Main.jar on every computer (java -jar Main.jar -i <insert computers ip>)
    - pull the folder from https://github.com/nobody0014/DatacomProject3
- Run Command.jar on Master (java -jar Command.jar -f <insert filename here>
    - pull the folder from https://github.com/nobody0014/projectCommand3