CSE 123B - Spring 2006
Project 3

Calvin Hubble
Amin Vahdat

Due: Friday 03/15/2007 at 5pm

1 Introduction

For this project you will be implementing a distributed file-sharing protocol loosely resembling the fast-track protocol used in applications such as Kazaa. The protocol is a flood-based protocol that leverages the idea of supernodes. Basically, the network consists of a number of supernodes, each of which is responsible for up to N clients including an active index of offered files for that client. Any request by a client is forwarded to its corresponding supernode, who will flood to all the other supernodes it knows about. Upon receiving the request, each supernode will check its active index and forward a request to all valid clients. Supernodes themselves are merely nodes in the work and can search / download like any other node, however they have the added responsibility of maintaining children and forwarding search requests to and from these children. Also, file transferring will be handled using HTTP get requests. Therefore, every node in the network will also be running an HTTP server listening on a port that will be made available to all nodes in the network.

2 Supernodes

Supernodes are responsible for managing up to a specified number of clients. In order to ease implementation, we will be providing some simplifications regarding supernodes and their selection. First, super nodes will be more or less selected based on their advertised bandwidth. In otherwords, when joining a network, a node will advertise some bandwidth, and this number will be used when deciding when a node should be upgraded to supernode status. For the scope of this project, assume that nodes all "play nice" that their advertised bandwidth is a rough approximation of their actual bandwidth. Also, every Supernode can have at *most* N nodes connected to it - where N will be a constant for the entire network. A supernode is considered "full" when it has N clients concurrently connected to it.

For this protocol supernode connections are uni-directional. Throughout this spec we use the term "connected to" and "knows about" interchangeably. If supernode A is connected to B, this does not imply that B is connected to A. This simply means that node A knows about B and considers B a neighbor. However, connections between a supernode and client are bidirectional - if a client knows about a supernode, then that supernode also knows about the client.

Super nodes are responsible for maintaining the following information about each one of the clients it has connected to it.

- Advertised bandwidth
- Http port the client is listening on
- Available files

Supernodes will also maintain a list of other supernode peers, however other information necessary for routing to that supernode, a supernode does not need to maintain any additional information about its peers. As can be seen above, supernodes are only connected to k other supernode. Below we will give the method for exactly which k other supernodes a given supernode will connet to.

We will now give a description of all the possible actions that can be taken in the protocol: joining, leaving, searching and requesting files.

2.1 Join

In order for a client to join, it must know the identity of at least one node in the system. A client initiates a join by sending a JOIN message to any node in the network. The below cases assume that a client sends a JOIN message to a supernode. If, however, the client being joined on is not a supernode, it will simply forward the request to its supernode.

2.1.1 Join when $\exists$ 1 supernode that is not full

Assume client c is joining the network by communicating with supernode s. c sends s a CLIENT_JOIN message requesting a join. If s has room for c, then s will accept c by sending it a CLIENT_ACCEPT message and adding it to its list of connected clients. If s does not have room for c, it will forward s's join request to one of its supernode peers chosen at random. The new random supernode will then repeat this process of trying to accept the client or forwarding the message to a random supernode if it cannot. A client's join message will therefore be randomly walking throughout the network until it finds a supernode that can accept it.

One subtle note, like many of the messages that will be sent throughout the network, the join message must have a counter associated with it that is decremented at every hop in order to remove the possibility of a message being bounced around forever in a dynamic network. Every time a supernode forwards a message it will decrement the count. If a supernode comes across a join message with a count 0, it will send a CLIENT_ACCEPT_FAILURE message to the client c that initiated the request.

A client will wait some specified timeout period and then retry to join the network. After 3 consecutive retries, the connecting client will send an JOIN_FAILED message to the supernode it is attempting to connect on and another timeout period. After that timeout period it retries the entire process.

2.1.2 Join when the network is full

When a new node is attempted to be added to a full network, a new supernode must be created. A supernode is given the hint to create a new supernode when it receives a JOIN_FAILED message from a client attempting to connect. The process of choosing what currently-connected client to add as a new supernode will be handled with a basic election algorithm, which is described below. When a client c is elected for supernode status, it is sent an UPGRADE message from s.

Upon receiving an UPGRADE message, a client will assume supernode status and being a supernode walk (described in next section) to obtain a set of peers. Because the upgrade was initially caused by a client attempting to join the network, the newly upgraded client must also add the client specified in the upgrade message (which corresponds to the initial peer attemping to join the network) as one of its clients and send it a CLIENT_ACCEPT message accordingly.

2.1.3 Random supernode walk

Another issue that needs to be addressed when creating a new supernode is which k supernodes to add as peers for the newly upgraded supernode. This will be accomplished by a random walk of supernodes. Lets say node t is the new supernode and currently knows about one supernode s. t will send s a RANDOM_SUPERNODE_WALK message, with the initial k set entries set to 0 and the *source* field set to t's address. s will forwarding this message to one of its k neighbors (chosen at random) adding that neighbor to the RANDOM_SUPERNODE_WALK message. Once a neighbor receives this message, he will continue until all k slots in the message contain IP addresses of supernodes, at which point the message is sent to node t (whos IP address will be set as the *source* field of the message). t will then consider the supernodes specified in the message as its neighbors.

Like the random walk when joining, the RANDOM_SUPERNODE_WALK message will also have a counter associated with it which is decremented at each step. If at any point a supernode cannot add any additional entries to the message because all of its clients are already there in the set, the message will be forwarded to a random entry without re-adding it to the set, and the counter will be decremented. If a supernode receives a RANDOM_SUPERNODE_WALK message with a counter at 0, it will send the message back to the client with whatever supernodes have been added so far. It is also important to note that this message

is NOT flooded to all supernodes. If the message is lost (a link goes down between two supernodes who were exchanging the message), then t will timeout and resend a RANDOM_SUPERNODE_WALK message. If t receives duplicate messages, then it will ignore the duplicates unless t does not have k supernode neighbors, in which case it will add the neighbors it does not know about.

Once the new supernode t receives back its RANDOM_SUPERNODE_WALK message, it must add all k neighbors in the message as a neighbor. He does this by sending an ADD_SN_NEIGHBOR message to all the supernodes specified in the Ack. Upon receiving an ADD_SN_NEIGHBOR message, a supernode must randomly choose one of this k neighbors to forget about and add t as a neighbor.

As an example of this, see figure 1. In this simple example, assume that $k = 3$ - each supernode is connected to 3 other supernodes. Client c first sends a message to supernode $s2$ adding it to the

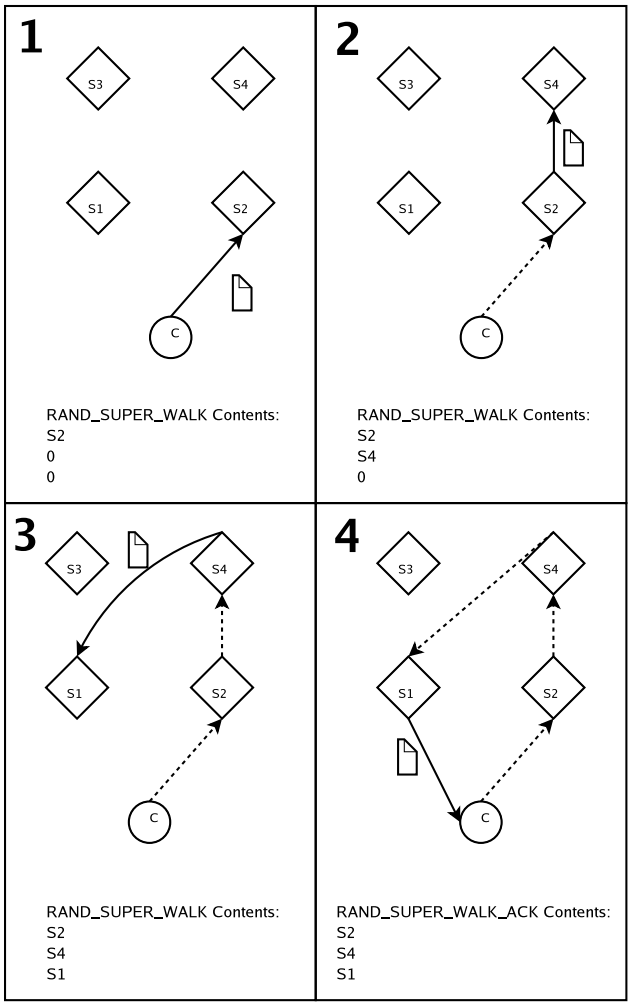

Figure 1: Caption

2.2 Supernode Leave

When a supernode s leaves the network, it will send all of its clients a SupernodeLeave message, which includes the k supernodes that it was connected to. Clients are then responsible for re-joining the network at one of the k supernodes it now knows about using the protocol described above.

2.3 Client Leave

When a client c leaves the network, it will send its primary supernode s a LEAVE message. s will remove t from its list of connected clients.

2.4 Search

Searching will be based on flooding searches to supernodes. If a client c wishes to search, it will send a SEARCH message to its supernode s. s will then check its active index and send SEARCH_RESULT message back to the client who initiated the search. s will decrement the counter field of the search message and forward the search message to all of its peers. If any supernode f receives a search with the counter equal to 0, it will check its active index, but not forward the message to any of its peers.

Each SEARCH message has a sequence number and source associated with it. Every supernode is responsible for maintaining a source/sequence number pair such that, when it sees a SEARCH message from a give source, it verifies that the sequence number for the source is strictly greater than the last sequence number it saw for that node. Otherwise the supernode simply ignores the search request. These search-sequence number pairs should have a time to live on every supernode - if N seconds pass without getting a search request from a client, the pair is removed from the supernodes memory.

2.5 Request

File transfers will all be handled using the HTTP protocol. Every node (both clients and supernodes) will bind on TCP port XX. When a node wants to request a file, it will execute the HTTP GET command to the particular client.

2.6 Supernode Elections

A supernode election will consist of another random walk thoughout the network, this time to find supernodes. The NEW_SUPERNODE_ELECTION will contain, among other things, the client with the best bandwidth that has been seen during the walk. Initially, a supernode s decides that it needs to create a new supernode election when it receives a JOIN_FAILED message from a connecting client. It will then create a NEW_SUPERNODE_ELECTION message, recording the best client it currently has, and forward this message to a random supernode peer in its peer list and, at each hop, the counter in the message is decremented. This process will continue until a timer in the message is 0, at which point the supernode who received the NEW_SUPERNODE_ELECTION message will forward an UPGRADE message to the client recorded with the best bandwidth.

2.7 Active Index Updates

In order for a supernode to be able to correctly handle search requests it maintains an active list of all the files offered by a client. Periodically, a supernode will send a REQ_ACTIVE_INDEX message to all of its clients, who will respond with an UPDATE_ACTIVE_INDEX message containing all the files offered by that client. The period for these updates will be once a minute.

3 Messages

In this section we will define all the messages used in the protocol described above. The message is named, then the fields of the message are described.

- JOIN - Sent from client to supernode and forwarded from supernode to supernode informing of a client's request to join.
  - source - the IP address of the client node trying to join the network
  - count - A counter decremented at each hop of the join
  - advertisedBandwidth - The bandwidth this client is advertising

  - httpPort - The HTTP port this client is currently listening on for file transfer requests.

- ACCEPT_CLIENT - Sent from a supernode to a client informing the client it has been accepted by that supernode into the network.
- ACCEPT_CLIENT_FAILURE - Sent from any supernode to a client informing that the join request has failed (counter on the CLIENT_JOIN message expired).
- NEW_SUPERNODE_ELECTION - A message used to elect a new supernode
  - Best client - client with the best bandwidth we have seen so far
  - Best Bandwidth - bandwidth of this client
  - SN with best client - the supernode who currently has this client.
  - counter - counter decremented at each hop
- UPGRADE - Sent from a supernode to a client to inform it of upgrading to supernode status
- RANDOM_SUPERNODE_WALK - A messaged passed randomly to different supernodes. At each step, if a supernode being forwarded to is not in the supernode_set, it is added to the set before the message is forwarded.
  - source - the address of the node who initiated the random walk
  - supernode_set - k addresses of supernodes initially set to 0.
  - hop_counter - A counter decremented at each hop
  - max_size_of_subset - total amount of peers we are looking for
- ADD_SN_NEIGHBOR - When supernode s sends supernode t this message, it means that t must forget about one of its peers and use s as a peer assuming t has its peer set full. Used when a new supernode is joining the network. One caveat is that the new peernode should be included when choosing a random node to remove from a peer set.
- JOIN_FAILED - a message sent from a client to a supernode informing the supernode it has tried 3 consecutive attempts to join and failed. Supernodes take this as a hint to elect a new supernode for the network.
  - source - The address of the client trying to join
  - advertisedBandwidth - The bandwidth of this client
  - httpPort - The port this client is listening on for HTTP requests.
- SEARCH - A search message that is flooded to supernodes whenever a node wishes to search for a file.
  - source
  - sequence number
  - search string
  - hop_counter
- SEARCH_RESULT - The result of a search message
  - sequence_number
  - Results of search
- LEAVE_SN - A message sent from a supernode to all its clients when the supernode wishes to leave the network.
  - List of alternate supernodes.

- LEAVE_CLIENT - A message sent from a client to its supernode when the client wishes to leave the network.
- UPDATE_ACTIVE_INDEX - A message sent from client to supernode informing the supernode to update its active index
  - List of available files
- REQ_ACTIVE_INDEX - A message sent from a supernode to a client requesting an updated active index.