Mcast: A Binary Tree-based Multicast File Distribution Protocol

Final Design Document
EECS 489, December 2005

Project Team: Tehmasp Chaudhri, Albert Lim, Kunal Mahajan

Project Overview / Goals

Mcast is a file distribution protocol based on top of a multicast design which makes use of a binary tree for the efficient delivery of a file from a central server to her client nodes. It allows a content distributor to serve files to clients by leveraging the network bandwidth of all the client nodes within the system and thus limiting the work and the network bandwidth required by the distributor. The most important goal was to create a system where a file (especially a large file) could be distributed across many clients as efficiently as possible over a binary tree network topology.

Architecture

Mcast is a distributed system based off of the client server paradigm. Within the system there is a central server which accepts incoming connections from new clients and joins the new clients to the multicast group tree. The server forces all new clients to either connect to the central server or another client node. Thus all nodes accept new connections and each node within the system (including the server) distributes the file directly to two children client nodes.

The central server keeps a directory containing a list of all the client nodes currently within the system. Each node within the directory is identified with its IP address and information about its parent node, left child, and right child. This information facilitates the proper creation of the distribution tree and the reordering of the tree in the event of node failures. Additionally, each parent node must make sure that each of its child node has a copy of all the file data it has received (essentially mimicking a reliable multicast protocol).

In the event of client failures, a client can attempt to reconnect to another parent node by requesting a new connection from the server. A client can also continue its file progress in the event that it must re-connect to a new parent node. The server will also attempt to fix the network topology by allowing new clients (or those requesting new parent nodes) to attempt to connect to parents which had previously been abandoned by their child nodes. The server also time stamps each new node joining the group, thus allowing it to prefer those nodes with earlier time stamps (with the assumption that a node with an earlier time stamp has more file data). This enforces that nodes with a larger percentage of the file are higher up in the distribution tree.

See appendix A for a schematic of a possible Mcast network topology.

Plan of Attack (milestones)

The following were our project milestones, the order in which we built Mcast:

- Develop the central server which accepts client requests and adds the clients to the directory. This directory will allow the server to build a network of connections and enforce a binary tree topology.
- Implement functionality to allow clients to receive file data from their parent's and pass this data on to their children nodes.
- Implement a global restructuring algorithm which will entail clients talking with the central server to ask for new parents in the event of parent node failures.

Mcast API

Mcast was designed like a library allowing an API user to make use of Mcast to create their own application.

The following API functions provide the functionality of the server within Mcast:

```
// Initializes the server's directory and other state information.
bool InitMcastServer(McastServerData &server, string host_name);

// Create a new GID for 'file_name' and add this
// to the server's 'filename_pool' and 'gid_pool'.
// Returns the new GID for 'file_name'.
// Returns '0' on failure (which can be because 'file_name'
// is greater than FILENAME_SIZE.
unsigned int Create(McastServerData &server, string file_name);

// Begin accepting client connections and sending file data.
// Also service any other requests from clients.
bool StartTransfer(McastServerData &server);

// Print the server's state of current connections and
// progress of file transfers.
bool Statistics(McastServerData &server);
```

See appendix B for a schematic of the server API.

The following API functions provide the functionality of the client within Mcast:

```
// Join the client to the multicast group identified by 'gid'.
void Join(McastClientData &client, const unsigned int gid);
```

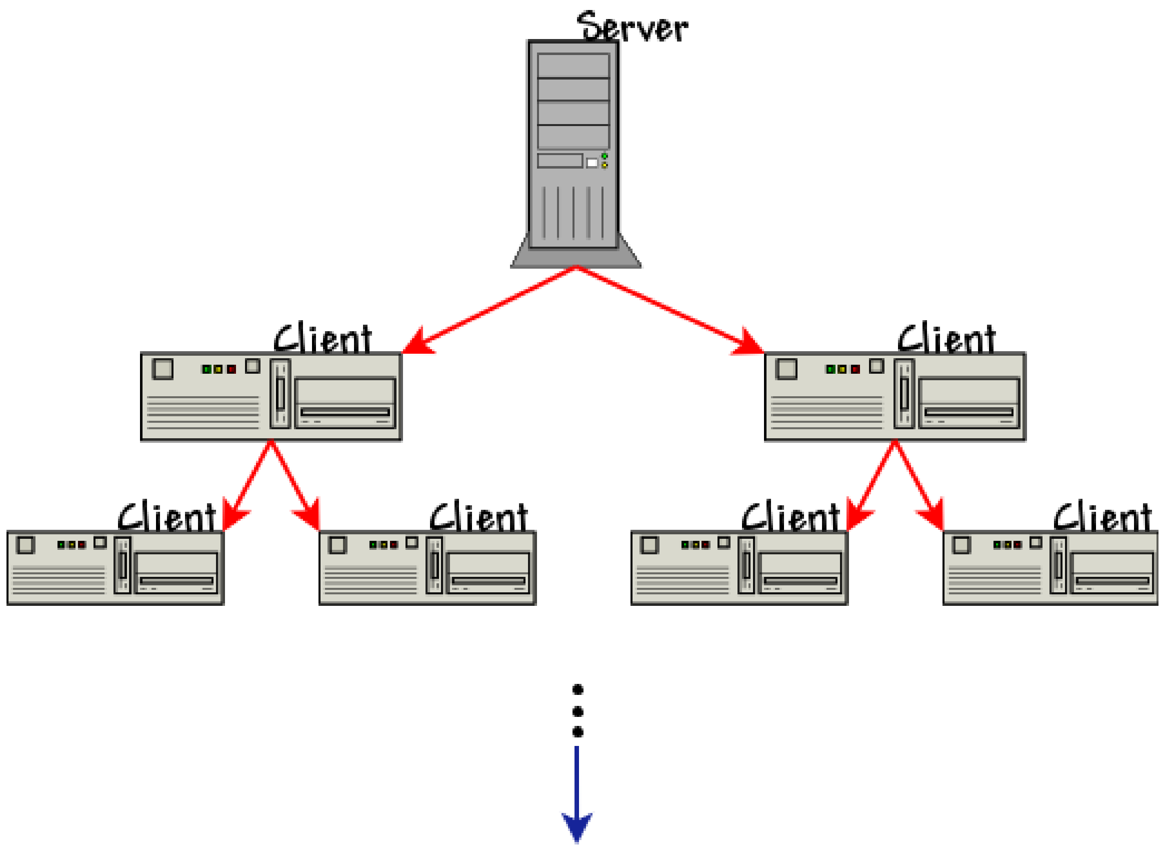
```
// Begin the receiving of file data from the parent.
// Also service any other child node requests and transfer file data to those children.
bool StartTransfer(McastClientData &client);
```

See appendix C for a schematic of the client API.

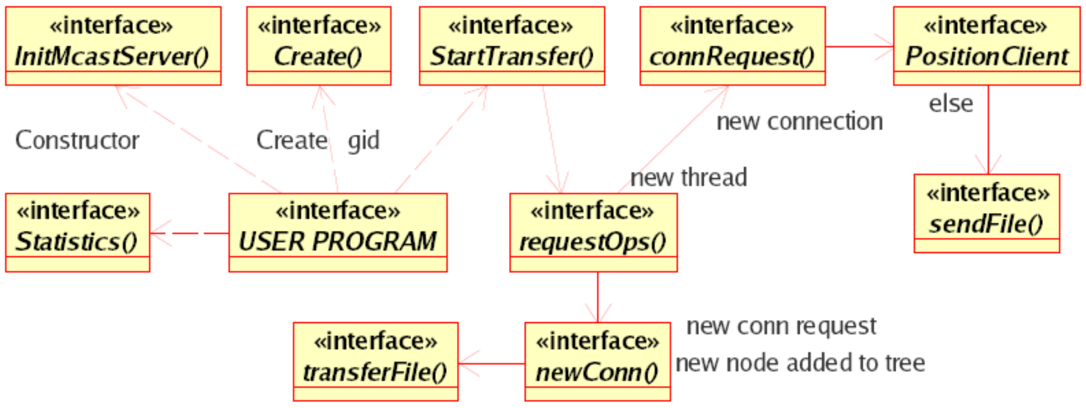
Conclusion

In conclusion, we feel that we gained great experience in designing, developing, and implementing a protocol from scratch. We realized that using a binary tree network topology was efficient for transferring a file in a multicast fashion. We feel that Mcast is a usable file transfer protocol and that it can serve as a foundation to bigger and better ideas.

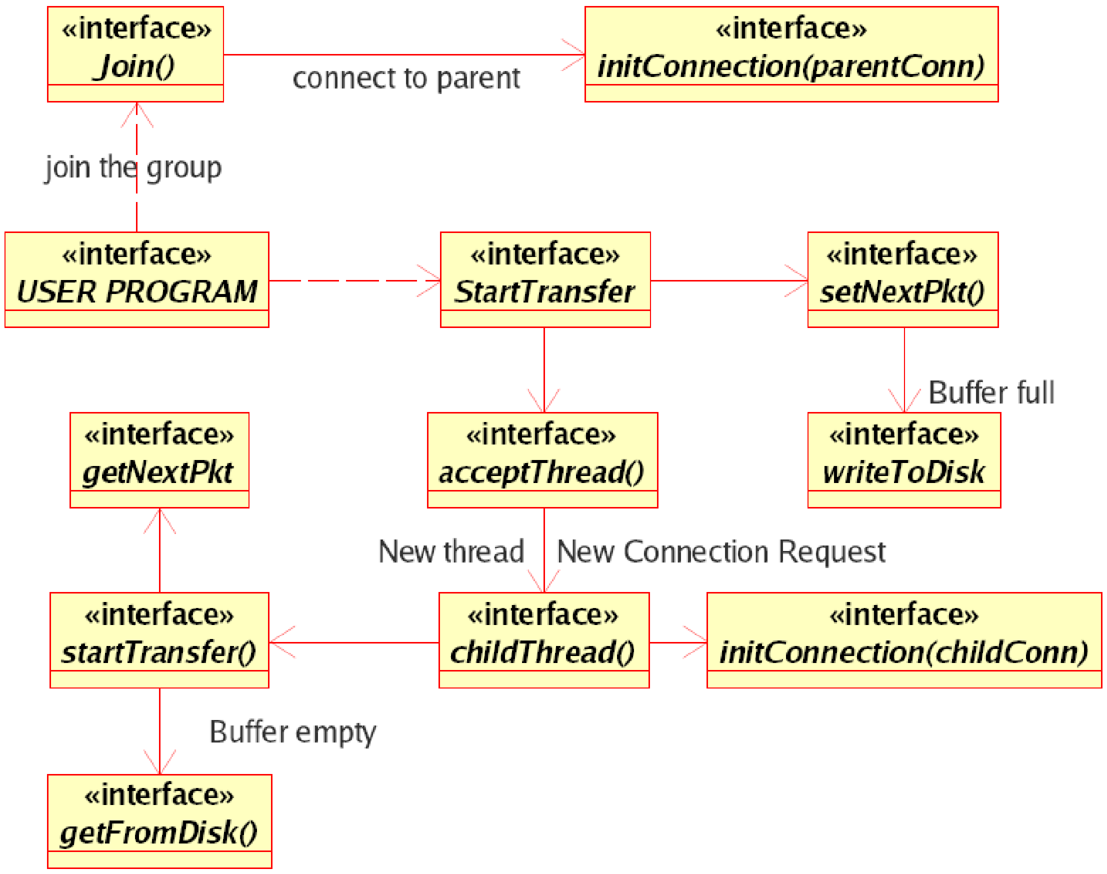
Appendix A



Appendix B


«interface» InitMcastServer()
«interface» Create()
«interface» StartTransfer()
«interface» connRequest()
«interface» PositionClient
else
new connection
Constructor
Create
gid
new thread
«interface» Statistics()
«interface» USER PROGRAM
«interface» requestOps()
«interface» sendFile()
«interface» transferFile()
«interface» newConn()
new conn request
new node added to tree

Appendix C


«interface» Join()
«interface» initConnection(parentConn)
connect to parent
join the group
«interface» USER PROGRAM
«interface» StartTransfer
«interface» setNextPkt()
Buffer full
«interface» getNextPkt
«interface» acceptThread()
«interface» writeToDisk
New thread
New Connection Request
«interface» startTransfer()
«interface» childThread()
«interface» initConnection(childConn)
Buffer empty
«interface» getFromDisk()