Telecommunications Assigment #3

Ciarán Moyne, Tom Murphy, Jonathan Rosca

January 10, 2014

0.1 Introduction

This a system of distribution of files over a local network, using UDP multicast sockets for efficient one-to-many transmissions.

It does not require a server, but requires a router capable of multicast networking, so only operates on the local network. Every participant is called a peer.

0.2 Protocol

0.2.1 Network membership

For a client to be part of a network of peers, it need only have access to other clients in the local computer network. Every client maintains a list of known peers. The network address is used to identify a peer, since it is unique within the local network.

In order to let the other peers know that a new client has joined the network, a small presence packet is sent to the multicast socket. When received by the other peers, they each add the new peer to their list, and keep track of when it was they received the last presence packet, as well as the nickname. The presence packet is sent periodically (every 200 ms) by each peer for their entire runtime. If a peer's presence was announced less recently than 5 seconds ago, they are thought to have left the network, and are deleted from the know peers list.

The packet also contains a nickname for the client, picked by the user. It cannot be longer than 30 unicode characters. It is only used to allow the users to identify peers in the network, and can be changed instantly. It does not have to be unique between the peers. It will be updated for the other peers when they receive the next presence packet from this client.

0.2.2 File distribution

Metadata

The relevant metadata of a file is the filename, size, type(image, video, audio or other), sha256 hash digest of the entire content and the time to live (picked by the user, not stored in the file). A file may optionally include a thumbnail of up to 90x90 32-bit RGBA pixels, where appropriate and possible, that will be transmitted uncompressed.

Recipients

The recipients are the known peers at the time of the start of the transmission. If a new peer joins the network after the transmission has started, it is excluded from the transmission.

File chunks

The file contents are sent as chunks, which are as large as possible (almost 64KiB) in the UDP protocol. The system thus relies on IP packet fragmentation and error detection to drop incorrect packets. If the layout of a packet is correct, an error in the payload of an individual chunk packet cannot be detected.

Transmission

The metadata is sent to all recipients. When every recipient acknowledges the metadata, every content chunk is sent until each one is acknowledged by every recipient. There is a delay between every packet from each file so that the network isn't flooded with redundant packets before the peers have chance to reply.

Lifetime

The "life" of a distributed file on a peer's client starts when it was fully received. The time this happens can be different for each recipient, since different peers can have different chances of dropping packets. Each recipient will delete its copy of the file when the time is $arrived\_at + ttl$. For this reason, in case the distributing peer does not want to prematurely stop the distribution of the file, the receiving peers do not have to send deletion acknowledgement packets when they have deleted the file.

At any point, from the start of the transmission of metadata, the user might change their mind and wish to stop sending the file and have it deleted from the peers. The peer sends a deletion request to the recipients, repeatedly, until they each acknowledge having deleted their copy of the file, partial or complete.

0.3 Implementation

The program is implemented in Python, using an object orientated style. The file peer.py contains the entry point, so to invoke the program, one runs $pythonpeer.py$. At the time of writing, the PIL library is not exist for Python 3, so Python 2 must be used insted. Care was taken to make the code forward compatible, though.

0.3.1 Libraries

The following (notable) Python libraries are used. They are part of the Python project and should come pre packaged.

1. **struct**: used to create and read binary data such as packet header.
2. **socket**: used for low level networking.
3. **hashlib**: used to compute the hash digest of files.
4. **PIL.Image**: used to create thumbnails for images.

0.3.2 Main thread

This thread initializes the system and spawn the networking thead, then starts taking user input, as a rudimentary console. The console allows the user to see a list of peers(ip and nickname), send a file (given a path and time to live), or end the program.

Transmission start

When the user initiates a transmission, an OutgoingFile object is created and added to the outgoing files list. The object loads the file entirely from the file system. It divides the file into chunks, and computes its sha256 hash digest. The type of the file is determined based on its suffix. If the file is determined to be an image, it is opened as a PIL.Image object and converted to a thumbnail and included in the metadata packet.

0.3.3 Network thread

This thread processes acknowledgements and presence packets, then sends a single packet from every file, if that file is still 'alive' and has a packet to send, in a loop.

Transmission

In every outgoing file, there are three lists of current recipients: for metadata, content chunks and deletion acknowledgements. They contain which packets were not acknowledged yet(even if the packets weren't yet sent). If a recipient times out. it is excluded from the transmission and deleted from the lists.

After every recipient has acknowledged the metadata packet (or timed out), every time the file is queried for a packet, it picks a recipient at random that has unacknowledged packets, and generates a chunk packet for the first unacknowledged chunk of that recipient.

Reception

When a metadata packet is received, an IncomingFile object is created with the data from packet, and it is added to the list of incoming files.

When all the chunks of a file are received, their hash in computed and compared to the one from the metadata. The latter is assumed to be correct, so if the hashes do not match, the content is thought to be erronous, and the file is discarded.

0.3.4 Drawbacks

If a file is completely received, and a single chunk is incorrect, causing the hash not to match the metadata hash, it cannot be salvaged and is dropped entirely. A possible solution would be to send a partial checksum along with every chunk packet, so that the recipient will know was the first erronous chunk it received, and then send negative acknowledgements for every packet after that.

The program cannot deal with disconnecting from the network.

Sadly the implementation is not complete.