Multipeer – An implementation of the Multicast based peer to peer architecture

This implementation is a RAD, proof of concept implementation. It has not been designed for speed, but rather for testing of the architecture and file transfer.

To increase the speed of development, Python has been chosen as the programming language, along with wxPython for the GUI. The program has been designed in a modular fashion, with an API to make it possible to create several UIs. There are currently two UIs, available a GUI and a server, non-UI:

The implementation is somewhat limited due to time constraints, but the following features are working:

- A rudimentary GUI for searching for files, downloading them and showing the download progress
- A GUI dialog to add new files which others may download
- Downloading
- Sending a multicast packet to query which peers has a particular file
- Filechecking to ensure that all blocks are received.

But with these limitations due to time constraints:

- It is currently only possible to download the entire file
- It is currently not possible to download the same file, but from different peers (automatically)
- It is currently not possible to send the same filesession to more than one client
- There is currently little input validation
- Program robustness has not yet been fully implemented
- The program is not yet truly platform independent

In order to run the program one needs:

A Unix or Linux workstation with at least python 2.2.x (earlier versions may work), they do of course need at least an IPv6 stack. In order to get the GUI working, wxPython 2.4.x (earlier versions may work) is needed. The program is known to run on Linux Red Hat 7.3 and 9.0, but should run on all systems which fulfill the requirements above.

The system should additionally run under all systems which can run Python and wxPython, but at least the Windows port of Python does not have IPv6 support, (contrary to what the documentation says), so the program cannot currently run under Windows. We are however looking into this problem.

To start the interactive GUI, use the file: ‘gui.py’

To start a peer from which only downloading is possible, and wxPython is not needed, use the file ‘test.py’, which is a pure server program.

The program is currently not designed for performance, and speeds over 4 KBps is currently unlikely, but it easily fixed. This also ensures that network congestion should not be a problem due to use of the program until a proper network congestion algorithm and functionality has been put in place.

Application concepts:

Due to time constraints, it has not been possible to design and fully implement the algorithms needed to enable downloading parts of the file from multiple peers, nor sending the filesession to more than one client at a time. Since file transfer has proven successful, this is a limitation of the implementation itself, and not the underlying principles.

Multicast groups:

File queries are sent as ASM on the SSM multicast group ff3e:30:2001:700:a00:5::1 , while the filestreams are sent on a multicast group chosen randomly with the prefix ff3e:30:2001:700:a00, these groups has been choses in order to avoid collision with other IPv6 multicast traffic, and since we have been delegated the prefix 2001:700:a00, it is by RFC our address area for SSM. Later versions will probably use SSM for the filestreams, and not ASM which is used today, this in order to support a very large number of possible sessions.

Due to problems with our providers connection with the M6Net and on to the M6Bone, this may cause a problem when the program is tested on a network, which has multicast connectivity with the M6Bone, as our provider is the RP for the entire ff3e:30:2001:: prefix.

Is it advisable to start the program on at least two nodes, as there is no guarantee that it will be able to communicate with our peer, or that other peers are currently running the program. The program is currently configured so that its node does not receive its own multicast traffic.