Detailed nlnet application

31 maart 2009

1 Abstract

This application concerns porting a distributed data store to the I2P anonymous network. First, there will be an explanation about the distributed data store I wish to port (Dijjer[1]). Secondly, I will go over the different tasks I wish to complete. This includes porting Dijjer to I2P, exploring the scalability and possible speed improvements to the data store. Also, some general improvements could increase the use of Dijjer.

Dijjer - An introduction Dijjer is a distributed data storage and web cache system. Dijjer decreases the amount of bandwidth needed to host a file, by downloading parts of the file through a network of other computers (also referred to as nodes or peers) running Dijjer. Through the use of integrity checks, it makes sure sections of the file are not accidentally or purposely altered.

Communication through the network is done using a Dijjer-specific protocol on top of UDP, but files are seemlessly passed through to the browser (or another program) by HTTP. If a regular file is downloaded from http ://www.mywebsite.org/hugefile.avi, downloading through Dijjer is possible using http://127.0.0.1:9115/http://www.mywebsite.org/hugefile.avi.

2 Project proposal

Porting Dijjer to I2P A first task I wish to complete, is porting Dijjer to the I2P network. Anonymous networks like I2P are often quite slow (due to the extra routing) and somewhat unreliable: downloads can terminate or get corrupted due to nodes going offline. In addition, many nodes only have a limited amount of bandwidth available, which can be problematic for people hosting popular content.

A distributed data storage system can mitigate the reliability and bandwidth problems, by contacting multiple peers for the content, and not re-

[1] http://code.google.com/p/dijjer/

questing everything from the same node. If parallel downloading is implemented, the speed issue can be taken care of as well.

I wish to port Dijjer to I2P without introducing any dependencies on the I2P network. To execute this port in a clean way, the process can be split up in two main steps. First, a plugin architecture needs to be created. This allows one to add a new network system to Dijjer, without new dependencies arising. I have already implemented such a system for Dijjer[2], but it's possible that implementation faults show up during further development. In a second phase, the plugin itself must be written. This will require reimplementing all the Dijjer classes involved in networking using the I2P API. In addition, I intend to write a lot of tests for this implementation, as it will differ a lot from the 'regular' Dijjer. For example, latency will be a lot higher and have a larger standard deviation (which can cause problems with timeouts).

Routing and throughput improvements A second task I plan to work on, is researching the routing as well as the throughput provided by Dijjer nodes. The routing algorithm Dijjer uses is very similar to Freenet routing[3][4], but has not changed in the last few years. It would be useful to research more recent changes in Freenet routing, as well as looking for other improvements. In addition, I wish to research how well suited more 'standard' Distributed Hash Table systems like Kademlia are for a distributed data store, both for routing and for spreading data through the network.

The routing matters a lot in a larger network, but throughput problems can appear even in very small ones. Currently, unexpected delays and changes in transmission speed can be noticed when transmitting or receiving a file through Dijjer. To take care of these problems, I will combine statistics found using packet sniffing with simultaneous debug output from Dijjer. I hope this will allow me to pinpoint the exact code that is responsible for these speed issues. In addition, tweaks to the parameters of the speed throttling algorithm used might squeeze a bit more performance out of the program.

General improvements There is a list of smaller improvements that I've bundled in a third task.

Dijjer currently uses the MD5 hashing algorithm, which has been shown

[2]The commits for the plugin system start with revision 21: `http://code.google.com/p/dijjer/source/list`. Commits on SVN started on March 20th, but these were checked into the I2P repository a lot earlier (starting in September 2008). Information on accessing the I2P monotone repository is available at `http://www.i2p2.de/monotone.html`, the branch is i2p.eepdijj

[3]Freenet routing explanation: http://freenetproject.org/ngrouting.html

[4]Both projects were started by the same person: Ian Clarke.

to no longer be safe[5]. As such, I would like to replace the algorithm by SHA-2 to retain security.

Automatic updates have also been implemented in Dijjer, but currently still centralised. It should be trivially possible to use Dijjer for this, but I'd like go through the update system to make sure such a change will not cause any problems. I also want to make sure the update process runs securely, as forged update files can compromise a nodes computer.

A status page is available for users at `http://localhost:9115`. I would like to add configuration options to this page, to improve usability.

Last but not least, Dijjer does not conform to the Java 1.6 standard. I would like to bring the code up to date to remove this issue.

3 How does the project help Dijjer and I2P?

- Porting Dijjer to I2P adds a data storage system to the anonymous network. This will make system updates a lot less heavy on the people hosting the updates, and will allow mirroring of websites that are offline or hard to reach.

  In addition, making sure the Dijjer plugin system works correctly, will allow porting Dijjer to other overlay networks as well. This should attract interest to the data store and spur development.

- Routing improvements will give Dijjer a much better scalability, while throughput improvements will allow fast downloads through the network.

- The general improvements will be useful for Dijjer as well:

  - Replacing the hashing algorithm will make Dijjer more secure by avoiding data alteration.
  - Decentralised automatic updates make the update process scalable by removing the 'single point of failure'.
  - Adding configuration options improves usability, since users will no longer be required to enter their configuration at the command line.

[5]Collision attacks are possible: http://www.win.tue.nl/hashclash/rogue-ca/