Bridging the two BitTorrent Worlds I2P and P2P

Tarang Chugh

N° XXXX

November 2012

Networks and Telecommunications

ISSN 0249-0803 ISRN INRIA/RT--XXXX--FR+ENG

Bridging the two BitTorrent Worlds
I2P and P2P

Tarang Chugh

Theme : Networks and Telecommunications
Équipe-Projet Madynes

Rapport technique n° XXXX — November 2012 — 23 pages

Abstract:

Key-words:

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Internship Report

Résumé :

Mots-clés :

Contents

Note

This work has been done in the continuation of Juan Pablo Timpanaro's previous works and under the guidance of Dr. Isabelle Chrisment. I would like to thank them for their support throughout the project. A paper published during the course of development explaining the work can be found here [1].

Chapter 1

Introduction

In today's world, with growing cyber-security threats one prefers to be as anonymous as possible in the virtual world. One of the major applications of this virtual world is data exchange through peer to peer networks. P2P networks have a drawback that user activity can be tracked and leaks one's identity.

Apart from the open-net P2P, there also exists a comparatively much smaller network known as I2P net, which makes the data exchange and other web services anonymous. A major drawback with I2P network is its low quantity of content.

This project aims at solving this problem by bridging the gap between open-net P2P and anonymous I2P. It will make the file sharing anonymous and also contain huge amount of data as available in P2P network.

The tool being developed, called as bridge is not just a proxy to hide the identity of the users but acts as an entity which is a part of both the networks. The bridge contains an I2P as well as P2P bit torrent client along with the functionalities that helps in synchronizations between the two. It has various components which enable the bridge to start bridging a torrent by just knowing its hash, perform good caching by analyzing which are the popular pieces of shared files, etc.

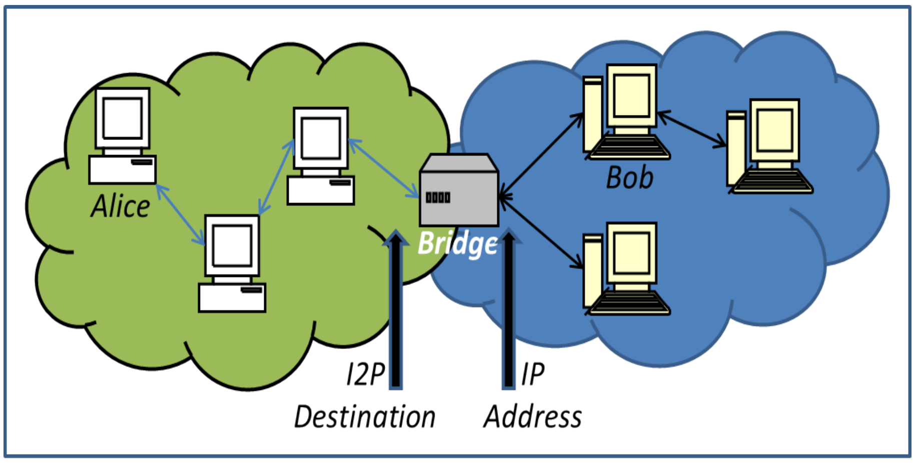

Figure 1.1: Bridging I2P and P2P Networks

Chapter 2

Initial State of the Art

2.1 Abstract Level

On an abstract level , the bridge can be seen as having three major components :

1. I2P Interface : responsible for receiving the bridge requests, file pieces and other control communication from I2P end;

2. P2P Interface : responsible for the similar communication as mentioned above in P2P side;

3. Central Module : responsible for the synchronization between the two interfaces, keep a track of various details about each connected client, interacting with cache manager, etc.

2.2 Detailed Level

The bridge at a detailed level can be broken down into various components responsible for a unique functionality.

2.2.1 I2P Snark Client

Initially the bridge consisted of an I2P client called I2P Snark. This belongs to the I2P interface and is responsible for the data communication between I2P network and central module of bridge.

This was one of the self sufficient components which did not require any major changes other than the one of DHT which will be explained later.

2.2.2 P2P BT Client

At the beginning, Bridge consisted of a BitTorrent library called JbitTorrentAPI which acted as a BT client to communicate with P2P network. This library lacked the support of UDP trackers. More information on JbitTorrent API can be found here : [3]

2.2.3 Shared Storage

It behaves as the major component of the central module as it is responsible for the synchronization between the two interfaces (I2P and P2P). It is also responsible for interacting with the cache manager.

2.2.4 Tracker Client

Each of the two clients have tracker clients which are responsible for contacting and tracking the peers in their respective networks. They also try to reconnect to disconnected or to new peers found.

2.2.5 Cache Manager

In the initial phase, caching was very basic and was present for just the proof of concept. It had a policy based on only time that would clear the complete cache on timeout.

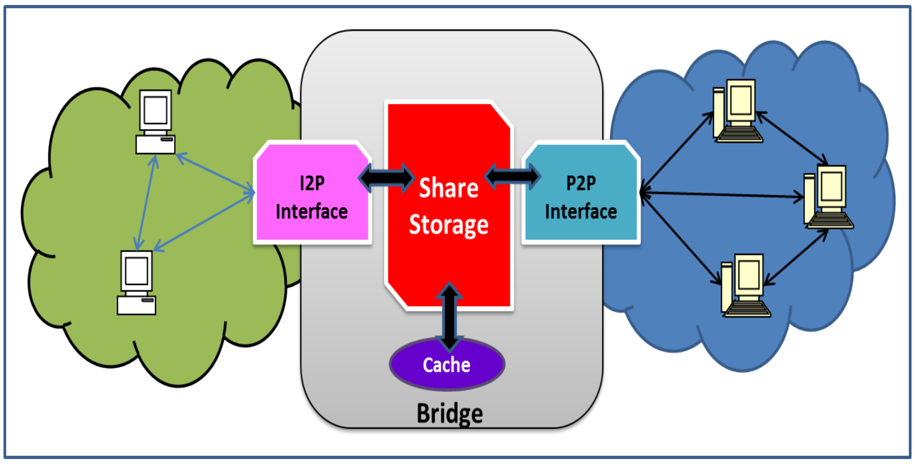

Figure 2.1: Initial Model of the Bridge

Chapter 3

Requirements

3.1 Snark Library

As described in the section of initial state, the bridge used snark library on I2P side and a very basic library called JbitTorrentAPI. It was perceived that if we use the BT Snark library(i.e. Snark clients on both interfaces) which has been designed for P2P network, it will make the bridge well synchronized. Therefore one of the initial requirements was to replace the JbitTorrent API with BT Snark Library.

3.2 UDP Tracker Support

Recently many torrents are making use of the UDP trackers instead of HTTP trackers. The bridge already had UDP tracker support on I2P side but lacked it on P2P side. Therefore, it was a requirement to impart the support of UDP trackers so that Bridge can be used for torrents supporting only UDP trackers as well.

3.3 Advanced Caching Policy

With the initial caching unit, the adopted policy was based only on time, which cleared the whole cache after a certain time-out. This policy was in place as just a working model for cache. It did not take care of any other factor.

It was required to introduced a new policy of caching based on time as well as popularity of pieces. To increase the efficiency of the cache it is important to have maximum cache hits, thereby caching the pieces which are likely to be queried most.

3.4 Bridge Client-Server Communication Protocol

In the initial state, the communication between bridge server and client were done by sending the strings as byte streams. It was a working model, but not a robust one.

It was required to have a defined protocol which would have a set of certain defined messages. Also to remove the use of Strings of long lengths and have a definite structure for the messages exchanged and ability to carry variable messages as payload.

3.5 MainLine DHT

It has been recent advancement to make the torrents tracker-less. It removes the dependency on a central service of trackers and remove them as a point of failure. Instead of a central approach, DHT introduces a distributed approach. It was one of the important requirements to be integrated in the bridge. After its integration the bridge would be able to start bridging a torrent given only its hash, without contacting the trackers.

3.6 MetaInfo Exchange

With integration of Mainline DHT, Bridge would be able to get a list of peers which are currently sharing the file. But the Bridge still requires the metainfo about the torrent to initialize various components like storage etc. It was required to implement the metainfo exchange manager to get the metainfo from the discovered peers.

3.7 Multi-Bridge Network

All the above requirements mentioned above were for a single unit of bridge. But with increasing load on one bridge, it is required to have load balanced between multiple instances of bridges. It was required to have an intercommunication between bridges to be able to communicate :

1. which all bridges are bridging the same torrent and can act as peers or alternate sources for the torrent, and
2. efficient caching of preferably non-overlapping pieces in order to contain the largest percentage of file cached in a distributed way.

This requirement itself is an aggregation of 3 sub-requirements:

1. Bridge Discovery : How a bridge will find new bridges which have been started in the network?
2. Communication Protocol : What all messages and in which structure will be exchanged between the bridges?
3. Distributed Caching : Which policy should be used to perform efficient distributed caching ?

Chapter 4

Development Cycle

Initially I started my work by getting acquainted with the I2P network and already developed bridge. I read the paper describing about the initial state of bridge and understood the concept of I2P net.

4.1 Snark Library and UDP Tracker Support

The first requirement was to replace the existing JBitTorrent API with the Snark Library. But the standalone version of BT Snark library itself was not able to download any file. Its last version was released in 2003 and the development of BT Snark was stopped then. All the work done to integrate it in the bridge and replace JBitTorrent API was not useful. But it helped in getting a better understanding of the working of the different modules of the Bridge.

The current model only had the support of HTTP trackers. But the whole Bit Torrent community is shifting towards use of the UDP trackers. Therefore it was necessary to incorporate the support of UDP trackers in addition to HTTP trackers. This was our next requirement. In the meanwhile I found a new BT Client library known as tTorrent Library [5] developed by Turn Inc. This new library included support of UDP trackers and also had a better performance than JBitTorrent API and BT Snark. It's a latest API for making BT clients in Java and is also updated regularly with latest features.

Then I created a standalone program for tTorrent API so that it is able to service the request of downloading a BT piece that user specifies. This simulation was necessary because a similar working prototype is required which will be furnishing requests that arise from I2P side of the bridge network.

Integrated the standalone within the bridge architecture and tested. The standalone integrated bridge was working well for the first request generated from the I2P client side but failed before arrival of second request. It was due to the fact that peer got ready to send a piece but the new request was delayed. Introduced a waiting state in tTorrent which helped the API to wait for the consequent piece requests even if the peer got ready before it. To solve this problem, random pieces were requested before the tTorrent library received a real request from the I2P side. After testing with the standalone version of tTorrent library, it was integrated in the Bridge thereby replacing JBitTorrent API.

4.2 Caching Policy

The initial state of the caching unit is already described in the above section. The requirement was to take into account the popularity of pieces and not just its time-stamp. This is because some pieces might be very popular and are likely to be queried again but are not accessed recently. Thus both the factors should be given some weight-age. For this purpose, we find out the pieces which are popular (in respect of number of queries) as well as pieces which are rare in terms of availability in network.

Initially : In the initial state, the policy adopted was to clear the complete cache after a time out of 10 minutes.

Enhanced Caching Policy

$$Time - Stamp(v_1)$$

$$Popularity(v_2)$$

$$\alpha v_1 + \beta v_2 = \delta(score);$$

$$where \quad \alpha + \beta = 1 \quad and \quad v_1, v_2 \quad are\, normalized\, values$$

Using the two factors of time (time of last access) and popularity (more frequency of piece requests and less availability in the network), we calculate the weighted score. Pieces with lower score are removed from the cache when cache is getting full. It increases performance as popular pieces still remain in cache even if they are a bit old as compared to recently queried not so popular pieces.
The weights assigned to each of the factor can be changed in different variable bridge environments depending on the number of current torrents, cache capacity, etc.

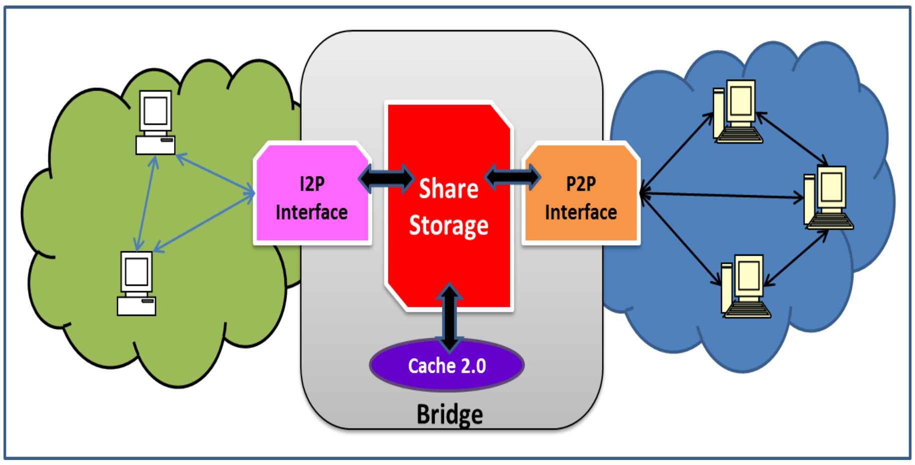

Figure 4.1: Bridge Model with advanced Caching Unit

4.3 Bridge Server Communication Protocol

We introduced the use of Byte Buffers over Writable/Readable Byte Channels with a proper structure of messages transferred. Each message contains an initial byte as type byte. The different types in the protocol are :

CLOSE_CONNECTION(0),
BRIDGE(1),
ALT_BRIDGES(2),
BRIDGE_I2P_DEST(3),
SEEDER_I2P_DEST(4),
I2P_PEERS(5),

ACCEPTBRIDGE(11),
DENYBRIDGE(12),
ALTERNATIVES(21),
BRIDGEI2PDESTINATION(31),
SEEDERI2PDESTINATION(41),
I2PPEERS(51),
BADREQUEST(99);

The advantages of this type of structure and communication is that it happens through ByteBuffers instead of Strings. It is designed to facilitate easy extension. It also has better performance and efficient memory usage as we use few bytes instead of long strings.

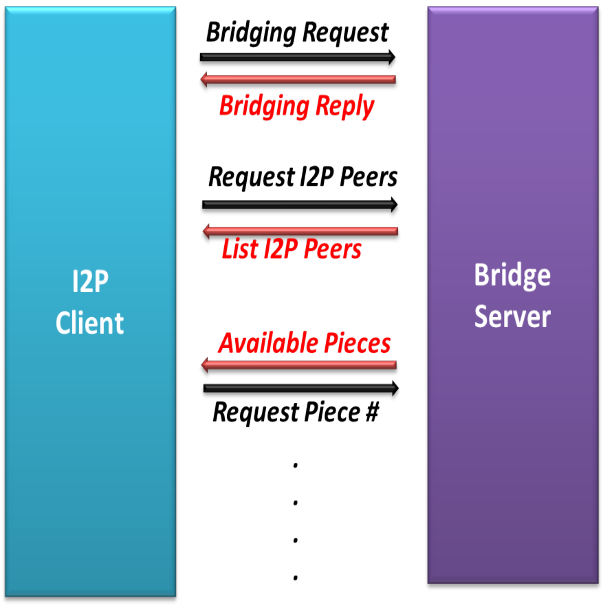

Figure 4.2: Sample Communication Messages

4.4 MainLine DHT

It is evident that how use of DHT makes the concept of torrents as distributed. It removes the point of failure of trackers and also returns a list of only recent peers sharing the given torrent. MainLine DHT is the name of the kademlia based DHT and more information can be found out about it here : [4]

It is part of the BEP 5 of BitTorrent Protocol. Every peer announces the torrents it is sharing in the DHT and can query for other peers given a torrent hash.
With DHT, another advantage is that we can have a continuous discovery of peers.

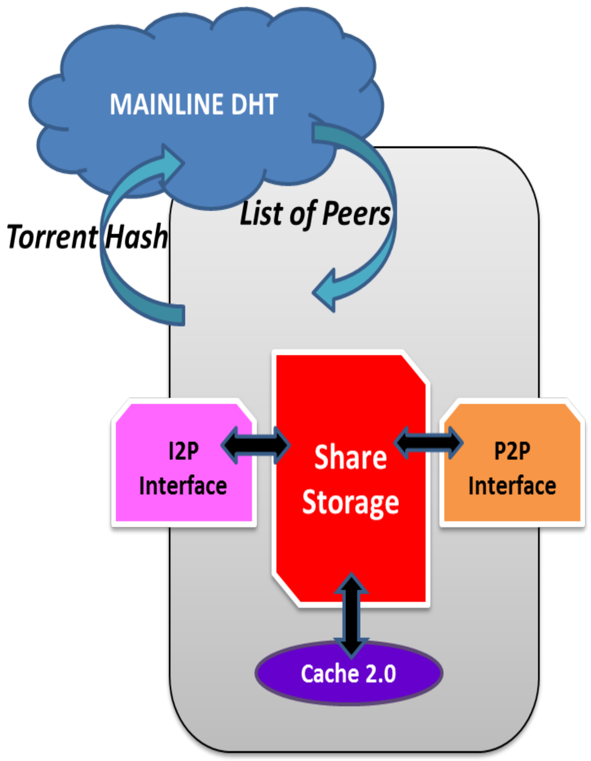

Figure 4.3: MainLine DHT

4.5 MetaInfo Exchange

After we get the list of peers from DHT, we want to get the MetaInfo to initialize various components. To exchange meta information, the protocol defined in BitTorrent is known as Extension Protocol (BEP 10 and BEP 9). These extension messages to request for meta info file is sent just after the handshake.

The process to obtain meta info from the peers discovered in DHT is multi-threaded to improve the performance. As soon as one of the contacted peers return the meta Info, connection with other peers is terminated and the bridge starts to initialize all the components.

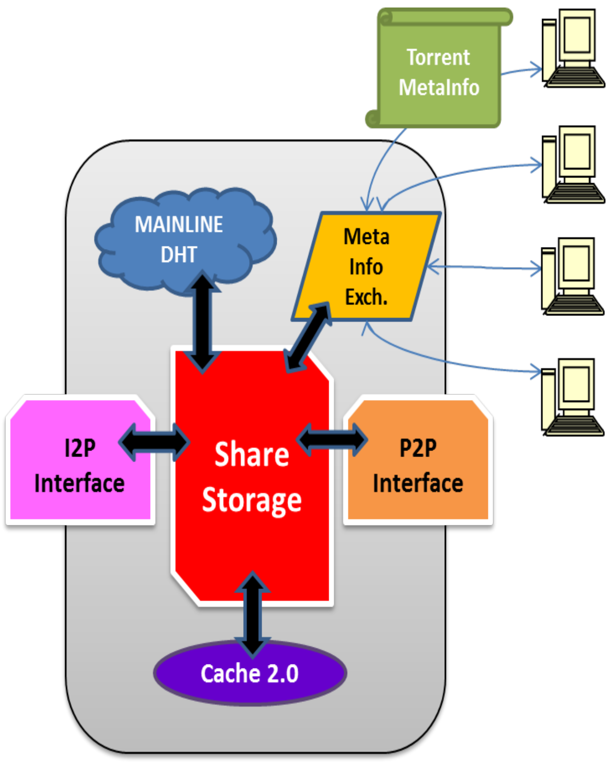

Figure 4.4: MetaInfo Exchange

4.6 Multi-Bridge Network

To develop the communication between multiple instances of bridge we had many possible options to be considered.
For multiple bridge overlay, we have the following 2 options:

Option 1 :
Use of Trackers for Content and Bridges: No Inter-bridge Direct Communication

In this option, we make use of trackers as the source of information about I2P peers and bridges which are sharing (i.e. seeding/leeching/bridging) the given torrent. Currently these trackers are supposed to be hosted at postman's trackers.

USE-CASE
When an I2P client wants to download a given torrent, it checks the trackers list for that torrent. If the tracker already exists then the client obtains information about other I2P peers and bridges and contacts them. In case a tracker does not exist, client requests a new bridge to start bridging that torrent from P2P network. When the bridge starts bridging the torrent, it is required to create a new tracker for that torrent and register itself and the I2P client (original requester) in it.

It is also possible that the tracker exists but the specified I2P peers are not available /

offline and bridges are not currently bridging that torrent. In that case a new bridge is requested for the torrent and that bridge is required to register with the tracker.

Thus, it becomes client's and bridge's own responsibility to take decisions based on information they acquire from trackers. Every new I2P client is required to contact tracker to get information about the Bridges/Other I2P peers leeching or seeding on basis of content.

Advantages

- It solves problem of bridge discovery in an easy way, and the discover of peers and bridges is based on content.
- The destinations of Bridges are known and not IP address so they remain anonymous. -But DOS attacks can still be made on that destination.
- It is good in case of showing a working model of multi-bridge.
- It does not involve any inter-bridge communication.

Disadvantages

- The control communication is through I2P network which makes it slow.
- It is likely that tracker based approach is not suitable for highly dynamic nature of bridges (especially in case of small files).
- It is like taking one step back from distributed approach and use trackers which can be a point of failure.
- Involves creating one tracker for each torrent being bridged, so a big overhead. We will also need to track the popularity of that tracker and remove it when it is not used anymore for a long time. Otherwise this approach will make huge amount of useless data. (We should remember that in this approach we query the trackers list before requesting the bridge; more redundant data in tracker's list longer it will take to process the query)
- Limitations of postman trackers can be find out only when tested. Currently the website mentions that we need to have the complete file in order to create tracker for a torrent. It calculates the metainfo, so we need to find if it is at client level or at the level of postman tracker's website.

Option 2:
Use of I2P DHT for content; Inter-Bridge Communication for optimized distributed caching and EEPSite for Bridges

I2P DHT for Content
This approach is similar to the Mainline DHT of P2P network.
The protocol is described at [2].
The modification needed is to have a DHT interface at I2P side of the bridge. Every bridge will be responsible for announcing the torrents it is bridging in I2P network. At I2P client level (and at bridge level), a query in the I2P network will be made about

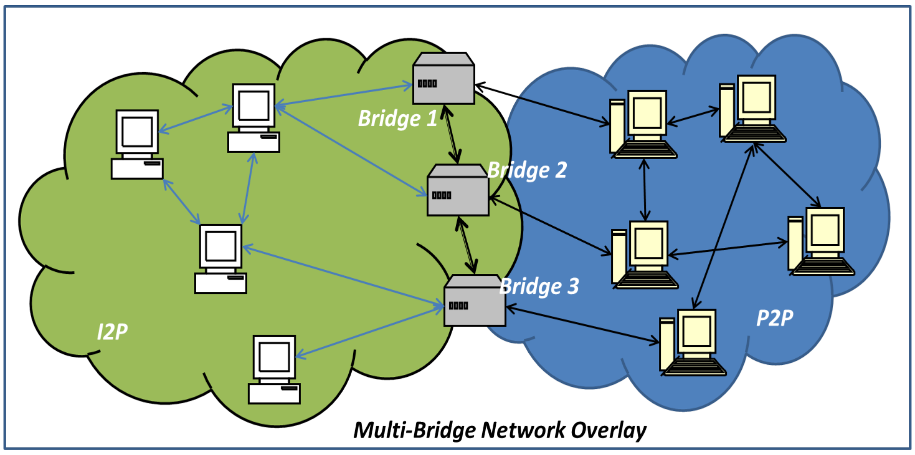

Figure 4.5: Possible MultiBridge Layout

the content, and it will reply with a list of destinations (can be peers or other bridges: we don't care about it; we care only about the content), which can be contacted for the content.

Inter-Bridge communication for optimized caching. (Using Extension Message)

When we query in DHT and obtain a list of destinations, we contact them for content. Our goal is to identify which peers are usual I2P clients and which are bridges. We propose to use the extension message bits in the handshake to identify if the contacted party is another bridge or a usual I2P client.

In this way we can maintain a list of other bridges which are bridging the same content. The purpose is to identify other bridges which are bridging the same content in the best possible way. This approach does not involve querying huge tracker list or flooding all bridges with query of content, as we have got those bridges from DHT which are sharing that content.

Ultimately we need to have an optimized caching i.e. availability of maximum pieces. It requires that bridges have non-overlapping parts of torrent in their cache and know about which other bridge has which piece in its cache.

EEPsite for Bridges

When a new bridge comes on-line, it needs to announce itself in the I2P network so that I2P peers can start requesting it to bridge torrents. The process of announce is simplified by using an EEPSite. When a new Bridge wants to announce itself, it needs to register itself at this EEPSite. From the I2P clients' point of view, this EEPSite can be requested to find a free bridge (or may be a couple of them). There can also be more than one EEPsite, new or replicas (to eliminate a single point of failure). They simplify the process of bridge discovery when the peer has just come online and lacks information about any bridge. Another approach is possible to identify bridges in case EEPSite fails : mentioned in the advantages.

Advantages

- Distributed approach for content discovery; no use of trackers. It avoids huge amount of overhead we could have in querying all the bridges / trackers for content.
- Inter-bridge communication for caching happens only between bridges having the content. We are not required to query separately for bridges sharing specific content as we get that information from DHT. Thus, only for caching Inter-bridge communication just between a small number of bridges will be simpler and is likely to result in a better caching.
- Easy to bootstrap with one bridge from inquiring an EEPSite, with multiple replicas they are also not a Point of Failure. In case we assume that EEPSite is not a good approach and can be attacked or Mal-fed with incorrect bridge destinations, then we can make use of DHT. Queries can be made in DHT for popular contents and it is likely that DHT will return some of the bridges.

Dis-advantages

- I2P DHT is still under progress. Recently it has been modified to be active by default in the new version.

4.7 I2P DHT Network

We have come up with amalgamation of above ideas and a better approach for inter bridge communication. An approach which will not require to make any changes on the I2P client side. The idea is to have an EEPSite where one can submit the requests giving the hash of the torrent one would like to have bridged in I2P network from P2P. This communication is within I2P net and totally anonymous.

The EEPSite then contacts the bridge and requests it to start bridging the torrent. Once the bridge starts the process, it will make use of available DHT network in I2P network and announce itself as a peer for that torrent.

The client who requested the torrent at EEPSite will discover the bridge as a normal I2P peer sharing the file. This approach is still in development and needs regressive testing for the amount of time required for this complete process.

Chapter 5

Conclusions

The working model of the bridge is as shown below in the image.

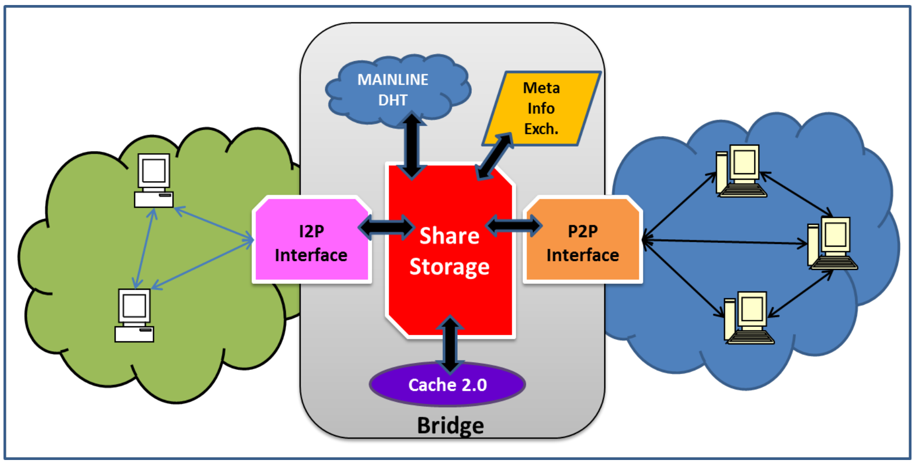

Figure 5.1: Current Working Model with required Functionalities

With this model the bridge has the integrated MainLine DHT, MetaInfo getter, advanced caching algorithm, a better BitTorrent API (tTorrent) than JBITorrent API.

The ongoing work with the bridge is to make it available to all the clients through I2P DHT and without making any changes in the client side.

Bibliography

[1] A bird's eye view on the i2p anonymous file-sharing environment. In the 6th International Conference on Network Security and Systems. NSS 2012. November 2012, Wuyishan, China., Juan Pablo Timpanaro, Isabelle Chrisment, Olivier Festor.

[2] Dht protocol. http://docs.i2p-projekt.de/javadoc/org/klomp/snark/dht/DHT.html.

[3] Jbittorrent api. http://sourceforge.net/projects/bitext/.

[4] Mainline dht. http://en.wikipedia.org/wiki/Mainline_DHT/.

[5] Turn's bittorrent java library. https://github.com/turn/ttorrent.

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803