P2PAP - P2P Anonymizing HTTP Proxy

for Peer-to-Peer and Overlay Networks

Final Report

Group 4:

Bravo, Angel Manuel
Lazetic, Strahinja
Tomsic, Alejandro Zlatko

About this Document

This is final report version in an update of the previous delivered one. At the beginning, the latter is included (just like before). Finally, the differences with the actual running version are described point by point, indicating the reason of the changes implemented.

Objective

The purpose of this protocol is to provide the definition of the communication primitives between peers within a Peer to Peer Anonymizing Proxy (from now on P2PAP) network. As it is the responsibility of this network to provide not only anonymity to clients, but also content caching, the definition of the way cached objects are going to be stored among peers and how lookups are going to be performed between them, is also provided.

First Version (Our Initial Intention)

System's Description

Overview

The system is comprised of several peer servers running P2PAP. They all conform a Chord Ring where the objects are the cached web pages stored by generating a hash key of their URL. Each of them knows about the existence of some others and their geographical location. Peers are able to generate a web page with this information when a client wants to retrieve a list of servers.

Client's View of the System

When a client wants to access a web page using P2PAP, it first requests to one of the servers (it already knows) for a list of the servers it knows and their location. Based on this, the client chooses the most convenient in terms of network metrics and configures it as it's proxy server. From this moment on, it will get all the pages he requests from this server. All further steps are transparent from the user's perspective.

How the System Runs

Once a peer server has received a client's request, it first checks whether the requested content is in its own cache and tries to directly reply. If it can't, it will check if the content has been cached in the network. At this step, the client's proxy will generate a hash key of the object (URL) and using Chord he will be able to find the responsible of the content. Then, it will check whether the responsible is in the same country of it or of the destination web server. if it is not, it will send a message to the responsible asking for the content (content_request message). If it is, it will send a type of message (cache_request) indicating the second server to only search in it’s cache and not to contact the destination server in case of a miss. The first server also contacts a peer which is not in the country of conflict. The first reply received is used. Afterwards, if the second message has been the fastest one, the first peer sends a store in cache message to the responsible. All involved peers cache the content.

When a node responsible for a content is contacted, it will check its cache and it will reply if there is a hit. Otherwise if it received a content_request message it will request straight to the destination web server and reply to client's proxy. The latter will reply to the client.

Maintenance

List of peers

When a peer joins the network, it will request the peer list to its successor. The latter will update its list with the new peer. Each time peers contact each other both will try to save the other one if it is possible. An algorithm is periodically run checking if known peers are alive removing the idle ones.

Cache

Nodes responsible for a given object are going to store it in their cache. Furthermore, in any case a peer gets a content, it will cache it. When the cache is full the oldest accessed entry will be removed. Every time a hit occurs the peer will update the referenced entry.

Messages between peers

CONTENT_REQUEST (url)

It requests a web page indicated by its URL.

CONTENT_REPLY (url, content)

It replies a content_request supplying a web page content.

CACHE_REQUEST(url)

The same as content_request but stating not to forward the message in case of a cache miss.

STORE_REQUEST(url, content)

It indicates to a given peer to store the content in its cache.

PEERS_REQUEST()

It requests the peer list.

PEERS_REPLY(peers)

It replies a peers_request message supplying its peer list.

Management Service

As the system is thought to provide a high level of service, a management service is provided. This feature does not intervene with the system's operation and peers need not contact the management services for any purpose regarding the full featured working of the system.

A management server will be in contact with every peer working in the network and maintain information about their cache (number of objects stored and for each object URL, size, age, number of accesses and other nodes where the object is stored). Each peer holds in it's configuration the IP address of this server. If this IP Address needs to be changed for any reason, the configuration of the nodes must be updated.

When running, each node sends to the server alive messages every thirty seconds to indicate of its existence. If a node has not sent an alive message for two minutes, the management server will remove it from its active peer list. A status message is included in the protocol to provide the information needed by the server. Each node sends a status message when expressly asked by the management server. Is responsibility of the server to ensure the network is not going to be flooded with status messages.

Messages between peers and management server

GOODBYE()

A peer will use this message to inform the management server when it is leaving the network.

STATUS_REQUEST()

The management server will use this message to request information about all the objects which a peer is storing.

STATUS_REPLY(CacheEntriesList)

It replies a status_request message supplying the information of the stored objects.

ALIVE()

A peer will use this message to inform the management server that it is still alive.

FLUSH_CACHE()

The management server will use this message to ask a peer for flushing its cache.

DELETE_ENTRY(url)

The management server will use this message to ask a peer for deleting an entry of its cache.

Final Version

Differences with the proposed version

When implementing the solution, we made some modifications. Most of them were intended to make the program simpler. The program remains fundamentally the same in functionality. The management service will remain unmodified, and is explained at the end of the document. The differences are the following stated:

Chord Implementation

The actual implementation does not store the cached objects in the chord ring. In fact, nothing is stored there. The chord network is only used to retrieve, for a given URL (key in string format), its responsible node. A node searching for an object will get its responsible from chord and contact it to retrieve it. The cached objects are stored in each peer's main memory.

List of Peers

The list of peers a node maintains is its finger table. We arose to this approach aiming for simplicity. If a client (user) wanting to connect to the network asks a peer for a list of available Proxy Servers, the peer will first try to contact the management server and retrieve a full list of working nodes from it. If the management server is unavailable, it will only show the client a reduced list of nodes from it's finger table in format <IP, Country> like the following:

The available proxy servers are listed below:

IP Address	Country
195.251.248.180	Greece
155.207.48.52	Greece
192.114.4.2	Israel
194.167.254.19	France
143.205.172.11	Austria
193.174.155.27	Germany

Cache

In the initial version we had not specified how the elements in cache were going to be deleted. A deleting routine was implemented which is ran every hour deleting the expired objects and also the ones which have not been accessed (hit). When a node tries to insert an object and cache is full the same routine is run.

Messages Between Peers

None of the previously declared messages were implemented. For communication between peers, we simply used HTTP. When a proxy asks for a certain content to another one, it issues a HTTP request including a special header we called “emdc”. A peer receiving a request responds with a regular HTTP Reply.

Messages Between Management Server and Peers

In this case, the messages were also replaced by HTTP communications using special headers. They are the following:

- Alive: Sent from a peer to the management server indicating it is alive.
- Peers List: Sent from the peer to a management server when it requests for the full peer list.
- Status Request: Sent from the management server when asking a node for it’s list of cached objects.
- Flush Cache: Sent from the management server to a node when indicating a cache flush.
- Delete Entry: Sent from the management server to all nodes which contain a given cache object indicating to delete it.

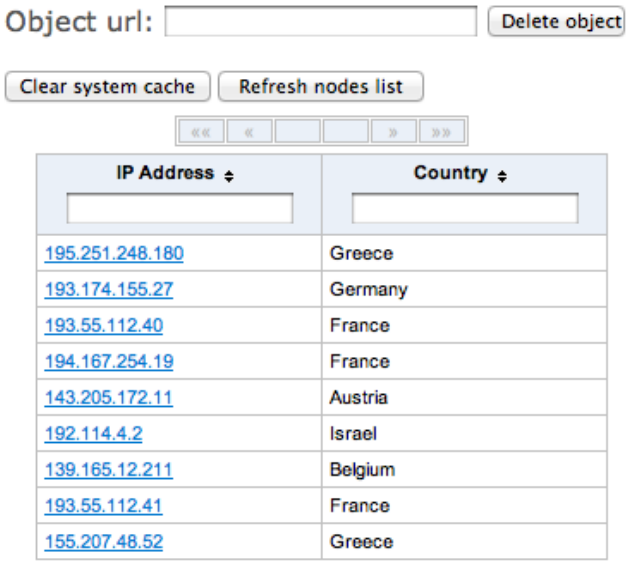
How the Actual System Runs (after modifications)

Once a peer server has received a client's request, it first checks whether the requested content is in its own cache and tries to directly reply. If it can't, it will get the responsible of the URL requested from the chord network. Then, it will check whether it is the responsible or the responsible is in the same country of it or of the destination web server.If none of this occurs, it will perform an HTTP request using the *emdc* header to the responsible peer. If it is, the first server performs such request to a peer which is not from the country of conflict. All involved peers cache the content.

When a node is contacted from another peer requesting some content, it will check its cache and it will reply if there is a hit. Otherwise it will perform a request straight to the destination web server, reply to the first proxy and cache the content. The latter will reply to the client.

Management

The management server is presented to the user as a web interface, where he can check the full list of working peers, look at the objects information each peer stores, flush a proxy's cache, flush full system cache, and remove a given element from all the proxies which have it. The above images show the two different views of the management site. The following image shows the initial page, where the list of working peers and its country is shown:



Object url: Delete object

Clear system cache | Refresh nodes list

« 1 2 3 4 5 6 7 8 9 10 »

IP Address	Country
195.251.248.180	Greece
193.174.155.27	Germany
193.55.112.40	France
194.167.254.19	France
143.205.172.11	Austria
192.114.4.2	Israel
139.165.12.211	Belgium
193.55.112.41	France
155.207.48.52	Greece

When clicking over the IP address of a peer, a page represented by the next image is shown, where the user can see all the objects cached by a server and their age, size and number of times they were accessed from cache:

Node address: 193.174.155.27

Node country: Germany

Clear node cache | Refresh node status | Back to node lists

« 1 2 3 4 5 6 7 8 9 10 »

Url	Age(s)	Size(bytes)	Access number
http://www.blic.rs/resources/images/icons/carosel_button.png	64193	342	0
http://i2.cdn.turner.com/cnn/dam/assets/111129113430-turkana-kenya-video-tease.jpg	65289	4083	1
http://www.blic.rs/resources/images/header/header_back.png	65208	9343	2
http://i2.ytimg.com/vi/MDQT8RUvejc/hqdefault.jpg	65893	11865	1
http://i2.cdn.turner.com/cnn/2011/images/11/03/ge_120x68.jpg	65262	23493	1
http://www.polovniautomobili.com/images/infostud_logo.png	65282	4219	0
http://www.youtube.com/opensearch?locale=fr_FR	65038	562	0
http://i.cdn.turner.com/cnn/.element/img/3.0/global/shade/BR_shadow_1000x6.gif	65424	394	1
http://i.cdn.turner.com/cnn/.element/img/3.0/search/btn_search_hp_text.gif	65486	858	2
http://www.blic.rs/resources/images/superodmor/search_button.png	65129	3486	2
http://www.polovniautomobili.com/_googleadmanager_banners.php?uri=/&position=vertikalni&isa=false&rnd=2&	65041	3809	0
http://i2.ytimg.com/vi/Ud6t5bIRcVg/market_thumb.jpg?v=4ec22986	85607	23035	1
http://i.cdn.turner.com/cnn/.element/img/3.0/global/buttons/Sprite_BT_master.gif	65469	62441	2
http://static.ak.fbcdn.net/rsrc.php/v1/yI/r/TQWcLo9tKYa.js	65058	59427	2
http://cdn.api.twitter.com/1/urls/count.json?url=http%3A%2F%2Fwww.minimal-mass.net%2F&callback=twttr.receiveCount	62811	68	0

Implementation on Planet-Lab

We implemented a system of nine running proxy servers running on european machines. We also run a management server on another machine. Fortunately, the system worked as we expected and we did not need to make major changes nor re implement any feature. We had some problems of speed because every different machine has different capabilities, and sometimes we realized some of them became a bottleneck. When using this platform, it is important to carefully pick the nodes used.

Conclusions

We found working on this project both interesting and challenging. We got to learn and implement different technologies, like HTTP and it's caching system, OpenChord and a geo localization database. We also had to implement our solution on Planet-Lab and got the possibility to see how a real distributed P2P system works.