Project in

Networked Software Systems

(044169)

DHT Firefox Extension

Midterm Report

**Supervisors:**

Mr. Ittay Eyal

Dr. Ilana David

**Team Members:**

Daniel Aranki

Hani Ayoub

May 2009

# Contents

[Contents 2](#_Toc230727833)

[Project abstract 3](#_Toc230727834)

[General description 3](#_Toc230727835)

[What is a DHT? 3](#_Toc230727836)

[Acronyms & Terminology 4](#_Toc230727837)

[Scenario description 5](#_Toc230727838)

[Project Progress 5](#_Toc230727839)

[Why Firefox? 5](#_Toc230727840)

[What statistics do we need to gather? 5](#_Toc230727841)

[High-Level Design 6](#_Toc230727842)

[High-Level components communication 6](#_Toc230727843)

[Database content 7](#_Toc230727844)

[Desired Results 7](#_Toc230727845)

[Indices 7](#_Toc230727846)

[Predictions 9](#_Toc230727847)

[References 10](#_Toc230727848)

# Project abstract

## General description

The project deals with the question of whether a DHT can be implemented on Mozilla Firefox or not. Mozilla Firefox is a widely used free web browser that features Linux, Mac OS X, Microsoft Windows and many Unix-like operating systems.

For us to give the answer for this question, we will present a road-map for collecting relevant information from Firefox users in order to determine if Firefox is capable of running DHT in sense of duty time (which is how many times, and for how long each time, a Firefox session is opened), bandwidth, load and utilization.

The project is going to be coded in JavaScript, PHP, MySQL and Firefox extension platform (XUL and XBL) and is going to be an open source project.

## What is a DHT?

DHT [1] stands for **D**istributed **H**ash **T**able and consists of a (generally) decentralized distributed system in which looking up (*key*, *value*) pairs is possible. The (*key*, *value*) pairs are stored in the DHT similar to normal Hash tables. DHT can form a basic infrastructure for more complex services such as distributed file systems, peer-to-peer (P2P) file sharing and distributed data processing.

The structure of the DHT consists of a number of parts:

* The keyspace which is the space from which the keys are taken
* The keyspace portioning scheme which is responsible for partitioning the (*key*, *value*) pairs between the nodes.
* The overlay network which is the structure that connects the nodes. There are several overlay network structures that lead to different performance tradeoffs between:
  + *Degree*: the maximum node degree in the overlay network, i.e: maximum number of neighbors for any node
  + *Search route length*: the maximum number of nodes in the way of looking for a (*key*, *value*) pair.

Several DHT implementations define a delta function as follows: ***δ:Keyspace\*Keyspace -> Distances space***

Delta has the meaning of the distance between two keys in the keyspace. In general, every node has an ID which is called the node id. Node IDs are taken from the keyspace. So several DHT implementations use the following keyspace partitioning scheme: Every node whose id is *id1* will contain every key *k* for which ***δ(id1,k)*** is minimal in sense of all other node IDs currently present in the DHT.

This partitioning scheme allows us to search in the DHT in a greedy way such that every node (*id1*) who gets a search query for some key will check if he has a neighbor whose id is (*id2*) where ***δ(id21,k)< δ(id1,k)*** if so, then the search query is sent to the minimal *id2* that satisfies the condition above (in sense of the delta function). Otherwise, then the key is present at the current node *id1* and then transferred to the node who asked for it. Please note that this way does not always assure that we find the key we’re asking for (for example in case of local minima of the delta function).

One DHT implementation that assures finding the requested key is Chord [2]. Chord is a specific implementation of DHT in which the keyspace partitioning scheme is to treat each key as a point in a circle [3], and the distance between two keys is the travelling distance between them clockwise. In this case, for each two nodes *i1* and *i2*, *i2* will contain all keys that fall between *i1* and *i2*. In this implementation, if a search query has reached node *i*, (and it knows its predecessor, which is the node which node *i* it comes after) it will decide if it has the requested key using the condition above, otherwise, it’ll pass the search requested to the next (successor) node.

## Acronyms & Terminology

Acronyms:

* ***DHT:*** **D**istributed **H**ash **T**able
* ***FF:*** Mozilla **F**ire**f**ox browser
* ***BW:* B**and**w**idth

Terminology:

* ***Station:*** a machine runs Mozilla Firefox browser with the extension installed.
* ***Server:*** the machine collects the data from *Nodes*.
* ***Station ID:*** a unique key for a each station.
* ***User ID:*** a unique identifier for each user.

# Scenario description

## Project Progress

* First, we will publish the extension for Firefox that will include all the features stated in this document.
* Afterwards, we will spread the extension to Firefox users over the internet.
* After we get enough statistics, we will evaluate them to decide whether a DHT can be run on Firefox, in sense of duty time, utilization and load, or not.
* Evaluate indices for bandwidth usage for each type of DHT use.

## Why Firefox?

Mozilla Firefox is the second most popular web browser among the internet users; Mozilla Firefox has recorded to be responsible for 22.48% of the total usage share of web browsers as of April 2009. It supports easy-to-build and easy-to-use modular extensions and modules that can be downloaded, managed, updated and used easily on every Firefox.

From these facts, it seems that Firefox is a great potential for implementing a DHT on. In addition, Firefox is part of the open source community in which people from all over the world share the interest of freedom of code; thus, implementing a DHT on Firefox will open a wide set of new projects and research.

## What statistics do we need to gather?

Following is a list of information we need to gather from the users:

* ***Duty time***: Duty time is defined to be the Times/Dates in which a user opens a Firefox session and closes it.
* ***PC Specifications***: CPU information, amount of memory installed on the computer, graphical processing unit and operating system. This information enables us to investigate more on what uses we are entitled to implement a DHT on Firefox for. Furthermore, this information enables us to classify the requirements for each use of DHT and gives us a good estimation for the percentage of Firefox users that can run such a DHT.
* ***Location***: Anonymous geographical location might also be needed to optimize the DHT performance by knowing the concentrations of Firefox users around the world.

# High-Level Design

This section comes to describe the high-level design of the system, implementation, components communication and data-structures used.

## High-Level components communication

Collecting the relevant data and information from the user is done using a server machine residing in the Technion Softlab. Its responsibility is to manage the data gathered and communicate with user machines; each machine will send the data that it holds regarding its duty time, PC specifications, machine and user identifiers and all other pieces of information to the server through the interface.

The machine’s FF browser has the logic for taking out the info using Java-Script code. It will send the data it gathered/taken out to the server described above through the server interface; the server then will collect and save the data in its MySQL server (and may add some interpretations).

The Following diagram describes the High-Level communication between main components:

## Database content

The collected data will be saved on the server machine using MySQL server. Besides the collected information described before, the system keeps a unique ID for each node joins the DHT which will be determined by the server the first time the node connects the system (upon installation), this ID will identify the machine every time it sends information to the server and will be used for mapping purposes. The server will hold the following tables:

* ***Duty-Time table***: holds the times every machine opens/closes a FF session.
* ***Node’s machine specifications table:*** for each node, machine’s specification info will be collected (info included described in previous section).
* ***Location:*** City, Country and IP address is stored for each node.

## Desired Results

### Indices

After implementing the extension, publishing it over a considerable amount of FF users and collecting the results, the data will be investigated considering indices\* in order to be able to decide wither it’s possible to implement DHT over FF or not. Using duty-time we will be able to determine how much FF sessions were opened simultaneously on the same node’s machine what allows us to set the load on a machine (relevant for some DHT uses, where every session is going to be a different node); using the machine specification such as CPU info and memory used along with its location may enable us performing DHT tweaks and improving its performance.

We will also output other indices regarding each type of DHT to be implemented, for example, we will give an indication on how much bandwidth will be needed for each DHT implementation as a function of the statistics results. For a data-store DHT, we will need more bandwidth than for machines mapping for example. These calculations will be based (generally) on the Chord [3] scalability of the DHT.

\* These indices are functions of the DHT type to be implemented later on FF; for example, for a data-store DHT we will need long duty times, but for machines mapping, we don’t need long duty times

### Predictions

As stated above, we will output indices for some DHT types. These indices will include (but not limited to):

* Affordable FF Bandwidth: This index will be evaluated out of the statistics we will gather to check how much bandwidth each type of DHT will use. For example, if we discover a short average duty time, then a data-store will need higher average bandwidth because a high number of nodes will check in and out per time unit; so the nodes will need to transfer more data per time unit.

We will also provide predictions of DHT types that can be implemented based on the statistics we will gather, for example:

* Based on the PC specifications of FF users, we will decide if a distributed processing system can be implemented using DHT on FF.
* Location information may be a good indicator for performance tweaks of the DHT to be implemented (in sense of the delta function declaration and other parameters such as the key space partitioning)

# References

[1] Wikipedia. Distributed Hash Table.

<http://en.wikipedia.org/wiki/Distributed_hash_table>

[2] Wikipedia. Chord.

<http://en.wikipedia.org/wiki/Chord_(DHT)>

[3] Stocia, I., Morris, R., Et al. Chord: A Scalable Peer-to-peer Lookup Service for Internet applications. MIT Laboratory for Computer Science. Cambridge, Massachusetts, 2001.